Modelization and identification of multi-step cyberattacks in sets of events

Julio Navarro Lara

UNIVERSITÉ DE STRASBOURG

ÉCOLE DOCTORALE MATHEMATIQUES, SCIENCES DE L'INFORMATION ET DE L'INGENIEUR

Equipe "Complex System & Translational Bioinformatics"

– Laboratoire ICube – UMR 7357

THÈSE présentée par :

Julio NAVARRO

soutenue le : **14 mars 2019**

pour obtenir le grade de : **Docteur de l'Université de Strasbourg**

Discipline/ Spécialité : Informatique

Modelization and Identification of Multi-step Cyberattacks in Sets of Events

THÈSE dirigée par :

Mme DERUYVER Aline	McF HDR, Université de Strasbourg, ICube, Strasbourg
et co-encadrée par :	
M PARREND Pierre	Enseignant-Chercheur HDR, ECAM-Strasbourg-Europe, ICube, Strasbourg

RAPPORTEURS :

M LATAPY Matthieu	Directeur de recherche, CNRS et Sorbonne Université, LIP6, Paris
M GARCIA-ALFARO Joaquin	Professeur, Université Paris-Saclay et Télécom SudParis, R3S, Paris

EXAMINATEURS :

Mme LEGRAND Véronique	Professeur, Conservatoire national des arts et métiers (CNAM), CEDRIC, Paris

Acknowledgements

One of the most important inflection points in my life happened when I quitted Telefónica and moved from Madrid to Strasbourg to begin the Ph.D. that culminates in this thesis. More than three years later, I can say that this decision was one of the best ones I have ever made. As Faulkner said, "you cannot swim for new horizons until you have courage to lose sight of the shore". And there are many people I would like to thank for the arrival to this new horizon.

First and foremost, I would like to thank my supervisor, Pierre Parrend, and my thesis director, Aline Deruyver. They made my arrival to France possible when accepting me to do this Ph.D. Thanks to the close supervision of Pierre I have been able to correctly build this manuscript and the related publications.

I would also like to express my gratitude to Joaquín García Alfaro, Matthieu Latapy and Véronique Legrand, for accepting to be part of the jury and for their valuable comments. I would like to thank Véronique also for leading the HuMa project and for putting me in contact with Pierre for the first time, when I contacted her after reading her research work. The conversations with them were the starting point of this adventure.

I am very grateful to the two directors of the CSTB team, Olivier Poch and Pierre Collet. I appreciate being so well received in their team and their recommendations about my work, but also some other non-academic contributions. Olivier has taught me to play chess (even if I am still a noob) and I got from him the best advices about life and the most ironic expressions of the *belle langue française*. Thanks to Pierre I had my first flight in a light aircraft, an amazing experience, and I have learnt many interesting things such as how to use vacuum to brew coffee.

Olivier and Pierre could be called the "parents" of the big CSTB family. I am very thankful to all its members for making me feel like at home. I did not only find excellent colleagues but also really good friends. I thank the permanent researchers, who has always been willing to help or just to talk about any topic: Laetitia, Luc, Odile, Julie, Raymond, Claudine, Anne J., Jean-Sébastien, Catherine, Christian. I am particularly grateful to Julie, who kindly reviewed many of my manuscripts in English, and to Raymond, who solved my numerous problems with bikes, my main means of transport to the lab. I highly appreciate the invaluable help of Anne N. Without her I would be still fighting against French administrative paperwork.

Now it is the turn of the youngest. I would like to thank my comrade Arnaud for his help with the computer infrastructure of the lab and for many great discussions about philosophy. I am really grateful to my brothers in arms, the fellows who were

Ph.D. students at the same time than me or who still are: Yannis, Pierre W., Carlos, Kirsley, Audrey, Gopal, Fabio, Renaud, François, Anna, Thomas, Nicolas S., Romain. We helped each other, fixed the world at every lunch time, had beers and mutually complained about the tasks to fulfill. Finally, I thank all the *stagiaires* who came to CSTB for an internship, specially Paul, Timothée, Raaj, Isaac, Hélène and Camille.

Regarding people outside CSTB, I apologize for not including you all, due to the restraint space. First, I am truly thankful to Adrián, who planted the seed of doing a Ph.D. in France when I was visiting him in Lyon in Spring 2015, after listening my professional and personal concerns. I am also very grateful to Nicolas T., the first person I met in Strasbourg. He was also student at CSTB but he deserves a special place, as he integrated me to his group of friends during the first year, when I could only babble some high-school-level French. Apart from the cited names, I would like to mention some of the friends whose support has been very relevant for the good accomplishment of this thesis: Ayrton, Javi and Antonio, who are always close even if they are far; Giosuè, for our journeys to the end of the night; Martín and Mikel, for how we had fun discovering the city and its surroundings; Juanjo, who became a brother to me before he left Strasbourg, and Pablo, Rosa and Isaura, who have been a great backing while writing and with whom I share many passions. Thanks also to all the members of ESN Strasbourg, to their engagement and their open mind. Through this association I could meet most of the people I know here. Additionally, I would like to thank the city of Strasbourg, its people and its cultural life. Living in this city has allowed me to learn French, to write opera critics or to play trumpet, among many other things. Last but not least, I am very thankful to Andrea, the person who has suffered my thesis the most and who has given me the best coaching ever.

Finally, I would like to express my warmest thanks to my family, to my parents Luisa and Gabi, and to Eva, the best sister in the world. They are always there when I need them and I owe them everything.

Gracias a todos.

Julio

Contents

1 Introduction **1**

1.1 Motivation . . . 2
1.2 Goals . . . 3
1.3 Contributions . . . 5
1.3.1 A survey on multi-step attack detection . . . 5
1.3.2 Abstract Attack Scenario Graphs (AASG) . . . 6
1.3.3 Detection and identification of scenarios . . . 6
1.3.4 Visual investigation of attack scenarios . . . 7
1.4 Publications . . . 7
1.5 Thesis outline . . . 8

2 Preliminary definitions **11**

2.1 Computer networks under threat . . . 11
2.1.1 The battleground: a connected world . . . 12
2.1.2 Threatened by cyberattacks . . . 13
2.1.3 Constrained by vulnerabilities . . . 14
2.2 Cybersecurity: objectives and mechanisms . . . 15
2.3 Traces in a network . . . 17
2.4 Definition of multi-step attack . . . 20
2.4.1 Sequences of actions . . . 21
2.4.2 The incidence of multi-step attacks . . . 22
2.5 Summary . . . 23

3 Multi-step attacks **25**

3.1 Multi-step attacks in context . . . 25
3.1.1 Comparing single and multiple steps . . . 26
3.1.2 A classification of multi-step attacks . . . 26
3.1.3 Advanced Persistent Threats . . . 28
3.2 Examples of multi-step attacks . . . 29
3.2.1 WannaCry . . . 29
3.2.2 LLDoS 1.0 and LLDoS 2.0.2 . . . 31
3.2.3 UNB ISCX island-hopping . . . 33
3.2.4 HuMa . . . 34

3.3 Modeling multi-step attacks . . . 35
3.3.1 The attack as a sequence of traces . . . 36
3.3.2 Languages to model multi-step attacks . . . 36
3.3.3 Concrete Attack Scenario Graphs (CASG) . . . 42
3.4 Connection of nodes in the CAGS . . . 44
3.4.1 Similarity features . . . 45
3.4.2 Time-based features . . . 51
3.4.3 Context-based features . . . 54
3.4.4 Combination of features . . . 57
3.5 Summary . . . 61

4 Multi-step attack detection 63
4.1 Presenting multi-step attack detection . . . 64
4.2 A methodology for bibliographic research . . . 65
4.2.1 Inclusion and exclusion criteria . . . 65
4.2.2 Research process . . . 67
4.3 Description of detection methods . . . 70
4.3.1 Similarity-based methods . . . 70
4.3.2 Causal correlation methods . . . 78
4.3.3 Structural-based methods . . . 84
4.3.4 Case-based methods . . . 85
4.3.5 Mixed methods . . . 88
4.4 The field in perspective . . . 90
4.4.1 Number of publications and duration of research . . . 90
4.4.2 Type of data and origin of knowledge . . . 93
4.4.3 Reproducibility of experiments . . . 96
4.5 The involvement of the security analyst . . . 98
4.6 Summary . . . 101

5 Abstract Attack Scenario Graphs (AASG) 103
5.1 From Concrete to Abstract ASG . . . 104
5.1.1 Representation of alternative hypotheses . . . 104
5.1.2 Requirements to define the model . . . 108
5.1.3 Rule-based event correlation . . . 110
5.1.4 Defining alternative attack scenarios . . . 112
5.1.5 Comparing CASG and AASG . . . 114
5.2 Definition of AASG . . . 114

5.2.1 Abstract events . . . 115
5.2.2 Nodes and arcs . . . 115
5.2.3 Matching events to nodes . . . 118
5.2.4 Absolute conditions . . . 120
5.2.5 Relative conditions . . . 122
5.2.6 Additional elements . . . 122
5.3 Creating Abstract Attack Scenario Graphs . . . 124
5.4 Implementation . . . 127
5.4.1 Structure and notation . . . 128
5.4.2 Example of implementation . . . 130
5.4.3 Graph editor . . . 131
5.5 Summary . . . 132

6 Algorithms to exploit AASG 135
6.1 Morwilog . . . 136
6.1.1 Ant Colony Optimization . . . 136
6.1.2 General overview of Morwilog . . . 139
6.1.3 AASG as the stigmergic scenario . . . 142
6.1.4 The Morwilog algorithm . . . 146
6.2 Bidimac . . . 152
6.2.1 Adapting Bayesian inference to AASG . . . 152
6.2.2 Morwilog and Bayesian inference . . . 155
6.2.3 The Bidimac algorithm . . . 156
6.3 Differences between Morwilog and Bidimac . . . 159
6.3.1 Choice among options . . . 160
6.3.2 Statistical meaning of weights . . . 162
6.3.3 Mechanism to penalize or reinforce paths . . . 163
6.3.4 Decrement of choosing probability . . . 166
6.4 Summary . . . 166

7 Evaluation 169
7.1 Introducing the evaluation datasets . . . 169
7.1.1 DARPA 2000 datasets . . . 170
7.1.2 ISCX dataset . . . 172
7.1.3 HuMa dataset . . . 172
7.1.4 Normalization of events . . . 174
7.1.5 The *eventgen* datasets . . . 175

7.2 Evaluation of different AASGs . . . 177
7.2.1 DARPA 2000 LLDoS 1.0 . . . 177
7.2.2 From LLDoS 1.0 to LLDoS 2.0.2 . . . 183
7.2.3 Introducing an aggregation phase . . . 187
7.2.4 AASG with scan and double attack . . . 188
7.3 Evaluation of the algorithms . . . 191
7.3.1 Time performance . . . 192
7.3.2 Evolution of the choosing probability . . . 194
7.3.3 Values of parameters . . . 196
7.3.4 Overlapped attacks . . . 197
7.4 Summary . . . 200

8 Visual investigation of attack scenarios 203
8.1 SimSC . . . 204
8.2 Implementations . . . 206
8.2.1 HuMa-SimSC . . . 206
8.2.2 Logan . . . 210
8.3 Raw-SimSC . . . 214
8.4 Summary . . . 216

9 Perspectives 217
9.1 On research about multi-step attack detection . . . 218
9.1.1 The OMMA framework . . . 218
9.1.2 Modularity in OMMA . . . 223
9.2 On the evolution of AASGs . . . 224
9.2.1 The structure of the AASG . . . 224
9.2.2 The creation of the AASG . . . 225
9.3 On attack case detection and identification . . . 226
9.3.1 Further evaluation . . . 227
9.3.2 Case confirmation and step prediction . . . 228
9.3.3 Real-time functioning . . . 228
9.3.4 Automatic functioning . . . 229
9.3.5 New algorithms . . . 231
9.4 On the visual investigation of multi-step attacks . . . 231
9.5 Summary . . . 234

10 Conclusion 235

A Summary of definitions 239

B List of acronyms 243

C List of inherent attributes 245

D Multi-step attack detection methods 247

E Summary of AASG conditions 257

F Examples of AASGs 259

Bibliography 265

List of Figures

1.1 Processes when finding cyberattacks 4

2.1 Number of reported CVE vulnerabilities per year 15
2.2 Definition of 'Cybersecurity' according to ENISA 16
2.3 Actions in computer networks 18
2.4 Scope of traces, packets, events and alerts. 20

3.1 Web log with SQL Injection attack 26
3.2 A subscenario of the WannaCry attack 27
3.3 Classification of multi-step attacks 28
3.4 LLDoS 1.0 attack 31
3.5 LLDoS 2.0.2 attack 32
3.6 UNB ISCX island-hopping attack 34
3.7 HuMa attack 35
3.8 Example of LAMBDA signature [Cuppens 2000] 38
3.9 A multi-step attack represented in a Petri net [Zhao 2014] 40
3.10 Example of hierarchy trees [Julisch 2003a] 51

4.1 Keywords used in the survey on multi-step attack detection 69
4.2 Taxonomy of multi-step attack detection classification. 71
4.3 Number of publications per year 90
4.4 Number of publications in each approach per year 91
4.5 Distribution of publications by the type of experiment 94
4.6 Distribution of publications by the origin of knowledge 95
4.7 Distribution of publications by reproducibility factors 97
4.8 Reproducibility of experiments per year 98
4.9 Diagram of the framework by Shaneck et al. [Shaneck 2006] 99
4.10 Diagram of the AI^2 system [Veeramachaneni 2016] 100

5.1 Trees with hypotheses in the AOH model [Zhong 2013] 106
5.2 Example of the AOH model [Zhong 2015] 107
5.3 Example of an issue tree [Wojick 1975] 108
5.4 Example of two attack detection rules for DARPA 2000 111
5.5 Example of alternative case derivation from WannaCry 113
5.6 Example of an AASG with a counter 124

5.7 Example of an AASG with optional nodes . . . 125
5.8 Representation of AASG Mill-1 . . . 128
5.9 Example of an AASG represented in Cytoscape . . . 132

6.1 Example real foraging ants . . . 137
6.2 Example about the progress of a *morwi* through an AASG. . . . 142
6.3 Evolution of pheromones for continuous reinforcement . . . 145
6.4 Evolution of the AASG when the attack is confirmed . . . 145
6.5 Evolution of the AASG when there is no attack . . . 146
6.6 Transformation of AASG with optional nodes . . . 151
6.7 Example of a Bayesian network. . . . 153
6.8 Equivalence between stigmergic and Bayesian AASG . . . 157
6.9 Morwilog vs. Bidimac with a simple AASG . . . 162
6.10 Continuous reinforcement . . . 164
6.11 Evolution of $P_{0,1}$ in continuous reinforcement . . . 165

7.1 Functional representation of the AASG Mill-1 . . . 178
7.2 Number of alarms in inside1-NoMill with AASG Mill-1 . . . 179
7.3 Results of Bidimac with AASG Mill-1 in inside1-NoMill . . . 181
7.4 Results of Morwilog with AASG Mill-1 in inside1-NoMill . . . 182
7.5 Number of branch matches in Morwilog . . . 182
7.6 Formal representation of AASG Mill-2. . . . 183
7.7 Functional representation of AASG Mill-2. . . . 184
7.8 Number of alarms with AASG Mill-2 in inside1-NoMill. . . . 185
7.9 Alarms with AASG Mill-2 in aggregated inside1-NoMill . . . 185
7.10 Results of Bidimac with AASG Mill-2 in inside1-NoMill . . . 186
7.11 Results of Morwilog with AASG Mill-2 in inside1-NoMill. . . . 186
7.12 Number of alarms with AASG Mill-2 in inside2. . . . 187
7.13 Formal representation of the AAGS S2A. . . . 189
7.14 Functional representation of the AASG S2A. . . . 191
7.15 Formal representation of the AASG Eventgen . . . 193
7.16 Execution time vs. the size of the dataset . . . 194
7.17 Execution time vs. T_{max} . . . 194
7.18 Choosing probability in continuous reinforcement . . . 195
7.19 Choosing probability in continuous penalization . . . 196
7.20 Choosing probability vs. η (Bidimac) . . . 197
7.21 Choosing probability vs. ρ and w after 3 attacks (Morwilog) . . . 198

7.22 Choosing probability vs. ρ and w after 16 attacks (Morwilog) 198
7.23 Choosing probability vs. ρ and w after 66 attacks (Morwilog) 198
7.24 TP and FP vs. step separation with AASG Eventgen 199
7.25 TP and FP vs. step separation with AASG Eventgen-Noipsrc 199

8.1 IDS log indicating SQL Injection attack 204
8.2 Diagram of the layers in HuMa architecture [Navarro 2017] 207
8.3 The integration of HuMa-SimSC in the HuMa architecture. 207
8.4 Three potential DoS attempts in HuMa-SimSC 208
8.5 Two port scans in HuMa-SimSC 208
8.6 Identification of periodic events in HuMa-SimSC 209
8.7 Logan interface 210
8.8 Logan global graph 211
8.9 Logan scenario 212
8.10 Script testing in Logan 212
8.11 ZmEu scanner in Logan 212
8.12 Results of Raw-SimSC in DARPA 2000 inside1 213
8.13 A 'Sadmind_AO' scenario extracted by Raw-SimSC 214
8.14 A 'Sadmind_AO' and a 'Rsh' scenarios extracted by Raw-SimSC . . . 215
8.15 An isolated 'Stream_DoS' alert in DARPA 2000 inside1 215

9.1 OMMA architecture 220
9.2 Example of a modular implementation of OMMA 224
9.3 ROC curves from automatic Morwilog 230
9.4 Example of transitive reduction 232
9.5 Similarity-based graph of alerts related to LLDoS 1.0 233

F.1 Formal representation of AASG Mill-1 259
F.2 Functional representation of AASG Mill-1 260
F.3 Formal representation of AASG Mill-2 261
F.4 Functional representation of AASG Mill-2 261
F.5 Formal representation of AAGS S2A 262
F.6 Functional representation of AASG S2A 263

List of Tables

1.1 List of events containing an instance of WannaCry . . . 3

2.1 2017 dwell time values of attacks . . . 23

3.1 Inherent attributes considered by Pei et al. [Pei 2016] . . . 47
3.2 Similarity features defined by Pei et al. [Pei 2016] . . . 49
3.3 Atomic similarity features in the literature . . . 52

4.1 Ranking of top 20 entries by number of citations . . . 92
4.2 Ranking of top authors by number of publications . . . 93
4.3 Types of trace and number of publications using them. . . . 94

5.1 Functions to define conditions in abstract events . . . 119
5.2 Alerts from DARPA 2000 representing the infection of Mill . . . 126
5.3 Implementation in JSON of condition functions . . . 129

6.1 Parameters of Morwilog . . . 139
6.2 Nodes containing the propagation steps in WannaCry . . . 141
6.3 Theoretical results for continuous reinforcement . . . 165

7.1 RealSecure IDS alerts . . . 171
7.2 Snort alerts in ISCX dataset . . . 173
7.3 Origins of logs in HuMa dataset . . . 174
7.4 Fields used in parsed events . . . 175
7.5 Summary of datasets . . . 176
7.6 Parameters to create the *eventgen* datasets . . . 176
7.7 Values of T_{max} for evaluation . . . 177
7.8 Results of AASG S2A . . . 190
7.9 Time performance of Morwilog and Bidimac . . . 192

8.1 Regular expressions for timestamps . . . 205
8.2 Shapes of nodes in Logan . . . 211

D.1 Similarity-based methods on progressive construction . . . 248
D.2 Similarity-based methods on clustering and anomaly detection . . . 249
D.3 Causal correlation methods on prerequisites and consequences . . . 250

D.4 Causal correlation methods on statistical inference 251
D.5 Causal correlation methods on statistical inference (cont'd) 252
D.6 Structural-based methods 253
D.7 Case-based methods 254
D.8 Case-based methods (cont'd) 255
D.9 Mixed methods 256

E.1 Functions used in the AASGs 257

Chapter 1

Introduction

Contents

1.1 Motivation 2
1.2 Goals 3
1.3 Contributions 5
1.4 Publications 7
1.5 Thesis outline 8

The secret to getting ahead is getting started.
— Mark Twain

Computer networks have become an indispensable tool in our everyday life. Thanks to them we can communicate with other people, share files, do shopping, complete paperwork or even make a reservation in a restaurant. It would be difficult to imagine nowadays life without them and their world-scale interconnection, the Internet. But despite its advantages, the interconnectivity of computers involves serious threats.

Think of WannaCry, a massive cyberattack against computer networks which caused havoc during Spring 2017. More than 230,000 computers in 150 countries, such as Russia and Spain, were infected [Ehrenfeld 2017]. The most affected organization was the Britain's National Health System (NHS), where almost 20,000 appointments were cancelled [Check Point 2018]. The attackers took advantage of a vulnerability in the Server Message Block (SMB) protocol, used for providing shared access to resources, to introduce a malicious software which propagated itself and encrypted the hard disks of infected computers. The objective was to ask for a ransom in exchange of the decryption key. WannaCry is an example of a *multi-step attack*, term used to denote a cyberattack composed of different phases or steps.

1.1 Motivation

As in any other cyberattack[1], the perpetrators of WannaCry take advantage of the vulnerabilities present in computer networks. Apart from unknown vulnerabilities [Bilge 2012], that cannot be patched until they are detected, there exist publicly disclosed ones that may not be able to be corrected for the sake of flexibility [Wang 2008]. Therefore, an early identification of ongoing cyberattacks is important to minimize their impact.

Organizations rely on tools, processes and analysts [Sundaramurthy 2015] to identify attacks. The analyst, a cybersecurity expert, is a key piece [Caulkins 2018] who incorporates human creativity and a background built by experience that is rarely fully captured by automatic machines [Vert 2018]. But when facing multi-step attacks such as WannaCry, the analyst can be overwhelmed trying to determine which were the exact actions taken by the attackers and in which order.

Moreover, once a multi-step attack is detected, it is important to determine how it could happen next time, so the organization can be better prepared against it. But the mere presence of several steps instead of just one makes things harder [Ussath 2016a]. This is evident from the perspective of basic combinatorics. Imagine that we detect the sequence of actions A-B-C, which represents a multi-step attack with three steps. It may be that any future occurrence of the same attack follows exactly the same sequence, because action C requires action B to be performed before, which at the same time requires action A. In this case there is a strong causality and a detector can be developed to exactly match the sequence A-B-C.

But the actions within a multi-step attack are not necessarily linked by a strong causal relationship. We can consider, for example, a sequence of actions where a malware is installed in a computer within the network (action A), and then a message to the attacker's server is sent to request additional software for future lateral movement (action B) before a privilege escalation is performed (action C). If the downloaded software is not necessary for doing the privilege escalation, both sequences A-B-C and A-C-B are possible. This example considers only three actions, but the number of them can be higher, which would normally increment the number of possible sequences.

Furthermore, there exist actions executed by an attacker that seem to belong to the attack itself but that are not part of it. These actions can be attempts of parallel attacks, exploratory probes or even purposeful noise to distract the defender. In some cases, it can be very difficult to distinguish the fundamental actions of the attack, the

[1] We abbreviate it as 'attack' from this point onwards.

Event	Asset	Time	Description
e_0	Proxy	09:10:34	Unsuccessful HTTP request from a host h_A to long domain name D_1
e_1	Proxy	09:10:42	Successful HTTP request to long domain name D_2
e_2	Firewall	09:10:54	Successful connection in port TCP 445 from h_A to h_B
e_3	Endpoint h_B	09:11:04	Malformed SMB headers for the NEGOTIATE PROTOCOL REQUEST from h_A to h_B
e_4	Proxy	09:11:28	Successful HTTP request to long domain name D_2
e_5	IDS	09:11:40	SQL injection alert coming from h_A
e_6	Firewall	09:12:38	SMBv1 communication between h_A and h_B using command 'transaction2_secondary'

Table 1.1: List of events containing an instance of WannaCry

ones which are repeated each time the attack takes place.

We see an example of this in Table 1.1, where we represent a list of events that happened during an execution of WannaCry. Among all the events shown, only the sequence of events $e_0 - e_2 - e_6$ is the one truly representing WannaCry. This sequence is camouflaged among the other events.

Detection of future instances of an attack is thus hindered by the independence of actions composing it and the presence of side actions not contributing to the goal of the attacker. Given a set of actions such as the one in Table 1.1, a security analyst can think of several alternative attack scenarios. The analyst works then upon a number of hypotheses or cases, without being certain of which one would match a future manifestation of the attack.

Taken this context, we can define our research question:

How can we help the security analyst to decide between alternative scenarios of multi-step attacks in order to ease the detection of future occurrences? Can we perform detection at the same time as the learning process evolves?

1.2 Goals

The actions executed by the authors of a cyberattack generally leave a trace in the victim network. For instance, devices in the network keep a journal of activities, whose entries are called *logs*, that can be exploited by a cybersecurity team in order to find the trace of the attack actions. There are three major search activities according to the period of time on which the cybersecurity analyst is focused: the past, the present or the future. If the analyst is focused on finding the effects of a cyberattack that happened in the past, we talk about ***investigation***, also called *forensics* or *forensic investigation* [Khan 2016]. When the goal is to find cyberattacks occurring in the

present moment, in order to fight against them and avoid further consequences, the activity is called **detection** [Zuech 2015]. Detection models referring to a specific type of cyberattack are commonly known as *signatures*. Finally, if we want to know in which possible ways the network could be attacked so we can be prepared against future threats we talk about **prediction**, that have *risk assessment* as one of its fields of action [Cherdantseva 2016].

The processes of prediction and detection are based on models. These models are built from the conclusions obtained from the investigation of past cyberattacks, together with what we call the *structural information* of the network. This information is the one that can be obtained by the analysis of the elements in the network assuming that no traffic exists between them. It contains, for example, the characteristics of assets, the vulnerabilities, the network configuration, among others. The analysis of structural information is out of the scope of this thesis. We are only focused on the dynamic information extracted from the actions performed in the network.

During the investigation there is a process of abstraction in which the analyst takes the available information and builds models from it. On the contrary, detection and prediction are a concretion of built models in order to search for instances of cyberattacks in the traces. This vision is represented in Figure 1.1. The stated research question (page 3) refers to the abstraction of the results coming from the investigation and to the confirmation of the defined abstract models. We call these two processes **modelization** and **identification**, respectively.

Motivated by the context presented in section 1.1, in this thesis our objective is to help the security analyst in the development of multi-step attack detection models from

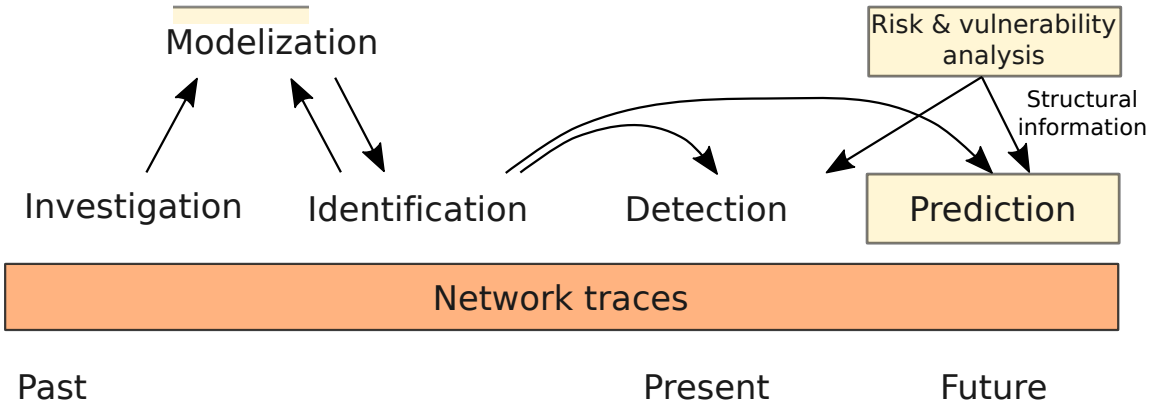


Figure 1.1: Diagram showing the different processes when finding cyberattacks. The temporal line below refers to the moment when the cyberattack happened with respect to when each process takes place.

an initial set of alternative scenarios. This process is executed in two phases. First, the analyst detects the symptoms of a multi-step attack and deduces a list of possible attack scenarios (modelization). Second, a human-guided algorithm is applied to learn which of the proposed scenarios is the one representing the successful cyberattack (identification). But to avoid missing another occurrence of the same multi-step attack, the modeled set of alternative scenarios have to also act as a detector, so detection alarms[2] are sent to the analyst while the learning process is performed. Feedback from the analyst is then used in the learning process. This problem has not been solved yet by the literature about multi-step attack detection, as we will see in Chapter 4.

This Ph.D. thesis has been financed by the French public investment bank Bpifrance (*Banque publique d'investissement*) in the frame of AAP-19 project. This project is called *HuMa* (*L'humain au cœur de l'analyse de données massives pour la sécurité*) and counts with the collaboration of both public institutions and private companies. The objective is the creation of a system for the analysis of complex attacks in massive sets of logs[3].

1.3 Contributions

In this thesis we present a series of contributions, that we introduce below, in order to answer our research question.

1.3.1 A survey on multi-step attack detection

Methods for multi-step attack detection has been developed since the beginning of 2000s, but the field lacked of an analysis of its evolution. One of our contributions is the first systematic bibliographic research about multi-step attack detection on traces. Only the methods focused on the detection of the whole structure of the attack and not of individual symptoms are considered. A total of 138 different methods presented in 201 publications are studied. This has allowed us seeing up to which point existent methods have contributed to answer our research question. The statistics extracted from this systematic survey are also intended to help future researchers in the development of multi-step attack detection methods. This contribution is presented in Chapter 4.

[2]We use the term 'alarms' to denote the alerts generated by the methods proposed by us. This allows the distinction with other types of alert, which could even be part of the input of the methods (see section 2.3)

[3]http://www.huma-project.org/

1.3.2 Abstract Attack Scenario Graphs (AASG)

In the scope of this thesis, events, represented as logs, are considered as the traces studied to search for the actions of the attackers. The analyst needs a mechanism to represent the alternative multi-step attack cases derived from investigation. We propose a new model, the Abstract Attack Scenario Graph (AASG), where these cases are represented to perform detection and identification on new incoming events. In opposition with the representation of a specific instance of a multi-step attack, that we call Concrete Attack Scenario Graph (CASG), an AASG can capture the abstraction involved in the deduction of future cases of the attack.

Alternative cases are arranged as branches of the AASG, which is a single-source directed acyclic graph (DAG). Each node in an AASG contains an abstract event, which corresponds to a set of possible events. Each abstract event can be seen as a set of rules indicating which events can be identified with the node. The representation of abstract events is based on the functions that define the relationship between two steps in a multi-step attack, as they are considered in the literature (see Chapter 3). This give the analyst enough flexibility to capture the conditions in the proposed hypotheses. AASGs are conceived to be easily read and interpreted by the analyst, which also eases the sharing with the community. Abstract events and the structure of the AASG are defined in Chapter 5.

1.3.3 Detection and identification of scenarios

An AASG is intended to be used by algorithms designed to perform identification and detection of the correct multi-step attack cases. Another contribution of this thesis is the proposal of two algorithms for this task: Morwilog and Bidimac. In both of them, the analyst is directly involved in the process, providing feedback about the malicious nature of the detected sequences.

Morwilog is an ant colony-based model where an artificial ant, called *morwi*, is created each time an event matching the root node of an AASG arrives at the system. When an AASG is used in Morwilog, a certain level of artificial pheromones is assigned to each one of the arcs, becoming a *stigmergic AASG*. The *morwi* goes through the stigmergic AASG waiting for the arrival of events matching subsequent nodes and randomly choosing a path based on the level of pheromones in each arc. The chosen path thus depends on previous history. The levels of pheromones are modified based on the feedback given by the analyst, that receives alarms each time a *morwi* detects a full multi-step attack case.

On its side, Bidimac is an adaptation of classical Bayesian inference for working with AASGs. The graph becomes then a *Bayesian AASG*, with probability parameters assigned to each arc. Every found case by Bidimac raises an alarm that the analyst has to evaluate. Her feedback results in the modification of the probability parameters in the Bayesian AASG. Due to the novelty of AASGs and the concept of identification of multi-step attacks, Bidimac offers a way to compare Morwilog with a classic method such as Bayesian inference.

Both Morwilog and Bidimac can detect the cases represented in the AASG while performing the identification of the correct cases. Methods in the literature of multi-step attacks do not consider the human process of thinking of a set of possible cases and confirming correct ones. Concerning detection, the originality of our proposal resides on its human-centered approach. The analyst gets involved in the process, going beyond the classical tasks of alert verification after detection or development of models and methods before detection. Morwilog and Bidimac do not work as a black box, but as assistance mechanisms where the human can control at any moment how the system learns and which are the conclusions drawn from the learning process.

1.3.4 Visual investigation of attack scenarios

Although our biggest contributions are the ones already mentioned, there is an additional proposal that has been developed when exploring the investigation process leading to the construction of AASGs. This model, called SimSC, is used to investigate attack scenarios in sets of raw logs. It has been conceived to be used together with the methods developed by the rest of partners in the HuMa project, the project that has funded this doctoral research. SimSC is based on the automatic extraction of IP addresses and timestamps from the raw logs and the construction of graphs based on the similarity between these features.

1.4 Publications

During the elaboration of this thesis, we have published a series of articles and conference papers related to the proposed research question:

Journal articles:

- Navarro, J.; Deruyver, A. & Parrend, P. **A systematic survey on multi-step attack detection**. Computers & Security, Elsevier, 2018, 76, 214-249
- Navarro, J.; Legrand, V.; Deruyver, A. & Parrend, P. **OMMA: open architecture for Operator-guided Monitoring of Multi-step Attacks**. EURASIP Journal on Information

Security, Nature Publishing Group, 2018, 2018, 6

- Parrend, P.; Navarro, J.; Guigou, F.; Deruyver, A. & Collet, P. **Foundations and applications of Artificial Intelligence for zero-day and multi-step attack detection**. EURASIP Journal on Information Security, Nature Publishing Group, 2018, 4

Conference papers:

- Parrend, P.; Guigou, F.; Navarro, J.; Deruyver, A. & Collet, P. **For a refoundation of Artificial Immune System research: AIS is a Design Pattern**. IEEE Symposium Series on Computational Intelligence (SSCI), 2018, 1122-1129
- Parrend, P.; Guigou, F.; Navarro, J.; Deruyver, A. & Collet, P. **Artificial Immune Ecosystems: the role of expert-based learning in artificial cognition**. Proceedings of the 2018 International Conference on Artificial Life and Robotics (ICAROB), 2018
- Navarro, J.; Legrand, V.; Lagraa, S.; François, J.; Lahmadi, A.; De Santis, G.; Festor, O.; Lammari, N.; Hamdi, F.; Deruyver, A. & others. **HuMa: A multi-layer framework for threat analysis in a heterogeneous log environment**. International Symposium on Foundations and Practice of Security, 2017, 144-159
- Navarro-Lara, J.; Deruyver, A. & Parrend, P. **Morwilog: an ACO-based system for outlining multi-step attacks**. IEEE Symposium Series on Computational Intelligence (SSCI), 2016, 1-8

1.5 Thesis outline

To end up with this introduction, we present here an outline of the chapters of this thesis and a brief description of their content:

- **Chapter 2. Preliminary definitions.** Definition of basic concepts such as *cyberattack*, *trace* or *multi-step attack*, as they are understood throughout this thesis.
- **Chapter 3. Multi-step attacks.** Analysis of the characteristics of multi-step attacks and the approaches used to model them.
- **Chapter 4. Multi-step attack detection.** A systematic survey on multi-step attack detection methods.
- **Chapter 5. Abstract Attack Scenario Graphs (AASG).** Definition of an Abstract Attack Scenario Graph and its elements.
- **Chapter 6. Algorithms to exploit AASG.** Definition of the algorithms Morwilog and Bidimac and comparison between the two.
- **Chapter 7. Evaluation.** Results of the evaluation of Morwilog and Bidimac on different datasets using AASGs.
- **Chapter 8. Visual investigation of attack scenarios.** Presentation of SimSC, a model for the investigation of attack scenarios on raw logs.

- **Chapter 9. Perspectives.** Exposition of open research questions and proposition of future steps for further developing the contributions of this thesis.
- **Chapter 10. Conclusion.** Summary of the key contributions of this thesis and lessons learnt.

Chapter 2

Preliminary definitions

Contents

2.1 Computer networks under threat 11
2.2 Cybersecurity: objectives and mechanisms 15
2.3 Traces in a network 17
2.4 Definition of multi-step attack 20
2.5 Summary 23

If they can get you asking the wrong questions, they don't have to worry about answers.
— Thomas Pynchon, *Gravity's Rainbow*

We have seen that the context where our research takes place is that of computer networks and their exposition to multi-step attacks. In this chapter, we take a brief tour through the scenarios and definitions linked to our research. We start by an overview of the current state of computer networks and their exposition to cyberattacks in section 2.1. We then define what is Cybersecurity in section 2.2 and the traces that can be found in a network in section 2.3. Traces are the source of information for the detection of multi-step attacks, that are introduced in section 2.4. These attacks will be further analyzed in Chapter 3.

2.1 Computer networks under threat

Cyberattacks can pose problems to any computer network. Both, cyberattacks and computer networks, have evolved concurrently. The effects of the first ones are still evident, thanks to the unavoidable vulnerabilities present in systems and despite of the progress made in the field of Cybersecurity.

2.1.1 The battleground: a connected world

Unauthorized use, appropriation, destruction or illegitimate seizure of someone else's resources, according to the limit of private property defined by each society, is probably as old as civilization. In the Mesopotamian Code of Ur-Namma, the oldest known law code (ca. 2100 BCE), we already find rules to punish the citizens who steal someone else's possessions or cultivate the field of another citizen [Roth 1995].

Information is something abstract, independent of the physical format where it is stored. That is why it is more difficult to protect information than physical possessions. Once known, a piece of information could be considered as 'stealth'. Interests on stealing information can be very numerous, as to disclose it publicly, to gain a competitive advantage or to get something else through extortion. But information can be also modified, and false information can be created for a dishonest purpose. Moreover, a source of information can be eliminated (e.g. 'erasing' someone knowing a secret) or the physical support containing the information can be destroyed.

A first requisite to alter, steal or erase information is to have access to it. Before the invention of digital computers, there were two ways to get unauthorized access to encoded information[1]: having physical access to the support where the information was stored or intercepting the *communication*, the process of information exchange [Huurdeman 2003]. Communication required a human emitter to initiate and coordinate the actions for sending the message. However, digital computing brought the possibility of making requests to a remote machine that automatically serves the information, without human intervention. The computer has access to the information in digital form and counts with a set of instructions to follow in respond of each type of request.

The evolution towards automatic remote access was progressive. Even if electronic storage of digital data was already possible since the invention of the Williams-Kilburn Tube at the end of the forties [Williams 1951], the remote connection of computers was not possible until 1965 [Marill 1966]. A few years later, in October 1972 [Leiner 1997], the first public demonstration of ARPANET, which soon became the Internet, took place. The ubiquitous access to information brought by the Internet since then has deeply transformed society. Now we can share a video with the rest of the world or send an image to a friend living in another continent in a matter of seconds. Email has become the preferred way of communication in business, and social networks allow us keeping in touch with people living abroad or even start a revolution. Lots of activities, including administrative transactions, increasingly rely on the Internet.

[1] 'Encoded' to distinguish it from information known by a person but not written down.

But the Internet also means global interconnectivity of computers. And since the moment when a user has remote access to a computer, he or she can try to exploit the weaknesses present in the machine or in the provided services to get unauthorized access. This brings a wide spectrum of new remote ways to commit crimes, from the illegal appropriation of intellectual property to the stealth of personal information, the ransom of critical data or the disruption of services. All that with just a computer located anywhere.

2.1.2 Threatened by cyberattacks

Crime committed exploiting remote network access exists since the beginning of computer networks [Akhgar 2014]. The goal of unauthorized access to network is not necessarily to get illicit economic benefit [Duffany 2018]. There are users who do it for the challenge, as a way to learn, to protest against an institution, to have access to shareable resources, to do a joke, etc.

In any case, from the point of view of the defended system we consider all these actions as *cyberattacks*. The definition of cyberattack followed in this thesis[2] is the one that follows: a set of actions, via cyberspace, targeting an organization's use of computer networks for the purpose of disrupting, disabling, destroying, exposing or controlling without authorization a computing environment/infrastructure; or destroying, modifying or stealing data or blocking the access to it.

The prefix 'cyber' intuitively points to something related to the culture of computers and the Internet. Once the context is well defined, we can refer to 'cyberattacks' just with the general word 'attack'. In this thesis, only attacks performed intentionally and using digital methods are considered. That means that we do not consider any physical contact between the attackers and the victims.

We mention in the definition of cyberattack the word 'cyberspace'. It is a neologism which appeared for the first time in July 1982, in the short story "Burning Chrome" [Gibson 1982] by the science fiction author William Gibson, and that was soon incorporated into the technical language as a way to refer to the digital environment created by the Internet. More formally, The Oxford English Dictionary tells us that the cyberspace is "the notional environment in which communication over computer networks occurs" [OED 2018b]. This englobes not only communications over the Internet but also those done within local networks. The computers or other devices

[2]It considers elements from the definitions given by the US Committee on National Security Systems (CNSS) [CNSS 2015], the US Federal Financial Institutions Examination Council (FFIEC) [FFIEC 2018] and the ISO/IED 27000 standard [ISO/IEC 2016]

involved in the communication are also part of the cyberspace.

Throughout this thesis, the agent performing a cyberattack is called an *attacker*, a neutral term only addressing the action and not presupposing the objective or nature of the agent[3]. In contraposition to this term, the entity responsible of preserving the target system against potential attackers and, if these last ones are successful in their attempts, of detecting the attack and mitigate its effects is called *defender*. The attackers focus their effort on different elements on the attacked system [Stallings 2015]: either a piece of data, a service, a system capability (processing power or bandwidth, for example) or an item of system equipment (software or hardware). From the point of view of the defense of computer networks, and to do abstraction of their characteristics, these elements are called *assets*. An asset can be the final target of an attacker or just a partial step in the pursuit of another asset.

2.1.3 Constrained by vulnerabilities

Attackers can act because there exist vulnerabilities in the assets they target. A *vulnerability* is a flaw or weakness in the design or implementation of a piece of software or hardware that can be exploited by an attacker to perform an unauthorized action [Stallings 2015, Duffany 2018]. A defender has to deal with attacks in a scenario composed of vulnerable assets. There exist *known* and *unknown vulnerabilities*. The defender is aware of the existence of the known ones, whose exploitation can be prevented by the development of specific security mechanisms. However, unknown vulnerabilities are the ones that have not been found yet by the defender. Their existence is assumed when deploying the defenses of the organization.

Attack detection has a special relevance in current computer networks as the number of available services increases, bringing an uncontrollable amount of vulnerabilities. New vulnerabilities are discovered everyday. Only in 2018 a staggering total of 16,555 new vulnerabilities were incorporated into the CVE (Common Vulnerabilities and Exposures) database [CVEDetails 2019], around 45 per day. Figure 2.1 shows the impressive increase in the number of reported vulnerabilities per year since 2000.

The classification used for vulnerabilities can be also extended to attacks. An attack is *known* if it has been widely detected, analyzed and understood by the security industry, so prevention mechanisms against it can be effectively deployed and a signature characterizing it could be developed for its detection. Once an attack is known,

[3]We do not judge in any case the sense of morality behind the actions of an attacker. A cyberattack is not forcibly seen by everybody as immoral (e.g. WikiLeaks, Anonymous). That is why we avoid in this thesis the terms *hacker*, with positive connotations, or *cybercriminal*, with negative ones

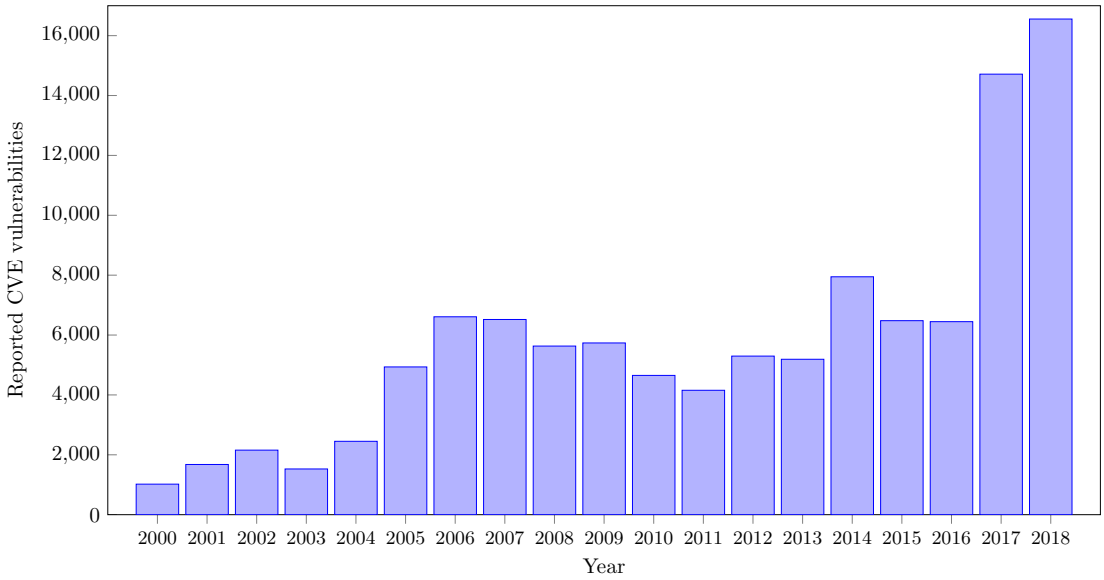


Figure 2.1: Number of reported CVE vulnerabilities per year. Data extracted from CVE Details [CVEDetails 2019]

the vulnerabilities exploited during its execution also become known. On the other hand, *unknown* attacks rely on unknown vulnerabilities or on ways of access that were not considered by the defender.

2.2 Cybersecurity: objectives and mechanisms

The domain studying and implementing methods against cyberattacks is called *Cybersecurity*. A complete report about the formal definition of the term 'Cybersecurity', also written 'Cyber Security', was published in December 2015 by the European Union Agency For Network And Information Security (ENISA) [Brookson 2015]. It analyzes the definitions given by the most relevant standard organizations. Surprising as it may seem, each one defines a different scope for the usage of the term. For example, the US CNSS [4], the NATO and the ITU-T consider within the scope of the term the 'physical security', the protection against physical threats which can affect the well-functioning of computer networks such as physical access to servers or insertion of malicious hardware. On the contrary, the ISO and the US NIST do not consider it. According to the report, it is not possible to formalize a definition to cover everything the term suggests to cover. The authors even conclude that there is no need of a conventional definition like the ones we apply to simple concepts, as 'Cybersecurity' is an enveloping term whose scope depends on the context where it is used. Craigen et al. [Craigen 2014]

[4]The full name of each organization can be found in Appendix B.

try to capture the enveloping spirit of the term in an inclusive definition, well-chosen but too broad for the scope of this thesis.

Having defined our scope, we have coined a definition of Cybersecurity as it is understood throughout this thesis: the organization and collection of resources, processes, and structures used to protect cyberspace, cyberspace-enabled systems and the information contained on them from intentional cyberspace-based actions that compromise their confidentiality, integrity and availability.

According to the report published by ENISA, each definition of Cybersecurity contains a series of components. They represent them as a diagram, which is reproduced in Figure 2.2. We have shaded in gray the components included in our definition. The name of each component is self-explanatory. A full definition of them is given in the original document [Brookson 2015].

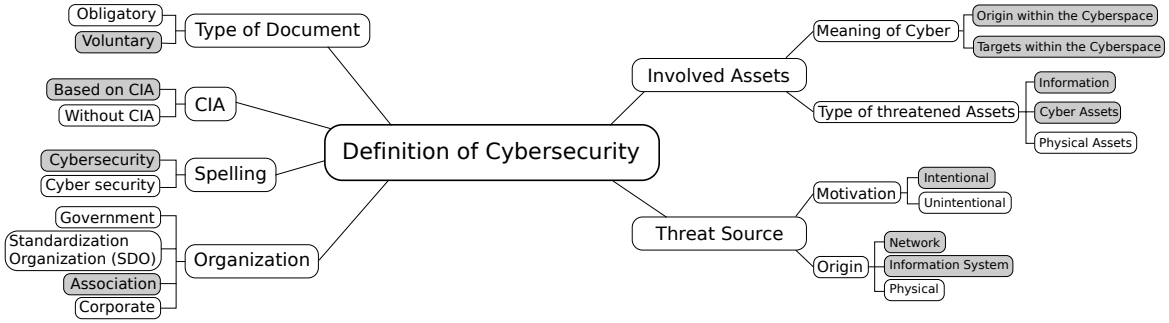


Figure 2.2: Components of a definition of 'Cybersecurity' according to ENISA [Brookson 2015]. The components considered in our definition are shaded in gray.

The three last properties mentioned at the end of the definition of 'Cybersecurity' compose the CIA paradigm. The description of each one of the properties [Qian 2008, Stallings 2015, Duffany 2018] is given below:

- **Confidentiality**: Information is disclosed only to authorized users.
- **Integrity**: Assets or messages exchanged between users can only be modified in a specified and authorized way.
- **Availability**: Access is guaranteed only to authorized users at any moment in a timely manner.

The preservation of those properties in every asset is an ideal goal for any organization, although it may be impossible to attain. Anyways, the goal is not to be achieved by a single security service, but it is the whole Cybersecurity strategy set up by the organization which could eventually guarantee the safeguarding of these properties [Qian 2008].

There is another property, *accountability* [Qian 2008, CNSS 2015], which ensures

that actions performed by each user can be traced uniquely to that user. Its preservation is not an essential task of Cybersecurity but provides a source of valuable information to perform attack investigation and detection. Checking the actions performed in the network can help to reconstruct those ones oriented toward the same malicious goal and, therefore, being part of the same attack. Each one of these actions is also called a *'step'* of the attack.

2.3 Traces in a network

The place of the defender is the one taken in the study of multi-step attacks done throughout this thesis, the same generally taken by the literature about attack detection. We could imagine an alternative model built from the point of view of the attackers if we take as a base, for example, the pieces of code or exploit kits used in every step [Ghosh 2015]. Placed on the defender's side, the only sources of information we have for knowing the actions performed by the attackers are the assets, whose capacity of recording the activities of users is guaranteed by the property of *accountability*.

We denominate *trace* to each preserved piece of information about actions performed within a computer network. The use of this term in Cybersecurity refers to the definition of the term 'trace' given by the Oxford English Dictionary: "a mark, object, or other indication of the existence or passing of something" [OED 2018f]. We denote the ensemble of all the possible traces by Θ.

A trace [OED 2018c] is then the representation of something that happened, of an event. The same as our actions as individuals are limited by Nature and its laws, computer networks establish an environment where actions are limited by the technical characteristics of the used devices. The point of contact between the user and the network, the interface, offers a big but finite number of possible actions. In any case, the number is much smaller that the actions a human can perform outside the network. Events happening in the network are the consequence of actions, so they have also a reduced number of possibilities. But this number is even smaller when they are expressed as traces, as they have to be translated to a specific code of symbols to be understood by humans and machines. On top of that, timestamps in traces have a certain limit in accuracy. Different actions can then lead to the generation of identical traces associated to the same moment.

A trace counts with a series of attributes that we can call *inherent*, as the own information of the trace is enough to determine them. Opposed to them we could

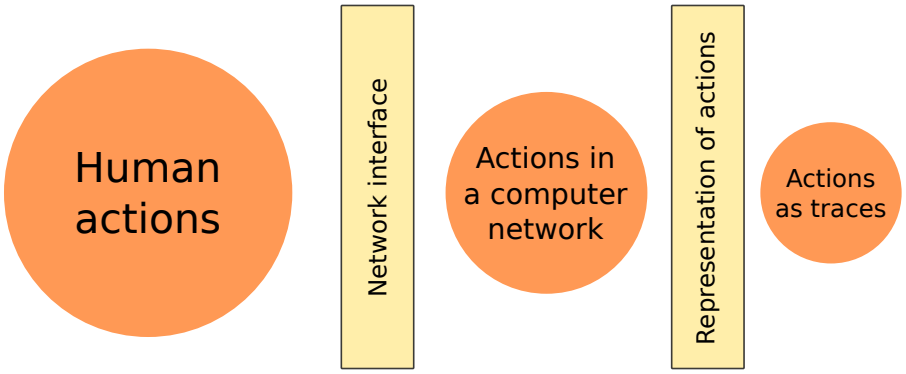


Figure 2.3: Diagram representing the range of actions in each environment.

define *extrinsic* attributes, derived from the place of the trace among a set of traces. In other words, extrinsic attributes depend on external complementary data coming from other actions, while inherent attributes do not. For instance, an inherent attribute could be the name of the machine generating it, which is usually contained within the information in the trace. Conversely, an example of extrinsic attribute is the probability of having a trace of the same type generated in a period of 5 seconds.

More formally, a trace $\theta_a \in \Theta$ can be represented as a finite tuple of inherent attributes, each of them containing information about a specific characteristic of the trace. We use att_a for referring to a generic attribute. The name of the attribute att defines a unique meaning for each attribute $\theta_a(att) = att_a$ in the trace θ_a. For example, commonly used names are *ipsrc* (source IP address), *time* (timestamp) or *type* (type of the trace). Names establish a correspondence with other traces, which could also contain attributes with the same name and, therefore, the same meaning. Moreover, a name univocally defines the type of value of the attribute, which could be a real number, an element from a finite set or an IP address, among others. The operation of attribute selection can be extended to be used with a set of attributes A, so $\theta_a(N) = A$, with N the set of names corresponding to the attributes in A.

In the context of this thesis we deal with two kinds of trace, defined below: *events* (including *alerts*) and *packets*. Other work, such as the one focused on malware analysis, can find useful other types of trace, as for instance the files left by a malicious software in the victim computer. The following definitions are issued from the wide use of the term in the reviewed literature and our own experience in the Cybersecurity industry.

- **Packet.** It is the minimal unit of data exchanged in a network communication protocol. In Cybersecurity, the word *network* is usually preceding the term to

specify the environment. A packet is composed of the transmitted data and the header [White 2017]. The latest describes the nature of the former and includes the instructions for handling the packet in the context of the communication protocol used. Apart from its function in communication, a packet indirectly contains information about an action that happens between two or more network assets.

- **Event.** In this thesis we consider the definition given by the Standard on Logging and Monitoring published by the European Commission in 2010 [EC 2010]: "an event is an identifiable action that happens on a device", considering that 'device' is equivalent to 'asset'. We refer to 'events' when this information is processed and stored by the own asset. The distinction is then clear from the general definition of event as "a thing that happens" [OED 2018c], under which packets would be also the manifestation of events (the communication processes between assets).

  An event is considered to be bound to the specific point in time when it took place [Meier 2004]. Events are usually represented as plain text in a format dependent on the used technology. This materialization of an event, stored in the asset, is called *log*. *Parsed logs* are the ones that have been preprocessed in order to make their attributes distinguishable, transformed into a format suitable for security analysis. On the contrary, *raw logs* are expressed in their original format as they are generated by the asset.

  The process of transforming a raw log into a parsed log is called *normalization*. The piece of software to do this job is called a *log parser*. In the industry, parsers are usually developed by the engineers working for the vendors and sent to the user. The problem faced by the manually created parsers is the need to be updated after changes in the log formats. Some interesting alternatives for automatic parsing exist, such as the ones based on NLP (Natural Language Processing) [Kobayashi 2014] or the LTE (Log Template Extraction) method [Ya 2015], but further development is needed until they can be applied to any log format.

  Security systems in charge of log collection, including normalization, and correlation [Kavanagh 2015] are known as SIEMs (System of Information and Event Management). Their main purposes are the unification and reinterpretation of logs collected from other assets in the network and the generation of alerts through event correlation (see page 57).

  Current assets are conceived to register and save a certain number of logs related

to their functioning and to their relationship with other assets and with users. For example, a router can generate logs containing information about failing interfaces or details about communications managed by it. Note that logs are created to work just as traces, while packets have a determinant role in the functioning of the network, independently that they can be used as traces.

- **Alert.** Also referred to as *alarm*[5], it is an event generated by a security system in response to the detection of alleged malicious activities or faults [Salah 2013]. In other words, it is an indicator that something is not working as it should or that a resource is not properly used. As its name suggests, an alert demands the security analyst to be vigilant. An alert is an event, but not all events are alerts. There exist events reporting normal functioning of systems. As any other event, alerts are usually represented as a log. Alerts are generally triggered by an Intrusion Detection System (IDS), a piece of software or hardware conceived to automate the attack detection process [Liao 2013].

To complement the given definitions, Figure 2.4 shows a visual representation of the considered types of trace.

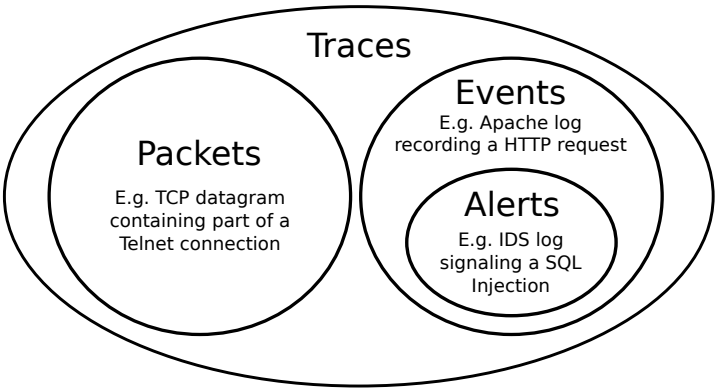


Figure 2.4: Diagram representing the scope of traces, packets, events and alerts. Every alert is an event signaling a security incident. Both events and packets are traces.

2.4 Definition of multi-step attack

The term 'cyberattack', or 'attack', is popularly identified to an individual malicious action. For instance, we speak of SQL Injection attacks [Nagpal 2017] or Denial of Service (DoS) attacks [Peng 2007], where just one action, possibly repeated, is required

[5]In this thesis, the term 'alarm' is exclusively used to denote an alert generated by Morwilog and Bidimac (see Chapter 6).

to threaten the system. We can call them 'single-step attacks'. In contraposition to them, when an attack is composed of several different actions it is called a *multi-step attack*

2.4.1 Sequences of actions

Attackers usually follow a step-by-step strategy in their attempts to attack a system [Qian 2008], thus performing multi-step attacks. There are two reasons to proceed in this way, both related to the probability of success of the attackers.

First, it is the only way to get access to the most valuable resources in a complex network topology with different layers of security. Considering that the most important assets in terms of information value are in the less reachable areas of the network [White 2017] and that more resources are deployed to protect and control the assets with higher sensitivity [Rhodes-Ousley 2013], it would be almost impossible to successfully complete an attack against them performing a single-step attack only. A progressive step-by-step access towards the deepest part of the network seems necessary for the success of the attack. The attacker can take control of intermediary assets, called *stepping stones* or *laundering hosts* [Strayer 2005, Goutam 2017, Daud 2018], in the hope of finding new information about the path to follow or an easier access to the final goal.

Second, if the attack is decomposed in several steps, it is more stealthy and difficult to be identified by the defender, especially if some of the steps do not pose a risk to the system by themselves. The correlation of more than one action is needed to understand how the attack works and to identify the threat. Even if some authors claim that an attack with several steps is composed only of elementary attacks [Brogi 2016], this is not necessarily true, and each individual step may seem harmless [Chintabathina 2012], which makes detection more difficult.

In contraposition to single-step attacks, attacks of this kind are called *multi-step attacks*. The name addresses the ensemble of steps taken by one or several agents with a single specific unauthorized objective inside the network. For an attack to be qualified as 'multi-step', it has to be composed of at least two distinct actions. If actions are similar between them, it should not be called a multi-step attack. For example, in a DoS attack against a device or a service we can find millions of packets but each of them represents an instance of the same type of action. We consider thus a regular DoS attack as a single-step attack. It would not be considered as multi-step even if it is launched from several locations, what we call a DDoS attack by annexation of the word 'Distributed'. Attacks concurrently executed by more than one attacker, whether

or not they are single-step or multi-step, can be also called *multi-agent*, *distributed* or *coordinated* attacks [Zhou 2010].

Since the beginning of 2000s [Cuppens 2001, Dain 2001b, Julisch 2001], the security research community has tried to propose solutions to detect this kind of attack and to predict further steps. If they started to be identified as a mere combination of regular attacks, they have later acquired relevance and even their own names, as it is the case of WannaCry [Mohurle 2017], cited in the opening paragraphs of this chapter.

Multi-step attacks have many alternative names: *multi-stage attacks* [Chen 2006], *multistage attacks* [Du 2010], *attack strategies* [Huang 1999], *attack plans* [Qin 2004], *attack scenarios* [Mathew 2009] or, finally, *attack sessions* [Cipriano 2011]. Some authors, such as Yu and Frincke [Yu 2007], even make a subtle distinction between 'multi-step' as the characteristic of being composed of "temporally and spatially separated legal and illegal actions" and 'multi-stage' as the term to indicate that the steps can be complex attacks by themselves. However, for most of the authors both terms are equivalent and correspond to the first definition. We prefer the term *multi-step attack*, the one used throughout this thesis, because it seems to be the one better reflecting the difficulties faced in the detection of this type of attack, as the main challenge resides in the fact that the attack is composed of many steps which can be of very different natures.

There are also some authors who talk about *Advanced Persistent Threats* (*APTs*) [de Vries 2012], *complex attacks* [Çamtepe 2007] or *targeted attacks* [Trustwave 2018], especially in the security industry. These three last terms refer to a type of multi-step attack that is specifically crafted against a single victim and where the access of the attacker to the target network is maintained during a long period of time. In the literature, most of the analysis and methods developed with a focus on APTs or targeted attacks address any type of multi-step attack.

2.4.2 The incidence of multi-step attacks

A complete multi-step attack needs to be represented by a group of correlated actions [Kawakani 2017]. That means that it is necessary to complement the process of detecting the individual actions with an additional process of finding the links between those actions [Ussath 2016b], highlighting the relationship between them. On top of this, the steps of some multi-step attacks can be scattered through several assets [Pei 2016]. This adds the additional difficulty of identifying and bringing together a set of heterogeneous and delocalized evidences. Moreover, this process is hindered by the high volume of events registered in the network [Zuech 2015].

The incidence of multi-step attacks in current computer networks is important. Most 2018 security predictions agree on the increasing sophistication of cyberattacks, which also points to an increment in the number of involved actions in a single attack [PandaLabs 2017, CrowdStrike 2018, Trustwave 2018]. An indicator of attack sophistication is the long *dwell time*, the period elapsed between the entrance of the attack in the network and its detection. If the time until detection is long, the attacker can perform more actions to get a goal. The figures given in several 2017 security reports are very similar, with an average dwell time never inferior to 80 days (see Table 2.1).

[Fire Eye 2018]	101 days of median dwell time
[CrowdStrike 2018]	86 days of average dwell time
[Trustwave 2018]	83 days of average dwell time

Table 2.1: 2017 dwell time values of attacks according to different sources.

There are at least two reasons for explaining the increasing incidence of multi-step attacks. First, the detection capabilities of organizations seem to have improved in the last years, as pointed out by Mandiant in its M-Trends report [Fire Eye 2018]. Second, the multiple security layers that protect the most valuable assets [White 2017] are forcibly multiplied [Albanese 2017] due to the increasing complexity of computer networks. This takes the appetizing assets further from the attacker. In consequence, attackers find more difficult to remain undetected and need to perform several intrusion steps to reach their final goal, aiming at being more stealthy and elusive [Pei 2016].

2.5 Summary

In this chapter, we have presented a series of definitions that are fundamental to fix the context where this research takes place. We have presented the concept of *cyberattack* and narrowed its scope to actions performed with a malicious intention using digital methods (sections 2.1.1 and 2.1.2). *Vulnerabilities* have been presented as an inevitable constraint of current networks (2.1.3) that impels us to the development of detection methods in the domain of *cybersecurity* (2.2). We have seen that *traces* preserve information about the actions of the users in any network (2.3). Finally, we have ended the chapter by explaining what are *multi-step attacks* (2.4.1) and considering their current incidence (2.4.2). This prepares us to the next chapter, where these attacks will be explored in detail.

Chapter 3

Multi-step attacks

Contents

3.1 Multi-step attacks in context 25
3.2 Examples of multi-step attacks 29
3.3 Modeling multi-step attacks 35
3.4 Connection of nodes in the CAGS 44
3.5 Summary 61

A man is nothing but the series of his actions.
— Hegel, *Encyclopedia of the Philosophical Sciences*

Cybersecurity is a discipline whose origin is determined by the existence of cyber-attacks. The threat they pose to computer networks justifies the deployment of solid defenses, detection processes and mitigation procedures. We have seen in the previous chapters that this thesis is focused on *multi-step attacks*, whose analysis will be the subject of the sections below.

We start our exposition in section 3.1 by giving a classification of multi-step attacks and clarifying the notion of Advanced Persistent Threats (APT), a subtype often unduly used as homonym. The chapter continues with the presentation of several examples of this kind of attack in section 3.2. In section 3.3, we review how multi-step attack are modeled. Finally, we analyze the existent types of relationship between the steps in section 3.4.

3.1 Multi-step attacks in context

The definition of multi-step attack was given in section 2.4. We complete below this definition by comparing the example of WannaCry with a single-step attack (section 3.1.1) and by offering a classification of multi-step attacks according to several criteria (3.1.2). Responding to the increasing popularity of the term 'Advanced Persistent Threat', we also clarify the place of APTs among multi-step attacks (3.1.3).

3.1.1 Comparing single and multiple steps

To be studied from the point of view of the defender, the actions performed by the attacker need to be captured on traces, collected and stored pieces of information about what happens in the network (see section 2.3). A whole single-step attack can be represented by only one trace. For example, a log from an Apache web server representing a successful SQL Injection attack is shown in Figure 3.1. Only this log suffices to describe the attack.

```
192.168.2.112 - - [06/Aug/2018:08:16:00 +0900] "GET /index.php?
page=store.php&action=buy&id=%27+or+1%3D1%23 HTTP/1.1" 200 4134
```

Figure 3.1: An Apache log representing a SQL Injection attack.

However, a multi-step attack has to be described by a consecution of traces. Think of WannaCry, the multi-step attack taken as an example during Chapter 1. We can identify the attack by a sequence of three actions. The first one is a HTTP request from a host h_A to a long domain name to test if the malware is executed in a sandbox. If the HTTP request is unsuccessful, the attack continues by a connection in port TCP 445 from h_A to another host h_B. Finally, there is a SMBv1 communication between h_A and h_B using command 'transaction2_secondary'.

At least these three steps, each one innocuous by itself, are necessary to characterize WannaCry. Note that the traces of these actions should not forcibly be searched in the same asset. The first one is more likely to be registered by an asset working with HTTP-related traffic, at the application level, such as a proxy or a DNS server. On the contrary, the other two are usually detected by a network asset able to deal with several layers in the OSI model, such as a firewall.

This set of steps constitutes a narrative of the attack. It is a straightforward matter to draw a temporal line joining the different actions or steps involved in the story. The result is a graph such as the one shown in Figure 3.2. We will see throughout this chapter that some other types of relationship between actions can be used to link them and model the attack as a graph, besides the temporal order.

3.1.2 A classification of multi-step attacks

In the reviewed literature, we have not found any reference proposing a classification of multi-step attacks. Nevertheless, the bibliographic research done during the elaboration of this thesis has revealed that multi-step attacks have certain features dif-

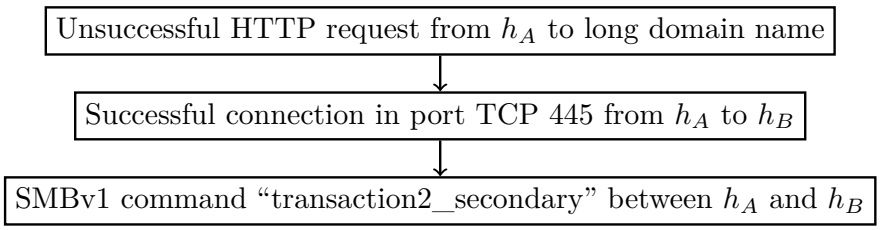


Figure 3.2: A sequence of actions revealing the presence of WannaCry.

ferentiating them. Based on these features, we propose the classification listed below and shown in Figure 3.3, according to four criteria.

- **Activity location**. A multi-step attack is **scattered** if the actions composing it take place in, at least, two different assets in the network. Conversely, it is **gathered** if all the steps happen in the same asset.
- **Attack agent**. An attack can be **human-driven** if the actions are directly executed by a human or **malware-driven** if the execution is automatically performed by a malicious piece of software. In a human-driven attack, the attackers gain knowledge about the target system after each of the steps of the attack [Katipally 2011] and are able to better prepare the subsequent steps. On the contrary, in malware-driven attacks, the malware tries actions from a list in a predefined order. Information about the state of the system is gathered but it is not processed by human creative thinking. There also exist attacks that can have parts driven by malware while others are directly executed by a human attacker. We call them **mixed-agent** attacks.
- **Presence of evidence**. An attack that does not raise an alert in any security system at the victim's organization is considered as **stealthy**. If its activity is signaled by at least one alert, it is called **discernible**.
- **Focus**. An attack can be **targeted** or **generic** depending on whether it has been specially adapted against the victim or not, respectively. A completely human-driven multi-step attack is considered to be targeted[1], as the behavior of the attacker is adapted to the particularities of the victim after each step.

When the term 'multi-step attack' or any of its synonyms is used, authors generally refers to *scattered human-driven discernible targeted attacks*. We will see attack examples and their classification later in this chapter.

It is important to note that the defined categories, except the one referring to the

[1]Do not confound a targeted multi-step attack with one using *targeted malware*, the one specifically programmed against the victim. A multi-step attack can be targeted even if standard software is used.

location of activity, could also be applied to single-step attacks. It is not the purpose of this thesis to offer a full classification of attack types. The classification is limited to the object of our study, multi-step attacks.

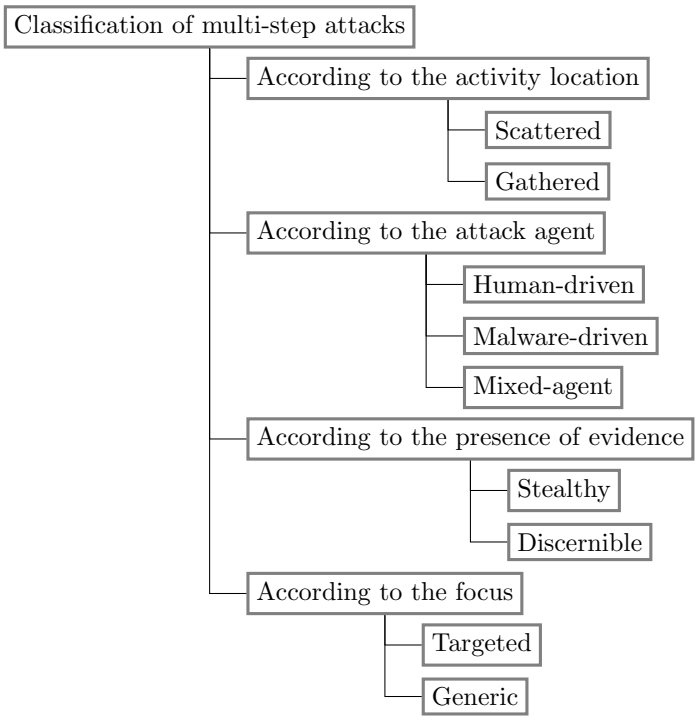


Figure 3.3: Classification of multi-step attacks.

Multi-step attacks can leave traces in the network where they take place. The ensemble of traces collected by any asset of the network containing information about any of the steps composing the attack is called in this thesis an *evidence set*. There are two kinds of evidence set: homogeneous and heterogeneous. An evidence set is *homogeneous* if all the traces contained in it have the same format, thus they are emitted by the same type of asset. A *heterogeneous* evidence set is composed of traces having at least two different formats, thus with a different origin. Among homogeneous evidence sets, if all the traces come from a single source the set is called *homogeneous single-source*. Otherwise, it is *homogeneous multi-source*. In Chapter 4, we will see that most of the past work dealing with multi-step attack detection use as an input a homogeneous single-source evidence set.

3.1.3 Advanced Persistent Threats

Since the beginning of the decade [de Vries 2012], the name *Advanced Persistent Threat* (*APT*) has been used to denote multi-step attacks that use specialized malware and

where the access of the attacker to the target network is maintained during a long period of time [Hu 2017]. Not every multi-step attack is an APT. As WannaCry infection [Mohurle 2017] has recently demonstrated, even global malware campaigns, that are not targeted, can be based on an attack that can be decomposed in multiple steps affecting several assets in the network.

The term APT has become somehow distorted after having been adopted by the industry and used in marketing campaigns. Moreover, a lack of rigorous definition of what an APT is [Chen 2014b] makes this term valid for other contexts, as for attacks using advanced malware but being single-step.

The steps involved in an APT and their order are described in the literature under many forms, depending on the author defining them [Luh 2016]. For instance, Chen et al. [Chen 2014b] define six stages: (1) reconnaissance and weaponization; (2) delivery; (3) initial intrusion; (4) command and control; (5) lateral movement, and (6) data exfiltration. In contrast, Hutchins et al. [Hutchins 2011] define seven stages: (1) reconnaissance; (2) weaponization; (3) delivery; (4) exploitation; (5) installation; (6) command and control, and (7) actions on objective.

3.2 Examples of multi-step attacks

In this section we give some illustrative examples of multi-step attacks. These attacks are used in the experiments performed to validate our scientific contribution (Chapter 7) and in the examples shown throughout this thesis. WannaCry, explained in section 3.2.1, is an example of *scattered malware-driven generic multi-step attack*. In terms of presence of evidence, it can be *stealthy* or *discernible* depending on the systems implemented in the attacked network. The rest of the presented attacks are *scattered human-driven discernible targeted multi-step attacks*: LLDoS 1.0 and LLDoS 2.0.2 (section 3.2.2); UNB ISCX island-hopping (3.2.3), and HuMa (3.2.4).

3.2.1 WannaCry

We have already talked about WannaCry, the most important cyberattack in 2017. It has been the most notorious example of a multi-step attack crafted for a massive campaign, infecting more than 230,000 computers based on around 150 different countries [Ehrenfeld 2017]. Due to its relevance, we have chosen it to illustrate the exposition of the methods and the examples given throughout this thesis.

WannaCry is based on the use of a *ransomware*, a type of malicious software that locks the access to the file system of a computer, asking for a ransom in exchange of

going back to normal functioning [Kharraz 2015]. To compromise the victim machines, WannaCry takes advantage of an exploit called EternalBlue, stolen from the United States NSA (National Security Agency).

The steps followed by WannaCry are listed below:

1. **Infection through EternalBlue**. EternalBlue exploits a SMBv1 vulnerability using remote code execution through the TCP port 445. After opening the connection with the victim, the exploit sends several 'transaction2_secondary' SMB requests and, thanks to an error in the processing of these requests, gets access to the system [Kulkarni 2018].

2. **Access to the victim**. The attacker guarantees her persistence on the victim machine using the DoublePulsar backdoor [CERT-MU 2017].

3. **Sandbox test**. The malware attempts to connect to a specific remote URL. The domain in the URL is not registered, so the connection in normal conditions would be unsuccessful. However, there exist virtual test environments (called *sandboxes*) with a mechanism that tries to fool malware code by giving a false signal of successful connection for any connection attempt. The theory proposed by researchers having studied WannaCry[2] is that this step is performed just to check if the malware is in a sandbox or not, as the execution only continues when the response to the request is unsuccessful [EY 2017].

4. **File encryption**. A new service is started by the malware for trying to encrypt all the victim's files using a random AES-128 key [Panda 2017].

5. **Demand of ransom**. A screen is shown to the victim asking for a ransom in bitcoins in exchange of the decryption key.

The malware used by WannaCry is a worm, so it has self-replicating capabilities [Kramer 2010]. The propagation through the network is done using the same EternalBlue exploit and therefore is only effective against assets running Windows and using a vulnerable version of SMB [Kulkarni 2018].

Notice that the first one of the representative actions represented in Figure 3.2 corresponds to the third infection step, while the two others represent the subsequent propagation of the malware (repetition of the first infection step).

[2]https://www.theverge.com/2017/5/13/15635050/wannacry-ransomware-kill-switch-protect-nhs-attack

3.2.2 LLDoS 1.0 and LLDoS 2.0.2

There are two multi-step attacks contained in the DARPA 2000 Intrusion Detection Scenario Specific Data Sets (see section 7.1.1, page 170), LLDoS 1.0 and LLDoS 2.0.2. The goal of both attacks is to take control of several hosts within the network and to launch from them a DDoS attack targeting an external victim. However, the followed process is not the same for the two.

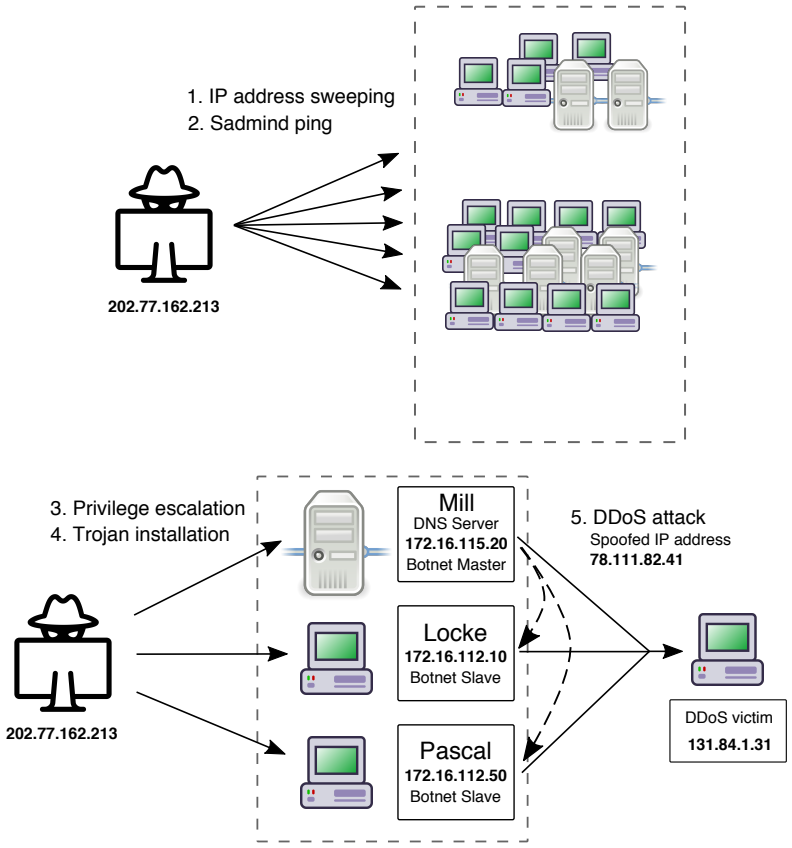


Figure 3.4: LLDoS 1.0 attack in DARPA 2000.

First, LLDoS 1.0 can be decomposed in the following five attack steps [Zhu 2006, Cheng 2011, Kavousi 2014, Ramaki 2016, Wang 2016, Holgado 2017], that are represented in Figure 3.4:

1. **IP address sweeping.** The attacker tests several IP addresses to identify which ones among them correspond to operative hosts. The scan is performed using ICMP (Internet Control Message Protocol) echo request packets [DARPA 1981a], known as 'pings', on several class C subnets in the network [Kavousi 2014].

2. **Sadmind ping.** Among the active IP addresses identified, the attacker searches for Solaris hosts with the *sadmind* program running. The *sadmind* program is a

daemon used for providing distributed system administration. It is implemented as a RPC (Remote Procedure Call), which allows providing a remote service using a method similar to classical procedure call [Nathoo 2005].

3. **Privilege escalation.** The attacker attempts to exploit a *sadmind* vulnerability in the found hosts. The exploitation is based on the insertion of a long string within a specific *sadmind* request, which could overflow a buffer associated to one of the function in *sadmind*, 'amsl_verify' [Internet Security Systems 2001]. The process is successful in three hosts (called Mill, Locke and Pascal), where the attacker creates an account named 'hacket2' with administration privileges [Lee 2008].

4. **Trojan installation.** The DDoS-Mstream trojan is installed on the three captured hosts using telnet and rsh. One of them (Mill) becomes the master who will control the other two when launching the DDoS attack.

5. **DDoS attack.** The DDoS attack against an external host (131.84.1.31) is launched from the three infected hosts using a spoofed IP address (78.111.82.41).

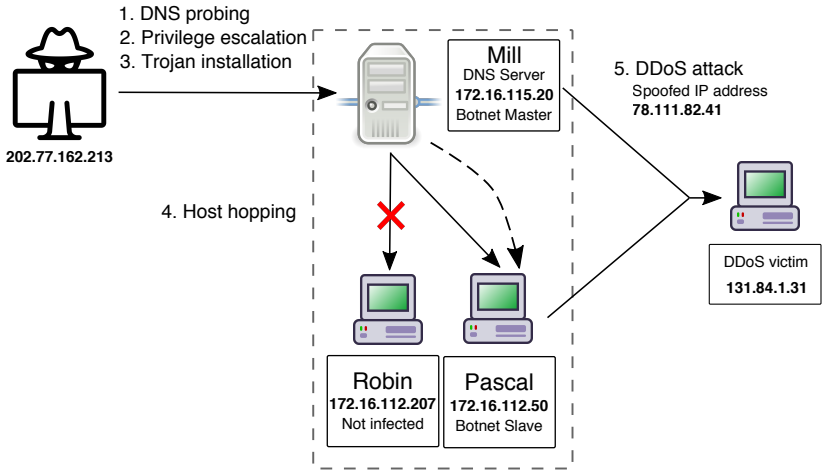


Figure 3.5: LLDoS 2.0.2 attack in DARPA 2000.

LLDoS 2.0.2, represented in Figure 3.5, has the same objective but it is stealthier. This time the attacker first breaks in a host and then uses it as a stepping stone (see page 21) to compromise another host. It is also composed of five steps [Farhadi 2011, Ramaki 2016, Shittu 2016]:

1. **DNS probing.** Instead of doing IP Address Sweeping, the attacker directly probes the public DNS server in the network, hosted in the Mill host, using the DNS HINFO query [Mockapetris 1983].

2. **Privilege escalation.** The *sadmind* vulnerability is exploited in Mill. The exploitation process is the same as the one followed in LLDoS 1.0 (step 3).

3. **Trojan installation.** The DDoS-Mstream trojan is uploaded to Mill using FTP. Note the difference with LLDoS 1.0, where the trojan is directly installed using telnet and rsh (step 4).

4. **Host hopping.** The attacker runs the trojan in Mill server and tries to break in two other hosts, using Mill as stepping stone. Only one of those attempts succeeds, the one targeting the host Pascal.

5. **DDoS attack.** The same DDoS attack performed in LLDoS 1.0 (step 5) is launched with Mill as a master and Pascal as a slave.

We shall not confound the DDoS attack launched at the end with the whole process of intrusion seen in LLDoS 1.0 and LLDoS 2.0.2. These two attacks are multi-step, but the DDoS attack is not. Several actions are performed in the DDoS attack (the packets sent by each host) but they are repetitive. The DDoS attack could be performed by itself from a computer legitimately under the control of the attacker. However, some authors [Yan 2004, Chen 2006, Shittu 2016, Holgado 2017] take the liberty of giving the name 'DDoS attack' to the two versions of LLDoS, leading to confusion.

3.2.3 UNB ISCX island-hopping

This island-hopping attack, represented in Figure 3.6, is contained in the ISCX 2012 Intrusion Detection Evaluation Data Set (see section 7.1.2, page 172). It is composed of the following steps:

1. **Email with attached exploit.** A system upgrade email is sent under the identity of *admin* to all the users in the network. This email contains a PDF file with an exploit that uses vulnerability CVE-2008-2992 in Adobe Reader for opening a reverse TCP shell on port 555.

2. **Reverse TCP shell**. One of the users, *user5* in the host with IP address 192.168.1.105, opens the file and get her machine infected. A remote connection with the attacker is established.

3. **Nmap scan.** From *user5*[3], the attacker launches a network scan using the program Nmap on subnets 192.168.1.0/24 and 192.168.2.0/24.

[3]Even if it is an abuse of language, we allow ourselves this identification of hosts by a user name to ease the explanation

4. **Vulnerability exploitation.** A host *user12* running Windows XP SP1 with a vulnerable SMB authentication protocol on port 445 is identified. The vulnerability (CVE-2008-4037) is exploited and *user12* is also captured by the attacker.

5. **Second scan.** Another scan is launched from *user12* towards the subnet where the internal servers are located, with IP addresses in range 192.168.5.0/24.

6. **SQL Injection.** The attacker identifies a server running an internal web application using MS SQL and creates a remote desktop connection to *user12* for launching a SQL Injection attack from it. The details of this attack are well explained by the creators of the dataset [Shiravi 2012].

7. **Backdoor.** In order to keep the connection from *user12* to the server active and thus be able to perform further web-based attacks, the attacker establishes a backdoor connection between *user12* and her own computer.

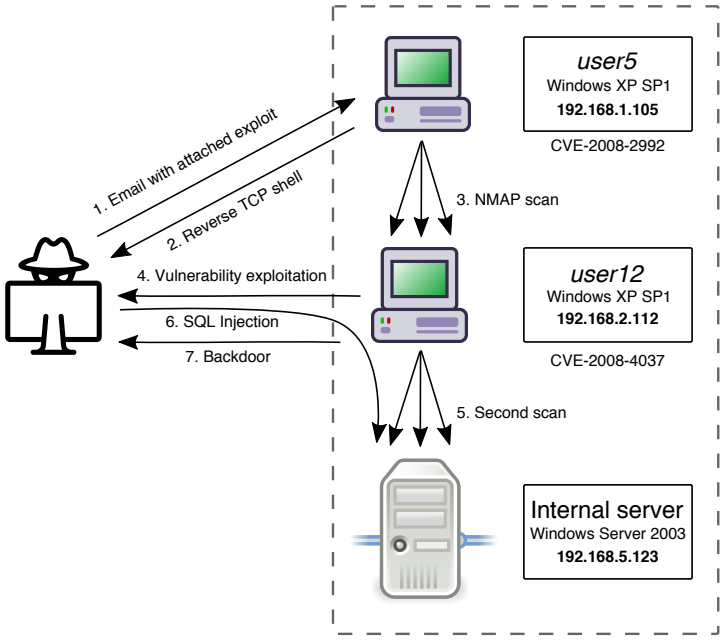


Figure 3.6: UNB ISCX island-hopping attack. Representation inspired by the one by Sadighian et al. [Zargar 2014].

3.2.4 HuMa

The multi-step attack contained in the HuMa dataset (see section 7.1.3, page 172), represented in Figure 3.7, is composed of the following steps:

1. **Port scan.** A discrete port scan against the web server of the organization, in order to collect information.

2. **Unsuccessful SSH brute-force attack.** A brute-force attack against the SSH service in the web server, identified thanks to the previous step. This attack consists in trying different passwords on the login system of SSH until one of them works.

3. **Successful SSH brute-force attack.** As the previous brute-force attack against the web server is not successful, a second attack of this kind is performed against the mail server of the organization from a local asset. The steps of scanning and capturing this local asset are not contained in the HuMa dataset. Probably they were done before the beginning of log collection. This time, the attacker successfully gets access to the SSH service.

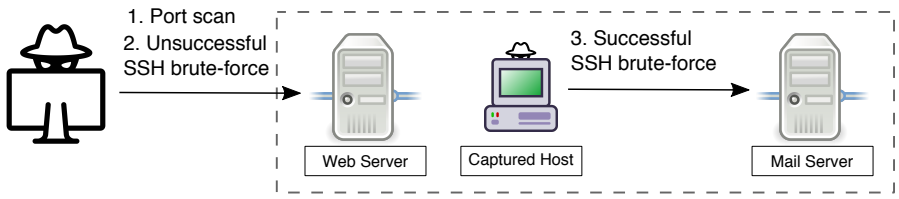


Figure 3.7: HuMa attack.

3.3 Modeling multi-step attacks

An important question when facing multi-step attacks is how we are able to describe them in order to a) share the information about them with other systems or partners and b) feed a detection system with an exploitable description. The answer given to this question strongly depends on the features that are considered relevant to describe the attack.

The purpose of this section is to review how multi-step attack modeling has been approached in the literature, taking the traces left by their execution as the building pieces of the model. We start defining the attack as a sequence of traces in section 3.3.1. The following section, 3.3.2, is dedicated to the languages to model multi-step attacks, used by some of the detection methods in the literature (see Chapter 4). In most of the publications about multi-step attack detection, a graph is used to model a multi-step attack from this sequence of traces. In section 3.3.3 we describe this type of model, that is called Concrete Attack Scenario Graph (CASG) throughout this thesis.

3.3.1 The attack as a sequence of traces

We can describe a multi-step attack by the set of traces left during its execution, as they were defined in section 2.3. This set can be ordered in time, as every trace has a certain timestamp associated to the moment when the represented action took place. To do this, we assume that all the clocks in the assets generating the traces are synchronized. It is a reasonable assumption if there is a NTP (Network Time Protocol) infrastructure in the network [Mills 1991]. Using this time synchronization protocol, we can share the clocks of every asset in the network to a very low precision (232 ps) [Mills 2016]. Given that the timestamps of the traces we study have a precision of 1 second, a network using NTP should not have any synchronization problem in their traces. However, apart from time synchronization there exist other problems that can lead to not reliable timestamps, such as implementation errors at software or hardware level [Micheel 2001]. These problems lie out of the scope of this thesis and will not be further considered.

The identification between step and trace is not always direct. A single action can be at the origin of multiple traces. For this reason, some authors aggregate the traces in a first stage and then work with abstract entities containing several traces. The goal is to make these abstract entities to better correspond to the attacker's actions. For example, a step corresponding to a brute-force attack, as it is the case of two of the steps in the HuMa attack (see section 3.2.4), generates one trace per failed login attempt. All these traces could be aggregated into one trace with the tag 'Brute-force attack', to make the ensemble to correspond to the step in the multi-step attack. In the case of having only alerts as traces, the resulting aggregated traces are called 'hyper-alerts' [Ning 2010, Anbarestani 2012]. Inspired by this notation, we give the name *'hyper-trace'* to the abstract entity containing an aggregated set of traces.

3.3.2 Languages to model multi-step attacks

In this section, we review some of the existent languages to describe multi-step attacks. We have found two families of languages: those based on prerequisites and consequences, presented in section 3.3.2.1, and another based on Petri nets, reviewed in section 3.3.2.2. Reviewed languages not fitting in any of these two families are presented in section 3.3.2.3.

3.3.2.1 Languages based on prerequisites and consequences

There is a family of languages for describing multi-step attacks that are conceived to be used on the detection methods based on prerequisites and consequences, that will be described in section 4.3.2.1. These languages assign a set of prerequisites or pre-conditions and consequences or post-conditions to each possible type of event. The cited detection methods use this knowledge to connect the events if they share pre and postconditions. Both the languages and the detection methods in this category limit themselves to the analysis of IDS alerts, as it is a trace with a limited and known list of types that can be classified and described in causal terms.

The first created language in this family was LAMBDA [Cuppens 2000], proposed by Cuppens and Ortalo in 2000. It was soon followed by JIGSAW [Templeton 2001] and CAML (Correlated Attack Modeling Language) [Cheung 2003]. All these languages are described below.

LAMBDA language [Cuppens 2000] is based on logical formulas that code the state associated to the pre and post-conditions of each event. It can be divided in three 'sublanguages': L_1, L_2 and L_3. L_1 uses the logic of predicates, with regular logical connectives ($\wedge, \vee, \neg$), to describe the pre and post-conditions. A meta-predicate 'knows' is also included to represent the gain of information of the attacker. For example, to describe the pre-condition "the server S has port 22 serving SSH and the attacker A knows that SSH is active in S", the predicate *port(ssh,22,tcp)* $\wedge$ *knows(A,active_service(S,sshd))* would be used. On its side, L_2 defines the events by the identification of its attributes and using the logical connectives $\neg$ and $\wedge$. To adapt it to the inherent attributes used in this thesis (listed in Appendix C), if we want to define an event e as "having 202.77.162.213 as source IP address and 'Admind' as type of alert"[4], we would express it as *ipsrc(e)=202.77.162.213* $\wedge$ *type(e)='Admind'*. Finally, L_3 is conceived to combine several events. The list of operators for doing so is given in the publication. An additional element of L_3 is the possibility of pointing out an event e as 'optional' in the attack sequence, using brackets ($[e]$). L_1, L_2 and L_3 are then combined to describe each individual attack, as it is shown in the example in Figure 3.8. Thanks to the pre and post-conditions, these attacks can be easily combined to represent a multi-step attack.

Capabilities and *concepts* are the two key elements in the definition of JIGSAW [Templeton 2001]. A capability is similar to a pre-condition in LAMBDA: it is the information required for an attack to occur. It is defined as a list of attribute/value pairs defining the condition. On its side, concepts are abstract situations that defines

[4]This represents a real alert in DARPA 2000 (see Table 5.2)

attack $NFS\_abuse(Target\text{-}IP)$
pre : $C_1 \wedge C_2 \wedge C_3 \wedge C_4 \wedge C_5 \wedge C_6 \wedge C_7 \wedge C_8 \wedge C_9 \wedge C_{10} \wedge C_{11} \wedge C_{12} \wedge P_6$
post : P_8
scenario : $((E_1 \,;\, (E_2 \,\&\, E_3)) \,\&\, E_4 \,\&\, E_5) \,;\, E_6$
where $action(E_1) =$ `rpcinfo -p` $Target\text{-}IP$
$\wedge\ action(E_2) =$ `showmount -e` $Target\text{-}IP$
$\wedge\ action(E_3) =$ `showmount -a` $Target\text{-}IP$
$\wedge\ action(E_4) =$ `finger @`$Target\text{-}IP$
$\wedge\ action(E_5) =$ `adduser --uid` $Userid\ U$
$\wedge\ action(E_6) =$ `mount -t nfs` P `\mnt`
$\wedge\ actor(E_1) = A\ \wedge\ actor(E_2) = A$
$\wedge\ actor(E_3) = A\ \wedge\ actor(E_4) = A$
$\wedge\ actor(E_5) = A\ \wedge\ actor(E_6) = A$
detection : $((F_1 \,;\, (F_2 \,\&\, F_3)) \,\&\, F_4) \,;\, noevent \,;\, F_5$
where $action(F_1) = detect(E_1)$
$\wedge\ action(F_2) = detect(E_2)$
$\wedge\ action(F_3) = detect(E_3)$
$\wedge\ action(F_4) = detect(E_4)$
$\wedge\ action(F_5) = detect(E_6)$
verification : $((G_1 \,;\, (G_2 \,\&\, G_3)) \,\&\, G_4) \,;\, noevent \,;\, G_5$
where $action(G_1) = test\_service(portmapper)$
$\wedge\ action(G_2) = test\_service(mountd)$
$\wedge\ action(G_3) = test\_service(mountd)$
$\wedge\ action(G_4) = test\_service(fingerd)$
$\wedge\ action(G_5) = test\_service(mountd)$

Figure 3.8: Example of LAMBDA signature of an *NFS_ abuse* attack. Image extracted from the first publication about LAMBDA [Cuppens 2000].

the connection between two steps. They have two blocks containing capabilities: 'requires', expressing the pre-conditions, and 'provides', containing the post-conditions. An additional block, 'action', indicates what to do if the concept is identified. The language used by Ning et al. in their detection methods (see page 79) is a variation of JIGSAW [Ning 2004a]. The idea of capabilities was later retaken by Zhou et al. in their model [Zhou 2007] and by the language ACML (Attack capability modelling language) [Pandey 2008].

In CAML (Correlated Attack Modeling Language) [Cheung 2003], every attack signature is called a *module*. Each module has three sections: 'activity', 'pre' and 'post'. The last two refer to a list of pre and post-conditions, and a list of predicate categories is given to define them. The activity section includes the events to be identified as part of the attack, by the definition of their attributes. CAML can be used by representing an individual IDS alert as a module or with each module containing the full set of events in the attack.

3.3.2.2 Languages based on Petri nets

A Petri net is a mathematical representation of the dynamic behavior of a system [Peterson 1981]. It consists on a series of nodes, called *places*, that are connected by *transitions*. An *input* and an *output* functions control the entry and the exit of the model, respectively. There exist a couple of languages to model multi-step attacks by means of a Petri net (EDL and EPNAM), and several detection methods using Petri nets as detection models [Yu 2004, Yu 2007, Vogel 2011a, Vogel 2011b, Chien 2012]. The most important of the language, in terms of impact in multi-step attack detection, is EDL, which is based on colored Petri nets and *taint analysis*.

The purpose of taint analysis is to track an information flow from the source to the sink of a graph. It is generally used for the analysis of flows of execution in computer programs [Schwartz 2010]. In the case of describing sequence of events, source nodes in the graph contain templates matching some of the events coming into the system. Some of these nodes are tainted. All the events matching with a tainted source are tainted too, while the others rest untainted. As tainted events are propagated in the graph, by the arrival of traces matching subsequent nodes, they create a tainted flow. A set of rules, called taint policies, determines how flows are tainted. For example, we can consider that the logs coming from an untrusted source are tainted from the beginning but not the ones coming from a trusted source. Taint policies also define how the taint is propagated through the graph and to other events.

The graph and the taint policies can be designed to represent multi-step attacks as tainted flows of events. EDL (Event Description Language) is based on this principle. It was developed by Meier [Meier 2007] and later improved by Jaeger, Ussath and Chen [Jaeger 2015], and it uses a colored Petri net as taint graph. In this Petri net, transitions lead from a state to another through the occurrence of an event.

Even if basic EDL is built on the intuitive idea of constructing a sequence of nodes for representing the concatenation of different events, its formal specification includes very innovative mechanisms which allow an accurate definition of the rules. For example, it incorporates the idea of a token for signaling which event the system should search. A token is placed in the first node to search and it is then moved to the second node once the first one is found. However, sometimes it may happen that we do not want to focus only on the first occurrence of the sequence, but we want to detect other events of the same nature as the first one but received after the first match. In this case, the token could be copied instead of being moved, so the system would be searching simultaneously (or by different processes, depending on the implementation) both the first event and the second one. Tokens in EDL are compliant

with the principles of taint analysis.

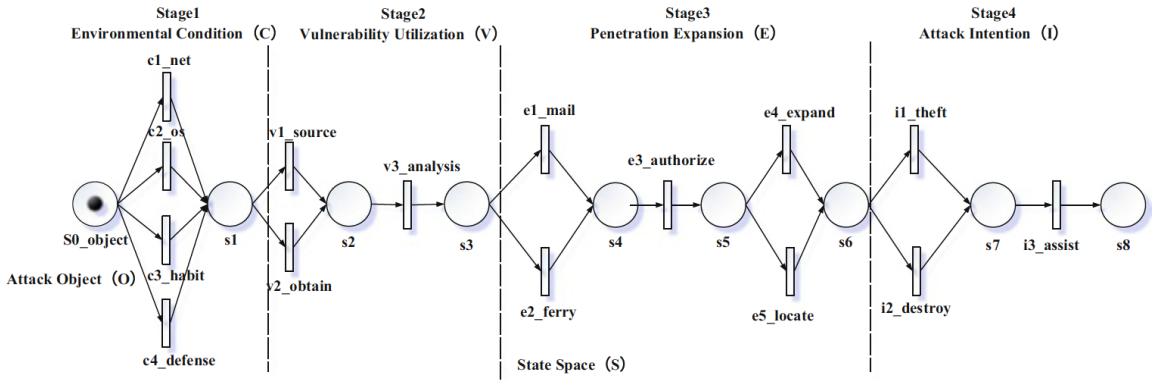


Figure 3.9: Example of a multi-step attack represented in a Petri net using the EPNAM language. Image from the publication where EPNAM is presented [Zhao 2014].

Meier implemented EDL in a real system in Java, called jSAM, which works only for specific types of plugin. For overcoming this limitation, Jaeger proposes an extension of EDL being able to deal with normalized events in *Object Log Format* (*OLF*) [Azodi 2013], so any type of event can be represented in this format. The correlation using EDL becomes then more versatile. This also allows a more general rule definition mechanism. For example, a general rule for a brute-force attack can be written if we properly define which type of event represents a 'failed login' in each of the authentication systems in the network. The same rule would be valid for any of the considered authentication systems. Moreover, Jaeger et al. [Jaeger 2015] also propose to use the STIX (Structured Threat Information eXpression) format to make CTI (*cyber threat intelligence*) directly loadable in the security system that uses EDL. The main limitation to this is that CTI information is usually missing some context, as companies share only a very limited amount of information. The same laboratory also proposes the automatic derivation of EDL signatures from taint graphs [Ussath 2016a].

Another proposition of language using Petri nets to represent multi-step attacks is EPNAM (Extended Petri Net-based Advanced Persistent Threat Analysis Model) [Zhao 2014]. However, it does not use tainting in flow propagation. The idea of EPNAM is an extension for multi-step attacks of the first attack model based on Petri nets that existed [McDermott 2001]. An example of a multi-step attack represented in a Petri net using EPNAM is shown in Figure 3.9.

3.3.2.3 Other languages

There are other languages for representing multi-step attacks that do not fit into any of the categories described above. This is the case of ADeLe (Attack Description

Language), that was developed in parallel to LAMBDA according to its authors Michel and Mé [Michel 2002]. Anyways, it is not a language based on prerequisites and consequences like LAMBDA, as it is oriented to the full description of the multi-step attack and not to the definition of individual attacks that can be combined. The description of each attack is divided in three parts: the *exploit*, the *detection* and the *response*. The exploit part represents the point of view of the attacker, giving details about the code used to perform the attack, its pre-conditions and what the attacker gets after the execution (the post-conditions). The detection part contains a list of the events involved in the attack and a description of how they appear when the attack takes place: in an ordered sequence, repeated, with alternative names, etc. This part is intended to be used by a detection system. Finally, the response part defines the actions that should be taken by the defensive system to face the described attack.

STATL (State Transition Analysis Technique Language), developed by Eckmann et al. [Eckmann 2002], is a transition-based language conceived for centralized case-based detection, also used later in distributed detection [Valeur 2004]. In STATL, an attack scenario is composed of a set of static elements and a dynamic behavior. The static elements are *states* or *transitions*, the two most fundamental concepts of this language. A state is a picture of the defended system at a certain moment. An attack represented in STATL is equivalent to a sequence of states, from a safe one to a compromised one. A transition represents the movement from one state to another by the means of an associated action. This action is only triggered by certain events, which make the next state possible. The dynamic behavior determines how a scenario evolves in a context of intrusion detection on incoming events.

While STATL is specifically thought for being used by a low-level search engine, Morin and Debar [Morin 2003] propose to apply a high-level description based on Dousson's chronicle formalism [Dousson 1994]. A chronicle is defined as a set of events that are connected according to certain time constraints. Time becomes in this formalism the main factor to determine the relationship between the events, which are defined by a series of rules. The chronicle formalism has been integrated in the M2D2 platform [Morin 2002]. Some other authors have adapted this language for other platforms [Wang 2004, Wang 2005a].

Apart from the languages already mentioned, there are some others, such as IDDL (Intrusion Detection Description Language) [Tabia 2011] or PDDL (Planning Domain Definition Language) [Hu 2013], that have been also used in the description of multi-step attacks. Almost every language conceived for event correlation [Sánchez 2003] could be use, in less or major extent, to represent multi-step attacks. The languages

reviewed here are the most relevant ones among the ones specifically created for multi-step attack representation, and the ones more connected to the contributions of this thesis.

3.3.3 Concrete Attack Scenario Graphs (CASG)

All the traces generated by a multi-step attack represent a sequence of steps in the pursuit of an objective. We could thus imagine a set of relationships between some of them and represent the set of traces as nodes in a graph, with the relationships defining the edges [Valeur 2004, Shin 2013, Holgado 2017]. A graph $G = (V, E)$ consists of two non-empty finite sets, $V(G)$ and $E(G)$ [Thulasiraman 2011]. The first one is the set of *vertices*, also called *nodes* [Wilson 2010]. In our case, each one of the nodes $v_1, v_2, \ldots, v_n$ represents a step of the attack in the form of traces $\theta_1, \theta_2, \ldots, \theta_n$. $E(G)$ is the set of *edges*, which are pairs of distinct elements of $V(G)$ that represent a relationship of certain kind between the two nodes (traces) composing them. Each edge contained in $E(G)$ is named by the juxtaposition of the symbols representing the nodes joined by it. For example, $v_a v_b$ is the edge joining nodes v_a and v_b.

Once a graph is built, we can consider traveling through it from node to node, using the edges as routes between two nodes. We define a *trail* as a finite alternating sequence of nodes and edges where all edges are distinct. If all nodes are also distinct, a trail is called a *path*. A graph is *connected* [Thulasiraman 2011] if there exists a path between every pair of nodes. A *circuit* is a trail with all nodes distinct except for the one starting and the one finishing the sequence, that are identical. The sequence $[v_0, v_0v_1, v_1, v_1v_2, \ldots, v_{k-1}, v_{k-1}v_k, v_k]$ is thus a circuit if $v_0 \neq v_1 \neq v_2 \neq \ldots \neq v_{k-1}$, $v_0v_1 \neq v_1v_2 \neq \ldots \neq v_{k-1}v_k$ and $v_0 = v_k$.

Another concept in graph theory that we are using in this thesis is that of a *directed graph*. An informal but intuitive definition of a directed graph is that it is a graph whose edges have an associated direction, namely $v_a v_b \neq v_b v_a$ [Bóna 2002, Guichard 2017]. Formally, a directed graph or *digraph* D consists of a non-empty finite set $V(D)$ of vertices and a finite family $A(D)$ of *ordered* pairs of elements of $V(D)$ called *arcs* [Wilson 2010]. Arcs can also be called *directed edges* [Bóna 2002, Bollobás 2013] to make the sense of direction explicit.

Each node v_a in a directed graph has an *in-degree*, the number of arcs of the form $v_x v_a$, and an *out-degree*, the number of arcs of the form $v_a v_x$. In other words, the in-degree is the number of arcs *going in* the node and the out-degree is the number of arcs *going out*. A node is called a *source* if it has in-degree 0 and a *sink* if it has out-degree 0. A directed graph with only one source is called *single-source*. A *tree* is

for example a type of single-source graph.

Most researchers use directed acyclic graphs (DAG) to represent multi-step attacks [Cuppens 2002c, Ning 2002c, Valeur 2004, Zhai 2006, Shin 2013, Holgado 2017]. A DAG is a directed graph without any directed circuit [Van Steen 2010]. In other words, if we choose a node in a DAG we cannot find a walk from node to node that brings us back to the initial node. To our knowledge, in every publication about multi-step attack detection, arcs in an attack model follow the temporal evolution of the steps, going from the step that happens before (trace with older timestamp) to the step that happens after (trace with newer timestamp).

Even if the arcs are not explicit, a multi-step attack represented as a sequence of traces [Chintabathina 2012, Brahmi 2013, Li 2016] can get its elements connected to form a graph. There are also authors using non-directed graphs [Wang 2006d, Pei 2016] or unordered sets [Qiao 2012, Zhang 2015]. These representations could be adapted to directed graphs, as we always count with a feature in the steps to give an order to the ensemble: the timestamp. A directed graph can be created just connecting the series of steps ordered in time. Even if there are two traces with exactly the same timestamp, we could assign an order based on any other of their attributes.

It is worth mentioning that there is an abuse of language in the multi-step attack detection literature when talking about directed graphs. Not many authors [Stotz 2007, Yu 2007, Luo 2014, Ramaki 2016] use the term 'arc' to refer to the connections between nodes in a directed graph. Authors do generally assign the term 'edges' [Cui 2002, Zhu 2006, Jemili 2008, Alserhani 2010, Saad 2012, Luo 2016, Shittu 2016] or 'links' [Mathew 2005, Brogi 2016] to the arcs, while they do not have the same mathematical meaning in Graph Theory [Wilson 2010, Bollobás 2013]. Some of them mention them as 'directed edges' [Kruegel 2002, Ning 2003b, Li 2007d, Khakpour 2009] but they abbreviate as 'edges' in most of the occasions. In this thesis, this abuse of language is not followed, and we call 'arc' to the objects joining two nodes in a directed graph.

Among the names assigned in the literature to a DAG representing a multi-step attack, we prefer the term 'attack scenario graph' [Ebrahimi 2011, Saad 2012, Zali 2012, Kavousi 2014], because it allows differentiating the trace-based models[5] from *attack graphs*. An attack graph is an abstract representation of the network with each node representing an asset with a set of associated vulnerabilities. The term 'attack graph' is well established and broadly used to designate this specific meaning. However, some

[5]The fact that the cited authors employ hyper-alerts instead of the original traces for the nodes of the graph is not important for the global idea of the model.

authors use it for referring to an attack scenario graph [Amos-Binks 2017], which can be confusing.

From the term 'attack scenario graph', we have coined a new one, *'Concrete Attack Scenario Graph' (CASG)*, to make a clear distinction between an attack scenario graph and the graph proposed in Chapter 5 to represent alternative multi-step attack cases. This term will be used throughout this thesis as an equivalent of an attack scenario graph.

3.4 Connection of nodes in the CAGS

We have seen that multi-step attacks can be modeled as Concrete Attack Scenario Graphs (CAGS), with the nodes representing the steps of the attack in the form of traces. The existence of an arc between two nodes depends on the relationship between both and is decided according to a set of *conditions*. The definition of the conditions for creating an arc has to be based on a series of *comparison features* associated to each pair of nodes.

To illustrate this, we present an example from outside the domain of multi-step attacks. Imagine for a moment that we want to model the network of citizens of Strasbourg that have met at least once. We would choose each citizen as a node and we would create an edge between any two citizens that have met. In this case, we have only one condition to create an edge, "the citizens have met at least once", based on the feature "number of times they have met". We could imagine many different conditions for joining two nodes (people), such as "both like jazz concerts" or "one is less than 2 years older than the other".

We distinguish between three kinds of feature associated to pairs of traces: similarity, time-based and context-based. *Similarity features* are only based on the own attributes of each trace, excluding time. For example, a similarity feature of a couple of traces could be having or not in common the author of the actions that generated the traces. *Time-based features* work with the time when the action represented in the trace took place. An example of time-based feature would be the difference in time between the actions. Finally, *context-based features* rely on the place of the trace in the context of all the registered traces in the network, on extrinsic information. For instance, a context-based feature of two traces can be the number of occurrences of both traces in the same day during the past year.

In our previous example using the citizens of Strasbourg, "both like jazz concerts" would be a condition based on a similarity feature; "one is less than 2 years older than

the other", one based on a time-based feature, and "they have met at least once", one based on a context-based feature. Remember that features in this context are always associated to a group of two entities, with the goal of performing a comparison between both.

The features chosen for our model will determine the information coded in the arcs of the CASG. An ample set of comparison features can be found in the literature. We review them in the pages that follow: similarity features in section 3.4.1, time-based features in section 3.4.2 and context-based features in section 3.4.3. Finally, in section 3.4.4 we will learn how they can be combined to define the arcs of our multi-step attack model.

3.4.1 Similarity features

According to the Oxford English Dictionary, 'similarity' is the "state or fact of being similar" [OED 2018e], where 'similar' refers to "having a resemblance in appearance, character, or quantity, without being identical" [OED 2018d]. Given that, it is clear that the similarity between two elements is related to the characteristics they share and their differences. The maximum similarity is reached when the two objects are identical [Lin 1998]. The degree of similarity depends on the subjective point of view of the observer [Wang 1997] and on the characteristics chosen to be compared. Even though, similarity can be numerically represented if we define the frame in which the comparison is made. Given two elements, an assumption considered is that the similarity between them is not influenced by a third element [Deepak 2015]. A pairwise *similarity function* can then be defined to compute the similarity between them. A similarity function is the expression in numbers of a similarity feature of the pair of elements. To unify the output of all similarity functions, independently of the nature of the feature, we work only with functions whose output is between 0 and 1, thus normalized.

According to Chen et al. [Chen 2007], given a set X of elements to compare, a function $f : X \times X \mapsto \mathbb{R}$ is a *normalized similarity function* if, $\forall x, y, z \in X$, it satisfies the following properties[6]:

1. $f(x, y) = f(y, x)$
2. $0 \leq f(x, y) \leq 1$

[6]The properties given by Chen et al. for non-normalized functions are adapted here to incorporate their idea of normalized function presented later in their paper.

3. $f(x,x) \geq f(x,y)$
4. $f(x,y) + f(y,z) \leq f(y,y) + f(x,z)$
5. $f(x,x) = f(y,y) = f(x,y) = 1$ if and only if $x = y$

Adapted to multi-step attack modeling, we define a similarity function [Li 2016] $S : \Theta \times \Theta \mapsto \mathbb{R}$ as a relation associating a value to two traces according to their degree of similarity with respect to the set of their attributes[7].

If a similarity function covers just one aspect of the compared elements, it can be called an *atomic similarity function*. The similarity between two objects is assumed to be the combination of all possible points of view of the object, under different perspectives. Given so, the global similarity between two elements can be calculated as the combination of the results obtained from atomic similarity functions [Lin 1998]. In the case of CASG, atomic similarity functions directly work with one or more *inherent attributes* of the compared traces (see section 2.3, page 17).

For example, we can think of an atomic similarity function returning 1 if both traces have the same source IP address and 0 otherwise. IP addresses are inherent attributes commonly associated to every trace, regardless the type of trace taken as input. They generally point out to the source of the action (*ipsrc*) and the destination or receptor of the action (*ipdst*). IP addresses are one of the most present attributes in the literature about multi-step attacks, because studied attacks usually go against victims that use the Internet Protocol and actions are generally executed at the level 3 of the OSI model.

Other attributes[8] for building atomic similarity functions are the ports used in OSI level 4 (*psrc* or *pdst*, for source and destination, respectively) or the Internet domain names (*dom*) [Zhang 2015]. In an environment where attacks generally happen at the link layer (OSI level 2), such as wireless networks, we can even find MAC addresses taking the role of IP addresses (*macsrc* or *macdst*) [Chen 2014a]. If steps are considered as actions of transformation on input objects (*inobj*) to create output objects (*outobj*), they can also be relevant attributes in the definition of the arcs [Brogi 2016]. The type of the trace (*type*), assigned conforming to a certain classification, is also used as an attribute to compute similarity, especially when the traces are IDS alerts and are associated to a specific type of attack [Qiao 2012, Saad 2014]. Finally, the most complete list of attributes used to define the link between steps in a scenario, with 19 inherent attributes, excluding timestamp, is given by Pei et al. [Pei 2016] and showed

[7] Remember that Θ is the set of all possible traces (see section 2.3, page 17)
[8] See Appendix C for the complete list of inherent attributes used in this thesis

in Table 3.1. They assign a name to the attribute (first column), which is associated to a certain type of event (second column). Most of them are specific of a certain service such as DNS (e.g. the queried domain name), HTTP (e.g. the response code) or the Windows Filtering Platform or WFP (e.g. the identifier of the process).

Field	Type of event	Description
q_domain	DNS	DNS queried domain name
r_ip	DNS	DNS resolved IP address
pid	WFP connect, process create and object access	Base-16 process ID
ppid	Auditd	Base-16 parent process ID
pname	WFP connect, process create, object access and authentication	Process name
h_ip	WFP connect	Host IP address
h_port	WFP connect	Host port number
d_ip	WFP connect	Destination IP address
d_port	WFP connect	Destination port number
type	HTTP	Request or response
get_q	HTTP	Absolute path of GET
post_q	HTTP	Absolute path of POST
res_code	HTTP	Response code
h_domain	HTTP	Host domain name
referer	HTTP	Referrer of requested URI
res_loc	HTTP	Location to redirect
acct	Object access	Principle of this access
objname	Object access	Object name
info	Authentication	Authentication information

Table 3.1: Inherent attributes considered by Pei et al. [Pei 2016] when modeling multi-step attacks.

An atomic similarity function does not need to forcibly work with the same attributes for both traces. For example, we can compare the *ipsrc* of one trace with the *ipdst* of another trace. This is useful when modeling attacks where there is a stepping stone, as there is a certain point when the destination of one step becomes the source of the following actions. For instance, in LLDoS 2.0.2 (see section 3.2.2), the IP address of the Mill host is first the destination of the privilege escalation and the trojan installation (steps 2 and 3) and becomes the source of the attack against Pascal (step 4). Comparison between different attributes in two traces is called *crossed comparison*. Oppositely, it is called *direct comparison* if the attributes are the same.

There are six kinds of atomic similarity function, according to the type of comparison done between the attributes. The arguments of each function are the attributes used, extracted from the compared traces θ_a and θ_b. Functions working with individual attributes use n_a and m_b to refer to the attributes in each trace, respectively, being n and m their names (see page 18). For example, if we compare the source

IP addresses of the two traces, we would make $n = m = ipsrc$, so $n_a = ipsrc_a$ and $m_b = ipsrc_b$. Conversely, functions working with a set of attributes from each trace refer to each set with capital letters. In this case, if we want to compare both source and destination IP addresses of the two traces we would have $N = M = \{ipsrc, ipdst\}$, so $N_a = \{ipsrc_a, ipdst_a\}$ and $M_b = \{ipsrc_b, ipdst_b\}$.

According to this nomenclature, atomic similarity functions used in the literature are described below:

- **Equality**. It returns 0 if the values of the attributes are not the same, and 1 if they are the same. It is expressed as follows:

$$f_1(n_a, m_b) = \begin{cases} 1, & \text{if } n_a = m_b \\ 0, & \text{otherwise} \end{cases} \tag{3.1}$$

  It is the simplest similarity function and the most used to describe the arcs between actions in a multi-step attack. Some authors use it to compare the source and destination IP addresses [Chen 2006, Saad 2012, Saad 2014] or ports [Chen 2006, Kavousi 2014, Wang 2016]. The destination of the action is often considered as the main binding element between the steps, and thus some authors focus on it to apply the equality function, either at port level [Zhu 2006, Shittu 2016] or at IP address level, taking the full address [Du 2009, Haas 2018] or just a subnet [Xian 2016, Zhang 2016]. Crossed comparison ($n = ipsrc, m = ipdst$) is also used [Kavousi 2014, Wang 2016]. In some work dealing only with alerts as traces, an equality function is defined by using the attack types assigned by the IDS [Qiao 2012, Saad 2012, Saad 2014]. Finally, the amplest set of equality similarity features is given by Pei et al. [Pei 2016] and shown in Table 3.2, using the attributes already presented in Table 3.1. They give a name to each one of the proposed features, from d_2 to d_{29}. d_1 corresponds to a time comparison so it is under the scope of section 3.4.2. Eight of them do direct comparison, such as d_4, which compares the destination port number (d_port) of two traces; and the rest do crossed comparison, as for example d_{16}, which compares the DNS queried domain name (q_domain) of the first trace with the HTTP host domain name (h_domain) of the second trace. Its opposite is d_{17}, which compares the other way around: the h_domain of the first trace against the q_domain of the second one.
- **Common element**. Given a list of attributes for each trace, this similarity function is 1 only if there is at least one attribute value in common between the sets of selected attributes of each trace, and 0 otherwise:

Name	Attr. 1 (n)	Attr. 2 (m)
d_2	*pid*	*pid*
d_3	*d_ip*	*d_ip*
d_4	*d_port*	*d_port*
d_5	*referer*	*referer*
d_6	*h_domain*	*h_domain*
d_7	*referer*	*h_domain*
d_8	*h_domain*	*referer*
d_9	*ppid*	*ppid*
d_{10}	*ppid*	*pid*
d_{11}	*pid*	*ppid*
d_{12}	*objname*	*objname*
d_{13}	*pname*	*pname*
d_{14}	*r_ip*	*d_ip*
d_{15}	*d_ip*	*r_ip*

Name	Attr. 1 (n)	Attr. 2 (m)
d_{16}	*q_domain*	*h_domain*
d_{17}	*h_domain*	*q_domain*
d_{18}	*q_domain*	*referer*
d_{19}	*referer*	*q_domain*
d_{20}	*q_domain*	*res_loc*
d_{21}	*res_loc*	*q_domain*
d_{22}	*get_q*	*pname*
d_{23}	*pname*	*get_q*
d_{24}	*get_q*	*objname*
d_{25}	*objname*	*get_q*
d_{26}	*pname*	*objname*
d_{27}	*objname*	*pname*
d_{28}	*r_ip*	*h_ip*
d_{29}	*h_ip*	*r_ip*

Table 3.2: Similarity equality features considered by Pei et al. [Pei 2016] (see attributes in Table 3.1)

$$f_2(N_a, M_b) = \begin{cases} 1, & \text{if } N_a \cap M_b \neq \varnothing \\ 0, & \text{otherwise} \end{cases} \tag{3.2}$$

This function is generally applied to the source and destination IP addresses ($N = M = \{ipsrc, ipdst\}$), so their place does not matter [Cipriano 2011, Xuewei 2014, Faraji Daneshgar 2016, Kawakani 2017]. The compared attribute lists do not necessarily need to be the same for the two traces, as shown by Wang et al. [Wang 2010], who use the sets $N = \{ipsrc, ipdst\}$ and $M = \{ipsrc\}$. Other attributes used are the MAC addresses [Chen 2014a], in a wireless environment, and input and output objects (*inobj* and *outobj*) such as computer processes or filenames [Brogi 2016].

- **Prefix similarity**. To compute this similarity, a transformation in binary form is made for the two compared values. Then, the number of common bits is counted, starting from the left and stopping when there is an uncommon bit. The result of the similarity function is a value between 0 and 1 coming from the division between this number, called l, and the total number of bits L in the attribute values, which are supposed to have the same length:

$$f_3(n_a, m_b) = \frac{l}{L} \tag{3.3}$$

In the literature, this function is exclusively used for the comparison of IP addresses [Zhu 2006, Qiao 2012, Li 2016, Shittu 2016, Wang 2016], where $L = 32$. It is a good measure of the proximity of hosts in the network because IP subnets are identified by the higher-order bits of the IP addresses.

- **Textual similarity**. A similarity function can be based on some standard

textual similarity metric, such as lexical matching [Metzler 2007] or the Jaccard similarity [Gomaa 2013]. We give the name $d(x, y)$ to the function returning the distance between the texts in an interval $[0, 1)$, with 0 representing equal texts. If x and y are strings of characters, we can define the textual similarity function as:

$$f_4(n_a, m_b) = 1 - d(n_a, m_b) \tag{3.4}$$

As far as we know, Zhang et al. [Zhang 2015] are the only ones to propose this type of similarity function to define the arcs in a multi-step attack model. They apply it to the source IP addresses, the attacked domain names and the alert types. The textual distance they use is the Levenshtein distance [Levenshtein 1966], which can be informally defined as "the minimal number of insertions, deletions and replacements to make two strings equal" [Navarro 2001]. For example, imagine that we have two different IP addresses: 80.101.114.80 and 80.101.110.101. To calculate the Levenshtein distance between the two in decimal representation we have to consider them as pieces of text and count the number of textual transformations we need to perform to pass from one to the other. In this case the distance is 3 because we need 3 transformations:

1. 80.101.11$\underline{4}$.80 → 80.101.11$\underline{0}$.80 (replacement)
2. 80.101.110.$\underline{8}$0 → 80.101.110.$\underline{1}$0 (replacement)
3. 80.101.110.10$_$ → 80.101.110.10$\underline{1}$ (insertion)

- **Hierarchy-based similarity**. This similarity function is based on hierarchy trees of concepts. They represent a taxonomy of the possible values of the attribute. Parent nodes include the concepts in children nodes. The similarity depends on the distance $D(n_a, m_b)$ in the tree between the values of the attributes. The output of the function $h(D(n_a, m_b))$ to calculate the similarity is inversely proportional to $D(n_a, m_b)$:

$$f_5(n_a, m_b) = h(D(n_a, m_b)) \tag{3.5}$$

This similarity feature was first proposed by Julisch [Julisch 2003a] to compare IP addresses and ports and then reviewed by other authors [Wang 2006a, Ren 2010]. The hierarchy trees used are shown in Figure 3.10. In these cases, the metric is defined as a dissimilarity, the opposite concept of similarity. Other authors [Xiao 2008, Li 2016] apply a hierarchy-based formula but without giving an ex-

plicit definition of the properties in the tree and the measurement of distances.

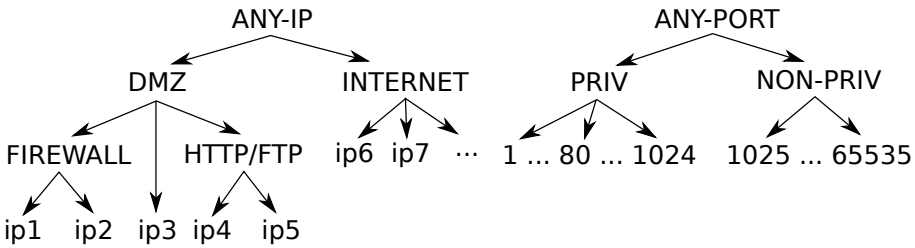


Figure 3.10: Hierarchy trees for IP addresses and port numbers [Julisch 2003a]

- **Ad-hoc formula**. Finally, we can find similarity functions created *ad-hoc* for a specific attribute. For example, Kavousi et al. [Kavousi 2012, Kavousi 2014] define a formula to compare IP addresses, where the resulting value depends on the lowest Internet Protocol address class (A, B, C, D or E) [DARPA 1981b] that they have in common:

$$f_{Kav}(n_a, m_b) = \begin{cases} 1, & \text{if } n_a = m_b \\ 0.8, & \text{if class C match} \\ 0.4, & \text{if class B match} \\ 0.2, & \text{if class A match} \\ 0, & \text{otherwise} \end{cases} \tag{3.6}$$

The presented similarity features are summarized in Table 3.3. In the first column we can see the general atomic similarity function associated to each feature next to the name of the feature. The second column shows the particular inherent attributes chosen for their implementation and in the third column we can find the references where those functions were presented and used for multi-step attack detection. Note that most of the similarity features are based on the source or destination IP addresses.

3.4.2 Time-based features

Every trace can store temporal information about relevant moments related to it: the moment when the represented action took place, when the trace was generated, when the trace was collected, etc. This information is represented as *timestamps*, strings recording a reference to an instant in time. For example, the log represented in Figure 3.1 (page 26) contained a timestamp signaling when the action took place: "06/Aug/2018:08:16:00 +0900".

Feature	Implementation	References
Equality $f_1(n_a, m_b) =$ $= \begin{cases} 1, & \text{if } n_a = m_b \\ 0, & \text{otherwise} \end{cases}$	$n = m = ipsrc$	[Chen 2006, Saad 2012, Haas 2018]
	$n = m = ipdst$	[Chen 2006, Du 2009, Saad 2012, Haas 2018]
	$n = m =$ $ipdst$, masked	[Saad 2012, Xian 2016, Zhang 2016]
	$n = ipsrc$ $m = ipdst$	[Kavousi 2014, Wang 2016]
	$n = ipdst$ $m = ipsrc$	[Kavousi 2014, Wang 2016]
	$n = m = psrc$	[Chen 2006, Kavousi 2014, Wang 2016, Haas 2018]
	$n = m = pdst$	[Chen 2006, Kavousi 2014, Shittu 2016] [Wang 2016, Haas 2018]
	See Table 3.2	[Pei 2016]
	$n = m = type$	[Saad 2012, Qiao 2012]
Common element $f_2(N_a, M_b) =$ $= \begin{cases} 1, & \text{if } N_a \cap M_b \neq \varnothing \\ 0, & \text{otherwise} \end{cases}$	$N = M =$ $\{ipsrc, ipdst\}$	[Cipriano 2011, Xuewei 2014, Kawakani 2016] [Faraji Daneshgar 2016, Kawakani 2017]
	$N =$ Set of $inobj$ $M =$ Set of $outobj$	[Brogi 2016]
	$N = M =$ $\{macsrc, macdst\}$	[Chen 2014a]
	$N = \{ipsrc, ipdst\}$ $M = \{ipsrc\}$	[Wang 2010]
Prefix similarity $f_3(n_a, m_b) = \frac{l}{L}$	$n = m = ipsrc$	[Qiao 2012, Li 2016, Shittu 2016, Wang 2016]
	$n = m = ipdst$	[Qiao 2012, Li 2016, Shittu 2016, Wang 2016]
	$n = ipsrc$ $m = ipdst$	[Qiao 2012, Li 2016, Shittu 2016]
	$n = ipdst$ $m = ipsrc$	[Qiao 2012, Li 2016, Shittu 2016]
Textual similarity $f_4(n_a, m_b) = 1 - d(n_a, m_b)$	$n = m = ipsrc$	[Zhang 2015]
	$n = m = dom$	[Zhang 2015]
	$n = m = type$	[Zhang 2015]
Hierarchy-based similarity $f_5(n_a, m_b) = h(D(n_a, m_b))$	$n = m = ipsrc$	[Julisch 2003a]
	$n = m = ipdst$	[Julisch 2003a, Wang 2006a]
	$n = m = pdst$	[Julisch 2003a, Wang 2006a, Xiao 2008, Li 2016]
	$n = m = pdst$	[Julisch 2003a, Xiao 2008, Li 2016]
Ad-hoc formula	$n = m = ipsrc$	[Kavousi 2014]
	$n = m = ipdst$	[Kavousi 2014]

Table 3.3: Atomic similarity features and their uses in the literature.

Timestamps are also inherent attributes of the traces, as they are independent of the information contained in other traces. However, they have certain particularities as a) its presence in every trace, no matter of which type, as it does not make sense to register of an action without an associated moment in time, and b) the special characteristics of the entity that it represents, time. Timestamps bind the trace to a certain moment in time. If two traces have everything the same except a timestamp, we can consider that they represent the same action repeated at different moments. Moreover, time serves to determine an order of actions, which is important for the causality: supposing that the timestamps are well synchronized, an action with a

greater timestamp cannot be the cause of an action with a lower timestamp. In other words, an action taking place before another one cannot be the consequence of this last one. Finally, as time can be represented as a scalar, we can easily define a metric. The distance in time between two traces can be a good indication of scenario membership [Dain 2001b].

The same as we defined similarity functions for other inherent attributes, we can define a set of functions linked to different time-based features. To simplify the presentation, the absolute value of the difference in time between θ_a and θ_b is called $\Delta t_{a,b}$, namely $\Delta t_{a,b} = |\theta_b(time) - \theta_a(time)| = |time_b - time_a|$. The functions associated to each time-based feature are presented below:

- **Threshold limit**. Used by many authors to defined the limits of a multi-step attack, it establishes that two traces are related if the difference of time between their occurrence is less than a certain threshold η [Wang 2010, Wang 2016, Pei 2016, Zhang 2016, Kawakani 2017]:

$$f_1^t(\Delta t_{a,b}, \eta) = \begin{cases} 1, & \text{if } \Delta t_{a,b} \leq \eta \\ 0, & \text{otherwise} \end{cases} \tag{3.7}$$

  In some cases, it is practical to check this limit using *sliding time windows* when an algorithm for multi-step attack detection is implemented [Soleimani 2008, Cipriano 2011, Chen 2014a, Xian 2016, Zhang 2016]. A time window is a subset of traces ordered in time where the difference in time between the first and the last traces of the subset is less than η. We can make this window 'slide' over the traces, always maintaining the same time difference between the limits of the window.

- **Gaussian function**. Li et al. [Li 2016] propose a Gaussian formula to express the time difference by a continuous value:

$$f_2^t(\Delta t_{a,b}, \eta) = \begin{cases} e^{-\pi \Delta t_{a,b}{}^2}, & \text{if } \Delta t_{a,b} \leq \eta \\ 0, & \text{otherwise} \end{cases} \tag{3.8}$$

- **Sigmoid function**. Dain and Cunningham propose a sigmoid-based formula [Dain 2001a], retaken by Shittu in her Ph.D. thesis [Shittu 2016]. They choose a sigmoid to make the resulting value quickly decay from 1 when $\Delta t_{a,b}$ increases. To ensure normalization between 0 and 1, we multiply the original formula by 2:

$$f_3^t(\Delta t_{a,b}, \eta) = \begin{cases} \frac{2}{1+e^{\Delta t_{a,b}}}, & \text{if } \Delta t_{a,b} \leq \eta \\ 0, & \text{otherwise} \end{cases} \tag{3.9}$$

- **Linear decay**. Kavousi et al. [Kavousi 2012, Kavousi 2014] propose a function with a linear decay that they use in combination with time windows. They define two time thresholds, TW_L and TW_T, called here η_1 and η_2, respectively, to be consistent with the notation used in this chapter. They do not specify in their publications why they choose a linear decay function. It is possible they were looking for a decay function with low computational complexity. The proposed function is:

$$f_{Kavousi}^t(\Delta t_{a,b}, \eta_1, \eta_2) = \begin{cases} 1, & \text{if } \Delta t_{a,b} < \eta_1 \\ 1 - \frac{\Delta t_{a,b}}{\eta_2 - \eta_1}, & \text{if } \eta_1 \leq \Delta t_{a,b} \leq \eta_2 \\ 0, & \text{if } \Delta t_{a,b} > \eta_2 \end{cases} \tag{3.10}$$

  This equation does not follow the rules for normalized similarity functions, as it is < 0 for some values, regardless the value of η_1 and η_2. To solve this, we define a new function which, furthermore, is continuous. The idea of linear decay is preserved:

$$f_4^t(\Delta t_{a,b}, \eta_1, \eta_2) = \begin{cases} 1, & \text{if } \Delta t_{a,b} < \eta_1 \\ \frac{\eta_2 - \Delta t_{a,b}}{\eta_2 - \eta_1}, & \text{if } \eta_1 \leq \Delta t_{a,b} \leq \eta_2 \\ 0, & \text{if } \Delta t_{a,b} > \eta_2 \end{cases} \tag{3.11}$$

- **Reciprocal function**. Qiao et al. [Qiao 2012] present a formula based on a reciprocal function:

$$f_5^t(\Delta t_{a,b}, \eta) = \begin{cases} 1, & \text{if } \Delta t_{a,b} < \eta \\ \frac{\eta}{\Delta t_{a,b}} & \text{otherwise} \end{cases} \tag{3.12}$$

3.4.3 Context-based features

Apart from being related by their similarity, traces can also be considered as part of the same scenario because of their place among other traces. A set of context-based features can be derived from their extrinsic attributes (see section 2.3, page 18). An extrinsic attribute represents a piece of information that is not contained in the trace, such as the probability of finding two traces of the same type together, which can be

computed from historic data, or the causal relationship between two types of trace, information that can be manually coded from experience and stored in a knowledge base.

Unlike similarity features, context-based features are not related to a function depending only on the content in the traces. But they can also be expressed by a value between 0 and 1, derived from the context of the trace. Once they are expressed in a normalized form, these features work in the same way as similarity ones, indicating how strong the relationship between two traces is in terms of extrinsic attributes. We distinguish three types of context-based feature used in the literature: probability of appearance, known causality and structural information.

- **Probability of appearance**. As two traces are seen more frequently together, it is more probable that there is a relationship between them [Mathew 2009, Ren 2010, Marchetti 2011b, Faraji Daneshgar 2016]. The *probability of appearance* feature is based on this observation. To determine it, researchers have to choose first a similarity feature to form pairs of traces to be studied. Then, the value assigned to the probability of appearance can be deduced using a frequentist approach: dividing the number of occurrences of each pair by the total number of pairs. A time threshold has to be established: only pairs of traces whose difference in time is lower than the defined threshold are chosen to compute the frequency and, from it, to deduce the probability.

  For example, a typical case is to associate the probability to the *type* of the trace, as in the work presented by Marchetti et al. [Marchetti 2011b]. In this case, two traces are considered as forming a pair if a) their difference in time is below certain threshold and b) the value of their inherent attribute *type* is identical. All the pairs fulfilling these two conditions are considered in the same group in order to compute the *probability of appearance*.
- **Known causality**, usually coded as a set of prerequisites and consequences related to each trace [Pandey 2008, Wang 2008, Ning 2010, Alserhani 2016]. In this case, the relationship of causality for each of the steps of the attack is known. Each step is supposed to prepare the conditions of the following steps so if the prerequisites of a trace match the consequences of a previous trace, they are linked and considered as part of a multi-step attack. The information about prerequisites and consequences have to be externally learnt and assigned to each type of trace. There exist languages to model this extra attributes, such as LAMBDA [Cuppens 2000] or JIGSAW [Templeton 2001] (see section 3.3.2.1, page 37).

There are other ways, more rigid, to consider known causality in the definition of the model. If we previously know the consecution of steps in the traces representing a multi-step attack, we can build a static and abstract model based on causality. It is the case of the Hidden Markov Models (HMMs) presented by some authors [Liu 2008, Mathew 2010, Chen 2016]. These models are based on the typical phases of a multi-step attack, such as *reconnaissance* (e.g. network scan), *attack* and *stepping stone*, as defined by Chen et al. [Chen 2016]. These authors consider that all the traces belonging to a multi-step attack can be classified in one of these phases, that are ordered in time following a rigid logical progression [Dain 2001b].

- **Structural information**. Another way of defining a multi-step attack is by connecting the vulnerabilities affecting the assets within the network [Wang 2006e, Roschke 2011, Zali 2012, Fayyad 2013, Luo 2016]. This feature is focused on the conditions that should be met to be able to exploit the vulnerabilities and on the consequences that exploiting them can have for the system. These conditions can be modeled in an *attack graph* or an *attack tree*, an abstract representation of the network containing the vulnerabilities of the network assets in each node [Sheyner 2002, Shostack 2014] (see page 43). In a CASG, arcs defined by structural information are created between two traces if the actions represented by them correspond to two adjacent nodes in the attack graph.

  A variation of the classic attack graph is the *cyber terrain* or *virtual terrain*, which incorporates additional data to each node, apart from the vulnerabilities, such as services and software version. Structural information is also used to build game models, where the links between the steps is based on the possible decisions of the attacker and the defender [Lin 2012, Luo 2014].

The richness of the information sources determines the quality of a model based on one of these features. In the case of the probability of appearance, the model is purely based on statistics, so it is more adjusted to reality as more representative data is available. The quality of the other two context-based features depends on the experience of the people building the database of context information. In the case of models based on known causality, the database contains the causality information associated to each trace. On its side, structural information is built from a good knowledge of the defended network and its vulnerabilities.

3.4.4 Combination of features

The characterization of the arcs in a CASG depends on the relationship between the involved traces. Each CASG defines certain conditions of existence of an arc. We say that there is a *correlation* between two traces or that they are *correlated* if the conditions to build an arc between them are met.

Trace correlation. In the field of Statistics, the term 'correlation' denotes the "linear relationship between pairs of variables for quantitative data", and it is quantified by a *correlation coefficient* [Witte 2010]. In Security, the broader sense of a mutual relationship between traces [OED 2018a] is taken, and the term is also applied to the process of finding a set of correlated traces to group them into an attack scenario [Kruegel 2004]. Therefore, we can say that "there is a correlation between these two traces" (equivalent to say "these two traces are correlated") but we can also refer to a method for building attack scenarios from traces as a "correlation method". The same as the correlation coefficient used in Statistics, we can define a correlation coefficient for each pair of traces in a dataset, giving it a value between 0 and 1 as we did with the features presented in the previous sections. We refer to this correlation coefficient as *correlation weight* to differentiate it from the statistical coefficient, which can take a value between -1 and 1. Depending on the author, it can also take other names, such as 'causal correlation' [AmirHaeri 2009] or 'correlation index' [Colajanni 2010].

The correlation weight. This element can be used to quantify the conditions for linking two traces [Soleimani 2012, Bateni 2013a], by the combination of several partial conditions. These conditions are determined by the three kinds of feature previously defined: similarity, time-based and context-based. The correlation weight can be based on just a single feature, such as equality of destination IP address ($f_1(ipdst_a, ipdst_b)$) [Du 2009] or an IP address in common ($f_2(N, M)$ with $N = M = \{ipsrc, ipdst\}$) [Xuewei 2014]. But the most used approach is the combination of several features [Cipriano 2011, Saad 2014, Pei 2016], in which the relationship between the traces in a CASG is considered to depend on several aspects of the traces [Mathew 2009].

A correlation weight can be easily built as a mathematical combination of similarity functions, because the properties of these functions are preserved under addition and multiplication [Chen 2007]. But if a time-based or a context-based feature is expressed as a function with the same properties (see page 45), it can also be combined with similarity features [Kavousi 2014, Wang 2016]. We give the generic name of *feature function* to a function that expresses numerically a feature associated to a pair of

traces and following the properties of similarity functions. Given that, we define a *correlation function* to compute the correlation weight as $C : F \times \Theta \times \Theta \mapsto \mathbb{R}$, with F a set of feature functions. All correlation and feature functions throughout this thesis have a range $[0, 1]$.

Once a correlation function is defined, a *threshold* has to be established to determine above which value an arc in the CASG exists between two traces. If the result of the correlation function is preserved and attached to the arc, the model is a *weighted* CASG. Otherwise, the CASG is *non-weighted*. Preserving the correlation value between the traces gives an idea of the strength of each arc under the imposed conditions.

We review below the most used methods for combining the results from several features to derive a correlation weight:

- **Maximum of a set**. One option to combine features is to take the maximum value among the results from the feature functions. This method is much used to reduce the set of features to be used for computing the correlation weight. As an example we can consider the atomic similarity function based on prefix similarity (f_3), conceived to work with IP addresses. Imagine that each trace has two attributes based on IP addresses, *ipsrc* and *ipdst*, as it is usually the case. Therefore, we can apply f_3 in four difference ways: $f_3(ipsrc_a, ipsrc_b)$, $f_3(ipdst_a, ipdst_b)$, $f_3(ipsrc_a, ipdst_b)$, $f_3(ipdst_a, ipsrc_b)$. As in a multi-step attack there can be stepping stones, the four functions are not equally appropriate to define all the arcs in the CASG (see page 47). A crossed comparison would signal the hop to the stepping stone, but not the relationship between actions performed in the same host. Choosing the maximum among the results returned by the four functions guarantees that we select the most relevant value for each pair of traces in the attack.

  Some authors follow this logic. Shittu, for instance, chooses the highest similarity between the results obtained from $f_3(ipsrc_a, ipsrc_b)$ and $f_3(ipsrc_a, ipdst_b)$, and does the same with the ones from $f_3(ipdst_a, ipdst_b)$ and $f_3(ipdst_a, ipsrc_b)$ [Shittu 2016]. However, Li et al. [Li 2016] preserve only the maximum between $f_3(ipsrc_a, ipsrc_b)$, $f_3(ipdst_a, ipdst_b)$, $f_3(ipsrc_a, ipdst_b)$ and $f_3(ipsrc_a, ipdst_b)$. For its part, Qiao et al. use a formula to combine them [Qiao 2012]:

$$f_{Qiao}(\theta_a, \theta_b) = \frac{max\left\{f_3(s_a, s_b) + f_3(d_a, d_b), f_3(s_a, d_b) + f_3(d_a, s_b)\right\}}{2} \tag{3.13}$$

  with $s = ipsrc$ and $d = ipdst$.

  The formula in general terms, given a set of feature functions F, is:

$$C_1(F, \theta_a, \theta_b) = maxF \tag{3.14}$$

- **Inclusive disjunction**. Given a list of features, resulting correlation weight based on inclusive disjunction can only take two values: 1 if at least one of the features in the set holds or 0 if none of them holds. This is the same as saying that for having a value of 1, at least one of the feature functions of the set F representing the selected features has to return a value greater than a certain threshold λ:

$$C_2(F, \theta_a, \theta_b) = \begin{cases} 1, & \text{if } \exists f(x,y) \in F \mid f(\theta_a, \theta_b) \geq \lambda \\ 0, & \text{otherwise} \end{cases} \tag{3.15}$$

  For instance, Chen et al. [Chen 2006] consider that two traces belong to the same scenario if they have either the same source IP address, or the same destination IP address or the same destination port. Cipriano et al. [Cipriano 2011] follow a similar logic but taking crossed equality of IP addresses instead of destination port equality. In the mentioned cases, the features are binary, so $\lambda = 1$.
- **Arithmetic mean**. The correlation weight is the average of several feature functions. Given that N is the number of feature functions $f(x, y)$ contained in F:

$$C_3(F, \theta_a, \theta_b) = \frac{1}{N} \sum_{f(x,y) \in F} f(\theta_a, \theta_b) \tag{3.16}$$

  This is the method used by Saad [Saad 2012, Saad 2014], who incorporates $f_1(ipsrc_a, ipsrc_b)$, $f_1(ipdst_a, ipdst_b)$ and $f_1(type_a, type_b)$.
- **Weighted sum**. A weight $0 \leq w_i \leq 1$ is assigned to each one of the considered features. These weights should not be confused with the correlation weight: the first ones determine the importance of each feature in the second one. Each w_i represents the importance of each feature for the final result. We choose the sum of all weights to be always equal to 1 [Li 2016, Haas 2018], to preserve the range of the correlation function. Some authors do so by normalizing the result afterwards [Kavousi 2014, Shittu 2016].

  The weighted sum is given by the following expression, which is equivalent to the arithmetic mean if all the weights are equal:

$$C_4(F, \theta_a, \theta_b) = \sum_{f_i(x,y) \in F} w_i f_i(\theta_a, \theta_b) \tag{3.17}$$

There are many examples in the literature of correlation functions using weighted sums. Wang and Chiou [Wang 2016] propose a set of eight features: two similarity features based on equality and two based on prefix similarity. They are applied on the source and destination IP addresses ($f_1(d_a, s_b)$, $f_1(s_a, d_b)$, $f_3(s_a, s_b)$ and $f_3(d_a, d_b)$, with $s = ipsrc$ and $d = ipdst$). They also use two more similarity features based on equality of port numbers ($f_1(psrc_a, psrc_b)$ and $f_1(pdst_a, pdst_b)$); a time-based feature based on threshold limit (f_1^t), and a context-based feature quantifying the probability of appearance of traces with the same *type*. On their side, Li et al. [Li 2016] propose three features: a hierarchy-based similarity feature on the destination port number ($f_5(pdst_a, pdst_b)$); a similarity feature on the source and destination IP addresses, which combines the maximum of a set of atomic prefix similarity functions (see page 58), and a time-based feature based on a Gaussian formula (f_3^t). Finally, Qiao et al. [Qiao 2012] combine a similarity function based on IP addresses (see equation 3.13), a reciprocal time-based function (f_5^t) and a similarity equality function working with the *type* of the traces ($f_1(type_a, type_b)$).

- **Sigmoid weighted function**. Pei et al. [Pei 2016] combine the similarity features shown in Table 3.1 using a sigmoid weighted function:

$$C_5(F, \theta_a, \theta_b) = S\left(\sum_{f_i(x,y) \in F} w_i f_i(\theta_a, \theta_b)\right) = \frac{1}{1 + e^{-\sum_{f_i(x,y) \in F} w_i f_i(\theta_a, \theta_b)}} \tag{3.18}$$

They choose to transform the weighted function in sigmoid because the learning algorithms they apply to determine the values of the weights do not guarantee a non-negative result. The sigmoid function maps the weighted sum to a bounded range between 0 and 1.

- **Machine learning techniques**. The correlation weight of a pair of traces can be also obtained by applying machine learning techniques on a set of features. Zhu and Ghorbani [Zhu 2006] proposed in 2006 to do so by a multi-layer perceptron, the most widely used type of neural network [Kruse 2013]. They use six features, the same as Wang and Chiou [Wang 2016] with the exception of $f_1(psrc_a, psrc_b)$ and f_1^t. The results of the feature functions become the input of

the neural network, which outputs a value between 0 and 1 corresponding to the correlation weight. The network is trained using a test dataset of alerts.

3.5 Summary

An overview of what multi-step attacks are and how they are considered in the literature has been presented in this chapter. We have started by comparing them with single-step attacks (section 3.1.1), assigning a classification according to different criteria (3.1.2) and explaining the particularities of a specific type of multi-step attack, the APT (3.1.3). We have then given several examples of multi-step attacks: WannaCry (3.2.1); LLDoS 1.0 and 2.0.2 (3.2.2); HuMa (3.2.4), and UNB ISCX island-hopping (3.2.3). We have continued by an explanation of how they can be represented as a sequence of traces (3.3.1) and, after reviewing the languages to model attacks in section 3.3.2, how such sequence can be modeled as a graph (3.3.3), called CASG. The arcs of this graph can be built according to several features, that have been defined next: similarity features (3.4.1), time-based features (3.4.2) and context-based features (3.4.3). We have finally explained how these features can be combined to build the arcs of the CASG (3.4.4). Now that multi-step attacks have been defined and we know how they can be modeled, we will see in the following chapter how the problem of detecting them has been addressed by the literature.

Chapter 4

Multi-step attack detection

Contents

4.1 Presenting multi-step attack detection 64
4.2 A methodology for bibliographic research 65
4.3 Description of detection methods 70
4.4 The field in perspective 90
4.5 The involvement of the security analyst 98
4.6 Summary 101

"No hay libro tan malo [...]*, que no tenga algo bueno"*
["There is no book so bad [...] that it does not have something good in it"]
— Miguel de Cervantes Saavedra, *Don Quixote*

Once we have analyzed how multi-step attacks are modeled, it is time to review the literature proposing methods to detect them. The goal is to collect, explain and put in context the detection methods that aim to reveal the whole structure of the attack from the study of real traces. As far as we know, it is the first time that the field is systematically studied taken this perspective. The survey presented in this chapter is then considered as a scientific contribution of this thesis

This survey was the object of a journal article published in July 2018 in the issue 76 of Computers & Security [Navarro 2018a], one of the most important journal in Cybersecurity with an Impact Factor of 2.65[1]. This chapter partially takes the bibliographic study presented in that article, whose corpus of publications has been updated to consider the period 2017-2018. Some papers have also been differently classified, thanks to the knowledge acquired since the final version of the article was submitted.

We open this chapter by presenting multi-step attack detection in section 4.1. We later explain the systematic methodology followed to compile and select the reviewed publications in section 4.2. The corpus of selected publications is then reviewed in

[1]From https://www.journals.elsevier.com/computers-and-security (Checked the 13 January 2019).

section 4.3. The conclusions and the statistics extracted from this corpus are the object of section 4.4. Finally, in section 4.5, we review some security systems where the analyst is directly involved in the detection process, as we did not find almost any method with this characteristic among the reviewed multi-step attack detection methods. All the models proposed as contributions of this thesis consider the human analyst as an essential element.

4.1 Presenting multi-step attack detection

The goal of studying and modeling multi-step attacks is to prevent their occurrence, detect their execution, response against them and recover from their consequences. We saw in section 1.1 that detection is a key piece in the Cybersecurity scenario. Prevention does not suffice in a scenario where newly discovered vulnerabilities are rising (see Figure 2.1) and hardening the systems against every known vulnerability can bring a loss in flexibility [Wang 2008].

Multi-step attacks pose a series of particular challenges for detection, such as the presence of innocuous steps or the existence of alternative actions to attain the same goal. But the sequence of traces left by the attackers often follows a logical progression [Dain 2001b], as it represents a set of actions with a single objective. If the security analyst is able to highlight the links between the involved traces, the strategy conceived by the attackers can be unveiled. Identifying the global structure of the attack in the set of traces reveals itself as important for detection.

This approach should not be confounded with the detection of the symptoms of the multi-step attack. For instance, a detection method that has been designed for detecting escalation of privileges in a Linux machine will certainly detect when an attacker tries to become superuser. But if this is just a single piece of a multi-step attack, we are far from identifying the whole threat [Ghafir 2018], which can have further consequences. Perhaps the infection of the Linux machine was a way to have a stepping stone or just a distraction movement.

Even if some seminal work about the importance of linking several traces to detect an attack already existed [Vigna 1998], Huang and Wicks were probably the first ones that pointed out the importance of attack strategies in detection in 1999 [Huang 1999]. The new millennium brought an awareness against this kind of attack and the analysis of the global attack strategy became important in security research.

In the publications addressing the problem of finding multi-step attacks in sets of traces there is not a clear distinction between the search in the past (investigation)

or in the present (detection). Detection methods are generally evaluated using past sets of traces, as it is difficult to generate real-time data containing the execution of a multi-step attack. They are then evaluated as we would do with an investigation method working with the past. Most of the proposed search methods could then be applied both for investigation and for detection.

As far as we know, the systematic survey we have presented in Computers & Security [Navarro 2018a] and continued here is the first published survey about multi-step attack detection methods that use traces as input. Apart from being a good point of reference for the work developed in this thesis, we consider this survey as a relevant work for the community. Because of that, we consider it as a contribution of this thesis in its own right.

4.2 A methodology for bibliographic research

Systematic search [Kitchenham 2004] is a rigorous method for literature review that makes the result easily reproducible but requires more effort than traditional reviews. Nevertheless, we have chosen this method to avoid missing relevant publications that are not much cited in the literature and thus difficult to find if the search is not exhaustive. The process we have applied to select a corpus of multi-step attack detection methods is inspired in the one used by Luh et al. [Luh 2016] for semantics-aware detection of targeted attacks. We present below the methodology used in the search, starting with the defined inclusion and exclusion criteria (section 4.2.1) and continuing with the description of each one of the three search phases (section 4.2.2).

4.2.1 Inclusion and exclusion criteria

In any bibliographic search it is important to establish a rigorous list of criteria defining which publications are included in the corpus and which are not. This list serves as a guide during the entire research process. The inclusion criteria pointing out the work to be included in our corpus is listed below:

- A multi-step attack detection method is described in the paper.
- The method works on real digital traces such as events, alerts or network packets. This does not mean that it needs to be evaluated with real traces. It can be tested by simulation or case studies but it should have been conceived for the analysis of traces.
- The method considers the structure of the multi-step attack and the potential

links between the steps.

- The structure of the document is that of a scientific research publication.
- The method is described in a clear and evident way.
- The publication is written in English and with an understandable style.

A special emphasis should be given to the second inclusion criterion, concerning the trace-based functioning. There exists a vast number of methods belonging to the domains of vulnerability analysis or risk assessment whose goal is to generate a list of possible paths of an attack in the network to deploy defenses, predict which assets could be affected and evaluate the risks of an attack. Possible paths are usually represented in an *attack graph*, also called *attack tree*, a concept already introduced in page 43. Methods only based on structural and static models, such as attack graphs, are excluded from this survey, except if those models are used as an instrument for the projection of real traces, such as the methods presented in section 4.3.3.

A set of exclusion criteria is also considered, referring to the categories of publications not included in the corpus:

- *Focused on multi-step attacks but not on detection.* There are many papers that address multi-step attacks but do not propose a detection method, focusing on other topics such as languages for attack modeling, risk assessment, vulnerability analysis or application of security policies. While we are not interested in the three last topics, we already reviewed the modeling languages in section 3.3.2.
- *Only focused on one aspect* of multi-step attack detection such as executable file analysis [Nath 2014] or C&C identification [Lamprakis 2017], among others. They do not consider the perspective of linking the different steps in the attack.
- *About flow-based detection*, where the structure of the multi-step attacks is not revealed. In these publications, detection is done analyzing the statistical effects on traffic produced by the attack (e.g. [Lu 2017]).
- *About state-based detection*, where the method analyzes the changes of state in each of the machines in the network and deduces from there the potential occurrence of a multi-step attack (e.g. [Vert 2018]).
- *Not about multi-step attacks.* For example, about coordinated or distributed attacks, such as DDoS attacks, where multiple attackers may have a common objective but they do not necessarily perform multiple distinct steps in the network.
- *Not in English.*
- *About single-step attack detection*, even if it is claimed that they are oriented towards multi-step attack detection.

- *Not research*, such as articles published in non-scientific magazines and commercial whitepapers.
- *Slides or posters.*
- *Duplicated.* Documents containing exactly the same content but that were published in different places or whose reference appears twice under different names.
- *Of low quality.* In the selection of the publications, we follow the position defended by Glass [Glass 2000], who considers that a systematic survey should include all the references found in the studied domain, both *good* and *bad* ones. Anyways, as there is not a clear benchmark to test multi-step attack detection methods and most of the publications do not offer the possibility of reproducing the experiments (see section 4.4.3), it would be difficult to apply a narrow exclusion criterion based on the quality of results. However, we discard some references which are below a minimum level of quality in style, form and presentation; where:
  - there is not enough detail to understand how the method works,
  - text cannot be well understood because a bad use of English or
  - some elements taken from the work made by other authors are not properly credited.

  We do not cite in this thesis any of the work excluded in terms of quality. Number of citations is such a relevant metric for the evaluation of quality in current scientific research that citing work that we do not consider acceptable would do research a disservice. Information about discarded publications is available under request.

4.2.2 Research process

In this section, we present the research process followed in the systematic review. We divide the process into three phases: A) bulk search of keywords in several research engines, B) selection of relevant results and C) recursive search for references. These three phases, developed when writing the article where our first multi-step attack detection survey was published [Navarro 2018a], have been followed again to update the corpus with the new material published on 2017 and 2018. After this process, the bibliographic corpus about multi-step attack detection has a size of **201 publications**.

4.2.2.1 Phase A: Using the research engines

We start the systematic bibliographic research by searching a selected set of keywords in IEEE, ACM, Web of Science and Google Scholar, the most important web-based search engines among the ones specialized in the scientific literature on Computer Science. The search is made only on the title of the publications. In the case of Google Scholar, we exclude patents and citations. Citations correspond most of the time to sources that are not openly available or to non-scientific resources.

The set of keywords used has been selected among the ones frequently appearing in the title of publications addressing multi-step attack detection. We use the symbol '+' to indicate the combination of different keywords in the same search, while the curly brackets ('{...}') enclose alternative choices. An 's' between parentheses denotes a word considered both in singular and plural. Words enclosed by double quotations are searched as an ensemble, literally as they are written. We list below the used sets of keywords. The number of full strings to search for each set is shown between square brackets:

- **Set of keywords 1:** advanced persistent threat(s) [2 strings]
- **Set of keywords 2:** APT + {analysis, architecture, defense, detection, framework, mechanism, mitigation, prediction, prevention, strategy, system} [11 strings]
- **Set of keywords 3:** {multi level, multi-layer, multi-stage, multi-step, multi-stage} + {intrusion(s), threat(s), attack(s)} [30 strings]
- **Set of keywords 4:** {"attack plan", "attack scenario", "attack strategy"} + {detection, prediction, recognition} [9 strings]

The resulting total number of strings to search is 52. A visual representation of these sets of keywords is shown in Figure 4.1, with the combination of their parts indicated by arrows.

Once the search is launched, we only download the publications related to cybersecurity. Some of the publications found belong to other domains, even if the keywords used are mainly related to cybersecurity. For instance, the acronym 'APT' can refer to concepts in Finances and Medicine.

4.2.2.2 Phase B: Filtering the results

The resulting publications from Phase A are reviewed one by one in Phase B. It is now when the inclusion and exclusion criteria listed in section 4.2.1 are applied for selecting only the references proposing multi-step attack detection methods.

The followed method for generating a "bibliography of candidate studies" is the

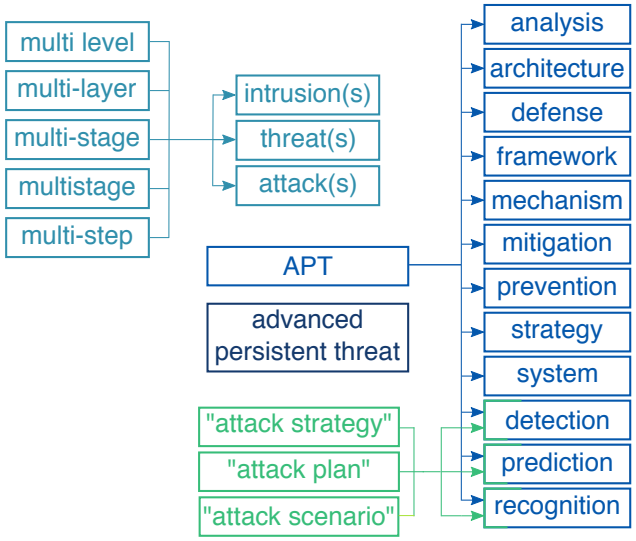


Figure 4.1: Diagram representing the keywords used in the Phase A of the systematic research. Different tones are assigned to each set of keywords used.

one proposed by Meline [Meline 2006]. This method is applied in two stages. First, the titles and the abstracts are reviewed and the publications that clearly meet one or more exclusion criteria are discarded. Then, the remaining documents are read and we include in the corpus those ones meeting all inclusion criteria and not exclusion criteria.

4.2.2.3 Phase C: Recursive search for references

Once the filtering in Phase B is done, we perform in Phase C a recursive search of references among the ones cited by the selected publications and among the ones citing them. First of all, we review the ones cited by the current corpus. This process is iteratively repeated on the newly discovered set of publications until no more new references matching the inclusion criteria are found.

This recursive search has the problem that it is oriented 'towards the past', as found publications are always older than the ones found before. That is why in a second stage we also look for references citing the ones included in the corpus so far. To do that we use Google Scholar, where the 'cited by' button associated to each reference returns a list of other references citing it. Again, the process is iteratively repeated on the new publications until no further reference is found.

4.3 Description of detection methods

The problem of multi-step attack detection has been addressed by many different ways during the 18 years elapsed since the beginning of the field. Proposed methods are varied and work on the basis of a model of the attack. For instance, if researchers consider that there is a common structure laying behind any multi-step attack, they will develop methods that try to found these common elements, perhaps using automatic tools based on machine learning. On the contrary, researchers that see a multi-step attack as an isolated instance could be more predisposed to build patterns adapted to each individual attack and arrange them in a knowledge base of signatures.

We have defined a classification for multi-step attack detection methods by the approach they follow. Some of the approaches are further split into categories. The taxonomy of the classification is shown in Figure 4.2. We find five kinds of approach:

- **Similarity-based**. The degree of similarity between traces determines the construction of the attack scenarios. We find three categories in this approach: *progressive construction*, *scenario clustering* and *anomaly detection*.
- **Causal correlation**. Detection is focused on the causal relationship between the steps of the attack. Methods under this approach can be further classify into one out of two categories: *prerequisites and consequences* or *statistical inference*.
- **Structural-based**. Incoming traces are projected into a model of the network, where future attack paths can be predicted.
- **Case-based**. Detection of well-known attack scenarios as sets of rules or attack signatures.
- **Mixed**. More than one of the approaches are followed but none of them stands out among the others.

In this section, we offer a brief presentation of the work under each one of the five approaches. In Appendix D, the reader can find all the 138 analyzed methods and the 201 publications presenting them, with a summary of their characteristics.

4.3.1 Similarity-based methods

Similarity-based methods propose the composition of scenarios according to the similarity between the individual steps of the attack, represented as traces. These methods are based on the idea that similar traces are related to the same root cause [Julisch 2001, Salah 2013] and they therefore belong to the same multi-step attack. Computation of the similarity degree is the central focus of these methods. This dis-

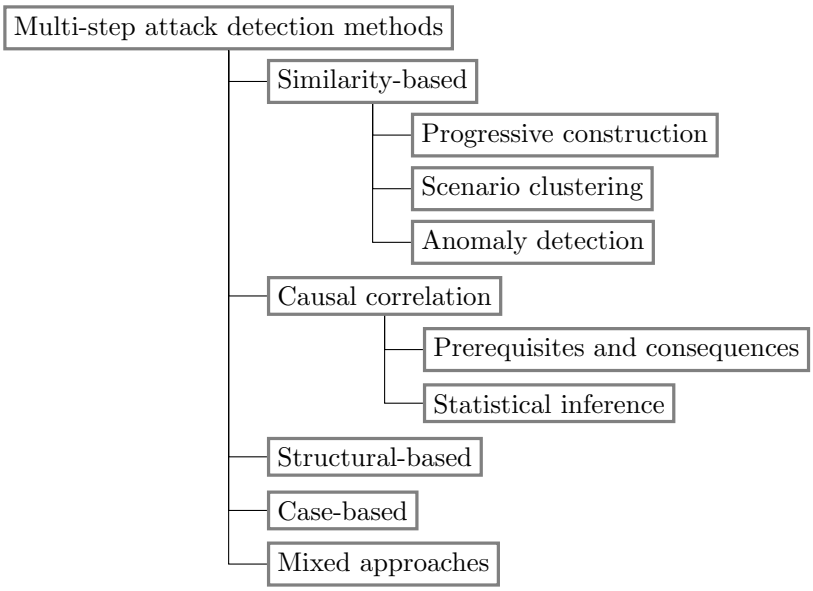


Figure 4.2: Taxonomy of multi-step attack detection classification.

tinguishes them from causal correlation methods, which do not consider how similar two steps are but how they are consequences of one prepare for the other or how often one follows the other.

The similarity degree between traces is based on one or several similarity features among the ones we presented in section 3.4.1 (page 45). The type and number of features and the attributes considered (IP addresses, port numbers, etc.) depend on the election of the authors. Some of them just consider the comparison of one feature [Shaneck 2006] while most of them look at the combination of several of them [Chen 2006, Zhu 2006]. When several ones are chosen, some of the features can be considered as more important than others through the application of a weighted correlation function (see page 59).

Time-based features (section 3.4.2) also come into play, as they do in almost every multi-step attack detection method, because the limited resources of the detection system do not allow the search of attack scenarios during an indefinitely long time span. A similarity-based method compares each trace with the traces that are in the temporal proximity of it. As the maximum temporal span chosen is larger, the number of comparisons is increased and the time spent during the detection process is longer.

The main advantage of similarity-based methods is that the implementation is easy and can return non-previously known multi-step attacks. The analysis is just based on comparison between pairs of traces, so systems implementing these methods have usually a good performance. However, choosing how the traces should be linked is far from being an easy task. If the linking process is kept simple, only relying

on the similarity of a few attributes, the results will contain too many false positive alerts. On the contrary, a complex linking process can be too specific to capture the characteristics of the whole range of multi-step attacks.

It is important to note that a method solely based on similarity-based connections between traces would never be able to detect all types of multi-step attacks. The logical statement "in every multi-step attack there are common inherent attributes between all its steps" can be falsified by a counterexample: the attacks in the dataset DARPA 2000 presented in section 3.2.2. In these attacks, the last step, represented by a 'Stream_DoS' alert if packets are processed by the RealSecure IDS, does not have any attribute in common with the rest of the alerts. This alert is not even similar to the other ones in the attack, at least no more similar than to many other alerts that are not part of the attack, due to the IP spoofing done by the attacker. More information about the alerts in DARPA 2000 can be found in section 7.1.1, page 170. Effective similarity-based methods have thus to incorporate other techniques to become universally applicable.

We can classify similarity-based methods in three categories according to how similarity features are used to bring the traces together and form attack scenarios. First of all, we have a set of methods based on *progressive construction*, where the sequence of actions conforming the attack is built step by step by the addition of similar traces. Secondly, methods doing *scenario clustering* apply a clustering method to all the set of traces and return the clusters as possible scenarios, without considering the order of the steps as it happens in progressive construction. Finally, in *anomaly detection* methods, the similarity of incoming sequences of traces is computed against a set of non-malicious traces and sequences are considered as part of an attack scenario if they differ from normality.

4.3.1.1 Progressive construction

In progressive construction methods, a potential multi-step attack sequence is built step by step. Traces are appended to a scenario according to the degree of similarity between them and the traces already contained in the scenario. The difference with clustering methods is that the order of the actions is an important factor. In progressive construction, the sequences are built step by step and following a logical progression.

The match between the compared attributes can be exact or partial. An exact match is only based on equality similarity features (using the atomic similarity function f_1, presented in page 48), while a partial match depends on a correlation weight (see section 3.4.4) combining several types of similarity features.

Valdes and Skinner [Valdes 2001] are one of the first authors in proposing a multi-step attack detection method based on progressive construction. Their method is integrated in the EMERALD system [Porras 1997] and is based on a similarity matrix that contains the manually crafted correlation weights between each type of trace. The do not specify which aspects of the traces are compared, but it is still one of the most cited references in multi-step attack detection. Inspired by this work, Dain and Cunningham [Dain 2001b, Dain 2001a] propose a system where each time an alert arrives at the system, the probability of being assigned to one or another scenario is computed based on a similarity comparison against the last trace in the scenario. The comparison depends on several factors, such as the IP addresses or the difference of time.

Apart from Dain and Cunningham, many other authors solely work with IDS alerts and consider the multi-step attacks as sequences of elementary attacks. For example, Ebrahimi et al. [Ebrahimi 2011] propose a matching algorithm using both source and destination IP addresses. On the other hand, Khakpour and S. Jalili [Khakpour 2009] consider more elaborated features, such as prefix similarity of destination IP addresses ($f_3(ipdst_a, ipdst_b)$) or equality of ports ($f_1(pdst_a, pdst_b)$). Convinced that the inherent attributes of IDS alerts are not enough to determine the correlation between them, they propose to also incorporate correlation weights between alert types coming from expert knowledge. They are arranged in a *correlation matrix*. The details of the deduction of the weights are not provided in their publication.

The idea of a manually crafted correlation matrix had been already proposed three years earlier as the main brick of the Statistical Filtering algorithm [Wang 2006d, Wang 2006c]. Although the construction of sequences is automatically made based on statistical measures, the final verdict about their nature as attacks relies on the human-crafted correlation weights. The first author of this proposal, Li Wang, continued the development of her idea several years later [Wang 2010], introducing more sophisticated time windows, whose size evolves in time, and a classification of candidate scenarios in three groups according to the relationship between their actions. In the work by Tian et al. [Tian 2017], the correlation matrix is built from the probability of having a type of alert following another. Weights in this matrix are applied in the linking process together with several similarity features A manually crafted correlation matrix is also the base of the RTEAS algorithm [AmirHaeri 2009].

On the contrary, in the correlation matrix proposed by Zhu and Ghorbani, called ACM (Alert Correlation Matrix) [Zhu 2006], the weights are automatically deduced from a reference set of alerts. This deduction, based on six selected similarity features,

is done by two machine learning methods: Multilayer Perceptron (MLP) and Support Vector Machine (SVM). They conclude that MLP is more precise but that SVM has a higher training speed. While they apply automatic learning once the similarity features are quantified, Chih-Hung Wang and Chiou [Wang 2016] do it inversely. They execute the Bayesian learning process developed by Kavousi and Akbari [Kavousi 2012, Kavousi 2014] (see section 4.3.2.2) and consider the result as an additional feature to be combined with seven other features (see Table 3.3 in page 52).

Another way to differentiate the membership of an alert to a specific scenario is by the assignation of tags containing a list of input and output objects. The search of additional alerts connected to a specific tag is made by linking inputs and outputs. This is the basis of TerminAPTor [Brogi 2016, Brogi 2018], which takes the sequence of alerts as a flow of information to propagate the tags.

Instead of only working with IDS alerts, some methods in this category use them just as the first indicator of the presence of an attack. Once the alert is detected, these methods launch a process to look for traces that have certain attributes equal to the ones in the triggering alert. Searched traces do not have to necessarily be alerts, but they can be events or network packets. Once found, they are appended to the original alert to form the attack scenario. These systems are mainly based on the match between IP addresses and they get the needed data from different locations in the network. While the Active Event Correlation (AEC) [Chen 2006] is placed in the same machine where the IDS is, the other ones are designed to be placed in other parts of the network. Both the BDB (Bi-directional, Distributed BackTracker) system [King 2005] and STARLITE (Stealthy Tracing Attackers Research Light TracE) [Strayer 2005] require agents installed in other assets, the first one in the kernel of the monitored machines and the second one in the routers. The work done for BDB has a continuity in the SLEUTH (Scenario LinkagE Using provenance Tracking of Host audit data) system [Hossain 2017], that has a higher performance and incorporates the collection of OS information. The framework developed by Shaneck et al. [Shaneck 2006] works with a centralized set of collected traces, in a similar way.

Another proposal based on progressive construction is HERCULE, a system for doing "attack story reconstruction" [Pei 2016]. This method is inspired by relationships in social networks. Their authors define a long list of possible relationships between events (see Table 3.2 in page 49). These relationships are exploited to create weighted attack scenario graphs. Weights are calculated using a quadratic optimization algorithm. A very interesting aspect of the paper presenting HERCULE is the emulation of an ample group of real APTs from the existing literature.

Conversely, Artificial Immune Systems (AIS) have been much used in single-step intrusion detection [Kim 2007], but the only proposal particularly addressing multi-step attack detection is iCorrelator [Bateni 2013b, Bateni 2013a, Bateni 2014]. This system emulates the human immune system in a three-layer architecture. It is based on *cells*, which are vectors of similarity features that represent the correlation degree between two traces. Correlation is based on the set of cells stored in memory, which evolves through supervised learning from an initial set of basic rules.

4.3.1.2 Scenario clustering

The goal of clustering is to discover *natural* groups in a set of elements [Jain 2010]. This is usually done through the application of automatic clustering algorithms. In this category, we find the methods applying a clustering algorithm to identify groups of correlated actions. These groups or *clusters* are then considered to be potential multi-step attacks. The degree of correlation between traces belonging to the same cluster should be higher than the degree between traces from different clusters.

As far as we know, the application of clustering to multi-step attack detection was first proposed by Julisch [Julisch 2001, Julisch 2002, Julisch 2003a, Julisch 2003b]. He was also the first one to propose hierarchy-based similarity features (see section 3.4.1, page 50) to be used in his approximative alert clustering method. The aim of the author is to reduce the huge number of alerts generated by IDS, mostly fruit of persistent configuration errors or particularities of devices, by bringing together alerts originated by the same root cause. The revelation of the root causes can also lead to the identification of attack scenarios and a better understanding of attacker's intention, even if his method was not design to fulfill this purpose. Julisch has become then a reference to multi-step attack detection, and his hierarchy-based similarity features have been much used in the literature [Xiao 2008, Li 2016]. A posterior proposal [Wang 2006a], based on a genetic algorithm, is proposed by other authors to improve the quality of the results returned by the method developed by Julisch.

As in the methods based on progressive construction, similarity for clustering methods is quantified by choosing a set of similarity features. But there are authors who have also incorporated extrinsic attributes to the methods. It is the case of Cuppens, that in the context of MIRADOR project developed a clustering method oriented to alert fusion [Cuppens 2001] which was successfully tested on multi-step attack detection. Their approach bases the similarity of two alerts on a set of expert rules defined for each one of the compared attributes and for each possible pair of alert type. However, his most important contribution to the domain is on causal correlation, about

which we speak in section 4.3.2.1. Another example is the clustering method developed by Murphy et al. [Murphy 2009, Murphy 2010], which uses a similarity matrix containing statistical data. The values of the cells represent the number of times each elementary attack has targeted each service in the network. Using this matrix, they find the links between elementary attacks using Divisive Hierarchical Clustering (DHC) on a social network graph. Hierarchical clustering techniques are also used in the work of de Alvarenga et al. [de Alvarenga 2018] to offer the analyst a selected and reduced visualization of attack scenarios.

On its side, the MLAPT system, proposed by Ghafir et al. [Ghafir 2018], takes as input alerts generated by a set of detection methods developed by the same authors and oriented to detect specific parts of the attack, such as infiltration of malware or bad usage of SSL certificates. Alerts are assigned to each one of the phases of a multi-step attack model. Clustering is based on the detection of alerts matching these phases in a proper order.

Nevertheless, most methods work only on similarity features based on the inherent attributes of the traces. For instance, Zhang et al. [Zhang 2015] introduce a clustering method based on a textual similarity feature applied to several attributes (see section 3.4.1, page 50). As the only published material is a poster in a conference, the method is not presented in detail. Their original contribution is that they work with alerts coming from a WAF (Web Application Firewall). A WAF works in a high level of abstraction, so the registered actions are more directly linked to the purpose of the attacker than, for example, the TCP connections that lay under them. Another example is GAC (Graph-based Alert Correlation) [Haas 2018], where the clusters are based on the equality of IP addresses and port numbers. On a later stage, clusters are joined to form multi-step attack scenarios by comparison of IP addresses using the Jaccard similarity index.

Clustering can also be applied to groups of traces instead of to individual traces. This is done by Kawakani et al. [Kawakani 2016, Kawakani 2017], who propose a method for hierarchical clustering of graphs representing attack strategies. The graphs are automatically derived from alerts in the same time window by a common element similarity feature considering the source and destination IP addresses (f_2 with $N = M = \{ipsrc, ipdst\}$). The resulting clusters can be later used to classify similar incoming scenarios.

There also exist approaches based on chaining several clustering steps. One example of this is the alert clustering method presented by Qiao et al. [Qiao 2012], composed of two stages. In the first one, alerts are grouped into sequences by means

of a correlation function built from the weighted sum of three similarity functions (see section 3.4.4, page 60). In the second stage, the sequences built in the first one are clustered together according to the distance between them. This distance depends on the number of operations of deletion and insertion needed to transform one sequence into the one we are comparing with. After the two clustering stages, the attack models are extracted by a loose application of LCS (Longest Common Subsequence).

Another example of chaining of methods is given by Manganiello, Colajanni and Marchetti [Colajanni 2010, Manganiello 2011], who propose the application of a Self-Organizing Map (SOM), a kind of auto associative neural network, followed by a k-means clustering phase. The input dataset of alerts is first preprocessed using the hierarchical clustering proposed by Julisch [Julisch 2003a] to merge similar alerts and reduce the number of elements to be processed. Then, the SOM reduces the dimensionality of the alerts by mapping similar alerts to close neurons in the model. The result is then clustered using k-means and the resulting clusters are joined according to a correlation weight, which is based on the distance between the neurons, the time difference between alerts and the alert type.

4.3.1.3 Anomaly detection

Anomaly detection methods learn from a dataset of traces clean of attacks and then consider as a threat the sequences differing from normal behavior. Similarity comparison is then made against a whole reference dataset, not only between the traces in the incoming data. The results are used differently than in the other similarity-based methods: we do not search the similarities but the differences. It is important to note that abnormal behavior do not necessarily correspond to an attack. The rate of false positives can then be high, but anomaly detection methods offer the possibility of finding previously unseen attacks.

While anomaly detection has been much used as a technique to detect single-step attacks [Ahmed 2016], not many solutions have been proposed for multi-step attack detection. One of them is presented by Mathew and Upadhyaya [Mathew 2009], who apply Principal Component Analysis (PCA) to build a model from attack-free data. Next, they project incoming data on the model. Abnormal sequences are then identified as those whose distance to the model is higher than a defined threshold.

In the anomaly detection method developed by Skopik, Friedberg et al., events from a training set free of attacks are linked in a random way to create hypotheses [Skopik 2014, Friedberg 2015]. They develop a mathematical framework to define hypotheses, rules and anomalies. Events not related to the hypothesis are considered as

anomalous and raise alerts. The multi-step attack perspective is only present during the characterization of the clean dataset, as detection is made event per event.

Hidden Markov Models (HMMs), that are considered in more detail when talking about methods doing statistical inference (section 4.3.2.2, page 81), have also been used in anomaly detection. This time, clean sequences are represented by HMMs, whose parameters are deduced from a clean training dataset. New sequences not corresponding to the trained HMMs are considered as attacks. Anming et al. [Anming 2004] implemented a method based on this in 2004, using the Segmental K-means algorithm to create the HMMs from a training dataset of OS audit data. 9 years later, Shin et al. [Shin 2013] apply the same method on the same dataset but on IDS alerts.

4.3.2 Causal correlation methods

In causal correlation, the causal progression of the sequences of traces is the key factor in the identification of multi-step attacks. In other words, previous steps in an attack determine the ones that follow, and a causal scheme can be derived from this relationship. This completely differs from similarity-based methods, where the found links depend on the similarity features between traces. In progressive construction methods (see section 4.3.1.1), causality can arise as a result, as a causal relationship may exist between traces connected by similarity features. But this is a causality observed *a posteriori*, while in causal correlation methods it is previously deduced from human experience or from the own traces and incorporated into the detection process.

Causal correlation methods have an important advantage: their process and results can be easily interpreted by a human analyst [Salah 2013], as causality is intuitively associated to the progression made by the attacker towards her goal. There may still be a high number of false positives, but less than in similarity-based methods, as safer hypotheses are made.

Depending on how causality is considered, we find two categories of causal correlation methods: prerequisites and consequences, and statistical inference. In methods based on *prerequisites and consequences*, the causal relationship of individual actions is explicitly coded in a database. *Statistical inference* focuses on the extraction of causality from the frequency of occurrence of actions with respect to other actions.

4.3.2.1 Prerequisites and consequences

In these methods, each trace is supposed to have associated a series of known prerequisites, also called pre-conditions, and consequences, or post-conditions. The *prerequisites* are the conditions to be given for an action to be performed, while the *consequences* are the possible effects of the action.

All the methods included in this category, except one [Zhai 2006], works with IDS alerts as the only type of trace. Each IDS alert represents an attack by itself, so the prerequisites become the conditions to be given for an attack to be successful, from the point of view of the attacker [Benferhat 2003, Ning 2010]. But as an IDS usually generates several alerts for each detected attack, there is usually a preprocessing phase to form *hyper-alerts* [Zhang 2017] (see page 36). The alerts contained in a hyper-alert have the same prerequisites and consequences and are temporally close to each other. Hyper-alerts are correlated through the automatic identification of prerequisites to consequences, returning attack scenarios composed of elementary attacks.

Shortly after the creation of the LAMBDA language [Cuppens 2000] (see section 3.3.2), Cuppens et al. became the pioneers in the development of a method based on prerequisites and consequences, under the MIRADOR project [Cuppens 2002c, Cuppens 2002a, Cuppens 2002b, Benferhat 2003]. In their proposal, the connection of two attacks, A and B, is made if the consequences of A partially match the prerequisites of B. Apart from a database of prerequisites and consequences, additional ontological rules are needed for the connection of some of the steps. Once this expert data is set up using the LAMBDA language, a set of correlation rules is automatically derived from it. These rules can then be applied to input alerts for finding attacks.

The other parent of multi-step attack detection based on prerequisites and consequences is Ning, the most cited author in the literature about multi-step attack detection. His team at North Carolina State University has been the most prolific one in this category of methods, both in terms of number of publications and of timespan of the project [Ning 2002a, Ning 2002c, Ning 2002b, Ning 2002d, Cui 2002, Ning 2003a, Ning 2003b, Xu 2004, Ning 2004a, Xu 2006, Zhai 2006, Ning 2010]. Their method was developed in parallel to the one by Cuppens et al., and independently according to Ning [Ning 2004a]. The formalism of both methods is different, but the principles behind the correlation process are very similar.

Over the course of the eight years during which Ning has made evolve his proposal, many improvements related to graph reduction or analysis have been incorporated. As an example, a proposed technique of link analysis gives some extra insight on the attributes of the steps in the attack scenario [Ning 2002b, Ning 2010]. Ning's team has

also developed a different way to represent the causal predicates [Xu 2004], focusing on triggering events and applying a hierarchical taxonomy to the attributes in the alerts. Finally, they have proposed the enrichment of IDS alerts with the inclusion of OS-level event logging [Zhai 2006] in the only method based on prerequisites and consequences that does not use only IDS alerts.

Cuppens and Ning have inspired a vast research about prerequisites and consequences for attack detection. Cheung et al. [Cheung 2003] use their own language CAML (see section 3.3.2) to develop a method under the EMERALD project [Porras 1997], the same as Zhou et al. [Zhou 2007] do with ACML (Attack capability modelling language) [Pandey 2008]. Wang, Liu and Jajodia [Wang 2005b, Wang 2006e, Wang 2008] improve the performance of the method proposed by Ning et al. [Ning 2002c] by adding a new element called Queue Graph. In their method, alerts are correlated only to the latest copy of each type of alert, not to all the past ones. Their attack graphs contain, apart from the causal information, the vulnerabilities of the attacked system. Finally, the most complete of the proposals extending the work of Ning is MARS (Multi-stage Attack Recognition System), the framework developed by Alserhani et al. [Alserhani 2010, Alserhani 2011, Alserhani 2012, Alserhani 2013, Alserhani 2016]. MARS also complements the prerequisites and consequences with information about the vulnerabilities in the assets. Using the method provided by MARS, Alnas et al. [Alnas 2013] are able to model the behavior of Zeus botnet.

Finally, Yan and Liu [Yan 2004, Yan 2005] propose an original approach: the use of a case grammar where the relationship between actions is expressed in plain English. In their model, actions are connected as verbs in linguistics using text categorization methods, after being transformed in semantic vectors.

4.3.2.2 Statistical inference

Statistical inference is the process of inferring from a dataset the distribution that generated it [Wasserman 2013]. Many methods have been developed based on this process. They base detection on a context-based feature, the probability of appearance (see section 3.4.3, page 55). These methods consider that there is a causal relationship between traces when they are statistically correlated [Qin 2003]. Following this idea, a statistical model is automatically extracted from a training dataset or from the own data where the detection process takes place. The probabilities contained in this model are used for detection and prediction of subsequent attacks, by linking pairs of traces with a high probability value. Although not specifying the details about the information that the statistical model should contain, the method proposed by

Geib and Goldman [Geib 2001] can be considered as the beginning point of statistical inference for multi-step attack detection. They propose the adaptation of probabilistic plan recognition, a good established field in Artificial Intelligence.

In general terms, the two main approaches to statistical inference are *frequentist inference* and *Bayesian inference* [Wasserman 2013]. Without going into specifics, frequentist inference assumes that conditions for the studied phenomena do not change in the long run, while Bayesian inference considers the probability of the phenomena as random variables [Koski 2011]. This is translated in two very different ways of deducing probabilities from a dataset: under the frequentist approach, the probability of a phenomenon is directly calculated by counting its number of occurrences and dividing by the total number of occurrences of all different phenomena, while under the Bayesian approach, a probability based on prior belief is assigned to the phenomenon and the value is adapted according to the frequency of the phenomenon.

There exist multi-step attack detection methods following any of the two approaches. First of all, we consider those following Bayesian inference. Markov models are one of the models incorporating Bayesian logic. They are composed of a finite set of states linked by probability values. They can be represented as a graph, which makes them a very suitable model for multi-step attacks, the states representing the actions executed by the attackers. The steps in the model are generally defined at the beginning of the process, and an initial value is assigned to transition probabilities. In any case, traces representing subsequent steps have one or several common attributes, whose choice depends on the author. For example, Farhadi et al. [Farhadi 2010] choose only the destination IP address, while Xuewei et al. [Xuewei 2014] also include the source IP address.

A Hidden Markov Model (HMM) is a special type of Markov model whose states are partially hidden. Only an observable outcome linked to each state is known. In multi-step attack detection, the states are the actions performed by the attackers and the observable outcomes are the traces corresponding to each action. The way to proceed stays the same as in classical Markov models, the HMM becoming a formalism to make the distinction between trace and action. Ourston et al. [Ourston 2003] were probably the first ones to use HMMs in the detection of multi-step attacks. They use IDS alerts, which are linked by IP address and classified into categories corresponding to each of the usual actions of a complex attack, which are, according to them, 'probe', 'consolidate', 'exploit' and 'compromise'. There are other methods using five states [Katipally 2011, Jia 2017] or even three [Chen 2016] when defining the HMM. The method proposed by Lee et al. [Lee 2008] considers distributed agents to detect each

stage of the attack. In the paper by Luktarhan et al. [Luktarhan 2012], HMMs are used but there is not a clear explanation of how the links between the states are made.

Instead of working with just a single HMM for any attack, a recent proposal by Holgado et al. [Holgado 2017] considers a different HMM for each type of multi-step attack. Another innovation with respect to the other methods is that the structure and the initial parameters of the HMMs are automatically derived from a training dataset of alerts. To do that, they apply a clustering algorithm based on the textual similarity between the alert types and the content of Common Vulnerabilities and Exposures (CVE) documents (f_4, see page 49). Shawly et al. [Shawly 2018] consider a very similar approach but not using the information from the CVE documents.

Furthermore, Kholidy et al. show in a set of three publications [Kholidy 2014b, Kholidy 2014c, Kholidy 2014a] that other variations of classical Markov models, such as Variable Order Markov Models (VMM), can be used to predict next steps in a multi-step attack. The approaches they propose highly depend on predefined models or signatures of attacks, but prediction is based on transition probabilities adapted from alert data. Fava et al. [Byers 2008, Fava 2008, Yang 2008] use another variation of Markov models, Variable-Length Markov Models (VLMM). They are derived from a set of known multi-step attacks, where incoming sequences are projected by attack description, category of the attack or destination IP address.

Despite of the ample use of Markov models, Bayesian networks seem the most pertinent model to model multi-step attacks for applying Bayesian inference. The main difference between a Bayesian network and a Markov model is that the former is directed and acyclic and the latter is not. This make Bayesian networks more useful in the expression of the causal relationships between the traces [Bishop 2006]. The same as with Markov models, methods working with Bayesian networks require a previously defined network with associated belief values. Some work exists where the Bayesian network is derived from expert data [Qin 2004, Ning 2007, Jemili 2008, Jemili 2009]. Even if the structure partially comes from expert data, Bayesian inference can be complemented by a statistical method not needing prior knowledge, as Qin and Lee do with a method based on the Granger Causality Test (GCT) [Qin 2003, Qin 2005, Qin 2007], a statistical method much used in Economics. This method is then retaken by Saikia et al. [Saikia 2018].

An alternative to Bayesian networks based on expert knowledge is the automatic deduction of the network and its initial beliefs from a training dataset during a previous offline phase. Doing so, no previous knowledge about the attacks is needed. For example, Ren et al. [Ren 2010] do this by testing several similarity features to find

the ones better representing the relationships between traces. Kavousi and Akbari [Kavousi 2012, Kavousi 2014] present a similar method but incorporate additional expert information to complete the model. Anbarestani et al. [Anbarestani 2012] base their model on a similarity feature involving only one attribute: the destination IP address. Sun et al. [Sun 2018] use system calls as nodes in the network, linked together by the objects they generate and use, such as files. Once built, the Bayesian network has to be prepared for online detection. To do that, Marchetti et al. [Marchetti 2011b] propose a pruning phase to remove the nodes whose correlation probability is lower than a dynamic threshold defined by the current statistics of the training dataset.

Apart from the work applying Bayesian inference, there is much work following a frequentist logic. In much of it, the sequences of traces potentially representing a multi-step attack are built through progressive construction (see section 4.3.1.1) and the probability associated to each pair of traces is deduced from the built sequences. This is the case of Nexat [Cipriano 2011], a system that groups into sessions the alerts with some connection between source and/or destination IP addresses. Lagzian et al. [Lagzian 2012] also consider only the IP addresses as linking feature, applying Bit-AssocRule, a variation of the Apriori algorithm. In the case of Man et al. [Man 2012], the matched features are the destination IP address, the attack type and the timestamp, while Kim and Park [Kim 2014] just use time differences and pairs of IP addresses, not giving much insight about the implementation. Other work [Ma 2008, Xian 2016, Zhang 2016, Lu 2018] simply takes the ID of the events and apply a sequential mining method to find frequent sequences.

There exist many different methods to extract the probabilities associated to each pair of traces from a training dataset. Z. Li et al. [Li 2007b, Li 2007c] present an algorithm based on classic association rule mining, where probabilities are obtained just by counting the times pairs of traces are found together. PrefixSpan algorithm is also used in some work, whether it be modified [Brahmi 2013, Lv 2015] or in its original form [Li 2016]. Sadoddin and Ghorbani [Sadoddin 2009] propose FSP_Growth, a method based on the FP_Growth algorithm [Bai 2011] that mines frequent patterns of alerts and arranges them in a tree. Ghorbani is also author of a later work with Soleimani [Soleimani 2012], where he reuses the correlation matrix proposed with Zhu [Zhu 2006] and commented in section 4.3.1.1, page 73. The method proposed by Ghorbani and Soleimani has a first phase of sequence identification using the correlation matrix. Then, a supervised Decision Tree (DT) learning method is applied. Other methods used are boosting algorithms [Ahmadinejad 2009] or the Generalized Sequential Pattern (GSP) algorithm [Bahareth 2013].

4.3.3 Structural-based methods

Many methods consider the structure of the network as a key element for intrusion detection. They incorporate in their detection engine information about the system to be defended, specially about the vulnerabilities affecting each asset. This information is *structural* in the sense that it only depends on the defended systems and not on the actions of the attackers. The potential actions done by an attacker can be deduced from this information. For example, if we know there is a Windows machine vulnerable to the EternalBlue exploit, we can predict that an attack such as WannaCry would include the infection of this machine in its path of actions (see section 3.2.1). As we mentioned in section 3.4.3, page 56, structural information is usually coded in the form of *attack graphs*.

We are not interested here in the construction of the attack graphs or in the prediction of the path taken on the resulting graph by a potential attack, but in the detection of multi-step attacks from real traces using attack graphs as a tool. This is the goal of structural-based detection methods. They project incoming traces into the attack graph representing the defended network to evaluate the most probable path the attack can follow.

Noel et al. [Noel 2004] were probably the first ones to project received IDS alerts into prebuilt attack graphs in 2004. Each vulnerability in the graph is linked to a known exploit that can take advantage of it. Alerts are matched to the exploits and the distance in the graph determines the final correlativity between alerts. Angelini et al. [Angelini 2018] follow a similar method but directly matching vulnerabilities and alerts, also incorporating a visualization interface to show the details of the detected attack to the analyst.

While some research uses traditional attack graphs as described in the literature, in standard [Roschke 2011, Fayyad 2013, Luo 2016, Kaynar 2017, Sicilia 2017] or enhanced [Çamtepe 2007] form, the network model used in TANDI [Holsopple 2006, Yang 2009] also includes information about the level of access privilege. TANDI is developed at the Rochester Institute of Technology (RIT), the birthplace of some of the most relevant ideas about structural-based multi-step attack detection. One of these ideas is the *cyber terrain* or *virtual terrain*, proposed by Fava, Holsopple et al. [Fava 2007, Holsopple 2008]. It is a manual model of a network where each asset is associated to its services, its version and the logical connections with other assets.

Another system born at the RIT [Du 2010] considers four categories: current state of the attacking source, current state of the target, firewall rule configuration and open services at the target. Prediction is made using two methods. The first one uses

Transferable Belief Model (TBM) to combine the concepts of Capability and Opportunity assessments developed in the context of the FuSIA system [Holsopple 2008]. The second one uses Fuzzy inference to merge Variable Length Markov Model (VLMM) estimates based on the attributes extracted from the alerts.

Chien and Ho [Chien 2012] present a system based on colored Petri nets. They model each attack plan in an abstract way and detection is made considering a metric of exploit certainty. On the other hand, Zhang et al. [Zhang 2008] propose to automatically build the trees used for detection. They do that assembling different elements (information about topology, vulnerability scan results, etc.) through the principles of causal correlation.

Some attempts have been made to apply Game Theory to capture the behavior of an attacker and a defender during the execution of a multi-step attack [Xupeng 2014, Haopu 2016, Hu 2017, Rass 2017]. Some of them are conceived to perform multi-step attack detection on incoming traces. In these proposals, the defender reduces the uncertainty of the attack through the signals received from IDS alerts, placing defenses in real time to avoid damages. For example, Lin et al. [Lin 2012] propose a model focused on attackers whose objective is to steal confidential data. And Yi Luo et al. [Luo 2014] present an algorithm called RDFP (Responses by Dynamic game tree-based Fictious Play) to be applied in a dynamic game tree.

4.3.4 Case-based methods

A broadly spread approach for intrusion detection is the comparison between the observations and a knowledge base of previously seen attacks. There are many methods applying this approach to multi-step detection. The knowledge base, containing a set of attack models or signatures, can be manually populated by security experts or attacks can be extracted from a dataset using automatic techniques. Honeypots can aid in the collection of real multi-step attacks for the development of case-based signatures [Vasilomanolakis 2016]. Modeling the behavior of human actors involved in security [Dutt 2013] can also be important to better understand how to develop the signatures. Newly discovered multi-step attacks, through an inference mechanism or by human intervention [Salah 2013], can be added to the database once they have been analyzed and a signature is built. Signatures can contain additional information that can ease detection, such as the criticality of the assets in the defended system [Soleimani 2008].

It is important to highlight the differences between case-based methods, structural-based methods and methods based on prerequisites and consequences, as their resemblance can lead to confusion. In cased-based methods, models of attacks representing

a specific type of multi-step attack are stored in the form of detection signatures in a database. Those signatures are then used to detect repeated occurrence of the same attack, with room for subtle variations in certain methods. Conversely, in structural-based methods, models are built only upon structural network characteristics. The received traces are projected against this model and the attack is detected by the match between the actions and the exploitable vulnerabilities in the network assets. Finally, in methods based on prerequisites and consequences, there is also a predefined model stored in a database: the set of prerequisites and consequences associated to each trace. Models are thus made trace by trace, and assumptions about a whole multi-step attack are deduced by combining them. Therefore, the way to proceed is different to that of case-based methods, where a model represents a complete attack.

The clear advantage of case-based methods over the others is that the number of false positives is low: we know what we search for and we look for the exact occurrence of it, so it is difficult to miss the mark. Nevertheless, if an attack is not in the database, it is not found. Case-based methods are only capable of detecting known multi-step attacks.

Multi-step attack signatures, generally conceived as a graph [Mathew 2005], can be represented using a language for rule representation. For example, Chintabathina et al. [Chintabathina 2012] propose A-prolog to apply logic programming to case-based detection, and Long and Schwartz propose XML [Long 2008]. However, there are languages that have been specifically created to model multi-step attacks, such as STATL [Eckmann 2002, Valeur 2004]; the chronicle formalism [Morin 2003, Wang 2004], or the tainting-based EDL [Jaeger 2015, Ussath 2016a, Ussath 2016b]. The details of these languages were explained in section 3.3.2 when addressing the languages to represent multi-step attacks.

Case-based detection methods can be very simplistic, just based on pattern matching [Kannadiga 2007, Panichprecha 2007, Katipally 2010, Xuewei 2010], or cryptic about how the signatures are built [Schindler 2017, Wen 2017]. Some of them embellish the process with a previous phase of preprocessing, such as the alert aggregation proposed by Xiao and Han [Xiao 2006]. But we can find more complex proposals, where the system incorporates additional mechanisms to improve the detection. For example, MASP (Mining Attack Sequential Pattern) [Li 2007e, Zhang 2007, Wang 2007b, Li 2007d, Wang 2007a] uses incremental mining of subsequences of known multi-step attacks. Thanks to that, MASP can find variations of the attack signatures. Zali et al. propose a similar method [Zali 2012, Zali 2013], where they represent the attack scenarios as Causal Relation Graphs (CRG), with queues of alerts placed in each of the

vertex of the graph. This structure eases the implementation for real-time detection and the prediction of missing alerts. The correlation system proposed by Chien et al. [Chien 2007] works with primitives representing parts of attack scenarios. Primitives are built from a pre-defined ontology, but not enough detail is given about how they are created. The ONTIDS framework [Zargar 2014] is also based on an ontology, which works at different levels: 'context', 'alert', 'attack' and 'vulnerability'. A similar ontology is proposed by Wang et al. [Wang 2018], who extend its application to general events. The connection between all these levels is exploited in the definition of combined signatures. Giura and Wang [Giura 2012a, Giura 2012b] propose a model where the stages of the attack are arranged in a layered pyramid, with the goal at the top of the pyramid and the previous steps stratified by layers. In each face of the pyramid we can find a different domain: physical, network, application, user, etc. This model is used in a detection framework where correlation rules are based on signatures, profiling or security policies. An example of knowledge base automatically built is JEAN (Judge Evaluation of Attack Intension) [Cheng 2011], where the attack database is built from a training set of IDS alerts using J-Fusion, an algorithm for alert fusion. The system mines other occurrences of the same attacks using a method inspired by the generalized Hough transform, an image processing method to identify geometric forms. Another example is the work of Amos-Binks et al. [Amos-Binks 2017], where attack scenarios are represented in a plan library automatically generated from a set of IDS alerts, according to a list of pre-defined goals.

There are several methods that assume a fixed structure for any multi-step attack, performing case-based detection from a generalized model of the steps that an attack should have. For example, in the INFERD system, [Sudit 2005, Stotz 2007, Mathew 2010] this model is called Guidance Template and has seven stages. Each one of these stages has a certain subset of IDS alerts associated to it, according to their type. INFERD groups the alerts to form sequences with common source and/or destination IP addresses according to the order defined in the Guidance Template. Z. Liu et al. [Liu 2008] propose a similar system but using just four stages: (1) 'probe', (2) 'scan', (3) 'intrusion' and (4) 'goal'.

The case-based methods presented so far are centralized: traces are collected in a central point, where the detection of attack scenarios takes place. However, there are also some distributed methods where detection of individual steps is done by local agents scattered throughout the network. This kind of detection system has been thoroughly studied as a particular field of security detection [Zhou 2010]. An example of a distributed method for multi-step attack case-based detection is Quicksand

[Kruegel 2002], where handmade multi-step attack signatures are translated into pattern graphs and sent to the local agents. Each agent is responsible of detecting the fraction of the signature represented by a node of the graph. Each time an agent detects its assigned pattern, a message is sent up in the tree, until Quicksand identifies the full signature, represented in the root node, and raises an alert. The distributed system proposed by Vogel et al. [Vogel 2011a, Vogel 2011b] is based on the Petri net principle and uses signatures written in EDL [Meier 2007] (see section 3.3.2). Signatures are divided in minimal parts and sent to the local agents. In both methods, rule division has to be manually made by the security expert.

4.3.5 Mixed methods

As we have mentioned earlier, there is some work where several approaches are integrated together in the same system, with no clear prevalence of one over the other. For instance, the framework RTECA [Ramaki 2015a] combines statistical inference, similarity of trace types through a correlation matrix and similarity of IP addresses and ports. Frequent sequences are arranged in event trees, which are fed with similar sequences during the execution. The analysis of frequent patterns can also be combined with a phase of clustering [Faraji Daneshgar 2016]. The creators of RTECA also propose in another paper [Ramaki 2015b] a very similar system to their previous one but based on Bayesian networks. Lessons learned from both systems are incorporated in a recent three-phase framework developed by the same authors [Ramaki 2016], where there is no dependence of a predefined correlation matrix.

In ASEA system [Farhadi 2011], statistical inference and similarity-based correlation are merged to apply plan recognition using HMMs. On the other hand, Shittu proposes in the third chapter of her Ph.D. thesis [Shittu 2016] to combine Bayesian inference with similarity-based correlation, offering a different perspective to existent Bayesian methods. Furthermore, Du et al. [Du 2009] start by identifying sequences of IDS alerts with the same victim IP address and assigning a severity score to each step. Then, the found sequences are mined using three techniques, each of them based on a totally different mechanism: Longest Common Subsequences (LCS), Fourier transform and social networks.

There is much work combining prerequisites and consequences with other approaches. Ning et al. propose to mix their method with similarity-based methods [Ning 2004c, Ning 2004b]. Their purpose is to merge the graphs belonging to the same attack scenario but being mistakenly separated due to missing IDS alerts. Yu and Frincke [Yu 2004, Yu 2007] combine prerequisites and consequences and statistical

inference, applying a colored Petri net with hidden states. Another proposal is that of Saad and Traore [Saad 2012], composed of a phase of clustering using an intrusion ontology and a subsequent phase based on prerequisites and consequences. This method is later improved by other authors to allow online analysis [Barzegar 2018]. A similar method is used by Al-Mamory and Hong Li Zhang [Al-Mamory 2007], who additionally propose the application of an Attribute Context-Free Grammar to model the attacks [Al-Mamory 2008, Al-Mamory 2009]. This grammar contains information about the level of similarity between the alerts, their prerequisites and consequences and the structure of known attack scenarios.

We also find some publications proposing the combination of methods that we have already mentioned in the context of other approaches. For instance, causality-based INFERD [Sudit 2005, Stotz 2007, Mathew 2010] is integrated with the structural-based TANDI system [Holsopple 2006] by Yang et al. [Yang 2009]. Furthermore, the SATA (Security Alerts and Threat Analysis) platform [Wang 2006b] combines the SF algorithm [Wang 2006d, Wang 2006c] with MASP [Wang 2007b, Li 2007d, Wang 2007a, Zhang 2007]. Marchetti, Colajanni and Manganiello propose a framework [Marchetti 2011a] to bring together their two approaches: the one using SOM [Colajanni 2010, Manganiello 2011] and the pseudo-Bayesian one [Marchetti 2011b].

There are some frameworks in the literature proposing a whole end-to-end correlation process focused on multi-step attack detection. In the one proposed by researchers from Palo Alto Research Center and Galois Inc. [Abreu 2015], a mixture of methods are applied in different stages, from activity classification to alert ranking. Valeur et al. [Valeur 2004] introduce a whole correlation system divided in several phases. Among those phases, the ones related to our research are the four aiming to link different alerts for composing scenarios: thread reconstruction, session reconstruction, focus recognition and multistep correlation. WMAPRM (Wireless Multi-step Attack Pattern Recognition Method) [Chen 2014a] also merges different detection approaches for the specific case of wireless data, where the level 2 of the OSI model is more relevant than the network level. Another correlation framework is the one proposed by Ahmed [Saad 2014], where attack scenario construction is made combining semantic-based clustering and analysis of pre-defined consequences of alerts. He also proposes previous phases for alert aggregation and verification.

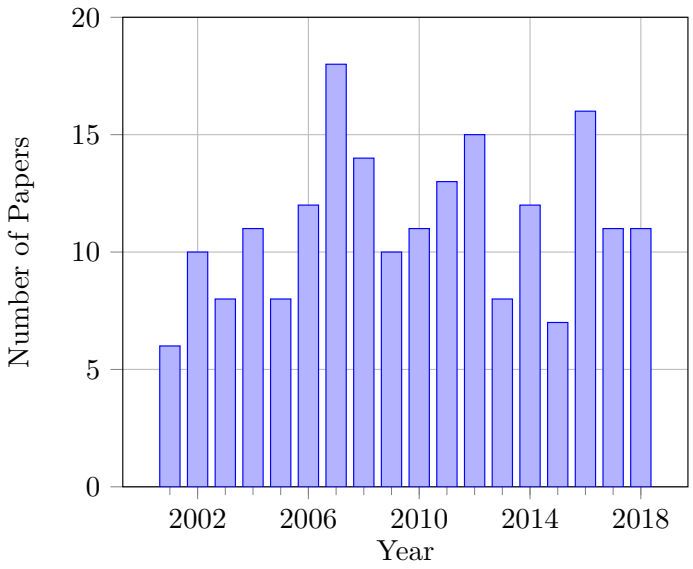


Figure 4.3: Number of publications on multi-step attack detection per year.

4.4 The field in perspective

Apart from the analysis of the methods proposed in the literature, a statistical analysis of the selected publications can give us an insight about how the field has evolved and its present state. The objective is to check how pertinent is our research question with respect to the trends in multi-step attack detection. We review the number of publications and the duration of research in section 4.4.1; the type of data and the origin of knowledge used in the methods in section 4.4.2, and the reproducibility of experiments in section 4.4.3.

4.4.1 Number of publications and duration of research

One of the selected metrics is the number of publications per year. It is represented in aggregated form in the histogram of Figure 4.3 and separated by approach in Figure 4.4. We see that the publication activity has been more or less constant during the years, giving an average of around 10 published papers about multi-step attack detection per year. This amount is not too high if we compare it with the total number of publications about Cybersecurity. Only in Springer[2], 5,822 conference papers about Cybersecurity were published in 2018. If we filter by the same year in Google Scholar, around 7,230 results are returned when searching for "intrusion detection", and 3,560 when searching for "attack detection"[3]. It seems clear that multi-step attack detection

[2] https://link.springer.com/

[3] Both terms are searched using the quotation marks in Google Scholar

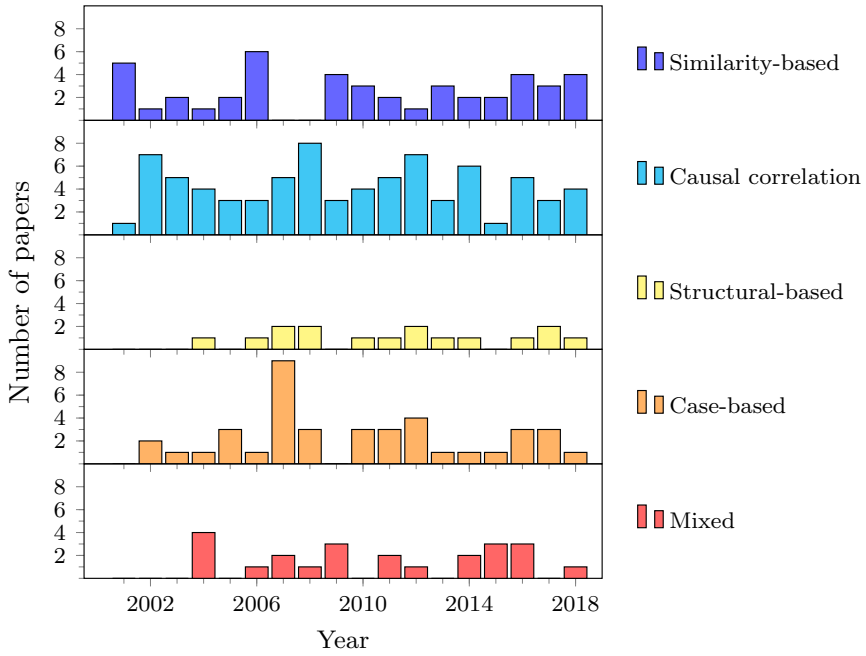


Figure 4.4: Number of publications on multi-step attack detection per year, classified according to the followed approach.

is still an active field but that does not represent much of the whole set of publications in Cybersecurity or even in general attack detection.

On the other hand, we have extracted the most relevant publications and authors in the domain. The 20 most cited publications are listed in Table 4.1. A list of researchers appearing in 5 or more publications is presented in Table 4.2. The number of citations, in these tables and in the rest of this thesis, has been extracted from Google Scholar, which we consider as the most complete search engine in terms of number of references[4].

There are two remarkable conclusions that can be extracted from these tables and from the ones presented in the Appendix D. The first one is that only a reduce number of authors gets a high number of citations. This is probably true for any research domain, especially for a young field as multi-step attack detection, with only 18 years old.

The second conclusion is that there is a lack of continuity in developing multi-step attack detection methods. The first indicator of this is the high number of proposed methods, 138, in only 201 publications. This means that almost 70% of the publications propose a new method instead of working in the improvement of existent ones.

[4]The lack of an API in this platform allowing a high number of requests has forced us to program a Python script where each request follows the previous one after a random interval of time, to avoid being blocked by Google

Ref.	Title	Year	Cit.
[Cuppens 2002c]	Alert correlation in a cooperative intrusion detection framework	2002	978
[Valdes 2001]	Probabilistic alert correlation	2001	961
[Ning 2002c]	Constructing attack scenarios through correlation of intrusion alerts	2002	693
[Julisch 2003a]	Clustering intrusion detection alarms to support root cause analysis	2003	543
[Valeur 2004]	Comprehensive approach to intrusion detection alert correlation	2004	542
[Eckmann 2002]	STATL: An attack language for state-based intrusion detection	2002	514
[Cuppens 2001]	Managing Alerts in a Multi-Intrusion Detection Environment	2001	379
[Ning 2004a]	Techniques and tools for analyzing intrusion alerts	2004	376
[Dain 2001b]	Fusing a heterogeneous alert stream into scenarios	2001	349
[Julisch 2002]	Mining intrusion detection alarms for actionable knowledge	2002	348
[Cheung 2003]	Modeling multistep cyber attacks for scenario recognition	2003	309
[Qin 2003]	Statistical causality analysis of infosec alert data	2003	280
[Julisch 2001]	Mining alarm clusters to improve alarm handling efficiency	2001	247
[Morin 2003]	Correlation of intrusion symptoms: an application of chronicles	2003	239
[Ning 2003b]	Learning attack strategies from intrusion alerts	2003	237
[Wang 2006e]	Using attack graphs for correlating, hypothesizing, and predicting intrusion alerts	2006	229
[Ning 2002b]	Analyzing intensive intrusion alerts via correlation	2002	228
[Qin 2004]	Attack Plan Recognition and Prediction Using Causal Networks	2004	213
[Geib 2001]	Plan recognition in intrusion detection systems	2001	206
[Ning 2004c]	Building Attack Scenarios through Integration of Complementary Alert Correlation Method	2004	200

Table 4.1: Top 20 entries according to the number of citations in Google Scholar.

117 out of 138 methods are so far considered only in one publication, with no further continuity.

This does not mean that there are no long-lasting projects in multi-step attack detection. Some of the most cited authors have been working in the field for long time, in some cases collaborating in the development of different methods. The best examples are Peng Ning [Ning 2004c, Ning 2002c, Ning 2002b, Ning 2004a, Ning 2003b], and Frédéric Cuppens [Cuppens 2002c, Cuppens 2002a, Benferhat 2003, Cuppens 2002b], who in parallel laid the foundations of causal correlation through prerequisites and consequences. Ning, with the help of other collaborators as Yun Cui, has developed this method at the North Carolina State University during 8 years, being the author with the highest number of citations in multi-step attack detection. Although less prolific, Cuppens counts with the most cited publication in the field [Cuppens 2002c] and he has also explored clustering [Cuppens 2001]. The team directed by Ali A. Ghorbani [Zhu 2006, Sadoddin 2009, Wang 2010, Soleimani 2012, Bateni 2013a] has also conducted long-lasting projects about multi-step attack detection. This team is also a reference in the development of new datasets for testing the methods (see section 7.1.2).

The lack of continuity in research about multi-step attack detection methods enormously contrast with the threat posed by these attacks, their high incidence and the effort devoted in public and private institutions to fight them. A good example of the bad consequences a multi-step attack can have is WannaCry, presented in the Intro-

Author	# of pub.	Citations			References
		Total	Max.	Min.	
Ning, Peng	11	2057	693	2	[Ning 2002a, Ning 2002b, Ning 2002c] [Ning 2003b, Ning 2004a, Ning 2004b] [Ning 2004c, Xu 2004, Xu 2006] [Zhai 2006, Ning 2010]
Wang, Li	11	168	42	2	[Wang 2006b, Wang 2006c, Wang 2006d] [Li 2007b, Li 2007c, Li 2007d] [Li 2007e, Wang 2007a, Wang 2007b] [Zhang 2007, Wang 2010]
Li, Zhitang	11	167	42	2	[Wang 2006b, Wang 2006c, Wang 2006d] [Li 2007b, Li 2007c, Li 2007d] [Li 2007e, Wang 2007a, Wang 2007b] [Zhang 2007, Ma 2008]
Yang, Shanchieh J.	10	343	80	1	[Holsopple 2006, Fava 2007, Byers 2008] [Fava 2008, Holsopple 2008, Yang 2008] [Du 2009, Yang 2009, Du 2010] [Murphy 2010]
Ghorbani, Ali A.	8	318	130	8	[Zhu 2006, Soleimani 2008, Sadoddin 2009] [Ren 2010, Wang 2010, Soleimani 2012] [Bateni 2013a, Bateni 2013b]
Xu, Dingbang	7	1015	376	2	[Ning 2003b, Ning 2004a, Ning 2004b] [Ning 2004c, Xu 2004] [Xu 2006, Ning 2010]
Holsopple, Jared	6	251	80	23	[Fava 2007, Yang 2009, Du 2010] [Holsopple 2008, Holsopple 2006, Yang 2008]
Lei, Jie	6	105	42	2	[Wang 2006c, Li 2007b, Li 2007c] [Li 2007d, Wang 2007a, Wang 2007b]
Cuppens, Frédéric	5	1545	978	36	[Cuppens 2001, Cuppens 2002a, Cuppens 2002b] [Cuppens 2002c, Benferhat 2003]
Cui, Yun	5	1406	693	17	[Cui 2002, Ning 2002a, Ning 2002b] [Ning 2002c, Ning 2004a]
Sudit, Moises	5	237	80	9	[Sudit 2005, Holsopple 2006, Stotz 2007] [Yang 2009, Mathew 2010]
Li, Dong	5	92	42	10	[Li 2007b, Li 2007c, Li 2007e] [Wang 2007b, Zhang 2007]
Meinel, Cristoph	5	103	88	2	[Roschke 2011, Fayyad 2013, Jaeger 2015] [Ussath 2016a, Ussath 2016b]
Alserhani, Faeiz	5	63	42	0	[Alserhani 2010, Alserhani 2011, Alserhani 2012] [Alserhani 2013, Alserhani 2016]

Table 4.2: Ranking of top authors according to the number of publications about multi-step attack detection included in the survey.

duction of this thesis. One of the reason of this lack of continuity can be the absence of available real data for developing and testing the methods. In consequence, there is also a lack of possible multi-step attack instances to study. In the most broadly used public dataset, DARPA 2000, there are just two different multi-step attacks, the ones described in section 3.2.2. In the supposed successor of DARPA 2000, ISCX 2012 dataset, only one multi-step attack is found (section 3.2.3).

4.4.2 Type of data and origin of knowledge

None of the public datasets contains a set of events where all the actions in the multi-step attack are represented. The traces contained in them are the raw packets captured

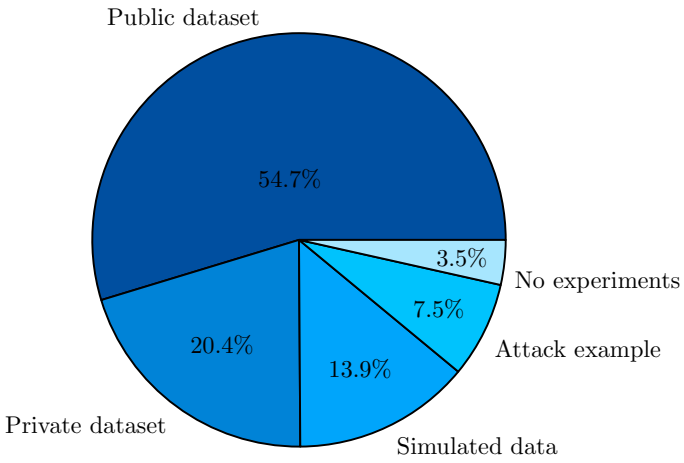


Figure 4.5: Distribution of publications by the type of data for experiments.

Type of Trace	Number of pub.	Approaches
Only alerts	172	All types
General events	22	Similarity-based, mixed, case-based and causal correlation
Traces with triggering alerts	5	Similarity-based and causal correlation
Packets	2	Case-based and causal correlation

Table 4.3: Types of trace and number of publications using them.

by sniffing. Correlation of actions generally need a higher level of abstraction that the one contained in the packets. To properly exploit these traces, it is necessary to preprocess them by replaying the traffic and feeding with it an IDS that can express the malicious actions in the packets as a set of alerts (events). As a result, and despite of the effort made by researchers in using public datasets in the evaluation of the methods (see diagram in Figure 4.5), there is a lack of variety in the type of trace used in evaluation. The results are shown in Table 4.3, where the number of publications using each type of trace is shown[5]. More than 85% of the publications are exclusively focused on the analysis of alerts. Even if authors such as Brogi and Tong [Brogi 2016] consider that detecting a multi-step attack is the same as highlighting the links between elementary attacks, general events can represent actions that are not necessarily malicious but that can be part of an attack. The development of a public dataset composed of logs containing information about several variations of a multi-step attack would improve the quality of the evaluation.

[5]Methods using "traces with triggering alerts" are those ones where alerts are used to identify the presence of the attack and other traces are incorporated then to complete the detection process.

Another aspect of the analyzed methods is how models are enriched to work in the detection of multi-step attacks. According to this, we classify methods into three categories: manual, supervised or automatic. In *manual methods*, detection models are manually built. The knowledge of the attack is directly coded by the analyst, who has to be experienced in attack modeling. The objective is to find known attacks or slight variations of them. All of the reviewed case-based methods (see section 4.3.4) and those based on prerequisites and consequences (section 4.3.2.1) are of this type. Then, *supervised methods* follow the same principle as the homonymous machine learning methods: attack models are created and enriched through a process of automatic learning from training data. Finally, in *automatic methods*, the specificities of the detection models are automatically learnt during the detection process. Supervised and automatic methods are developed thinking *how* an attack is, as no specific detection model is given.

In Figure 4.6 we represent the distribution of publications under each of these categories. We can see that publications presenting manual methods represent a 55.2% of the total. The lack of a global detection model, a set of characteristics shared by every multi-step attack, does not allow developing solid automatic methods. On its side, supervised methods are hindered by the availability of sound and reliable datasets for the training phase.

The prevalence of manual methods is an indicator of how the security analyst is a key piece in the process of detection. These methods require a knowledge base of detection models, sometimes called signatures. The research question of this thesis addresses this gap between the discovery of an attack, generally after investigation, and the construction of the detection model. The goal of this thesis is to propose a

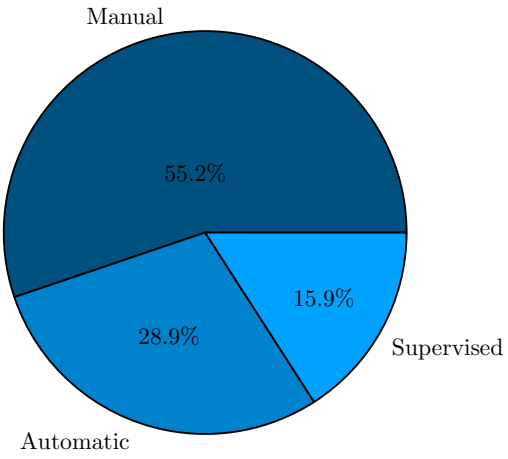


Figure 4.6: Distribution of publications by the origin of the knowledge about attacks.

model which can act as a bridge joining the two processes and maintaining an active investment of the analyst. This process is what we call *identification* (see page 4). The wide presence of manual methods for detection and the absence of research addressing the identification problem for the context of multi-step attacks justify the need to answer the research question.

4.4.3 Reproducibility of experiments

An important aspect to study in multi-step attack detection is the reproducibility of presented methods and experiments. *Reproducibility* is the ability to replicate the results obtained in a published experiment in similar conditions as were set by the original researcher [Goodman 2016]. It can be distinguished from *replicability*, the ability of reproducing the results but using different data that the one proposed in the original research. Replicability is harder to attain, as the proposed hypotheses need to be true for any dataset and not only for the one used in the published experiments. Reproducibility is a minimum standard for defending a claim as scientific [Peng 2011].

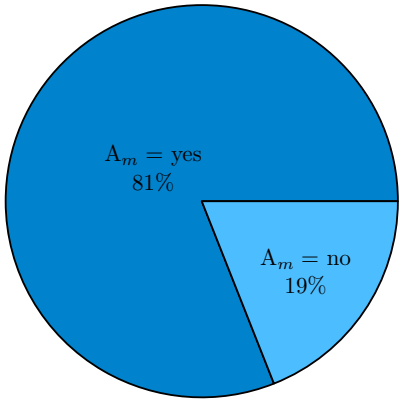
We have studied the reproducibility of the experiments presented in the selected corpus of publications. To well structure our study, we have defined reproducibility through a set of criteria that is similar to the one proposed by Leek and Peng [Leek 2015]. Three conditions have to be met for considering a publication as 'reproducible' in terms of experiments: method, data and knowledge have to be accessible. Accessibility of these three elements is explained below:

- **Accessibility of method** (A_m): The proposed multi-step attack detection method is exposed in such a clear way that its functioning can be reproduced without elements from outside the publication. To fulfill this condition, method can rely on an explanation of the method in plain language or in the form of pseudocode. Another alternative is to present a downloadable implementation of the method [Brogi 2016].
- **Accessibility of data** (A_d): At least a public dataset is used in the experiments. It can be a well-known public dataset or a new one made available by the authors of the publication. This condition is not considered to be true if the method is tested with a case study.
- **Accessibility of models** (A_k): If the detection method relies on models, e.g. a set of signatures in case-based methods, the ones used in the experiments are provided in the publication. In the case that the method uses a training dataset for automatic learning, we consider that this condition is accomplished if this training dataset is available. This condition is not considered for automatic

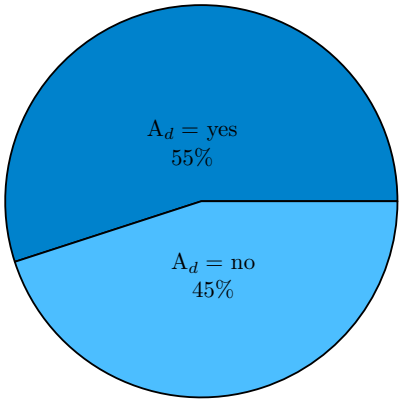
methods (see section 4.4.2), where no previous model is needed.

These three conditions have been studied for all the publications in the corpus and the results are shown in Figures 4.7a, 4.7b and 4.7c. In Figure 4.7d, we divided the corpus in publications with reproducible experiments, for which the three conditions are true, and those without them, when at least one of the conditions is not fulfilled. It is of serious concern that most of the publications (69%) do not offer full reproducible experiments. There is even a 19% of publications proposing multi-step attack detection methods that cannot be reproduced.

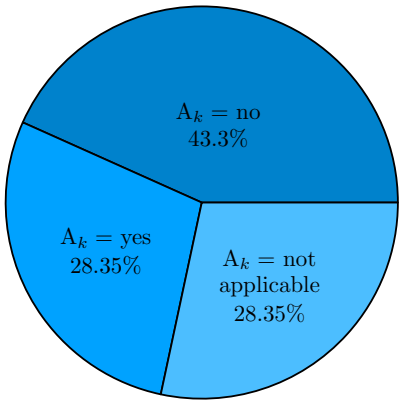
These are global statistics of a field lasting 18 years, and the situation could be better in the last few years. But if we look at the proportion of the publications including reproducible experiments per year, in Figure 4.8, we see that the evolution has not been very favorable. This casts doubts on the scientific quality of the work in



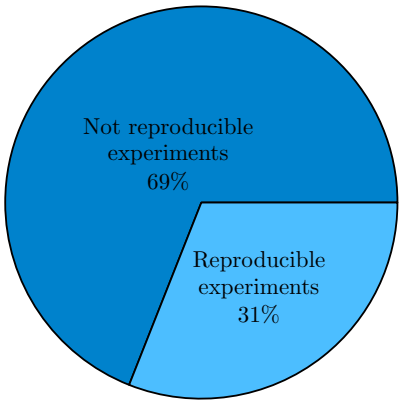
(a) Accessibility of method A_m.



(b) Accessibility of data A_d.



(c) Accessibility of models A_k.



(d) Experiments reproducibility.

Figure 4.7: Distribution of publications by reproducibility factors

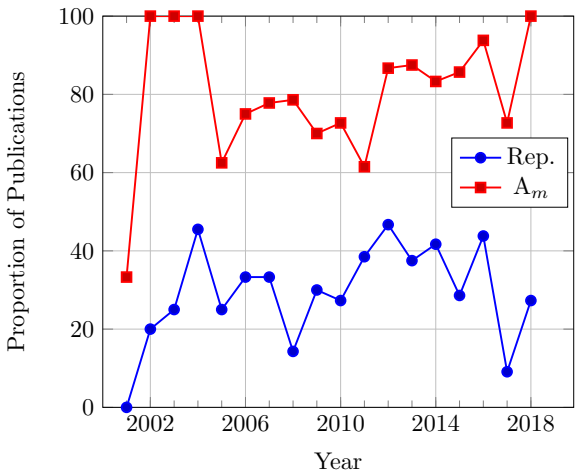


Figure 4.8: Papers with access to methods and reproducible experiments per year.

multi-step attack detection.

The reasons behind this lack of reproducibility do not forcibly rely on neglects by authors but on the limitations of the field. Public datasets are scarce and a big number of researchers have to simulate their own data. On top of that, the most well-known and spread datasets containing multi-step attacks, DARPA 2000 (see section 7.1.1) and ISCX (section 7.1.2), do not have enough instances of multi-step attacks. Conversely, datasets furnished by private companies, such as the HuMa dataset used in this thesis (section 7.1.3), cannot be published for confidentiality reasons, even if the benefit of sharing research data are evident [Wicherts 2012]. The reason of this secrecy is the nature of the data, which can contain lots of sensitive information, both in terms of security and privacy. A possible solution to share this data would be to apply anonymization methods [Slagell 2005], but they have been proved to be far from infallible [Narayanan 2008, Ji 2014].

To bring this topic to a conclusion, many of the studied publications that contains reproducible experiments have a few or zero citations in Google Scholar, as we can see in the tables of Appendix D. We hope this systematic review will help to make them more known so other researchers can improve the methods proposed in them.

4.5 The involvement of the security analyst

After having analyzed the multi-step attack detection methods, we can conclude that the human analyst is considered most of the time as placed before or after the detection process. On the one hand, its role before detection is the development of the detection models and the configuration of the parameters of the detection method. For example,

in methods based on prerequisites and consequences (section 4.3.2.1), a list of the possible events enriched with their pre and post conditions has to be provided. In the case of structural-based methods, structural information from the defended network is needed. On the other hand, the role of the analyst after the detection process is to verify the generated alerts and to fix the consequences of the attacks or even to stop them in time.

Only a couple of publications about multi-step attack detection propose the involvement of the human analyst during the detection process. On one side, we have the methods based on Game Theory [Lin 2012, Luo 2014], where the analyst has an active part responding to each attack attempt by the deployment of defenses in real time. However, they are just reduced to theoretical models that have not been tested with data.

Another proposal is the framework developed by Shaneck et al. [Shaneck 2006]. They claim that this framework keeps the "human analyst in the loop" by the control of the output from the first two phases of the system: the identification of relevant alerts (anchor points) and the context extraction. The analyst would have the role of selecting interesting events or critical points in the network to make the system focus on them. Unfortunately, the involvement of the human is just limited to this description and to the promising diagram of the framework, reproduced in Figure 4.9.

To find other examples of involvement of the security analyst in the process of detection we need to look to other domains. Majeed et al. [Majeed 2018] propose a system thanks to which the analyst can explore in real time how rules are matched in a SIEM (see page 19). The objective is to detect abnormal behavior the earliest as

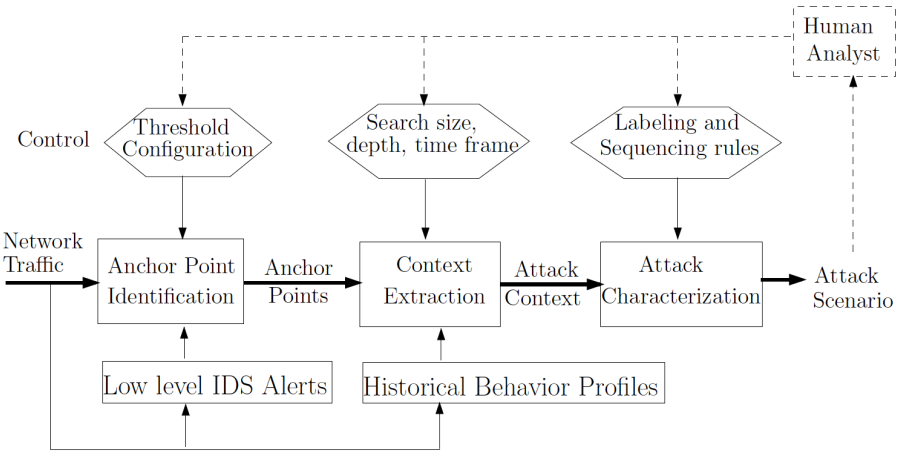


Figure 4.9: Diagram of the framework by Shaneck et al., adapted from [Shaneck 2006].

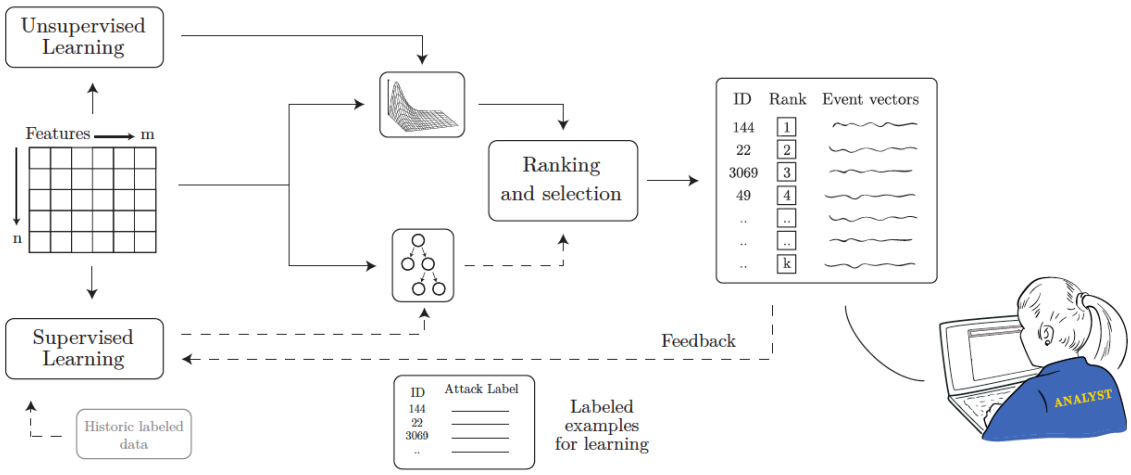


Figure 4.10: The AI^2 system. Image adapted from [Veeramachaneni 2016].

possible and being able to react against ongoing attacks. This is especially critical if the attack matched by the SIEM rule spans on a long period of time. In this system, the security analyst is involved in detection in the sense that she can check and analyze the process for preventing unwanted situations. However, the flow of information happens only from the system to the analyst, and it is not envisaged the improvement of the detection capabilities by online feedback from the analyst.

There exist several proposals where the analyst is involved in the online tagging of data. These methods are based on machine learning algorithms that send selected queries to the analyst for improving the learning process. This process is called *active learning*.

One system using active learning is AI^2 [Veeramachaneni 2016], based on a continuous process with several phases (see Figure 4.10). First, an outlier detection system reduces the amount of income events by the combined used of several unsupervised machine learning algorithms. This results in a much smaller set that can be presented to the analyst for labelling. Data is presented to the analyst through a user interface that ranks the outliers by their importance. Then, the labeled result is used by a supervised learning module to create a detection model.

Another example is ALIDS (Active Learning Intrusion Detection System), which uses a random forest classifier and k-means clustering [McElwee 2017]. Events sent to the analyst, called 'oracle', are those ones that are uncertainly classified, not a small set found using an unsupervised method, as it happens in AI^2. A similar approach is followed by ActivSVDD [Görnitz 2009], where the results obtained using support vector domain description (SVDD), a data description for outlier detection, are improved

by querying the analyst about low-confidence observations. On its side, Mao et al. [Mao 2009] consider active learning in a multi-view approach using two views: network-based and host-based. Lastly, the method proposed by Li and Guo [Li 2007a] applies TCM-KNN (Transductive Confidence Machines for K-Nearest Neighbors), querying the analyst for the most informative events.

Finally, we cannot end up this section without addressing the HuMa architecture [Navarro 2017], the one proposed in the context of the homonymous project financing this doctoral research. HuMa considers the human as the main actor in the analysis of threats. In fact, the name HuMa directly refers to the presence of the human in the attack analysis: it stands for "the human at the center of Big Data analysis for Cybersecurity"[6]. The analysis is organized around three layers: the *event layer*, where individual traces are represented; the *context and attack pattern layer*, which gathers information about technical requirements of the attacks; and the *assessment layer*, where information from complex attacks is extracted. Models developed in this thesis are intended to work in the assessment layer, together with the ones furnished by other partners in the project.

The model at the origin of HuMa is KILS (Knowledge and Information Logs-based System), proposed by Legrand et al. [Legrand 2014]. Its structure is divided in four levels: *real world*, *storage*, *inference* and *expert knowledge*. This model includes a feedback loop to introduce the knowledge from the human analyst into the system. Doing so, the attack detection algorithms contained on it can learn and improve their results. Communication with the human is eased by the decomposition of incoming events into abstract concepts. These concepts are inspired by the Action Theory and the principles of criminal investigations, but their details are not disclosed.

4.6 Summary

This chapter has been devoted to present one of our contributions: a systematic survey on multi-step attack detection. The results of the survey have been already published in a scientific journal [Navarro 2018a]. In this chapter, we have updated this research and improved the exposition of the reviewed methods. We have started describing the methodology followed in the systematic survey, listing the inclusion and exclusion criteria in section 4.2.1 and describing the research process in section 4.2.2. Following this, the methods chosen to be part of the bibliographic corpus have been described. They are classified by approach: similarity-based methods (section 4.3.1), causal corre-

[6] *L'humain au cœur de l'analyse de données massives pour la sécurité* in French

lation (4.3.2), structural-based methods (4.3.3), case-based methods (4.3.4) and mixed methods (4.3.5). In section 4.4, we have presented some conclusions about the state of the field based on the statistics extracted from the selected corpus. Finally, some examples of security systems placing the human analyst as an important piece in the detection process have been introduced in section 4.5. Little has been done in multi-step attack detection to address this topic, so we had to look to other domains to get information that allows us responding our research question.

At the same time as being one of the contributions, this survey constitutes a solid bibliography for this thesis. We can conclude that, up to now, proposed multi-step attack detection methods consider the security analyst as being *before* the detection process, designing the detection models, or *after* it, verifying the alerts and acting in consequence. But no method proposes the identification of the detection models among the alternative cases derived by the analyst during investigation. We need to look in different fields to identify a similar process. This and other related topics are reviewed in the next chapter.

Chapter 5

Abstract Attack Scenario Graphs (AASG)

Contents

5.1 From Concrete to Abstract ASG 104
5.2 Definition of AASG 114
5.3 Creating Abstract Attack Scenario Graphs 124
5.4 Implementation 127
5.5 Summary 132

Hold to the now, the here, through which all future plunges to the past.

— James Joyce, *Ulysses*

Our research question, as defined in page 3, points out to two objectives: a) to help the security analyst to decide between alternative scenarios of multi-step attacks (the identification process) and b) to perform detection of the attacks at the same time as this process takes place. The origin of these goals is the problem confronted by the security analyst of knowing which are the exact actions composing a multi-step attack among all the set of actions in the network registered in traces.

We propose in this chapter a structure to meet these objectives, called *Abstract Attack Scenario Graph* (AASG). It is a graph with each node representing a set of possible events generated in the network. Nodes are arranged in the AASG to represent different multi-step attack scenarios with some steps in common. Even if they seem similar, an AASG is very different to a CASG, the model to represent multi-step attacks presented in section 3.3.3, page 42. A CASG represents an existent multi-step attack, but an AASG codes the alternative scenarios conceived by the security analyst.

We start this chapter presenting the ideas behind AASGs in section 5.1. Then, the definition of an AASG is given in section 5.2, together with the definition of other concepts needed to build AASGs, such as *abstract events* or the operations to apply

for matching real events. In section 5.3, we show an example of the steps followed to build an AASG from a detected multi-step attack. Finally, in section 5.4, we present details about how to code an AASG to be implemented in a real system.

5.1 From Concrete to Abstract ASG

The originality of AASGs mainly lies in the idea of flexibly representing alternative multi-step attack scenarios in the same structure, based on the similarity features defined in section 3.4.1, adapted to the needs of the security analyst. Before explaining the details of the AASG model, we have to describe its conception, from the attack scenario graphs used in the literature (CASG) to the definitive idea of AASG. We start exploring in section 5.1.1 the existent mechanisms for representing alternative hypotheses in a graph. A list of requirements derived from the reviewed literature to make the AASG a valid model to respond to our research question is presented in section 5.1.2. AASGs are inspired by two sources: the models for representation of alternative hypotheses, reviewed in section 5.1.1, page 104, and the mechanism of rule-based event correlation used in SIEMs (see page 19). In section 5.1.3, we explain how the idea of rule-based log correlation has influenced the creation of AASGs, before presenting in section 5.1.4 how alternative scenarios can be presented in the same structure. Finally, in section 5.1.5, we clarify the difference between CASGs and AASGs.

5.1.1 Representation of alternative hypotheses

The manifestation of a multi-step attack from the point of view of the defender is a sequence of traces, as we said in section 3.3.1, page 36. The hypotheses formulated by the analyst should then be identifiable with an eventual set of traces left by the attacker. The concept of alternative hypotheses has been already addressed in the literature about multi-step attack detection. However, these hypotheses are not considered as proposals coming from the human analyst in his quest of detection models, but as stated by the own detection methods. The hypotheses are then alternative choices created by the system from the information it has, for presenting them to the analyst [Mathew 2010] or as an intermediary step of the detection process [Skopik 2014]. They refer to one of the following four elements:

- Alternative goals or root causes of an attack [Geib 2001, Julisch 2001].
- The correlation between traces [Cuppens 2002c, Ren 2010, Friedberg 2015]. Rules

are tentatively proposed by the system but they need to be confirmed.

- The presence of missing steps [Cuppens 2002b, Ning 2004b, Zhai 2006], if the expected sequence of events is incoherent with the observations.
- The whole attack scenario [Mathew 2010], as it could be a false positive.

As far as we know, the only publication in the selected corpus about multi-step attack detection that addresses alternative hypotheses proposed by the analyst is the one where the system SLEUTH [Hossain 2017] is proposed. But hypothesis decision is not fully integrated in SLEUTH. Its authors just propose to reclassify the data according to a different hypothesis and to "re-run the analysis".

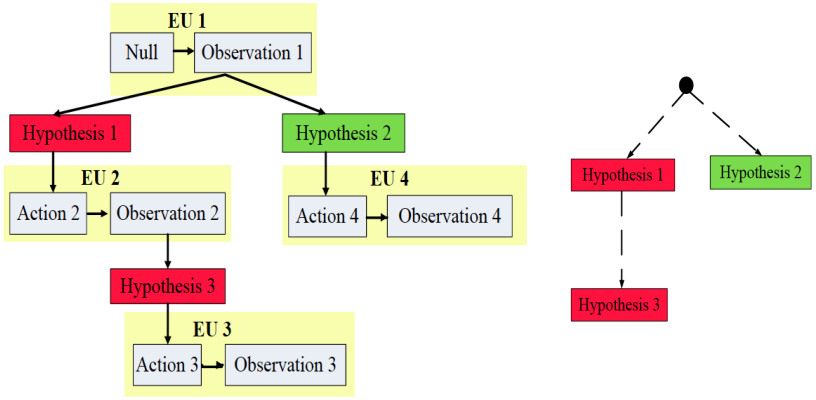
In the literature about attack graphs, that is out of the scope of the systematic survey presented in Chapter 4, the hypotheses proposed by the analyst are more present. These hypotheses correspond to the different paths that a multi-step attack can take in a network according to structural data. Much work has been done about attack graphs [Sheyner 2002, Shostack 2014, Singhal 2017], which constitutes the base of the structural-based detection methods (section 4.3.3). However, hypotheses made on an attack graph consider only the structural information of the network and not the sequence of traces representing the attack. What interests us to address our research question is the proposal of hypotheses from an evidence set of traces, not from the topology and characteristics of the defended network.

Models for the representation of hypotheses can be found in other domains of Cybersecurity. For example, Yen et al. [Yen 2010] propose a hypothesis reasoning framework for cyber situation awareness. Their objective is to translate the hypotheses created in the "mental world" of the analyst to a specific platform allowing team collaboration. The framework is based on recognition-primed decision (RPD), a model of rapid decision making. The authors do not specify any formal detail of the model representing the hypotheses, but they give a very interesting list of requirements that such a model should meet:

1. **Representativeness**. The model should be able to represent all the important elements of the hypotheses.

2. **Simplicity and intuitiveness**. The analyst should be capable of building an instance of the model in a simple way and to intuitively interpret instances created by other analysts or the ones she could have created in the past.

3. **Expressed in mathematical form**. Hypotheses in the model should be expressed in a mathematical-based language and the model itself should be a math-



(a) E-Tree (b) H-Tree

Figure 5.1: Trees in the AOH model. Image adapted from [Zhong 2013].

ematical object. The purpose of this requirement is to be able to reduce the work of the analyst by an efficient analysis using mathematical-based tools.

4. **Computation friendliness**. The model and the hypotheses represented on it should be manageable by a computer system, that could assist the human analyst.

Yen et al. argue that traditional graphs cannot meet these requirements and that hypothesis models have to be represented as hypergraphs. They say so due to the broad scope of the problem they tackle: to represent any human hypothesis related to cyber security awareness. We will see in Chapter 5 that if we restraint the hypotheses to sequences of events representing multi-step attacks, models based on traditional graphs are perfectly possible.

The AOH (Action, Observation and Hypothesis) model is another reasoning system intended to capture the know-how of expert analysts to guide unexperienced ones [Zhong 2013, Zhong 2014, Zhong 2015]. The experience-based reasoning process is expressed in a model called E-Tree, where actions are related to observations in the frame of a working hypothesis. A representation of an E-Tree is shown in Figure 5.1a. Each relationship action-observation is called an Experience Unit (EU). The disjunctive hypotheses in the E-Tree can be extracted into another model, the H-Tree (Figure 5.1b), so the analyst can just focus on them.

This model has evolved since it was first published. The idea of extracting the H-Tree has been abandoned and the E-Tree has become the AOH model itself. It has been used for other purposes: the identification of multimedia data that is interesting for the cybersecurity analysis [Alnusair 2017], the triage of security data coming to

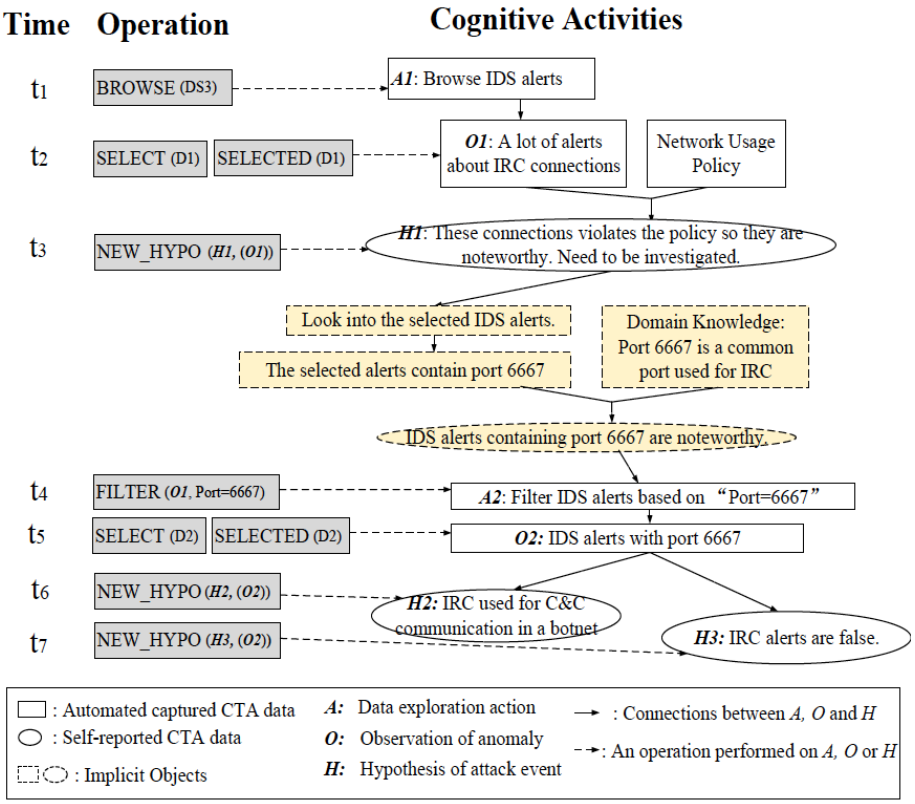


Figure 5.2: Example of the AOH model for capturing the cognitive process of the analyst in the investigation of an abnormal quantity of IRC alerts. Image adapted from one of the publications by Zhong et al. [Zhong 2015].

a Security Operations Center (SOC) [Zhong 2017] and the sharing of the processes followed by the analyst [Thomas 2018]. All these proposals, always with Chen Zhong as the author in common, are based on capturing the activities performed by the analyst in the form of cognitive traces[1]. Both actions (e.g. browse a log file) and observations (e.g. a lot of alerts representing an IRC connection) generate traces that can be automatically incorporated into the AOH model. Actions become traces thanks to a defined catalogue. On their side, observations become traces through the selection of data performed by the analyst. However, the model does not provide a format to represent the hypotheses, that are just described in plain text. One of the examples proposed by Zhong et al. [Zhong 2015] is represented in Figure 5.2.

We do not want to end up this section without addressing the *issue trees*, also called *logic trees*. They were created to simplify the resolution of issues by logic reasoning [Wojick 1975]. They are applied in business decision taking. There are not many scientific publications analyzing them but they are very similar in structure

[1]Not to be confounded with the traces considered in this thesis (see section 2.3)

to our AASG model presented in Chapter 5. In an issue tree, nodes represent a brief statement or question about an issue. The moves down the tree are made by responding to the parent node, getting to other nodes with new statements or questions. Paths in the tree represent a line of reasoning. An example of an issue tree representing a discussion about the appearance of the stirrup in History as the origin of Feudalism [Wojick 1975] is shown in Figure 5.3.

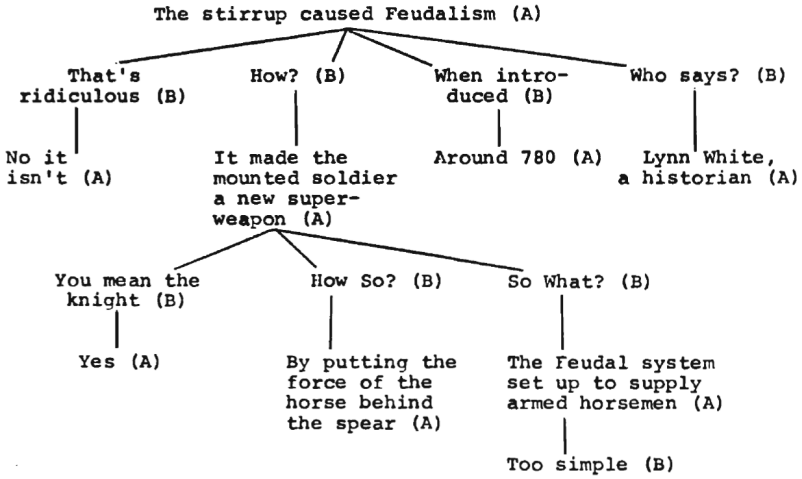


Figure 5.3: Example of an issue tree representing the discussion about the creation of the stirrup marking the beginning of Feudalism [Wojick 1975].

5.1.2 Requirements to define the model

We need to define a series of requirements that an AASG needs to fulfill to address the research question. Before doing so, the object with which an AASG will be dealing has to be defined. Said differently, as multi-step attack cases are conceived as sequences of traces, we should determine which type of trace is used (see section 2.3). An AASG is built only considering *events*, including alerts, being other types of trace, such as packets, excluded from the definition presented in this thesis.

The reason of this choice is that events are much more suitable than other traces to the proposition of alternative multi-step attack cases. They are concise in the expression of the actions and therefore better readable by a human. If we take the case of packets, for example, their design is conceived to be optimal for communications between computers and their content is difficult to read for a human. On top of that, there is an enormous quantity of them in a network. Nassar et al. [Nassar 2013] give an example of a university network with an average traffic load of 650 Mbps and maximum peaks of 1 Gbps that creates several hundred million network flows per day,

each flow representing a full communication session between two machines, so therefore composed of several packets.

On the contrary, the textual representation of an event, the log, has its origin in the need for stocking information that can be read by humans. They are created to keep a track of actions and to make the analyst understand what is happening in the network. Their volume is also lower than the volume of packets, because only the most relevant actions are chosen by each asset to be logged. Moreover, the most important datasets considered for multi-step attack detection contain events or they have to be expressed as IDS alerts, that are also events, to be used (see section 7.1, page 169).

Once it is decided that an AASG works with events, the first requirements to fulfill are the ones defined by Yen et al. [Yen 2010] for a model representing hypotheses for cyber situation awareness. These requirements have been listed in section 5.1.1, page 105, and adapted here for our specific problem:

1. **Representativeness**. The design of the model should be able to capture the hypotheses of the analyst about multi-step attacks in terms of sequences of traces.

2. **Simplicity and intuitiveness**. The model has to be human readable, so the analyst can directly represent the conceived alternative scenarios.

3. **Mathematical form**[2]. Concepts in the multi-step attack cases have to be expressed in a mathematical language. The structure of the model itself should be the one of a mathematical object, and a set of mathematical operations should be defined for working with traces.

4. **Computation friendliness**. The model needs to have a format that can be used by computers, so we can develop algorithms using this representation.

Apart from these requirements, we propose a list of five additional ones, referring to the purpose and the functionality of the model:

5. **Compliance with the purpose**. The model should be adapted both for helping the decision between alternative attack scenarios (identification) and for detecting attack scenarios.

6. **Boundness**. The end of each represented attack needs to be well defined.

7. **Uniqueness**. A criterion needs to be defined to univocally distinguish one model among the others in a set.

[2]We change the name to also make it a noun, like the others.

8. **Flexibility**. The design of the model should be flexible enough to allow the representation of varied attack scenario.

9. **Suitability for learning**. A quantification of the probabilities of each subsequent step has to be permitted, to allow the application of learning algorithms.

We will see through this chapter how the AASG fulfills all the presented requisites.

5.1.3 Rule-based event correlation

SIEMs, the security systems for log collection and correlation (see page 19), generally apply *rule-based reasoning* [Müller 2009], which is based on the identification of matches between collected events and static correlation rules. The rules represent a set of conditions to be met by a set of events for raising an alert and they are written by security experts. They can be general or adapted to the characteristics of the network where the monitored devices are placed. Many commercial SIEMs base their detection mechanisms on rules furnished by the vendor [Oliver Rochford 2016] in combination with specific rules crafted by the security analysts in the organization.

The set of conditions represented in a rule can follow a specific order or not. For example, imagine a set of three conditions: "find event A", "find event B" and "find event C". We can define a rule looking for the fulfillment of the three conditions in any order: "find event A, B and C in a certain time window W_T". This is what we call a *non-ordered rule*. But we can also define a rule where the order is important, an *ordered rule*: "find event A and then event B and then event C in a certain time window W_T". The specification of a time window W_T limits the time during which the search of conditions take place. This is necessary to limit the usage of computational resources in the SIEM. Henceforth, we only consider ordered rules, which better fit the idea of a multi-step attack as a sequence of actions.

In a SIEM applying rule-based reasoning on ordered rules, there is a linear check of each incoming event against the beginning of each rule. If an event matches the first condition of a rule, a new detection thread is opened to wait for other events matching the rest of the conditions in that rule. Different rules have forcibly some points in common. In that case, the search of following events is done concurrently for all the rules whose first conditions are matched so far. This would be the case of the LLDoS 1.0 and LLDoS 2.0.2 attacks, contained in the dataset DARPA 2000 (section 3.2.2, page 31). The second step of LLDoS 2.0.2 is the same as the third step in LLDoS 1.0, and the two last steps also involve the same actions in both attacks.

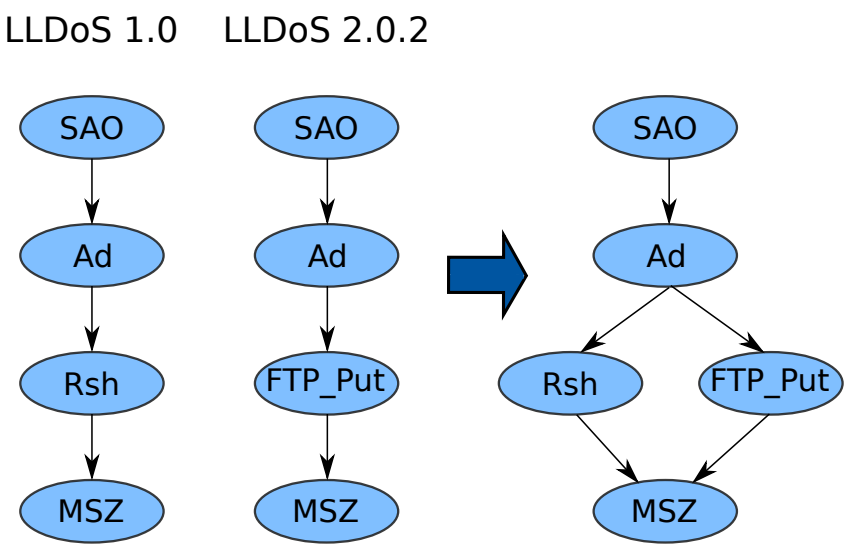


Figure 5.4: Combination in a graph of two detection rules for DARPA 2000

The conditions in an ordered rule can be represented in the form of a graph, as a sequence of nodes. In Figure 5.4, we represent two sequences of conditions characterizing the steps 3 and 4 of LLDoS 1.0 and the steps 2, 3 and 4 of LLDoS 2.0.2. The conditions are expressed as the search for a RealSecure[3] IDS alert of certain type in the set of incoming events. We use only the type of alert in each step to keep it simple and because it serves well to the purpose of the explanation. Nevertheless, rules generally contain other conditions based on other event attributes, such as the IP addresses or the port numbers.

What is interesting is that we can easily merge the two rules together in only one graph, such as the one shown in the right side of the figure. The concurrent search of alerts is then done as follows:

1. If the system finds an alert of type 'Sadmind_Amslverify_Overflow', a new detection thread is opened to search for alerts matching the subsequent conditions.

2. The system searches for an alert of type 'Admind'. So far the process for finding the two attacks is the same.

3. The detection thread looks for an alert of type 'FTP_Put' or one of type 'Rsh'. Depending on which of the two alerts is found the match corresponds to LLDoS

[3]RealSecure is a brand of IDS, used by many authors in the extraction of alerts in DARPA 2000 [Ning 2004a, Zhu 2006, Yu 2007, Liu 2008, Sadoddin 2009, Farhadi 2011, Anbarestani 2012, Kavousi 2014, Ramaki 2015a, Faraji Daneshgar 2016] (see section 7.1.1)

1.0 or LLDoS 2.0.2.

4. Independently of which is the path taken in the graph, both rules finish with a condition referring to the same type of alert ('MStream_Zombie').

The SIEM finally triggers an alert if all the conditions in a rule are matched or, in other words, when the end of a graph representing a rule is reached. Again, if a rule is not completely matched within a certain time window W_T, the detection thread in charge of analyzing this rule is closed. This classical method is used by many correlation systems, such as OSSIM[4] (Open Source SIEM), whose open code has helped to widely disseminate the idea of security event correlation [Alien Vault 2014].

The structure of AASGs and the mechanism to use them is directly inspired by ordered rules in rule-based event correlation. There are at least two reasons to take them as a reference. First, the expression of rules in the form of statements 'if ... then' is very intuitive for the human security analyst. At the same time, the possibility of arranging the rules in a graph is suitable for their use by automatic algorithms, as it is done in SIEMs. The requirements of simplicity and intuitiveness, and of computation friendliness (numbers 2 and 4, respectively, in the list defined in section 5.1.2) are thus met by the graphs for rule-based event correlation. We take therefore this model as the reference on top of which we append additional features to meet the rest of the defined conditions and fully define the AASG.

5.1.4 Defining alternative attack scenarios

Once we know that an AASG could have the same structure as a graph representing rule-based event correlation, we focus on how the alternative multi-step attack cases could be represented in the graph. A big challenge for the characterization of multi-step attacks is that we usually have only one instance of them. A security analyst is then confronted to derive a general model from just this single occurrence. This is not an easy task, as the sequence of actions representing a multi-step attack can be hidden among other actions, some of them possibly performed by the same actors. The concept of AASG is born from a need of arranging the set of hypotheses made by the expert in a unified structure.

Continuing with the example given in page 3, we can imagine that the security analyst considers several hypotheses when facing an instance of WannaCry[5], after an

[4] https://www.alienvault.com/products/ossim

[5] We imagine that the example is built on one of the first instances of WannaCry, before its details were disclosed to the public

infected computer is detected. The list of involved events was shown in Table 1.1, also reproduced in Figure 5.5. First, the analyst chooses the event e_0 because it represents the first action made by the malware. She knows that an unsuccessful HTTP request to a long domain name D_1, which is the event represented in e_0, is used by many pieces of malware to check if they are executed in a sandbox that tries to fool them giving a positive response to any request. But she is not sure if the event e_1, a successful HTTP request to another domain name D_2, could be a step or not of the attack. Maybe the response coming from D_2 was containing an additional piece of malware to continue the infection process. Then, she considers than the event e_2 is one of the step of the attack because it represents a connection in the port 445, used by SMB, which seems to be the protocol used for the propagation. As she is not an expert in the SMB protocol, and she is facing one of the first instances of WannaCry, she does not know if the propagation of the malware is performed through the malformed headers in the 'NEGOTIATE PROTOCOL REQUEST' packet (e_3) or it is done through the command 'transaction2_secondary' (e_6). The final result is a list of four hypothesis or alternative scenarios: $e_0 - e_2 - e_6$, $e_0 - e_2 - e_3$, $e_0 - e_1 - e_2 - e_6$ and $e_0 - e_1 - e_2 - e_3$. As we saw in page 3, only the first scenario represents the set of events to track for detecting an occurrence of WannaCry.

Event	Asset	Time	Description
e_0	Proxy	09:10:34	Unsuccessful HTTP request from a host h_A to long domain name D_1
e_1	Proxy	09:10:42	Successful HTTP request to long domain name D_2
e_2	Firewall	09:10:54	Successful connection in port TCP 445 from h_A to h_B
e_3	Endpoint h_B	09:11:04	Malformed SMB headers for the NEGOTIATE PROTOCOL REQUEST from h_A to h_B
e_4	Proxy	09:11:28	Successful HTTP request to long domain name D_2
e_5	IDS	09:11:40	SQL injection alert coming from h_A
e_6	Firewall	09:12:38	SMBv1 communication between h_A and h_B using command 'transaction2_secondary'

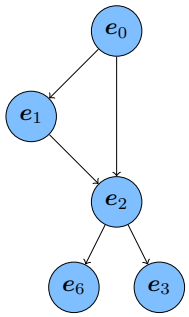


Figure 5.5: Example of alternative case derivation from WannaCry.

If we adopt the same perspective as in rule-based event correlation, these four alternative scenarios can be arranged in a graph all together, as they share several steps. Alternative scenarios considered by the security analyst as the representation of a multi-step attack generally have one or more steps in common. The common steps correspond to those ones that take an evident part in the development of the attack under the point of view of the analyst. We can see the resulting graph on the right side of Figure 5.5. This is the definitive shape of an AASG, for which we will give a full definition in section 5.2. Once done so, it will become clear how this structure can

fulfill the list of requirements defined in section 5.1.2.

5.1.5 Comparing CASG and AASG

Before formally defining an AASG, we present a list of the differences between CASGs and AASGs. The list is not exhaustive but it reflects how these two models are very different despite of the fact that share a similar structure:

- A CASG is a model of a single multi-step attack, built from a set of involved traces. Conversely, an AASG represents the alternative attack scenarios conceived by a security analyst.
- In a CASG, the nodes are traces or abstraction of traces (with not all the attributes represented) corresponding to each of the steps in the attack. In an AASG, the nodes represent a set of conditions defined by the analyst. These conditions indicate which events correspond to each of the nodes.
- An AASG can only have one source, while a CASG can have many.
- Similarity functions in a CASG are used to define the links between the different steps of the attack. On the contrary, similarity functions in an AASG are used to define the conditions under which incoming events match the nodes in the graph. They can be applied using predefined values in the model or the values of the attributes of previously found events (see sections 5.2.4 and 5.2.5).

5.2 Definition of AASG

Once the concept of AASG is introduced, we give in this section a rigorous definition of this model. As any directed graph, an AASG is composed of nodes and arcs. The nodes point out to a set of incoming events through a series of conditions, codified in what we call *abstract events*, which are defined in section 5.2.1. Alternative multi-step attack cases are represented as sequences of nodes joined by the arcs. Their structure and the rest of the formal characteristics of the AASG are explained in section 5.2.2. The mechanism allowing the match between real events and the abstract events in the nodes is presented in section 5.2.3. This matching process is based on two types of condition contained in the abstract events: *absolute* and *relative*. They are described in section 5.2.4 and 5.2.5, respectively. Finally, two additional elements of an AASG, counters and optional nodes, are introduced in section 5.2.6

5.2.1 Abstract events

As we said, Abstract Attack Scenario Graphs are conceived to work with events (including alerts), and not with other types of trace such as packets. A formal definition of an event can be given adapting the one proposed for a trace (see page 18). An *event* $\boldsymbol{e}$ is the record of an action happening in a certain asset. It is an entity in the *set of all possible events* $\mathbb{E}$. As any trace, it contains a series of inherent attributes that can be arranged in a finite tuple $\boldsymbol{e}_a = \{m_a, l_a, \ldots, n_a, \ldots\}$[6]. This characterization is independent of how the inherent attributes are textually represented in the log. The same as in any trace, the name n of each attribute n_a defines the type of value of the attribute and its meaning: an IP address, a port number, etc.

The concept of *abstract event* lies in the base of an AASG. We define an abstract event $\boldsymbol{e}^*$ as a set of K conditions $\{c_1, c_2, \ldots, c_K\}$ met by a subset of events in $\mathbb{E}$. For example, a condition c_1 could be to have the attribute indicating the destination port number (named *pdst*) equal to 23. An abstract event $\boldsymbol{e}_a^*$ with only this condition would refer to the subset of events having this destination port number. Conditions in an abstract event allow thus the selection of a set of events without specifying the concrete events within the selection, as filters in a search engine.

Apart from the condition of equality seen in the example ("having X value in an attribute"), we can define other types of condition, such as numerical intervals (e.g. process ID higher than 650, for events coming from a host), discrete ranges (e.g. *ipdst* in the IP address range 175.68.22.0/24), sets of values (e.g. destination port *pdst* is 80 or 443) or complementary definitions (e.g. logged user is *not* 'admin'). Conditions can also be built using the comparison features defined in section 3.4, that are based on how nodes in attack scenario graphs are linked in the literature. We will come back to how conditions can be built in section 5.2.3.

The *set of all possible abstract events* $\boldsymbol{e}^*$ is denoted by $\mathbb{E}^*$. Each event can be represented by many abstract events. The set of all possible abstract events corresponding to an event $\boldsymbol{e}_a$ is denoted by E_a^*.

5.2.2 Nodes and arcs

We can define an AASG using abstract events and considering matching operations against incoming events. Nodes in an AASG represent as abstract events the steps of

[6]We write an event $\boldsymbol{e}$ in bold characters to distinguish it from its individual components, not written in bold, adopting the same notation used for mathematical vectors. However, in cases where this ambiguity does not exist, we write e, not using the bold font. We adopt the same formalism for abstract events $\boldsymbol{e}^*$.

attack scenarios, while arcs indicate the order in which the events matching the nodes are arranged to correspond to an attack sequence, always starting from a single initial node. We have thus three properties the graph has to satisfy: it has to be directed, acyclic and single-source.

Regarding the first property, an AASG is directed because the alternative attack scenarios it represents are defined with a certain chronological order. If the order of the events in a multi-step attack is always preserved, it can be a sign of causality between the events: one event follows another one because it depends on its previous occurrence. Because of this, it is important to the analyst to be able to represent her hypotheses about the temporal succession of steps in the attack. This is possible because we assume that the timestamps in the logs are synchronized (see 36).

The need for having an acyclic graph is related to the requirement of having a well-defined end for each alternative scenario (requirement 6 of boundness in section 5.1.2). If there are no cycles, an algorithm using AASGs can define the arrival to a node with no arcs going out from it as the condition to stop the search of events. The scenario would then be considered as found. Allowing cycles leads to the possibility of defining infinitely long scenarios, which are not realistic from a practical point of view.

Finally, AASG are single-source (see page 42) because we need a way to identify a particular AASG among many others. In a system exploiting this structure we could have several AASGs containing alternative cases for different multi-step attacks at the same time. The security analyst can be simultaneously investigating several multi-step attacks or using several AASGs to perform detection. We choose to force an AASG to have a single *source* representing the first element in the sequence of all the scenarios proposed in a particular AASG. This mechanism is the same as the one applied in rule-based event correlation (see Figure 5.4). The source of an AASG is called *root node*. Each AASG is unambiguously defined by its root node, which fulfills the requirement of uniqueness, the number 7 in the list presented in section 5.1.2. Any newly proposed alternative attack scenario starting by a node being the root of an existent AASG should be appended to it.

We have then that an AASG is a single-source directed acyclic graph (DAG). Moreover, we add the condition that all the nodes except the root node should have an in-degree of at least 1. Doing so, we guarantee that all the nodes contained in an AASG have an arc to or from other nodes. With all these elements, we can present a formal definition of an AASG:

Definition 6.1.
An **Abstract Attack Scenario Graph (AASG)** $\psi = (K, A)$ is a single-source directed acyclic graph (DAG) represented by an ensemble of N_κ nodes $K = \{\kappa_0, \kappa_1, \ldots, \kappa_{N_\kappa}\}$ and an ensemble of arcs A, with each arc $\kappa_n\kappa_m$ named after the juxtaposition of the names of its start node κ_n and its end node κ_m, with $\kappa_n\kappa_m \neq \kappa_m\kappa_n$. ψ has the following properties:

1. Each node κ_n contains an abstract event $\boldsymbol{e}^*_n$.
2. κ_0 is the root node and it is the source of the graph.
3. $\forall \kappa_n \in K | \kappa_n \neq \kappa_0$, in-degree of $\kappa_n \geq 1$
4. Any possible path $[\kappa_0, \kappa_0\kappa_p, \kappa_p, \ldots, \kappa_m, \kappa_m\kappa_n, \kappa_n]$ from the root node κ_0 to a specific node κ_n has the same length.

The set of N_ψ AASGs used by a system is $\Psi = \{\psi_1, \psi_2, \ldots, \psi_{N_\psi}\}$. The system can then select which ψ_i to use based on its root node.

To complete the definition of AASG we have to introduce several concepts related to the path from the root node (the source) to one of the sinks (a node with an out-degree of 0):

- **Definition 6.2:** A **branch** $\boldsymbol{b}_\kappa$ is a path $[\kappa_0, \kappa_0\kappa_l, \kappa_l, \ldots, \kappa_j, \kappa_j\kappa_k, \kappa_k]$ that goes from the root node κ_0 to any of the sinks. It corresponds to one of the alternative attack scenarios represented in the AASG. The set of all the branches in an AASG ψ is denoted by B_ψ. We refer to each branch by the ordered sequence of the numbers corresponding to each node. For example, branch $[\kappa_0, \kappa_0\kappa_2, \kappa_2, \kappa_2\kappa_6, \kappa_6]$ would be expressed as $[0, 2, 6]$.
- **Definition 6.3:** The node κ_b is a **child** of node κ_a if there exists an arc $\kappa_a\kappa_b$ with its origin in κ_a and its end in κ_b.
- **Definition 6.4:** The node κ_a is a **parent** of node κ_b if there exists an arc $\kappa_a\kappa_b$ with its origin in κ_a and its end in κ_b.
- **Definition 6.5:** All the nodes in the AASG with a parent κ_a in common are called **$\boldsymbol{\kappa_a}$-siblings**, or just **siblings** to abbreviate.
- **Definition 6.6:** The **depth** of a node κ_n is the number of arcs that exists in the path between the root node κ_0 and κ_n. Due to the fourth property of an AASG, the depth of a node is unique, having all the paths starting from the root node and leading to the node the same number of arcs.

5.2.3 Matching events to nodes

We saw in section 5.2.1 that abstract events are designed to select a subset in a set of events. The subset of events selected by an abstract event is composed of the events fulfilling the conditions expressed in the abstract event. For each event $\boldsymbol{e}_a$ meeting all the conditions in abstract event $\boldsymbol{e}_n^*$ we say that "$\boldsymbol{e}_a$ *matches* $\boldsymbol{e}_n^*$" or that "$\boldsymbol{e}_n^*$ *is matched* by $\boldsymbol{e}_a$".

We can define a *match function* M to express the operation of checking if an event $\boldsymbol{e}_a$ matches $\boldsymbol{e}_n^*$. This function can be extended to work with conditions and nodes.

Definition 6.7.
Given an event $\boldsymbol{e}_a$, a node in an AASG κ_n, an abstract event $\boldsymbol{e}_n^* \in \kappa_n$ and a condition $c_k \in \boldsymbol{e}_n^*$ we can define a **match function** M in three different domains:

- Between $\boldsymbol{e}_a$ and c_k:

$$M(\boldsymbol{e}_a, c_k) = \begin{cases} 1, & \text{if } \boldsymbol{e}_a \textit{ meets } c_k \\ 0, & \text{otherwise} \end{cases} \tag{5.1}$$

- Between $\boldsymbol{e}_a$ and $\boldsymbol{e}_n^*$:

$$M(\boldsymbol{e}_a, \boldsymbol{e}_n^*) = \begin{cases} 1, & \text{if } M(\boldsymbol{e}_a, c_l) = 1 \, \forall c_l \in \boldsymbol{e}_n^* \\ 0, & \text{otherwise} \end{cases} \tag{5.2}$$

  or differently said, $M(\boldsymbol{e}_a, \boldsymbol{e}_n^*) = 1$ if $\boldsymbol{e}_n^* \in E_a^*$.
- Between $\boldsymbol{e}_a$ and κ_n:

$$M(\boldsymbol{e}_a, \kappa_n) = \begin{cases} 1, & \text{if } M(\boldsymbol{e}_a, \boldsymbol{e}_n^*) = 1 \\ 0, & \text{otherwise} \end{cases} \tag{5.3}$$

The condition c_k is the basic brick to build abstract events and to define the events that match an AASG. There can exist two kinds of condition in an abstract event:

- **Absolute conditions**, only based on the inherent attributes of each individual event. All the examples we have seen so far in this section use absolute conditions. Another example of absolute condition is "the type of the event is textually similar to the 'fail login' string".
- **Relative conditions**, that refer to the previous event in the sequence, the one that matches the parent node. For example, "the source IP address is the same as the destination IP address of the previous event" is a relative condition.

Absolute and relative conditions can be combined when defining an abstract event. They are defined in the following two sections (5.2.4 and 5.2.5, respectively) and listed

in Table E.1. In any case, both absolute and relative conditions have certain common characteristics:

- They are associated to a function g, which returns the value 0 or 1 depending on the inherent attributes of the event $\boldsymbol{e}_a$.
- In some cases, they have an associated threshold λ to conform the result of an internal comparison function to the value 0 or 1 returned by g. In these cases, λ determines the value of the internal comparison function above which the condition is met ($g = 1$).

Many of the functions g that we have defined for the conditions are directly adopted from the set of atomic similarity functions used in the literature (see Table 3.3 in 52). In the literature on multi-step attack detection they are used to define a model of the attack by linking the set of traces corresponding to the different steps. However, the application is radically different in AASG: the search for incoming events matching an abstract model derived from an instance of a multi-step attack but not representing the instance itself, as several alternative scenarios appear together in the same AASG.

Type	**Name of function**	**Formula**
Absolute conditions	Equality	$g_1(n_a, r) = \begin{cases} 1, & \text{if } n_a = r \\ 0, & \text{otherwise} \end{cases}$
	Inequality	$g_2(n_a, r) = \begin{cases} 1, & \text{if } n_a \neq r \\ 0, & \text{otherwise} \end{cases}$
	Prefix similarity	$g_3(n_a, r, \lambda) = \begin{cases} 1, & \text{if } \frac{l}{L} > \lambda \\ 0, & \text{otherwise} \end{cases}$
	Textual similarity	$g_4(n_a, r, \lambda) = \begin{cases} 1, & \text{if } jac(n_a, r) > \lambda \\ 0, & \text{otherwise} \end{cases}$
	Set-based	$g_5(n_a, R) = \begin{cases} 1, & \text{if } n_a \in R \\ 0, & \text{otherwise} \end{cases}$
Relative conditions	Equality	$f_1(n_a, m_b) = \begin{cases} 1, & \text{if } n_a = m_b \\ 0, & \text{otherwise} \end{cases}$
	Common element	$f_2(N_a, M_b) = \begin{cases} 1, & \text{if } N_a \cap M_b \neq \varnothing \\ 0, & \text{otherwise} \end{cases}$
	Prefix similarity	$f_3^*(n_a, m_b, \lambda) = \begin{cases} 1, & \text{if } \frac{l}{L} > \lambda \\ 0, & \text{otherwise} \end{cases}$
	Textual similarity	$f_4^*(n_a, m_b, \lambda) = \begin{cases} 1, & \text{if } jac(n_a, m_b) > \lambda \\ 0, & \text{otherwise} \end{cases}$
	Inequality	$f_{neq}(n_a, m_p) = \begin{cases} 1, & \text{if } n_a \neq m_p \\ 0, & \text{otherwise} \end{cases}$

Table 5.1: Functions used in the definition of the conditions in abstract events.

5.2.4 Absolute conditions

An absolute matching condition always refers to a certain inherent attribute in the event. We refer to it with a superscript n (c_k^n) indicating the name of the considered attribute[7]. For example, a condition c_k^{pdst} depends on the destination port number, while a condition c_k^{ipsrc} depends on the source IP address.

The function g in an absolute condition takes as input the event $\boldsymbol{e}_a$ to be tested and a *reference* r, the fixed element against which the test is made. We define two types of absolute condition, depending on which is the kind of data taken as a reference: *value-based* and *set-based*. We explain below the characteristics of each one:

- **Value-based absolute conditions**. The reference r is a single value, whose nature depends on the inherent attribute considered by the condition. For instance, in the case of taking the *type* of the event, r would be a string of characters. In other cases, it could be an IP address, a port number or a numerical value.
  The following functions are defined:
  - **Equality**. The same as f_1 in Table 3.3 (Equation 3.1, page 48), this function checks if the selected attribute is equal to r:

$$g_1(n_a, r) = \begin{cases} 1, & \text{if } n_a = r \\ 0, & \text{otherwise} \end{cases} \tag{5.4}$$

  - **Inequality**. It is the exact opposite of g_1: this function checks if the selected attribute is different from r:

$$g_2(n_a, r) = \begin{cases} 1, & \text{if } n_a \neq r \\ 0, & \text{otherwise} \end{cases} \tag{5.5}$$

  - **Prefix similarity**. The result of this function is determined by calculating f_3 (Equation 3.3, page 49) and comparing to a threshold λ to get a binary result. We use this function only for comparison between IP addresses. Remember that in f_3, l represents the number of common bits in the binary representations of the compared IP address n_a and r, starting the count from the left and stopping when there is an uncommon bit. L is the total number of bits in the binary representation, which in this case is 32.

[7] See Appendix C for the list of attribute names used in this thesis

$$g_3(n_a, r, \lambda) = \begin{cases} 1, & \text{if } \frac{l}{L} > \lambda \\ 0, & \text{otherwise} \end{cases} \tag{5.6}$$

– **Textual similarity**. Taking the idea from f_4 (Equation 3.4, page 50), we also incorporate a textual similarity function. We saw that Zhang et al. [Zhang 2015] use the Levenshtein distance as textual metric for comparing IP addresses. The Levenshtein distance works well if the two compared strings have a similar length, as it is the case of IP addresses. However, a generic text could have any length, and the Levenshtein distance is no longer valid. For example, imagine we want to use the string s_1 = "SQL" in our condition of matching and we want to calculate the textual distance against s_2 = "This is a SQL injection attack" and s_3 = "CSS attack". We would expect $lev(s_1, s_2) < lev(s_1, s_3)$, but using the Levenshtein distance we have that $lev(s_1, s_2) = 27$ and $lev(s_1, s_3) = 9$, just because s_2 is longer than s_3. Even normalizing by the maximum length of the two compared strings we have $\widetilde{lev}(s_1, s_2) \approx 0.87$ and $\widetilde{lev}(s_1, s_3) \approx 0.82$.

The example given above represents a typical case of application of textual similarity in an AASG: the analyst wants to find a type of event but she does not know exactly which are the set of text strings referring to this type, so she uses one or several keywords when defining the abstract event. Because of this, we choose another similarity function, the Jaccard index [Jaccard 1901], applied to the set of words in the text strings. If we name W_x and W_y the set of words in strings x and y, respectively, we can define the Jaccard index as:

$$jac(x, y) = \frac{|W_x \cap W_y|}{|W_x \cup W_y|} = \frac{|W_x \cap W_y|}{|W_x| + |W_y| - |W_x \cap W_y|} \tag{5.7}$$

The Jaccard index is proportional to the similarity between the strings, oppositely to the Levenshtein distance, that measures the difference instead of the similarity. Taking the previous example, we have that $jac(s_1, s_2) \approx 0.17$ and $jac(s_1, s_3) = 0$. Considering a threshold λ, we can then define a function based on the Jaccard index:

$$g_4(n_a, r, \lambda) = \begin{cases} 1, & \text{if } jac(n_a, r) > \lambda \\ 0, & \text{otherwise} \end{cases} \tag{5.8}$$

- **Set-based absolute conditions**. They do the comparison against a set of values, instead of against an individual value. r becomes then a set, which we call R. We define only one function of this type, that consists on checking if the value of the attribute is in the set R:

$$g_5(n_a, R) = \begin{cases} 1, & \text{if } n_a \in R \\ 0, & \text{otherwise} \end{cases} \tag{5.9}$$

5.2.5 Relative conditions

Relative conditions defined for the abstract event in a node κ_y refer to a parent κ_x of κ_y in the AASG. Each branch of an AASG represents a sequence ordered in time, so events matching κ_x always precedes those matching κ_y.

The function g defined for each relative condition takes as least two arguments. The first one is an attribute n_a or a set of attributes N_a from an incoming event $\boldsymbol{e}_a$. The second, an attribute m_p or a set of attributes M_p from the event $\boldsymbol{e}_p$ that has matched one of the parents of the node containing the abstract event with the relative condition.

We derive the matching functions directly from the atomic similarity functions considered in the literature and listed in Table 3.3, page 52. For f_1 and f_2, the identification of $M(\boldsymbol{e}_a, c_k)$ with the value returned by the function is direct, as they can just take values 0 and 1. However, functions such as f_3 or f_4, which returns a real value between 0 and 1, should be modified to have a binary result depending on a threshold λ. In an AASG, we are not interested in the degree of similarity between two traces but in knowing if an incoming event matches or does not match the node. The resulting modified functions f_3^* and f_4^* are shown in Table 3.3. For textual similarity (f_4) we use the Jaccard index as in g_4.

Additionally, we define another atomic similarity function to evaluate the inequality of an attribute between the events (similar to g_2):

$$f_{neq}(n_a, m_p) = \begin{cases} 1, & \text{if } n_a \neq m_p \\ 0, & \text{otherwise} \end{cases} \tag{5.10}$$

5.2.6 Additional elements

We end the definition of AASGs with two additional elements: counters and optional nodes. This two elements serve to simplify the definition of some kinds of scenario,

which could be anyways described just using the properties defined so far.

5.2.6.1 Counters

Sometimes, there are repetitions of actions in a multi-step attack. Think of an attack starting with a simple ping-based port scan. Supposing that the ping action is registered as an event in the firewall, we could define an abstract event to match it containing two absolute conditions with the following functions:

- $g_1(type_a, \text{'ping'})$
- $g_1(origin_a, \text{'firewall'})$

However, we cannot consider we are being scanned just because we detect one ping event. Several repetitions of the action against different ports are necessary to arrive at this conclusion. Each repetition would have the same target IP address and a different target port number[8]. We can define this as a sequence of nodes, where each abstract event has, apart from the already defined absolute conditions, two abstract conditions referring to the previous event $\boldsymbol{e}_p$ in the sequence:

- $f_1(ipdst_a, ipdst_p)$
- $f_{neq}(pdst_a, pdst_p)$

Given that the abstract event is the same for each node representing an individual ping action, we develop a mechanism, the *counters*, to indicate this repetition with just one node. A counter is a natural number associated to a node indicating how many events matching the abstract event in the node have to be found before considering the node as matched.

In a graphical representation of an AASG, counters are represented as self-loops. We show in Figure 5.7 both representations, before and after the implementation of a counter, for an AASG with a repetition of the same abstract event in subsequent nodes. Restraining counters as an element for representation, AASG do not still lose their acyclic character. Counters are just a formality to express differently a concept that could be implemented using the base definition of an AASG. However, we will see that when using algorithms such as Morwilog (Chapter 6), the self-loop represented by the counter is considered during the execution as being different than its alternative representation with all the nodes expanded. In this case, the AASG is not acyclic anymore and becomes a single-root directed graph where only self-loops are allowed.

[8]Notice that this rule considers that the attacker could change its IP address.

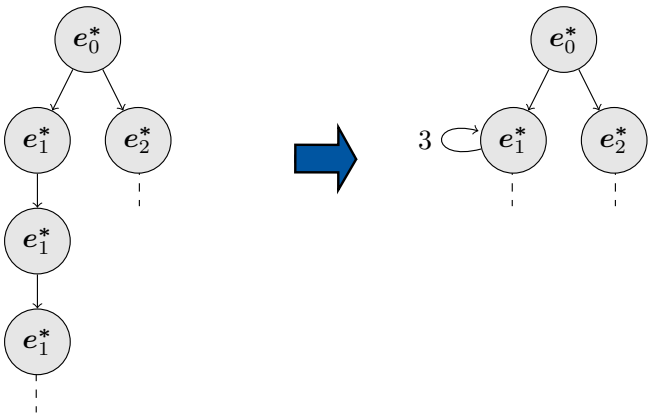


Figure 5.6: Example of the implementation of a counter in an AASG. The AASG on the right is the same as the one on the left but expressed with a counter.

5.2.6.2 Optional nodes

Another additional mechanism to define AASGs is the use of *optional nodes*. This idea is similar to the one of optional event used in the LAMBDA language [Cuppens 2000] (see section 3.3.2.1, page 37). An optional node is a node representing a step in the scenario that the analyst does not know if it takes part in the attack or not. Think of event e_1 represented in Figure 5.5 and mentioned in the example of page 112. It corresponds to a "successful HTTP request to long domain name D_2" in the set of logs containing the evidence of WannaCry attack. The analyst is not sure about its membership in the attack. We could then mark this event as 'optional' in our hypothesis.

Optional nodes are drawn with a dashed contour in the graphical representation of an AASG. Figure 5.7 shows the process of transforming several alternative scenarios into an AASG with two optional nodes, κ_2 and κ_5. The representation of the middle, without optional nodes, is totally equivalent. Again, this mechanism serves just to help the analyst in the definition of AASGs but it does not change the structure of the graph.

5.3 Creating Abstract Attack Scenario Graphs

Once the concept of AASG has been defined, we have yet to explain how this structure can be used by the security analyst. We do it by an example of AASG creation.

Imagine an instance of LLDoS 1.0, one of the multi-step attacks contained in the DARPA 2000 dataset (section 3.2.2). In Table 5.2 we have a list of alerts generated in

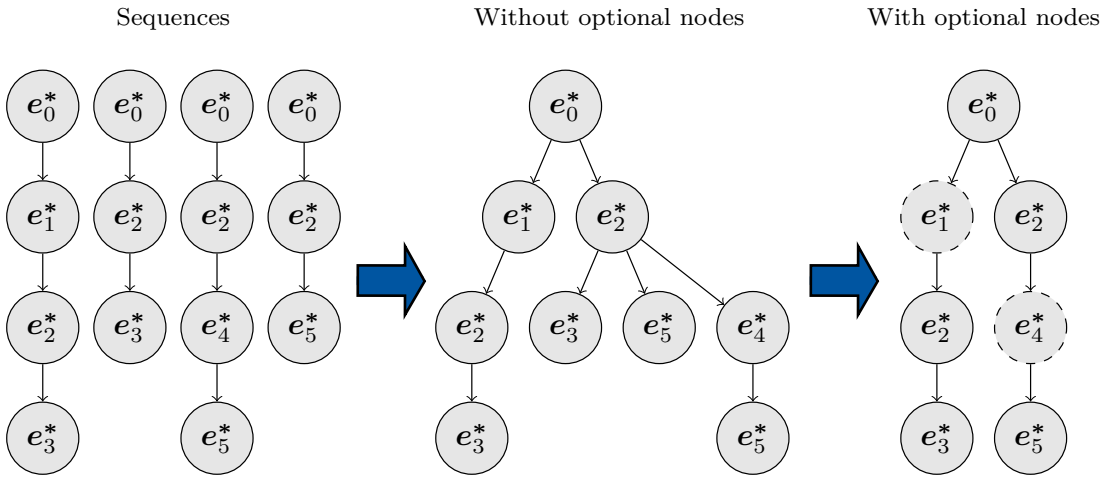


Figure 5.7: Example of the implementation of optional nodes in an AASG. On the left we have the alternative scenarios, on the center the AASG defined without optional nodes and on the right, the same AASG expressed with optional nodes.

the inside network during the infection (steps 2, 3 and 4) of one of the machines, called *Mill*, with IP address 172.16.115.20. The attacker has the IP address 202.77.162.213. In fact, the infection of Mill is by itself a multi-step attack, even if it is part of the biggest LLDoS 1.0 attack.

We display only seven fields: an identifier assigned to each alert; the timestamp; the type of the alert, and the IP address and port number for both the source and the destination. These alerts come from the dataset provided by Ning et al. [Ning 2002a], who generated them using the RealSecure IDS on the original captures of traffic in DARPA 2000. That is why the timestamps of the alerts do not correspond to the moment when the original dataset was generated (between March and April 2000 [Shittu 2016]). We will come back to this topic in section 7.1.1. The list of all the RealSecure alerts included in this thesis is presented in Table 7.1.

If it is the first time that the analyst faces this attack, it can be difficult to develop a signature representing it to detect future occurrences. An additional difficulty is the presence of alerts containing Mill's IP address but not representing any step in the attack, such as that one of type 'Email_Ehlo' (alert 67405). Here is an example of how the analyst could proceed to reflect her ideas about the multi-step attack in an AASG. The resulting AASG, called *Mill-1*, is represented in Figure 5.8 together with the conditions included within the abstract events of each node.

id	*time*	*ipsrc*	*psrc*	*ipdst*	*pdst*	*type*
67285	10/11 04:46:23	202.77.162.213	54792	172.16.115.20	32773	Admind
67286	10/11 04:46:23	202.77.162.213	54792	172.16.115.20	32773	Sadmind_Ping
67405	10/11 05:07:25	135.8.60.182	46847	172.16.115.20	25	Email_Ehlo
67415	10/11 05:11:27	202.77.162.213	60251	172.16.115.20	32773	Admind
67416	10/11 05:11:27	202.77.162.213	60251	172.16.115.20	32773	Sadmind_AO
67425	10/11 05:11:29	202.77.162.213	60255	172.16.115.20	32773	Admind
67426	10/11 05:11:29	202.77.162.213	60255	172.16.115.20	32773	Sadmind_AO
67422	10/11 05:11:35	202.77.162.213	60269	172.16.115.20	32773	Sadmind_AO
67427	10/11 05:11:35	202.77.162.213	60269	172.16.115.20	32773	Admind
67423	10/11 05:11:37	202.77.162.213	60276	172.16.115.20	32773	Admind
67424	10/11 05:11:37	202.77.162.213	60276	172.16.115.20	32773	Sadmind_AO
67419	10/11 05:11:43	202.77.162.213	60289	172.16.115.20	32773	Admind
67420	10/11 05:11:43	202.77.162.213	60289	172.16.115.20	32773	Sadmind_AO
67417	10/11 05:11:45	202.77.162.213	60300	172.16.115.20	32773	Sadmind_AO
67421	10/11 05:11:45	202.77.162.213	60300	172.16.115.20	32773	Admind
67488	10/11 05:21:29	197.182.91.233	47288	172.16.115.20	23	TelnetTermtype
67549	10/11 05:28:18	172.16.115.20	1023	202.77.162.213	514	Rsh
67550	10/11 05:28:18	172.16.115.20	1022	202.77.162.213	514	Rsh
67546	10/11 05:28:20	172.16.115.20	1022	202.77.162.213	514	Rsh
67547	10/11 05:28:20	172.16.115.20	1022	202.77.162.213	514	Rsh
67543	10/11 05:28:21	172.16.115.20	1021	202.77.162.213	514	Rsh
67545	10/11 05:28:22	172.16.115.20	1021	202.77.162.213	514	Rsh
67540	10/11 05:28:23	202.77.162.213	1023	172.16.115.20	514	Rsh
67537	10/11 05:28:37	172.16.115.20	33786	255.255.255.255	9325	Mstream_Zombie
67542	10/11 05:28:37	202.77.162.213	1023	172.16.115.20	514	Rsh
67728	10/11 05:59:11	195.73.151.50	49054	172.16.115.20	23	TelnetTermtype
67762	10/11 06:04:32	202.77.162.213	49212	172.16.115.20	23	TelnetTermtype
67763	10/11 06:04:32	202.77.162.213	49212	172.16.115.20	23	TelnetXdisplay
67764	10/11 06:04:32	202.77.162.213	49212	172.16.115.20	23	TelnetEnvAll
67767	10/11 06:05:25	172.16.115.20	33799	255.255.255.255	9325	Mstream_Zombie
67776	10/11 06:06:07	172.16.115.20	33800	172.16.112.50	7983	Mstream_Zombie
67777	10/11 06:06:07	172.16.115.20	33800	172.16.112.10	7983	Mstream_Zombie
67809	10/11 06:15:42	196.227.33.189	49619	172.16.115.20	23	TelnetTermtype
67918	10/11 06:46:15	194.7.248.153	51115	172.16.115.20	23	TelnetTermtype
67923	10/11 06:48:07	197.182.91.233	51220	172.16.115.20	23	TelnetTermtype
67977	10/11 06:59:53	196.37.75.158	51923	172.16.115.20	23	TelnetTermtype
67989	10/11 07:08:49	172.16.112.50	33488	172.16.115.20	23	TelnetTermtype

Table 5.2: Alerts from DARPA 2000 representing the infection of Mill

1. The analyst clearly sees that the attack started with a phase for checking if the *sadmind* daemon is installed in Mill (alert 'Sadmind_Ping', 67286). It is also clear for her that the culmination of the infection is the installation of the *mstream* daemon, whose communications are captured by the alerts of type 'Mstream_Zombie' (67537, 67767, 67776 and 67777). The AASG should then have the identification of 'Sadmind_Ping' in the first node (κ_0) and that of 'Mstream_Zombie' in the last one.

2. The 'Sadmind_Amslverify_Overflow' alert (abbreviated in the table as 'Sadmind_AO') represents an attempt of buffer overflow against the *sadmind* daemon. This is the attack through which the attacker can have access to the Mill machine. It is evident that each action involving the *sadmind* daemon ('Sadmind_Ping' or 'Sadmind_AO') also involves an 'Admind' alert. However, the order of the alerts seems to vary, sometimes having the 'Admind' alert preceding

the other alert and some others the opposite. This can be coded by the analyst using absolute and relative conditions, contained in abstract events $\boldsymbol{e}_1^*$ and $\boldsymbol{e}_3^*$.

3. As the alert 'Sadmind_AO' is repeated exactly 6 times, she considers this repetition as another possible indicator of the attack. Moreover, there is a 'TelnetTerminaltype' alert (abbreviated as 'TelnetTermtype') that she decides to incorporate to the AASG as the next step. These suppositions are reflected in $\boldsymbol{e}_2^*$ and $\boldsymbol{e}_4^*$.

4. After the buffer overflow attack, the attacker seems to get access to Mill using rsh (remote shell command). There are 'Rsh' alerts where the IP address of the attacker is in the attribute *ipsrc* (67540 and 67542), while there are others where it is in *ipdst* (e.g. 67549 or 67545). The analyst does not know if in a future occurrence of the attack the IDS will generate alerts of the two types, so she includes both options in the AASG ($\boldsymbol{e}_5^*$ and $\boldsymbol{e}_6^*$).

5. The analyst thinks that the presence of telnet communication after the 'Rsh' alerts could be part of the attack. However, it is not clear if it will be always the case or not, so she adds a node referring to 'TelnetTermtype' as an optional node in the AASG (κ_7).

6. Finally, the analyst identifies two different types of 'Mstream_Zombie' alert, one having as destination a specific IP address in the network (67776 and 67777) and the other being a broadcast message (67767 and 67537). Both possibilities are included in the AASG in separated nodes, coded by abstract events $\boldsymbol{e}_8^*$ and $\boldsymbol{e}_9^*$, so both could be detected in the future.

5.4 Implementation

Given that the structure of the AASG has been well defined in section 5.2, it is fairly straight to express it in a language that can be interpreted by a system using this model. An AASG is implemented in JavaScript Object Notation format (JSON) [IETF 2017]. The implementation captures all the defined elements in an AASG (abstract events, conditions, etc.), allowing at the same time the possibility of incorporating new elements to be used by algorithms exploiting AASGs (see Chapter 6).

Node	Abstract event	List of conditions
κ_0	$\boldsymbol{e}_0^*$	$g_1(type_a, \text{'Sadmind\_Ping'}) = 1$
κ_1	$\boldsymbol{e}_1^*$	$g_5(type_a, \{\text{'Sadmind\_AO'}, \text{'Admind'}\}) = 1$, $f_1(ipsrc_a, ipsrc_p) = 1$, $f_1(ipdst_a, ipdst_p) = 1$
κ_2	$\boldsymbol{e}_2^*$	$g_1(type_a, \text{'Sadmind\_AO'}) = 1$, $f_1(ipsrc_a, ipsrc_p) = 1$, $f_1(ipdst_a, ipdst_p) = 1$
κ_3	$\boldsymbol{e}_3^*$	$g_5(type_a, \{\text{'Sadmind\_AO'}, \text{'Admind'}\}) = 1$, $f_{neq}(type_a, type_p) = 1$, $f_1(ipsrc_a, ipsrc_p) = 1$, $f_1(ipdst_a, ipdst_p) = 1$
κ_4	$\boldsymbol{e}_4^*$	$g_1(type_a, \text{'TelnetTerminaltype'}) = 1$, $f_1(ipdst_a, ipdst_p) = 1$
κ_5	$\boldsymbol{e}_5^*$	$g_1(type_a, \text{'Rsh'}) = 1$, $f_1(ipdst_a, ipdst_p) = 1$
κ_6	$\boldsymbol{e}_6^*$	$g_1(type_a, \text{'Rsh'}) = 1$, $f_1(ipsrc_a, ipdst_p) = 1$
κ_7	$\boldsymbol{e}_7^*$	$g_1(type_a, \text{'TelnetTerminaltype'}) = 1$, $f_2(\{ipsrc_a\}, \{ipsrc_p, ipdst_p\}) = 1$, $f_2(\{ipdst_a\}, \{ipsrc_p, ipdst_p\}) = 1$
κ_8	$\boldsymbol{e}_8^*$	$g_1(type_a, \text{'Mstream\_Zombie'}) = 1$, $f_{neq}(ipdst_a, ipdst_p) = 1$
κ_9	$\boldsymbol{e}_9^*$	$g_1(type_a, \text{'Mstream\_Zombie'}) = 1$, $f_2(\{ipsrc_a\}, \{ipsrc_p, ipdst_p\}) = 1$, $g_1(ipdst_a, 255.255.255.255) = 1$

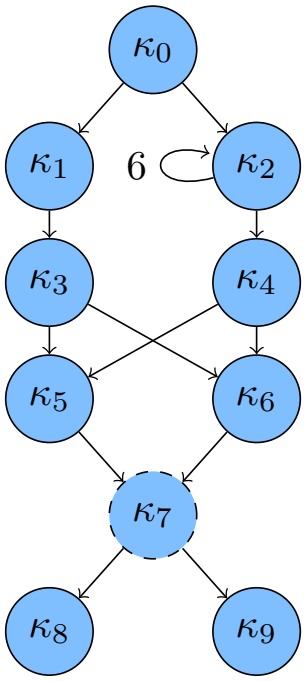


Figure 5.8: Representation of AASG Mill-1, built from an instance of LLDoS 1.0. Logs used to build the AASG are shown in Table 5.2

5.4.1 Structure and notation

In the implementation, each AASG is a JSON object containing the following members:

- **`id`**: A unique integer identifying the AASG among a set of AASGs. This allows making reference to any AASG during working operations.
- **`nodes`**: A list of objects representing the nodes of the AASG.
- **`arcs`**: A list of objects, each one representing all the arcs starting from a certain node in the AASG.

Each node κ_n has the following members:

- **`id`**: A unique integer i identifying the node. It is mandatory to assign the value 0 to the root node, the source of the graph.
- **`e_star`**: The abstract event $\boldsymbol{e}^*$, expressed as a list of conditions. Each condition is also an object, whose members are different depending on whether the condition is absolute or relative. Both types have the member **`function`**, which indicates the type of function to apply in each case. The names given to each function in the implementation are collected in Table 5.3. There is also an

additional member, **`threshold`**, which is a float representing λ, the variable determining when the condition is met for functions g_3, g_4, f_3^* and f_4^*. This member is not needed if we use other functions.

An **absolute condition** has the following specific members:

- **`att`**: The name of the attribute to which the condition is applied. The full list of attribute names can be found in Appendix C.
- **`r`**: The reference value, the value against which the comparison is made. It can be any kind of scalar corresponding to the type of value defined for the attribute (a string, an integer, etc.) or an array in the case of the set-based function (g_5).

A **relative condition** has other two different specific members:

- **`attP`**: The name of the attribute of the previous event matching the parent node.
- **`attC`**: The name of the attribute of the event found to match the node.

Type	Name of function	Implem. name
Absolute conditions	Equality	`EQL`
	Inequality	`NEQ`
	Prefix similarity	`PFX`
	Textual similarity	`TXT`
	Set-based	`SET`
Relative conditions	Equality	`SIM_EQL`
	Common element	`SIM_COM`
	Prefix similarity	`SIM_PFX`
	Textual similarity	`SIM_TXT`
	Inequality	`SIM_NEQ`

Table 5.3: Implementation in JSON of condition functions

Finally, there are two additional and optional members in each node object, to implement counters and optional nodes:

- **`counter`**: It is an integer different from 0 corresponding to the counter of the node. If the member does not exist, the counter is considered to be 1.
- **`optional`**: A boolean being `true` if the node is optional or `false` if it is not. When the member is not present, it is assumed as `false`. The root node can never be optional.

Objects representing arcs do not contain the information of just one arc in the node, but of all the arcs that start from each node with children. This eases the implementation of algorithms going through the AASG from the root node to the end

of a branch. Each time an algorithm of this kind evaluate a node, it can easily check if this node has any children and, if this is the case, find all the arcs leading to them. The members contained in an object representing a set of arcs are:

- **`start`**: The identifier of the node where this set of arcs starts. There is only one object representing a set of arcs for each starting node, so it also acts as a unique identifier of the object.
- **`children`**: A list of objects representing the nodes to which the arcs going out from `start` are pointing. In the basic implementation of an AASG, each object representing a child has only one member, **`id`**, corresponding to the identifier of the child node. We will see in Chapter 6 that representing it as a list of objects and not as a mere list of identifiers allows adding other members required by the algorithms using AASGs.

5.4.2 Example of implementation

We present below an example of implementation in JSON format of the AASG represented in Figure 5.8.

```
"id": 1,
"nodes": [
   {"id": 0,"e_star": [
      {"function": "EQL","att": "type","r": "Sadmind_Ping"}]},
   {"id": 1,"e_star": [
      {"function": "SET","att": "type","r": ["Sadmind_Amslverify_Overflow","Admind"]},
      {"function": "SIM_EQL","attC": "ipsrc","attP": "ipsrc"},
      {"function": "SIM_EQL","attC": "ipdst","attP": "ipdst"}]},
   {"id": 2,"e_star": [
      {"function": "EQL","att": "type","r": "Sadmind_Amslverify_Overflow"},
      {"function": "SIM_EQL","attC": "ipsrc","attP": "ipsrc"},
      {"function": "SIM_EQL","attC": "ipdst","attP": "ipdst"}],
    "counter": 6},
   {"id": 3,"e_star": [
      {"function": "SET","att": "type","r": ["Sadmind_Amslverify_Overflow","Admind"]},
      {"function": "SIM_NEQ","attC": "type","attP": "type"},
      {"function": "SIM_EQL","attC": "ipsrc","attP": "ipsrc"},
      {"function": "SIM_EQL","attC": "ipdst","attP": "ipdst"}]},
   {"id": 4,"e_star": [
      {"function": "EQL","att": "type","r": "TelnetTerminaltype"},
      {"function": "SIM_EQL","attC": "ipdst","attP": "ipdst"}]},
   {"id": 5,"e_star": [
      {"function": "EQL","att": "type","r": "Rsh"},
      {"function": "SIM_EQL","attC": "ipdst","attP": "ipdst"}]},
   {"id": 6,"e_star": [
      {"function": "EQL","att": "type","r": "Rsh"},
      {"function": "SIM_EQL","attC": "ipsrc","attP": "ipdst"}]},
   {"id": 7,"e_star": [
      {"function": "EQL","att": "type","r": "TelnetTerminaltype"},
      {"function": "SIM_COM","attC": ["ipsrc"],"attP": ["ipsrc","ipdst"]},
      {"function": "SIM_COM","attC": ["ipdst"],"attP": ["ipsrc","ipdst"]}]},
    "optional": true},
   {"id": 8,"e_star": [
      {"function": "EQL","att": "type","r": "Mstream_Zombie"},
      {"function": "SIM_NEQ","attC": "ipdst","attP": "ipdst"}]},
   {"id": 9,"e_star": [
      {"function": "EQL","att": "type","r": "Mstream_Zombie"},
      {"function": "EQL","att": "ipdst","r": "255.255.255.255"},
      {"function": "SIM_COM","attC": ["ipsrc"],"attP": ["ipsrc","ipdst"]}]}],
```

```
40   "arcs": [
41      {"start": 0,"children": [{"id": 1},{"id": 2}]},
42      {"start": 1,"children": [{"id": 3}]},
43      {"start": 2,"children": [{"id": 4}]},
44      {"start": 3,"children": [{"id": 5},{"id": 6}]},
45      {"start": 4,"children": [{"id": 5},{"id": 6}]},
46      {"start": 5,"children": [{"id": 7}]},
47      {"start": 6,"children": [{"id": 7}]},
48      {"start": 7,"children": [{"id": 8},{"id": 9}]}]
```

5.4.3 Graph editor

The JSON format defined to represent AASGs can become difficult to use by the analyst if the defined AASGs contain many nodes. Special care must be observed to avoid punctuation errors or unbalanced brackets. A graphical representation of the AASG is more human friendly. For this reason, we propose a method for graphically designing an AASG. The graph is then automatically converted to JSON, so it can be used by a computer system.

We propose the use of Cytoscape[9], a renowned open source program for graph design and processing. It was conceived for being used on biological networks, but its interface is generic and allows working with any kind of graph. A screenshot of the AASG of Figure 5.8 as it is represented in Cytoscape is shown in Figure 5.9. We have given a name to each node based on the type of event it identifies. The names given to the nodes do not affect the final structure of the AASG in JSON, and it is just a form to help the analyst during the creation of the graph.

Each node should have an attribute `e_star` where the list of conditions is coded. Conditions are strings composed of three parts separated by the character '`#`'. Each part represents a member of the condition in order: `function`, `att` and `r` if it is an absolute condition, and `function`, `attP` and `attC` if it is a relative condition. In case that one of the parts is a list, as it happens for example with `r` when using function `SET`, its elements are separated by '`$_`'. For instance, the first condition of $\boldsymbol{e}_1^*$ would be expressed as '`SET#type#Sandmind_Amslverify_Overflow$_Admind`'.

We have developed a program in Python to do the automatic conversion between one of the standard exportation format used by Cytoscape, Cytoscape.js JSON (.cyjs), and the JSON format defined in section 5.4.1. It also allows defining several AASGs in the same Cytoscape document without specifying which node belongs to each AASG. The coded algorithm automatically analyzes all the nodes and arcs to return a JSON where the AASG are separated and their root nodes identified. We have also developed a program to convert back from AASG JSON to Cytoscape.js format.

[9] https://cytoscape.org/

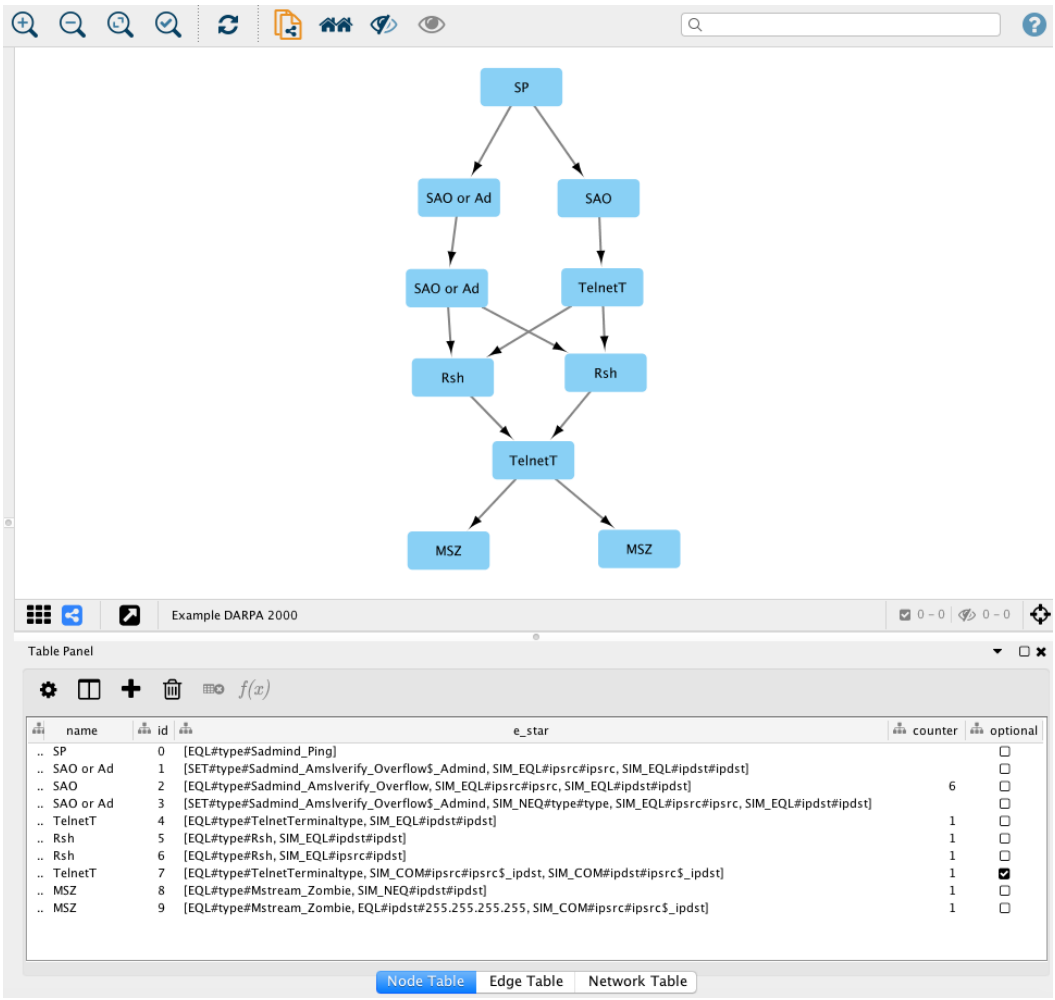


Figure 5.9: The AASG in Figure 5.8 represented in the program Cytoscape.

5.5 Summary

In this chapter we have defined what is an Abstract Attack Scenario Graph (AASG) and how we can use it to represent alternative attack scenarios. AASGs have been introduced in section 5.1 using some examples of practical applications, involving multi-step attacks such as WannaCry or the attacks contained in the DARPA 2000 dataset. We have then given a formal definition of AASG in terms of graph theory in section 5.2, followed by a full example of creation of an AASG in section 5.3. Details about the implementation of an AASG using a JSON format are shown in section 5.4, together with a format for graphically representing the AASG in Cytoscape. These formats are conceived to be understood both by humans and machines. All the examples of AASG used throughout this thesis are gathered together in Appendix F.

The AASG model fulfills all the requirements we had fixed in section 5.1.2:

1. **Representativeness**. An AASG is able to capture the hypotheses of the analyst about multi-step attacks. We saw an example in section 5.3.

2. **Simplicity and intuitiveness**. The AASG is human readable, whether in its mathematical formulation (section 5.2), in its visual representation (e.g. Figure 5.8) or in the JSON format (section 5.4.1).

3. **Mathematical form**. A mathematical definition of the AASG using elements from graph theory has been given in section 5.2.

4. **Computation friendliness**. An AASG can be easily adapted to work with computers. We have proposed the JSON format in section 5.4.1 specifically to do so.

5. **Compliance with the purpose**. We will see in Chapter 6 how AASGs are suitable for the tasks of identification and detection of multi-step attacks.

6. **Boundness**. The end of a scenario is well defined in an AASG as a node with no further children. This requirement is enforced because an AASG has no cycles.

7. **Uniqueness**. Each AASG is univocally identified by its root node κ_0.

8. **Flexibility**. Absolute and relative conditions, presented in sections 5.2.4 and 5.2.5, respectively, allow representing a wide variety of sequences of events.

9. **Suitability for learning**. Probabilities can easily be assigned to each arc in an AASG. This is the reason why children are represented as objects in the JSON format (see page 130). We will address this topic in Chapter 6.

Once the AASG has been defined, the challenge now is the development of algorithms able to exploit its characteristics. We will be seeing a couple of them in the next chapter.

Chapter 6

Algorithms to exploit AASG

Contents

6.1 Morwilog 136
6.2 Bidimac 152
6.3 Differences between Morwilog and Bidimac 159
6.4 Summary 166

In that deep senselessness I had a vision:
There was the oak, as many-leaved as ever,
As many ants among its many branches —
— Ovid, *Metamorphoses (The Myrmidons)*

We have seen in Chapter 5 how different multi-step attack cases can be arranged into an AASG. The motivation under the design of AASGs is to have a solid link between the alternative cases of attack conceived by the human analyst and the exploitation of these alternatives by an algorithm. The objective is to guide the human in a) the determination of which alternatives better represent the attack and b) the detection of any of the proposed alternatives when they are identified within the events in the network.

To reach this goal, an algorithm exploiting AASGs should be built on the following premises:

- **Identification**. An evaluation of the importance of each arc is needed to determine which branch better correspond to the attack. An AASG allows the assignation of weights to numerically represent this importance.
- **Detection**. Alarms should be generated to signal potential attacks.
- **Human in the loop**. The evaluation of these alarms as truth detections or false positive should be used to improve the information coded in the AASGs.

One of our proposal to this respect is Morwilog, a method based on Ant Colony Optimization (see section 6.1.1). We present it in detail in section 6.1. A second method is Bidimac, an adaptation of Bayesian inference for the exploitation of AASGs,

presented in 6.2. The evaluation of different multi-step attack alternatives proposed by the human analyst is original and has not been considered in the current literature. As it is based on classical Bayesian inference, Bidimac provides a reference to compare Morwilog with an existent paradigm. The theoretical similarities between the two methods are discussed in section 6.3.

6.1 Morwilog

Morwilog is the main algorithm we propose for the exploitation of AASGs. It was first presented in 2016 at the Symposium Series on Computational Intelligence (SSCI) [Navarro 2016] in Athens and retaken later for another paper published in the EURASIP Journal on Information Security [Navarro 2018b]. The implementation presented then worked on what we called 'event trees', which have evolved to become the current AASGs.

Before presenting Morwilog, we start with a review of Ant Colony Optimization (ACO), the metaheuristic on which Morwilog is based, section 6.1.1. We continue giving a general overview of its functioning in section 6.1.2, before diving into the technical details. In section 6.1.3, we present how the AASG is adapted to become a stigmergic AASG that can be used by Morwilog. We also explain how the evolution of pheromones in the model works. Finally, in section 6.1.4 we present the algorithm in the form of pseudocode, describing its functioning step by step.

6.1.1 Ant Colony Optimization

Morwilog is based on Ant Colony Optimization (ACO) [Dorigo 2004], a metaheuristic to solve discrete optimization problems that is inspired by the behavior of foraging ants. We refer to the three diagrams in Figure 6.1, presented in chronological order, during the explanation of this behavior. When the first foraging ant goes out from the anthill (diagram A), it does not know which direction it should take to arrive at the nearest food source. It then starts to randomly wander in the search of food, and so is done by the next ants following it. When one ant eventually discovers a food source, it deposes chemical substances, called pheromones, in its way back home, creating a trail between the anthill and the food source. This trail helps the own ant to retrace the same path back to the food source, but it can also be perceived by other ants. Ants are attracted by pheromones, and this attraction is stronger as the level of pheromones gets higher. There is then a process of indirect communication between the members

of the colony through the modification of the environment, which is called *stigmergy* [Theraulaz 1999].

As an ant follows a trail created by other ant, it also deposes pheromones, which are added up to the previous ones in the trail. It may happen that several ants find different food sources or different paths to the same food source (diagram B). In that case, a phenomenon of emergence arises and only the shortest path tends to prevail after a certain time, as we can see in the diagram C. This happens because the deposition of pheromones is higher in the shortest paths, where each ant takes less time to go back and forth.

This does not mean that new food sources could not be found once a trail is established. Pheromones are not imperatively followed by all ants, and there exist some of them which explore other directions, such as the one on the right of the diagram C. If one of them finds a closer food source, all the rest will eventually end up going to this source, even if it can take time to do the change. This is possible because the pheromones evaporate upon air contact, avoiding stagnation when disappearing from a previously preferred trail as more and more ants use the new one.

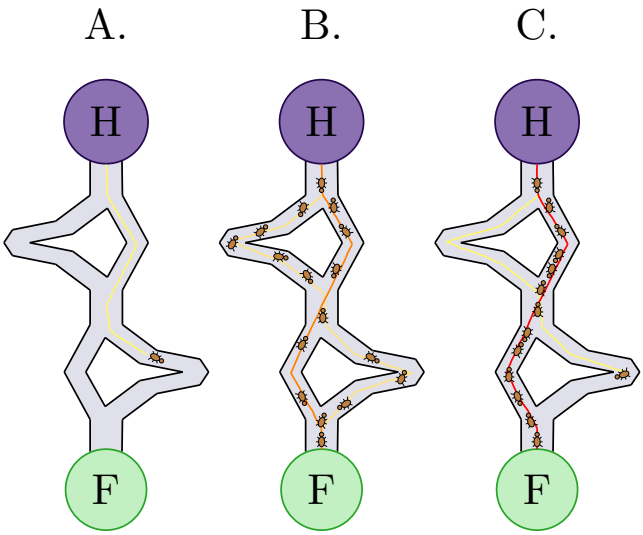


Figure 6.1: Example of the convergence to shortest path done by real foraging ants. *H* denotes the anthill and *F* the food source. Image inspired by: https://upload.wikimedia.org/wikipedia/commons/thumb/a/af/Aco_branches.svg/800px-Aco_branches.svg.png.

ACO is intended to find the optimal solution to a problem by generating artificial ants that, as the real ones, randomly move through a parameter space, tending to follow the trails of pheromones left by themselves and by other ants. The same as an ant colony converges to the shortest path to food, ACO converges to the optimal

solution of the problem. We will explain more specifically how the natural behavior of foraging ants is translated into the mechanisms in ACO when explaining the details of Morwilog in section 6.1.2.

There are many algorithms based on the ACO metaheuristic. The first one was Ant System, published in 1991 in the Ph.D. thesis by Marco Dorigo [Dorigo 1991], the founding father of ACO. He applied the algorithm to the traveling salesman problem (TSP), where the search space is fairly straight to define, as routes can be directly translated to the trails taken by the artificial ants. Since then, ACO has been successfully applied to many NP-hard optimization problems such as control of traffic control signals [Haldenbilen 2013] or galvanizing line scheduling [Fernandez 2014]. Note that ACO-based methods are not the only ones inspired by the behavior of ants. There is another family of algorithms to perform clustering that is inspired by how a certain type of ant piles the bodies of its dead mates [Zhe 2011].

We find several proposals of ACO-based algorithms applied to attack detection. Some of them implement the artificial ants as agents that move from one asset to the other and collaborate in the detection and anticipation of attacks [Hui 2009, Fink 2014]. Other work uses the inspiration from ant behavior to perform anomaly detection through the clustering of traces [Jeyepalan 2014, Fernandes 2016], in some cases in combination with supervised methods such as SVM [Feng 2014]. ACO has also been combined with an evolutionary fuzzy system for the generation of detection rules [Abadeh 2015]. More examples have been collected in two surveys, one about swarm intelligence in intrusion detection [Kolias 2011] and the other about bio-inspired security approaches [Rauf 2018]. None of these examples addresses multi-step attack detection. We have found some approaches modeling the behavior of an attacker in an attack graph when performing a multi-step attack [Mahanti 2005, Zhang 2009], but none of them proposes a detection method on incoming traces.

The version of ACO that directly inspired the creation of Morwilog is the *Hommilière* (Manhill) system developed by Valigiani [Valigiani 2006] in his Ph.D. thesis. This system has been implemented in an e-learning platform, and its goal is to recommend personalized learning paths to each student. The recommendations depend on a mix between the past performance of the student and the results obtained by other students following similar paths. In the Manhill algorithm, the paths traversed by the artificial ants are built through the different lessons, which are arranged in the form of a graph [Gutiérrez 2007]. The level of deposed pheromones depends on how good are the results got by the student after each lesson, confirming or not the pertinence of that lesson in the educational evolution of the student. The most interesting

Name	Description	Ref. value
T_{max}	Maximum search time	62 s
τ_0	Initial number of pheromones	1000
ρ	Evaporation rate	0.02
w	Spreading of $\Delta\tau^{+,-}$ function	1000
$\Delta\tau_0^+$	Change of pheromones when $\tau[t] = \tau_0$	500
τ_{min}	Minimum level of pheromones	100

Table 6.1: List of parameters for Morwilog, together with their description and the reference value used in the examples

contribution made by this algorithm is the interaction between the students and the ACO model. An artificial ant is created when a student connects to the platform. This notably differs from most of the existent implementations of ACO, where a fixed amount of artificial ants is created when the algorithm begins its execution. This brings the possibility of incorporating the variability of natural events, in this case the arrival of students, to the generation of the artificial ants. We have taken this idea in the conception of Morwilog, where the natural entity are the traces, that generate an artificial ant when they are collected by the system.

6.1.2 General overview of Morwilog

Morwilog applies an ACO-based mechanism on AASGs to help the security analyst in the confirmation or refusal of the alternative scenarios represented in each of the branches. An artificial ant generated within the execution of the algorithm is called *morwi*. This term is the prefix meaning 'ant' in Proto-Indo-European [Pokorny 2007], a theoretical reconstruction of the common ancestor of the Indo-European languages [Clackson 2007].

The creation of each *morwi* is linked to an external phenomenon: the arrival of an event matching the root node κ_0 of one of the AASGs stored in the database. Once a *morwi* is created, it moves through the AASG in a process similar to the one used by rule-based correlation methods (see section 5.1.3, page 110). From each node, the *morwi* searches for events matching the children of that node among the incoming events. This search is performed during a maximum time T_{max}, defined for avoiding the saturation of system resources. T_{max} is one of the parameters in Morwilog. All the parameters used are listed in Table 6.1 together with a reference value used in the examples of this chapter. Their purpose will be explained as we describe how the algorithm works.

Once the matched children are found, the *morwi* chooses one of them to continue its journey through the AASG. This choice is based on a certain level of pheromones previously deposed by other *morwis*. We explain how pheromones are associated to the different arcs in the AASG in section 6.1.3. So far we just need to consider that each arc $\kappa_n\kappa_m$ has a number $\tau_{n,m}$ associated to it representing a level of pheromones.

A *morwi* chooses the following node with a probability that is proportional to the level of pheromones associated to the arc leading to that node. There is then a higher probability of choosing a path with a higher level of pheromones, like in classic ACO. We define the *choosing probability* of an arc $\kappa_n\kappa_m$ in the AASG as the probability that a *morwi* being in the node κ_n chooses the node κ_m given that events matching also all the rest of the children of κ_n are found. This probability is expressed as:

$$P_{n,m} = P(\kappa_n\kappa_m) = \frac{\tau_{n,m}}{\sum_{\kappa_a\kappa_b \in A | a=n} \tau_{a,b}} \tag{6.1}$$

The *morwi* follows this process of event search and random selection until it reaches a node with no children. When this happens, it returns the found sequence to the security analyst. The security analyst needs to check the effects of the actions reflected in the events to verify if they represent an attack or not. Once a verdict is taken, the analyst transmits it to Morwilog, which accordingly updates the level of pheromones associated to each arc in the AASG. If the sequence is considered as an attack, the levels of pheromones associated to the selected branch are incremented and the branch is thus reinforced. If it is not evaluated as an attack, the level of pheromones is decremented. In both cases, a mechanism of pheromone evaporation takes place to avoid stagnation, as it is proposed in classic ACO. More details about the evolution of pheromones are presented in section 6.1.3.

Signals sent to the analyst to indicate the identification of a branch are called 'alarms' to differentiate it from 'alerts'. This distinction avoids confusion with IDS alerts, that can be at the input of both Morwilog and Bidimac. Anyways, alarms generated by both algorithms are also alerts in the sense of the definition in section 2.3.

Before going into the technical details of Morwilog, we present an example of how it works in general terms. The AASG used is built for the WannaCry attack (see section 3.2.1). The example is illustrated in Figure 6.2 and the list of the types of event represented by each node is given in Table 6.2. The evolution of pheromones in the example will be explained later in this chapter.

The steps followed by Morwilog from the creation of a *morwi* until it returns a sequence of events as an alarm are described hereunder.

Level	Node	Matched event	Origin
0	κ_0	Unsuccessful HTTP request from a host, called A, to long domain name D_1	Proxy
1	κ_1	Successful connection in port TCP 445 from A to any another endpoint (B) in the local network	Internal firewall
1	κ_2	Successful HTTP request to long domain name D_2	Proxy
2	κ_3	SMBv1 communication between A and other machine using command 'transaction2_secondary'	Internal Firewall
2	κ_4	Malformed SMB headers for the NEGOTIATE PROTOCOL REQUEST from A to B (CVE-2009-3103)	Endpoint B
2	κ_5	SQL Injection alert coming from A	IDS

Table 6.2: Nodes in the example represented in Figure 6.2, containing the propagation steps in WannaCry

a) Each time a suspicious event e_i arrives at Morwilog, a *morwi* is generated and it looks for the AASG, if there is any, whose root node κ_0 matches the event. In this case, the chosen AASG is ψ_u.

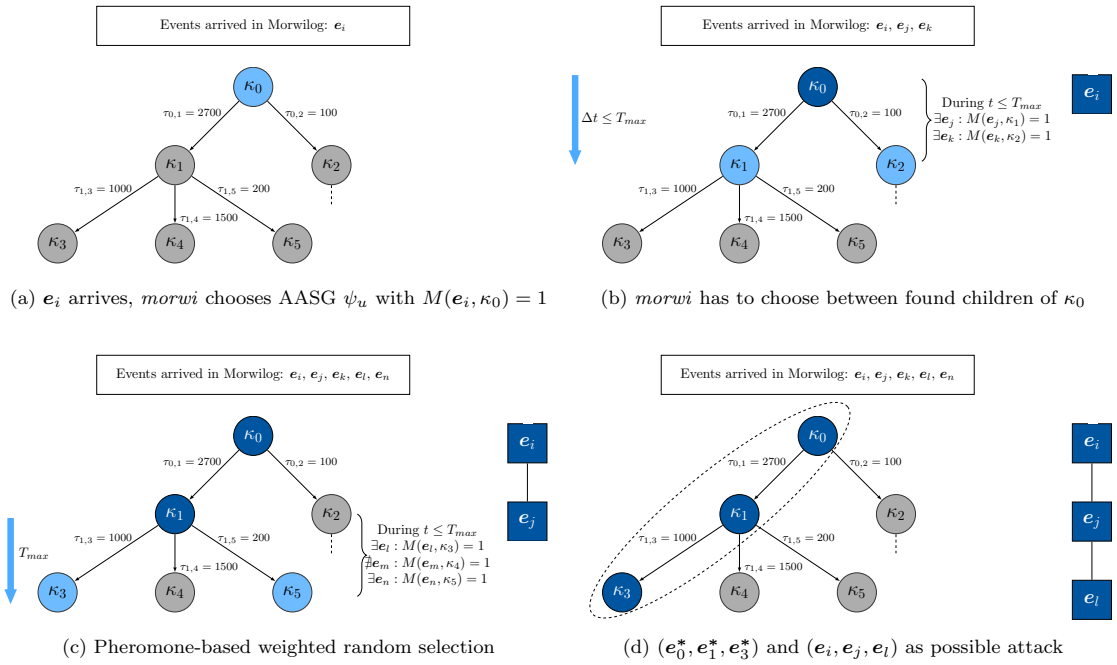
b) Starting in κ_0, the *morwi* searches for events matching each one of the nodes being children of κ_0. The maximum search time is T_{max}. In this case, events e_j and e_k are found, corresponding to κ_1 and κ_2, respectively. Now the *morwi* has to decide a node for continuing its trip through the AASG doing a pheromone-based weighted random selection. The node whose connection with the present node has a higher level of pheromones, κ_1, is the most probable option.

c) Imagine the *morwi* chooses this node κ_1, matched by the event e_j. Now it has to search the events matching the nodes among the children of κ_1. In this case, no event is found corresponding to κ_4, which represents the exploitation of a vulnerability in SMBv2[1]. Only events e_l and e_n are found, matching κ_3 and κ_5, respectively. The *morwi* needs to choose between these two nodes, the first one having a higher probability.

d) We can imagine the *morwi* chooses κ_3. It then stops because the node has no children. The sequence (e_i, e_j, e_l) is returned then as a possible attack.

In this case, the sequence corresponds to a real instance of WannaCry and the hypothesis is reinforced. The reinforcement is done through the modification of the level of pheromones based on the feedback coming from the human analyst. When she confirms the malicious nature of the sequence, the pheromones associated to the matched branch in the AASG are incremented. We will continue with the example in the following section (page 146), where we analyze how the pheromones change.

[1] http://seclists.org/fulldisclosure/2009/Sep/39



(a) e_i arrives, *morwi* chooses AASG ψ_u with $M(e_i, \kappa_0) = 1$

(b) *morwi* has to choose between found children of κ_0

(c) Pheromone-based weighted random selection

(d) (e_0^*, e_1^*, e_3^*) and (e_i, e_j, e_l) as possible attack

Figure 6.2: Example about the progress of a *morwi* through an AASG.

6.1.3 AASG as the stigmergic scenario

The AASG is then the scenario where the process of stigmergy (see page 137) between *morwis* is performed. Pheromones are deposited in the arcs of the AASG for reinforcing a path leading to a confirmed attack or retired to penalize one leading to a false alarm. This stigmergic process gives information to the newly generated *morwis*.

The values of pheromones are easily incorporated into the AASG model. Their accumulation is represented by $\tau_{n,m} \in \mathbb{R}_{>0}$ for each existent arc $\kappa_n\kappa_m$. A modified AASG including $\tau_{n,m}$ in its arcs is called a *stigmergic AASG*. To represent pheromones in the JSON format, we just add the numeric member `ph` to each object in `children` for each one of the sets of arcs (see page 130).

The increment and decrement of pheromones is performed according to the evaluation of the alarms done by the security analyst, who provides feedback to the system. The amount of pheromones to be added up to the existent level of pheromones in an arc once the analyst has decided to reinforce the branch is called $\Delta\tau^+$. The amount of the decrement is called $\Delta\tau^-$. To refer to any of the two, we use $\Delta\tau^{+,-}$. We have that $\Delta\tau^+ \in \mathbb{R}_{>0}$ and $\Delta\tau^- \in \mathbb{R}_{<0}$.

If we consider $\tau[t]$ as the level of pheromones after update t, with $t \in \mathbb{N}$:

$$\tau[t+1] = \tau[t] + \Delta\tau^{+,-} \tag{6.2}$$

The same as in classic ACO, a mechanism of pheromone evaporation is needed to avoid stagnation. Thanks to this, the *morwis* statistically have an opportunity to stop favoring branches where the level of pheromones is high but that have not been identified for long time. The evaporation consists in the decrease of the previous level of pheromones by an evaporation rate ρ, with $\rho \in \mathbb{R}$ and $0 < \rho < 1$. The equation to calculate the resulting level of pheromones after evaporation is:

$$\tau[t+1] = (1-\rho) \cdot \tau[t] \tag{6.3}$$

Evaporation is applied to every arc in the AASG each time there is an update of pheromones, right before the addition of $\Delta\tau^{+,-}$, independently if the selected branch has its pheromones incremented or decremented.

While evaporation is done as in classic ACO, we have introduced an important variation in the increment and decrement. In the previous literature, both changes are constant and independent of the level of pheromones at the moment when the update is made. The values of $\Delta\tau^+$ and $\Delta\tau^-$ are thus fixed before the execution of the algorithm, as parameters. The combination of this fixed addition with the evaporation of Equation 6.3 results in a variation of pheromones whose absolute value decays as subsequent updates of pheromones of the same sign are applied in an arc.

This constant pheromone variation is not enough for strongly penalizing badly chosen branches. We need to have in mind that the analyst can arrive at wrong conclusions when building the AASG, and we want those wrong paths to be out of the selective process as soon as possible. That is why we have introduced in $\Delta\tau^+$ and $\Delta\tau^-$ a dependence on the current level of pheromones ($\tau[t]$). The proposed function is Gaussian, and $\Delta\tau^+ = -\Delta\tau^-$:

$$\Delta\tau^+(\tau[t]) = \Delta\tau_0^+ e^{-\frac{(\tau[t]-\tau_0)^2}{2w^2}} \tag{6.4}$$

There are three parameters in this equation: τ_0, $\Delta\tau_0^+$ and w. The first one, τ_0, is the initial level of pheromones in the arc ($\tau[0] = \tau_0$). It is taken as a parameter of the system and every arc in every new AASG is initialized with τ_0 pheromones. The Gaussian function for $\Delta\tau^{+,-}$ is centered to τ_0 to have the highest rate of change right after the creation of the AASG. The amount of change for the first update, when $\tau[t] = \tau_0$, is precisely $\Delta\tau_0^+$. It has the same value for both $\Delta\tau^+$ and $\Delta\tau^-$. Finally, w is a parameter determining how spread the Gaussian function is.

In a complete update, pheromones are first evaporated and then incremented or decremented, based on the new value after the evaporation. Having this in mind, we have the following final equations for increment and decrement updates, respectively:

$$\tau[t+1] = (1-\rho)\cdot\tau[t] + \Delta\tau_0^+ e^{-\frac{((1-\rho)\cdot\tau[t]-\tau_0)^2}{2w^2}} \tag{6.5}$$

$$\tau[t+1] = (1-\rho)\cdot\tau[t] - \Delta\tau_0^+ e^{-\frac{((1-\rho)\cdot\tau[t]-\tau_0)^2}{2w^2}} \tag{6.6}$$

It is desirable that the absolute value of the level of pheromones in each arc do not exceed certain threshold, to detect the moment from which a branch is clearly confirmed over the others. This also prevents the system to unlimitedly favor an arc. To do so, the difference between consecutive updates of the level of pheromones should decrease as the system evolves. This is the case of the classic ACO equation, whose convergence has been proven by Dorigo and Stützle [Dorigo 2004]. Equations 6.5 and 6.6 given here also converge to a certain value once the parameters are fixed.

For proving that, we choose one of the equations, for example the one corresponding to the increment (Equation 6.5). We can imagine a really large number N of continuous increments in one of the arcs, so we constantly apply the same equation for each update. This scenario is known as a situation of *continuous reinforcement*. Calculating the exact limit of the Equation 6.5 in this scenario is not as trivial as in classic ACO, due to the recursive nature of the function. But we have empirically proven that this upper limit exists for certain values of the parameters just running the recursive function during a high number of pheromone updates.

The results are shown in Figure 6.3, where the evolution of the level of pheromone in one arc is represented against the number of pheromone updates. We use the parameters shown in Table 6.1. The curve 'Increment' represents the evolution of pheromones under the situation of continuous reinforcement, supposing there is an increment in every update. An upper limit, 3051, is almost reached in around 10 updates.

For comparing this result with classic ACO, we have also represented the result of the fixed increment in a continuous reinforcement scenario in the curve called 'Classic'. To make the two curves comparable, we have chosen an increment value of 62.1, which is approximately the one needed to reach the same upper limit as in 'Increment', 3104,63. We see that our approach reaches the limit much earlier than the classical one.

The same logic can be applied to the decrement function $\Delta\tau^-$. We could imagine

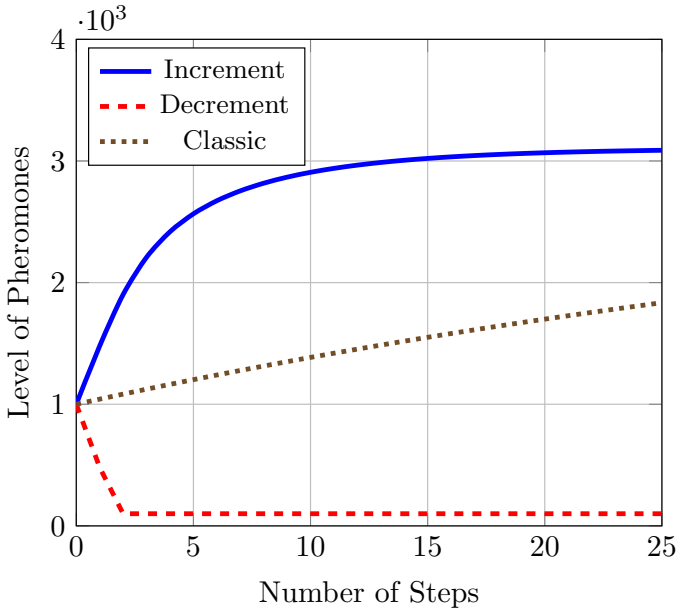


Figure 6.3: Graph representing the evolution of pheromones for continuous reinforcement in Morwilog ('Increment') and classic ACO ('Classic'), and for continuous penalization ('Decrement')

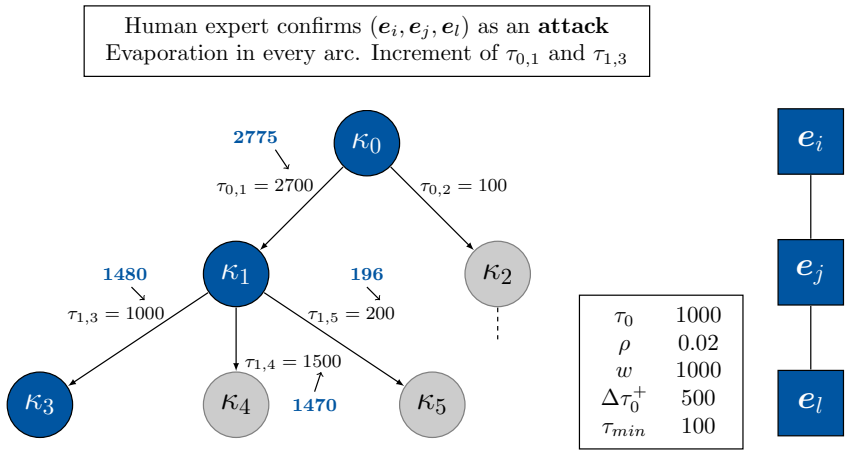


Figure 6.4: Example of evolution of the AASG when the attack is confirmed.

a scenario of *continuous penalization*, where an arc sees its level of pheromones decremented after each pheromone update in the AASG. The function also converges, in this case to a lower limit. But this limit can be negative depending on the parameters chosen. Negative values of pheromones should be avoided, not only to adjust to the biological metaphor but also to make the operations in the algorithm easier, without needing the implementation of negative integers. As in classic ACO, a minimum value of pheromones τ_{min} is artificially set. The level of pheromones in any arc is then force

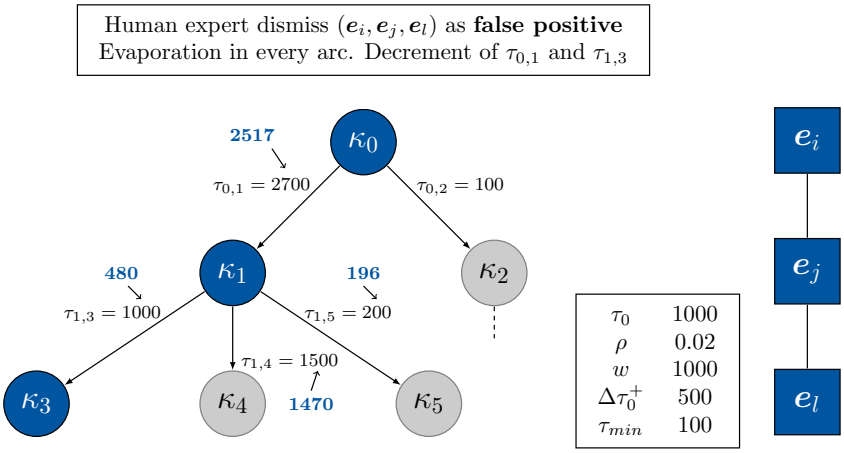


Figure 6.5: Example of evolution of the AASG when the attack is a false alarm.

to stay over this value. The curve 'Decrement' in Figure 6.3 shows the results of this continuous penalization, applying $\tau_{min} = 100$.

To finish this introduction, we retake the example we presented in last section (page 140), where the main steps of WannaCry were represented in the AASG. If the sequence returned by the *morwi* is evaluated as an attack, the level of pheromones of that branch is incremented. The result is shown in Figure 6.4.

On the contrary, we show the mechanism of decrement of pheromones in Figure 6.5. This process is performed because the returned sequence is considered as a false alarm by the analyst. In the case of WannaCry, the command 'transaction2_secondary' in SMBv1, represented in node κ_3, can be used for testing purposes and not with a malicious objective, even if it is an important step in the attack. If this happens, the feedback provided by the analyst leads to the process of decrement.

6.1.4 The Morwilog algorithm

The process explained above is formalized in the Morwihill and Morwi algorithms. Morwihill is in charge of generating a *morwi* each time an event arrives at the system, whose behavior is determined by the algorithm Morwi. We explain first the operations made by each *morwi*, expressed in pseudocode in Algorithm 6.1, for later presenting the Morwihill operations for interpreting the results returned by the *morwi* and update the pheromones in the corresponding AASG (Algorithm 6.2).

Each *morwi* is created after the arrival of an event $\boldsymbol{e}_i$ from the set E_{in} of all events that arrive at the system. E_{in} is assumed to be ordered in time. A *morwi* returns the following elements:

Algorithm 6.1 Morwi algorithm

Require: $E_{in} \subseteq \mathbb{E}$; $\boldsymbol{e}_i \in E_{in}$
Ensure: $\boldsymbol{s}_i \in S_{E_{in}}$; $\psi_u = (K_u, A_u) \in \Psi$; $\boldsymbol{b}_\kappa \in B_{\psi_u}$; $isresult \in \{\text{true, false}\}$

1: $isresult \leftarrow$ false
2: $\psi_u \leftarrow \psi \in \Psi \mid M(\boldsymbol{e}_i, \kappa_0) = 1$
3: **if** $\psi_u \neq \varnothing$ **and not** $(\psi_u = \psi_{last} \wedge \Delta t_{i,last} < T_{max} \wedge \Pi_{last} = \text{true})$ **then**
4: $\quad \boldsymbol{s}_i \leftarrow [\boldsymbol{e}_i]$
5: $\quad \kappa_n \leftarrow \kappa_0 \in K_u$
6: $\quad \boldsymbol{b}_\kappa \leftarrow [\kappa_n]$
7: $\quad F_n \leftarrow \{\kappa_a\kappa_b \in A_u \mid a = n\}$
8: $\quad$ **while** $F_n \neq \varnothing$ **do**
9: $\quad\quad C \leftarrow \varnothing$
10: $\quad\quad$ **for each** $\kappa_n\kappa_b \in F_n$ **do**
11: $\quad\quad\quad$ Find $1^{st}\, \boldsymbol{e}_q \in E_{in} \mid q > i, M(\boldsymbol{e}_q, \kappa_b) = 1, f_1^t(\Delta t_{i,q}, T_{max}) = 1$
12: $\quad\quad\quad$ **if** $\exists\, \boldsymbol{e}_q$ **then**
13: $\quad\quad\quad\quad$ Add $(\kappa_n\kappa_b, \boldsymbol{e}_q)$ to C
14: $\quad\quad\quad$ **end if**
15: $\quad\quad$ **end for**
16: $\quad\quad$ **if** $C \neq \varnothing$ **then**
17: $\quad\quad\quad$ $(\kappa_n\kappa_c, \boldsymbol{e}_c) \leftarrow c \in C$, τ-based selection
18: $\quad\quad\quad$ Add $\boldsymbol{e}_c$ to $\boldsymbol{s}_i$
19: $\quad\quad\quad$ Add κ_c to $\boldsymbol{b}_\kappa$
20: $\quad\quad\quad$ $\kappa_n \leftarrow \kappa_m$
21: $\quad\quad\quad$ $F_n \leftarrow \{\kappa_a\kappa_b \in A_u \mid a = m\}$
22: $\quad\quad\quad$ $\boldsymbol{e}_i \leftarrow \boldsymbol{e}_q$
23: $\quad\quad$ **else**
24: $\quad\quad\quad$ **return** $\boldsymbol{s}_i; \psi_u; \boldsymbol{b}_\kappa; isresult$
25: $\quad\quad$ **end if**
26: $\quad$ **end while**
27: $\quad isresult \leftarrow$ true
28: **end if**
29: **return** $\boldsymbol{s}_i; \psi_u; \boldsymbol{b}_\kappa; isresult$

- $\boldsymbol{s}_i$ - A sequence of the found events, in temporal order.
- $\psi_u = (K_u, A_u)$ - An AASG among the ones stored in the system with a set of nodes K_u and a set of arcs A_u. It corresponds to the AASG used by the *morwi*.
- $\boldsymbol{b}_\kappa$ - A succession of nodes representing the branch of ψ_u traversed by the *morwi*.
- *isresult* - A boolean variable that is *true* only if the *morwi* has reached a node without children in ψ_u.

First of all, the variable *isresult* is set to $false$: it becomes true only when the *morwi* reaches a sink in the AASG. The AASG ψ_u is chosen in line 2 as that one whose root node κ_0 is matched by the event $\boldsymbol{e}_i$ ($M(\boldsymbol{e}_i, \kappa_0) = 1$). If no AASG is found, the

execution of Morwi is interrupted: the event does not represent the beginning of any of the cases proposed by the analyst.

There is another reason why a *morwi* can interrupt its trip: if the previous *morwi* generated by the system is similar to the new one, uses the same AASG and it was created less than a period T_{max} before. This interruption is introduced to avoid the creation of several *morwis* when there is a burst of very similar events, such as the ones present during a port scan. There are three elements referring to the last *morwi* that help us to set the conditions for this interruption:

- ψ_{last} - The AASG used by the last *morwi*.
- $\boldsymbol{e}_{last}$ - The event that caused the creation of the last *morwi*.
- Π_{last} - A function that is *true* if the current *morwi* is similar to the last *morwi*. This similarity can be defined by the atomic similarity functions listed in Table 3.3, page 52. In the implementation of Morwilog done for this thesis, we have considered two *morwis* as similar if the events at the origin of them share the type, the source IP address and the source port number. Assuming the value 1 as *true* and the value 0 as $false$, we can express the used function as $\Pi_{last} = f_1(type_i, type_{last}) \wedge f_1(ipsrc_i, ipsrc_{last}) \wedge f_1(psrc_i, psrc_{last})$.

The conditions for the interruption, in the same order as presented, can be then expressed as: $\Pi_{last} = true$, $\psi_u = \psi_{last}$ and $\Delta t_{i,last} < T_{max}$.

If the trip of the *morwi* is not interrupted for one of the two reasons stated above, Morwilog substitutes by this *morwi* the one considered as *last* at that moment. The *morwi* then starts going through the AASG ψ_u, storing $\boldsymbol{e}_i$ and κ_0 as first event and first node in the sequence, respectively (lines 4 to 6). The arcs going out from κ_0 are selected in F_n and an iterative process for traversing the AASG starts. The process only finishes if the *morwi* arrives at a sink in the AASG ($F_n = \varnothing$) or no event matching any of the children of the current node is found.

The set of found children is stored in a variable that we have called C. The process of finding an event $\boldsymbol{e}_q$ matching a children node κ_b (line 11) takes into account not only the matching condition itself ($M(\boldsymbol{e}_q, \kappa_b) = 1$), but also two temporal conditions:

(a) $\boldsymbol{e}_q$ should come later in time than $\boldsymbol{e}_i$ ($q > i$), something straightforward as we consider that elements in E_{in} are ordered in time.

(b) The time difference between $\boldsymbol{e}_q$ and $\boldsymbol{e}_i$ should not be superior to T_{max}. This condition is expressed by the function in Equation 3.7, page 53, as $f_1^t(\Delta t_{i,q}, T_{max}) = 1$.

Each found event is stored, together with the corresponding arc, in the set C (line 13). Have in mind that the Morwi algorithm only looks for the first of the events, $\boldsymbol{e}_q$,

matching the conditions defined by the node to which each arc leads, even if there are subsequent events that could be eligible.

If no event has been found, the *morwi* finishes its journey in line 24. If several events are found, a weighted random selection of one of the elements in C is made in line 17. The selection, called τ-based selection, is based on the level of pheromones of the arcs leading to the matched nodes. The probability of each node κ_c of been chosen is like the choosing probability defined in Equation 6.1 but only applied to the found children:

$$P(\kappa_c) = \frac{\tau_{n,c}}{\sum_{(\kappa_n \kappa_j, \boldsymbol{e}_c) \in C} \tau_{n,i}} \tag{6.7}$$

The selected event is directly added to the output sequence $\boldsymbol{s}_i$ and its corresponding node κ_c is added to $\boldsymbol{b}_\kappa$. The process is repeated from κ_c if it has further children (lines 18 to 21). The event $\boldsymbol{e}_q$ is preserved in line 22 because the matching operation $M(\boldsymbol{e}, \kappa)$ can depend on the previously found event, as we saw in section 5.2.3, if there are relative conditions in the abstract event contained in the node. If the *morwi* successfully arrives at a sink in the AASG, *isresult* becomes *true*.

The set of *morwis* is managed by the Morwihill algorithm, whose pseudocode is shown in Algorithm 6.2. Notice that every time an event $\boldsymbol{e}_i$ arrives at the system, a *morwi* is created (line 2) but as an independent thread. Morwihill has to be prepared to pass right after to new events just when they arrive, so a new *morwi* can be generated each time. For each event, the algorithm verifies if any *morwi* has finished its execution returning an event sequence that ends by $\boldsymbol{e}_i$ (line 3). If it is the case, Morwihill manages the results returned by all these *morwis*.

The boolean variable *isresult* is a way to tell the Morwihill algorithm to consider the returned sequence as matching one of the multi-step attack cases proposed by the analyst in an AASG. If the *morwi* properly finishes its journey through an AASG, reaching a node with no further children, a sequence of events matching one of the branches of that AASG is signaled to the analyst by raising an alarm (line 6). Depending on whether the analyst considers the sequence an attack or not, pheromones in the branch will be incremented (line 10) or decremented (line 12), respectively. In any of the two cases, all pheromones in the AASG are previously evaporated (line 7).

It is difficult to give an exact order of the computational complexity of Morwilog. There are too many factors involved to theoretically set the limiting behavior of the algorithm. Apart from the size of the dataset, it also depends on the structure of the AASGs or on the parameters of the system. We give then a list of dependencies related to the computational complexity of Morwilog:

Algorithm 6.2 Morwihill algorithm

Require: $E_{in} \subseteq \mathbb{E}$
1: **for each** $\boldsymbol{e}_i$ in E_{in} **do**
2: Create thread Morwi($E_{in}, \boldsymbol{e}_i$) ▷ Launch *morwi* μ_i
3: **for each** *morwi* μ_x | last element in $\boldsymbol{s}_x = \boldsymbol{e}_i$ **do**
4: $\{\boldsymbol{s}_x, \psi_u = (K_u, A_u), \boldsymbol{b}_\kappa, isresult\} \leftarrow$ result from μ_x
5: **if** $isresult =$ true **then**
6: Raise an alarm returning $\{\boldsymbol{s}_x, \psi_u = (K_u, A_u), \boldsymbol{b}_\kappa\}$
7: $\tau_{n,m} \leftarrow (1-\rho) \cdot \tau_{n,m} \,\forall\, \kappa_n\kappa_m \in A_u$ ▷ Evaporation of pheromones
8: **for each** $\kappa_n\kappa_m \in \boldsymbol{b}_\kappa$ **do**
9: **if** $\boldsymbol{s}_x$ is an attack **then**
10: $\tau_{n,m} \leftarrow \tau_{n,m} + \Delta\tau_0^+ \cdot exp(-(\tau_{n,m} - \tau_0)^2/2w^2)$
11: **else**
12: $\tau_{n,m} \leftarrow \tau_{n,m} - \Delta\tau_0^+ \cdot exp(-(\tau_{n,m} - \tau_0)^2/2w^2)$
13: **end if**
14: **end for**
15: **end if**
16: **end for**
17: **end for**

- The **number of created *morwis*** is equal to the number of incoming events, as each event generate a *morwi*.
- The **number of *morwis* that start to work with an AASG** is equal to the number of events matching a root node κ_0 of one of the AASGs used.
- The **number of comparisons between a node and an event** depends on several factors:
  - The maximum depth of the branch chosen by the *morwi* in the selected AASG.
  - The distance in terms of number of events between the events found for that specific branch.
  - The maximum search time T_{max}.
  - The number of children of each evaluated node.

Notice that there is an important assumption in the functioning of this algorithm: the response of the analyst is considered to be immediate after an alarm is raised. This allows a better understanding of how the algorithm works and eases the execution of the experiments. However, we know that this is never the case in a real security system, where the alarms accumulate in a console until the analyst can verify and acknowledge them. Morwilog would work exactly in the same way if we hold the generated alarms in a queue and pheromones are modified once the analyst has been able to confirm or

refuse the returned sequences as attacks. The only constraint in this case is that if an event arrives and matches an AASG whose pheromones are pending to be updated, the new *morwi* will go through a non-updated version of the AASG.

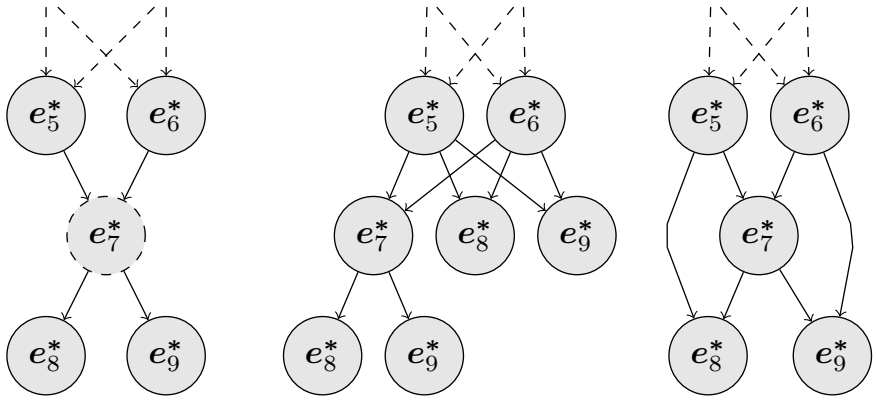


Figure 6.6: Two alternative transformations of (a) an AASG excerpt to express the optionality of node containing e_7^*: (b) duplicating the nodes to preserve the uniqueness of depth or (c) adding additional arcs to get the representation for Morwilog.

To simplify the explanation, the description of Morwilog given so far does not consider either *counters* nor *optional nodes*, presented in section 5.2.6. When a *morwi* finds a node with a counter higher than 1 among the children of a node, it will simply repeat the process of looking for an event matching the node as many times as the counter indicates. The maximum difference of time T_{max} is then considered between each one of repeated matches. If every repetition is found, the node is considered to be 'matched' and the first of the events in the sequence is returned as part of $\boldsymbol{s}_i$.

Optional nodes are considered when loading new AASGs, before the execution of the algorithm. Each AASG with optional nodes is transformed into a new structure where the number of branches is augmented to reflect the facultative occurrence of the optional nodes. This new structure is an AASG except for one detail: the fourth property of AASGs (page 117) is not preserved, meaning that not all the nodes have a unique depth. We choose to do so to simplify the execution of the algorithm. An alternative AASG that preserves this property could be easily built duplicating the nodes, but that would force the algorithm to consider them as distinct nodes, which could be confusing for the analyst that has designed the AASG. In Figure 6.6 we see how the lower part of the AASG coming from Figure 5.8 is transformed under both approaches: option a) maintains the fourth property of AASGs duplicating the nodes

and option b) does not, but it eases the implementation in Morwilog.

6.2 Bidimac

The second of the proposed algorithms to perform identification and detection using AASGs is Bidimac. Its details have not been published yet. Bidimac is based on Bayesian inference, a type of statistical inference. In section 6.2.1, we explain how Bayesian inference works and introduce several assumptions to adapt Bayesian networks to AASGs. We have chosen a Bayesian approach for Bidimac because of its similarities with Morwilog. These similarities, reviewed in section 6.2.2, are at the basis of choosing Bayesian inference as an alternative to Morwilog, so it is important to understand them before explaining the Bidimac algorithm in section 6.2.3. Bidimac is more than another algorithm to work with AASGs, but also a way to compare Morwilog with the widely used Bayesian inference.

6.2.1 Adapting Bayesian inference to AASG

Bayesian inference is a type of statistical inference based on the use of the Bayes' theorem. We already introduced the concept of statistical inference in section 4.3.2.2, page 80. In Computer Science it is also referred to as 'learning'.

Bayesian networks are the support used for this kind of inference. A Bayesian network is a model that represents a set of probability-based relationships between n random variables [Husmeier 2005]. It is composed of a DAG, with the nodes representing the variables and a set of probability distributions associated to each variable with regard to its parents [Koski 2011]. Two nodes are connected if the model considers a causal relationship between them. Conversely, two variables that are not connected by any arc are conditionally independent of each other [Stephenson 2000]. Figure 6.7 shows an example of a Bayesian network. In this network, variables X_2 and X_1, for example, are conditionally independent, while X_0 and X_4 are not.

We will be working only with Bayesian networks in which every random variable X_i has the same sample space. Moreover, they will be discrete variables, having a finite set of possible values x_k associated to them. Notation is remarkably simplified under this scope. We can define a *probability parameter* θ_{ijk}, which represents the probability that $X_i = x_k$ given that X_j is the parent. In other words:

$$\theta_{ijk} \propto P(X_i = x_k | X_j) \tag{6.8}$$

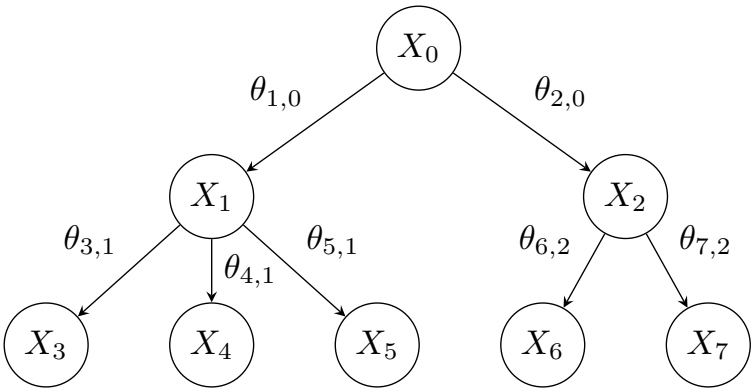


Figure 6.7: Example of a Bayesian network.

This represents the probability of what we call an ijk *case*, an instance of a variable X_i associated to a parent j with a value corresponding to k. If the sample space of the variables has only two elements ($\{x_1, x_2\}$), we can select one of them (e.g. x_1) to use it as a reference and simplify notation, removing the k from the probability parameter:

$$\theta_{ij} \propto P(X_i = x_1 | X_j) \tag{6.9}$$

The information about the transition become complete if we assume that $\theta_{ij} = P(X_i = x_1|X_j) = 1 - \theta_{ij2} = 1 - P(X_i = x_2|X_j)$.

Each Bayesian network has an associated joint probability function. Given that the network is composed of the set of n variables $\boldsymbol{X} = X_0, \ldots, X_n$, the probability $P(\boldsymbol{X})$ can be calculated as the product of every conditional probability between pairs parent-children [Stephenson 2000, Jensen 2007]:

$$P(\boldsymbol{X}) = \prod_{i=0}^{n} P(X_i|\text{parents}(X_i)) \tag{6.10}$$

Bayesian inference always needs a starting point, a previous definition of the model. It does not matter if the structure of the network is manually built or if it is learned from a set of training data [Husmeier 2005], but prior probabilities are needed to perform the induction. These probabilities thus represent a degree of *belief* of each causal relationship in the network, which is updated as more information about the modeled process is obtained.

Different methods exist to do Bayesian learning from new data D, depending on how complete is the network topology and the used data. In the context of Bidimac, we already have the Bayesian network, which is an AASG previously defined by the

security analyst (see section 6.2.3). We just need to learn the parameters or beliefs. Concerning the data, we assume that it is fully observable. In consequence, we just need to focus on the simplest case [Cao 2014]: Bayesian learning with known network structure and full observability.

Heckerman derives a formula for this Bayesian estimation assuming that the prior probabilities are represented by Dirichlet distributions. The reader is referred to Heckerman's technical report [Heckerman 1995] to get details about the deduction. The result is summed up by Stephenson [Stephenson 2000] as:

$$\theta_{ijk} = \frac{\alpha_{ijk} + N_{ijk}}{\alpha_{ij} + N_{ij}} \tag{6.11}$$

Here, α_{ijk} represents the number of times that a certain ijk case appears in the learning dataset D. N_{ijk} represents the prior belief about how many times the ijk case would occur, assuming D is unknown. α_{ij} and N_{ij} are equivalent to the previous ones but adding up all the occurrences in every possible instantiation k: $\alpha_{ij} = \sum_{\forall k} \alpha_{ijk}$ and $N_{ij} = \sum_{\forall k} N_{ijk}$.

However, it is not very intuitive to express prior beliefs as the number of times an event could happen. We always tend to directly assign a probability to uncertain events. The prediction "there is a 30% chance of rain" is more understandable than "if the day of today was repeated 100 times, we would have rain in 30 of them". We can express the probability parameters in terms of the prior estimation without specifying the estimated number of times for each ijk case. To represent the importance of prior estimations against new ones, we can use a learning rate $0 < \eta \leq 1$, as the one used by Qin and Lee [Qin 2007]. The expression to derive the new probability parameter θ^t_{ijk} from the previous one θ^{t-1}_{ijk} is now:

$$\theta^t_{ijk} = \eta \frac{N_{ijk}}{N_{ij}} + (1 - \eta)\theta^{t-1}_{ijk} \tag{6.12}$$

The main difference between the two approaches is that if we use equation 6.11 the size of the dataset D used is taken into account for each estimation. If D is bigger, the statistical update is supposed to be more reliable and the prior belief is less important. This approach is thus very convenient if we update the network with datasets of different sizes. Another advantage of this method is that θ^t_{ijk} exactly corresponds to the probability of the transition.

On the contrary, using equation 6.12, we give the same importance to any new dataset, independently of its size. The statistical relevance of the dataset is thus not considered. However, this second approach is more advantageous if the datasets used

have more or less the same size, as it happens if we want to update the parameters in real time, when the size of D is 1 if we update sample by sample. Moreover, instead of having an input parameter α_{ijk} for each ijk case, we only have the learning rate η. We lose in statistical accuracy but we gain in simplicity of computation and number of parameters.

6.2.2 Morwilog and Bayesian inference

The reader may see a certain parallelism between Morwilog and methods doing Bayesian inference. This resemblance indeed exists, even if classic ACO, in which Morwilog is based, is not directly comparable with Bayesian inference. ACO and Bayesian inference are applied for different purposes: while the former is a metaheuristic for optimization, the latter is a method for statistical inference. That does not mean that they cannot complement each other. As ACO is used for optimization, it can be adapted to find the best possible Bayesian model for a giving problem. In general terms, the two elements defining a Bayesian network are the structure of the DAG representing it and the probabilities or beliefs assigned to each node. For the second one there exist several algorithms for Bayesian inference, but none of them uses, as far as we know, the ACO metaheuristic. However, some work exists in the literature about how to learn the structure of the Bayesian network using ACO [Daly 2006, Wu 2010, Salama 2013].

Even if Morwilog is ACO-based, it does not work as classic ACO, and the comparison with Bayesian is more than pertinent. The objective of Morwilog is not to look for an optimal solution among a set of possible solutions, but to give the security analyst the events corresponding to a multi-step attack case and to modify the AASG according to the analyst's verdict. The result after several iterations is finally similar to that of Bayesian inference methods, a DAG whose transitions have an associated probability, coded as artificial pheromones. Morwilog is comparable to Bayesian inference if we follow the general definition given by Dempster [Dempster 2008]. The same as Bayesian inference, it starts with a global probability distribution for all the sequences (initialization of pheromones) and deduces the conditional dependences between the events after verification by the analyst. A direct comparison between the two techniques reveals then necessary to define the place of Morwilog among data analysis methods.

The similarities between Morwilog and Bayesian inference are listed below:

- **Bayesian logic**. Both of them follow the Bayesian logic: the probabilities are defined *a priori* based on personal belief and evolve based on the incoming data. In Morwilog, initial beliefs, represented as pheromones, are the same for every

transition.

- **DAG**. They use Directed Acyclic Graphs (DAG) to represent the relationships between different pieces of data. In the implementation presented in this thesis, Bayesian inference is adapted to work with AASGs.
- **Probabilities of transition**. Nodes in the DAG are linked by numbers suggesting the probability of transition. In the case of Morwilog, these numbers are the artificial pheromones.
- **Statistical information**. The resultant graph gives information about the statistical distribution of the events associated to each node, after the adaptation of the numbers associated to each transition to new data.

We will see in section 6.3, after presentation of the Bidimac algorithm, that there are also many differences between the two approaches.

6.2.3 The Bidimac algorithm

We adapt the Bayesian inference mechanism to work towards the same goal as Morwilog, using the AASGs as Bayesian networks. The resulting algorithm is called Bidimac, that stands for Bayesian Inference for Detection and Identification of Multi-step Attack Cases. Some requirements have to be taken into account:

1. Each one of the arcs $\kappa_n\kappa_m$ in the AASG should have a probability parameter $\theta_{n,m}$ associated to it, in the same way as the level of pheromones in Morwilog.

2. Feedback from the security analyst has to be incorporated into the system, so the probability parameters can evolve according to her verdict.

3. The AASG has to be updated in real time, as the feedback from the analyst is received.

The first requirement is easy to be fulfilled, just by substitution of the level of pheromones $\tau_{n,m}$ in the stigmergic AASG by the probability parameter $\theta_{n,m}$. It should represent the probability of κ_n being matched by an event e_i and κ_m being matched by an event e_j given that e_j is immediately following e_i in a sequence of events that a) entirely matches event by event a branch in the AASG and b) is considered a multi-step attack after verification by the analyst. The resulting AASG is called a *Bayesian AASG*. In the JSON format of the AASG, $\theta_{n,m}$ is incorporated into each set of arcs as the numeric member `prob` inside the objects in `children` (see page 130).

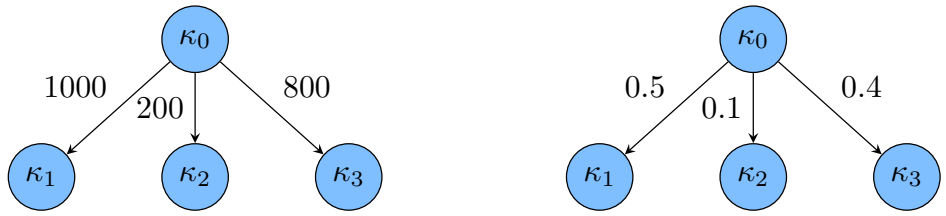
The probability parameters are equivalent to the choosing probability defined in Equation 6.1 for the pheromones. This gives us a way for weight comparison between stigmergic AASGs (pheromones) and Bayesian AASGs (probability parameters):

$$P_{n,m} = P(\kappa_n \kappa_m) = \frac{\theta_{n,m}}{\sum_{\kappa_a \kappa_b \in A | a=n} \theta_{a,b}} = \theta_{n,m} \tag{6.13}$$

For the initialization of the stigmergic AASG, the same level of pheromones τ_0 is assigned to every arc. This result in the same value of $P_{n,m}$ for all the arcs leading to a set of siblings. In other words, right after initialization of the stigmergic AASG, $\tau_{n,m} = \tau_0 \forall \kappa_n \kappa_m \in A$ and thus $P_{n,m} = P_{o,p} \forall P_{n,m}, P_{o,p} \mid n = o$.

In the initialization of a Bayesian AASG, we preserve the same property by assigning the same probability parameter to every arc going out from a parent and establishing that the addition of the elements in this set should be equal to 1. In fact, what we do when initializing a Bayesian AASG is to take a newly created stimergic tree and to change the level of pheromones $\tau_{n,m}$ to $\theta_{n,m} = P_{n,m}$ as defined in Equation 6.1.

This equivalence between initial value of $\theta_{n,m}$ in the Bayesian AASG and the choosing probability $P_{n,m}$ can also be used in the adaptation of a stigmergic AASG where there have already been pheromone updates. An example of this is shown in Figure 6.8, where the total number of pheromones in all the siblings $(1000 + 200 + 800 = 2000)$ becomes the denominator to do the normalization and $\theta_{1,0} = P_{1,0} = \frac{1000}{2000} = 0.5$, $\theta_{2,0} = P_{2,0} = \frac{200}{2000} = 0.1$ and $\theta_{3,0} = P_{3,0} = \frac{800}{2000} = 0.4$.



(a) Stigmergic AASG (pheromones) (b) Bayesian AASG (prob. parameters)

Figure 6.8: Equivalence between a stigmergic and a Bayesian AASG.

The second requirement is the incorporation of feedback from the analyst. This is done by sending an alarm to the analyst each time that a sequence of events matches a branch in the Bayesian AASG and updating the probability parameters only when that sequence is confirmed as an attack. As in Morwilog, we only consider the sequences whose consecutive events have a difference in time not bigger than T_{max}.

Finally, to fulfill the third requirement, addressing real time update, we just need to consider that the learning dataset D contains only one element each time an update is made, so $\frac{N_{ijk}}{N_{ij}} = 1$ in Equation 6.12. Sequences matching a branch are evaluated by the analyst one by one. Once she considers that the returned sequence is an attack, the update of the probability parameters is done according to the following formula:

$$\theta^t_{n,m} = \begin{cases} \eta + (1-\eta)\theta^{t-1}_{n,m} & \text{if } \kappa_n\kappa_m \text{belong to the matched branch} \\ (1-\eta)\theta^{t-1}_{n,m} & \text{otherwise} \end{cases} \tag{6.14}$$

That means that only those arcs in the matched branch are reinforced in terms of their probability parameters, while all the others are penalized. The statistical coherence with respect to the incoming data is then preserved, and no specific penalization mechanism is considered when a false positive is found. As the Bayesian AASG does not reflect all the possible relationships of causality between the events, but only those contained in the hypotheses formulated by the analyst, sequences that are not an attack do not contribute to the update of the AASG. Penalization of branches not representing an attack is indirectly done when an attack matching any other branch is found.

The Bidimac algorithm is represented in the form of pseudocode in Algorithm 6.3. The subroutine to search for new matched branches from each given event $\boldsymbol{e}_i$, called BranchesSearch, is shown in Algorithm 6.4. Notice that oppositely as it happens in the Morwi algorithm, this routine can return several matches, corresponding to different branches of the selected AASG. In this way, the Bayesian AASG fully represent the statistics of the incoming events. We consider only the first match when looking for events corresponding to the nodes in the branch (line 8 in Algorithm 6.4).

The same as it happens with Morwilog, it is difficult to precise the complexity of the algorithm because it depends on many factors. All the dependencies mentioned for Morwilog are totally applicable to Bidimac if we substitute the word '*morwi*' by 'search process' in the list of page 149. However, there is an additional characteristic of the AASGs that directly affects the complexity of Bidimac: the number of branches. Each time an event matches the root node of an AASG, Bidimac searches for events matching each one of the branches, while Morwilog chooses just one. The number of searched sequences in Bidimac for a given AASG is thus proportional to the number of branches.

Algorithm 6.3 Bidimac algorithm

Require: $E_{in} \subseteq \mathbb{E}$
1: **for each** $\boldsymbol{e}_i$ in E_{in} **do**
2: Create thread BranchesSearch$(E_{in}, \boldsymbol{e}_i)^*$ ▷ Launch search ν_i
3: **for each** element $r_x \in R_{found}$ | last element in $\boldsymbol{s}_x = \boldsymbol{e}_i$ **do**
4: $(\boldsymbol{s}_x, \psi_u = (K_u, A_u), \boldsymbol{b}_\kappa, isresult) \leftarrow r_x$
5: **if** $isresult =$ true **then**
6: Raise an alarm returning $\{\boldsymbol{s}_x, \psi_u = (K_u, A_u), \boldsymbol{b}_\kappa\}$
7: **if** $\boldsymbol{s}_x$ is an attack **then**
8: **for each** $\kappa_n\kappa_m \in A_u$ **do** ▷ Update probability parameters
9: **if** $\kappa_n\kappa_m \in \boldsymbol{b}_\kappa$ **then**
10: $\theta_{n,m} \leftarrow \eta + (1-\eta)\theta_{n,m}$
11: **else**
12: $\theta_{n,m} \leftarrow (1-\eta)\theta_{n,m}$
13: **end if**
14: **end for**
15: **end if**
16: **end if**
17: **end for**
18: **end for**

*When the execution finishes, the returned set of elements is stocked in R_{found}

6.3 Differences between Morwilog and Bidimac

There are some characteristics of Morwilog and Bidimac that make them very different, even if they share a similar approach towards the use of AASGs. These differences lie in the following elements:

- **Choice among options**. In Morwilog, each *morwi* evaluates the nodes in the AASG step by step, choosing only one path among the ones found. When more than one sequences of events match an AASG, just one of them is chosen to be evaluated by the security analyst. In Bidimac all the found sequences are returned to the analyst.
- **Statistical meaning of weights**. The level of pheromones does not correspond to the probability of appearance of a sequence, even if some information in this regard can be deduced from its value. It indicates the probability of a node to be chosen against its found siblings as the following element in the path. In Bidimac, the probability parameters directly represent the transition probability between nodes according to the existence of pairs of events matching the nodes.
- **Mechanism to penalize or reinforce paths**. The mechanism for incrementing or decrementing the level of pheromones can be implemented by any kind

Algorithm 6.4 BranchesSearch algorithm

Require: $E_{in} \subseteq \mathbb{E}$; $\boldsymbol{e}_i \in E_{in}$
Ensure: Set R_x of 4-tuples with the form ($\boldsymbol{s}_i \in S_{E_{in}}$, $\psi_u = (K_u, A_u) \in \Psi$, $\boldsymbol{b}_\kappa \in B_{\psi_u}$, $isresult \in \{\text{true, false}\}$)

$\psi_u \leftarrow \psi \in \Psi \mid M(\boldsymbol{e}_i, \kappa_0) = 1$
$R_x \leftarrow \varnothing$
if $\psi_u \neq \varnothing$ **then**
  for each branch $\boldsymbol{b}_\kappa \in \psi_u$ **do**
    $\boldsymbol{s}_i \leftarrow [\boldsymbol{e}_i]$
    $isresult \leftarrow$ false
    for each node $\kappa_a \neq \kappa_0 \in \boldsymbol{b}_\kappa$, taken in order **do**
      Find $1^{st}\, \boldsymbol{e}_q \in E_{in} \mid q > i, M(\boldsymbol{e}_q, \kappa_a) = 1, f_1^t(\Delta t_{i,q}, T_{max}) = 1$
      if $\exists\, \boldsymbol{e}_q$ **then**
        Add $\boldsymbol{e}_q$ to $\boldsymbol{s}_i$
        if $\nexists \kappa_x \kappa_y \in A_u \mid x = a$ **then** ▷ κ_a has no children
          $isresult \leftarrow$ true
        end if
      else
        break
      end if
    end for
    Add ($\boldsymbol{s}_i$, ψ_u, $\boldsymbol{b}_\kappa$, $isresult$) to R_x
  end for
end if
return R_x

of function, while in Bidimac the penalization or reinforcement of a path only depends on statistical evidence.

- **Decrement of choosing probability**. Morwilog incorporates two mechanisms for decrementing the choosing probability: evaporation and penalization. Bidimac does not incorporate a mechanism of explicit penalization for found sequences that are not attacks.

In this section we present each one of the listed differences and their connotations.

6.3.1 Choice among options

In both algorithms, events matching each node in the AASG are searched sequentially, considering a maximum difference of time T_{max} between consecutive events. This T_{max} should be long enough to ensure that all the events matching each set of children are found, but not so long as the resources of the system in terms of number of processes are fully used.

In Morwilog, once all the children have been found or no more events fulfill the condition of time difference, a weighted random choice is made based on the level of pheromones assigned to each arc. This process is also called a τ-based selection (see page 149). For every event matching the root node κ_0 of an AASG, only one branch is followed, even if other sequences that start with the same event could match another branch in the same AASG.

Conversely, in Bidimac there are no options. Bayesian inference is intended to reflect the statistics of the incoming dataset, so it takes into account all the sequences matching the AASG and representing a multi-step attack, even if some of them have events in common. Every matching sequence is presented to the analyst for evaluation.

The consequence of this is that, in general, Bidimac returns more sequences for verification (alarms) than Morwilog, thereby demanding more effort to the security analyst. The number of sequences is exactly the same if no sequence matching the AASG has an event in common with the others, considering the limit of time of T_{max}. In this case there is no election of choices in Morwilog because there are no choices, and both approaches return the same sequences.

Figure 6.9 shows an example of the evolution of two equivalent AASGs under the two approaches. We have chosen a very simple AASG, with a root node κ_0 that has two children κ_1 and κ_2. For the evolution of the pheromones, we use the parameters presented in Table 6.1. To update the probability parameters in Bidimac, we use a learning rate $\eta = 0.4$. The events involved in the evolution of the AASG are shaded in dark blue.

Suppose that in a first phase an event matching κ_0 is found, followed by two other events matching κ_1 and κ_2 with a difference of time with the first one inferior than T_{max}. Both sequences represent an attack. When this happens in Morwilog, the *morwi* chooses just one branch. We can then consider two alternative cases: one in which κ_1 is chosen (Case A) and a second one in which κ_2 is chosen (Case B). The chosen sequence is sent to the analyst for verification. In Bidimac, both sequences generate alarms because no choice process exists.

We also represent in the right of the figure the effect on the AASGs of a second phase in which another sequence of events matching κ_0, κ_1 and representing an attack arrives to the system. As there is only one sequence matching the AASG and no event is found matching κ_2, there is not a choice process in Morwilog and both approaches, Morwilog and Bidimac, return just one alarm.

Concerning Morwilog, Case A is the most probable in terms of the τ-based selection. It is evident how the presence of a choice makes the result from Morwilog different than

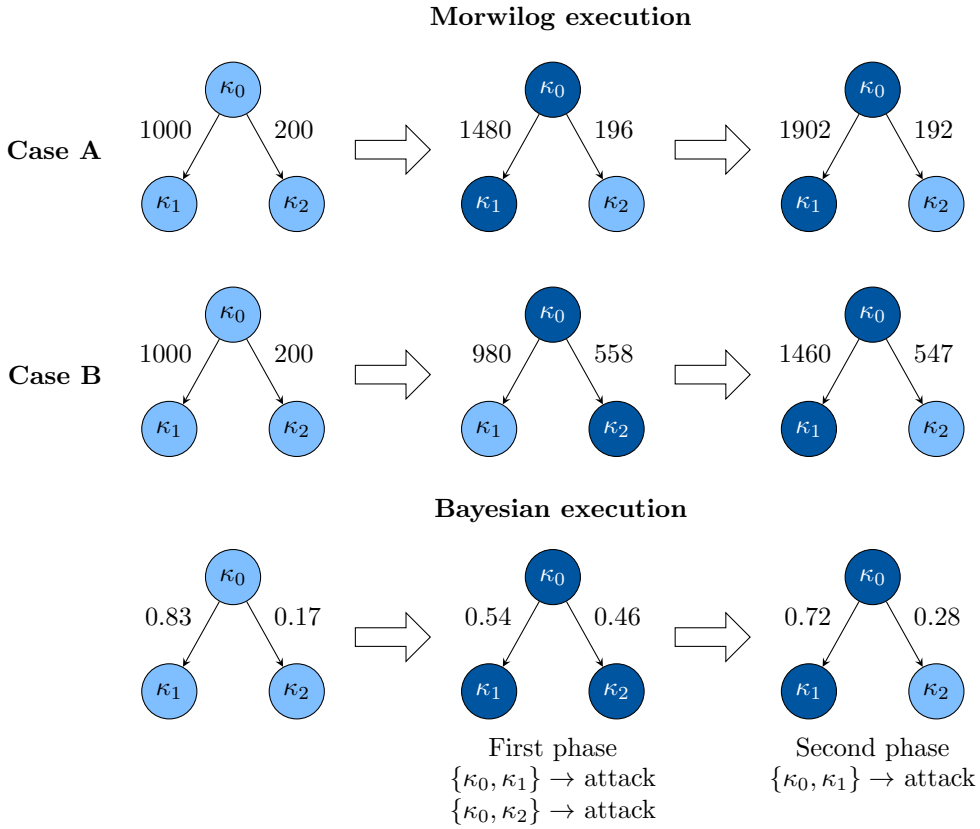


Figure 6.9: Example of Morwilog (parameters from Table 6.1) and Bidimac ($\eta = 0.4$) with a simple AASG.

the one from Bidimac. The security analyst reviews just 2 sequences with Morwilog, while using Bidimac she checks 3.

To compare the resulting AASGs, we need to calculate the choosing probability for the two branches. If we focus on Case A in Morwilog, $P_{0,1} = \frac{1902}{2094} = 0.91$ and $P_{0,2} = \frac{192}{2094} = 0.09$. Note that this does not correspond at all to the probabilities found using Bidimac, where the equality $P_{n,m} = \theta_{n,m}$ is preserved.

6.3.2 Statistical meaning of weights

The choice mechanism in Morwilog does not allow finding a direct correspondence between the probability of occurrence of one attack sequence and the level of pheromones. The level of pheromones indicates the probability of choosing one arc against the others when events matching several children nodes are found, based on the previous history of events. But this history is not deterministically built from the observed data. There is a random factor in the increment or decrement of pheromones coming

from the choice made in each one of the nodes.

Apart from this randomness, another reason why the level of pheromones does not directly correspond to the statistics of the events is given by the evaporation. We have seen in section 6.1.3 that the mechanism of pheromone evaporation avoids the stagnation of the choice in one branch of the AASG. The consequence is a reduction in the number of pheromones. This reduction is not linked to the statistical distribution of the incoming events, as it is done in every arc and not just in those connecting the matched nodes.

Due to this two factors, the level of pheromones is not a direct translation from statistical evidence, even if an intuition about which is the most probable path representing a multi-step attack can still be grasped from the pheromones. Pheromones are conceived as a resource for Morwilog, not as a numerical result, independently that they give valid information about correct multi-step attack cases among the ones proposed in the AASG. On the contrary, the probability parameters used in Bidimac directly code the probability of a sequence to be an attack, as there is neither a phase of random choice of paths nor of evaporation.

Going back to Figure 6.9, presented in section 6.3.1, the stigmergic AASG in Case A and Case B has in the last phase different choosing probabilities: $P_{0,1} = 0.91$ and $P_{0,2} = 0.09$ for Case A, and $P_{0,1} = \frac{1460}{2007} = 0.73$ and $P_{0,2} = \frac{547}{2007} = 0.27$ for Case B. Being the input dataset identical in both cases, this difference is due to the random choice and the evaporation mechanism. In contrast to Bidimac, choosing probabilities do not necessarily correspond to the statistical distribution of the dataset.

6.3.3 Mechanism to penalize or reinforce paths

The idea behind both Morwilog and Bidimac is to reinforce the branches in the AASG representing multi-step attacks, thus confirming the hypotheses, and to penalize the branches not corresponding to attacks, thus refusing false positives. However, the way of doing so is very different: in Morwilog, the increment or decrement is made by an arbitrary function while in Bidimac, the change of probabilities always follows the statistics of the events.

In other words, when using Bidimac, the increment of probability corresponds to an increment in the number of sequences representing an attack. This entails a linear change in the probability parameters, always controlled by the learning rate η. However, the evolution of pheromones in Morwilog can be implemented using any function. In our case, we have chosen a Gaussian function (see equations 6.5 and 6.6) to get a higher initial change in pheromones than in the linear formula used in classic

ACO, but we could have chosen any kind of function. This gives freedom to set the maximum attainable value for the choosing probability at any level.

To compare the evolution of the choosing probabilities both under Morwilog and Bidimac, we consider the scenario of continuous reinforcement introduced in section 6.1.3, page 144. The AASG chosen is the simple one presented in Figure 6.9 but right after the initialization, with $\tau_{0,1} = \tau_{0,2} = \tau_0 = 1000$ for the stigmergic AASG and $\theta_{0,1} = \theta_{0,2} = 0.5$ for the Bayesian AASG. To set a continuous reinforcement, we imagine that the incoming data contains instances of $\{\kappa_0, \kappa_1\}$ confirmed as attacks but no instances of $\{\kappa_0, \kappa_2\}$. This situation is represented in Figure 6.10.

Table 6.3 contains the evolution of the elements in the continuous reinforcement scenario: the pheromones and the choosing probability $P_{0,1}$ for Morwilog, and the probability parameters for Bidimac. Morwilog takes the parameters from Table 6.1. Bidimac is executed with $\eta = 0.4$ and $\eta = 0.6$. The choosing probabilities are represented in the graph of Figure 6.11 (remember that $\theta_{0,1} = P_{0,1}$).

We can observe that in Bidimac, the continuous reinforcement scenario brings a convergence of $P_{0,1}$ to 1 whose speed depends on the value of η. On the contrary, in the case of Morwilog, a maximum and a minimum level of pheromones are eventually reached. In this case, the upper limit is 3051 pheromones (see page 144) and the minimum level is defined by $\tau_{min} = 100$. When doing continuous reinforcement, we have a maximum choosing probability $P_{0,1} = \frac{3051}{3151} = 0.97$ when no further pheromones

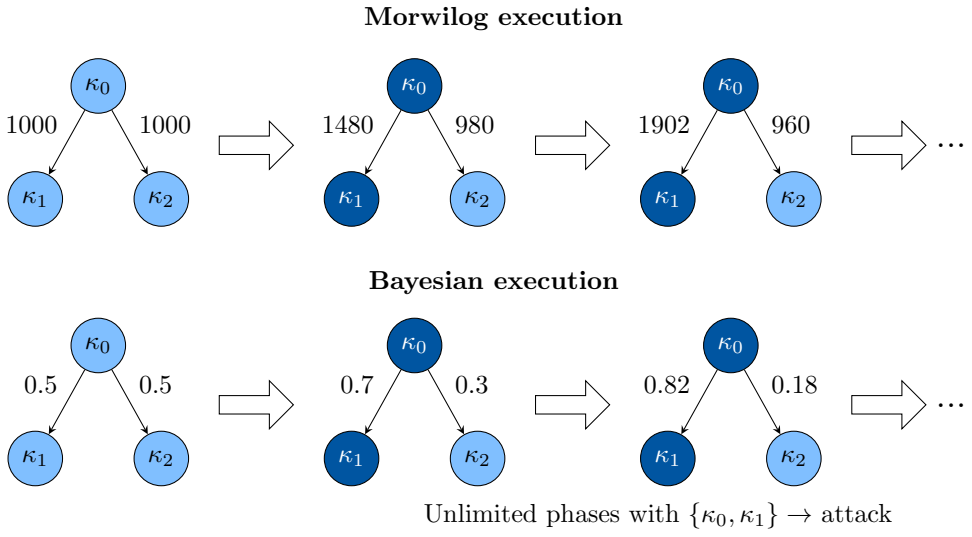


Figure 6.10: Example of continuous reinforcement in Morwilog (parameters from Table 6.1) and Bidimac ($\eta = 0.4$) with a simple AASG

#	Morwilog			Bay. $\eta = 0.4$		Bay. $\eta = 0.6$	
	$\tau_{0,1}$	$\tau_{0,2}$	$P_{0,1}$	$\theta_{0,1}$	$\theta_{0,2}$	$\theta_{0,1}$	$\theta_{0,2}$
0	1000	1000	0.50	0.50	0.50	0.50	0.50
1	1479.9	980	0.60	0.70	0.30	0.80	0.20
2	1902.1	960.4	0.66	0.82	0.18	0.92	0.08
3	2208.3	941.2	0.70	0.89	0.11	0.97	0.03
4	2418	922.4	0.72	0.94	0.06	0.99	0.01
5	2565.4	903.9	0.74	0.96	0.04	0.99	0.01
6	2673	885.8	0.75	0.98	0.02	1	0
7	2754.2	868.1	0.76	0.99	0.01	1	0
8	2817.2	850.8	0.77	0.99	0.01	1	0
9	2867	833.7	0.77	0.99	0.01	1	0
10	2906.9	817.1	0.78	1	0	1	0

Table 6.3: Pheromones and probability parameters for the case of continuous reinforcement (parameters of Morwilog from Table 6.1)

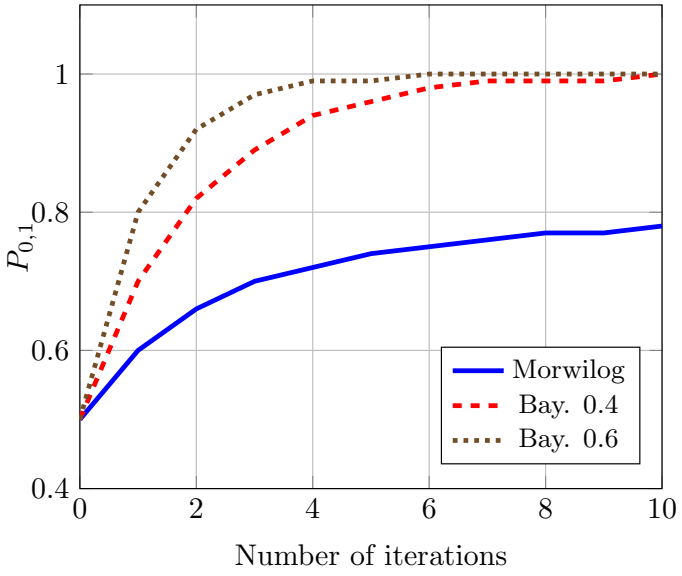


Figure 6.11: Evolution of choosing probability $P_{0,1}$ in continuous reinforcement (example from Table 6.3).

are accumulated in the arc $\kappa_0\kappa_1$ and the evaporation brings $\tau_{0,2}$ to $\tau_{min} = 100$. This probability is high enough to be comparable to 1, but Morwilog takes a lot of iterations to get to this value. In Table 6.3, we see that in 10 iterations the maximum level reached is $P_{0,1} = 0.78$.

This fast convergence seems an advantage of Bidimac with respect to Morwilog. The sooner a branch gets a high choosing probability, the sooner we can confirm it as a correct hypothesis. However, there is a subtle reason to not want a too fast convergence in this scenario: no sequence of events matching $\{\kappa_0, \kappa_2\}$ has been found. What if this sequence is just rarer but it also represents the attack? Morwilog is able

to distinguish between not found sequences and found ones that are not evaluated as attacks after verification, as we explain in the following subsection.

6.3.4 Decrement of choosing probability

In Morwilog, not found sequences and found sequences that are not attacks are not evaluated in the same way. For the branches corresponding to the first ones, evaporation guarantees a slow diminution in the choosing probability. On the contrary, for found sequences that are not attacks, the mechanism of pheromone decrement (Equation 6.6) makes the choosing probability of the matched branches to rapidly decrease.

The existence of these two separated mechanisms in Morwilog is necessary because of the decision process performed by the *morwis*. Pheromone-based elections are made when several sequences starting with the same event match different branches in the AASG. Paths involving not found sequences have to be kept with a level of pheromones high enough to allow its selection in case of the eventual appearance of events corresponding to these paths.

On the other hand, when applying Bidimac, the decrement of the choosing probability is made only responding to the statistics of found sequences that are evaluated as attacks. There is no additional mechanism to penalize false positives. The possibility of implementing a mechanism of this sort should be discarded if we want to preserve the correspondence of the probability parameters to the statistics of the sequences represented in the AASG.

6.4 Summary

This chapter has been devoted to present Morwilog and Bidimac, the algorithms we propose to exploit the AASG. They perform the identification of the correct multi-step attack case among the alternative ones proposed by the security analyst. At the same time, both are able to perform detection of matched cases, returning alarms to the analyst.

We have first presented Morwilog, that is based on the behavior of foraging ants. After presenting Ant Colony Optimization in section 6.1.1 and a general overview of the algorithm in section 6.1.2, we have explained in section 6.1.3 how an adapted AASG becomes the stigmergic scenario where the artificial ants, called *morwis*, build their trails. The Morwilog algorithm is presented in detail in section 6.1.4. For the presentation of Bidimac, we have first devoted section 6.2.1 to explain Bayesian inference, on which the algorithm is based. In section 6.2.2, the parallelism between

Morwilog and Bayesian inference has been analyzed. The Bidimac algorithm itself has been then explained in section 6.2.3. Once the two algorithms have been explained, they have been compared to understand their differences: the choice among options (section 6.3.1), the statistical meaning of weights (6.3.2), the mechanism to penalize or reinforce paths (6.3.3) and the decrement of choosing probability (6.3.4). In the next chapter, we will proceed to evaluate Morwilog and Bidimac with examples of AASGs.

Chapter 7

Evaluation

Contents

7.1 Introducing the evaluation datasets 169
7.2 Evaluation of different AASGs 177
7.3 Evaluation of the algorithms 191
7.4 Summary 200

> *"Ciertamente, no lo sé. En materia de conocimiento he tratado siempre de limitarme al terreno de mis experiencias."*
> ["Certainly, I do not know. In matters of knowledge, I have always tried to limit myself to the realm of experience."]
> — Juan Benet, *Return to Región*

We have seen how an analyst can capture her hypotheses about multi-step attacks in an AASG, which can later be used to perform detection and identification of correct cases using Morwilog and Bidimac. In this chapter, we perform a set of experiments to evaluate both the AASGs and the algorithms. All the experiments are carried out on an Intel Core i5 machine running at 1.4 GHz with 8 GB RAM. The parameters used in Morwilog take the values listed in Table 6.1, page 139, unless stated otherwise. The exception is T_{max}, whose value depends on the dataset used in each occasion. In the same way, the learning rate in Bidimac takes the value $\eta = 0.4$ by default.

This chapter starts with the presentation of the datasets used for the experiments in section 7.1. Then, evaluation is divided into two parts. In the first one, addressed in section 7.2, some examples of AASG are proposed and evaluated in different datasets using Morwilog and Bidimac. The second part, presented in section 7.3, is focused on the evaluation of different aspects of these algorithms.

7.1 Introducing the evaluation datasets

Several datasets have been used in the evaluation of AASGs and the algorithms working with them. In this section, we introduce these datasets, their composition and the way

they need to be preprocessed to be used for the proposed evaluations: DARPA 2000 Intrusion Detection Scenario Specific Data Sets, abbreviated as DARPA 2000 (section 7.1.1); ISCX 2012 Intrusion Detection Evaluation Data Set, abbreviated as ISCX (7.1.2), and the one generated within the HuMa project, called the HuMa dataset (7.1.3). The first two are publicly available, but the HuMa dataset is private and cannot be disclosed due to restrictions on the project. We then present in section 7.1.4 the common format into which we parse the events coming from the three datasets. Finally, a new set of simulated datasets, the *eventgen* datasets, is presented in section 7.1.5. It is created to compensate for the lack of multiplicity of multi-step attacks in the existent datasets.

7.1.1 DARPA 2000 datasets

DARPA 2000 is actually composed of two datasets, one containing the attack LLDoS 1.0 and the other, the LLDoS 2.0.2 (see section 3.2.2, page 31). They were created by the MIT Lincoln Laboratory [MIT Lincoln Laboratory 2018] between March and April 2000 [Shittu 2016] to respond to the interest shown by the scientific community in the evaluation of techniques for complex attack detection [Valeur 2004].

We saw in section 4.4, page 94, that most of the publications about multi-step attack detection use public datasets in the evaluation of the methods. This represents 110 publications, out of which 89 use DARPA 2000. It is then the most used public dataset containing multi-step attacks. Even if it has received some critics about not being very representative [Valeur 2004], the attacks LLDoS 1.0 and LLDoS 2.0.2 are well known and contain a sufficient number of steps [Wang 2008].

The attacks in DARPA 2000 are performed in a simulated network with several Linux and Solaris machines. Each dataset is composed of two tcpdump files. The information in the first one, called 'inside', comes from a sniffer located in the inner part of the network, where the attacked machines Mill, Locke and Pascal are located. The second one, called 'dmz', contains information coming from the DMZ, where public servers are connected. This is complemented by the Solaris BSM audit data coming from Pascal.

The audit data from the Pascal machine is only used by two of the reviewed methods [Anming 2004, Wang 2018]. The rest of the methods work only with a set of alerts generated by an IDS from the packets in DARPA 2000. Some of them use Snort IDS [Yan 2004, Yan 2005, Lee 2008, Alserhani 2010, Xuewei 2014, Wang 2016, Shittu 2016, Holgado 2017]; some others, RealSecure [Ning 2004a, Zhu 2006, Yu 2007, Liu 2008, Farhadi 2011, Anbarestani 2012, Kavousi 2014, Ramaki 2016], and a few,

Bro IDS [Chen 2006, Saad 2014]. We have chosen RealSecure for our evaluation because the resulting alerts have the same name independently of the IDS software version, something that does not happen with Snort. Moreover, some authors [Zali 2013, Ramaki 2015a] consider the alerts generated by RealSecure as more accurate and less redundant.

Alert name	Description	Risk
Admind	The rpc.admind daemon, which allows the remote administration of Solaris computers, is used with insecure authentication.	High
Email_Ehlo	SMTP (Simple Mail Transfer Protocol) daemon supports EHLO (Extended Hello) command.	Low
FPT_Pass	FTP login using a cleartext password, which is stored in the alert.	Medium
Mstream_Zombie	Communication is detected between a master and a slave of the *mstream* DDoS tool using UDP port 10498, 7983, 6838 or 9325.	High
Rsh	A remote shell command (rsh) is executed between two machines.	Medium
Sadmind_AO	A possible buffer overflow attack against the amsl_verify() function in the *sadmind* (Solaris Solstice AdminSuite) daemon. It is caused by a long string within a NET_MGT_PROC_SERVICE. It is the abbreviation of 'Sadmind_Amsl_Overflow'.	High
Sadmind_Ping	Possible scan on the destination to check if there are active *sadmind* daemons running on it.	Low
Stream_DoS	Possible DoS attack, detected by an unusually high volume of TCP ACK packets being sent to a host.	Medium
TelnetEnvAll	Telnet environment variables are used.	Low
TelnetTerminaltype	Beginning of a telnet session detected using certain terminaltype.	Low
TelnetXdisplay	Beginning of a telnet session detected using certain XDisplay.	Low

Table 7.1: RealSecure IDS alerts [Internet Security Systems 2001].

Ning et al. [Ning 2002a] already provide an output of RealSecure alerts from DARPA 2000 that is used by some of the cited authors [Zhou 2007, Sadoddin 2009, Ahmadinejad 2009, Anbarestani 2012]. We also use this data in our evaluation. It is divided in four files: 'inside1' and 'dmz1' for LLDoS 1.0, and 'inside2' and 'dmz2' for LLDoS 2.0.2. Alerts in these files have a timestamp associated to the moment when the RealSecure IDS processed the original tcpdump files, so the reader should not be surprised if the time does not correspond to the moment when DARPA 2000 was created. Conversely, the period of 'inside' alerts and the one of 'dmz' alerts is not the same because files had to be processed sequentially. A simultaneous analysis of the alerts coming from the two subnetworks is then incoherent in terms of time. In consequence, we just keep the files 'inside1' and 'inside2', where all the steps of the attacks are represented.

As the RealSecure IDS drops ICMP packets, there are no alerts corresponding to the first step of both attacks [Ramaki 2016]. The rest of the RealSecure alerts mentioned in this thesis are shown in Table 7.1. The description of each alert and

its level of risk have been extracted from the version 6.5 of the RealSecure Signatures Reference Guide [Internet Security Systems 2001].

7.1.2 ISCX dataset

The lack of public datasets to evaluate detection methods motivated the ISCX team at the University of New Brunswick (UNB) to create a systematic approach for the generation of evaluation data in 2011 [Shiravi 2012]. The ISCX 2012 Intrusion Detection Evaluation Data Set and the CICIDS2017 dataset have been generated thanks to this approach. Both datasets contain several types of attack mixed with simulated normal traffic. Even if the second one is more recent, the multi-step attacks it contains have a limited number of steps and they have not been extensively described. However, the 2012 dataset contains an interesting multi-step attack with 7 steps, which has been used by some published multi-step attack detection methods [Saad 2014, Zargar 2014, Ramaki 2016, Faraji Daneshgar 2016].

This island-hopping multi-step attack described in section 3.2.3, page 33, is contained in the third day of the seven days of network activity captured by the dataset. This period goes from Friday 6th to Thursday 17th November 2010. The network activity from the rest of the days contains other types of attack that are not multi-step.

We need to process the network activity in the dataset by an IDS if we want to work with events, as we explained for DARPA 2000 in the previous section. This time, we have done this process in our lab using Snort IDS, because a public set of previously generated alerts has not been found. In this case, it does not make sense to use RealSecure IDS, a product that is not used since the beginning of 2000's, way before the ISCX dataset was created.

The same as it happens in DARPA 2000, the timestamps of the resultant alerts correspond to the moment when they were generated, in June 2017, and not with the date shown in the original network packets. 57 different Snort alert types have been generated. They are listed in Table 7.2, together with the number of alerts of each type.

7.1.3 HuMa dataset

To test and improve the methods developed under the HuMa project, a dataset of logs containing the traces of a multi-step attack (see section 3.2.4, page 34), among other single-step attacks, has been furnished by one of the industrial partners. The dataset has been created using a virtual environment which replicates the infrastructure of a

Type of alert	#
(http_inspect) NO CONTENT-LENGTH OR TRANSFER-ENCODING IN HTTP RESPONSE	3515
(ftp_telnet) FTP command parameters were too long	3509
PROTOCOL-DNS TMG Firewall Client long host entry exploit attempt	3423
(spp_sdf) SDF Combination Alert	3284
PROTOCOL-DNS potential dns cache poisoning attempt - mismatched txid	3206
Consecutive TCP small segments exceeding threshold	2603
(http_inspect) INVALID CONTENT-LENGTH OR CHUNK SIZE	1468
ICMP test detected	1028
Reset outside window	422
(http_inspect) SERVER CONSECUTIVE SMALL CHUNK SIZES	61
(http_inspect) LONG HEADER	51
INDICATOR-SHELLCODE Shikata Ga Nai x86 polymorphic shellcode decoder detected	31
ET POLICY GNU/Linux APT User-Agent Outbound likely related to package management	30
(http_inspect) OVERSIZE REQUEST-URI DIRECTORY	25
Bad segment, adjusted size <= 0	23
ET WEB_SERVER HTTP 414 Request URI Too Large	17
ET SCAN Potential SSH Scan	15
(ftp_telnet) TELNET CMD on FTP Command Channel	13
(http_inspect) JAVASCRIPT WHITESPACES EXCEEDS MAX ALLOWED	10
(http_inspect) SIMPLE REQUEST	8
ET WEB_SERVER Possible SQLi xp_cmdshell POST body	7
ET WEB_SERVER /bin/sh In URI Possible Shell Command Execution Attempt	5
ET POLICY Suspicious inbound to PostgreSQL port 5432	5
ET POLICY Suspicious inbound to MSSQL port 1433	5
ET POLICY Suspicious inbound to mySQL port 3306	4
ET POLICY Suspicious inbound to Oracle SQL port 1521	4
(http_inspect) UNESCAPED SPACE IN HTTP URI	4
(http_inspect) POST W/O CONTENT-LENGTH OR CHUNKS	4
Limit on number of overlapping TCP packets reached	3
(spp_ssh) Protocol mismatch	3
(http_inspect) UNKNOWN METHOD	3
PROTOCOL-DNS domain not found containing random-looking hostname - possible DGA detected	2
ET SHELLCODE Rothenburg Shellcode	2
ET SCAN Potential FTP Brute-Force attempt response	2
ET FTP Suspicious Quotation Mark Usage in FTP Username	2
(IMAP) Unknown IMAP4 command	2
SERVER-WEBAPP JBoss JMX console access attempt	1
ET WEB_SPECIFIC_APPS Possible HP Power Manager Management Web Server Login Remote Buffer Overflow Attempt	1
ET WEB_SERVER SQL Errors in HTTP 200 Response (SqlException)	1
ET WEB_SERVER Possible DD-WRT Metacharacter Injection Command Execution Attempt	1
ET WEB_SERVER Possible Cookie Based BackDoor Used in Drupal Attacks	1
ET WEB_SERVER PHP tags in HTTP POST	1
ET WEB_SERVER HTTP POST Generic eval of base64_decode	1
ET WEB_SERVER HP OpenView Network Node Manager OvWebHelp.exe Heap Buffer Overflow Attempt	1
ET TROJAN Double HTTP/1.1 Header Inbound - Likely Hostile Traffic	1
ET SCAN Toata Scanner User-Agent Detected	1
ET SCAN Potential VNC Scan 5900-5920	1
ET SCAN Potential VNC Scan 5800-5820	1
ET POLICY Python-urllib/ Suspicious User Agent	1
ET POLICY Incoming Basic Auth Base64 HTTP Password detected unencrypted	1
ET FTP Suspicious Percentage Symbol Usage in FTP Username	1
ET EXPLOIT Wscript Shell Run Attempt - Likely Hostile	1
ET EXPLOIT HP OpenView Network Node Manager Toolbar.exe CGI Buffer Overflow Attempt	1
ET DOS Microsoft Remote Desktop (RDP) Syn then Reset 30 Second DoS Attempt	1
(http_inspect) NON-RFC DEFINED CHAR	1
(ftp_telnet) Invalid FTP Command	1
(POP) Unknown POP3 command	1

Table 7.2: Snort alerts in ISCX dataset.

typical business network. The logs generated in the network assets during the execution of the attacks have been brought together to conform the dataset.

This dataset is really big in number of lines, demonstrating that many filters and automatic processes are needed to deal with the massive amount of logs generated by unit of time in an average size company. It has a size of 2,844,691 lines, once it is cleaned from events that do not contain enough information to be processed. The data correspond to a period of five days, from a Monday to a Friday.

Logs are coming from different origins, that are listed in Table 7.3. To preserve the confidentiality of the test environment, we have been told not to reveal any of the involved IP addresses.

Origin	**Number of logs**
Web server (application)	1,484,134
Firewall	1,271,576
FTP server (OS)	32,762
Web server (OS)	19,829
DNS server (OS)	15,471
Mail server (OS)	15,317
IDS	5,602

Table 7.3: Origins of logs in HuMa dataset.

7.1.4 Normalization of events

Events contained in the presented datasets need to be adapted to a common format if we want to easily define AASGs and to work with Morwilog and Bidimac. This transformation process is called *normalization* (see section 2.3, page 19), and it is usually done in every event-based security system in the industry, to compare events coming from different sources and with different formats.

We have defined a list of 13 fields in the normalized format. They are listed and described in Table 7.4. For DARPA 2000 and ISCX datasets we have just developed a parser that maps attributes in the IDS alerts to the correspondent equivalents in the proposed format. It is straightforward to do it because all the events come from the same device. However, adapting HuMa requires more work due to the heterogeneous origins of the events in the dataset. Special parsers have been developed for each one of the devices listed in Table 7.3, trying to keep the equivalence in the meaning of the attributes between all the devices. Another purpose of normalization is to order the

Field	Description
id	Number unambiguously identifying the event
time	Timestamp in POSIX format
origin	Device generating the event
service	Service associated to the event (e.g. NTP, Kerberos)
ipsrc	Source IP address
ipdst	Destination IP address
type	Type of event
action	Type of action associated to the event (e.g. request, exit, blocked)
process_id	Identifier of the OS process related to the event
psrc	Source port number
pdst	Destination port number
log	The text of the raw log
tag	Indication of attack for evaluation

Table 7.4: Fields used in parsed events.

events in time, guaranteeing that each log has a timestamp equal or higher than the previous ones.

When evaluating Morwilog and Bidimac, we need to know where the multi-step attacks are, so we can simulate the feedback provided by the analyst after verification of the generated alarms. To do that, we include in the normalized format the field *tag*, where a number indicates if the event belongs to a multi-step attack, and to which one of them in case that there are several ones. None of the used datasets are labelled, so one of our tasks has been the manual identification of the attacks in each dataset and the assignation of the tags.

To summarize the information about the datasets, we present in Table 7.9 a comparison of them in terms of number of events, number of aggregated events (see section 7.2.3, page 187), duration of the actions represented in the dataset and duration of the multi-step attack contained in it.

7.1.5 The *eventgen* datasets

The datasets mentioned so far have the big limitation of only containing a single instance of a multi-step attack. We have then created a new set of simulated datasets using Splunk Event Generator[1], a software developed to create artificial events to

[1] https://github.com/splunk/eventgen

Name	Attack	Events	Agg. events	Duration	Duration attack
DARPA 2000 inside1	LLDoS 1.0	922	855	3h 13′ 53″	1h 19′ 45″
DARPA 2000 inside2	LLDoS 2.0.2	494	450	1h 43′ 15″	48′ 32″
ISCX	Island-hopping	22,820	9,608	24h	5h 53′ 31′
HuMa	Scan + SSH brute-force	2,844,691	146,748	5d 15h	3d

Table 7.5: Summary of the datasets used in evaluation.

test Splunk, a commercial SIEM. The generation of events has been configured to be adapted to our needs in each experiment. More than creating a realistic dataset, our goal is to produce sequences of events that can be confounded among background data. The exact composition of an attack does not matter, the important thing is that we are able to identify it as a different sequence.

Parameter	Meaning
num_log	Total number of events in the dataset
num_typ_attack	Number of different kinds of attack sequence
num_typ_inn	Number of different kinds of innocent sequence
prop_attack	Proportion of events corresponding to attack sequences
prop_inn	Proportion of events corresponding to innocent sequences
seq_size	Number of events (steps) in the sequences
seq_step	Maximum difference between events in a sequence

Table 7.6: Parameters in the creation of the *eventgen* datasets.

When generating the dataset, each new event has one out of three origins randomly assigned according to a probability distribution: firewall ('FW'), web server ('WS') and mail server ('MS'). Once the attribute *origin* has been selected among these three values, the values for *service*, *type* and *action* are also randomly selected, with the possible values depending on the origin. The result is a total of 1038 different kinds of event, considering the combination of the fields *origin*, *service*, *type* and *action*.

Source IP addresses are also randomly chosen, with a 95% of chance of picking up one that has already been used in another event. Timestamps are generated by a counter, with intervals of increment that are also random.

Once the events have been created, a set of event sequences with the source IP address as the overarching element are injected in the dataset. Some of them are labelled as attacks and some of them are not. The distance between the events in the

sequences is also random, with a maximum distance given in number of events by the parameter *seq_step*. The proportion of events that represents an attack is one of the parameters in the generation of events, called *prop_attack*. All the parameters are listed in Table 7.6.

The datasets created using this method are called the *eventgen* datasets. They are used in the evaluation of the algorithms in section 7.3.

7.2 Evaluation of different AASGs

In this section we propose several use cases of different AASGs. They have been built following the logic of a security analyst. The application to each dataset is done through Morwilog and Bidimac algorithms. Therefore, the algorithms and the utility of the designed AASGs are evaluated in parallel, although a specific evaluation of the algorithms is presented in section 7.3.

We assume so far that the security analyst is infallible when evaluating the raised alarms. Her behavior is simulated within the execution of the algorithms, thanks to the tags assigned to the attacks contained in each dataset (see Table 7.4). If the sequence of logs that triggers the alarm is marked as an attack, the alarm is considered as true. We also suppose that the evaluation of the analyst takes place immediately after the alarm is raised. The values used for T_{max} depend on the dataset and are shown in Table 7.7.

Dataset	$\boldsymbol{T_{max}}$
DARPA 2000 inside1	1200 seconds (20 minutes)
DARPA 2000 inside2	1200 seconds (20 minutes)
ISCX	28,800 seconds (8 hours)
HuMa	172,800 seconds (48 hours)

Table 7.7: Values of T_{max} for evaluation.

7.2.1 DARPA 2000 LLDoS 1.0

The first AASG we evaluate is Mill-1, the one represented in Figure 5.8 (page 128), which was built from the excerpt of DARPA 2000 shown in Table 5.2. To remind the structure of Mill-1, we show it again in Figure 7.1, but expressed differently than in Figure 5.8. Instead of basing the representation on the numbering of functions

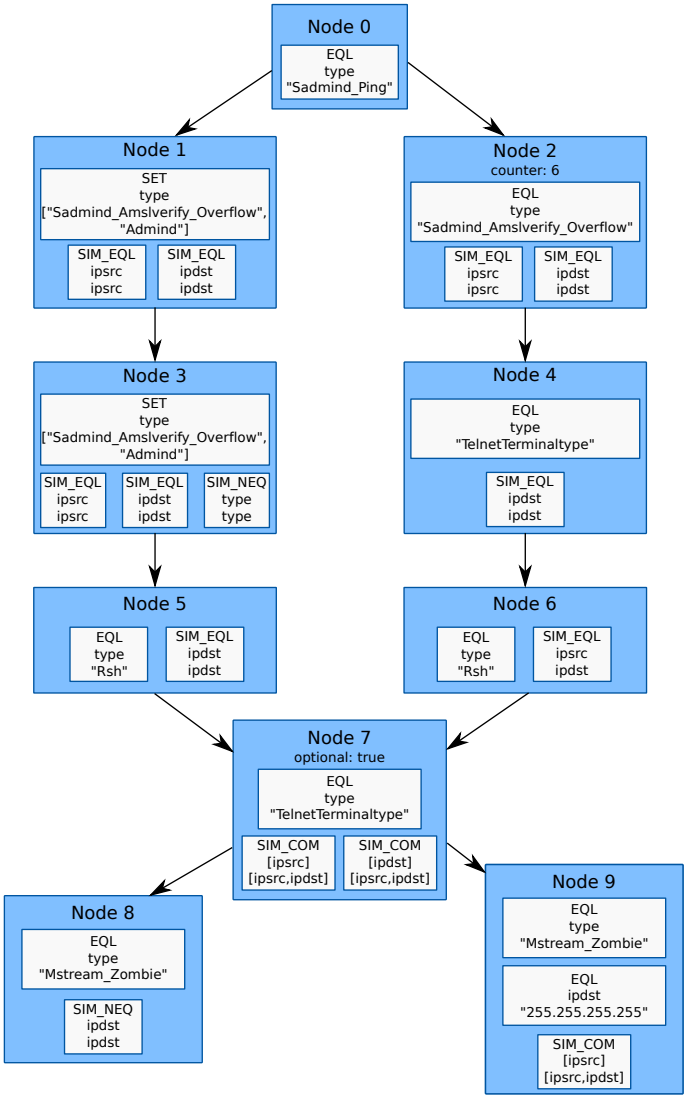


Figure 7.1: Functional representation of the AASG Mill-1.

defining each condition, we use the JSON members defined in section 5.4.1 for AASG implementation. Even if this representation does not exactly correspond to the formal definition of an AASG, it is more intuitive when working with real AASG and it has the same elements as the one used in the graph editor presented in section 5.4.3. This new form is called *functional representation*, while the one in Figure 5.8 is the *formal representation*. In the functional representation, conditions are expressed as boxes with the members arranged in order: the name of the function is followed by the name of the attribute and the reference value, in case of being absolute, or the names of the attribute of the parent and children nodes, in case of being relative. If there is a counter associated to the node or it is an optional node, this is also indicated next to the name of the node.

Remember that the attack from which Mill-1 is built corresponds to a part of the LLDoS 1.0 attack (see section 3.2.2), more precisely to the infection of the Mill server. In LLDoS 1.0, the attacker infects two other machines, Locke and Pascal. The three infections are similar and separately correspond to the steps 1 to 4 of the attack. We have thus three different occurrences of the scenario represented in the AASG, which are multi-step attacks by themselves, even if in absolute terms the dataset only contains one multi-step attack.

We want to see if the infections of Locke and Pascal are detected using the AASG Mill-1. First, we remove the 37 IDS alerts corresponding to the infection of Mill from the dataset inside1. This results in a new dataset, called *inside1-NoMill*. Then, we execute Morwilog and Bidimac.

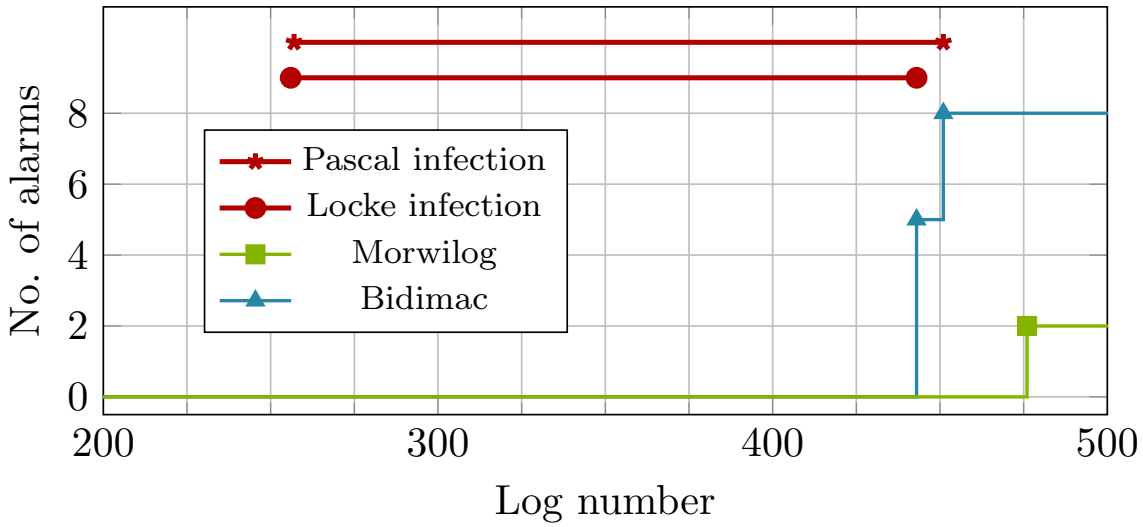


Figure 7.2: Number of alarms vs. log number in inside1-NoMill using Mill-1

In both cases, the two infections raise alarms, which means that the AASG Mill-1 is valid for the detection of the attacks against Locke and Pascal. But not all the branches match the involved alerts as they would do when facing an exact repetition of the infection of Mill. This reflects the fact that some of the cases proposed by the analyst were valid for that infection but not for the other ones. The branches matched by Locke and Pascal[2] are actually the same ones in both cases: $[0, 1, 3, 5, 8]$, $[0, 1, 3, 5, 9]$, $[0, 1, 3, 6, 8]$ and $[0, 1, 3, 6, 9]$.

The analysis of the results returned by Bidimac is the best way to determine how many branches are matched by each attack, because this method looks for matches against every single branch in the AASG, raising an alarm for each match. As every

[2]To simplify, we refer to the 'infection of machine X' as 'X'.

branch corresponding to a sequence in the data raises an alarm, the generated alarms are a direct representation of the statistics of the dataset in terms of the cases proposed by the analyst. Conversely, *morwis* in Morwilog explore only one branch of the AASG for each event matching κ_0. There is then a difference in the number of generated alarms, up to 8 alarms for Bidimac and just 2 for Morwilog.

Apart from counting the number of alarms, it is interesting to check when they are generated. A plot of the number of alarms versus the number of each log in the dataset is presented in Figure 7.2. This 'log number' is just the position of each log in the list of logs conforming the dataset. Each log has thus a different number and, because they are ordered in time, it is guaranteed that if a log number b is bigger than another one a, the timestamp of the event numbered a is equal or inferior than that of the one numbered b.

The reason of using the log number and not the timestamp in the figures is to avoid that two different logs with the same timestamp can lay in the same x coordinate, making visualization of results more difficult. Anyways, the timestamp of each coordinate is indicated below the log number in the subsequent figures, as a reference. Figure 7.2 covers the whole period where the infections take place, but in subsequent plots representing the number of alerts in DARPA 2000, only the period when the alarms are launched will be covered.

The researchers that have previously worked with DARPA 2000 are probably surprised about not using the alert ID as the number to identify each log in the figures. The reason is that an order based on alert ID does not correspond at all with an order based on the time of the event. We will see it clearly in Figure 7.10, where we do use the alert ID because we are not interested in show the distance between the logs like in Figure 7.2.

To have an overview of the possible branches matched for this dataset, all the Bidimac matches are represented in Figure 7.3, in diagrams where the matched nodes are placed in the vertical axis and the log number is represented in the horizontal axis. The red node at the end of each branch indicates the moment when the alarm is risen.

In the case of Morwilog, only two of the represented patterns raise alarms. We see three of these patterns in Figure 7.4. Paths for both infections, Locke and Pascal, are separated to ease visualization because they happen almost simultaneously. The path to nodes that have not been chosen by the *morwi* but have been matched is represented by a dashed line. The moment of launching the two alarms is indicated with a red bar. They are always launched in position 467 (time 5:33:18) because it is the moment when the search of children of node κ_0 finishes due to the limit established

by T_{max}. If we look closely to the figure, the node κ_2 has never been found, so when the *morwi* arrives at the last node in the AASG (κ_8 or κ_9, depending on the decision made) and the alarm is launched, logs before the one in position 467 have already arrived at the system. This translates into a late pheromone update, after the two infections are detected.

As both attacks are detected when the stigmergic AASG Mill-1 has just been initialized, thus having all the arcs the same level of pheromones, the decisions of the *morwi* are uniformly random. When running the execution, a high number of

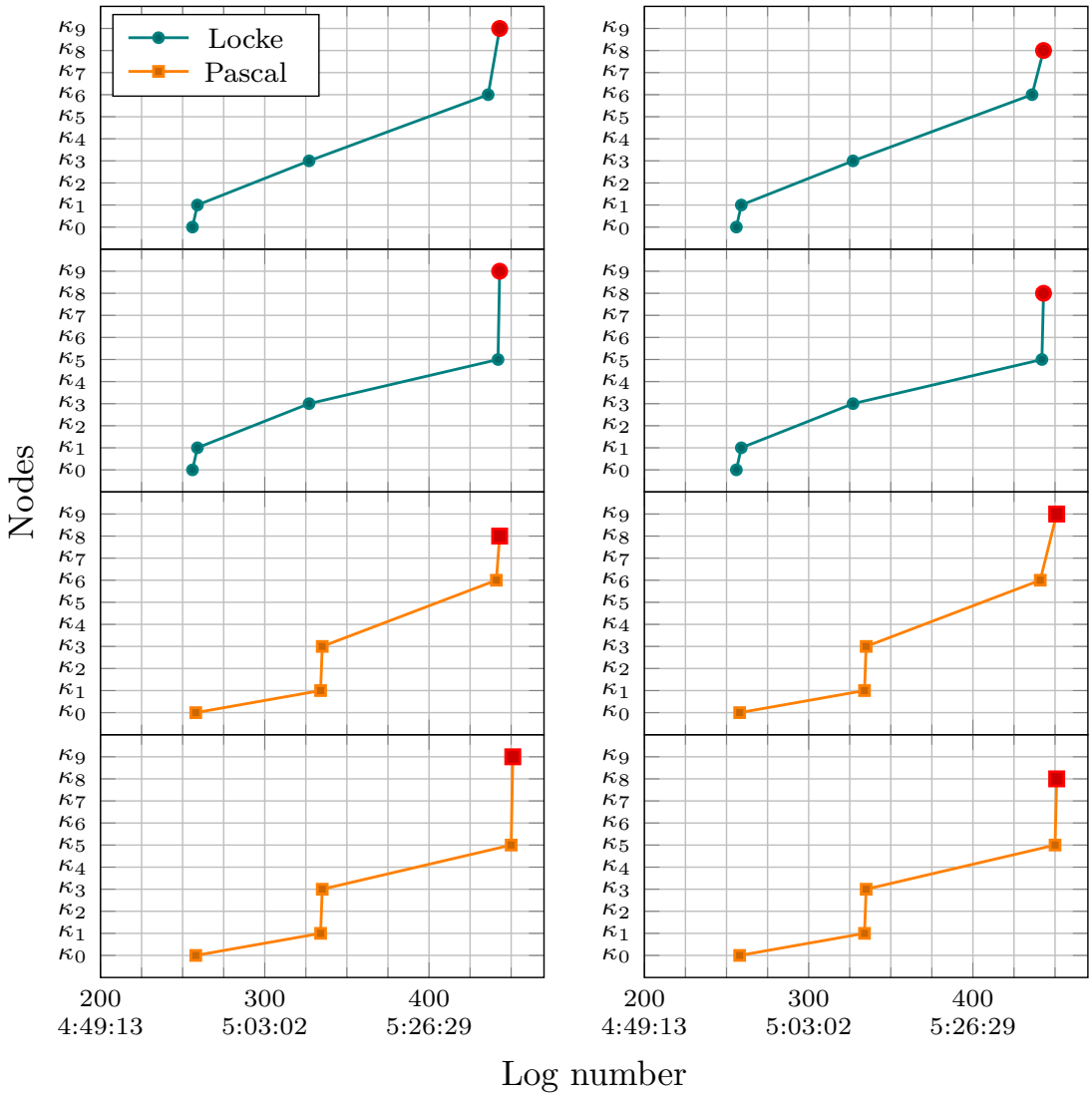


Figure 7.3: Results of the execution of Bidimac with AASG Mill-1 in inside1-NoMill. The red-colored nodes represent the generated alarms.

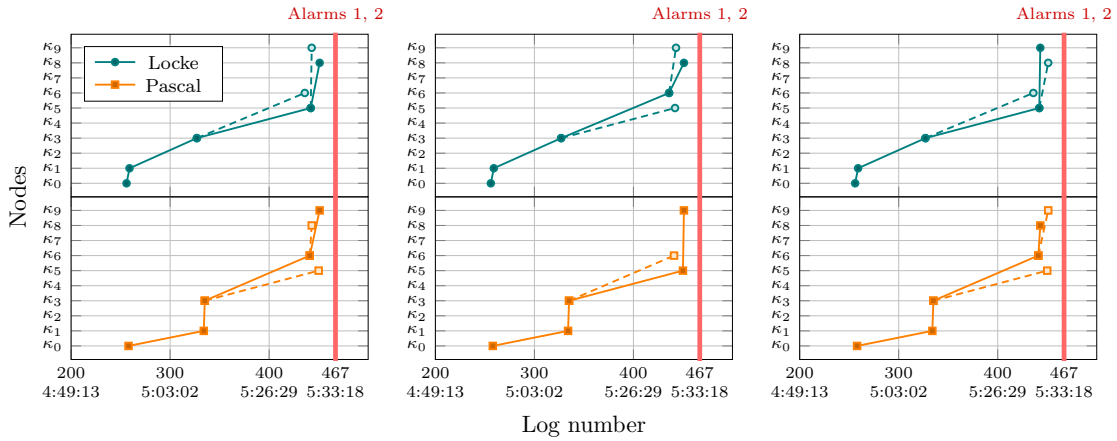


Figure 7.4: Examples of execution of Morwilog with AASG Mill-1 in inside1-NoMill. The uncolored nodes represent matched events that have not been chosen by the *morwi*.

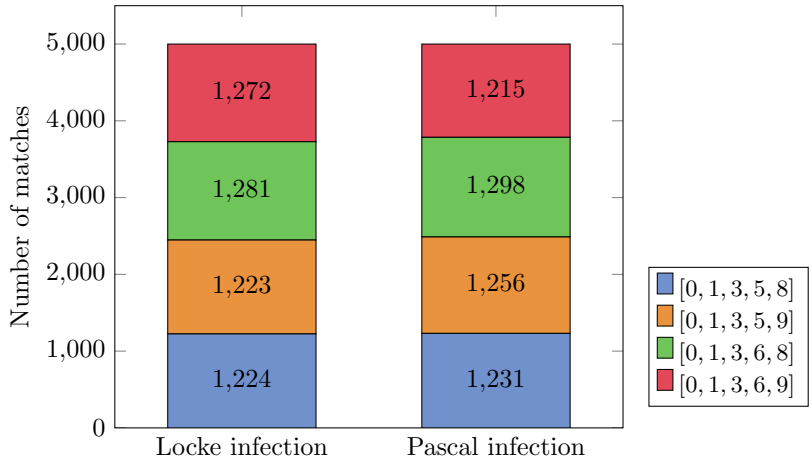


Figure 7.5: Number of branch matches in Morwilog using dataset inside1-NoMill with AASG Mill-1 after 5000 executions.

times, we should see that any branch is equally chosen. This is shown in Figure 7.5, where we represent the number of times each branch is matched for each one of the infections after 5000 executions. The small differences in the number of branches is due to randomness, but it is easy to infer that an infinite number of executions would end up with the same number of occurrences for each branch.

As a conclusion of this section, the advantage of Morwilog over Bidimac in terms of number of alarms is clear. It could be argued that with Morwilog we do not know every possible branch that could be matched by the sequences of alerts, but at the same time the work for processing the alarms and providing feedback is less. Think that when building an AASG from a sample of a multi-step attack it is normal that

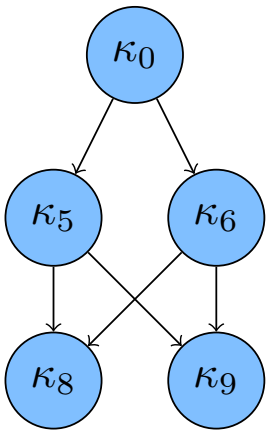
several of the proposed cases are correct, because they are based on solid ground and developed by a security expert. But the goal is not to match all the occurrences but a) to launch an alarm when any of the branches is matched so we can detect the new instance of the attack and b) to develop a detection model with the essential elements of the attack or the most repeated ones. We look for a case that could be always valid for detection, independently that other cases can be equally valid. Many alarms would be risen at any repetition of the multi-step attack if we execute Bidimac on an AASG with many correct hypotheses.

7.2.2 From LLDoS 1.0 to LLDoS 2.0.2

Imagine that the analyst wants to create an AASG more general than Mill-1 to detect and identify a broader range of similar attacks. The following modifications to Mill-1 could be done to create Mill-2, a different AASG:

1. **Suppression of node matching the 'SadmindPing' alert (κ_0)**. The 'SadmindPing' alert represents a reconnaissance stage in which the machines are probed to check if they have the *sadmind* service running. This stage could be done weeks before the infections take place, resulting in a partial match of the AASG Mill-1. The new AASG Mill-2 now matches alerts of the type 'Sadmind_Amslverify_Overflow' in node κ_0.

2. **Consideration of another protocol different than rsh to send the malware**, such as FTP (nodes κ_5 and κ_6).



Node	Abstract event	List of conditions
κ_0	e_0^*	$g_1(type_a, \text{'Sadmind\_AO'}) = 1$
κ_5	e_5^*	$g_4(type_a, \{\text{'Rsh'}, \text{'FTP\_Put'}\}) = 1$, $f_1(ipdst_a, ipdst_p) = 1$
κ_6	e_6^*	$g_1(type_a, \{\text{'Rsh'}, \text{'FTP\_Put'}\}) = 1$, $f_1(ipsrc_a, ipdst_p) = 1$
κ_8	e_8^*	$g_1(type_a, \text{'Mstream\_Zombie'}) = 1$, $f_6(ipdst_a, ipdst_p) = 1$
κ_9	e_9^*	$g_1(type_a, \text{'Mstream\_Zombie'}) = 1$ $f_2(\{ipsrc_a\}, \{ipsrc_p, ipdst_p\}) = 1$ $g_1(ipdst_a, 255.255.255.255) = 1$

Figure 7.6: Formal representation of AASG Mill-2.

3. **Suppression of optional node κ_7.** As 'TelnetTerminaltype' is such a common alert in our network and the analyst was not sure about its presence in the attack when she created Mill-1, the node κ_7 can be erased to simplify the AASG.

This results in a new AASG Mill-2 whose formal representation is shown in Figure 7.6 and its functional one in Figure 7.7. The numbers assigned to each node have been kept to ease the comparison with Mill-1.

Reducing the number of steps and conditions associated to the AASG results in a more general AASG, but the cost to pay is an increase in the number of alarms for both algorithms: 8 alarms in Morwilog and 32 in Bidimac. The diagram in Figure 7.8 shows the evolution in the number of alarms in terms of the log number. Notice that for Morwilog we have two cases: one in which the first four alarms are triggered at log number 451 and the second four alarms at 574, and another one in which all of them are triggered at 451. This just depends on the randomness of the path followed by the *morwis*.

More details about which are the matched paths are shown in Figure 7.10 for Bidimac and in Figure 7.11 for Morwilog. In both images, we can see which is the alert ID of the events matching each one of the branches in the AASG. In this case,

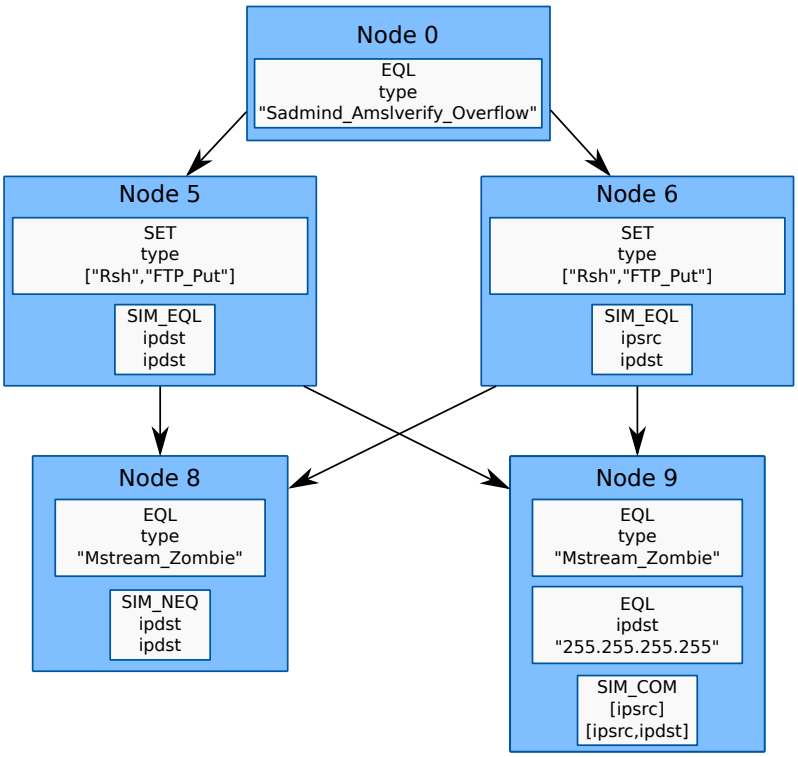


Figure 7.7: Functional representation of AASG Mill-2.

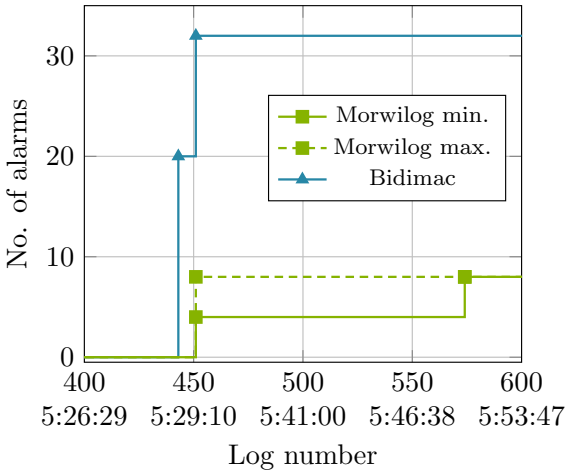


Figure 7.8: Number of alarms with AASG Mill-2 in inside1-NoMill.

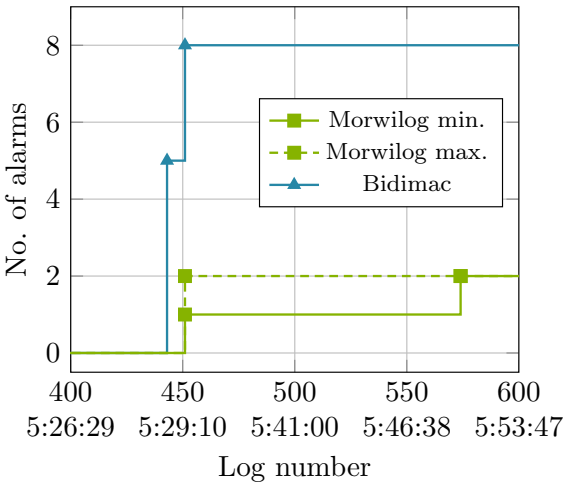


Figure 7.9: Number of alarms with AASG Mill-2 in the aggregated version of inside1-NoMill.

the distance between the events in the plot is not proportional to the distance between the corresponding events in term of log number, as it happens in Figure 7.8. Anyways, the time when each event occurs is indicated underneath. This helps the visualization of the branches, as some events matching subsequent nodes are very close in terms of log number. In both figures, we can see how the order of the alert IDs does not correspond to the temporal order of the events, as we pointed out in page 180. For example, event with alert ID 67442 precedes in time the one with alert ID 67438.

In the case of Bidimac (Figure 7.10), the sequences are superimposed to the plot of the number of triggered alarms. Again, alarms in Bidimac are always generated at the last matched node in the sequence. We see that the effect of having 32 alarms is a consequence of having 4 branches that always match with the attacks Locke and Pascal, with eight events matching κ_0 and starting the sequence: the ones with alert IDs 67430, 67428, 67442, 67438, 67440, 67436, 67432 and 67434.

Conversely, only one branch is chosen in Morwilog for each time event matching κ_0. We have represented the possibilities that each generated *morwi* has for choosing in Figure 7.11. If we compare with Figure 7.10, we can see that the matched branches are the same and the only cause determining a lower number of alarms in Morwilog is the random pheromone-based choice.

The interesting thing about the AASG Mill-2 is that it is also able to detect the attack LLDoS 2.0.2. The number of alarms triggered by the application of Mill-2 on the dataset inside2 is shown in Figure 7.12. Both Morwilog and Bidimac trigger two

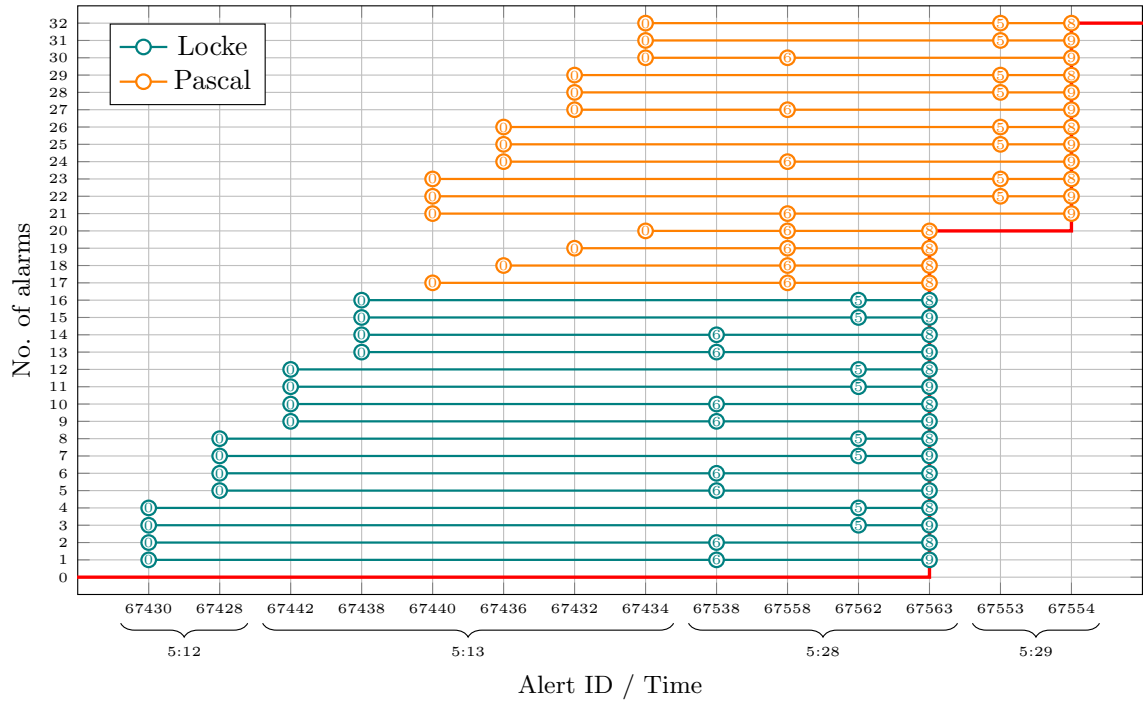


Figure 7.10: Results of Bidimac with AASG Mill-2 in DARPA 2000 inside1-NoMill, on the infections of the machines Locke and Pascal.

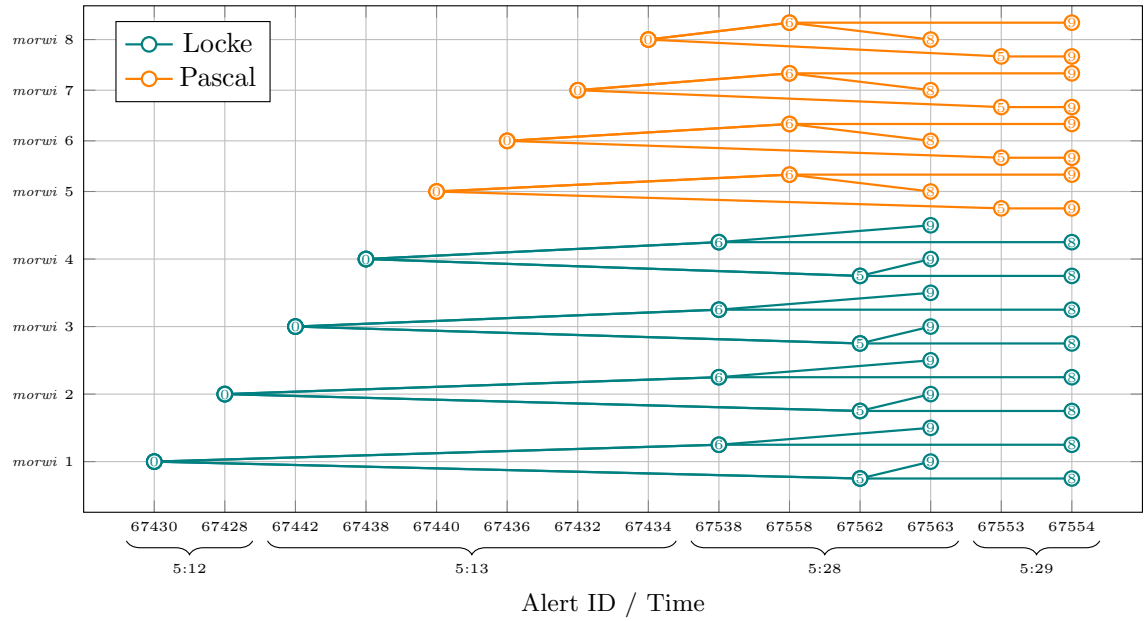


Figure 7.11: Results of Morwilog with AASG Mill-2 in DARPA 2000 inside1-NoMill, on the infections of the machines Locke and Pascal.

alarms, but at different moments. The branch matched this time is only one for both alarms: $[0, 5, 8]$. It is possible then to create an AASG that is able to detect two similar but not identical attacks. It depends on the generalization of the conditions expressed in each one of the cases represented in the branches.

7.2.3 Introducing an aggregation phase

The repetition of alarms representing the same matched branches in Figures 7.10 and 7.11 may be a result of the duplication of events in the dataset.

Rules for the generation of logs are not necessarily too complex. Their simplicity can easily lead to duplicated or very similar logs representing the same event. For example, imagine a rule in an IDS that creates a 'SSH Brute-force' event each time it detects five failed SSH login requests. A real SSH brute-force attack where the attacker makes fifty failed attempts before getting in the system will result in ten consecutive alerts with a variation in the timestamp at most.

Many authors [Alserhani 2013, Ramaki 2016, Zhang 2017, Saikia 2018] apply a pre-filtering to the dataset to avoid redundancies. The process of merging several events meaning the same into one event is called *aggregation*. To transform the datasets, we have designed an aggregation process inspired by the Rule 1 in the ONTIDS system [Zargar 2014]. This process considers that two events are the same if they share *ipsrc*, *ipdst* and *type*, and the difference between timestamps (attribute *time*) is less than 60 seconds. To keep it simple, only the first log in the aggregated set is preserved.

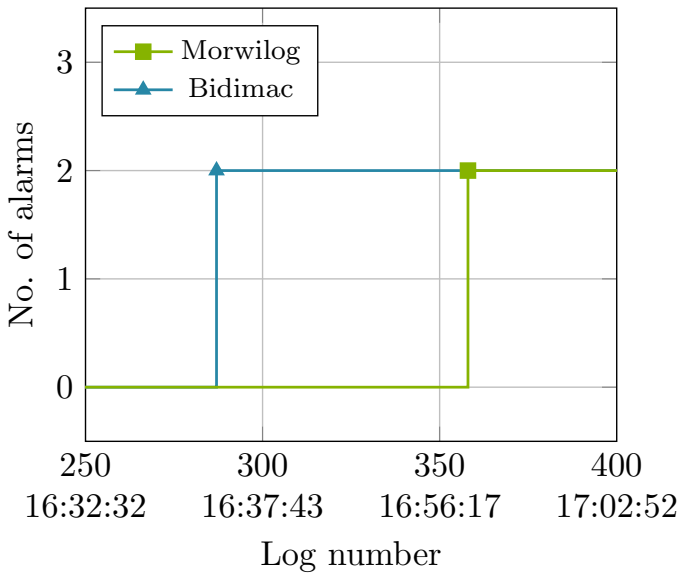


Figure 7.12: Number of alarms with AASG Mill-2 in inside2.

The results of applying Morwilog and Bidimac using the AASG Mill-2 in inside1-NoMill after it has been processed by the aggregation algorithm are shown in Figure 7.9. To easily compare with the results before aggregation, Figure 7.9 is placed next to Figure 7.8 and the original log numbers are preserved. The number of alarms after aggregation is considerably lower: 8 alarms in Bidimac and just 2 in Morwilog. However, the moments when the alarms are triggered are still the same.

This experiment shows how filtering the input events is important to avoid the generation of duplicated alarms. Notice that the number of alarms has been divided by 4 using a simple aggregation algorithm. When implementing a detection algorithm in a real network, it is expected to have even more complex and adapted algorithms for event aggregation. The aggregation algorithm has been applied to all the datasets we use in the evaluation. The resulting number of logs is shown in the fourth column of Table 7.9 in page 192. The reduction of logs is impressive in HuMa, where just around 5% of the logs are preserved.

7.2.4 AASG with scan and double attack

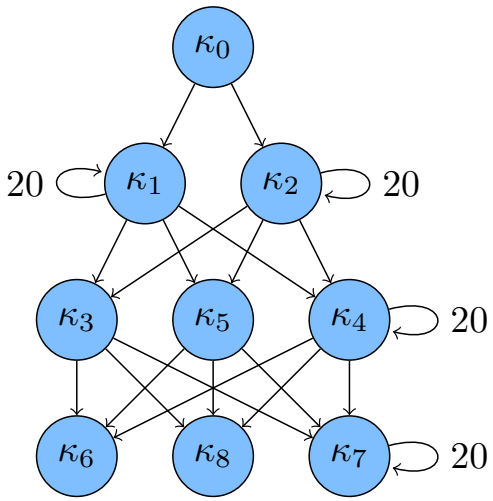
Hypotheses about multi-step attacks represented in an AASG do not forcibly need to be related to a single attack process. The analyst may be interested in a succession of alternative attacks. She could make of the AASG a tool for detecting any of these alternatives and for identifying the most probable one. For example, if she knows that a port scan could be followed by the exploit of two different vulnerabilities, both could be coded in separate nodes that would be children of the node representing the scan.

Taking this point of view, we have developed an AASG representing a scan followed by two consecutive attacks in which the attacker changes her IP address. This AASG is called 'Scan and double Attack' (S2A). Its formal representation is shown in Figure 7.13 and its functional one, in Figure 7.14. The represented multi-step attack is composed of the following steps:

1. **Beginning of the scan**. The node κ_0 represents the initial identification of an ICMP message sent to an internal IP address. To identify the target as an internal host, we use the prefix similarity function g_3 to check if the destination IP address (*ipdst*) belongs to a subnetwork of private addresses of the form '192.168.0.0'. A threshold $\lambda = 0.45$ is used, slightly lower than the one corresponding to a network mask /16 (threshold $\lambda = 0.5$). To avoid false positives, we do not consider events coming from the internal servers. For doing that, the inequality function g_2 is used, in this case with 192.168.5.122 and 192.168.5.123,

which correspond to the IP addresses of two servers in the ISCX dataset.

2. **Continuation of the scan**. The next nodes of the AASG represent two alternative ways to perform the scan: on IP addresses (node κ_1) and on services (node κ_2). Both nodes have an associated counter of 20, which means that 20 events matching the node are expected to be found. It is the minimum number of events



Node	Abstract event	List of conditions
κ_0	$\boldsymbol{e}_0^*$	$g_4(type_a, \text{'ICMP detected'}, 0.3) = 1$, $g_3(ipdst_a, \text{'192.168.0.0'}, 0.45) = 1$, $g_2(ipsrc_a, \text{'192.168.5.122'}) = 1$, $g_2(ipsrc_a, \text{'192.168.5.123'}) = 1$
κ_1	$\boldsymbol{e}_1^*$	$f_1(type_a, type_b) = 1$, $f_1(ipsrc_a, ipsrc_b) = 1$, $f_3^*(ipdst_a, ipdst_b, 0.75) = 1$, $f_{neq}(ipdst_a, ipdst_b) = 1$
κ_2	$\boldsymbol{e}_2^*$	$f_1(type_a, type_b) = 1$, $f_1(action_a, action_b) = 1$, $f_1(ipsrc_a, ipsrc_b) = 1$, $f_1(ipdst_a, ipdst_b) = 1$, $f_1(psrc_a, psrc_b) = 1$, $f_{neq}(service_a, service_b) = 1$
κ_3	$\boldsymbol{e}_3^*$	$f_1(ipsrc_a, ipsrc_b) = 1$, $f_3^*(ipdst_a, ipdst_b, 0.75) = 1$, $g_4(type_a, \text{'SQL'}, 0.1) = 1$
κ_4	$\boldsymbol{e}_4^*$	$f_1(ipsrc_a, ipsrc_b) = 1$, $f_3^*(ipdst_a, ipdst_b, 0.75) = 1$, $g_4(type_a, \text{'failure SSH'}, 0.5) = 1$
κ_5	$\boldsymbol{e}_5^*$	$f_1(ipsrc_a, ipsrc_b) = 1$, $f_3^*(ipdst_a, ipdst_b, 0.75) = 1$, $g_4(type_a, \text{'EXPLOIT Attempt'}, 0.1) = 1$
κ_6	$\boldsymbol{e}_6^*$	$f_{neq}(ipsrc_a, ipsrc_b) = 1$, $f_3^*(ipdst_a, ipdst_b, 0.75) = 1$, $g_4(type_a, \text{'SQL'}, 0.1) = 1$
κ_7	$\boldsymbol{e}_7^*$	$f_{neq}(ipsrc_a, ipsrc_b) = 1$, $f_3^*(ipdst_a, ipdst_b, 0.75) = 1$, $g_4(type_a, \text{'failure SSH'}, 0.5) = 1$
κ_8	$\boldsymbol{e}_8^*$	$f_{neq}(ipsrc_a, ipsrc_b) = 1$, $f_3^*(ipdst_a, ipdst_b, 0.75) = 1$, $g_4(type_a, \text{'EXPLOIT Attempt'}, 0.1) = 1$

Figure 7.13: Formal representation of the AAGS S2A.

from which we consider a scan occurs. On the one hand, events considered by κ_1 as being part of an IP address scan have a destination IP address that is different than the previous event ($f_{neq}(ipdst_a, ipdst_b) = 1$) but belongs to the same class C subnetwork ($f_3^*(ipdst_a, ipdst_b, 0.75) = 1$). On the other hand, in the service scan represented in κ_2, the destination IP address is the same ($f_1(ipdst_a, ipdst_b) = 1$) but it is the service that changes ($f_{neq}(service_a, service_b) = 1$).

3. **First attack**. A first attack launched from the same IP address that launched the scan ($f_1(ipsrc_a, ipsrc_b) = 1$) is represented in the nodes κ_3, κ_4 and κ_5. The kind of attack is identified by textual similarity (g_4) with a string naming it. Three kinds are considered: SQL injection, identified by 'SQL' in κ_3; SSH brute-force, identified by 'failure SSH' repeated 20 times in κ_4, and an exploit execution, identified by 'EXPLOIT Attempt' in κ_5.

4. **Second attack**. This step consists on a second attack of any of the three kinds proposed in the previous step but launched from a different IP address ($f_{neq}(ipsrc_a, ipsrc_b) = 1$).

This AASG S2A fully represents the HuMa attack described in section 3.2.4, but also the three last steps of the ISCX island-hopping attack presented in section 3.2.3. In terms of evaluation, it does not make sense to create an AASG fully representing the ISCX island-hopping attack because in the ISCX dataset there are only alerts corresponding to the three last steps. This is one of the problems of this dataset, together with the fact that it contains only one multi-step attack.

The AASG S2A has been successfully tested both in ISCX and HuMa datasets in their aggregated version. The results are shown in Table 7.8. Only one branch in the AASG is matched for each dataset, corresponding to the multi-step attack they contain. We see that the number of alarms returned by Bidimac is really high, despite of only existing one attack in each dataset. This is due to the high number of events involved in each attack and the repetition of actions within them (e.g. in the scans), partially maintained even after the aggregation process. In any case, the AASG S2A

	Agg. ISCX	Agg. HuMa
Number of alarms Morwilog	1	9
Number of alarms Bidimac	108	73
Branch matched	[0, 1, 3, 8]	[0, 2, 4, 7]

Table 7.8: Results of AASG S2A.

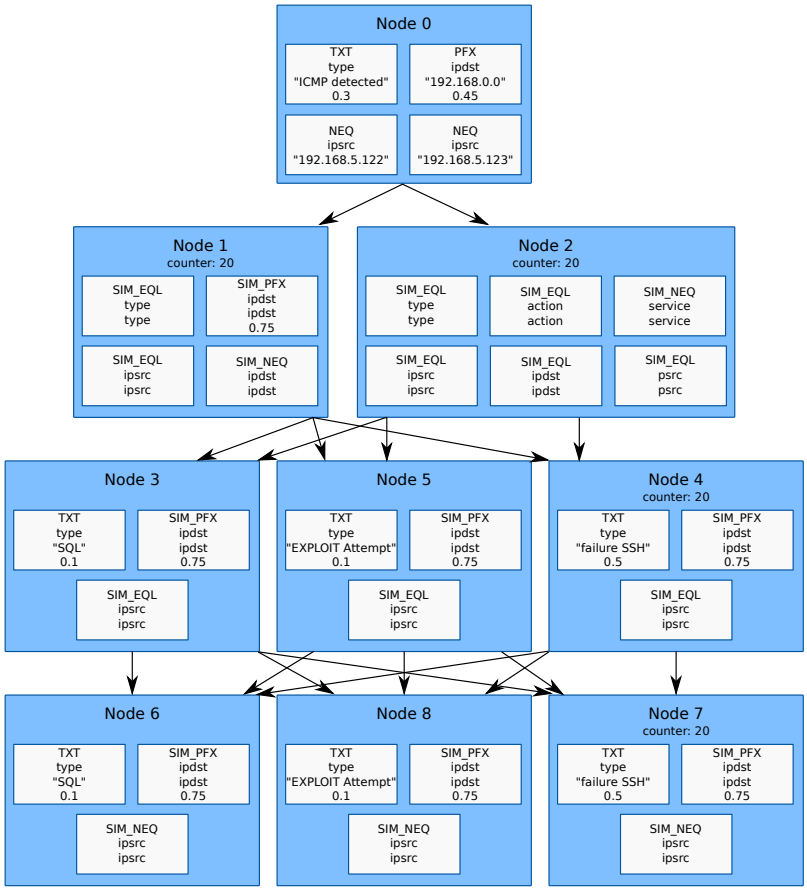


Figure 7.14: Functional representation of the AASG S2A.

has demonstrated its versatility to find two attacks with similar structure but different content in two different datasets.

7.3 Evaluation of the algorithms

In the previous section, we have evaluated different examples of AASG on the pre-built datasets presented in section 7.1. Now it is time to focus on the evaluation of the algorithms working with AASGs: Morwilog and Bidimac. In this case, the datasets used so far are no longer valid, because they have just one instance of each attack. We need several instances of them to see how the levels of pheromones and the probability parameters evolve. To solve this issue, we have created the *eventgen* datasets, presented in section 7.1.5.

The evaluation of the algorithms starts with the analysis of their time performance in section 7.3.1. We continue studying how the choosing probability evolves with respect to the found attacks in section 7.3.2. The effects of varying the parameters

of the algorithms are analyzed in section 7.3.3. Finally, the consequences of facing overlapped sequences of events representing an attack are addressed in section 7.3.4.

7.3.1 Time performance

To evaluate the time performance of the algorithms, we should check first the execution time of the evaluations done so far. The results are shown in Table 7.9. Each value of the execution time, expressed in seconds, is the resulting average from 10 executions. The exception is the HuMa dataset, whose execution time comes from a single execution.

The first observation about the results in Table 7.9 is that none of the executions takes a time longer than the period represented in each dataset. This constitutes a minimum requisite for applying Morwilog and Bidimac on real-time events. The computer used for the simulations, whose characteristics were indicated in page 169, is a working laptop with more than 4 years old. Times are thus expected to be much lower in a real implementation.

The execution times are in fact very low with respect to the period covered by the datasets, with the exception of the execution of Bidimac on HuMa. This execution took almost half of the time represented by the logs, probably because T_{max} is long and most of the branches in the AASG S2A are not matched by any event after κ_0 is matched, so Bidimac keeps looking for matches until the limit established by T_{max} is reached.

Otherwise, it is difficult to infer general conclusions about the execution time from the information given in the table: Bidimac is slower than Morwilog in some executions

Dataset	**AASG**	**Execution time (s)**		**Duration dataset**	**Duration attack**	**Number of logs**
		Morwilog	**Bidimac**			
inside1-NoMill	Mill-1	1.03	4.81	3h 13′ 53″	1h 19′ 45″	885
	Mill-2	1.20	1.41			
inside2	Mill-2	0.60	0.85	1h 43′ 15″	48′ 32″	494
ISCX	S2A	23.12	2,936.67 (∼49′)	24h	5h 53′ 31′	9,608
HuMa	S2A	21,509.40 (∼6h)	231,112.84 (∼2d 16h)	5d 15h	3d	146,748

Table 7.9: Execution time for Morwilog and Bidimac on the combinations of AASG and dataset used.

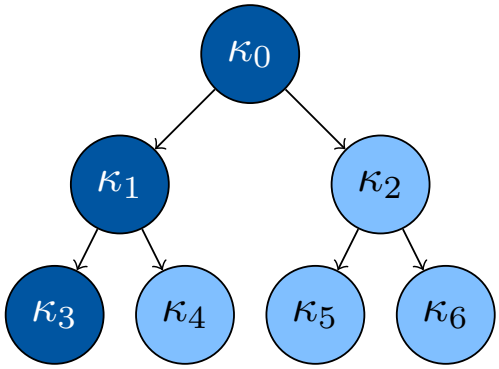


Figure 7.15: Formal representation of the AASG Eventgen. Nodes matched by the attack are colored in dark blue.

(e.g. inside1-NoMill with Mill-1 or ISCX), but both behave similarly in some others (e.g. inside1-NoMill with Mill-2 or inside2). This happens because there are many factors in play, such as the AASGs, the size of the dataset or the distribution of the attacks.

To be able to properly study the execution time, some *eventgen* datasets have been generated (see section 7.1.5). Each one of the generated datasets has only 1 type of multi-step attack of length 3 and maximum distance between the steps of 3 logs. The injections of the attack sequences in the dataset is calculated for resulting in around 0.5% of logs involved in an attack.

Respecting the AASG, a structure like the one shown in Figure 7.15 is generated, with the nodes in dark blue coding the events corresponding to the multi-step attack. The goal is just to check how the algorithms behave with a fixed structure of AASG being matched by the same attack. All the nodes in the AASG are related with their parents by the condition $f_1(ipsrc_a, ipsrc_b) = 1$, which means that the source IP address will be preserved for all the events in the matched sequences. We will be using this AASG, called AASG Eventgen, for the rest of the simulations in this chapter.

Under these conditions, the execution time has been evaluated for different sizes of the dataset and for different values of T_{max}. The results are represented in Figure 7.16 and Figure 7.17, respectively. Each represented point represents the average result from 10 different executions. In Figure 7.16, $T_{max} = 20s$. For both Morwilog and Bidimac, the execution time seems to be linearly proportional to the size of the dataset and the maximum search time T_{max}. However, while the two algorithms seem equally affected by the size of the dataset, with a slight advantage for Bidimac, the increment of the parameter T_{max} clearly affects Bidimac more than Morwilog.

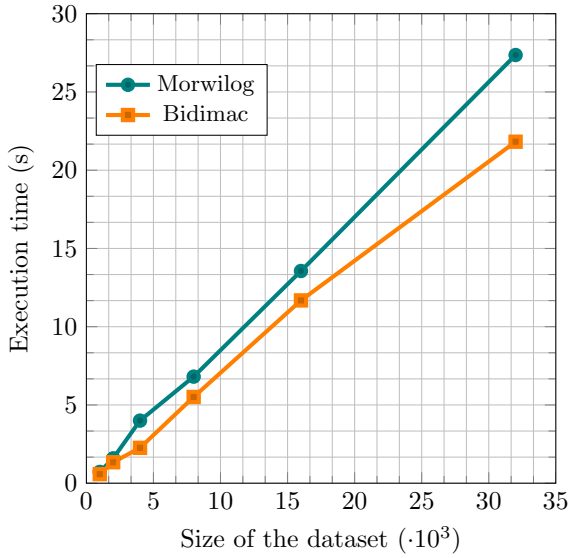


Figure 7.16: Execution time with respect to the size of the dataset.

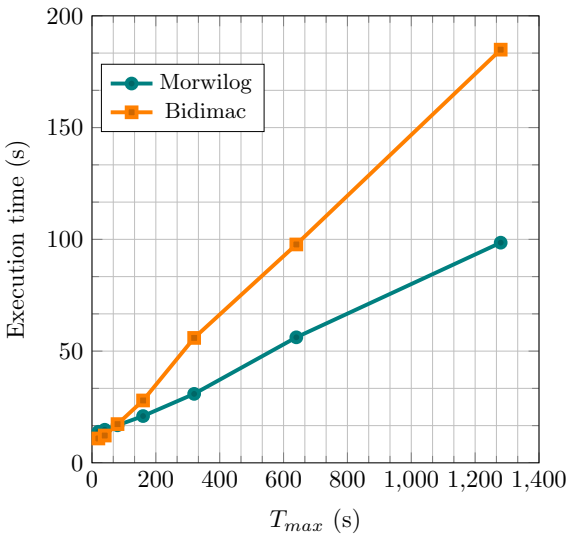


Figure 7.17: Execution time with respect to T_{max} in an *eventgen* dataset with 16000 logs.

This difference is not linked to the matched branches but to the non-matched ones. In Bidimac there is not a choice of path like the one we found in Morwilog. Therefore, sequences of nodes that are not found heavily penalize the algorithm, as it continues the search until the end of the period determined by T_{max}. This is the price to pay for being coherent with the statistics of the incoming events and trying to find all the possible hypotheses.

7.3.2 Evolution of the choosing probability

The levels of pheromones in a stigmergic AASG and the probability parameters in a Bayesian AASG are the elements that determine the importance of each path. We defined the choosing probability for both types of AASG in Equation 6.1 and Equation 6.13, respectively. To study the evolution of the strength of a branch over the others after the detection of several attacks, we define the choosing probability of one branch as:

$$P_{\boldsymbol{b}_\kappa} = \prod_{\kappa_n \kappa_m \in \boldsymbol{b}_\kappa} P(\kappa_n \kappa_m) \tag{7.1}$$

As it is represented in Figure 7.15, the attacks in the *eventgen* datasets are matched by the branch $[0, 1, 3]$ in the AASG Eventgen. We are interested in the choosing

probability of this branch, denoted as $P_{[0,1,3]}$. Several *eventgen* datasets have been created to study the evolution of this probability. They have a size of 1000 logs and different number of 3-step multi-step attacks, determined by the *eventgen* parameter *prop_attack* (see Table 7.6, page 176).

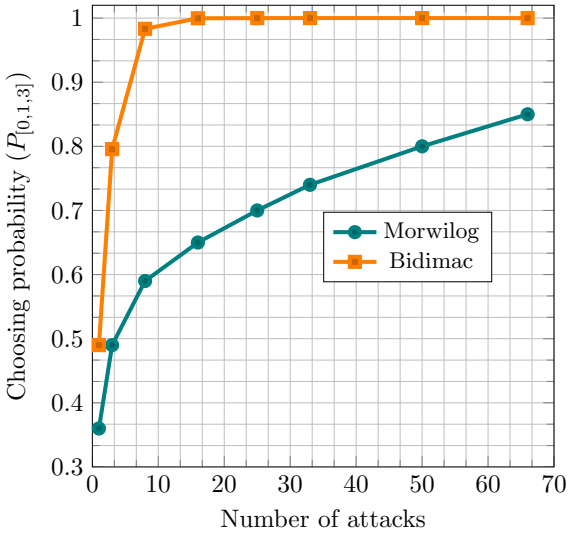


Figure 7.18: $P_{[0,1,3]}$ in AASG Eventgen with respect to the number of attacks in the case of continuous reinforcement (*eventgen* datasets).

The algorithms are first tested in a scenario of continuous reinforcement, like the one that was used for the theoretical analysis of the evolution of pheromones presented in page 144. The results are represented in Figure 7.18 for both Morwilog and Bidimac, with $T_{max} = 20s$. We see how $P_{[0,1,3]}$ grows faster in Bidimac and gets almost equal to 1 after 16 attack sequences. On its side, $P_{[0,1,3]}$ in Morwilog grows fast at the beginning but the growing rate gradually decreases as the number of attacks is incremented.

This is the same behavior we could see in the theoretical analysis, whose results were presented in Figure 6.11. There, we just focused on the choosing probability of one arc, which in that case also corresponded to the choosing probability of the matched branch. Having a choosing probability in Morwilog that does not get up to the maximum of 1 avoids the *morwi* to get stagnated in the same path, even if it has been matched many times. Anyways, once the choosing probability is high enough, the analyst can consider that the hypothesis represented in this branch has been confirmed and she can retire this branch from the AASG.

Another aspect to study is the evolution of the choosing probability when found sequences are not confirmed as attacks. The results for a scenario of continuous penalization are represented in Figure 7.19. This scenario, where only false positive sequences matching branch $[0, 1, 3]$ are found, is similar to the one proposed in page

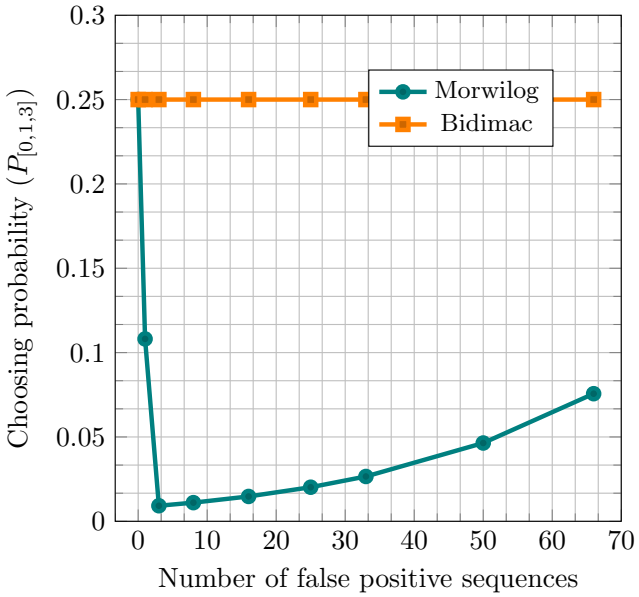


Figure 7.19: $P_{[0,1,3]}$ in AASG Eventgen with respect to the number of false positive sequences in the case of continuous penalization (*eventgen* datasets).

145. We see how $P_{[0,1,3]}$ does not vary at all in Bidimac. As we explained in section 6.3.4, the probability parameters in the Bayesian AASG are never affected by false positives, only by confirmed attacks. This is done to be coherent with the statistics of attack sequences.

Conversely, the decrement of pheromones in Morwilog leads to a big drop of $P_{[0,1,3]}$ right after a few false positive sequences have been found. However, the value of $P_{[0,1,3]}$ starts then to slowly rise as more false positive sequences match the branch $[0, 1, 3]$. This behavior is explained by the minimum level of pheromones τ_{min} and the evaporation mechanism. The levels of pheromones in the arcs of the branch $[0, 1, 3]$ never goes below τ_{min}. Once this limit is reached, these levels of pheromones do not change, but the levels of pheromones of the rest of the arcs in the stigmergic AASG decrease due to the evaporation mechanism, resulting in an increase of $P_{[0,1,3]}$.

7.3.3 Values of parameters

The evolution of the choosing probability $P_{\boldsymbol{b}_\kappa}$ can be controlled thanks to some of the parameters of the algorithms. In Bidimac, the only parameter we have apart from T_{max} is the learning rate η. On the contrary, in Morwilog we have five parameters excluding T_{max}, among which only the evaporation rate (ρ) and the spreading of the $\Delta\tau^{+,-}$ function (w) actually affect $P_{\boldsymbol{b}_\kappa}$. The rest ($\tau_0$, $\Delta\tau_0^+$ and τ_{min}) are just used as reference points to determine the range of values taken by the levels of pheromones.

The evolution of $P_{[0,1,3]}$ in the AASG Eventgen with respect to the learning rate η is shown in Figure 7.20, after having detected 3 and 16 attack sequences in a set of *eventgen* datasets with 1000 logs. We see that η allows an ample variation of $P_{[0,1,3]}$ after 3 attacks have been found. However, after the detection of 16 attacks, $P_{[0,1,3]}$ is equal to 1 for $\eta > 2$ and really high for the values below. This means that the possibility of personalizing the convergence speed of the choosing probability to a maximum value using η is just limited to the first few found attacks.

On the contrary, in Morwilog we have the exactly opposite situation: the variation of the parameters ρ and w has a bigger influence on $P_{[0,1,3]}$ if more attack sequences are found. This can be seen in the three-dimensional graphs shown in Figures 7.21, 7.22 and 7.23, which represent the $P_{[0,1,3]}$ with respect to ρ and w for a set of *eventgen* datasets with 1000 logs and 3, 16 and 66 attacks, respectively. The amplitude of choice in the convergence of $P_{[0,1,3]}$ is clear in the results from Figure 7.23, where even after 66 attacks we could regulate the parameters to get a $P_{[0,1,3]}$ in a range from 0.4 to almost 1.

7.3.4 Overlapped attacks

The *eventgen* datasets used so far contain multi-step attacks whose steps are really close in time from each other, having no more than 3 logs between them. In this section, we want to analyze the behavior of Morwilog and Bidimac when the distance between the steps is higher and, in consequence, the attacks gets overlapped in time.

For doing so, we have generated five *eventgen* datasets of 1000 logs with 33 attacks

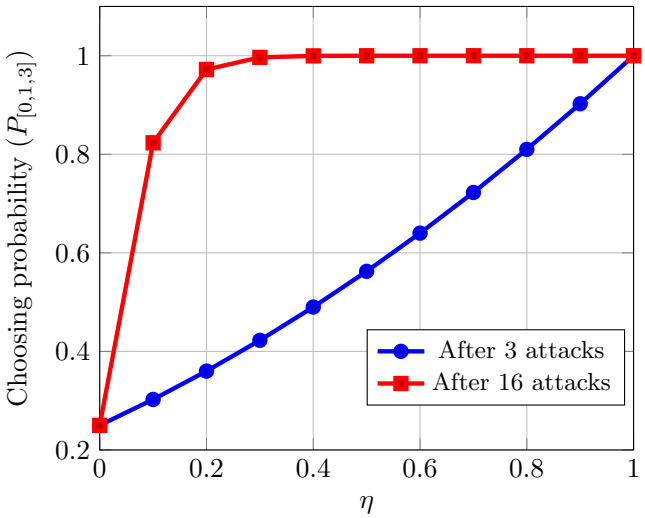


Figure 7.20: $P_{[0,1,3]}$ in AASG Eventgen with respect to the parameter η in Bidimac after 3 and 16 attacks (*eventgen* datasets).

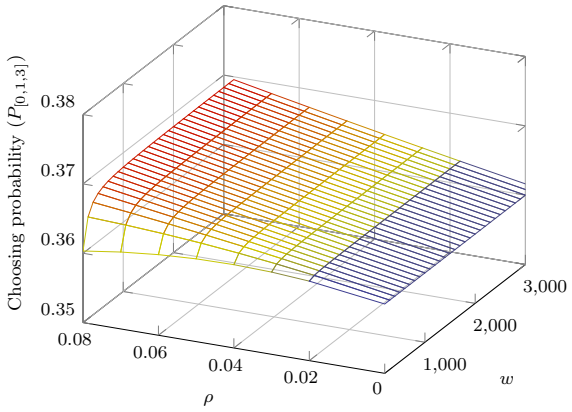


Figure 7.21: $P_{[0,1,3]}$ in AASG Eventgen vs. parameters ρ and w in Morwilog after 3 attacks (*eventgen*).

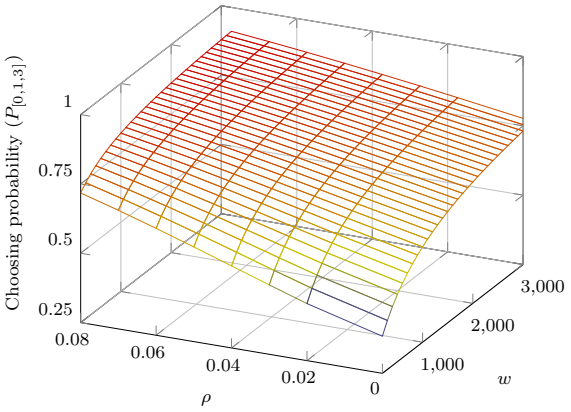


Figure 7.22: $P_{[0,1,3]}$ in AASG Eventgen vs. parameters ρ and w in Morwilog after 16 attacks (*eventgen*).

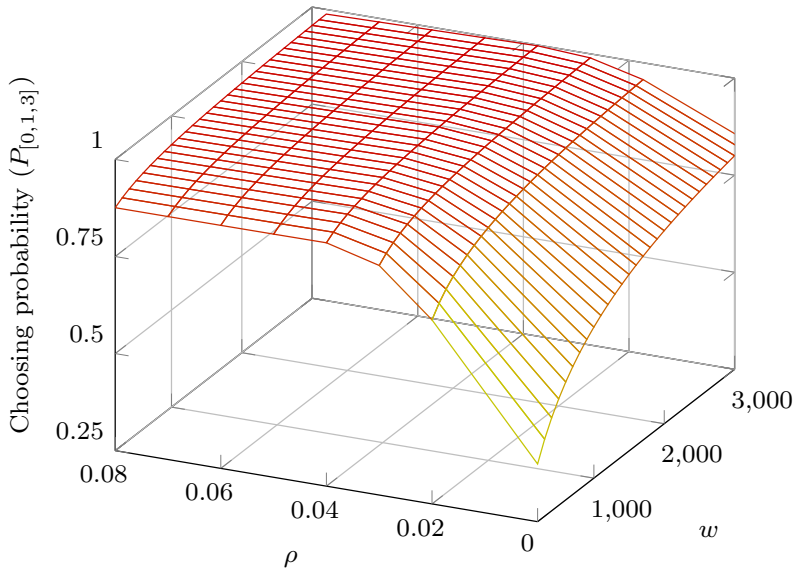
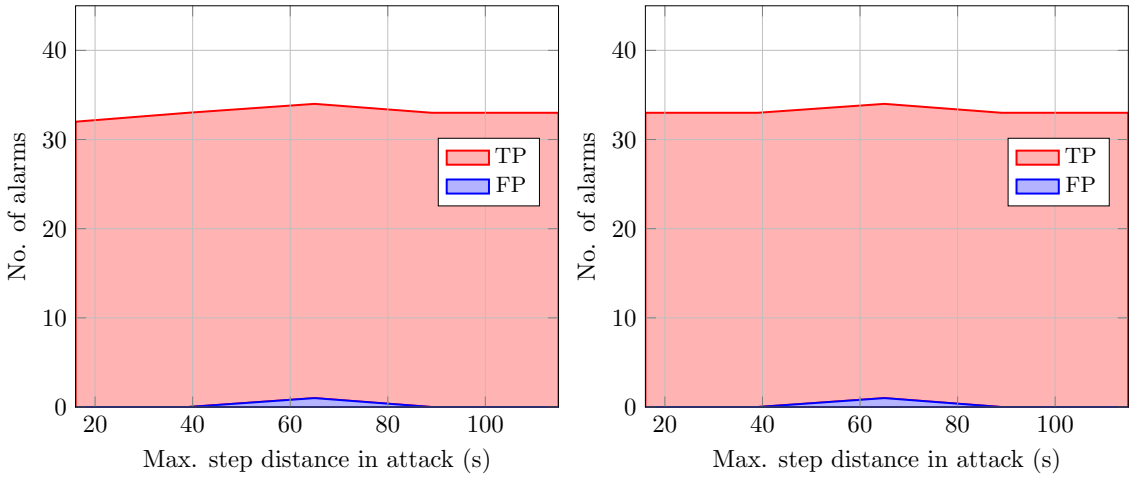


Figure 7.23: $P_{[0,1,3]}$ in AASG Eventgen vs. parameters ρ and w in Morwilog after 66 attacks (*eventgen*).

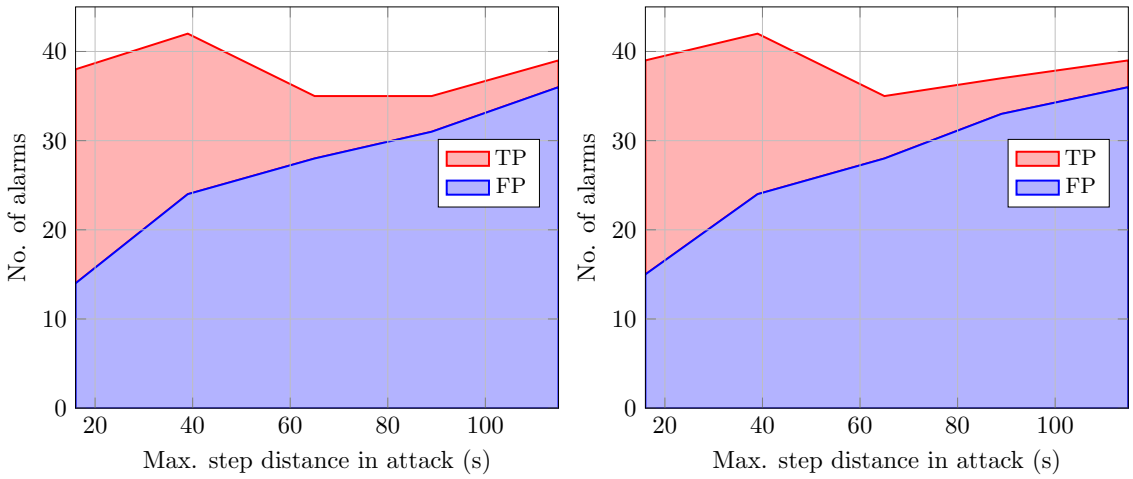
(*prop_attack*= 10% and 3 steps per attack) and different maximum distances in number of logs (*max_step*) between the steps of the attacks: 10, 25, 50, 75 and 100, which result in a maximum distance in seconds of 16, 39, 65, 89 and 115, respectively. We have chosen $T_{max} = 120s$ for all the executions.

Two metrics are considered to evaluate the algorithms: the number of alarms corresponding to attacks (true positives or TP) and the numbers of alarms rejected by the analyst as false positives (FP). The addition of these two metrics is the total number of alarms.



(a) Morwilog. (b) Bidimac.

Figure 7.24: TP and FP with respect to the maximum step distance in the attacks in *eventgen* datasets with AASG Eventgen



(a) Morwilog. (b) Bidimac.

Figure 7.25: TP and FP with respect to the maximum step distance in the attacks in *eventgen* datasets with AASG Eventgen-Noipsrc

The results of the execution of Morwilog and Bidimac are represented in Figures 7.24a and 7.24b, respectively. There is only a single difference between the execution of the two algorithms: in the first dataset, with a maximum difference between the steps of 16 seconds, Morwilog misses one of the attacks. This is the consequence of a *morwi* taking one of the alternative branches in the AASG that ends up without generating any alarm. Regarding the false positive alarms, there is just one for both algorithms, in the third dataset. As the datasets are randomly generated, it can happen that the position of certain sequence of actions within the dataset can be at the origin of a false

positive alarm in one of the executions and not in the other ones.

There is not a visible impact of the presence of overlapped attacks in these results. The reason is that our AASG is very well defined, with conditions of preservation of the source IP address in each of the branches. But we can try to remove this condition ($f_1(ipsrc_a, ipsrc_b) = 1$), so the relationship between the events matching the sequence is less rigid. This modification results in a new AASG called Eventgen-Noipsrc.

The results in terms of TP and FP on the same datasets are shown in Figure 7.25a for Morwilog and in Figure 7.25b for Bidimac. Now the effect of overlapped attacks is more clear, resulting in an increase of false positive alarms for both algorithms. Notice that, apart from some minor variation probably due to the random choices of the *morwis*, the results from Morwilog and Bidimac are equivalent.

These results suggest that detection mainly relies on the design of the AASG. The choice of the algorithm should then be made thinking on a) time performance with respect to the input data and the used T_{max}, and b) number of alarms generated by the system.

7.4 Summary

This chapter has addressed the evaluation done on the AASGs and the algorithms conceived for working with them, Morwilog and Bidimac. We have started introducing the datasets used in this evaluation: DARPA 2000 (section 7.1.1), ISCX (7.1.2) and HuMa (7.1.3). The first two are the most popular public datasets containing multi-step attacks, while the third one is a private dataset developed under the HuMa project. To be treated by our algorithms, the events in these datasets have to be normalized to a common format, that has been introduced in section 7.1.4. As the mentioned datasets have only one multi-step attack each, we have developed a set of artificial datasets, called *eventgen* datasets, to evaluate the algorithms using several instance of the same multi-step attack. The details of these datasets have been presented in section 7.1.5.

Evaluation has been divided in two parts: the evaluation of the AASGs and the evaluation of the algorithms. The first part has started in section 7.2.1 with the evaluation of the AASG Mill-1 (see section 5.3) on the dataset DARPA 2000 inside1. We have then proposed in section 7.2.2 a new AASG, Mill-2, that works with both DARPA 2000 inside1 and inside2. A simple aggregation algorithm to reduce the number of logs in a dataset has been proposed in section 7.2.3 and applied to the three datasets used in this first part. After aggregation, the multi-step attacks contained in

the ISCX and HuMa datasets have been found using the same AASG, called S2A, in both cases. This AASG and the results of its evaluation are shown in section 7.2.4.

For the second part of the evaluation, the evaluation of the algorithms, we have used the *eventgen* datasets. We have started by evaluating the time performance of both Morwilog and Bidimac in section 7.3.1. The following aspect to evaluate, in section 7.3.2, has been the evolution of the choosing probability in the scenarios of continuous reinforcement and continuous penalization. We have then evaluated in section 7.3.3 how this probability is affected by the choice of the parameters in the algorithms. Finally, we have studied the effect of overlapped attacks on the two systems in section 7.3.4.

In the next chapter, we will present the last of the contributions of this thesis: SimSC, a model for the visual investigation of attack scenarios.

Chapter 8

Visual investigation of attack scenarios

Contents

8.1 SimSC 204
8.2 Implementations 206
8.3 Raw-SimSC 214
8.4 Summary 216

"You see, but you do not observe."
— Arthur Conan Doyle, *A Scandal in Bohemia*

Up to this point, we have not paid much attention to the phase of investigation of past attacks, the one leading to the identification of the different multi-step attack cases that form an AASG. In this chapter, we propose a model for the investigation of attack scenarios based on the idea of similarity: SimSC (Similarity-based Scenario Creation). The difference with the methods reviewed in section 4.3.1 is that we have limited ourselves to work with raw logs, in plain text. This constraint limits the analysis to attributes that can be automatically extracted, but it allows working with heterogeneous sets of logs that have not been processed. Attack scenarios found using SimSC are intended to be used in the creation of AASGs, the same as the infection of Mill in LLDoS 1.0 was used to create the AASG Mill-1 (see section 5.3).

SimSC is directly linked to the contributions made by the other partners in the HuMa project, and conceived to work and be complemented by them. An introduction to HuMa reference architecture can be found in the work published in the proceedings of the 10th International Symposium on Foundations and Practice of Security (FPS) [Navarro 2017] and presented by the author of this thesis in Nancy (France) in October 2017. Since then, methods proposed within HuMa have evolved and new ones have been developed. The work made by the industrial partners cannot be publicly disclosed, but new publications by the academic partners are expected.

This chapter is divided as follows. We explain in section 8.1 how SimSC works for later presenting two working implementations of the model in section 8.2: as part of the HuMa project and on the medical information platform GenIDA. Finally, in section 8.3, Raw-SimSC, a version conceived to work with any dataset, is presented and it is tested with events from DARPA 2000.

8.1 SimSC

The format of logs is dependent on the system generating them. If we compare the Apache log shown in Figure 3.1 (page 26) and the log representing an IDS alert in Figure 8.1, we see how their aspect is completely different. These logs could even be related, as both represents a SQL Injection attack coming from the same IP address (192.168.2.112). SimSC offers a mechanism to link these two logs without the need for normalizing them into a common format.

```
[**] [1:2010937:2] ET POLICY Suspicious inbound to mySQL port 3306
[**] [Classification:  Potentially Bad Traffic] [Priority:  2] Aug
6 08:21:40 192.168.2.112:53508 -> 192.168.5.122:3306 TCP TTL:38
TOS:0x0 ID:31936 IpLen:20 DgmLen:44 ******S* Seq:  0x69268A6B Ack:
0x0 Win:  0xC00 TcpLen:  24 TCP Options (1) => MSS: 1460 [Xref =>
http://doc.emergingthreats.net/2010937]
```

Figure 8.1: An IDS log from ISCX indicating a possible SQL Injection attack.

There exist two types of attribute that preserve their format and can be automatically found: IP addresses and timestamps. Both are present in almost every log contained in the studied datasets. They are the types of attribute used by SimSC. We could think of other types of attribute with a fixed format, such as MAC addresses, but they are not as common as the selected ones.

For the extraction of IP addresses, we identify their format as four numbers between 0 and 255 separated by points. From the log in Figure 3.1 we have the address 192.168.2.112, while for the one in Figure 8.1 we got the same one and also 192.168.5.122. If we automatically extract the IP addresses, we cannot make the distinction between which is the source and which is the destination, as we know nothing about the position of each attribute in the log. The result of this extraction is thus an unordered set of IP addresses.

Respecting the timestamp, several formats exist. We have built a set of regular expressions, shown in Table 8.1, to identify all the timestamp formats that we have found in the studied datasets. Once extracted, timestamps are converted into POSIX

ID	Regex	Example
1a	`(... \d\d \d\d:\d\d:\d\d)\D`	Jan 24 03:34:56
1b	`(... \d \d\d:\d\d:\d\d)\D`	
2	`(\d\d\d\d-\d\d-\d\d \d\d:\d\d:\d\d)`	2019-01-24 03:34:56
3a	`\D(\d\d/.../\d\d\d\d:\d\d:\d\d:\d\d)\D`	24/Jan/2019:03:34:56
3b	`\D(\d/.../\d\d\d\d:\d\d:\d\d:\d\d)\D`	
4a	`\D(\d\d/\d\d/\d\d\d\d \d\d:\d\d:\d\d ..)\D`	24/01/2019 3:34:56 am
4b	`\D(\d/\d\d/\d\d\d\d \d\d:\d\d:\d\d ..)\D`	
4c	`\D(\d\d/\d\d/\d\d\d\d \d:\d\d:\d\d ..)\D`	
4d	`\D(\d/\d\d/\d\d\d\d \d:\d\d:\d\d ..)\D`	
5a	`\D(\d\d ... \d\d\d\d \d\d:\d\d:\d\d)\D`	24 jan 2019 03:34:56
5b	`\D(\d ... \d\d\d\d \d\d:\d\d:\d\d)\D`	

Table 8.1: Regular expressions used for the extraction of timestamps.

format to be easily compared. Again, it is possible to find several timestamps in a log, referring to different moments such as the occurrence of the event, the moment when the log was stocked in the file, etc. In SimSC, we need to attach each log to a single moment, so we need a unique timestamp. We have tested the automatic extraction of timestamps in the HuMa dataset, that has also been parsed (see section 7.1.4). The result is that for each raw log, the timestamp with the minimum value among the ones automatically extracted always corresponds to the one selected for the parsed log. Therefore, the minimum timestamp is the one used in SimSC.

For example, the extracted timestamps in Figure 3.1 and Figure 8.1 are, respectively, "06/Aug/2018:08:16:00", matching to regular expression 3a in Table 8.1, and "Aug 6 08:21:40", matching to 1b.

The output of SimSC is a graph that is built using the extracted IP addresses and timestamps. Each node in the graph represents a log. An edge between two logs is built depending on their difference in time and the IP addresses they have in common. The sets of IP addresses extracted from the two logs are called A_{ip} and B_{ip}. The minimum number of IP addresses in common to create an edge is determined by the parameter n_{ip}. For any $n_{ip} \geq 0$, an edge is built if the difference in time is below a certain threshold T_{max} and one of the following conditions is met: a) $A_{ip} \subseteq B_{ip}$, b) $B_{ip} \subseteq A_{ip}$ or c) $|A \cap B| \geq n_{ip}$. In Algorithm 8.1, the steps of this process are detailed in the form of pseudocode.

Algorithm 8.1 SimSC

Require: $E_{in} \subseteq \mathbb{E}$ ordered in time, T_{max}, n_{ip}
Ensure: Graph $G = (list\_nodes, list\_edges)$

$list\_nodes \leftarrow E_{in}$
$list\_edges \leftarrow \emptyset$
for each $\boldsymbol{e}_i$ in E_{in} **do**
    $A_{ip} \leftarrow$ Extracted IP addresses from $\boldsymbol{e}_i$
    $t_a \leftarrow min$(Extracted timestamps from $\boldsymbol{e}_i$)
    if $t_a > 0$ **then**
        for each $\boldsymbol{e}_j$ in $E_{in}|j > i$ **do**
            $B_{ip} \leftarrow$ Extracted IP addresses from $\boldsymbol{e}_j$
            $t_b \leftarrow min$(Extracted timestamps from $\boldsymbol{e}_j$)
            if $t_b > 0$ and $t_b - t_a < T_{max}$ **then**
                if $A_{ip} \subseteq B_{ip}$ **or** $B_{ip} \subseteq A_{ip}$ **or** $|A \cap B| \geq n_{ip}$ **then**
                    Append edge $\overline{\boldsymbol{e}_i \boldsymbol{e}_j}$ to $list\_edges$
                end if
            else
                Break
            end if
        end for
    end if
end for
return $G = (list\_nodes, list\_edges)$

8.2 Implementations

The development of SimSC has led to two working implementations. The first implementation (section 8.2.1), HuMa-SimSC, is the one designed to work within the HuMa architecture [Navarro 2017] and it is at the origin of the conception of SimSC. The second one (section 8.2.2) has been developed by Timothée Mazzucotelli to work with web logs generated by the web medical platform GenIDA.

8.2.1 HuMa-SimSC

HuMa-SimSC is the implementation giving birth to SimSC. It was conceived to be integrated in the HuMa architecture (see section 4.5), where the analysis is organized in three layers, as we mentioned in section 4.5. These layers are the ones shown in Figure 8.2. HuMa-SimSC works in the top layer, helping the analyst to build the assessment model of the attack. It takes as input the output of a previous process of filtering and clustering, as it is represented in Figure 8.3. More precisely, there are two previous steps, that are described in the publication about the HuMa architec-

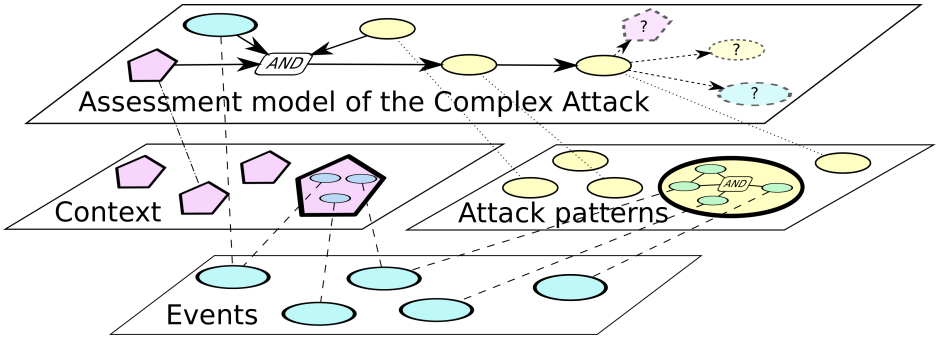


Figure 8.2: Diagram of the layers in HuMa architecture [Navarro 2017].

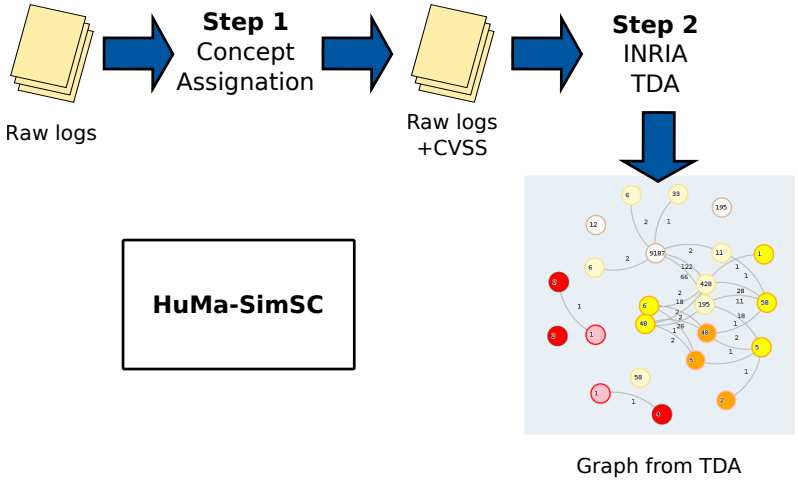


Figure 8.3: The integration of HuMa-SimSC in the HuMa architecture.

ture [Navarro 2017]: assignation of vulnerability-based concepts and Topological Data Analysis (TDA).

The first step (Step 1) consists on the extraction from the logs of general concepts based on the CVSS code[1]. The catalog of concepts and the automatic process for assigning them to the logs have been developed by one of the industrial partners of the HuMa project. Their details must thus be kept confidential. The output of this process is the same list of raw logs from the input with several abstract concepts appended to each log as additional attributes. Logs related to similar vulnerabilities share similar concepts.

This output is then processed using an implementation of TDA (Step 2) developed by a research team based at the INRIA (French Institute for Research in Computer Science and Automation) in Nancy (France) [Coudriau 2016]. The objective of TDA

[1]The Common Vulnerability Scoring System (CVSS) is a standard to scoring vulnerabilities in computer networks [Mell 2007].

[Offroy 2016] is to reduce the high dimensionality of data by decomposing the space in overlapped hypercubes and clustering the data laying in each of them. The connection of common points in different hypercubes results in a graph at the output.

Each node of the graph returned by TDA is a cluster containing several logs. It is here where HuMa-SimSC comes into play, as a sort of 'zooming' tool to analyze the logs contained in each cluster with up to 1000 logs. It works on the raw logs, whose content is preserved. The color assigned to each log in the visual representation of HuMa-SimSC corresponds to the CVSS assigned in Step 1.

In Figures 8.4, 8.5 and 8.6, we show three examples of output returned by HuMa-SimSC when analyzing three clusters issued by TDA.

First of all, each ensemble of nodes shown in Figure 8.4 belongs to a potential DoS

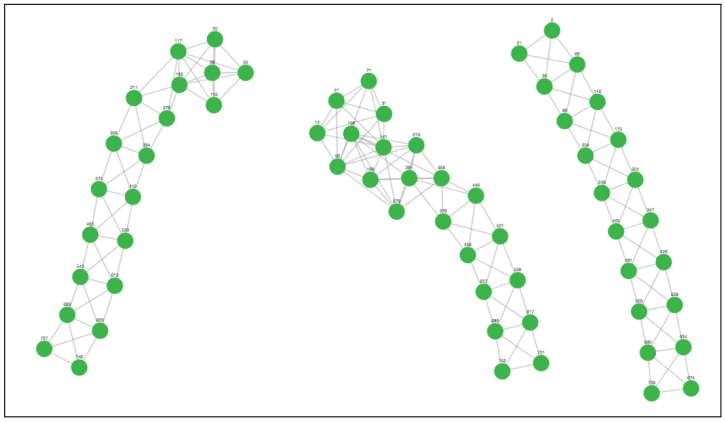


Figure 8.4: Three potential DoS attempts in HuMa-SimSC. $T_{max} = 10s$ and $n_{ip} = 2$.

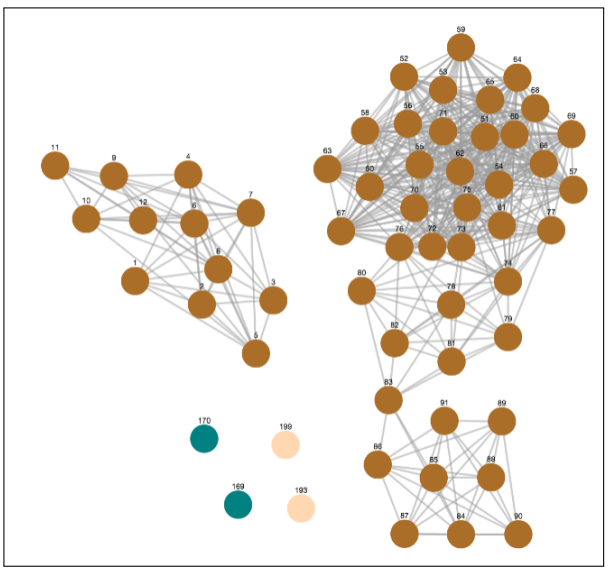


Figure 8.5: Two port scans in HuMa-SimSC. $T_{max} = 5s$ and $n_{ip} = 3$.

attempt. We use $T_{max} = 10s$ and $n_{ip} = 2$ to create them. All the nodes represent logs coming from a firewall and containing the message "possible attack on capacity (host)", which indicates a threat against the maximum capacity of the destination. These three potential attacks are performed from different IP addresses and have the same mail server as target.

In Figure 8.5, there are two ensembles of logs in color brown that correspond to two different port scans targeting the same mail server. We have used $T_{max} = 5s$ and $n_{ip} = 3$ to generate the graph. Each scan is executed from a different IP address: the one represented by the logs on the left from an external one and the other, from a local IP address. The isolated nodes in the lower left corner correspond to logs generated by the Centreon monitoring system. The turquoise ones do not have any IP address and the light pink ones have just one each corresponding to different machines.

Finally, we show in Figure 8.6 two screenshots associated to the same example. All the logs in the images represent the execution of Linux CRON tasks. These tasks are always executed at regular intervals. In the image on the left, we execute HuMa-SimSC with $n_{ip} = 0$, as CRON logs do not have any associated IP address, and $T_{max} = 59s$. Only those logs representing the same task, and thus being coincident in timestamp, are linked together. However, if we increase the time to $T_{max} = 60s$, all the logs are chained together, as we can see in the image on the right side. This means that they represent periodic tasks intended to be executed each minute. This result is very interesting for the HuMa project as many malicious actions, such as communications with a Command & Control server, are periodically performed at regular intervals.

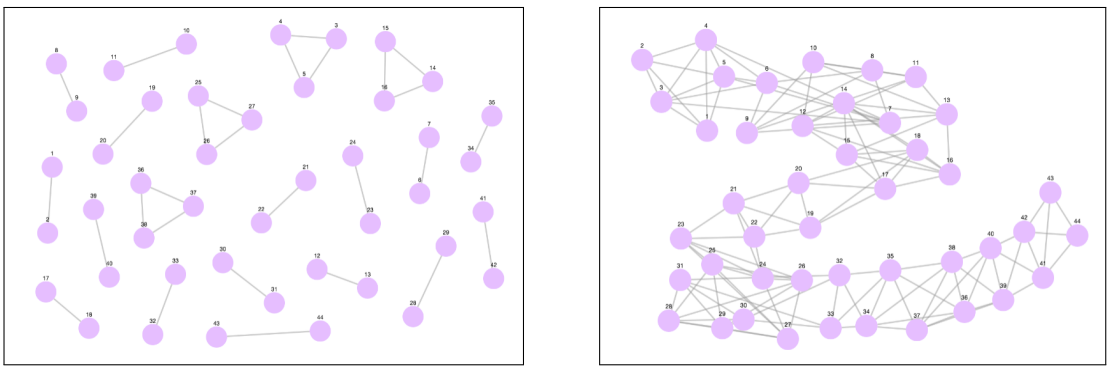


Figure 8.6: The regularity of interval between periodic Linux CRON tasks in HuMa-SimSC. Left: $n_{ip} = 0$ and $T_{max} = 59s$. Right: $n_{ip} = 0$ and $T_{max} = 60s$.

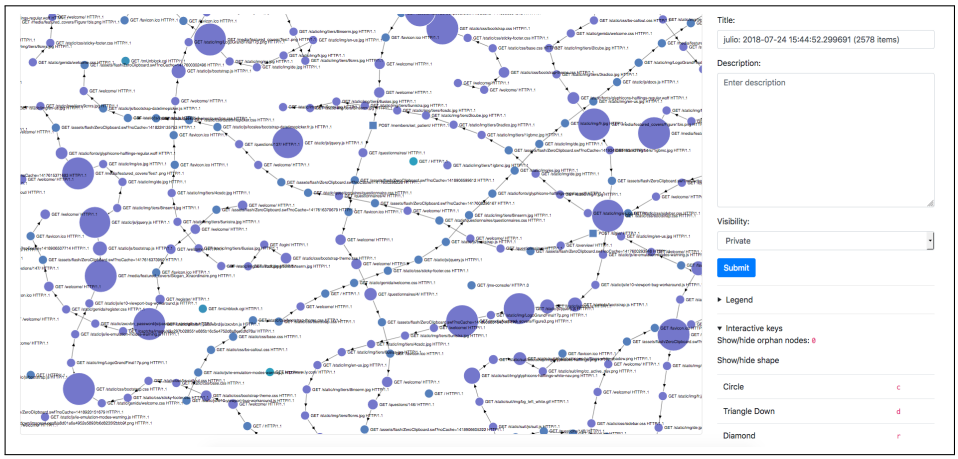


Figure 8.7: Screenshot of Logan interface.

8.2.2 Logan

GenIDA (Genetically determined Intellectual Disabilities and Autism Spectrum Disorders) is a project started in 2014 by Jean-Louis Mandel, member of the IGBMC (*Institut de Génétique et de Biologie Moléculaire et Cellulaire*) in Strasbourg. Its goal is the collection and analysis of medical information about genetic intellectual disabilities (ID) and autism spectrum disorders (ASD). The collection of data is done through a website[2] that allows the participation and sharing of families affected by one of these diseases.

One of the developer of the GenIDA website, Timothée Mazzucotelli, has adapted SimSC to work with the logs generated by the Apache server hosting the website. The resulting interface, that could be adapted to any kind of Apache web log, is called Logan. A screenshot of the interface of Logan is shown in Figure 8.7.

There are three characteristics of each node that give information about the represented log:

- Its **size** is proportional to the number of bytes in the response sent by the web server.
- Its **color** depends on the returned HTTP status code: purple for 100 and 200; blue for 300 and 400, and turquoise for 500 and 600. These sets of codes are considered as representing activities with *low*, *medium* and *high* risk, respectively.
- Its **shape**, that represents the type of HTTP request by the codes in Table 8.2.

Another adaptation of Logan with respect to the basic SimSC is that edges are now directed according to the time order of connected logs. Regarding the IP addresses,

[2] https://genida.unistra.fr/

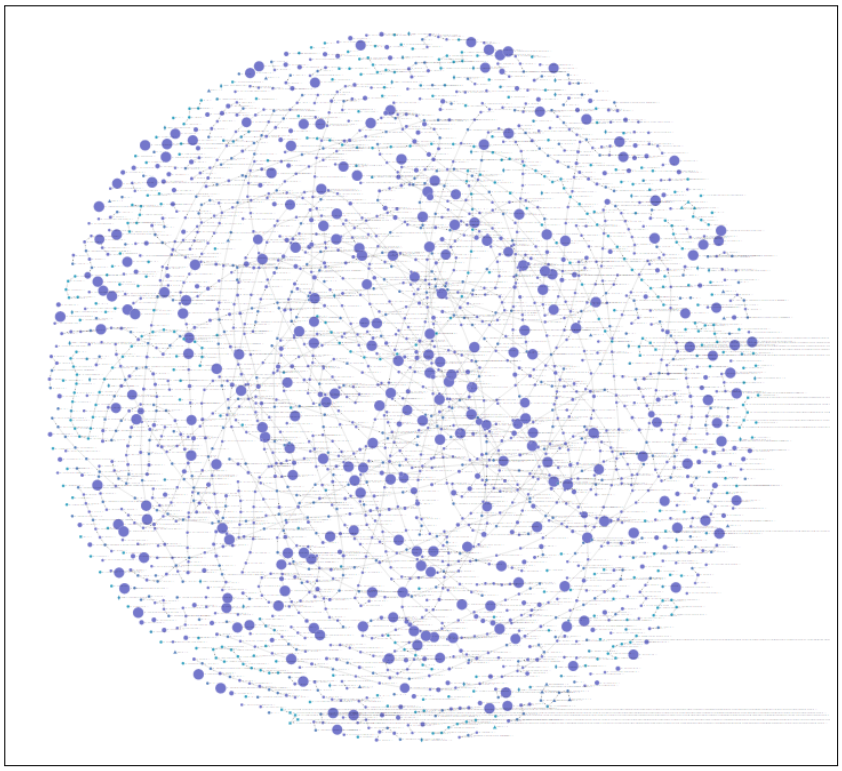


Figure 8.8: The global graph in Logan.

Apache logs only contain the one from the user doing the request, so we use $n_{ip} = 1$.

To begin the inspection of the logs, the analyst has to specify the period she wants to analyze. The amount of logs in a certain period of time depends on how many users access to the website but also on the logging configuration in the server, that determines which actions are recorded as logs. In the case of GenIDA, if we execute Logan for the period from the 23/09/2014 to the 2/11/2014, we obtain the graph shown in Figure 8.8.

Some transformation is needed to comfortably dive in this graph. One of the functions in Logan is to filter the visualized logs by their HTTP status code. Web

Shape	**HTTP request**
Circle	GET
Square	POST, PUT, PATCH
Triangle Up	HEAD
Triangle Down	OPTIONS
Cross	DELETE
Diamond	CONNECT, TRACE

Table 8.2: Shapes of nodes in Logan, according to the HTTP request.

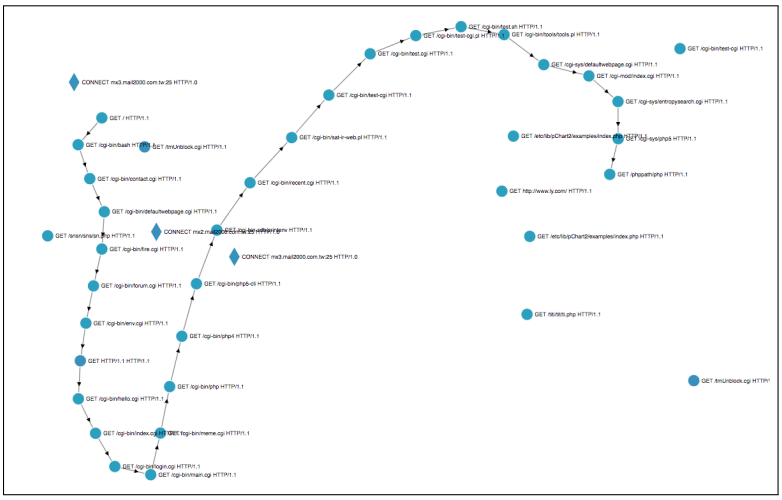


Figure 8.9: Screenshot of a high risk scenario in Logan.

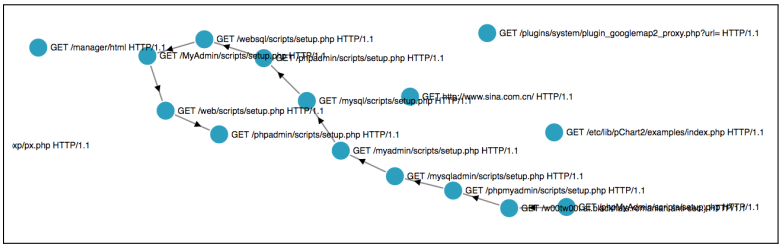


Figure 8.10: Screenshot of Logan with a scenario where a user test to execute a script.

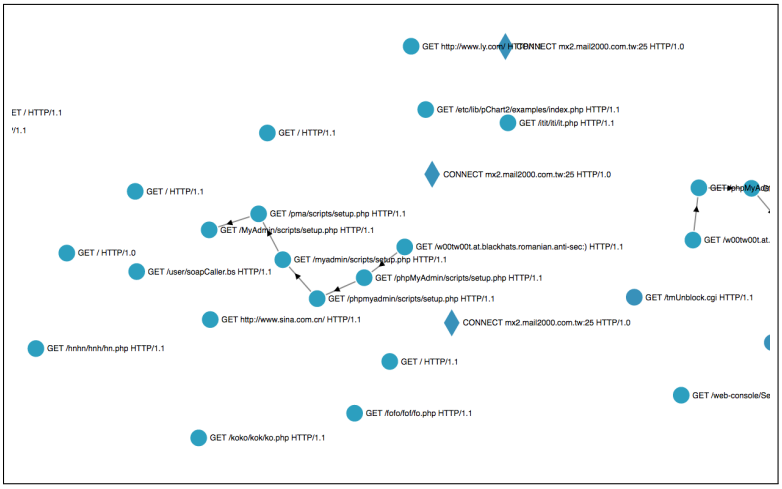


Figure 8.11: Screenshot of Logan with script testing using ZmEu scanner.

threats are most of the time identified by logs representing a high risk, the ones with codes 500 and 600. If we select only high risk logs for the visualization, we can now better identify the scenarios of actions performed by the same user, such as the one in Figure 8.9.

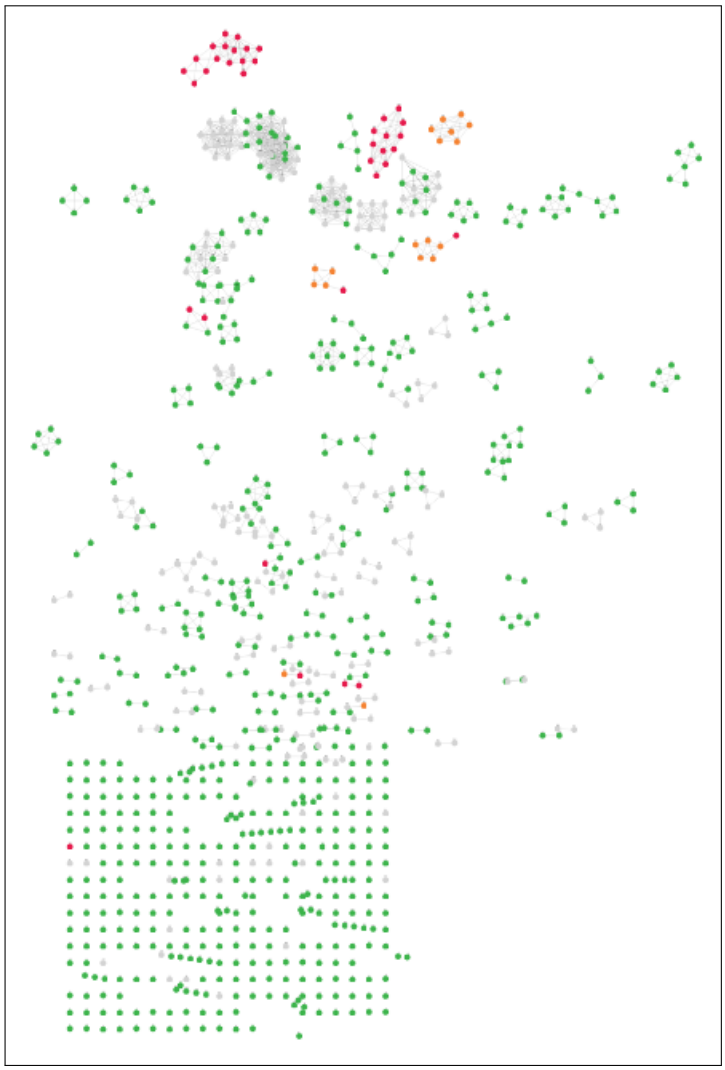


Figure 8.12: The events in DARPA 2000 inside1 represented in Raw-SimSC. $T_{max} = 8s$ and $n_{ip} = 2$.

An example of web attack found using this method is shown in Figure 8.10. We clearly see that the attacker has searched for the PHP script 'setup.php' in several paths of the URL. This is the typical behavior of the ZmEu scanner[3]. The presence of this bot in the high risk logs is confirmed by the sequence of requests shown in Figure 8.11, containing the message "*w00tw00t.at.blackhats.romanian.anti-sec:)*". Notice that we are dealing with real requests coming from the outer world, and not with simulated data.

[3] https://ensourced.wordpress.com/2011/02/25/zmeu-attacks-some-basic-forensic/

8.3 Raw-SimSC

The mentioned implementations of SimSC, HuMa-SimSC and Logan, are conceived for working with a specific type of log. The process behind is the same as the one shown in Algorithm 8.1, but the interfaces include additional elements to ease visualization. However, we have implemented a version of SimSC, called Raw-SimSC, where the user can upload any raw log file, execute the algorithm and visualize the results. Nodes have no colors this time, except for the RealSecure alerts listed in Table 7.1, page 171. The names of the listed alerts are searched in the raw log and, if one is found, a color is assigned according to the associated risk level: red for 'high', orange for 'medium' and green for 'low'. The rest of the nodes are colored in gray.

Although there are no theoretical limitations about the number of logs SimSC can manage, there is a practical limitation in the visualization using Raw-SimSC, which can represent up to 1000 nodes. This means that a full representation of ISCX (section 7.1.2) or HuMa (7.1.3) is not feasible, as it is indicated by their size in Table 7.9, page 192. However, the DARPA 2000 dataset is not that big and we can test Raw-SimSC with it.

In Figure 8.12 we show a screenshot taken after the execution of Raw-SimSC in inside1, with $T_{max} = 8s$ and $n_{ip} = 2$. Of course, the amount of logs in the dataset is

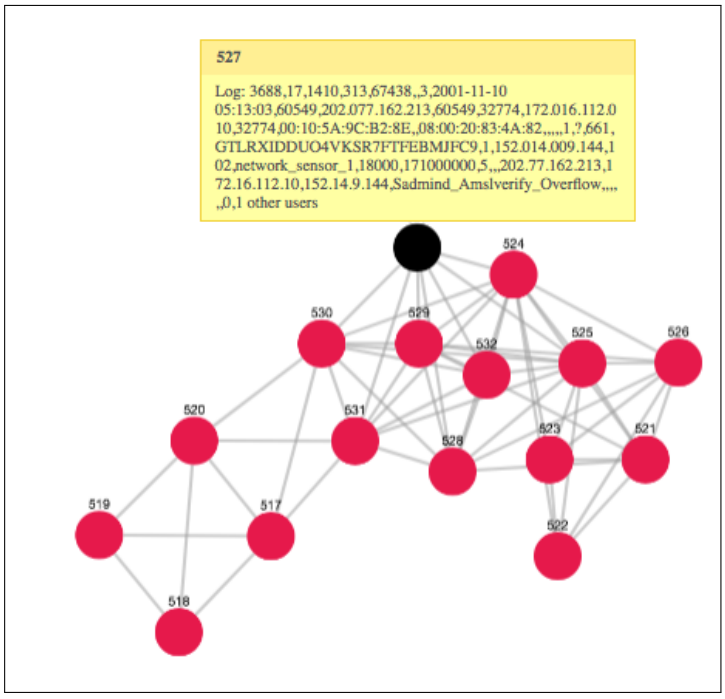


Figure 8.13: A 'Sadmind_AO' scenario extracted by Raw-SimSC on the events in DARPA 2000 inside1.

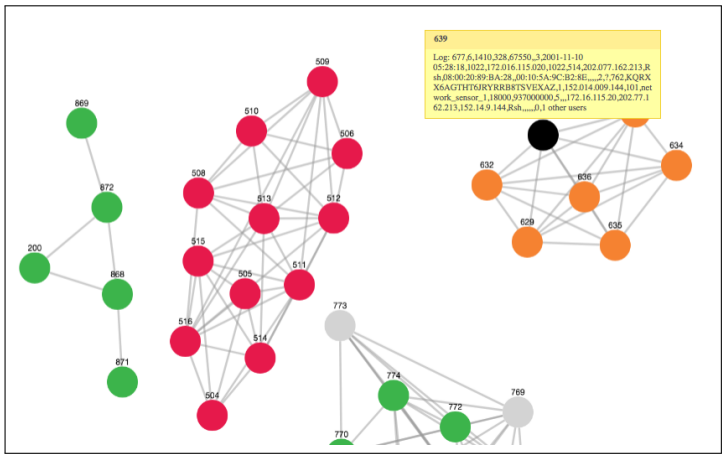


Figure 8.14: A 'Sadmind_AO' scenario and a 'Rsh' scenario extracted by Raw-SimSC on the events in DARPA 2000 inside1.

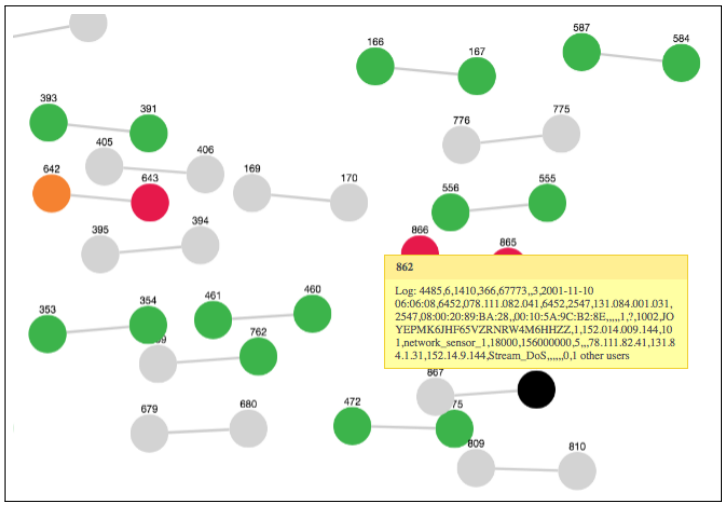


Figure 8.15: An isolated 'Stream_DoS' alert extracted by Raw-SimSC on the events in DARPA 2000 inside1.

too high to see the details. But as some of the nodes are colored according to the risk level in the RealSecure alert, we can locate the 'hot spots', those parts of the graph where there are alerts with high risk level (nodes in red).

Zooming in one of these areas, the one on the top of Figure 8.12, we can find that 'Sadmind_AO' and 'Admind' alerts that are related have been connected in the same scenario, as we can see in Figure 8.13. If we click in a node, the corresponding raw log is displayed next to it. Notice that we do not have processed the alerts using the normalization process described in section 7.1.4. The log is preserved in raw, as it was furnished by Ning (see page 171).

More on the right of this scenario, there is another one with 'Sadmind_AO' and

'Admind' alerts. Next to it, there is a scenario containing the traces of the rsh communication ('Rsh' alerts). Both are shown in Figure 8.14.

LLDoS 1.0, the attack contained in DARPA inside1, has another interesting characteristic that can be visualized in Raw-SimSC: the last step of the attack, represented by the RealSecure alert 'Stream_DoS', does not have any IP address in common with any other step in the attack. We can see its isolation in Figure 8.15, where the node representing it, colored in black because it is 'selected' in the interface, is only connected to a 'Port_Scan' alert not related to LLDoS 1.0. The reason for this lies in the source IP address spoofing performed when launching the DDoS attack against the external machine, a process explained in section 3.2.2. Source and destination IP addresses are actually not the only attributes of this step that are different from the ones in the other steps. There is no attribute in common between the 'Stream_DoS' alert and the others, unless a textual similarity in type with 'Mstream_Zombie' alerts because of the word 'stream'. This is a clear example of similarity-based methods (see section 4.3.1, page 70) not being valid for the identification of all kinds of scenario.

8.4 Summary

We have presented our last contribution, SimSC, in this chapter. SimSC is conceived for the visual investigation of scenarios in raw logs. First, this model has been explained in section 8.1. Then, two implementations, HuMa-SimSC and Logan, have been presented in section 8.2.1 and section 8.2.2, respectively. We have ended up the chapter presenting Raw-SimSC, an interface for SimSC that can be applied to any set of raw logs, in section 8.3. Raw-SimSC has been tested with the DARPA 2000 dataset. In the next chapter, we will present the perspectives of this thesis.

Chapter 9

Perspectives

Contents

9.1 On research about multi-step attack detection 218
9.2 On the evolution of AASGs 224
9.3 On attack case detection and identification 226
9.4 On the visual investigation of multi-step attacks 231
9.5 Summary 234

Et c'est toi maintenant, qui vas ouvrir la porte à l'ère nouvelle que j'entrevois...
[And it is thou who wilt now open the door for the new era I have glimpses of...]
— Maurice Maeterlick, *Pelléas and Mélisande*

Several contributions have been presented throughout this thesis: the systematic bibliography about multi-step attack detection; the concept and creation of Abstract Attack Scenario Graphs for expressing alternative multi-step attack cases; two algorithms, Morwilog and Bidimac, to perform detection and identification of attacks using AASGs, and SimSC, a model for visual investigation of attack scenarios. The concepts and models presented here provide new research elements to the not much explored field of modelization and identification of multi-step attacks.

In this chapter, we discuss about the future work that can be done from the advances presented in this thesis. We have divided the chapter in four sections, corresponding to the four different ambits in which we can classify the presented contributions. In section 9.1, we present some perspectives on the conclusions extracted from the multi-step attack detection literature. Concretely, we propose an architecture called OMMA that could contribute to open research in the future. Then, in section 9.2, we discuss the possible evolution of the AASG model, to make it more flexible and with a higher number of available functions. Perspectives on the detection and identification of attack scenarios are the subject of section 9.3, where we propose the

evolution of Morwilog and Bidimac and the creation of new algorithms. Finally, in section 9.4 we discuss about the future of SimSC and its different implementations.

9.1 On research about multi-step attack detection

Apart from being the first time, as far as we now, that the literature about multi-step attack detection on traces is brought and analyzed together, the systematic survey presented in Chapter 4 has allowed us to make a statistic about the selected corpus and extract some conclusions about the field. Some future ways of research can be derived from them:

- To apply the systematic bibliographic methodology used for the survey and presented in section 4.2 to other fields, not necessarily related to Cybersecurity. Being systematic and rigorous is a way of giving bibliographic research a scientific character.
- To develop new public test datasets containing different instances of multi-step attacks and containing several types of logs, not only IDS alerts. We will come back to this topic in section 9.3, as it is a question that has directly affected the evaluation of Morwilog and Bidimac.
- To keep an updated database of multi-step attack detection methods, taking as a basis the effort made in the elaboration of the survey.

Despite of all these fields of action, what worries us the most is the lack of reproducible research in multi-step attacks, pointed out in section 4.4.3, page 96. To reduce the amount of non-reproducible work, we have conceived the OMMA architecture, an open framework for combining event-based investigation and detection methods coming from different researchers. It is a first brick in the pursue of a broadly used open system. Its design has been thought to be compatible with the bricks proposed in this thesis (SimSC, Morwilog, Bidimac) and presented in an article in the EURASIP Journal on Information Security [Navarro 2018b]. The development of each of the modules in OMMA can be a good subject for other Ph.D. thesis or postdoctoral projects.

9.1.1 The OMMA framework

The name OMMA stands for Operator-guided Monitoring of Multi-step Attacks, but it also comes from ancient Greek ὄμμα, which means [Aristophanes 1998] 'the eye of heaven'. OMMA follows the same perspective as HuMa [Navarro 2017] and KILS [Legrand 2014]: the investment of the human analyst in the investigation and detection

of the attacks (see section 4.5). However, there are some key differences between OMMA and the mentioned systems:

- Unlike HuMa, that is solely oriented towards the forensic investigation, OMMA is also focused on real-time detection, taking the approach of classic signature-based detection but giving elements for the integration of learning algorithms.
- OMMA differs from KILS in that it offers an open architecture for the integration of different detection methods. KILS presents a general cognitive model but not a description of the specific modules to be implemented.

The design of OMMA is based on three axis: our experience in security research, part of it acquired during the writing of this thesis; our knowledge about attack detection methods used in the industry, especially of SIEM (Security Information and Event Management) [Kavanagh 2015], and the study of the bibliography about multi-step attack detection presented in Chapter 4. The two goals present in the conception of OMMA are:

- To make it an **open system**, in the sense that its components should be modular and other researchers could develop new methods or adapt existing ones for including them into the system. The open challenge is to offer a framework for improving research collaborations and profit from past work. This would help to mitigate the lack of reproducible research in multi-step attack detection.
- To keep the **human in the loop** from investigation to detection, as proposed by Legrand et al. [Legrand 2014]. We consider that the last verdict about what suppose a threat must be given by the human, whose creative thinking and knowledge about the network she is defending allow her to determine the consequences of events marked as malicious by the system. This information can be fed into OMMA, which makes it compatible with Morwilog, Bidimac or SimSC, whose functioning is based on feedback from the human.

The OMMA architecture is organized in different modules that can be grouped in four domains: Connection, Analysis, History and Knowledge. These domains are compliant to the four levels defined in the KILS model: *real world*, *storage*, *inference* and *expert knowledge*. A global diagram of OMMA is represented in Figure 9.1. In the following subsections we will briefly explain each one of the modules so they can be completely implemented in the future.

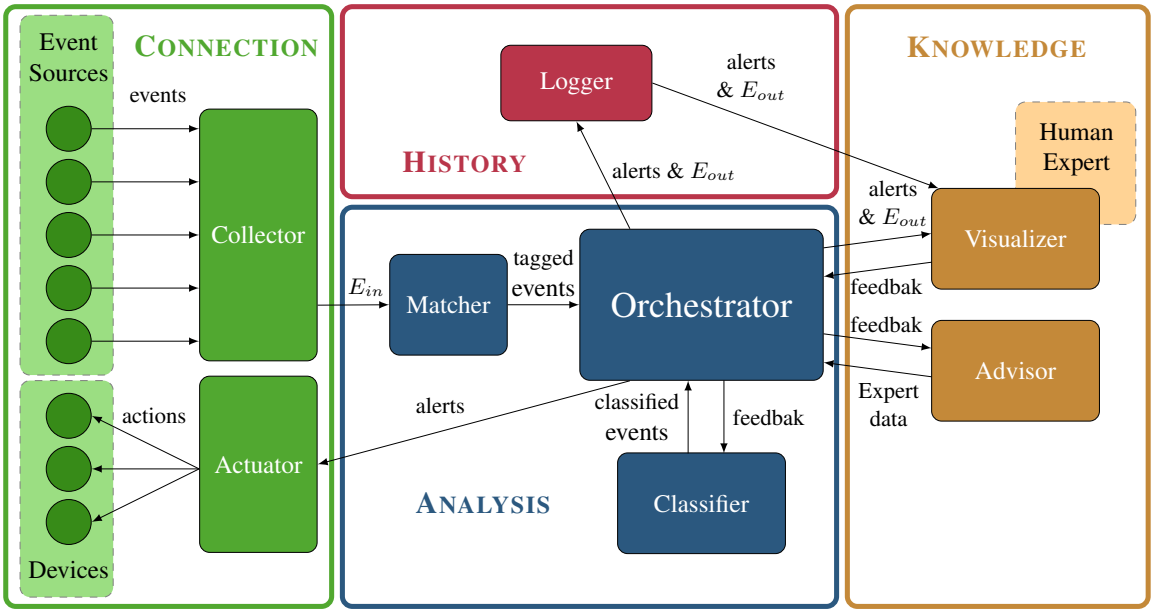


Figure 9.1: Global architecture of the OMMA framework.

9.1.1.1 Connection domain

The Connection domain is the interface between OMMA and the network. It is composed of two modules: the Collector and the Actuator.

Collector (CT) The objective of the Collector is to offer a common platform to integrate log parsers (see page 19) developed by different research teams. The normalized log format defined in section 7.1.4, whose attributes are shown in Table 7.4, page 175, represents a first step in the development of a universal format for event-based security research. We have developed several log parsers to transform the datasets used in this thesis (DARPA 2000, ISCX and HuMa) to that common format, that would integrate a library of parsers in the Collector that would be shared with the community. Another task of the Collector is the application of basic filtering, discarding events that are not intended for analysis. Additionally, the algorithm for event aggregation presented in section 7.2.3, page 187, could be included in this module.

Actuator (AC) The other piece in the Connection domain is the Actuator, that would be responsible for performing actions in the network in response to detected attacks or security flaws. These actions can be very diverse, from the activation of a firewall rule to the disconnection of a server from the Internet. Intrusion response is not easy to automate. Even a security analyst may not have a direct answer about how to fight a threat and has to try different methods before getting to block an

attack. However, it is expected that it will become a hot topic in the near future [Ossenbühl 2015], so a module like the Actuator cannot be missing in a framework that tries to englobe all the aspects of investigation and detection.

9.1.1.2 Analysis domain

Two of the modules in the Analysis domain perform real-time detection of multi-step attacks: the Matcher and the Classifier. The third module, the Orchestrator, is in charge of the coordination of the whole system.

Matcher (MT) This module would apply classical attack detection based on signatures or rules, so well-known threats can be rapidly detected. Detected events would be tagged as malicious, so the rest of the system knows about their nature. It is important to keep them in the analysis, as they can be part of a more complex attack scenario, like it happens with the individual IDS alerts in DARPA 2000 (see section 7.1.1). The resulting tagged set of events is called E_{tag}. Tags can be just appended to the log as an additional attribute. Particular attention should be given to the input and output formats, which have to be well designed to ease the implementation of new methods.

Orchestrator (OR) After the Matcher, events are sent to the Orchestrator, the central module of the system. It coordinates the detection process and makes the final decision about which set of events is a threat for the network. It counts with the tagged events coming to the Matcher, that can be sent to the machine learning algorithms in the Classifier (see below). It also manages the feedback provided by the analyst through the Visualizer. Once a verdict is issued about the nature of the events, attack alerts ($\hat{A}_{E_{out}}$) are sent to the Visualizer, to the Actuator and to the Logger, which also receives the events to store them (E_{out}), possibly enriched during the process.

Classifier (CS) Machine learning classifiers of attacks are intended to be contained in this module. A workflow could be defined inside the Classifier to decide which types of events are processed by which of the implemented algorithms and in which order. The output would be a set of classified events that can be used by the Orchestrator. The development of this module would require a deep knowledge about the different families of machine learning methods and the possible outcomes.

9.1.1.3 History domain

This domain is in charge of storing historical data and it includes just one module, the Logger.

Logger (LG) This module is the responsible of stocking and managing the historical data of the system, both events and alerts. Its implementation should be prepared for the big amount of logs generated by current networks per instant of time. Routines have to be developed to manage the usage of disk space and memory. Stocking events and alerts for a long period allows to do investigation of past attacks, that would be managed by the Visualizer.

9.1.1.4 Knowledge domain

The Knowledge domain has two modules in charge of expert data: the Advisor, that stores information from past experiences, and the Visualizer, that takes feedback from the analyst from a visualization interface.

Advisor (AD) This is the module in charge of managing and storing expert knowledge, which is either manually introduced by security analysts or automatically learned by machine learning algorithms. Within this knowledge we could find, for instance, the AASGs used by Morwilog and Bidimac (Chapter 6) or the list of prerequisites and consequences used by the methods presented in section 4.3.2.1. If methods using automatic learning are integrated in OMMA, data in the Advisor could be updated as new events are processed. This would be managed by the Orchestrator, that also distributes data from the Advisor to the modules needing it, such as the Matcher and the Classifier. A challenge in the development of the Advisor is to create a format for expert data that could be useful for different methods.

Visualizer (VZ) This module is intended to visually represent the information of alerts and events to the human analyst and to get feedback from her. The visualization interface has to be adapted to any new method incorporated into the system. An implementation of the Visualizer should consider the best way to combine different kinds of visualization and to manage the input coming from the analyst. It would also act as a console for system configuration. Moreover, it should contain tools for forensic investigation on the events and alerts contained in the Logger.

9.1.2 Modularity in OMMA

A future implementation of OMMA should be totally modular. Each module has been designed for being independent from the others and replicable to several instances. There are several good reasons to preserve this idea of modularity:

- **Different types of event can be processed separately.** We can have, for instance, several Classifiers, each one with a set of algorithms dedicated to a specific type of event. This can also ease the combination of modules developed by different research units.
- A **hierarchical design** is possible, with an instance of one module acting as master and some others as slaves. This is important if we deploy the system in a large network. We can, for example, have an Orchestrator master controlling some other ones acting as agents, in different areas of the network or remote sites.
- We can do **load balancing** between different instances of the same module. Doing so, we can combine small modules for having a processing power equivalent to a bigger unit, therefore saving cost. For example, we can think of some instances of the Collector module working together for processing a high volume of events.
- A module can be doubled for **High Availability**. If one instance fails, the other one can automatically take its place, avoiding system interruptions. The most suitable element for High Availability is the Logger, as it stores sensitive information which is worthy to be replicated, avoiding data loss in case of failure.
- It can lead to **economies of hardware or design**, as several modules can be grouped into the same device, especially the ones belonging to the same domain. For example, we can have an Orchestrator, a Classifier and a Matcher in the same box.

In Figure 9.2, an example of modular implementation of the OMMA architecture in a final network is represented. There is a Collector for each of the subnetwork A and B, reporting to the same Matcher, and only one Actuator for both subnetworks. In the case of subnetwork C, it has its own Actuator and Matcher, but also two active Collectors configured to do load balancing. The Orchestrator, the Classifier and the Advisor share the same hardware, which reduces delays in the transmission of information. However, the Visualizer is implemented in a different asset. Finally, there are two Loggers in High Availability configuration (HA) into which all the information is duplicated. If one stop working, the other can continue its task.

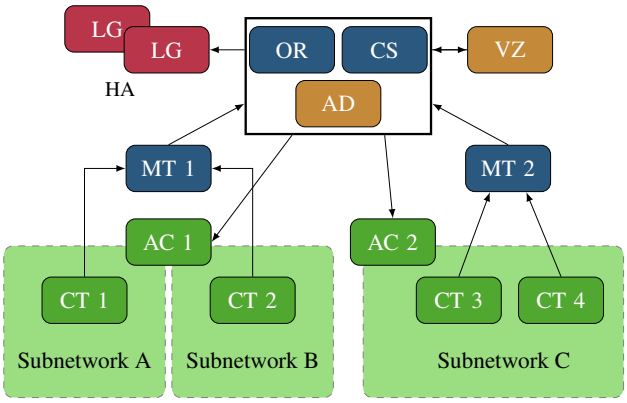


Figure 9.2: Example of an implementation of OMMA where we can see its modularity

9.2 On the evolution of AASGs

The AASG model (Chapter 5) is probably the first one conceived for representing a set of alternative multi-step attack hypotheses for their detection and identification. There is substantial room for new contributions in both the composition of the AASG itself and its creation from a detected instance of a multi-step attack. Perspectives on this two aspects are presented below, in sections 9.2.1 and 9.2.2, respectively.

9.2.1 The structure of the AASG

The building blocks of an AASG are the abstract events contained in its nodes, which determine the events that could match each node. Matching is determined by a set of conditions composing the abstract event. We have defined 10 different conditions, that are listed in the Appendix E. Half of them are *absolute*, thus referring to a certain external value or set of values, and the other half are *relative*, thus pointing towards the previous event in the sequence.

The conditions defined so far give the analyst much flexibility in the creation of the AASG, as we saw in the AASGs presented throughout this thesis, listed in Appendix F. However, an analyst could still imagine comparisons that would be hard to implement using the conditions proposed so far. The first evident way to improve the definition of the abstract events is the development of new conditions. For example, the hierarchy-based similarity feature defined by Julisch (see page 50) could be implemented for different types of inherent attributes: IP addresses, port numbers, etc.

Another way of improving conditions would be by the extension of the existent ones. For instance, textual similarity (conditions TXT and SIM_TXT) could be implemented by means of another function different from the Jaccard index, such as

lexical matching [Metzler 2007] or the Levenshtein distance [Levenshtein 1966]. On its side, the set-based condition (SET) could also incorporate alternative definitions of sets R for different types of attribute, without the need of listing all the elements in the set. These definitions could be based, for example, on a numerical interval (e.g. $[1024, 5000]$) or, in the case that the attribute has IP addresses as values, a subnetwork defined by a mask (e.g. 175.68.22.0/24).

An additional improvement could come from the objects that are compared in the relative conditions. So far, we have only defined relative comparisons between attributes in different events, but we could think of comparing attributes within the event itself. For example, it could be interesting to define a condition based on the similarity between the source and destination IP addresses. A prefix comparison (condition SIM_PFX) between the two values could tell us when the source and destination are too far in terms of network distance.

Regarding the combined used of the conditions, all the conditions contained in the abstract event have to be fulfilled to considered the event as matched with the abstract event at the present state of the model. This is equivalent to an AND ($\wedge$) logical relationship between the conditions. To broaden the possibilities given to the analyst for the definition of the hypotheses, some other logical relationships between conditions could be proposed, such as OR ($\vee$). This logical combination of conditions could be inspired in the ones used by modeling languages such as LAMBDA [Cuppens 2000] (see page 37).

9.2.2 The creation of the AASG

An example of how the analyst can create an AASG from a set of logs representing a multi-step attack was given in section 5.3. However, we have not addressed in detail the steps for creating a general AASG. A formal methodology for the creation of AASGs is fundamental if we want to continue working with the model.

This methodology could be first developed from the study of several examples of multi-step attacks. The process of hypothesis derivation can then be reproduced taking the point of view of the security analyst. Once this basic methodology is developed, it could be improved by the development of AASGs in a real environment. A set of working security analyst, from a public institution or a private company, would be trained in what an AASG is and how it can be built. They would then choose some multi-step attacks to develop their own AASGs, writing down all the steps done during the deduction process. This notes, together with the resulting AASGs, would be studied by the researcher to improve the methodology firstly developed. On top

of that, the researcher would have a new set of AASGs to perform experiments. The evaluation of the difficulties found in the creation of the AASGs could also lead to an improvement in the definition of an AASG and in the conditions contained in the abstract events.

The AASG model has been conceived to be manually built from the analyst's working hypotheses. But we could also imagine the automatic derivation of AASGs by means of an automatic learning algorithm. Automatic methods could derive a set of alternative cases from a dataset that would be part of the AASG, which would be later used to detect and identify the correct hypotheses in real events.

As we saw in Chapter 3, there does not exist a standard way of modeling a multi-step attack. Having a library of AASGs shared by the community would allow having together the working hypotheses about multi-step attacks. Inspiration can be taking from existent alert sharing platforms [Husák 2018]. Specific models, in the form of CASGs, could be built once the hypotheses are confirmed, by deduction from the AASGs and the associated sample events detected by them. We could even extract different hypotheses about the same attack in several alternative CASGs. Improvements done on the structure of the AASGs would not diminish the power of a library of AASG if changes are thought to be compliant with the existent structure.

Comparing existent AASGs would allow finding common patterns between the alternative cases of different multi-step attacks. The fusion of AASGs could result on more general AASGs that would have never been separately developed. AASGs in the common library could be also studied using graph theory and characterized by some of the metrics used in this field, such as the dimension. This could maybe lead to a classification of the multi-step attacks according to the arrangement of the deduced hypotheses.

9.3 On attack case detection and identification

The goal of the algorithms presented in Chapter 6, Morwilog and Bidimac, is the detection and identification of the alternative cases represented in an AASG. Even if they work well, as we saw in Chapter 7, they still have limitations and there is much room for improvement. Possible future ways of improving attack case detection and identification using AASGs are presented below. First of all, we address in section 9.3.1 the limitations of datasets and how further evaluation of the presented algorithms could be done if they are overcome. In section 9.3.2, we propose the exploration of case confirmation and the incorporation of step prediction into the models. To implement

these two functions, there is no need of modifying the current AASG, but only a different view on pheromones and probability parameters. We continue by proposing the implementation of real-time functioning in section 9.3.3 and automatic functioning in section 9.3.4, as possible improvements of Morwilog and Bidimac. Finally, we open the door to the development of new algorithms in section 9.3.5.

9.3.1 Further evaluation

It was mentioned in section 4.4.2 that existent public datasets are composed by raw packets, so they need to be processed by an IDS in order to get a set of events, the alerts, with the IDS as the single source. Moreover, none of the most used public datasets contains several instances of a multi-step attack, but just one. Remember that for the experiments with DARPA 2000 we had to consider the infections of the Mill, Locke and Pascal machines as different multi-step attacks, even if they constitute three parts of the same attack, performed in parallel (see section 7.2.1). A complete evaluation of Morwilog and Bidimac requires similar instances of a multi-step attack, so we had to artificially create the Eventgen datasets, presented in section 7.1.5.

Of course, many private datasets such as HuMa (section 7.1.3) exist, but their content cannot be disclosed, as we commented at the end of section 4.4.3. Some researchers use data collected from their own university [Chen 2014a] or company [Julisch 2001], or from partners [Sudit 2005, Skopik 2014, Zhang 2015]. There is even a shared private dataset, the one generated by Skaion Corporation [Shaneck 2006, Mathew 2009], but access to it is limited to official U.S. government's research.

An important future line of research is the generation of new public event-based datasets, either from real events or from a test network. To be useful to our research, they should a) contain several instances of a multi-step attack and b) have labels in the events indicating where the attacks are[1]. The availability of these datasets would allow the execution of new experiments to evaluate Morwilog, Bidimac and any other new method working with AASGs.

For example, with new correctly built datasets we could better study the effect of different types of AASG in the execution of the algorithms, checking the influence of their size or depth. A catalog of AASGs (see section 9.2.2) would be developed in parallel to allow this evaluation.

[1] http://www.mlsecproject.org/blog/the-importanceof-good-labels-in-security-datasets

9.3.2 Case confirmation and step prediction

Both in Morwilog and Bidimac, we have left at the sole discretion of the analyst the moment when one of the cases in the AASG is considered as confirmed. Future work could address different ways to determine when to stop in the identification process based on the values of the levels of pheromones and the probability parameters. The most intuitive conditions to consider a case as confirmed would be based on the choosing probability $P_{\boldsymbol{b}_\kappa}$ of the branches, defined in Equation 7.1, page 194. For example, if we set a certain threshold λ_p, some possible conditions to stop could be:

- The highest $P_{\boldsymbol{b}_\kappa}$ is above λ_p.
- The different between the highest $P_{\boldsymbol{b}_\kappa}$ and the second highest one is above λ_p.
- The different between the highest $P_{\boldsymbol{b}_\kappa}$ and the addition of the rest is above λ_p.

Conditions to identify when a branch in a DAG is clearly dominant with respect to the other ones are also found in the literature and can be adapted to the algorithms working with AASGs. For instance, Jiang et al. [Jiang 2017] propose the concept of importance of a branch as an indicator for pruning a tree. Defining an entropy of an AASG in terms of Information Theory could also serve to decide at which moment the AASG stops giving enough new information.

Apart from the confirmation of cases, another way to explore is the use of the AASG as a model for prediction. When the proposed algorithms are executed, information about the history of incoming sequences is coded in the level of pheromones or in the probability parameters. This information could be used to trigger early warning alarms to inform the analyst about which are the most probable events continuing the sequence. This would be a complement to the current mode of functioning of Morwilog and Bidimac, in which alarms are only triggered once the sequence has been totally discovered. We could imagine a predictive models in which alarms are sent right before a very probable path is detected, even if the execution of the full detection process continues in parallel.

9.3.3 Real-time functioning

Even if both Morwilog and Bidimac have been conceived for working in real time, they have been neither implemented nor tested yet in a real environment. However, special attention has been taken to simulate this real behavior:

- Datasets are loaded event per event, in order, simulating the incoming events in a real environment.
- Even if each search process (the journey of a *morwi* in the case of Morwilog)

happens sequentially, alarms are not raised until the last event in the found sequence appears in the input.

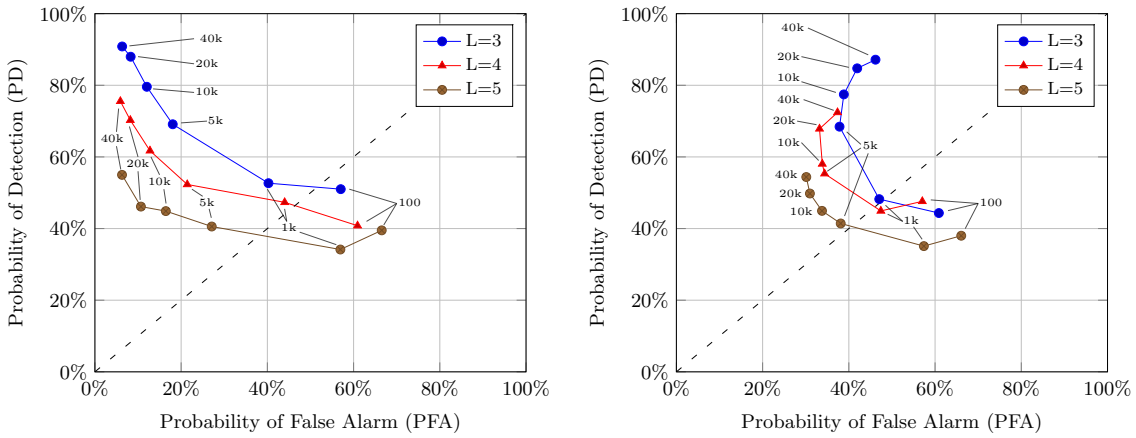
- The stigmergic or Bayesian AASGs do not evolve until the simulated analyst evaluates the alarms.

Despite of this measures, some assumptions not present in a real environment has been also taken into consideration. The development of new implementations overcoming these assumptions constitutes a good base for future work:

- **Events are stocked in a file and load into the system during initialization**. Without modifying the algorithms, the two models could be adapted to work with real incoming data by deploying a previous layer in charge of the collection of events. To do that, some available software already exists, such as Logstash [Turnbull 2013].
- **The simulated analyst evaluates the alarms immediately after they are triggered**. It is clear that a human security analyst could not be able to do so. To get a more realistic model, a system of alarm queues could be developed and the simulated analyst could incorporate a random delay. The only effect this would have in the algorithms is that the evolution of the AASGs would be performed later. The impact of this delay could be evaluated in real data. Inspiration to create the queues could be taken, for instance, from the work by Zali et al. [Zali 2013], which could also help in the implementation of a predictive version of the models (see section 9.3.2).
- As our security analyst is simulated, **there is no monitor to visualize and acknowledge the alarms**. The development of such a monitor would allow the researcher to test the models in a real environment with a human analyst.
- **The parallel execution is simulated**. The search processes are not executed in different computer threads. However, both Morwilog and Bidimac have been conceived to be parallelized, being each search process autonomous from the others. A parallel implementation would be an excellent goal for future work, certainly increasing the speed of the execution and making the systems able to deal with big amounts of real-time data.

9.3.4 Automatic functioning

Morwilog and Bidimac have been conceived to work with AASGs created by the analyst from her working hypotheses, but we already mentioned in section 9.2.2 the possibility of having these AASGs automatically created from event data using a learning algo-



(a) With pheromone update

(b) Without pheromone update

Figure 9.3: ROC curves (PFA vs. PD) representing the results from automatic Morwilog. Each point is the average of the result of 50 simulations with the parameters in Table 6.1 and $P_{tree} = 0.5$, $P_{jump} = 0.1$. The size of the dataset is indicated next to the points. L is the length of injected sequences.

rithm. This process could be directly integrated in the execution of Morwilog, Bidimac or any other algorithm doing detection and identification with AASGs.

In fact, we already explored this possibility for Morwilog when the algorithm was first presented in public, at the SSCI conference in December 2016 [Navarro 2016]. Back then, we had not developed all the theory around AASGs. The structures used were then called *event trees*. They were created randomly from incoming data, as we wanted to test how the pheromones made the system evolve and not how the trees could be built by a human analyst.

Random creation of event trees was controlled by two parameters: P_{tree} and P_{jump}. For each event $\boldsymbol{e}_i$ creating a *morwi*, if $\boldsymbol{e}_i$ did not match the root node κ_0 of any event tree among the existent ones, a new event tree with a single branch was created from a random sequence of events starting in $\boldsymbol{e}_i$. Moreover, every time a *morwi* had to choose among events matching the children of a certain node, there existed a probability P_{jump} of choosing a random event instead of one in the selected list. If this was the case, a node containing an abstract event representing this random event was appended to the tree with a minimum level of pheromones.

This automatic mechanism was not practical since it returned a lot of alarms to the analyst, but even without having a previous library of event trees we could see the positive effect of the evolution of pheromones as the size of the dataset increased. These results, obtained from simulations on a set of Eventgen datasets (see section 7.1.5), are shown in the two ROC plots in Figure 9.3. They were previously published

in the article about OMMA [Navarro 2018b].

New mechanisms for automatizing the functioning of Morwilog and Bidimac, not necessarily related to the one just mentioned, could be incorporated into the systems in future work.

9.3.5 New algorithms

The last future line of research about attack case detection and identification is the development of new algorithms working with AASGs. The three premises on which such an algorithm should be built are listed in the introduction of Chapter 6, in page 135: capability of performing identification, generation of alarms when a sequence matching a branch is detected and presence of the human in the loop. These premises establish the minimum requisites an algorithm should fulfill to be comparable to Morwilog and Bidimac.

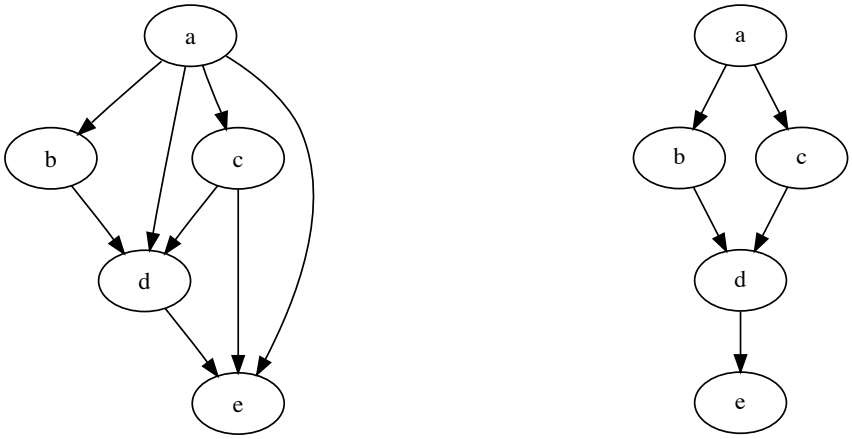
To create new algorithms, the first step could be to modify the two existing ones. For example, remember that some conditions were defined for a *morwi* to interrupt its journey if a similar *morwi* was recently created (see page 148). This avoids the creation of too many *morwis* when there are a lot of very similar events in a short period of time, as it happens in a scan. This feature was not implemented in Bidimac because we wanted it to be a pure Bayesian method, with probability parameters always in accord to the statistics of the matched attack sequences. A new pseudo-Bayesian model could be derived from Bidimac introducing this feature or other similar ones.

9.4 On the visual investigation of multi-step attacks

SimSC, that constitutes the last contribution of this thesis and was presented in Chapter 8, was conceived when exploring how the analyst could perform investigation on a set of raw logs. Even if some implementations of SimSC, such as HuMa-SimSC and Logan, have been created, there is still a lot of aspects to explore and improve that could constitute the subject of future research.

For instance, we did not define how found attack scenarios could give rise to the cases that would conform an AASG. SimSC could be implemented together with an AASG editor, so the analyst could build the hypotheses by directly selecting the suspicious nodes and arranging them in branches.

Moreover, the inherent attributes with which SimSC works so far are just two, the IP addresses and the timestamp. Further work could be done to automatically extract other attributes that also have a fixed and particular format, such as MAC



(a) Original graph (b) After transitive reduction

Figure 9.4: Example of transitive reduction of a graph. Source: Wikimedia Commons

addresses or URLs, and use them to make similarity comparisons. Apart of basing the extraction on the format of the attribute, as we did so far, it could be based on the context of the information within the log. For example, we know that in many logs a port number usually follows the IP addresses, separated by them by a colon (':'). Additional information for attribute extraction could come from the position of strings within the log. For instance, the source IP address is usually placed before the destination IP address.

Regarding the shape of the scenarios, the order of events is difficult to visualize in the current version of SimSC, even if it allows the easy identification of port scans as conglomerations of events, as we could see in Figure 8.5. To make the order of events explicit, direction could be added to the edges joining each couple of nodes in the visualization screen. This direction would point from the node representing the earlier event in the couple to the one representing the later one.

On top of that, calculating and representing the *transitive reduction* of the resulting graphs could help the visualization of the sequences. The transitive reduction [Aho 1972] of a directed graph D is the graph resulting from the preservation of only the longest path among all the existent paths joining two nodes. An example of transitive reduction is shown in Figure 9.4. The arc ae, for example, is erased, and only the paths a-b-d-e and a-c-d-e joining a and e are preserved. As we can see in this example, the number of arcs is reduced and the order followed by the nodes is more easily visualized. This could probably lead to a better deduction of the causality of events within the scenarios in SimSC.

Finally, an interesting line of research would be to adapt SimSC to work with

parsed logs, in which the inherent attributes are distinguishable. Doing so, all the similarity and time-based features presented in sections 3.4.1 and 3.4.2, respectively, could be applied to build connections between the logs. Features could be used either individually or combined by means of correlation functions, such as the ones presented in section 3.4.4. The resulting model could be called Parsed-SimSC.

Parsed-SimSC would substantially differ from the similarity-based multi-step attack detection methods reviewed in section 4.3.1. While those ones are oriented towards detection and use just a fixed set of similarity and time-based features, Parsed-SimSC would constitute an investigation tool in which the analyst could analyze a set of logs under different perspectives. The analyst would be able to dynamically choose the features, their combinations and their parameters to better explore the selected set and the relationships within the logs. This could result in a) a better understanding on what happened at that time in the past, b) the extraction of AASGs containing new hypotheses and c) the development of similarity-based multi-step attack models

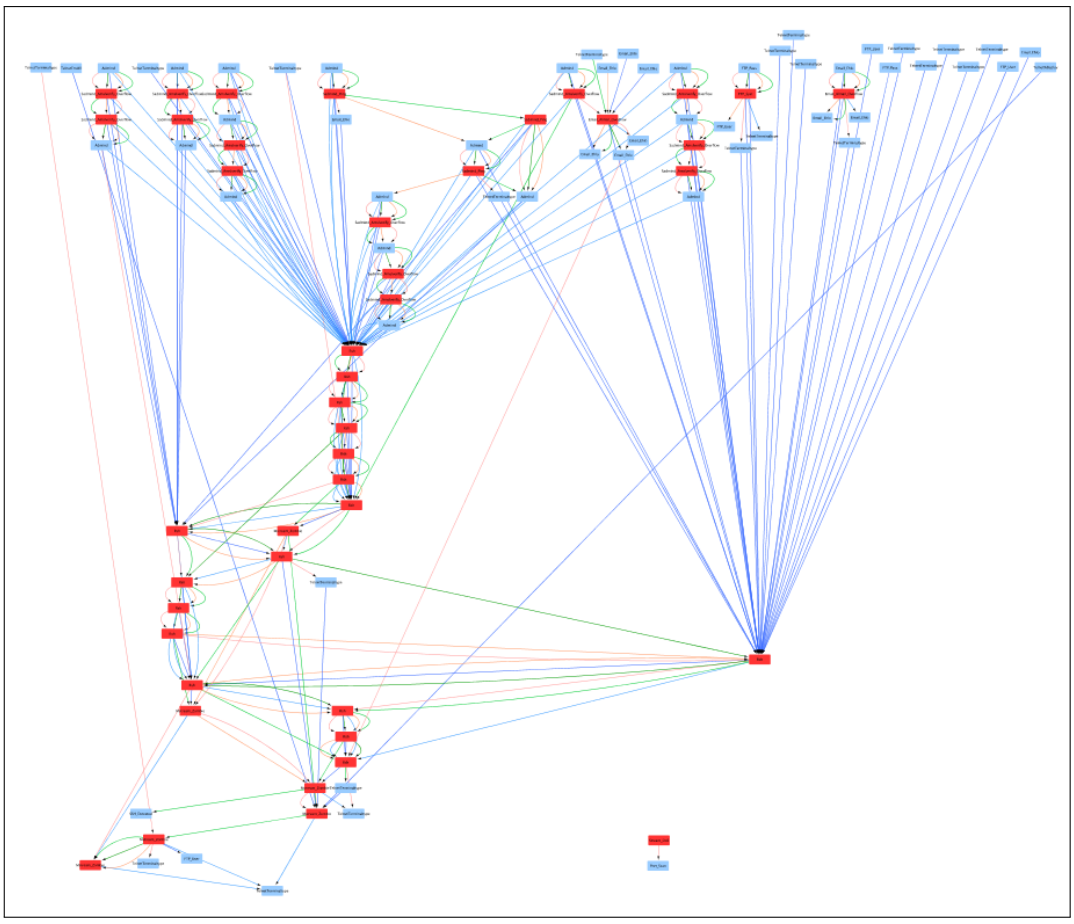


Figure 9.5: Similarity-based graph of alerts related to LLDoS 1.0. Representation made in Cytoscape (https://cytoscape.org/).

to improve automatic detection methods.

We have explored different similarity features on several attributes to compose a set of graphs with the alerts related to the attack LLDoS 1.0 in DARPA 2000. The obtained results have been represented in Cytoscape. One of the resulting graphs is shown in Figure 9.5. This work is still in a preliminary stage, but we are convinced that a further exploration of similarity and time-based features in a selected corpus of multi-step attacks would give us a better insight on how steps are related. Developing a tool for the exploration of these features would help researchers on the development of new methods. Anyways, researchers should be aware that a pure similarity-based method would never be able to detect all types of multi-step attacks, as the attacks in DARPA 2000 clearly indicate. This limitation was addressed in page 72.

9.5 Summary

A Ph.D. thesis would not be complete without the exposition of the perspectives opened by the performed research. In this chapter, new lines of research have been proposed based on each of the four contributions of this thesis:

- In section 9.1, we have proposed OMMA, an open framework that tries to alleviate the lack of reproducibility in the multi-step attack detection methods reviewed in Chapter 4. OMMA has been presented in section 9.1.1 and its modularity is explained in section 9.1.2.
- In section 9.2, we have presented possible future work for the evolution of the AASG model (Chapter 5) in two ambits: the improvement of it structure (section 9.2.1) and the development of methods for the creation of AASGs (section 9.2.2).
- In section 9.3, several lines of research for the development of models for multi-step attack detection and identification using AASGs, such as Morwilog and Bidimac (Chapter 6), have been introduced. We have proposed the further evaluation of Morwilog and Bidimac with new test datasets (section 9.3.1), and their improvement by the incorporation of case confirmation and prediction mechanisms (9.3.2) or an automatic functioning mode (9.3.4). Finally, we have proposed the development of new algorithms in section 9.3.5.
- In section 9.4, some improvements of SimSC (Chapter 8) has been proposed, such as the automatic extraction of additional attributes, the transformation of the graphs to their transitive reduction or a version of the model for parsed logs.

Chapter 10

Conclusion

"And I walked for a long time, with my boots covered in mud, looking back every minute to see if someone was following me."
— Benno von Archimboldi, *Lüdicke*

The present Ph.D. thesis is written in a context in which multi-step attacks, such as WannaCry, continue to wreak havoc on organizations from all around the world. Once an attack with these characteristics is performed, it is a challenge for a security analyst within the attacked organization to identify the actions that took part on it and, more importantly, to determine how the attack could be repeated in the future. A correct identification of the involved steps is essential for the development of solid detection models.

Our research question, presented in Chapter 1, page 3, is:

How can we help the security analyst to decide between alternative scenarios of multi-step attacks in order to ease the detection of future occurrences? Can we perform detection at the same time as the learning process evolves?

It addresses the development of a model for helping the analyst in the modelization of alternative multi-step attack cases, the detection of these cases on incoming events and the identification of the correct ones.

Before giving answer to this question, Chapter 2 is devoted to the definition of the concepts used throughout this thesis. This step is fundamental, as we find that not even englobing concepts such as Cybersecurity have a standard and widespread definition. We see that the 'traces' constitute a set in which the actions of the attacks are represented. They act as a source of information for the security analyst.

The object of our study are the multi-step attacks, that are addressed in Chapter 3. Multi-step attacks can be modeled as directed graphs, called CASGs, with the arcs of the model representing a certain relationship between two traces, that constitute the nodes. This relationship is characterized in the literature by one or more features. We distinguish between three types of feature: similarity, time-based and context-based. They are generally expressed by functions, whose results can be combined to numerically define the arcs in the CASG.

Once the multi-step attacks are well defined, we study in Chapter 4 the methods developed for detecting them. The resulting corpus of publications addressing multi-step attack detection methods and the conclusions extracted from them constitute the first contribution of this thesis. At the same time, the selected corpus, built following a systematic review methodology, represents a solid bibliography for this thesis. We find five kinds of approach in multi-step attack detection: similarity-based, causal correlation, structural-based, case-based and mixed. The most important conclusion we can extract from this chapter is that no method exists to integrate the multi-step attack cases proposed by the human analyst in the detection process, in pursuit of their identification on the incoming traces. Another conclusion, that is secondary for answering our research question but very relevant for the field, is that most of the publications in the selected corpus contain experiments that are not reproducible, which is probably a collateral effect of the lack of publicly shareable datasets containing multi-step attacks.

The model we propose to represent the alternative cases or hypotheses of multi-step attacks as they are conceived by the analyst constitutes the second of our contributions of this thesis. We focus on the events as the traces in which attacks are expressed. The model is called *Abstract Attack Scenario Graph (AASG)* and it is presented in Chapter 5. An AASG is a directed acyclic single-source graph where the alternative cases are arranged in different branches, with each node represented by an object, called *abstract event*, that refers to a selection of events through a set of conditions. The formal definition of the model is complemented by a functional JSON-based language and by a graph editor to easily express the AASGs.

Specific algorithms have to be developed to work with the AASG model, which is prepared both for detection and identification of correct multi-step attack cases. Our third contribution is composed of two models to perform these tasks, *Morwilog* and *Bidimac*, which are presented in Chapter 6. Both models rely on the feedback provided by the security analyst once a case is detected and the malicious nature of the corresponding sequence of events is evaluated. Morwilog is based on the behavior of foraging ants when they look for food sources. An artificial ant, called *morwi* in the model, is created each time an event arrives to the system. It then looks for a branch in an AASG matched by this event and subsequent ones. Bidimac transforms the AASGs in a Bayesian model reflecting the statistics of the correct multi-step attack cases.

Evaluation of the AASG model, Morwilog and Bidimac is presented in Chapter 7. Regarding the first model, we see that the implemented mechanisms are generic

enough to define AASGs for the three studied datasets: DARPA 2000, ISCX and HuMa. In the two last datasets, we can even define one general AASG capturing the general sequence of actions of several types of attack. All the defined AASGs allows successful attack detection in the respective datasets. Regarding Morwilog and Bidimac, the first one returns fewer alarms to the analyst than the second one, as it chooses only one path in the AASG among the possible ones. Because of this, it is also more adaptable to big amounts of incoming events. In terms of precision and recall, both models behave similarly. The ability of detecting multi-step attacks relies on how the AASG is defined.

Our fourth and last contribution is a model for visual investigation of attack scenarios in sets of raw logs. It is called *SimSC* and it is presented in Chapter 8. SimSC is based on the automatic extraction of IP addresses and timestamps from logs that have not been normalized, thus expressed as plain text. The logs are represented as nodes in a graph whose edges are defined by similarity of the extracted attributes. Our goal conceiving SimSC is to explore the process of investigation performed by the analyst after the occurrence of an attack. The evaluation of the model is done through its implementation on the HuMa architecture and on a web platform working with medical data. A standalone version is also implemented and tested with the events in the DARPA 2000 dataset. Although the current visualization of SimSC is simple and the analyst is responsible for the interpretation of what is represented, the idea we want to highlight is the possibility of identifying scenarios on logs that have not been processed. This is important in a computer environment allowing the connection of non-standard devices with unknown log formats.

The lack of public datasets with several instances of multi-step attacks hinders the evaluation of the presented contributions. For evaluating Morwilog and Bidimac beyond the performance of specific AASGs, we had to create our own artificial set of datasets. Solving this problem with datasets constitutes the base of one of the future lines of research presented in Chapter 9. This Ph.D. thesis opens an ample range of perspectives linked to any of the four contributions. The systematic survey of multi-step attack detection methods reveals the lack of reproducibility of experiments in the field. This brings us to the proposal of OMMA, an open framework for multi-step attack investigation and detection that we expect will be further developed in future research. AASGs can be improved in many ways, by the development of new matching conditions or the establishment of a formal methodology to create them. The limitations of Morwilog and Bidimac with respect to a functional implementation are numerous. We expect that future research will continue the development of these

models to better adapt them to real situations so they can be tested by security analysts in real time. Finally, SimSC still represents a preliminary contact to the vast and unexplored world of attack scenario investigation in raw logs. By conceiving it as visually-based, we want to emphasize the importance of the human analyst in the investigation and detection of multi-step attacks.

To these two activities, investigation and detection, we append in this thesis the process of *identification*, the confirmation of correct hypotheses about a found attack. There is no much research work addressing this phase of Cybersecurity that lies between investigation and detection, even if it is obvious that this process takes place every time a security analyst develops a detection model. Machines are far from substituting human beings in this process. Instead of trying to unsuccessfully develop automatic detection methods tested on scarce and not very representative datasets, we propose to concentrate the research effort on models to help the human analyst, assuming the essential role of her creative thinking. We modestly expect that the proposed contributions will lead the way to further progress on this, especially with respect to multi-step attacks, a threat that seems to have come to stay.

Appendix A

Summary of definitions

In this appendix we collect the definitions of basic terms as they are used in this thesis. Most of the terms were defined in Chapter 2.

AASG (Abstract Attack Scenario Graph): A DAG for representing alternative hypotheses of multi-step attacks. Each step is represented as a node in the graph, containing an abstract event. More information in Chapter 5.

Accountability: Referring to a system, the property of guaranteeing that actions performed by each user can be traced uniquely to that user.

Advanced Persistent Threat (APT): Also called *complex attack* or *targeted attack*, it is a type of multi-step attack that is specifically crafted against a single victim and where the access of the attacker to the target network is maintained during a long period of time.

Alert: An event generated by a security system in response to the detection of alleged malicious activities or faults [Salah 2013].

Analyst: As it is used in this thesis, a cybersecurity expert in charge of the analysis and evaluation of potential threats, of the development of detection signatures and of the actions to be taken in the organization when an attack is detected.

Asset: A piece of data, a service, a system capability (processing power or bandwidth, for example) or an item of system equipment (software or hardware) being part of or connected to a computer network [Stallings 2015].

Attack: In the context of this thesis, a cyberattack.

Attack graph: An abstract representation of the network with each node representing an asset with a set of associated vulnerabilities.

Attack scenario: In the context of this thesis, a multi-step attack.

Attacker: The agent or set of agents performing a cyberattack, i.e. the author or the set of authors of the actions involved.

Availability: Referring to a system, the property of guaranteeing access only to authorized users at any moment in a timely manner.

CASG (Concrete Attack Scenario Graph): A DAG that represents a multi-step attack, with the nodes corresponding to the traces describing each step of the attack. See section 3.3.3.

Confidentiality: Referring to a system, the property of ensuring that information contained in that system is disclosed only to authorized users.

Cyberattack: A set of actions, via cyberspace, targeting an organization's use of computer networks for the purpose of disrupting, disabling, destroying, exposing or controlling without authorization a computing environment/infrastructure; or destroying, modifying or stealing data or blocking the access to it [CNSS 2015, ISO/IEC 2016, FFIEC 2018].

Cybersecurity: The organization and collection of resources, processes, and structures used to protect cyberspace, cyberspace-enabled systems and the information contained on them from intentional cyberspace-based actions that compromise their confidentiality, integrity and availability.

Cyberspace: The notional environment in which communication over computer networks occurs [OED 2018b].

Defender: Entity responsible of preserving the target system against potential attackers and, if these last ones are successful in their attempts, of detecting the attack and mitigate its effects.

Device: An term equivalent to 'asset'.

Distributed attack: See 'multi-agent attack'.

Event: A trace corresponding to an identifiable action that happens on an asset. It is preserved as a line of text called log.

Evidence set: The ensemble of traces collected by any asset of the network containing information about any of the steps composing a multi-step attack. It can be *homogeneous*, if all the traces contained in it have the same format, or *heterogeneous*, if there are at least two different formats.

Extrinsic attribute: An attribute of a trace that is determined by the place of the trace among a set of traces. E.g. the probability of having a trace of the same type generated in a period of 5 seconds.

Hyper-trace: An abstract entity containing an aggregated set of traces.

Inherent attribute: An attribute of a trace that is fully determined by the information contained in the trace. E.g. the name of the machine generating the trace. See Appendix C for a full list of the inherent attributes mentioned in this thesis.

Integrity: Referring to a system, the property of ensuring that assets or messages exchanged between users of that system can only be modified in a specified and authorized way.

Intrusion Detection System (IDS): A piece of software or hardware conceived to automate the attack detection process.

Known attack: A cyberattack that has been widely detected, analyzed and understood by the security industry, so prevention mechanisms against it can be effectively deployed and a signature characterizing it could be developed for its detection.

Known vulnerability: A vulnerability which the defender is aware of and whose exploitation can be prevented by the development of specific security mechanisms.

Laundering host: See 'stepping stone'.

Log: A representation of an event in text.

Multi-agent attack: An attack concurrently executed by more than one attacker. They are also called 'distributed' or 'coordinated' attacks.

Multi-step attack: A cyberattack composed of at least two distinct actions or steps. Also called by many alternative names: *multi-stage attacks*, *multistage attacks*, *attack strategies*, *attack plans*, *attack scenarios* or *attack sessions*.

Packet: The minimal unit of data exchanged in a network communication protocol. It can be preserved as a trace.

Parsed log: A log that has been preprocessed in order to make its attributes distinguishable, transformed into a format suitable for security analysis.

Path: In a CASG or an AASG, a finite alternating sequence of nodes and arcs with all of them distinct.

Raw log: A log in its original format as it is generated by the asset, in plain text.

Single-step attack: A cyberattack where just one action, possibly repeated, is required to threaten the system.

Step: In a computer network, each one of the actions being part of the same multi-step attack.

Stepping stone: In a multi-step attack, an asset attacked and controlled by the attacker as a intermediary step in the consecution of her goal. It is also called a *laundering host*.

Structural information: In a computer network, the information that can obtained supposing that no traffic is exchanged between any of the assets. Some examples of structural information are the characteristics of each asset, the vulnerabilities or the network configuration.

Trace: A preserved piece of information about actions performed within a computer network.

Unknown attack: A cyberattack that relies on the exploitation of unknown vulnerabilities or on ways of access that were not considered as such by the defender.

Unknown vulnerability: A vulnerability not found yet by the defender but whose existence is assumed when deploying the defenses of the organization.

Vulnerability: A flaw or weakness in the design or implementation of a piece of software or hardware that can be exploited by an attacker to perform an unauthorized action [Stallings 2015, Duffany 2018].

Appendix B

List of acronyms

- **AASG**. Abstract Attack Scenario Graph
- **ACO**. Ant Colony Optimization
- **CASG**. Concrete Attack Scenario Graph
- **CIA**. Confidentiality Integrity Availability
- **CVE**. Common Vulnerabilities and Exposures
- **CVSS**. Common Vulnerability Scoring System
- **DAG**. Directed Acyclic Graph
- **DARPA**. (United States) Defense Advanced Research Projects Agency
- **DDoS**. Distributed Denial of Service
- **DNS**. Domain Name System
- **DoS**. Denial of Service
- **ENISA**. European Union Agency For Network And Information Security
- **FTP**. File Transfer Protocol
- **HMM**. Hidden Markov Model
- **HTTP**. Hypertext Transfer Protocol
- **ICMP**. Internet Control Message Protocol
- **IDS**. Intrusion Detection System
- **IP**. Internet Protocol
- **ISO**. International Organization for Standardization
- **ITU-T**. International Telecommunication Union (Telecommunication Standardization Sector)
- **MAC**. Media Access Control
- **NATO**. North Atlantic Treaty Organization
- **NTP**. Network Time Protocol
- **OS**. Operating System
- **OSI**. Open Systems Interconnection
- **PDF**. Portable Document File

- **RPC**. Remote Procedure Call
- **rsh**: remote shell command
- **SIEM**: Security Information and Event Management
- **SMB**: Server Message Block
- **SMTP**. Simple Mail Transfer Protocol
- **SOM**. Self-Organizing Map
- **TCP**. Transmission Control Protocol
- **TDA**. Topological Data Analysis
- **URL**. Uniform Resource Locator
- **US CNSS**. United States Committee on National Security Systems
- **US NIST**. United State National Institute of Standards and Technology

Appendix C

List of inherent attributes

The reader can find below the list of inherent attributes from traces used throughout this thesis. The names have been chosen to be intuitively understood. A brief description is shown next to them. We have chosen not to list here the attributes proposed by Pei et al. [Pei 2016] and listed in Table 3.1, as they are not used in any other part of this thesis.

- *action* - Type of action associated to the trace.
- *dom* - Internet domain name.
- *id* - Identifier of the trace.
- *inobj* - Input object.
- *ipdst* - Destination IP address.
- *ipsrc* - Source IP address.
- *log* - The original raw trace.
- *macdst* - Destination MAC address.
- *macsrc* - Source MAC address.
- *origin* - Device generating the trace.
- *outobj* - Output object.
- *pdst* - Destination port number.
- *process_id* - Identifier of the OS process related to the trace.
- *psrc* - Source port number.
- *service* - Service associated to the trace.
- *tag* - Indication of the malicious nature of the trace (label).
- *time* - Timestamp.
- *type* - Type of the trace.

Appendix D

Multi-step attack detection methods

In this Appendix we present several tables with information about the methods reviewed in Chapter 4. We include information about the type of data used by the methods as presented in section 4.4.2 for further details: 'A' stands for only alerts; 'E', for events; 'P', for packets, and 'T' indicates that alerts are used in triggering the detection process but other traces like events or packets are used in the identification of the steps.

Moreover, we show the type of dataset used in the experiments, also indicating when a private dataset is used ('Private'), when there is just an attack example ('Case study'), when there is a simulation with data expressly created for the experiment ('Simulation') and when there are no experiments ('No exp.'). We also indicate how the knowledge about the attacks is extracted ('Automatic', 'Manual' or 'Supervised', as in section 4.4.2) under the column 'Knowledge extraction'. Regarding reproducibility, we indicate if the method, the data and the models are accessible ('A_m','A_d','A_k', respectively), if the experiments are reproducible ('Rep.') and if the model of an example attack is provided (see section 4.4.3). Finally, the last column show the total number of citations of the publications presenting the method ('Total cit.'), extracted from Google Scholar.

Authors	References	Period	Type of data	Datasets	Knowledge extraction	A_m	A_d	A_k	Rep.	Attack model	Total cit.
Dain and Cunningham	[Dain 2001b] [Dain 2001a]	2001	A	DEFCON 8	Supervised	No	Yes	Yes	No	No	349 132
Valdes and Skinner	[Valdes 2001]	2001	A	Private	Manual	No	No	Yes	No	No	961
King et al.	[King 2005]	2005	T	Simulation Private	Manual	No	No	No	No	Yes	120
Strayer et al.	[Strayer 2005]	2005	T	Private	Automatic	No	No	–	No	No	25
B. Chen et al.	[Chen 2006]	2006	T	DARPA 2000 Private	Automatic	Yes	Yes	–	Yes	No	14
Li Wang et al.	[Wang 2006d] [Wang 2006c] [Wang 2010]	2006-2010	A	DARPA 2000 Private	Manual	Yes	Yes	Yes	Yes	Yes	11 11 25
Shaneck et al.	[Shaneck 2006]	2006	T	Private	Automatic	No	No	–	No	No	0
Zhu and Ghorbani	[Zhu 2006]	2006	A	DARPA 2000	Automatic	Yes	Yes	–	Yes	Yes	130
AmirHaeri and R. Jalili	[AmirHaeri 2009]	2009	A	DARPA 2000	Manual	Yes	Yes	Yes	Yes	Yes	0
Khakpour and Jalili	[Khakpour 2009]	2009	A	DARPA 2000	Supervised	No	Yes	No	No	Yes	1
Ebrahimi et al.	[Ebrahimi 2011]	2011	A	DARPA 2000	Automatic	Yes	Yes	–	Yes	Yes	12
Bateni et al.	[Bateni 2013a] [Bateni 2013b] [Bateni 2014]	2013-2014	A	DARPA 2000 Private	Supervised	Yes	Yes	No	No	Yes	18 9 0
Brogi et al.	[Brogi 2016] [Brogi 2018]	2016	A	Case study	Automatic	Yes	No	–	No	Yes	9 0
Pei et al.	[Pei 2016]	2016	E	Simulation	Automatic	Yes	No	–	No	Yes	7
C. Wang et al.	[Wang 2016]	2016	A	DARPA 2000	Automatic	Yes	Yes	–	Yes	Yes	6
Hossain et al.	[Hossain 2017]	2017	E	Private	Manual	Yes	No	Yes	No	Yes	5
Tian et al.	[Tian 2017]	2017	A	Simulation	Automatic	Yes	No	–	No	Yes	0

Table D.1: List of reviewed **similarity-based** multi-step attack detection methods in the category of **progressive construction**.

Category	Authors	References	Period	Type of data	Datasets	Knowledge extraction	A_m	A_d	A_k	Rep.	Attack model	Total cit.
Scenario clustering	Cuppens	[Cuppens 2001]	2001	A	Private	Automatic	Yes	No	–	No	No	379
	Julisch	[Julisch 2001] [Julisch 2002] [Julisch 2003a] [Julisch 2003b]	2001-2003	A	Private	Automatic	Yes	No	–	No	Yes	247 348 543 79
	J. Wang et al.	[Wang 2006a]	2006	A	Private	Automatic	Yes	No	–	No	No	15
	Murphy et al.	[Murphy 2009] [Murphy 2010]	2009-2010	A	Simulation	Automatic	Yes	No	–	No	Yes	1 1
	Colajanni, Marchetti and Manganiello	[Colajanni 2010] [Manganiello 2011]	2010-2011	A	DEFCON 18	Automatic	No	Yes	–	No	Yes	0 14
	Qiao et al.	[Qiao 2012]	2012	A	Private	Automatic	Yes	No	–	No	No	9
	Zhang et al.	[Zhang 2015]	2015	A	Private	Automatic	Yes	No	–	No	No	0
	Kawakani et al.	[Kawakani 2016] [Kawakani 2017]	2016-2017	A	Private	Automatic	Yes	No	–	No	Yes	4 1
	de Alvarenga et al.	[de Alvarenga 2018]	2018	A	Private	Automatic	Yes	No	–	No	Yes	2
	Ghafir et al.	[Ghafir 2018]	2018	A	Simulation	Automatic	Yes	No	–	No	No	1
	Haas and Fischer	[Haas 2018]	2018	A	Simulation Private	Automatic	Yes	No	–	No	No	0
Anomaly detection	Anming and Chunfu	[Anming 2004]	2004	E	DARPA 2000	Supervised	Yes	Yes	Yes	Yes	No	17
	Mathew and Upadhyaya	[Mathew 2009]	2009	E	Private	Automatic	Yes	No	–	No	No	2
	Shin et al.	[Shin 2013]	2013	A	DARPA 2000	Supervised	Yes	Yes	Yes	Yes	No	41
	Skopik, Friedberg et al.	[Skopik 2014] [Friedberg 2015]	2014-2015	E	Private Simulation	Supervised	Yes	No	No	No	No	25 72

Table D.2: List of reviewed **similarity-based** multi-step attack detection methods in the categories of **scenario clustering** and **anomaly detection**.

Authors	References	Period	Type of data	Datasets	Knowledge extraction	A_m	A_d	A_k	Rep.	Attack model	Total cit.
Cuppens et al.	[Cuppens 2002a] [Cuppens 2002b] [Cuppens 2002c] [Benferhat 2003]	2002-2003	A	Case study	Manual	Yes	No	Yes	No	No	82 70 978 36
Ning et al.	[Cui 2002]	2002-2010	A	DEFCON 8 DARPA 2000 DARPA GCP Simulation	Manual	Yes	Yes	Yes	Yes	Yes	17
	[Ning 2002a]										92
	[Ning 2002b]										228
	[Ning 2002c]										693
	[Ning 2003b]										237
	[Ning 2004a]										376
	[Xu 2004]										94
	[Xu 2006]										38
	[Zhai 2006]										29
	[Ning 2010]										2
Cheung et al.	[Cheung 2003]	2003	A	DARPA GCP	Manual	Yes	Yes	No	No	Yes	309
Yan and Liu	[Yan 2004]	2004-2005	A	DARPA 2000	Manual	Yes	Yes	Yes	Yes	Yes	2
	[Yan 2005]										1
Wang et al.	[Wang 2005b]	2005-2008	A	DARPA 2000 UCSB 2004	Manual	Yes	Yes	No	No	No	53
	[Wang 2006e]										229
	[Wang 2008]										5
Zhou, Pandey et al.	[Zhou 2007] [Pandey 2008]	2007-2008	A	DARPA 2000 Private	Manual	Yes	Yes	Yes	Yes	Yes	136 9
Alserhani et al.	[Alserhani 2010]	2010-2016	A	DARPA 2000 Private Simulation	Manual	Yes	Yes	No	No	No	42
	[Alserhani 2011]										2
	[Alserhani 2012]										2
	[Alserhani 2013]										7
	[Alserhani 2016]										0
D. Zhang et al.	[Zhang 2017]	2017	A	Private	Manual	Yes	No	No	No	Yes	0

Table D.3: List of reviewed multi-step attack detection methods based on **causal correlation** in the category of **prerequisites and consequences**.

Authors	References	Period	Type of data	Datasets	Knowledge extraction	A_m	A_d	A_k	Rep.	Attack model	Total cit.
Geib et al.	[Geib 2001]	2001	A	No exp.	Manual	No	No	No	No	No	206
Ourston et al.	[Ourston 2003]	2003	A	Private	Supervised	Yes	No	Yes	No	No	156
Qin and Lee	[Qin 2003]	2003-2007	A	DARPA GCP	Automatic	Yes	Yes	–	Yes	Yes	280
	[Qin 2004]			DEFCON 9							213
	[Qin 2005]			Private	Manual						86
	[Qin 2007]										87
Z. Ning and Gong	[Ning 2007]	2007	A	DARPA 2000	Manual	Yes	Yes	Yes	Yes	Yes	2
Z. Li et al.	[Li 2007b]	2007	A	DARPA 2000	Supervised	Yes	Yes	Yes	Yes	Yes	14
	[Li 2007c]			DARPA 1999							42
Fava, Yang, Byers et al.	[Byers 2008]	2008	A	Simulation	Supervised	Yes	No	No	No	No	9
	[Fava 2008]										70
	[Yang 2008]										23
Jemili et al.	[Jemili 2008]	2008-2009	A	DARPA GCP	Manual	No	Yes	No	No	Yes	0
	[Jemili 2009]										2
Lee et al.	[Lee 2008]	2008	A	DARPA 2000	Manual	No	Yes	No	No	No	25
Ma et al.	[Ma 2008]	2008	A	DARPA 2000	Automatic	Yes	Yes	–	Yes	Yes	14
Ahmadinejad and S. Jalili	[Ahmadinejad 2009]	2009	A	DARPA 2000	Supervised	Yes	Yes	Yes	Yes	Yes	23
Sadoddin & Ghorbani	[Sadoddin 2009]	2009	A	DARPA 2000	Automatic	Yes	Yes	–	Yes	Yes	55
				Private							
				Simulation							
Farhadi et al.	[Farhadi 2010]	2010	A	DARPA 2000	Supervised	Yes	Yes	Yes	Yes	No	2
Ren et al.	[Ren 2010]	2010	A	DARPA 2000	Supervised	Yes	Yes	Yes	Yes	Yes	60
				Private							
Bai et al.	[Bai 2011]	2011	A	Simulation	Manual	Yes	No	No	No	Yes	5
Cipriano et al.	[Cipriano 2011]	2011	A	UCSB 2008	Supervised	Yes	Yes	Yes	Yes	No	26
Katipally et al.	[Katipally 2011]	2011	A	Simulation	Supervised	No	No	No	No	No	9
Marchetti et al.	[Marchetti 2011b]	2011	A	DEFCON 18	Automatic	Yes	Yes	–	Yes	Yes	17

Table D.4: List of reviewed multi-step attack detection methods based on **causal correlation** in the category of **statistical inference**.

Authors	References	Period	Type of data	Datasets	Knowledge extraction	A_m	A_d	A_k	Rep.	Attack model	Total cit.
Anbarestani et al.	[Anbarestani 2012]	2012	A	DARPA 2000	Automatic	Yes	Yes	–	Yes	Yes	9
Lagzian et al.	[Lagzian 2012]	2012	A	DARPA 2000	Automatic	Yes	Yes	–	Yes	Yes	12
Luktarhan et al.	[Luktarhan 2012]	2012	A	DARPA 2000	Supervised	No	Yes	Yes	No	No	5
Man et al.	[Man 2012]	2012	A	DARPA 2000	Automatic	Yes	Yes	–	Yes	Yes	1
Soleimani et al.	[Soleimani 2012]	2012	A	DARPA 2000 Private	Supervised	Yes	Yes	Yes	Yes	Yes	13
Kavousi and Akbari	[Kavousi 2012] [Kavousi 2014]	2012-2014	A	DARPA 2000	Automatic	Yes	Yes	–	Yes	Yes	11 5
Bahareth et al.	[Bahareth 2013]	2013	A	No exp.	Supervised	Yes	No	No	No	No	5
Brahmi et al.	[Brahmi 2013]	2013	A	NSA	Automatic	Yes	Yes	–	Yes	No	2
Kholidy et al.	[Kholidy 2014a] [Kholidy 2014b] [Kholidy 2014c]	2014	A	DARPA 2000	Supervised	Yes	Yes	Yes	Yes	Yes	2 10 2
Kim & Park	[Kim 2014]	2014	A	Private	Automatic	No	No	–	No	No	22
Xuewei et al.	[Xuewei 2014]	2014	A	DARPA 2000	Supervised	Yes	Yes	Yes	Yes	Yes	14
Lv et al.	[Lv 2015]	2015	A	Private	Automatic	Yes	No	–	No	Yes	0
Chen et al.	[Chen 2016]	2016	E	Private	Manual	Yes	No	Yes	No	Yes	9
Y. Li et al.	[Li 2016]	2016	A	DARPA 2000	Automatic	Yes	Yes	–	Yes	Yes	1
Zhang, Xian et al.	[Xian 2016] [Zhang 2016]	2016	A	DEFCON 9 DEFCON 19	Automatic	Yes	Yes	–	Yes	No	0 0
Holgado et al.	[Holgado 2017]	2017	A	DARPA 2000 Simulation	Manual	Yes	Yes	Yes	Yes	Yes	6
Jia and Xu	[Jia 2017]	2017	P	Simulation	Manual	Yes	No	Yes	No	Yes	0
Lu et al.	[Lu 2018]	2018	A	Private	Manual	Yes	No	No	No	No	0
Saikia et al.	[Saikia 2018]	2018	A	DARPA 2000	Automatic	Yes	Yes	–	Yes	No	0
Shawly et al.	[Shawly 2018]	2018	A	DARPA 2000	Manual	Yes	Yes	Yes	Yes	Yes	0
Sun et al.	[Sun 2018]	2018	E	Simulation	Automatic	Yes	No	No	No	Yes	0

Table D.5: List of reviewed multi-step attack detection methods based on **causal correlation** in the category of **statistical inference** (**cont'd**).

Authors	References	Period	Type of data	Datasets	Knowledge extraction	A_m	A_d	A_k	Rep.	Attack model	Total cit.
Noel et al.	[Noel 2004]	2004	A	Simulation	Manual	Yes	No	Yes	No	No	171
Holsopple et al.	[Holsopple 2006]	2006	A	Simulation	Manual	No	No	No	No	No	54
Çamtepe et al.	[Çamtepe 2007]	2007	A	Simulation	Manual	Yes	No	No	No	Yes	61
Fava et al.	[Fava 2007] [Holsopple 2008]	2007-2008	A	Private Case study	Manual	Yes	No	Yes	No	Yes	23 48
Zhang et al.	[Zhang 2008]	2008	A	Simulation	Automatic	Yes	No	–	No	No	36
Du et al.	[Du 2010]	2010	A	Simulation	Manual	Yes	No	No	No	No	23
Roschke et al.	[Roschke 2011]	2011	A	Simulation	Manual	Yes	No	No	No	No	88
Chien and Ho	[Chien 2012]	2012	A	DARPA 2000	Manual	Yes	Yes	Yes	Yes	Yes	5
Lin et al.	[Lin 2012]	2012	A	Case study	Manual	Yes	No	Yes	No	Yes	12
Fayyad et al.	[Fayyad 2013]	2013	A	No exp.	Manual	Yes	No	No	No	No	4
Y. Luo et al.	[Luo 2014]	2014	A	Case study	Manual	Yes	No	Yes	No	Yes	6
S. Luo et al.	[Luo 2016]	2016	A	Case study	Manual	Yes	No	Yes	No	Yes	7
Kaynar	[Kaynar 2017]	2017	E	Simulation	Manual	Yes	No	No	No	No	0
Sicila et al.	[Sicilia 2017]	2017	A	Simulation	Manual	Yes	No	No	No	No	1
Angelini et al.	[Angelini 2018]	2018	A	Private	Manual	Yes	No	No	No	No	0

Table D.6: List of reviewed **structural-based** multi-step attack detection methods.

Authors	References	Period	Type of data	Datasets	Knowledge extraction	A_m	A_d	A_k	Rep.	Attack model	Total cit.
Eckmann et al.	[Eckmann 2002]	2002	E	Case study	Manual	Yes	No	Yes	No	No	514
Kruegel et al.	[Kruegel 2002]	2002	E	Private	Manual	Yes	No	No	No	No	136
Morin and Debar	[Morin 2003]	2003	A	Case study	Manual	Yes	No	No	No	Yes	239
L.-M. Wang et al.	[Wang 2004]	2004-2005	A	Case study	Manual	Yes	No	No	No	Yes	4
	[Wang 2005a]										0
Mathew et al.	[Mathew 2005]	2005	A	Private	Manual	Yes	No	No	No	No	39
Stotz, Sudit et al.	[Sudit 2005]	2005-2010	A	Private	Manual	No	No	No	No	No	47
	[Stotz 2007]										47
	[Mathew 2010]										9
Y. Xiao and C. Han	[Xiao 2006]	2006	A	DARPA 2000 Private	Manual	No	Yes	No	No	Yes	7
S.-H. Chien et al.	[Chien 2007]	2007	A	DARPA 2000	Manual	No	Yes	No	No	No	14
Kannadiga et al.	[Kannadiga 2007]	2007	A	Simulation	Manual	No	No	No	No	Yes	6
Z. Li et al.	[Li 2007d]	2007	A	DARPA 2000 Private	Manual	Yes	Yes	No	No	No	22
	[Li 2007e]										10
	[Wang 2007a]										2
	[Wang 2007b]										14
	[Zhang 2007]										12
Panichprecha et al.	[Panichprecha 2007]	2007	E	Case Study	Manual	Yes	No	No	No	Yes	1
Z. Liu et al.	[Liu 2008]	2008	A	DARPA 2000	Automatic	Yes	Yes	–	Yes	Yes	44
Long and Schwartz	[Long 2008]	2008	A	DARPA GCP	Manual	Yes	Yes	No	No	No	5
Soleimani and Ghorbani	[Soleimani 2008]	2008	A	DARPA 2000	Manual	Yes	Yes	No	No	Yes	8
Katipally et al.	[Katipally 2010]	2010	A	No exp.	Manual	No	No	No	No	No	13
F. Xuewei et al.	[Xuewei 2010]	2010	A	DARPA 2000	Manual	Yes	Yes	No	No	Yes	10

Table D.7: List of reviewed **case-based** multi-step attack detection methods.

Authors	References	Period	Type of data	Datasets	Knowledge extraction	A_m	A_d	A_k	Rep.	Attack model	Total cit.
Cheng et al.	[Cheng 2011]	2011	A	DARPA 2000 DARPA GCP Simulation	Supervised	Yes	Yes	Yes	Yes	No	19
Vogel et al.	[Vogel 2011a] [Vogel 2011b]	2011	E	Private	Manual	No	No	No	No	No	2 1
Chintabathina et al.	[Chintabathina 2012]	2012	A	No exp.	Manual	No	No	No	No	No	3
Zali et al.	[Zali 2012] [Zali 2013]	2012-2013	A	DARPA 2000	Manual	Yes	Yes	Yes	Yes	Yes	13 12
Giura and Wang	[Giura 2012a] [Giura 2012b]	2012	E	Private	Manual	Yes	No	No	No	No	53 56
Zargar	[Zargar 2014]	2014	A	DARPA 2000, ISCX	Manual	Yes	Yes	No	No	Yes	26
Jaeger et al.	[Jaeger 2015] [Ussath 2016a] [Ussath 2016b]	2015-2016	E	Private, Simulation	Manual	Yes	No	Yes	No	Yes	7 2 2
Navarro et al.	[Navarro 2016]	2016	E	Simulation	Manual	Yes	No	No	No	No	3
Amos-Binks et al.	[Amos-Binks 2017]	2017	A	DARPA GCP	Manual	No	Yes	No	No	Yes	3
Schindler	[Schindler 2017]	2017	A	Simulation	Manual	No	No	No	No	Yes	1
Wen et al.	[Wen 2017]	2017	P	Private	Manual	No	No	No	No	No	0
Q. Wang et al.	[Wang 2018]	2018	E	DARPA 2000, VAST 2012	Manual	Yes	Yes	No	No	Yes	0

Table D.8: List of reviewed **case-based** multi-step attack detection methods (**cont'd**).

Authors	References	Period	Type of data	Datasets	Knowledge extraction	A_m	A_d	A_k	Rep.	Attack model	Total cit.
Valeur et al.	[Valeur 2004]	2004	A	DARPA 2000 Others	Manual	Yes	Yes	No	No	Yes	542
Ning et al.	[Ning 2004b] [Ning 2004c]	2004	A	DARPA 2000	Manual	Yes	Yes	Yes	Yes	Yes	68 200
Yu and Frincke	[Yu 2004] [Yu 2007]	2004-2007	A	DARPA 2000 DARPA GCP	Supervised	Yes	Yes	Yes	Yes	Yes	49 69
L. Wang et al.	[Wang 2006b]	2006	A	DARPA 2000	Manual	Yes	Yes	No	No	No	5
Al-Mamory and Zhang, H.	[Al-Mamory 2007] [Al-Mamory 2008] [Al-Mamory 2009]	2007-2009	A	DARPA 2000 DEFCON 8	Manual	Yes	Yes	No	No	Yes	8 29 0
Du et al.	[Du 2009]	2009	A	Simulation	Automatic	Yes	No	–	No	No	12
Yang et al.	[Yang 2009]	2009	A	Private, Simulation	Manual	No	No	Yes	No	No	80
Farhadi et al.	[Farhadi 2011]	2011	A	DARPA 2000	Automatic	Yes	Yes	–	Yes	Yes	34
Marchetti et al.	[Marchetti 2011a]	2011	A	DEFCON 18	Automatic	No	Yes	–	No	Yes	12
Saad and Traore	[Saad 2012]	2012	A	DARPA 2000 UCSB 2002	Manual	Yes	Yes	No	No	No	9
Ahmed	[Saad 2014]	2014	A	DARPA 2000, ISCX	Manual	Yes	Yes	No	No	Yes	1
Chen et al.	[Chen 2014a]	2014	A	Private	Automatic	Yes	No	–	No	Yes	2
Abreu et al.	[Abreu 2015]	2015	E	No exp.	Manual	No	No	No	No	No	1
Ramaki et al.	[Ramaki 2015a] [Ramaki 2015b] [Ramaki 2016]	2015-2016	A	DARPA 2000 DARPA GCP ISCX	Automatic	Yes	Yes	–	Yes	Yes	25 1 12
Faraji Daneshgar et al.	[Faraji Daneshgar 2016]	2016	A	DARPA 2000, ISCX	Automatic	Yes	Yes	–	Yes	Yes	3
Shittu	[Shittu 2016]	2016	A	DARPA 2000 Private	Supervised	Yes	Yes	Yes	Yes	Yes	1
Barzegar and Shajari	[Barzegar 2018]	2018	A	DARPA 2000 MACCDC 2012	Manual	Yes	Yes	Yes	Yes	Yes	1

Table D.9: List of reviewed **mixed** multi-step attack detection methods.

Appendix E

Summary of AASG conditions

Type	**Name of function**	**Short name**	**Formula**
Absolute conditions	Equality	EQL	$g_1(n_a, r) = \begin{cases} 1, & \text{if } n_a = r \\ 0, & \text{otherwise} \end{cases}$
	Inequality	NEQ	$g_2(n_a, r) = \begin{cases} 1, & \text{if } n_a \neq r \\ 0, & \text{otherwise} \end{cases}$
	Prefix similarity	PFX	$g_3(n_a, r, \lambda) = \begin{cases} 1, & \text{if } \frac{l}{L} > \lambda \\ 0, & \text{otherwise} \end{cases}$
	Textual similarity	TXT	$g_4(n_a, r, \lambda) = \begin{cases} 1, & \text{if } jac(n_a, r) > \lambda \\ 0, & \text{otherwise} \end{cases}$
	Set-based	SET	$g_5(n_a, R) = \begin{cases} 1, & \text{if } n_a \in R \\ 0, & \text{otherwise} \end{cases}$
Relative conditions	Equality	SIM_EQL	$f_1(n_a, m_b) = \begin{cases} 1, & \text{if } n_a = m_b \\ 0, & \text{otherwise} \end{cases}$
	Common element	SIM_COM	$f_2(N_a, M_b) = \begin{cases} 1, & \text{if } N_a \cap M_b \neq \varnothing \\ 0, & \text{otherwise} \end{cases}$
	Prefix similarity	SIM_PFX	$f_3^*(n_a, m_b, \lambda) = \begin{cases} 1, & \text{if } \frac{l}{L} > \lambda \\ 0, & \text{otherwise} \end{cases}$
	Textual similarity	SIM_TXT	$f_4^*(n_a, m_b, \lambda) = \begin{cases} 1, & \text{if } jac(n_a, m_b) > \lambda \\ 0, & \text{otherwise} \end{cases}$
	Inequality	SIM_NEQ	$f_{neq}(n_a, m_p) = \begin{cases} 1, & \text{if } n_a \neq m_p \\ 0, & \text{otherwise} \end{cases}$

Table E.1: Functions used in the definition of the abstract events in the AASGs.

Appendix F

Examples of AASGs

Throughout this thesis, we have worked with three examples of AASG: Mill-1, Mill-2 and S2A. The first two were intended to be applied to the dataset DARPA 2000, while S2A has been conceived for both ISCX and HuMa. In this appendix, the formal and functional representations of the three mentioned AASGs are gathered together.

Node	Abstract event	List of conditions
κ_0	$\boldsymbol{e}_0^*$	$g_1(type_a, \text{'Sadmind\_Ping'}) = 1$
κ_1	$\boldsymbol{e}_1^*$	$g_5(type_a, \{\text{'Sadmind\_AO'}, \text{'Admind'}\}) = 1$, $f_1(ipsrc_a, ipsrc_p) = 1$, $f_1(ipdst_a, ipdst_p) = 1$
κ_2	$\boldsymbol{e}_2^*$	$g_1(type_a, \text{'Sadmind\_AO'}) = 1$, $f_1(ipsrc_a, ipsrc_p) = 1$, $f_1(ipdst_a, ipdst_p) = 1$
κ_3	$\boldsymbol{e}_3^*$	$g_5(type_a, \{\text{'Sadmind\_AO'}, \text{'Admind'}\}) = 1$, $f_{neq}(type_a, type_p) = 1$, $f_1(ipsrc_a, ipsrc_p) = 1$, $f_1(ipdst_a, ipdst_p) = 1$
κ_4	$\boldsymbol{e}_4^*$	$g_1(type_a, \text{'TelnetTerminaltype'}) = 1$, $f_1(ipdst_a, ipdst_p) = 1$
κ_5	$\boldsymbol{e}_5^*$	$g_1(type_a, \text{'Rsh'}) = 1$, $f_1(ipdst_a, ipdst_p) = 1$
κ_6	$\boldsymbol{e}_6^*$	$g_1(type_a, \text{'Rsh'}) = 1$, $f_1(ipsrc_a, ipdst_p) = 1$
κ_7	$\boldsymbol{e}_7^*$	$g_1(type_a, \text{'TelnetTerminaltype'}) = 1$, $f_2(\{ipsrc_a\}, \{ipsrc_p, ipdst_p\}) = 1$, $f_2(\{ipdst_a\}, \{ipsrc_p, ipdst_p\}) = 1$
κ_8	$\boldsymbol{e}_8^*$	$g_1(type_a, \text{'Mstream\_Zombie'}) = 1$, $f_{neq}(ipdst_a, ipdst_p) = 1$
κ_9	$\boldsymbol{e}_9^*$	$g_1(type_a, \text{'Mstream\_Zombie'}) = 1$, $f_2(\{ipsrc_a\}, \{ipsrc_p, ipdst_p\}) = 1$, $g_1(ipdst_a, 255.255.255.255) = 1$

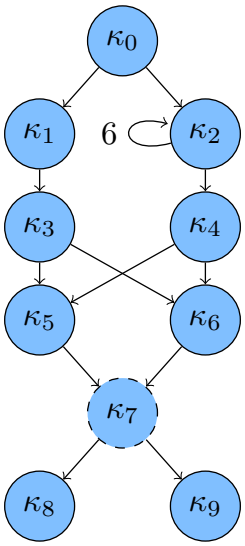


Figure F.1: Formal representation of AASG Mill-1. Presented in page 128.

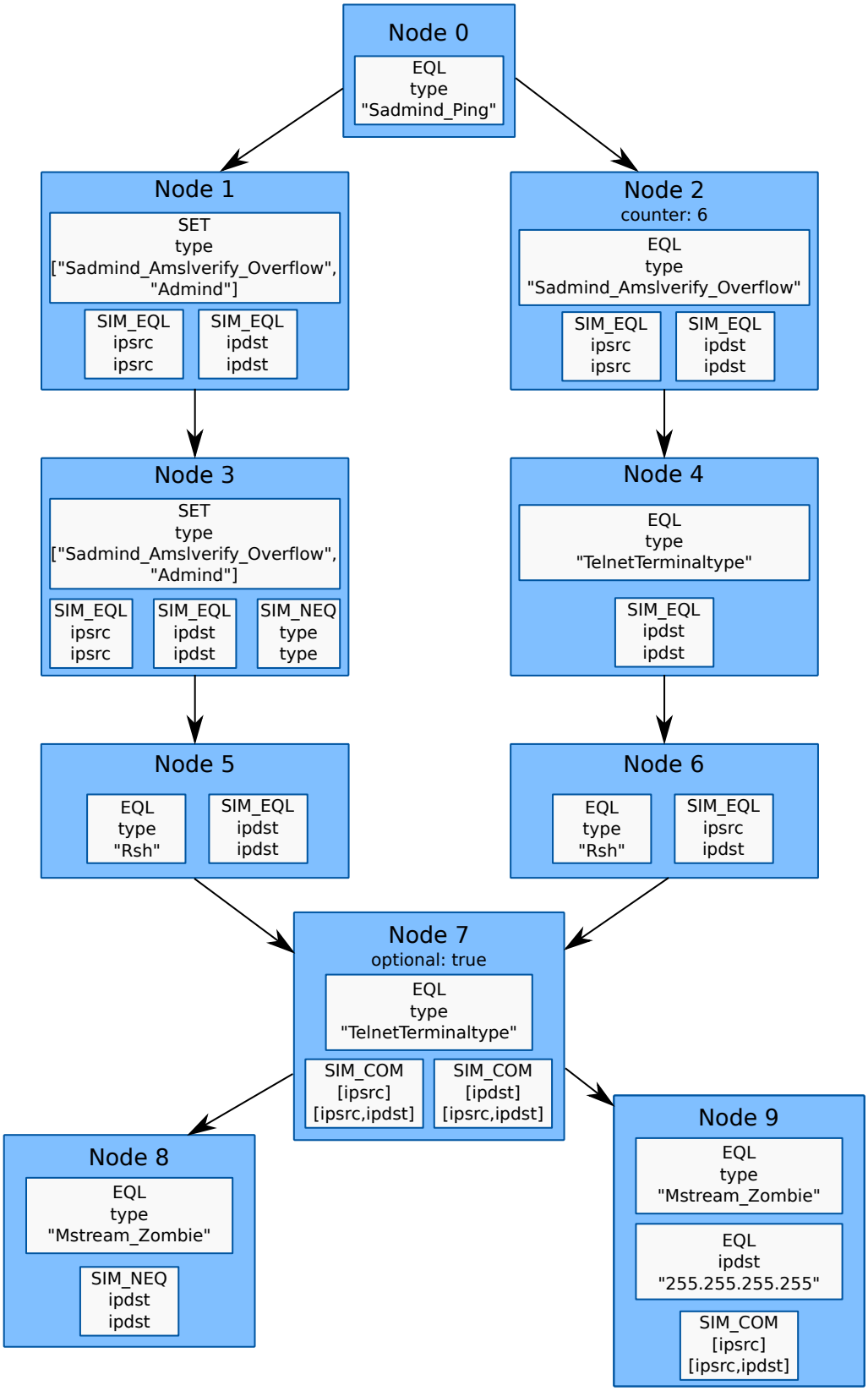
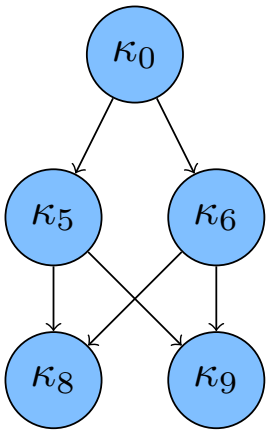


Figure F.2: Functional representation of AASG Mill-1. Presented in page 178.



Node	Abstract event	List of conditions
κ_0	$\boldsymbol{e}_0^*$	$g_1(type_a, \text{‘Sadmind\_AO’}) = 1$
κ_5	$\boldsymbol{e}_5^*$	$g_4(type_a, \{\text{‘Rsh’}, \text{‘FTP\_Put’}\}) = 1,$ $f_1(ipdst_a, ipdst_p) = 1$
κ_6	$\boldsymbol{e}_6^*$	$g_1(type_a, \{\text{‘Rsh’}, \text{‘FTP\_Put’}\}) = 1,$ $f_1(ipsrc_a, ipdst_p) = 1$
κ_8	$\boldsymbol{e}_8^*$	$g_1(type_a, \text{‘Mstream\_Zombie’}) = 1,$ $f_6(ipdst_a, ipdst_p) = 1$
κ_9	$\boldsymbol{e}_9^*$	$g_1(type_a, \text{‘Mstream\_Zombie’}) = 1$ $f_2(\{ipsrc_a\}, \{ipsrc_p, ipdst_p\}) = 1$ $g_1(ipdst_a, 255.255.255.255) = 1$

Figure F.3: Formal representation of AASG Mill-2. Presented in page 183.

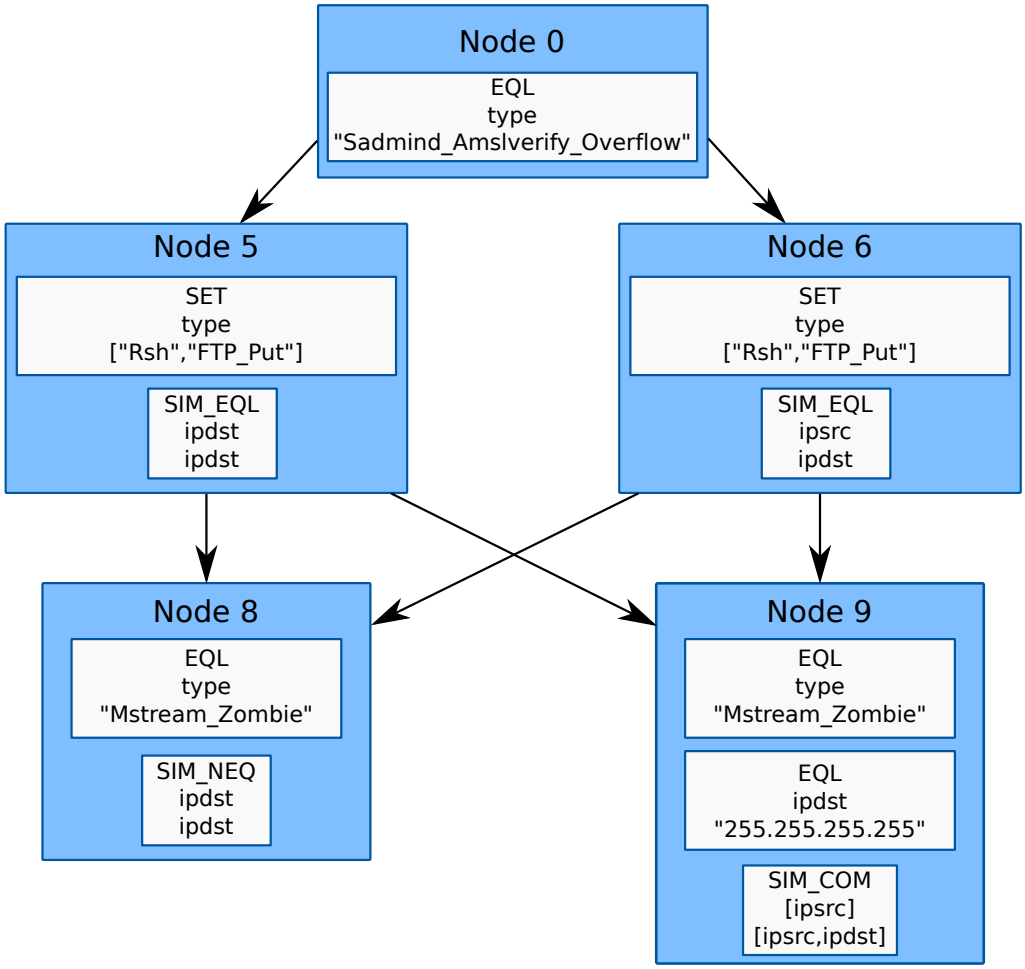
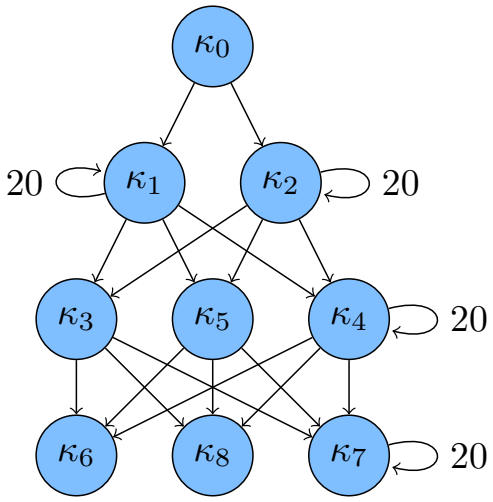


Figure F.4: Functional representation of AASG Mill-2. Presented in page 184.



Node	Abstract event	List of conditions
κ_0	e_0^*	$g_4(type_a, \text{'ICMP detected'}, 0.3) = 1$, $g_3(ipdst_a, \text{'192.168.0.0'}, 0.45) = 1$, $g_2(ipsrc_a, \text{'192.168.5.122'}) = 1$, $g_2(ipsrc_a, \text{'192.168.5.123'}) = 1$
κ_1	e_1^*	$f_1(type_a, type_b) = 1$, $f_1(ipsrc_a, ipsrc_b) = 1$, $f_3^*(ipdst_a, ipdst_b, 0.75) = 1$, $f_{neq}(ipdst_a, ipdst_b) = 1$
κ_2	e_2^*	$f_1(type_a, type_b) = 1$, $f_1(action_a, action_b) = 1$, $f_1(ipsrc_a, ipsrc_b) = 1$, $f_1(ipdst_a, ipdst_b) = 1$, $f_1(psrc_a, psrc_b) = 1$, $f_{neq}(service_a, service_b) = 1$
κ_3	e_3^*	$f_1(ipsrc_a, ipsrc_b) = 1$, $f_3^*(ipdst_a, ipdst_b, 0.75) = 1$, $g_4(type_a, \text{'SQL'}, 0.1) = 1$
κ_4	e_4^*	$f_1(ipsrc_a, ipsrc_b) = 1$, $f_3^*(ipdst_a, ipdst_b, 0.75) = 1$, $g_4(type_a, \text{'failure SSH'}, 0.5) = 1$
κ_5	e_5^*	$f_1(ipsrc_a, ipsrc_b) = 1$, $f_3^*(ipdst_a, ipdst_b, 0.75) = 1$, $g_4(type_a, \text{'EXPLOIT Attempt'}, 0.1) = 1$
κ_6	e_6^*	$f_{neq}(ipsrc_a, ipsrc_b) = 1$, $f_3^*(ipdst_a, ipdst_b, 0.75) = 1$, $g_4(type_a, \text{'SQL'}, 0.1) = 1$
κ_7	e_7^*	$f_{neq}(ipsrc_a, ipsrc_b) = 1$, $f_3^*(ipdst_a, ipdst_b, 0.75) = 1$, $g_4(type_a, \text{'failure SSH'}, 0.5) = 1$
κ_8	e_8^*	$f_{neq}(ipsrc_a, ipsrc_b) = 1$, $f_3^*(ipdst_a, ipdst_b, 0.75) = 1$, $g_4(type_a, \text{'EXPLOIT Attempt'}, 0.1) = 1$

Figure F.5: Formal representation of AAGS S2A. Presented in page 189.

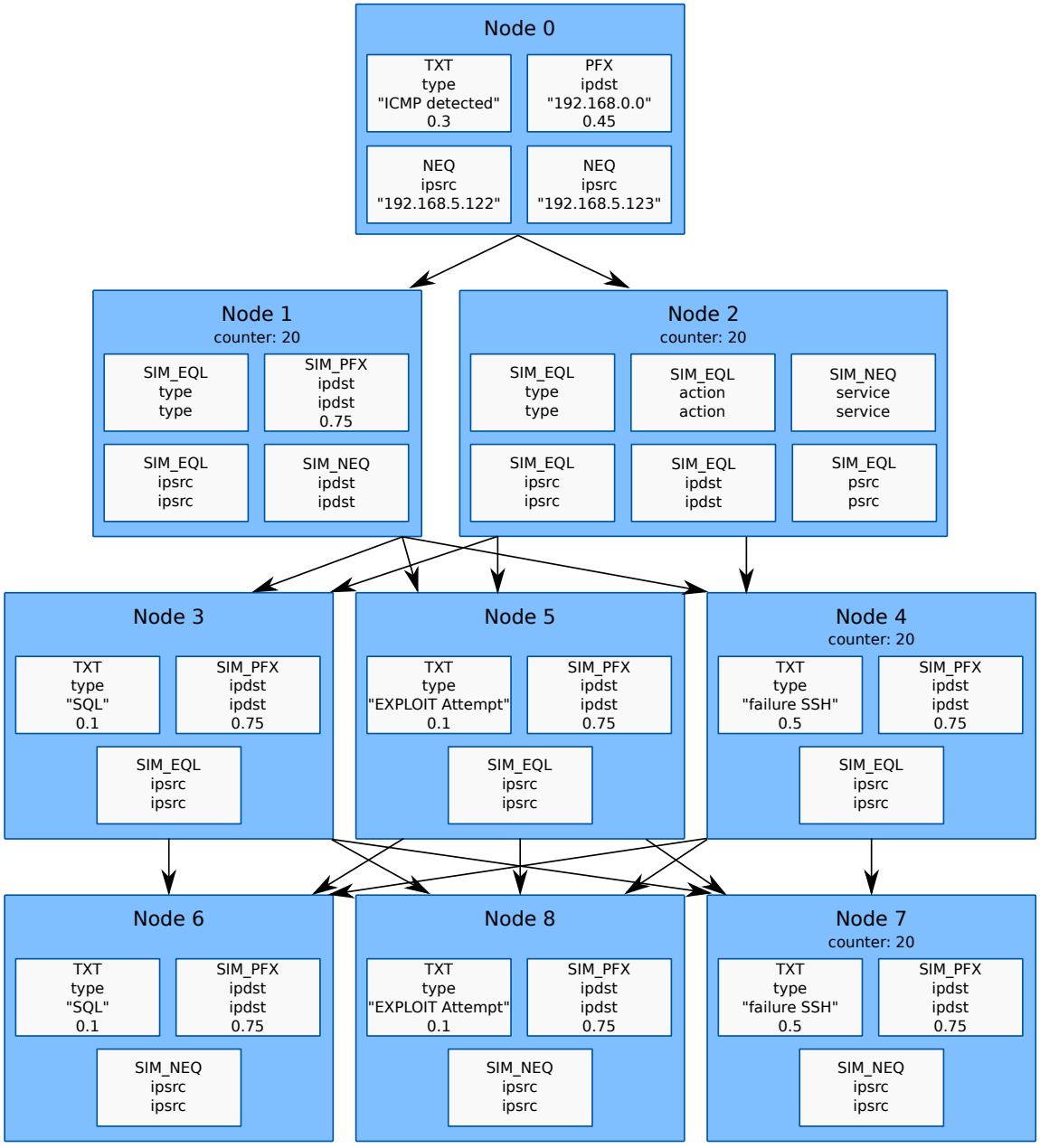


Figure F.6: Functional representation of AASG S2A. Presented in page 191.

Bibliography

[Abadeh 2015] Mohammad Saniee Abadeh and Jafar Habibi. A Hybridization of Evolutionary Fuzzy Systems and Ant Colony Optimization for Intrusion Detection. The ISC International Journal of Information Security (ISeCure), vol. 2, no. 1, 2015. (Cited on page 138.)

[Abreu 2015] Rui Abreu, Danny Bobrow, Hoda Eldardiry, Alexander Feldman, John Hanley, Tomonori Honda, Johan de Kleer, Alexandre Perez, Dave Archer and David Burke. Diagnosing Advanced Persistent Threats: A position paper. In Proceedings of the 26th International Workshop on Principles of Diagnosis (DX-2015), pages 193–200, 2015. (Cited on pages 89 and 256.)

[Ahmadinejad 2009] Seyed Hossein Ahmadinejad and Saeed Jalili. Alert correlation using correlation probability estimation and time windows. In International Conference on Computer Technology and Development (ICCTD'09), volume 2, pages 170–175, 2009. (Cited on pages 83, 171 and 251.)

[Ahmed 2016] Mohiuddin Ahmed, Abdun Naser Mahmood and Jiankun Hu. A survey of network anomaly detection techniques. Journal of Network and Computer Applications, vol. 60, pages 19–31, 2016. (Cited on page 77.)

[Aho 1972] Alfred V. Aho, Michael R Garey and Jeffrey D. Ullman. The transitive reduction of a directed graph. SIAM Journal on Computing, vol. 1, no. 2, pages 131–137, 1972. (Cited on page 232.)

[Akhgar 2014] Babak Akhgar, Andrew Staniforth and Francesca Bosco. Cyber crime and cyber terrorism investigator's handbook. Syngress, 2014. (Cited on page 13.)

[Al-Mamory 2007] Safaa O Al-Mamory and Hong Li Zhang. Scenario discovery using abstracted correlation graph. In International Conference on Computational Intelligence and Security, pages 702–706, 2007. (Cited on pages 89 and 256.)

[Al-Mamory 2008] Safaa O. Al-Mamory and Hong Li Zhang. Multistep attacks extraction using compiler techniques. In International Conference on High Performance Switching and Routing (HPSR), pages 183–188, 2008. (Cited on pages 89 and 256.)

[Al-Mamory 2009] Safaa O Al-Mamory and Hong Li Zhang. IDS alerts correlation using grammar-based approach. Journal in Computer Virology, vol. 5, pages 271–282, 2009. (Cited on pages 89 and 256.)

[Albanese 2017] Massimiliano Albanese and Sushil Jajodia. A Graphical Model to Assess the Impact of Multi-Step Attacks. The Journal of Defense Modeling and Simulation, vol. 15, no. 1, pages 79–93, 2017. (Cited on page 23.)

[Alien Vault 2014] Alien Vault. Life Cycle of a Log, 2014. (Cited on page 112.)

[Alnas 2013] Mohammed Alnas, Abdalla M. Hanashi and Elmabruk M. Laias. Detection of Botnet Multi-Stage Attack By Using Alert Correlation Model. The International Journal Of Engineering And Science (IJES), vol. 2, no. 10, pages 24–34, 2013. (Cited on page 80.)

[Alnusair 2017] Awny Alnusair, Chen Zhong, Majdi Rawashdeh, M Shamim Hossain and Atif Alamri. Context-aware multimodal recommendations of multimedia data in cyber situational awareness. Multimedia Tools and Applications, vol. 76, no. 21, pages 22823–22843, 2017. (Cited on page 106.)

[Alserhani 2010] Faeiz Alserhani, Monis Akhlaq, Irfan U. Awan, Andrea J. Cullen and Pravin Mirchandani. MARS: Multi-stage Attack Recognition System. In 24th IEEE International Conference on Advanced Information Networking and Applications, pages 753–759. IEEE, 2010. (Cited on pages 43, 80, 93, 170 and 250.)

[Alserhani 2011] Faeiz Alserhani and Monis Akhlaq. Event-based correlation systems to detect SQLI activities. In Proceedings Of the International Conference on Advanced Information Networking and Applications (AINA), Bioplis, Singapore, 2011. (Cited on pages 80, 93 and 250.)

[Alserhani 2012] Faeiz Alserhani. A framework for correlation and aggregation of security alerts in communication networks. Thesis, University of Bradford, 2012. (Cited on pages 80, 93 and 250.)

[Alserhani 2013] Faeiz Alserhani. A Framework for Multi-stage Attack Detection. In 2013 Saudi International Electronics, Communications and Photonics Conference (SIECPC), pages 1–6. IEEE, 2013. (Cited on pages 80, 93, 187 and 250.)

[Alserhani 2016] Faeiz Alserhani. Alert correlation and aggregation techniques for reduction of security alerts and detection of multistage attack. International Journal of Advanced Studies in Computers, Science and Engineering, vol. 5, no. 2, 2016. (Cited on pages 55, 80, 93 and 250.)

[AmirHaeri 2009] Maryam AmirHaeri and Rasool Jalili. RTEAS: A Real-Time Algorithm for Extracting Attack Scenarios from Intrusion Alert Stream. In 6th International ISC Conference on Information Security and Cryptology (ISCISC'09), 2009. (Cited on pages 57, 73 and 248.)

[Amos-Binks 2017] Adam Amos-Binks, Joshua Clark, Kirk Weston, Michael Winters and Khaled Harfoush. Efficient attack plan recognition using automated planning. In IEEE Symposium on Computers and Communications (ISCC), pages 1001–1006. IEEE, 2017. (Cited on pages 44, 87 and 255.)

[Anbarestani 2012] Reza Anbarestani, Behzad Akbari and Fariba Fathi. An iterative alert correlation method for extracting network intrusion scenarios. In 20th Iranian Conference on Electrical Engineering (ICEE), pages 684–689, 2012. (Cited on pages 36, 83, 111, 170, 171 and 252.)

[Angelini 2018] Marco Angelini, Silvia Bonomi, Emanuele Borzi, Antonella Del Pozzo, Simone Lenti and Giuseppe Santucci. An Attack Graph-based On-line Multi-step Attack Detector. In Proceedings of the 19th International Conference on Distributed Computing and Networking. ACM, 2018. (Cited on pages 84 and 253.)

[Anming 2004] Zhong Anming and Jia Chunfu. Study on the applications of Hidden Markov Models to computer intrusion detection. In Fifth World Congress on Intelligent Control and Automation (WCICA), volume 5, pages 4352–4356. IEEE, 2004. (Cited on pages 78, 170 and 249.)

[Aristophanes 1998] Aristophanes. "Clouds. Wasps. Peace". Loeb Classical Library. Hardvard University Press, Cambridge, MA, USA, 1998. (Cited on page 218.)

[Azodi 2013] Amir Azodi, David Jaeger, Feng Cheng and Christoph Meinel. Pushing the limits in event normalisation to improve attack detection in IDS/SIEM systems. In International Conference on Advanced Cloud and Big Data (CBD), pages 69–76. IEEE, 2013. (Cited on page 40.)

[Bahareth 2013] Fatmah A. Bahareth and Omaima O. Bamasak. Constructing attack scenario using sequential pattern mining with correlated candidate sequences. The Research Bulletin of JORDAN ACM-ISWSA, 2013. (Cited on pages 83 and 252.)

[Bai 2011] Hao Bai, Kunsheng Wang, Changzhen Hu, Gang Zhang and Xiaochuan Jing. Boosting performance in attack intention recognition by integrating multiple techniques. Frontiers of Computer Science in China, vol. 5, pages 109–118, 2011. (Cited on pages 83 and 251.)

[Barzegar 2018] Mahdiyeh Barzegar and Mehdi Shajari. Attack scenario reconstruction using intrusion semantics. Expert Systems with Applications, vol. 108, pages 119–133, 2018. (Cited on pages 89 and 256.)

[Bateni 2013a] Mehdi Bateni, Ahmad Baraani and Ali A. Ghorbani. Using Artificial Immune System and Fuzzy Logic for Alert Correlation. IJ Network Security, vol. 15, pages 190–204, 2013. (Cited on pages 57, 75, 92, 93 and 248.)

[Bateni 2013b] Mehdi Bateni, Ahmad Baraani, Ali A. Ghorbani and Abbas Rezaei. An AIS-inspired architecture for alert correlation. International Journal of Innovative Computing, Information & Control, vol. 9, pages 231–255, 2013. (Cited on pages 75, 93 and 248.)

[Bateni 2014] Mehdi Bateni and Ahmad Baraani. An Architecture for Alert Correlation Inspired By a Comprehensive Model of Human Immune System. International Journal of Computer Network and Information Security, vol. 6, page 47, 2014. (Cited on pages 75 and 248.)

[Benferhat 2003] Salem Benferhat, Fabien Autrel and Frédéric Cuppens. Enhanced correlation in an intrusion detection process. In MMM-ACNS, pages 157–170. Springer, 2003. (Cited on pages 79, 92, 93 and 250.)

[Bilge 2012] Leyla Bilge and Tudor Dumitras. Before we knew it: an empirical study of zero-day attacks in the real world. In Proceedings of the 2012 ACM conference on Computer and Communications Security, pages 833–844. ACM, 2012. (Cited on page 2.)

[Bishop 2006] Christopher M. Bishop. Pattern recognition and machine learning. Springer, 2006. (Cited on page 82.)

[Bollobás 2013] Béla Bollobás. Modern graph theory, volume 184. Springer Science & Business Media, 2013. (Cited on pages 42 and 43.)

[Bóna 2002] Miklós Bóna. A walk through combinatorics: an introduction to enumeration and graph theory. World Scientific Publishing Company, 2002. (Cited on page 42.)

[Brahmi 2013] Hanen Brahmi and Sadok Ben Yahia. Discovering Multi-stage Attacks Using Closed Multi-dimensional Sequential Pattern Mining. In International Conference on Database and Expert Systems Applications, pages 450–457. Springer, 2013. (Cited on pages 43, 83 and 252.)

[Brogi 2016] Guillaume Brogi and Valérie Viet Triem Tong. TerminAPTor: Highlighting Advanced Persistent Threats through Information Flow Tracking. In 8th IFIP International Conference on New Technologies, Mobility and Security (NTMS), pages 1–5. IEEE, 2016. (Cited on pages 21, 43, 46, 49, 52, 74, 94, 96 and 248.)

[Brogi 2018] Guillaume Brogi. Real-time detection of Advanced Persistent Threats using Information Flow Tracking and Hidden Markov Models. PhD thesis, Conservatoire National des Arts et Métiers (CNAM), 2018. (Cited on pages 74 and 248.)

[Brookson 2015] Charles Brookson, Scott Cadzow, Ralph Eckmaier, Jörg Eschweiler, Berthold Gerber, Alessandro Guarino, Kai Rannenberg, Jon Shamah and Slawomir Górniak. Definition of Cybersecurity. Gaps and overlaps in standardisation. Technical report, European Union Agency For Network And Information Security (ENISA), 2015. (Cited on pages 15 and 16.)

[Byers 2008] Stephen R. Byers and Shanchieh J. Yang. Real-time fusion and projection of network intrusion activity. In 11th International Conference on Information Fusion (FUSION), pages 1–8, 2008. (Cited on pages 82, 93 and 251.)

[Cao 2014] Yonghui Cao. Study of Four Types of Learning Bayesian Networks Cases. Applied Mathematics & Information Sciences, vol. 8, no. 1, page 379, 2014. (Cited on page 154.)

[Caulkins 2018] Bruce Caulkins, Tiffani Marlowe and Ashley Reardon. Cybersecurity Skills to Address Today's Threats. In International Conference on Human Factors in Cybersecurity, pages 187–192. Springer, 2018. (Cited on page 2.)

[Çamtepe 2007] Seyit Ahmet Çamtepe and Bülent Yener. Modeling and detection of complex attacks. In Third International Conference on Security and Privacy in Communications Networks and the Workshops (SecureComm), pages 234–243. IEEE, 2007. (Cited on pages 22, 84 and 253.)

[CERT-MU 2017] CERT-MU. The Wannacry ransomware. Technical report, The Computer Emergency Response Team of Mauritius, 05 2017. (Cited on page 30.)

[Check Point 2018] Check Point. 2018 Security Report, 2018. (Cited on page 1.)

[Chen 2006] Bing Chen, Joohan Lee and Annie S. Wu. Active event correlation in Bro IDS to detect multi-stage attacks. In Fourth IEEE International Workshop on Information Assurance (IWIA'06), pages 16 pp.–50. IEEE, 2006. (Cited on pages 22, 33, 48, 52, 59, 71, 74, 171 and 248.)

[Chen 2007] Shihyen Chen, Bin Ma and Kaizhong Zhang. The normalized similarity metric and its applications. In IEEE International Conference on Bioinformatics and Biomedicine (BIBM), pages 172–180. IEEE, 2007. (Cited on pages 45 and 57.)

[Chen 2014a] Guanlin Chen, Yujia Zhang and Can Wang. A wireless multi-step attack pattern recognition method for WLAN. Expert Systems with Applications, vol. 41, no. 16, pages 7068–7076, 2014. (Cited on pages 46, 49, 52, 53, 89, 227 and 256.)

[Chen 2014b] Ping Chen, Lieven Desmet and Christophe Huygens. A study on Advanced Persistent Threats. In IFIP International Conference on Communications and Multimedia Security, pages 63–72, 2014. (Cited on page 29.)

[Chen 2016] Chia-Mei Chen, Dah-Jyh Guan, Yu-Zhi Huang and Ya-Hui Ou. Anomaly network intrusion detection using Hidden Markov Model. International Journal of Innovative Computing, Information and Control,, vol. 12, pages 569–580, 2016. (Cited on pages 56, 81 and 252.)

[Cheng 2011] Bo-Chao Cheng, Guo-Tan Liao, Chu-Chun Huang and Ming-Tse Yu. A Novel Probabilistic Matching Algorithm for Multi-Stage Attack Forecasts. IEEE Journal on Selected Areas in Communications, vol. 29, no. 7, pages 1438–1448, 2011. (Cited on pages 31, 87 and 255.)

[Cherdantseva 2016] Yulia Cherdantseva, Pete Burnap, Andrew Blyth, Peter Eden, Kevin Jones, Hugh Soulsby and Kristan Stoddart. A review of cyber security risk assessment methods for SCADA systems. Computers & security, vol. 56, pages 1–27, 2016. (Cited on page 4.)

[Cheung 2003] Steven Cheung, Ulf Lindqvist and Martin W. Fong. Modeling multistep cyber attacks for scenario recognition. In Proceedings of DARPA information survivability conference and exposition, volume 1, pages 284–292. IEEE, 2003. (Cited on pages 37, 38, 80, 92 and 250.)

[Chien 2007] Sheng-Hui Chien, Erh-Hsien Chang, Chih-Yung Yu and Cheng-Seen Ho. Attack subplan-based attack scenario correlation. In International Conference on Machine Learning and Cybernetics, volume 4, pages 1881–1887, 2007. (Cited on pages 87 and 254.)

[Chien 2012] Sheng-Hui Chien and Cheng-Seen Ho. A Novel Threat Prediction Framework for Network Security. Advances in Information Technology and Industry Applications, pages 1–9, 2012. (Cited on pages 39, 85 and 253.)

[Chintabathina 2012] Sandeep Chintabathina, Jorge Villacis, Jessie J. Walker and Hugo R. Gomez. Plan recognition in intrusion detection systems using logic programming. In IEEE Conference on Technologies for Homeland Security (HST), pages 609–613, 2012. (Cited on pages 21, 43, 86 and 255.)

[Cipriano 2011] Casey Cipriano, Ali Zand, Amir Houmansadr, Christopher Kruegel and Giovanni Vigna. Nexat: A history-based approach to predict attacker actions. In Proceedings of the 27th Annual Computer Security Applications Conference, pages 383–392. ACM, 2011. (Cited on pages 22, 49, 52, 53, 57, 59, 83 and 251.)

[Clackson 2007] James Clackson. Indo-european linguistics: An introduction. Cambridge University Press, 2007. (Cited on page 139.)

[CNSS 2015] CNSS. Committee on national security systems (CNSS) glossary. Technical report, US Comittee on National Security Systems, 2015. (Cited on pages 13, 16 and 240.)

[Colajanni 2010] Michele Colajanni, Mirco Marchetti and Fabio Manganiello. Machine learning algorithms for clustering and correlating security alerts in Intrusion Detection Systems. Report, Universita degli Studi di Modena e Reggio Emilia, 2010. (Cited on pages 57, 77, 89 and 249.)

[Coudriau 2016] Marc Coudriau, Abdelkader Lahmadi and Jerome François. Topological Analysis and Visualisation of Network Monitoring Data: Darknet case study. In International Workshop on Information Forensics and Security (WIFS), Abu Dhabi, United Arab Emirates, 2016. IEEE. (Cited on page 207.)

[Craigen 2014] Dan Craigen, Nadia Diakun-Thibault and Randy Purse. Defining Cybersecurity. Technology Innovation Management Review, vol. 4, no. 10, 2014. (Cited on page 15.)

[CrowdStrike 2018] CrowdStrike. Global Threat Report: Blurring the Lines Between Statecraft and Tradecraft, 2018. (Cited on page 23.)

[Cui 2002] Yun Cui. A toolkit for intrusion alerts correlation based on prerequisites and consequences of attacks, 2002. (Cited on pages 43, 79, 93 and 250.)

[Cuppens 2000] Frédéric Cuppens and Rodolphe Ortalo. LAMBDA: A language to model a database for detection of attacks. In International Workshop on Recent Advances in Intrusion Detection, pages 197–216. Springer, 2000. (Cited on pages ix, 37, 38, 55, 79, 124 and 225.)

[Cuppens 2001] Frédéric Cuppens. Managing Alerts in a Multi-Intrusion Detection Environmen. In Annual Computer Security Applications Conference (ACSAC), volume 1, page 22, 2001. (Cited on pages 22, 75, 92, 93 and 249.)

[Cuppens 2002a] Frédéric Cuppens, Fabien Autrel, Alexandre Miège and Salem Benferhat. Correlation in an intrusion detection process. In Internet Security Communication Workshop, pages 153–172, 2002. (Cited on pages 79, 92, 93 and 250.)

[Cuppens 2002b] Frédéric Cuppens, Fabien Autrel, Alexandre Miège and Salem Benferhat. Recognizing malicious intention in an intrusion detection process. In Second International Conference on Hybrid Intelligent Systems (HIS), pages 806–817, 2002. (Cited on pages 79, 92, 93, 105 and 250.)

[Cuppens 2002c] Frédéric Cuppens and Alexandre Miège. Alert correlation in a cooperative intrusion detection framework. In IEEE Symposium on Security and Privacy, pages 202–215. IEEE, 2002. (Cited on pages 43, 79, 92, 93, 104 and 250.)

[CVEDetails 2019] CVEDetails. Browse cve vulnerabilities by date, 2019. https://www.cvedetails.com/browse-by-date.php (Date last accessed 10 Jan, 2019). (Cited on pages 14 and 15.)

[Dain 2001a] Oliver M. Dain and Robert K. Cunningham. Building scenarios from a heterogeneous alert stream. In Proceedings of the 2001 IEEE workshop on Information Assurance and Security, volume 235. West Point, NY, USA, 2001. (Cited on pages 53, 73 and 248.)

[Dain 2001b] Oliver M. Dain and Robert K. Cunningham. Fusing a heterogeneous alert stream into scenarios. In Proceedings of the 2001 ACM workshop on Data Mining for Security Applications, volume 13, 2001. (Cited on pages 22, 53, 56, 64, 73, 92 and 248.)

[Daly 2006] Ronan Daly, Qiang Shen and Stuart Aitken. Using Ant Colony Optimization in learning Bayesian network equivalence classes. In Proceedings of the 2006 UK Workshop on Computational Intelligence, pages 111–118, 2006. (Cited on page 155.)

[DARPA 1981a] Internet Program Protocol Specification DARPA. Internet Control Message Protocol. RFC 792, RFC Editor, September 1981. https://tools.ietf.org/html/rfc792 (Date last accessed 28 Aug, 2018). (Cited on page 31.)

[DARPA 1981b] Internet Program Protocol Specification DARPA. Internet Protocol. RFC 791, RFC Editor, September 1981. https://tools.ietf.org/html/rfc792 (Date last accessed 28 Aug, 2018). (Cited on page 51.)

[Daud 2018] Ali Yusry Daud, Osman Ghazali and Mohd Nizam Omar. Anomaly Techniques in Stepping Stone Detection (SSD): A Review. Journal of Telecommunication, Electronic and Computer Engineering (JTEC), vol. 10, no. 1-10, pages 61–66, 2018. (Cited on page 21.)

[de Alvarenga 2018] Sean Carlisto de Alvarenga, Sylvio Barbon Jr, Rodrigo Sanches Miani, Michel Cukier and Bruno Bogaz Zarpelão. Process mining and hierarchical clustering to help intrusion alert visualization. Computers & Security, vol. 73, pages 474–491, 2018. (Cited on pages 76 and 249.)

[de Vries 2012] Johannes de Vries, Hans Hoogstraaten, Jan van den Berg and Semir Daskapan. Systems for Detecting Advanced Persistent Threats: A Development Roadmap Using Intelligent Data Analysis. In International Conference on Cyber Security (CyberSecurity), pages 54–61. IEEE, 2012. (Cited on pages 22 and 28.)

[Deepak 2015] P Deepak and Prasad M Deshpande. Operators for similarity search: Semantics, techniques and usage scenarios. Springer, 2015. (Cited on page 45.)

[Dempster 2008] Arthur P. Dempster. A generalization of Bayesian inference. Technical report, Harvard University, 2008. (Cited on page 155.)

[Dorigo 1991] Marco Dorigo, Vittorio Maniezzo and Alberto Colorni. Positive Feedback as a Search Strategy. Report 91-016, Politecnico di Milano, 1991. (Cited on page 138.)

[Dorigo 2004] Marco Dorigo and Thomas Stützle. Ant colony optimization. MIT Press, 2004. (Cited on pages 136 and 144.)

[Dousson 1994] Christophe Dousson. Suivi d'évolutions et reconnaissance de chroniques. Thesis, Université de Toulouse, 1994. (Cited on page 41.)

[Du 2009] Haitao Du, Christopher T. Murphy, Jordan Bean and Shanchieh J. Yang. Toward unsupervised classification of non-uniform cyber attack tracks. In 12th International Conference on Information Fusion (FUSION), pages 1919–1925. IEEE, 2009. (Cited on pages 48, 52, 57, 88, 93 and 256.)

[Du 2010] Haitao Du, Daniel F. Liu, Jared Holsopple and Shanchieh J. Yang. Toward Ensemble Characterization and Projection of Multistage Cyber Attacks. In Proceedings of 19th International Conference on Computer Communications and Networks, pages 1–8. IEEE, 2010. (Cited on pages 22, 84, 93 and 253.)

[Duffany 2018] Jeffrey L. Duffany. Computer and Network Security Essentials, chapter Computer Security, pages 3–20. Springer, Cham, 2018. (Cited on pages 13, 14, 16 and 241.)

[Dutt 2013] Varun Dutt and Amanjot Kaur. Cyber security: testing the effects of attack strategy, similarity, and experience on cyber attack detection. International Journal of Trust Management in Computing and Communications, vol. 1, no. 3-4, pages 261–273, 2013. (Cited on page 85.)

[Ebrahimi 2011] Ali Ebrahimi, Ahmad Habibi Zad Navin, Mir Kamal Mirnia, Hadi Bahrbegi and Amir Azimi Alasti Ahrabi. Automatic attack scenario discovering based on a new alert correlation method. In IEEE International Systems Conference (SysCon), pages 52–58, 2011. (Cited on pages 43, 73 and 248.)

[EC 2010] European Commission. EC. Standard on Logging and Monitoring, 2010. (Cited on page 19.)

[Eckmann 2002] Steven T. Eckmann, Giovanni Vigna and Richard A. Kemmerer. STATL: An attack language for state-based intrusion detection. Journal of Computer Security, vol. 10, no. 1-2, pages 71–103, 2002. (Cited on pages 41, 86, 92 and 254.)

[Ehrenfeld 2017] Jesse M. Ehrenfeld. Wannacry, cybersecurity and health information technology: A time to act. Journal of Medical Systems, vol. 41, no. 7, page 104, 2017. (Cited on pages 1 and 29.)

[EY 2017] EY. "WannaCry" ransomware attack - Technical intelligence analysis. Technical report, Ernst & Young Global Limited, 05 2017. (Cited on page 30.)

[Faraji Daneshgar 2016] Fatemeh Faraji Daneshgar and Maghsoud Abbaspour. Extracting fuzzy attack patterns using an online fuzzy adaptive alert correlation framework. Security and Communication Networks, vol. 9, pages 2245–2260, 2016. (Cited on pages 49, 52, 55, 88, 111, 172 and 256.)

[Farhadi 2010] H Farhadi, Rasool Jalili and Mohammad Khansari. Attack Plan Recognition Using Markov Model. In Proceedings of the 7th International ISC Conference on Information Security and Cryptology, 2010. (Cited on pages 81 and 251.)

[Farhadi 2011] Hamid Farhadi, Maryam AmirHaeri and Mohammad Khansari. Alert correlation and prediction using data mining and HMM. The ISC International Journal of Information Security, vol. 3, no. 2, 2011. (Cited on pages 32, 88, 111, 170 and 256.)

[Fava 2007] Daniel S. Fava, Jared Holsopple, Shanchieh J. Yang and Brian Argauer. Terrain and behavior modeling for projecting multistage cyber attacks. In 10th International Conference on Information Fusion, pages 1–7. IEEE, 2007. (Cited on pages 84, 93 and 253.)

[Fava 2008] Daniel S. Fava, Stephen R. Byers and Shanchieh J. Yang. Projecting cyberattacks through variable-length Markov models. IEEE Transactions on Information Forensics and Security, vol. 3, no. 3, pages 359–369, 2008. (Cited on pages 82, 93 and 251.)

[Fayyad 2013] Seraj Fayyad and Christoph Meinel. New Attack Scenario Prediction Methodology. In 10th International Conference on Information Technology: New Generations (ITNG), pages 53–59. IEEE, 2013. (Cited on pages 56, 84, 93 and 253.)

[Feng 2014] Wenying Feng, Qinglei Zhang, Gongzhu Hu and Jimmy Xiangji Huang. Mining Network Data for Intrusion Detection Through Combining SVMs with Ant Colony Networks. Future Generation Computer Systems, vol. 37, pages 127–140, 2014. (Cited on page 138.)

[Fernandes 2016] Gilberto Fernandes, Luiz F. Carvalho, Joel J. P. C. Rodrigues and Mario Lemes Proença. Network Anomaly Detection Using IP Flows with Principal Component Analysis and Ant Colony Optimization. J. Netw. Comput. Appl., vol. 64, pages 1–11, 2016. (Cited on page 138.)

[Fernandez 2014] Silvino Fernandez, Segundo Alvarez, Diego Díaz, Miguel Iglesias and Borja Ena. Scheduling a Galvanizing Line by Ant Colony Optimization. In Swarm Intelligence, pages 146–157. Springer, 2014. (Cited on page 138.)

[FFIEC 2018] US Federal Financial Institutions Examination Council. FFIEC. Glossary in FFIEC IT Examination Handbook InfoBase, 2018. https://ithandbook.ffiec.gov/glossary.aspx (Date last accessed 5 Jun, 2018). (Cited on pages 13 and 240.)

[Fink 2014] Glenn A. Fink, Jereme N. Haack, A. David McKinnon and Errin W. Fulp. Defense on the move: ant-based cyber defense. IEEE Secur. Priv., vol. 12, no. 2, pages 36–43, 2014. (Cited on page 138.)

[Fire Eye 2018] Fire Eye. M-Trends 2018, 2018. (Cited on page 23.)

[Friedberg 2015] Ivo Friedberg, Florian Skopik, Giuseppe Settanni and Roman Fiedler. Combating Advanced Persistent Threats: From network event correlation to incident detection. Computers & Security, vol. 48, no. C, pages 35–57, 2015. (Cited on pages 77, 104 and 249.)

[Geib 2001] Christopher W. Geib and Robert P. Goldman. Plan recognition in intrusion detection systems. In DARPA Information Survivability Conference & Exposition II (DISCEX), volume 1, pages 46–55. IEEE, 2001. (Cited on pages 81, 92, 104 and 251.)

[Ghafir 2018] Ibrahim Ghafir, Mohammad Hammoudeh, Vaclav Prenosil, Liangxiu Han, Robert Hegarty, Khaled Rabie and Francisco J Aparicio-Navarro. Detection of advanced persistent threat using machine-learning correlation analysis. Future Generation Computer Systems, vol. 89, pages 349–359, 2018. (Cited on pages 64, 76 and 249.)

[Ghosh 2015] Nirnay Ghosh, Ishan Chokshi, Mithun Sarkar, Soumya K Ghosh, Anil Kumar Kaushik and Sajal K Das. Netsecuritas: An integrated attack graph-based security assessment tool for enterprise networks. In Proceedings of the 2015 International Conference on Distributed Computing and Networking, page 30. ACM, 2015. (Cited on page 17.)

[Gibson 1982] William Gibson. Burning Chrome. Omni, no. 46, 1982. (Cited on page 13.)

[Giura 2012a] Paul Giura and Wei Wang. A Context-Based Detection Framework for Advanced Persistent Threats. In Proceedings of the 2012 International Conference on Cyber Security, pages 69–74. IEEE Computer Society, 2012. (Cited on pages 87 and 255.)

[Giura 2012b] Paul Giura and Wei Wang. Using large scale distributed computing to unveil Advanced Persistent Threats. Science Journal, vol. 1, no. 3, pages 93–105, 2012. (Cited on pages 87 and 255.)

[Glass 2000] Gene V. Glass. Meta-analysis at 25, 2000. http://www.gvglass.info/papers/meta25.html (Date last accessed 24 Sept, 2017). (Cited on page 67.)

[Gomaa 2013] Wael H Gomaa and Aly A Fahmy. A survey of text similarity approaches. International Journal of Computer Applications, vol. 68, no. 13, 2013. (Cited on page 50.)

[Goodman 2016] Steven N Goodman, Daniele Fanelli and John PA Ioannidis. What does research reproducibility mean? Science Translational Medicine, vol. 8, pages 341ps12–341ps12, 2016. (Cited on page 96.)

[Görnitz 2009] Nico Görnitz, Marius Kloft, Konrad Rieck and Ulf Brefeld. Active learning for network intrusion detection. In Proceedings of the 2nd ACM workshop on Security and Artificial Intelligence, pages 47–54. ACM, 2009. (Cited on page 100.)

[Goutam 2017] Rajesh Kumar Goutam. Design Issues in Stepping Stone Detection. International Journal of Computer Applications, vol. 166, no. 9, 2017. (Cited on page 21.)

[Guichard 2017] David Guichard. An introduction to combinatorics and graph theory. Self-published, 2017. (Cited on page 42.)

[Gutiérrez 2007] Sergio Gutiérrez, Grégory Valigiani, Pierre Collet and Carlos Delgado Kloos. Adaptation of the ACO heuristic for sequencing learning activities. In Proceedings of the European Conference on Technology Enhanced Learning (EC-TEL), pages 17–20, 2007. (Cited on page 138.)

[Haas 2018] Steffen Haas and Mathias Fischer. GAC: graph-based alert correlation for the detection of distributed multi-step attacks. In Proceedings of the 33rd Annual ACM Symposium on Applied Computing, pages 979–988. ACM, 2018. (Cited on pages 48, 52, 59, 76 and 249.)

[Haldenbilen 2013] Soner Haldenbilen, Cenk Ozan and Ozgur Baskan. An Ant Colony Optimization algorithm for area traffic control, book Ant Colony Optimization - Techniques and Applications, Chapter 4, pages 87–105. INTECH Open Access Publisher, 2013. (Cited on page 138.)

[Haopu 2016] Yang Haopu. Method for behavior-prediction of APT attack based on dynamic Bayesian game. In IEEE International Conference on Cloud Computing and Big Data Analysis (ICCCBDA), pages 177–182. IEEE, 2016. (Cited on page 85.)

[Heckerman 1995] David Heckerman. A tutorial on learning with Bayesian networks. Technical report, Microsoft, 1995. (Cited on page 154.)

[Holgado 2017] Pilar Holgado, Victor A. Villagra and Luis Vazquez. Real-time multistep attack prediction based on Hidden Markov Models. IEEE Transactions on Dependable and Secure Computing, 2017. (Cited on pages 31, 33, 42, 43, 82, 170 and 252.)

[Holsopple 2006] Jared Holsopple, Shanchieh J. Yang and Moises Sudit. TANDI: Threat assessment of network data and information. In Defense and Security Symposium. International Society for Optics and Photonics, 2006. (Cited on pages 84, 89, 93 and 253.)

[Holsopple 2008] Jared Holsopple and Shanchieh J. Yang. FuSIA: Future situation and impact awareness. In 11th International Conference on Information Fusion, pages 1–8. IEEE, 2008. (Cited on pages 84, 85, 93 and 253.)

[Hossain 2017] Md Nahid Hossain, Sadegh M Milajerdi, Junao Wang, Birhanu Eshete, Rigel Gjomemo, R Sekar, Scott D Stoller and VN Venkatakrishnan. SLEUTH: Real-time attack scenario reconstruction from COTS audit data. In Proceedings of the 26th USENIX Security Symposium, pages 487–504, 2017. (Cited on pages 74, 105 and 248.)

[Hu 2013] Liang Hu, Nannan Xie, Sheng Chai and Nurbol. A Description Model of Multi-Step Attack Planning Domain Based on Knowledge Representation. Chinese Journal of Electronics, vol. 22, no. CJE-3, pages 437–441, 2013. (Cited on page 41.)

[Hu 2017] Qing Hu, Shichao Lv, Zhiqiang Shi, Limin Sun and Liang Xiao. Defense Against Advanced Persistent Threats with Expert System for Internet of Things. In International Conference on Wireless Algorithms, Systems, and Applications, pages 326–337. Springer, 2017. (Cited on pages 29 and 85.)

[Huang 1999] Ming-Yuh Huang, Robert J. Jasper and Thomas M. Wicks. A large scale distributed intrusion detection framework based on attack strategy analysis. Computer Networks, vol. 31, no. 23, pages 2465–2475, 1999. (Cited on pages 22 and 64.)

[Hui 2009] Xie Hui, Wu Min and Zhang Zhi-ming. Using Ant Colony Optimization to modeling the network vulnerability detection and restoration system. In International Conference on Industrial Mechatronics and Automation (ICIMA), pages 21–23, Chengdu, China, 2009. IEEE. (Cited on page 138.)

[Husák 2018] Martin Husák and Jaroslav Kašpar. Towards Predicting Cyber Attacks Using Information Exchange and Data Mining. In 14th International Wireless Communications & Mobile Computing Conference (IWCMC), pages 536–541. IEEE, 2018. (Cited on page 226.)

[Husmeier 2005] Dirk Husmeier. Introduction to learning Bayesian networks from data. In Probabilistic modeling in bioinformatics and medical informatics, pages 17–57. Springer, 2005. (Cited on pages 152 and 153.)

[Hutchins 2011] Eric M. Hutchins, Michael J. Cloppert and Rohan M. Amin. Intelligence-driven computer network defense informed by analysis of adversary campaigns and intrusion kill chains. In Proceedings of the 6th International Conference on Information Warfare and Security, pages 113–125, March 2011. (Cited on page 29.)

[Huurdeman 2003] Anton A. Huurdeman. The worldwide history of telecommunications. John Wiley & Sons, 2003. (Cited on page 12.)

[IETF 2017] Internet Engineering Task Force IETF. The JavaScript Object Notation (JSON) Data Interchange Format. RFC 8259, RFC Editor, December 2017. https://tools.ietf.org/html/rfc8259 (Date last accessed 28 Aug, 2018). (Cited on page 127.)

[Internet Security Systems 2001] Internet Security Systems. RealSecure Signatures Reference Guide Version 6.5, December 2001. (Cited on pages 32, 171 and 172.)

[ISO/IEC 2016] ISO/IEC. 27000:2016 Standard, 2016. (Cited on pages 13 and 240.)

[Jaccard 1901] Paul Jaccard. Étude de la distribution florale dans une portion des Alpes et du Jura. Bulletin de la Societé Vaudoise des Sciences Naturelles, vol. 37, no. 142, pages 547–579, 1901. (Cited on page 121.)

[Jaeger 2015] David Jaeger, Martin Ussath, Feng Cheng and Christoph Meinel. Multi-step Attack Pattern Detection on Normalized Event Logs. In IEEE 2nd International Conference on Cyber Security and Cloud Computing (CSCloud), pages 390–398. IEEE, 2015. (Cited on pages 39, 40, 86, 93 and 255.)

[Jain 2010] Anil K Jain. Data clustering: 50 years beyond K-means. Pattern recognition letters, vol. 31, pages 651–666, 2010. (Cited on page 75.)

[Jemili 2008] Farah Jemili, Montaceur Zaghdoud and Mohamed Ben Ahmed. Attack correlation and prediction system based on possibilistic networks. In IADIS'08 International Conference Applied Computing, 2008. (Cited on pages 43, 82 and 251.)

[Jemili 2009] Farah Jemili, Montaceur Zaghdoud and Mohamed Ben Ahmed. Attack Prediction based on Hybrid Propagation in Bayesian Networks. In Proc. of the Internet Technology And Secured Transactions Conference, ICITST-2009, 2009. (Cited on pages 82 and 251.)

[Jensen 2007] Finn V. Jensen and Thomas D. Nielsen. Bayesian networks and decision graphs. Springer Science & Business Media, 2007. (Cited on page 153.)

[Jeyepalan 2014] D. P. Jeyepalan and E. Kirubakaran. Agent Based Parallelized Intrusion Detection System using Ant Colony Optimization. International Journal of Computer Applications (IJCA), vol. 105, no. 10, 2014. (Cited on page 138.)

[Ji 2014] Shouling Ji, Weiqing Li, Mudhakar Srivatsa and Raheem Beyah. Structural data de-anonymization: Quantification, practice, and implications. In Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security, pages 1040–1053. ACM, 2014. (Cited on page 98.)

[Jia 2017] Shan-Shan Jia and Ya-Bin Xu. The APT detection method in SDN. In 3rd IEEE International Conference on Computer and Communications (ICCC), pages 1240–1245. IEEE, 2017. (Cited on pages 81 and 252.)

[Jiang 2017] Xiangkui Jiang, Chang-an Wu and Huaping Guo. Forest pruning based on branch importance. Computational Intelligence and Neuroscience, vol. 2017, 2017. (Cited on page 228.)

[Julisch 2001] Klaus Julisch. Mining alarm clusters to improve alarm handling efficiency. In Proceedings of the 17th Annual Computer Security Applications Conference (ACSAC), pages 12–21. IEEE, 2001. (Cited on pages 22, 70, 75, 92, 104, 227 and 249.)

[Julisch 2002] Klaus Julisch and Marc Dacier. Mining intrusion detection alarms for actionable knowledge. In Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining, pages 366–375. ACM, 2002. (Cited on pages 75, 92 and 249.)

[Julisch 2003a] Klaus Julisch. Clustering intrusion detection alarms to support root cause analysis. ACM Transactions on Information and System Security (TISSEC), vol. 6, no. 4, pages 443–471, 2003. (Cited on pages ix, 50, 51, 52, 75, 77, 92 and 249.)

[Julisch 2003b] Klaus Julisch. Using root cause analysis to handle intrusion detection alarms. Thesis, Universität Dortmund, 2003. (Cited on pages 75 and 249.)

[Kannadiga 2007] Pradeep Kannadiga, Mohammad Zulkernine and Anwar Haque. E-NIPS: An event-based network intrusion prediction system. Information Security, pages 37–52, 2007. (Cited on pages 86 and 254.)

[Katipally 2010] Rajeshwar Katipally, Wade Gasior, Xiaohui Cui and Li Yang. Multistage attack detection system for network administrators using data mining. In Proceedings of the Sixth Annual Workshop on Cyber Security and Information Intelligence Research, page 51, Oak Ridge, Tennessee, USA, 2010. ACM. (Cited on pages 86 and 254.)

[Katipally 2011] Rajeshwar Katipally, Li Yang and Anyi Liu. Attacker behavior analysis in multi-stage attack detection system. In Proceedings of the Seventh Annual Workshop on Cyber Security and Information Intelligence Research, page 63, Oak Ridge, Tennessee, USA, 2011. ACM. (Cited on pages 27, 81 and 251.)

[Kavanagh 2015] Kelly M. Kavanagh and Oliver Rochford. Magic Quadrant for Security Information and Event Management, 2015. (Cited on pages 19 and 219.)

[Kavousi 2012] Fatemeh Kavousi and Behzad Akbari. Automatic learning of attack behavior patterns using Bayesian networks. In Sixth International Symposium on Telecommunications (IST), pages 999–1004, 2012. (Cited on pages 51, 54, 74, 83 and 252.)

[Kavousi 2014] Fatemeh Kavousi and Behzad Akbari. A Bayesian network-based approach for learning attack strategies from intrusion alerts. Security and Communication Networks, vol. 7, pages 833–853, 2014. (Cited on pages 31, 43, 48, 51, 52, 54, 57, 59, 74, 83, 111, 170 and 252.)

[Kawakani 2016] Cláudio Toshio Kawakani, Sylvio Barbon Junior, Rodrigo Sanches Miani, Michel Cukier and Bruno Bogaz Zarpelão. Intrusion alert correlation to support security management. In XII Brazilian Symposium on Information Systems-Information Systems in the Cloud Computing Era, pages 313–320, 2016. (Cited on pages 52, 76 and 249.)

[Kawakani 2017] Cláudio Toshio Kawakani, Sylvio Barbon, Rodrigo Sanches Miani, Michel Cukier and Bruno Bogaz Zarpelão. Discovering Attackers Past Behavior to Generate Online Hyper-Alerts. iSys-Revista Brasileira de Sistemas de Informação, vol. 10, pages 122–147, 2017. (Cited on pages 22, 49, 52, 53, 76 and 249.)

[Kaynar 2017] Kerem Kaynar. Distributed log analysis for scenario-based detection of multi-step attacks and generation of near-optimal defense recommendations. PhD thesis, Technischen Universität Berlin, 2017. (Cited on pages 84 and 253.)

[Khakpour 2009] Narges Khakpour and Saeed Jalili. Using supervised and transductive learning techniques to extract network attack scenarios. In 14th International CSI Computer Conference (CSICC), pages 71–76, 2009. (Cited on pages 43, 73 and 248.)

[Khan 2016] Suleman Khan, Abdullah Gani, Ainuddin Wahid Abdul Wahab, Muhammad Shiraz and Iftikhar Ahmad. Network forensics: Review, taxonomy, and open challenges. Journal of Network and Computer Applications, vol. 66, pages 214–235, 2016. (Cited on page 3.)

[Kharraz 2015] Amin Kharraz, William Robertson, Davide Balzarotti, Leyla Bilge and Engin Kirda. Cutting the gordian knot: A look under the hood of ransomware attacks. In International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment, pages 3–24. Springer, 2015. (Cited on page 30.)

[Kholidy 2014a] Hisham A. Kholidy, Abdelkarim Erradi and Sherif Abdelwahed. Attack Prediction Models for Cloud Intrusion Detection Systems. In 2nd International Conference on Artificial Intelligence, Modelling and Simulation (AIMS), pages 270–275, 2014. (Cited on pages 82 and 252.)

[Kholidy 2014b] Hisham A. Kholidy, Abdelkarim Erradi, Sherif Abdelwahed and Abdulrahman Azab. A Finite State Hidden Markov Model for Predicting Multistage Attacks in Cloud Systems. In IEEE 12th International Conference on Dependable, Autonomic and Secure Computing (DASC), pages 14–19. IEEE, 2014. (Cited on pages 82 and 252.)

[Kholidy 2014c] Hisham A. Kholidy, Ahmed M. Yousof, Abdelkarim Erradi, Sherif Abdelwahed and Hisham Arafat Ali. A Finite Context Intrusion Prediction Model for Cloud Systems with a Probabilistic Suffix Tree. In European Modelling Symposium (EMS), pages 526–531, 2014. (Cited on pages 82 and 252.)

[Kim 2007] Jungwon Kim, Peter J. Bentley, Uwe Aickelin, Julie Greensmith, Gianni Tedesco and Jamie Twycross. Immune system approaches to intrusion detection – a review. Natural computing, vol. 6, pages 413–466, 2007. (Cited on page 75.)

[Kim 2014] Yong-Ho Kim and Won Hyung Park. A study on cyber threat prediction based on intrusion detection event for APT attack detection. Multimedia Tools and Applications, vol. 71, no. 2, pages 685–698, 2014. (Cited on pages 83 and 252.)

[King 2005] Samuel T. King, Zhuoqing Morley Mao, Dominic G. Lucchetti and Peter M. Chen. Enriching Intrusion Alerts Through Multi-Host Causality. In NDSS, 2005. (Cited on pages 74 and 248.)

[Kitchenham 2004] Barbara Kitchenham. Procedure for undertaking systematic reviews. Technical report, Keele University, 2004. (Cited on page 65.)

[Kobayashi 2014] Satoru Kobayashi, Kensuke Fukuda and Hiroshi Esaki. Towards an NLP-based log template generation algorithm for system log analysis. In Proceedings of The Ninth International Conference on Future Internet Technologies, page 11, Tokyo, Japan, 2014. ACM. (Cited on page 19.)

[Kolias 2011] Constantinos Kolias, Georgios Kambourakis and M. Maragoudakis. Swarm Intelligence in Intrusion Detection: A Survey. Comput. Secur., vol. 30, no. 8, pages 625–642, 2011. (Cited on page 138.)

[Koski 2011] Timo Koski and John Noble. Bayesian networks: an introduction, volume 924. John Wiley & Sons, 2011. (Cited on pages 81 and 152.)

[Kramer 2010] Simon Kramer and Julian C Bradfield. A general definition of malware. Journal in computer virology, vol. 6, no. 2, pages 105–114, 2010. (Cited on page 30.)

[Kruegel 2002] Christopher Kruegel, Thomas Toth and Clemens Kerer. Decentralized event correlation for intrusion detection. Information Security and Cryptology (ICISC), pages 59–95, 2002. (Cited on pages 43, 88 and 254.)

[Kruegel 2004] Christopher Kruegel, Fredrik Valeur and Giovanni Vigna. Intrusion detection and correlation: challenges and solutions, volume 14. Springer Science & Business Media, 2004. (Cited on page 57.)

[Kruse 2013] Rudolf Kruse, Christian Borgelt, Frank Klawonn, Christian Moewes, Matthias Steinbrecher and Pascal Held. Multi-layer perceptrons. In Computational Intelligence, pages 47–81. Springer, 2013. (Cited on page 60.)

[Kulkarni 2018] Pradeep Kulkarni, Sameer Patil, Prashant Kadam and Aniruddha Dolas. EternalBlue: A prominent threat actor of 2017-2018. Technical report, Quick Heal Security Labs, 06 2018. (Cited on page 30.)

[Lagzian 2012] Samira Lagzian, Fatemeh Amiri, AliReza Enayati and Hossien Gharaee. Frequent Item set mining-based Alert Correlation for Extracting multi-stage Attack Scenarios. In 6th International Symposium on Telecommunications (IST), pages 1010–1014. IEEE, 2012. (Cited on pages 83 and 252.)

[Lamprakis 2017] Pavlos Lamprakis, Ruggiero Dargenio, David Gugelmann, Vincent Lenders, Markus Happe and Laurent Vanbever. Unsupervised Detection of APT C&C Channels using Web Request Graphs. In International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment, pages 366–387. Springer, 2017. (Cited on page 66.)

[Lee 2008] Do-hyeon Lee, Doo-young Kim and Jae-il Jung. Multi-Stage Intrusion Detection System Using Hidden Markov Model Algorithm. In International Conference on Information Science and Security (ICISS), pages 72–77. IEEE, 2008. (Cited on pages 32, 81, 170 and 251.)

[Leek 2015] Jeffrey T Leek and Roger D Peng. Opinion: Reproducible research can still be wrong: Adopting a prevention approach. Proceedings of the National Academy of Sciences, vol. 112, pages 1645–1646, 2015. (Cited on page 96.)

[Legrand 2014] Véronique Legrand, Pierre Parrend, Pierre Collet, Stéphane Frénot and Marine Minier. Vers une architecture «big-data» bio-inspirée pour la détection d'anomalie des SIEM. In Cesar 2014: Detection et reaction face aux attaques informatiques, Rennes, France, 2014. (Cited on pages 101, 218 and 219.)

[Leiner 1997] Barry M. Leiner, Vinton G. Cerf, David D. Clark, Robert E. Kahn, Leonard Kleinrock, Daniel C. Lynch, Jon Postel, Lawrence G. Roberts and Stephen S. Wolff. The past and future history of the Internet. Communications of the ACM, vol. 40, no. 2, pages 102–108, 1997. (Cited on page 12.)

[Levenshtein 1966] Vladimir I Levenshtein. Binary codes capable of correcting deletions, insertions, and reversals. Soviet physics doklady, vol. 10, no. 8, pages 707–710, 1966. (Cited on pages 50 and 225.)

[Li 2007a] Yang Li and Li Guo. An active learning based TCM-KNN algorithm for supervised network intrusion detection. Computers & security, vol. 26, no. 7-8, pages 459–467, 2007. (Cited on page 101.)

[Li 2007b] Zhitang Li, Jie Lei, Li Wang and Dong Li. Assessing attack threat by the probability of following attacks. In International Conference on Networking, Architecture, and Storage (NAS), pages 91–100. IEEE, 2007. (Cited on pages 83, 93 and 251.)

[Li 2007c] Zhitang Li, Jie Lei, Li Wang and Dong Li. A data mining approach to generating network attack graph for intrusion prediction. In Fourth International Conference on Fuzzy Systems and Knowledge Discovery (FSKD), volume 4, pages 307–311. IEEE, 2007. (Cited on pages 83, 93 and 251.)

[Li 2007d] Zhitang Li, Aifang Zhang, Jie Lei and Li Wang. Real-time correlation of network security alerts. In IEEE International Conference on e-Business Engineering (ICEBE), pages 73–80. IEEE, 2007. (Cited on pages 43, 86, 89, 93 and 254.)

[Li 2007e] Zhitang Li, Aifang Zhang, Dong Li and Li Wang. Discovering Novel Multistage Attack Strategies. In 3rd International Conference on Advanced Data Mining and Applications, pages 45–56, Harbin, China, 2007. Springer. (Cited on pages 86, 93 and 254.)

[Li 2016] Yang Li, Ying Xue, Yuangang Yao, Xianghui Zhao, Jianyi Liu and Ru Zhang. An attack pattern mining algorithm based on fuzzy logic and sequence pattern. In 4th International Conference on Cloud Computing and Intelligence Systems (CCIS), pages 234–238, 2016. (Cited on pages 43, 46, 49, 50, 52, 53, 58, 59, 60, 75, 83 and 252.)

[Liao 2013] Hung-Jen Liao, Chun-Hung Richard Lin, Ying-Chih Lin and Kuang-Yuan Tung. Intrusion detection system: A comprehensive review. Journal of Network and Computer Applications, vol. 36, no. 1, pages 16–24, 2013. (Cited on page 20.)

[Lin 1998] Dekang Linet al. An information-theoretic definition of similarity. In Proceedings of the Fifteenth International Conference on Machine Learning (ICML), volume 98, pages 296–304, 1998. (Cited on pages 45 and 46.)

[Lin 2012] Jingqiang Lin, Peng Liu and Jiwu Jing. Using signaling games to model the multi-step attack-defense scenarios on confidentiality. In International Conference on Decision and Game Theory for Security, pages 118–137. Springer, 2012. (Cited on pages 56, 85, 99 and 253.)

[Liu 2008] Zhijie Liu, Chongjun Wang and Shifu Chen. Correlating Multi-Step Attack and Constructing Attack Scenarios Based on Attack Pattern Modeling. In International Conference on Information Security and Assurance (ISA), pages 214–219. IEEE, 2008. (Cited on pages 56, 87, 111, 170 and 254.)

[Long 2008] Jidong Long and Daniel G. Schwartz. Case-oriented alert correlation. W. Trans. on Comp, vol. 7, pages 98–112, 2008. (Cited on pages 86 and 254.)

[Lu 2017] Jiazhong Lu, Kai Chen, Zhongliu Zhuo and XiaoSong Zhang. A temporal correlation and traffic analysis approach for APT attacks detection. Cluster Computing, pages 1–12, 2017. (Cited on page 66.)

[Lu 2018] Xindai Lu, Jiajia Han, Qianbo Ren, Hua Dai, Jiyuan Li and Jing Ou. Network threat detection based on correlation analysis of multi-platform multi-source alert data. Multimedia Tools and Applications, pages 1–15, 2018. (Cited on pages 83 and 252.)

[Luh 2016] Robert Luh, Stefan Marschalek, Manfred Kaiser, Helge Janicke and Sebastian Schrittwieser. Semantics-aware detection of targeted attacks: a survey. Journal of Computer Virology and Hacking Techniques, vol. 13, pages 47–85, 2016. (Cited on pages 29 and 65.)

[Luktarhan 2012] Nurbol Luktarhan, Xue Jia, Liang Hu and Nannan Xie. Multi-stage attack detection algorithm based on Hidden Markov Model. In International Conference on Web Information Systems and Mining, pages 275–282, Chengdu, China, 2012. Springer. (Cited on pages 82 and 252.)

[Luo 2014] Yi Luo, Ferenc Szidarovszky, Youssif Al-Nashif and Salim Hariri. A fictitious play-based response strategy for multistage intrusion defense systems. Security and Communication Networks, vol. 7, no. 3, pages 473–491, 2014. (Cited on pages 43, 56, 85, 99 and 253.)

[Luo 2016] Shibo Luo, Jun Wu, Jianhua Li and Longhua Guo. A Multi-stage Attack Mitigation Mechanism for Software-defined Home Networks. IEEE Transactions on Consumer Electronics, vol. 62, no. 2, pages 200–207, 2016. (Cited on pages 43, 56, 84 and 253.)

[Lv 2015] Yanli Lv, Shuang Xiang, Jingxin Geng, Yuanlong Li and Chunhe Xia. An alert correlation algorithm based on the sequence pattern mining. In Advanced Information Technology, Electronic and Automation Control Conference (IAEAC), pages 1146–1151, 2015. (Cited on pages 83 and 252.)

[Ma 2008] Jie Ma, Zhitang Li and Wei-ming Li. Real-time alert stream clustering and correlation for discovering attack strategies. In Fifth International Conference on Fuzzy Systems and Knowledge Discovery (FSKD), volume 4, pages 379–384. IEEE, 2008. (Cited on pages 83, 93 and 251.)

[Mahanti 2005] P. Mahanti, Mohammed Al-Fayoumi and Soumya Banerjee. Simulating Targeted Attacks Using Research Honeypots Based on Ant Colony Metaphor. Eur. J. Sci. Res., vol. 17, no. 4, pages 509–522, 2005. (Cited on page 138.)

[Majeed 2018] Abdul Majeed, Raihan ur Rasool, Farooq Ahmad, Masoom Alam and Nadeem Javaid. Near-miss situation based visual analysis of SIEM rules for real time network security monitoring. Journal of Ambient Intelligence and Humanized Computing, pages 1–18, 2018. (Cited on page 99.)

[Man 2012] Da-peng Man, Xue-zhen Li, Wu Yang, Wei Wang and Shi-chang Xuan. A Multi-step Attack Recognition and Prediction Method Via Mining Attacks Conversion Frequencies. International Journal of Wireless and Microwave Technologies (IJWMT), vol. 2, no. 2, page 20, 2012. (Cited on pages 83 and 252.)

[Manganiello 2011] Fabio Manganiello, Mirco Marchetti and Michele Colajanni. Multistep attack detection and alert correlation in intrusion detection systems. In International Conference on Information Security and Assurance, pages 101–110. Springer, 2011. (Cited on pages 77, 89 and 249.)

[Mao 2009] Ching-Hao Mao, Hahn-Ming Lee, Devi Parikh, Tsuhan Chen and Si-Yu Huang. Semi-supervised co-training and active learning based approach for multi-view intrusion detection. In Proceedings of the 2009 ACM symposium on Applied Computing, pages 2042–2048. ACM, 2009. (Cited on page 101.)

[Marchetti 2011a] Mirco Marchetti, Michele Colajanni and Fabio Manganiello. Framework and models for multistep attack detection. International Journal of Security and Its Applications, vol. 5, pages 73–90, 2011. (Cited on pages 89 and 256.)

[Marchetti 2011b] Mirco Marchetti, Michele Colajanni and Fabio Manganiello. Identification of correlated network intrusion alerts. In Third International Workshop on Cyberspace Safety and Security (CSS), pages 15–20, 2011. (Cited on pages 55, 83, 89 and 251.)

[Marill 1966] Thomas Marill and Lawrence G. Roberts. Toward a cooperative network of time-shared computers. In Proceedings of the November 7-10, 1966, fall joint computer conference, pages 425–431. ACM, 1966. (Cited on page 12.)

[Mathew 2005] Sunu Mathew, Chintan Shah and Shambhu Upadhyaya. An Alert Fusion Framework for Situation Awareness of Coordinated Multistage Attacks. In Third IEEE International Workshop on Information Assurance (IWIA'05), pages 95–104. IEEE, 2005. (Cited on pages 43, 86 and 254.)

[Mathew 2009] Sunu Mathew and Shambhu Upadhyaya. Attack scenario recognition through heterogeneous event stream analysis. In IEEE Military Communications Conference (MILCOM), pages 1–7. IEEE, 2009. (Cited on pages 22, 55, 57, 77, 227 and 249.)

[Mathew 2010] Sunu Mathew, Shambhu Upadhyaya, Moises Sudit and Adam Stotz. Situation Awareness of Multistage Cyber Attacks by Semantic Event Fusion. In IEEE Military Communications Conference (MILCOM), pages 1286–1291. IEEE, 2010. (Cited on pages 56, 87, 89, 93, 104, 105 and 254.)

[McDermott 2001] James P McDermott. Attack net penetration testing. In Proceedings of the 2000 workshop on New security paradigms, pages 15–21. ACM, 2001. (Cited on page 40.)

[McElwee 2017] Steven McElwee. Active learning intrusion detection using k-means clustering selection. In SoutheastCon, pages 1–7. IEEE, 2017. (Cited on page 100.)

[Meier 2004] Michael Meier. A model for the semantics of attack signatures in misuse detection systems. In International Conference on Information Security, pages 158–169. Springer, 2004. (Cited on page 19.)

[Meier 2007] Michael Meier. Intrusion detection effektiv!: Modellierung und analyse von angriffsmustern. Springer-Verlag, 2007. (Cited on pages 39 and 88.)

[Meline 2006] Timothy Meline. Selecting studies for systematic review: Inclusion and exclusion criteria. Contemporary issues in communication science and disorders, vol. 33, 2006. (Cited on page 69.)

[Mell 2007] Peter Mell, Karen Scarfone and Sasha Romanosky. A complete guide to the Common Vulnerability Scoring System version 2.0. FIRST-Forum of Incident Response and Security Teams, 2007. (Cited on page 207.)

[Metzler 2007] Donald Metzler, Susan Dumais and Christopher Meek. Similarity measures for short segments of text. In European conference on information retrieval, pages 16–27. Springer, 2007. (Cited on pages 50 and 225.)

[Micheel 2001] Jörg Micheel, Stephen Donnelly and Ian Graham. Precision timestamping of network packets. In Proceedings of the 1st ACM SIGCOMM Workshop on Internet Measurement, pages 273–277. ACM, 2001. (Cited on page 36.)

[Michel 2002] Cédric Michel and Ludovic Mé. ADeLe: an attack description language for knowledge-based intrusion detection. In Trusted Information, pages 353–368. Springer, 2002. (Cited on page 41.)

[Mills 1991] David L Mills. Internet time synchronization: the network time protocol. IEEE Transactions on communications, vol. 39, no. 10, pages 1482–1493, 1991. (Cited on page 36.)

[Mills 2016] David L Mills. Computer network time synchronization: the network time protocol on earth and in space. CRC Press, 2016. (Cited on page 36.)

[MIT Lincoln Laboratory 2018] MIT Lincoln Laboratory. 2000 DARPA Intrusion Detection Scenario Specific Data Sets, 2018. https://ithandbook.ffiec.gov/glossary.aspx (Date last accessed 28 Aug, 2018). (Cited on page 170.)

[Mockapetris 1983] Paul Mockapetris. Domain names - Implementation and specification. RFC 883, RFC Editor, November 1983. https://tools.ietf.org/html/rfc792 (Date last accessed 28 Aug, 2018). (Cited on page 32.)

[Mohurle 2017] Savita Mohurle and Manisha Patil. A brief study of Wannacry Threat: Ransomware Attack 2017. International Journal, vol. 8, 2017. (Cited on pages 22 and 29.)

[Morin 2002] Benjamin Morin, Ludovic Mé, Hervé Debar and Mireille Ducassé. M2D2: A formal data model for IDS alert correlation. In International Workshop on Recent Advances in Intrusion Detection, pages 115–137, 2002. (Cited on page 41.)

[Morin 2003] Benjamin Morin and Hervé Debar. Correlation of intrusion symptoms: an application of chronicles. In International Workshop on Recent Advances in Intrusion Detection, pages 94–112. Springer, 2003. (Cited on pages 41, 86, 92 and 254.)

[Müller 2009] Andreas Müller. Event Correlation Engine. Master thesis, Eidgenössische Technische Hochschule Zürich, 2009. (Cited on page 110.)

[Murphy 2009] Christopher T. Murphy. CACTUSS: Clustering of attack tracks using significant services, 2009. (Cited on pages 76 and 249.)

[Murphy 2010] Christopher T. Murphy and Shanchieh J. Yang. Clustering of multistage cyber attacks using significant services. In 13th International Conference on Information Fusion (FUSION), pages 1–7. IEEE, 2010. (Cited on pages 76, 93 and 249.)

[Nagpal 2017] Bharti Nagpal, Naresh Chauhan and Nanhay Singh. A Survey on the Detection of SQL Injection Attacks and Their Countermeasures. JIPS (Journal of Information Processing Systems), vol. 13, no. 4, pages 689–702, 2017. (Cited on page 20.)

[Narayanan 2008] Arvind Narayanan and Vitaly Shmatikov. Robust de-anonymization of large sparse datasets. In IEEE Symposium on Security and Privacy, pages 111–125. IEEE, 2008. (Cited on page 98.)

[Nassar 2013] Mohamed Nassar, Bechara al Bouna and Qutaibah M Malluhi. Secure Outsourcing of Network Flow Data Analysis. In BigData Congress, pages 431–432, 2013. (Cited on page 108.)

[Nath 2014] Hiran V Nath and Babu M Mehtre. Static malware analysis using machine learning methods. In International Conference on Security in Computer Networks and Distributed Systems, pages 440–450. Springer, 2014. (Cited on page 66.)

[Nathoo 2005] Karim Nathoo. A Case Study in Solaris Sadmind Exploitation. Sans Institute, January 2005. (Cited on page 32.)

[Navarro 2001] Gonzalo Navarro. A guided tour to approximate string matching. ACM computing surveys (CSUR), vol. 33, no. 1, pages 31–88, 2001. (Cited on page 50.)

[Navarro 2016] Julio Navarro, Aline Deruyver and Pierre Parrend. Morwilog: an ACO-based system for outlining multi-step attacks. In 2016 IEEE Symposium Series on Computational Intelligence (SSCI), pages 1–8. IEEE, 2016. (Cited on pages 136, 230 and 255.)

[Navarro 2017] Julio Navarro, Véronique Legrand, Sofiane Lagraa, Jérôme François, Abdelkader Lahmadi, Giulia De Santis, Olivier Festor, Nadira Lammari, Fayçal Hamdi, Aline Deruyveret al. HuMa: A multi-layer framework for threat analysis in a heterogeneous log environment. In International Symposium on Foundations and Practice of Security, pages 144–159. Springer, 2017. (Cited on pages xi, 101, 203, 206, 207 and 218.)

[Navarro 2018a] Julio Navarro, Aline Deruyver and Pierre Parrend. A systematic survey on multi-step attack detection. Computers & Security, vol. 76, pages 214–249, July 2018. (Cited on pages 63, 65, 67 and 101.)

[Navarro 2018b] Julio Navarro, Véronique Legrand, Aline Deruyver and Pierre Parrend. OMMA: open architecture for Operator-guided Monitoring of Multi-step Attacks. EURASIP Journal on Information Security, vol. 2018, no. 1, page 6, 2018. (Cited on pages 136, 218 and 231.)

[Ning 2002a] Peng Ning and Yun Cui. An Intrusion Alert Correlator Based on Prerequisites of Intrusions. Report, North Carolina State University, 2002. (Cited on pages 79, 93, 125, 171 and 250.)

[Ning 2002b] Peng Ning, Yun Cui and Douglas S. Reeves. Analyzing intensive intrusion alerts via correlation. In International Workshop on Recent Advances in Intrusion Detection, pages 74–94. Springer, 2002. (Cited on pages 79, 92, 93 and 250.)

[Ning 2002c] Peng Ning, Yun Cui and Douglas S. Reeves. Constructing attack scenarios through correlation of intrusion alerts. In Proceedings of the 9th ACM Conference on Computer and Communications Security, pages 245–254. ACM, 2002. (Cited on pages 43, 79, 80, 92, 93 and 250.)

[Ning 2002d] Peng Ning and Dingbang Xu. Adapting query optimization techniques for efficient intrusion alert correlation. In Proceedings of the 17th IFIP WG, volume 11, 2002. (Cited on page 79.)

[Ning 2003a] Peng Ning, Pai Peng, Yiquan Hu and Dingbang Xu. TIAA: A visual toolkit for intrusion alert analysis. North Carolina State University. Center for Advanced Computing and Communication, 2003. (Cited on page 79.)

[Ning 2003b] Peng Ning and Dingbang Xu. Learning attack strategies from intrusion alerts. In Proceedings of the 10th ACM conference on Computer and communications security, pages 200–209. ACM, 2003. (Cited on pages 43, 79, 92, 93 and 250.)

[Ning 2004a] Peng Ning, Yun Cui, Douglas S. Reeves and Dingbang Xu. Techniques and tools for analyzing intrusion alerts. ACM Transactions on Information and System Security (TISSEC), vol. 7, no. 2, pages 274–318, 2004. (Cited on pages 38, 79, 92, 93, 111, 170 and 250.)

[Ning 2004b] Peng Ning and Dingbang Xu. Hypothesizing and reasoning about attacks missed by intrusion detection systems. ACM Transactions on Information and System Security (TISSEC), vol. 7, no. 4, pages 591–627, 2004. (Cited on pages 88, 93, 105 and 256.)

[Ning 2004c] Peng Ning, Dingbang Xu, Christopher G. Healey and Robert St Amant. Building Attack Scenarios through Integration of Complementary Alert Correlation Method. In NDSS, volume 4, pages 97–111, 2004. (Cited on pages 88, 92, 93 and 256.)

[Ning 2007] Zhuo Ning and Jian Gong. An Intrusion Plan Recognition Algorithm Based on Max-1-Connected Causal Networks. In International Conference on Computational Science, pages 809–816, 2007. (Cited on pages 82 and 251.)

[Ning 2010] Peng Ning and Dingbang Xu. Toward automated intrusion alert analysis, pages 175–205. Springer, 2010. (Cited on pages 36, 55, 79, 93 and 250.)

[Noel 2004] Steven Noel, Eric Robertson and Sushil Jajodia. Correlating intrusion events and building attack scenarios through attack graph distances. In 20th Annual Computer Security Applications Conference, pages 350–359. IEEE, 2004. (Cited on pages 84 and 253.)

[OED 2018a] Oxford English Dictionary Online. OED. "correlation", 2018. https://en.oxforddictionaries.com/definition/correlation (Date last accessed 13 Sept, 2018). (Cited on page 57.)

[OED 2018b] Oxford English Dictionary Online. OED. "cyberspace", 2018. https://en.oxforddictionaries.com/definition/cyberspace (Date last accessed 5 Jun, 2018). (Cited on pages 13 and 240.)

[OED 2018c] Oxford English Dictionary Online. OED. "event", 2018. https://en.oxforddictionaries.com/definition/event (Date last accessed 31 Dec, 2018). (Cited on pages 17 and 19.)

[OED 2018d] Oxford English Dictionary Online. OED. "similar", 2018. https://en.oxforddictionaries.com/definition/similar (Date last accessed 11 Sept, 2018). (Cited on page 45.)

[OED 2018e] Oxford English Dictionary Online. OED. "similarity", 2018. https://en.oxforddictionaries.com/definition/similarity (Date last accessed 11 Sept, 2018). (Cited on page 45.)

[OED 2018f] Oxford English Dictionary Online. OED. "trace", 2018. https://en.oxforddictionaries.com/definition/trace (Date last accessed 30 Jul, 2018). (Cited on page 17.)

[Offroy 2016] Marc Offroy and Ludovic Duponchel. Topological data analysis: A promising big data exploration tool in biology, analytical chemistry and physical chemistry. Analytica Chimica Acta, vol. 910, pages 1 – 11, 2016. (Cited on page 208.)

[Oliver Rochford 2016] Toby Bussa Oliver Rochford Kelly M. Kavanagh. Critical Capabilities for Security Information and Event Management. Technical report, Gartner, 2016. (Cited on page 110.)

[Ossenbühl 2015] Sven Ossenbühl, Jessica Steinberger and Harald Baier. Towards automated incident handling: How to select an appropriate response against a network-based attack? In Ninth International Conference on IT Security Incident Management & IT Forensics (IMF), pages 51–67. IEEE, 2015. (Cited on page 221.)

[Ourston 2003] Dirk Ourston, Sara Matzner, William Stump and Bryan Hopkins. Applications of Hidden Markov Models to detecting multi-stage network attacks. In Proceedings of the 36th Annual Hawaii International Conference on System Sciences, page 10 pp. IEEE, 2003. (Cited on pages 81 and 251.)

[Panda 2017] Panda. #WannaCry Report. Technical report, Panda, 05 2017. (Cited on page 30.)

[PandaLabs 2017] PandaLabs. Annual Report, 2017. (Cited on page 23.)

[Pandey 2008] Navneet Kumar Pandey, Sudhir K. Gupta, Shaveta Leekha and Jingmin Zhou. ACML: Capability based attack modeling language. In Fourth International Conference on Information Assurance and Security (ISIAS), pages 147–154. IEEE, 2008. (Cited on pages 38, 55, 80 and 250.)

[Panichprecha 2007] Sorot Panichprecha, George Mohay and Andrew Clark. Multi-step scenario matching based on unification. In Australian Digital Forensics Conference. School of Computer and Information Science, Edith Cowan University, Perth, Western Australia, 2007. (Cited on pages 86 and 254.)

[Pei 2016] Kexin Pei, Zhongshu Gu, Brendan Saltaformaggio, Shiqing Ma, Fei Wang, Zhiwei Zhang, Luo Si, Xiangyu Zhang and Dongyan Xu. HERCULE: attack story reconstruction via community discovery on correlated log graph. In Proceedings of the 32nd Annual Conference on Computer Security Applications, pages 583–595, 2016. (Cited on pages xiii, 22, 23, 43, 46, 47, 48, 49, 52, 53, 57, 60, 74, 245 and 248.)

[Peng 2007] Tao Peng, Christopher Leckie and Kotagiri Ramamohanarao. Survey of network-based defense mechanisms countering the DoS and DDoS problems. ACM Computing Surveys (CSUR), vol. 39, no. 1, page 3, 2007. (Cited on page 20.)

[Peng 2011] Roger D Peng. Reproducible research in computational science. Science, vol. 334, pages 1226–1227, 2011. (Cited on page 96.)

[Peterson 1981] James L Peterson. Petri net theory and the modeling of systems. Prentice Hall PTR, 1981. (Cited on page 39.)

[Pokorny 2007] Julius Pokorny. Proto-Indo-European etymological dictionary. A Revised Edition of Julius Pokorny's Indogermanisches Etymologisches Wörterbuch. Indo-European Language Revival Association, 2007. (Cited on page 139.)

[Porras 1997] Phillip A Porras and Peter G Neumann. EMERALD: Event monitoring enabling response to anomalous live disturbances. In Proceedings of the 20th national information systems security conference, pages 353–365, 1997. (Cited on pages 73 and 80.)

[Qian 2008] Yi Qian, James Joshi, David Tipper and Prashant Krishnamurthy. Information assurance: dependability and security in networked systems. Morgan Kaufmann, 2008. (Cited on pages 16 and 21.)

[Qiao 2012] Lin-Bo Qiao, Bo-Feng Zhang, Zhi-Quan Lai and Jin-Shu Su. Mining of attack models in IDS alerts from network backbone by a two-stage clustering method. In IEEE 26th International Parallel and Distributed Processing Symposium Workshops & PhD Forum (IPDPSW), pages 1263–1269. IEEE, 2012. (Cited on pages 43, 46, 48, 49, 52, 54, 58, 60, 76 and 249.)

[Qin 2003] Xinzhou Qin and Wenke Lee. Statistical causality analysis of INFOSEC alert data. In International Workshop on Recent Advances in Intrusion Detection, pages 73–93. Springer, 2003. (Cited on pages 80, 82, 92 and 251.)

[Qin 2004] Xinzhou Qin and Wenke Lee. Attack Plan Recognition and Prediction Using Causal Networks. In 20th Annual Computer Security Applications Conference, pages 370–379. IEEE, 2004. (Cited on pages 22, 82, 92 and 251.)

[Qin 2005] Xinzhou Qin. A probabilistic-based framework for INFOSEC alert correlation. Thesis, Georgia Institute of Technology, 2005. (Cited on pages 82 and 251.)

[Qin 2007] Xinzhou Qin and Wenke Lee. Discovering novel attack strategies from INFOSEC alerts. In Data Warehousing and Data Mining Techniques for Cyber Security, pages 109–157. Springer, 2007. (Cited on pages 82, 154 and 251.)

[Ramaki 2015a] Ali Ahmadian Ramaki, Morteza Amini and Reza Ebrahimi Atani. RTECA: Real time episode correlation algorithm for multi-step attack scenarios detection. Computers & Security, vol. 49, pages 206–219, 2015. (Cited on pages 88, 111, 171 and 256.)

[Ramaki 2015b] Ali Ahmadian Ramaki, Masoud Khosravi-Farmad and Abbas Ghaemi Bafghi. Real time alert correlation and prediction using Bayesian networks. In 12th International Iranian Society of Cryptology Conference on Information Security and Cryptology (ISCISC), pages 98–103, 2015. (Cited on pages 88 and 256.)

[Ramaki 2016] Ali Ahmadian Ramaki and Abbas Rasoolzadegan. Causal knowledge analysis for detecting and modeling multi-step attacks. Security and Communication Networks, vol. 9, pages 6042–6065, 2016. (Cited on pages 31, 32, 43, 88, 170, 171, 172, 187 and 256.)

[Rass 2017] Stefan Rass, Sandra König and Stefan Schauer. Defending Against Advanced Persistent Threats Using Game-Theory. PloS one, vol. 12, no. 1, page e0168675, 2017. (Cited on page 85.)

[Rauf 2018] Usman Rauf. A Taxonomy of Bio-Inspired Cyber Security Approaches: Existing Techniques and Future Directions. Arabian Journal for Science and Engineering, pages 1–16, 2018. (Cited on page 138.)

[Ren 2010] Hanli Ren, Natalia Stakhanova and Ali A. Ghorbani. An online adaptive approach to alert correlation. In International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment, pages 153–172. Springer, 2010. (Cited on pages 50, 55, 82, 93, 104 and 251.)

[Rhodes-Ousley 2013] Mark Rhodes-Ousley. Information security: the complete reference. McGraw Hill Education, 2013. (Cited on page 21.)

[Roschke 2011] Sebastian Roschke, Feng Cheng and Christoph Meinel. A new alert correlation algorithm based on attack graph. In Computational intelligence in security for information systems, pages 58–67. Springer, 2011. (Cited on pages 56, 84, 93 and 253.)

[Roth 1995] Martha T. Roth, Harry A. Hoffner and Piotr Michalowski. Law collections from Mesopotamia and Asia minor. Scholars Press Atlanta, 1995. (Cited on page 12.)

[Saad 2012] Sherif Saad and Issa Traore. Extracting attack scenarios using intrusion semantics. In International Symposium on Foundations and Practice of Security, pages 278–292. Springer, 2012. (Cited on pages 43, 48, 52, 59, 89 and 256.)

[Saad 2014] Sherif Saad. Intrusion Alert Analysis Framework Using Semantic Correlation. Thesis, University of Victoria, 2014. (Cited on pages 46, 48, 57, 59, 89, 171, 172 and 256.)

[Sadoddin 2009] Reza Sadoddin and Ali A. Ghorbani. An incremental frequent structure mining framework for real-time alert correlation. Computers & Security, vol. 28, no. 3, pages 153–173, 2009. (Cited on pages 83, 92, 93, 111, 171 and 251.)

[Saikia 2018] Manaswita Saikia, Nazrul Hoque and Dhruba Kumar Bhattacharyya. MaNaDAC: An Effective Alert Correlation Method. In Recent Developments in Machine Learning and Data Analytics, pages 249–260. Springer, 2018. (Cited on pages 82, 187 and 252.)

[Salah 2013] Saeed Salah, Gabriel Maciá-Fernández and Jesús E. Díaz-Verdejo. A model-based survey of alert correlation techniques. Computer Networks, vol. 57, no. 5, pages 1289–1317, 2013. (Cited on pages 20, 70, 78, 85 and 239.)

[Salama 2013] Khalid M Salama and Alex A Freitas. Learning Bayesian network classifiers using ant colony optimization. Swarm Intelligence, vol. 7, no. 2-3, pages 229–254, 2013. (Cited on page 155.)

[Sánchez 2003] César Sánchez, Sriram Sankaranarayanan, Henny Sipma, Ting Zhang, David Dill and Zohar Manna. Event correlation: Language and semantics. In International Workshop on Embedded Software, pages 323–339. Springer, 2003. (Cited on page 41.)

[Schindler 2017] Timo Schindler. Anomaly Detection in Log Data using Graph Databases and Machine Learning to Defend Advanced Persistent Threats. In Gesellschaft für Informatik e.V. (Hrsg.): Informatik 2017, pages 2371–2378, September 2017. (Cited on pages 86 and 255.)

[Schwartz 2010] Edward J Schwartz, Thanassis Avgerinos and David Brumley. All you ever wanted to know about dynamic taint analysis and forward symbolic execution (but might have been afraid to ask). In IEEE symposium on Security and privacy (SP), pages 317–331. IEEE, 2010. (Cited on page 39.)

[Shaneck 2006] Mark Shaneck, Varun Chandola, Haiyang Liu, Changho Choi, György Simon, Eric Eilertson, Yongdae Kim, Zhi-Li Zhang, Jaideep Srivastava and Vipin Kumar. A Multi-Step Framework for Detecting Attack Scenarios. Report, University of Minnesota, 2006. (Cited on pages ix, 71, 74, 99, 227 and 248.)

[Shawly 2018] Tawfeeq Shawly, Ali Elghariani, Jason Kobes and Arif Ghafoor. Architectures for Detecting Real-time Multiple Multi-stage Network Attacks Using Hidden Markov Model. arXiv preprint arXiv:1807.09764, 2018. (Cited on pages 82 and 252.)

[Sheyner 2002] Oleg Sheyner, Joshua Haines, Somesh Jha, Richard Lippmann and Jeannette M Wing. Automated generation and analysis of attack graphs. In IEEE Symposium on Security and privacy, pages 273–284, 2002. (Cited on pages 56 and 105.)

[Shin 2013] Seongjun Shin, Seungmin Lee, Hyunwoo Kim and Sehun Kim. Advanced probabilistic approach for network intrusion forecasting and detection. Expert Systems with Applications, vol. 40, no. 1, pages 315–322, 2013. (Cited on pages 42, 43, 78 and 249.)

[Shiravi 2012] Ali Shiravi, Hadi Shiravi, Mahbod Tavallaee and Ali A. Ghorbani. Toward developing a systematic approach to generate benchmark datasets for intrusion detection. Computers & Security, vol. 31, pages 357–374, 2012. (Cited on pages 34 and 172.)

[Shittu 2016] Riyanat O. Shittu. Mining intrusion detection alert logs to minimise false positives & gain attack insight. Thesis, City University London, 2016. (Cited on pages 32, 33, 43, 48, 49, 52, 53, 58, 59, 88, 125, 170 and 256.)

[Shostack 2014] Adam Shostack. Threat modeling: Designing for security. John Wiley & Sons, 2014. (Cited on pages 56 and 105.)

[Sicilia 2017] Miguel-Angel Sicilia, Javier Bermejo-Higuera, Elena García-Barriocanal, Salvador Sánchez-Alonso, Daniel Domínguez-Álvarez and Miguel Monzón-Fernández. Querying streams of alerts for knowledge-based detection of long-lived network intrusions. In International Conference on Flexible Query Answering Systems, pages 186–197. Springer, 2017. (Cited on pages 84 and 253.)

[Singhal 2017] Anoop Singhal and Xinming Ou. Security risk analysis of enterprise networks using probabilistic attack graphs. In Network Security Metrics, pages 53–73. Springer, 2017. (Cited on page 105.)

[Skopik 2014] Florian Skopik, Ivo Friedberg and Roman Fiedler. Dealing with Advanced Persistent Threats in Smart Grid ICT Networks. In IEEE Innovative Smart Grid Technologies Conference (ISGT), pages 1–5. IEEE, 2014. (Cited on pages 77, 104, 227 and 249.)

[Slagell 2005] Adam Slagell and William Yurcik. Sharing computer network logs for security and privacy: A motivation for new methodologies of anonymization. In Workshop of the 1st International Conference on Security and Privacy for Emerging Areas in Communication Networks, pages 80–89, 2005. (Cited on page 98.)

[Soleimani 2008] Mahbobeh Soleimani and Ali A. Ghorbani. Critical episode mining in intrusion detection alerts. In 6th Annual Communication Networks and Services Research Conference (CNSR), pages 157–164, 2008. (Cited on pages 53, 85, 93 and 254.)

[Soleimani 2012] Mahbobeh Soleimani and Ali A. Ghorbani. Multi-layer episode filtering for the multi-step attack detection. Computer Communications, vol. 35, no. 11, pages 1368–1379, 2012. (Cited on pages 57, 83, 92, 93 and 252.)

[Stallings 2015] William Stallings and Lawrie Brown. Computer security: principles and practice. Pearson Education, 2015. (Cited on pages 14, 16, 239 and 241.)

[Stephenson 2000] Todd A. Stephenson. An introduction to Bayesian network theory and usage. Technical report, IDIAP, 2000. (Cited on pages 152, 153 and 154.)

[Stotz 2007] Adam Stotz and Moises Sudit. INformation Fusion Engine for Real-time Decision-making (INFERD): A perceptual system for cyber attack tracking. In 10th International Conference on Information Fusion, pages 1–8. IEEE, 2007. (Cited on pages 43, 87, 89, 93 and 254.)

[Strayer 2005] W. Timothy Strayer, Christine E. Jones, Beverly I. Schwartz, Joanne Mikkelson and Carl Livadas. Architecture for multi-stage network attack traceback. In The IEEE Conference on Local Computer Networks 30th Anniversary (LCN'05), pages 8 pp.–785. IEEE, 2005. (Cited on pages 21, 74 and 248.)

[Sudit 2005] Moises Sudit, Adam Stotz and Michael Holender. Situational awareness of a coordinated cyber attack. In Defense and Security, pages 114–129. International Society for Optics and Photonics, 2005. (Cited on pages 87, 89, 93, 227 and 254.)

[Sun 2018] Xiaoyan Sun, Jun Dai, Peng Liu, Anoop Singhal and John Yen. Using Bayesian Networks for Probabilistic Identification of Zero-Day Attack Paths. IEEE Transactions on Information Forensics and Security, vol. 13, no. 10, pages 2506–2521, 2018. (Cited on pages 83 and 252.)

[Sundaramurthy 2015] Sathya Chandran Sundaramurthy, Alexandru G. Bardas, Jacob Case, Xinming Ou, Michael Wesch, John McHugh and S. Raj Rajagopalan. A Human Capital Model for Mitigating Security Analyst Burnout. In Symposium on Usable Privacy and Security (SOUPS), pages 347–359, 2015. (Cited on page 2.)

[Tabia 2011] Karim Tabia, Salem Benferhat, Philippe Leray and Ludovic Mé. Alert correlation in intrusion detection: Combining AI-based approaches for exploiting security operators' knowledge and preferences. Security and artificial intelligence (SecArt), page NC, 2011. (Cited on page 41.)

[Templeton 2001] Steven J. Templeton and Karl Levitt. A requires/provides model for computer attacks. In Proceedings of the 2000 workshop on New security paradigms, pages 31–38. ACM, 2001. (Cited on pages 37 and 55.)

[Theraulaz 1999] Guy Theraulaz and Eric Bonabeau. A Brief History of Stigmergy. Artif. Life, vol. 5, no. 2, pages 97–116, 1999. (Cited on page 137.)

[Thomas 2018] Lindsey Thomas, Adam Vaughan, Zachary Courtney, Chen Zhong and Awny Alnusair. Supporting Collaboration Among Cyber Security Analysts Through Visualizing Their Analytical Reasoning Processes. In IEEE International Conference on Multimedia & Expo Workshops (ICMEW), pages 1–6. IEEE, 2018. (Cited on page 107.)

[Thulasiraman 2011] Krishnaiyan Thulasiraman and Madisetti NS Swamy. Graphs: theory and algorithms. John Wiley & Sons, 2011. (Cited on page 42.)

[Tian 2017] Jian-wei Tian, Xi Li, Zhen Tian and Wei-hui Qi. Network attack path reconstruction based on similarity computation. In 13th International Conference on Natural Computation, Fuzzy Systems and Knowledge Discovery (ICNC-FSKD), pages 2457–2461. IEEE, 2017. (Cited on pages 73 and 248.)

[Trustwave 2018] Trustwave. Global security report. Technical report, Trustwave, 2018. (Cited on pages 22 and 23.)

[Turnbull 2013] James Turnbull. The logstash book. Autopublishe, 2013. (Cited on page 229.)

[Ussath 2016a] Martin Ussath, Feng Cheng and Christoph Meinel. Automatic multi-step signature derivation from taint graphs. In IEEE Symposium Series on Computational Intelligence (SSCI), pages 1–8. IEEE, 2016. (Cited on pages 2, 40, 86, 93 and 255.)

[Ussath 2016b] Martin Ussath, Feng Cheng and Christoph Meinel. Event attribute tainting: A new approach for attack tracing and event correlation. In IEEE/IFIP Network Operations and Management Symposium (NOMS), pages 509–515, 2016. (Cited on pages 22, 86, 93 and 255.)

[Valdes 2001] Alfonso Valdes and Keith Skinner. Probabilistic alert correlation. In Recent advances in intrusion detection, pages 54–68, 2001. (Cited on pages 73, 92 and 248.)

[Valeur 2004] Fredrik Valeur, Giovanni Vigna, Christopher Kruegel and Richard A. Kemmerer. Comprehensive approach to intrusion detection alert correlation. IEEE Transactions on Dependable and Secure Computing, vol. 1, no. 3, pages 146–169, 2004. (Cited on pages 41, 42, 43, 86, 89, 92, 170 and 256.)

[Valigiani 2006] Grégory Valigiani. Développement d'un paradigme d'optimisation par Hommilière et application à l'enseignement assisté par ordinateur sur Internet. Ph.d. thesis, Université du Littoral Côte d'Opale, 2006. (Cited on page 138.)

[Van Steen 2010] Maarten Van Steen. Graph theory and complex networks. an introduction. Self-publishing, 2010. (Cited on page 43.)

[Vasilomanolakis 2016] Emmanouil Vasilomanolakis, Shreyas Srinivasa, Carlos García Cordero and Max Mühlhäuser. Multi-stage Attack Detection and Signature Generation with ICS Honeypots. In IEEE/IFIP Network Operations and Management Symposium (NOMS), pages 1227–1232, 2016. (Cited on page 85.)

[Veeramachaneni 2016] Kalyan Veeramachaneni, Ignacio Arnaldo, Vamsi Korrapati, Constantinos Bassias and Ke Li. AI^2: training a big data machine to defend. In IEEE 2nd International Conference on Big Data Security on Cloud (BigDataSecurity), IEEE International Conference on High Performance and Smart Computing (HPSC), and IEEE International Conference on Intelligent Data and Security (IDS), pages 49–54. IEEE, 2016. (Cited on pages ix and 100.)

[Vert 2018] Gregory Vert, Ann Leslie Claesson-Vert, Jesse Roberts and Erica Bott. A technology for detection of Advanced Persistent Threat in networks and systems using a finite angular state velocity machine and vector mathematics, in Computer and Network Security Essentials, chapter 3, pages 41–64. Springer, 2018. (Cited on pages 2 and 66.)

[Vigna 1998] Giovanni Vigna and Richard A. Kemmerer. NetSTAT: A network-based intrusion detection approach. In Computer Security Applications Conference, 1998. Proceedings. 14th Annual, pages 25–34, 1998. (Cited on page 64.)

[Vogel 2011a] Michael Vogel and Sebastian Schmerl. Efficient Distributed Intrusion Detection applying Multi Step Signatures. In OASIcs-OpenAccess Series in Informatics, volume 17, pages 188–193. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2011. (Cited on pages 39, 88 and 255.)

[Vogel 2011b] Michael Vogel, Sebastian Schmerl and Hartmut König. Efficient distributed signature analysis. In IFIP International Conference on Autonomous Infrastructure, Management and Security, pages 13–25, 2011. (Cited on pages 39, 88 and 255.)

[Wang 1997] Wen-June Wang. New similarity measures on fuzzy sets and on elements. Fuzzy sets and systems, vol. 85, no. 3, pages 305–309, 1997. (Cited on page 45.)

[Wang 2004] Liang-Min Wang, Jian-Feng Ma and Yong-Zhao Zhan. Enhancing the content of the intrusion alerts using logic correlation. In Content Computing, pages 137–142. Springer, 2004. (Cited on pages 41, 86 and 254.)

[Wang 2005a] Liang-Min Wang and Jian-Feng Ma. Two-stage algorithm for correlating the intrusion alerts. Wuhan University Journal of Natural Sciences, vol. 10, pages 89–92, 2005. (Cited on pages 41 and 254.)

[Wang 2005b] Lingyu Wang, Anyi Liu and Sushil Jajodia. An efficient and unified approach to correlating, hypothesizing, and predicting intrusion alerts. In European Symposium on Research in Computer Security, pages 247–266. Springer, 2005. (Cited on pages 80 and 250.)

[Wang 2006a] Jianxin Wang, Hongzhou Wang and Geng Zhao. A GA-based Solution to an NP-hard Problem of Clustering Security Events. In International Conference on Communications, Circuits and Systems Proceedings, volume 3, pages 2093–2097, 2006. (Cited on pages 50, 52, 75 and 249.)

[Wang 2006b] Li Wang, Zhitang Li and Jun Fan. Learning attack strategies through attack sequence mining method. In International Conference on Communication Technology (ICCT), pages 1–4. IEEE, 2006. (Cited on pages 89, 93 and 256.)

[Wang 2006c] Li Wang, Zhitang Li, Jie Lei and Yao Li. A novel algorithm SF for mining attack scenarios model. In IEEE International Conference on e-Business Engineering (ICEBE), pages 55–61. IEEE, 2006. (Cited on pages 73, 89, 93 and 248.)

[Wang 2006d] Li Wang, Zhitang Li and Qi-hong Wang. A novel technique of recognizing multi-stage attack behaviour. In International Workshop on Networking, Architecture, and Storages (IWNAS), pages 188–193. IEEE, 2006. (Cited on pages 43, 73, 89, 93 and 248.)

[Wang 2006e] Lingyu Wang, Anyi Liu and Sushil Jajodia. Using attack graphs for correlating, hypothesizing, and predicting intrusion alerts. Computer Communications, vol. 29, no. 15, pages 2917–2933, 2006. (Cited on pages 56, 80, 92 and 250.)

[Wang 2007a] Li Wang, Zhitang Li and Jie Lei. Learning attack strategies through mining and correlation of security alarms. In 10th IFIP/IEEE International Symposium on Integrated Network Management (IM), pages 713–716, 2007. (Cited on pages 86, 89, 93 and 254.)

[Wang 2007b] Li Wang, Zhitang Li, Dong Li and Jie Lei. Attack scenario construction with a new sequential mining technique. In Eighth ACIS International Conference on Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing (SNPD), volume 1, pages 872–877. IEEE, 2007. (Cited on pages 86, 89, 93 and 254.)

[Wang 2008] Lingyu Wang and Sushil Jajodia. An approach to preventing, correlating, and predicting multi-step network attacks. Intrusion Detection Systems, vol. 93, 2008. (Cited on pages 2, 55, 64, 80, 170 and 250.)

[Wang 2010] Li Wang, Ali A. Ghorbani and Yao Li. Automatic multi-step attack pattern discovering. International Journal of Network Security, vol. 10, no. 2, pages 142–152, 2010. (Cited on pages 49, 52, 53, 73, 92, 93 and 248.)

[Wang 2016] Chih-Hung Wang and Ye-Chen Chiou. Alert Correlation System with Automatic Extraction of Attack Strategies by Using Dynamic Feature Weights. International Journal of Computer and Communication Engineering, vol. 5, page 1, 2016. (Cited on pages 31, 48, 49, 52, 53, 57, 60, 74, 170 and 248.)

[Wang 2018] Qiwen Wang, Jianguo Jiang, Zhixin Shi, Wen Wang, Bin Lv, Biao Qi and Qieli Yin. A Novel Multi-source Fusion Model for Known and Unknown Attack Scenarios. In 17th IEEE International Conference On Trust, Security And Privacy In Computing And Communications/12th IEEE International Conference On Big Data Science And Engineering (TrustCom/BigDataSE), pages 727–736. IEEE, 2018. (Cited on pages 87, 170 and 255.)

[Wasserman 2013] Larry Wasserman. All of statistics: a concise course in statistical inference. Springer Science & Business Media, 2013. (Cited on pages 80 and 81.)

[Wen 2017] Senhao Wen, Nengqiang He and Hanbing Yan. Detecting and Predicting APT Based on the Study of Cyber Kill Chain with Hierarchical Knowledge Reasoning. In Proceedings of the 2017 VI International Conference on Network, Communication and Computing, pages 115–119. ACM, 2017. (Cited on pages 86 and 255.)

[White 2017] Russ White and Ethan Banks. Computer networking problems and solutions: An innovative approach to building resilient, modern networks. Addison-Wesley Professional, 2017. (Cited on pages 19, 21 and 23.)

[Wicherts 2012] Jelte M Wicherts and Marjan Bakker. Publish (your data) or (let the data) perish! Why not publish your data too? Intelligence, vol. 40, pages 73–76, 2012. (Cited on page 98.)

[Williams 1951] Frederic C. Williams, Tom Kilburn and Geoffrey C. Tootill. Universal high-speed digital computers: a small-scale experimental machine. Proceedings of the IEE-Part II: Power Engineering, vol. 98, no. 61, pages 13–28, 1951. (Cited on page 12.)

[Wilson 2010] Robin J Wilson. Introduction to graph theory. Addison Wesley Longman Limited, 5 édition, 2010. (Cited on pages 42 and 43.)

[Witte 2010] Robert S. Witte and John S. Witte. Statistics. John Wiley & Sons, Inc., 2010. (Cited on page 57.)

[Wojick 1975] David E Wojick. Issue analysis. an introduction to the use of issue trees and the nature of complex reasoning. Autopublished, 1975. (Cited on pages ix, 107 and 108.)

[Wu 2010] Yanghui Wu, John McCall and David Corne. Two novel Ant Colony Optimization approaches for Bayesian network structure learning. In IEEE Congress on Evolutionary Computation (CEC), pages 1–7. IEEE, 2010. (Cited on page 155.)

[Xian 2016] Minyi Xian and Yongtang Zhang. A privacy-preserving multi-step attack correlation algorithm. In IEEE Advanced Information Management, Communicates, Electronic and Automation Control Conference (IMCEC), pages 1389–1393, 2016. (Cited on pages 48, 52, 53, 83 and 252.)

[Xiao 2006] Yun Xiao and Chongzhao Han. Correlating intrusion alerts into attack scenarios based on improved evolving self-organizing maps. International Journal of Computer Science and Network Security, vol. 6, pages 199–203, 2006. (Cited on pages 86 and 254.)

[Xiao 2008] Shisong Xiao, Yugang Zhang, Xuejiao Liu and Jingju Gao. Alert fusion based on cluster and correlation analysis. In International Conference on Convergence and Hybrid Information Technology (ICHIT), pages 163–168. IEEE, 2008. (Cited on pages 50, 52 and 75.)

[Xu 2004] Dingbang Xu and Peng Ning. Alert correlation through triggering events and common resources. In 20th Annual Computer Security Applications Conference, pages 360–369. IEEE, 2004. (Cited on pages 79, 80, 93 and 250.)

[Xu 2006] Dingbang Xu and Peng Ning. Correlation analysis of intrusion alerts. Thesis, North Carolina State University, 2006. (Cited on pages 79, 93 and 250.)

[Xuewei 2010] Feng Xuewei, Wang Dongxia, Zeng Jiemei, Ma Guoqing and Li Jin. Analyzing and correlating security events using state machine. In IEEE 10th International Conference on Computer and Information Technology (CIT), pages 2849–2854, 2010. (Cited on pages 86 and 254.)

[Xuewei 2014] Feng Xuewei, Wang Dongxia, Huang Minhuan and Sun Xiaoxia. An approach of discovering causal knowledge for alert correlating based on data mining. In IEEE 12th International Conference on Dependable, Autonomic and Secure Computing (DASC), pages 57–62, 2014. (Cited on pages 49, 52, 57, 81, 170 and 252.)

[Xupeng 2014] Fang Xupeng, Zhai Lidong, Jia Zhaopeng and Bai Wenyan. A Game Model for Predicting the Attack Path of APT. In Proceedings of the 2014 IEEE 12th International Conference on Dependable, Autonomic and Secure Computing, pages 491–495. IEEE Computer Society, 2014. (Cited on page 85.)

[Ya 2015] Jing Ya, Tingwen Liu, Haoliang Zhang, Jinqiao Shi and Li Guo. An Automatic Approach to Extract the Formats of Network and Security Log Messages. In IEEE Military Communications Conference (MILCOM), pages 1542–1547, Tampa, FL, USA, 2015. IEEE. (Cited on page 19.)

[Yan 2004] Wei Yan and Fang Liu. Semantic scheme to extract attack strategies for Web service network security. In IEEE Workshop on IP Operations and Management, pages 104–111, 2004. (Cited on pages 33, 80, 170 and 250.)

[Yan 2005] Wei Yan. Network Attack Scenarios Extraction and Categorization by Mining IDS Alert Streams. J. UCS, vol. 11, pages 1367–1382, 2005. (Cited on pages 80, 170 and 250.)

[Yang 2008] Shanchieh J. Yang, Stephen R. Byers, Jared Holsopple, Brian Argauer and Daniel S. Fava. Intrusion activity projection for cyber situational awareness. In IEEE International Conference on Intelligence and Security Informatics (ISI), pages 167–172, 2008. (Cited on pages 82, 93 and 251.)

[Yang 2009] Shanchieh J. Yang, Adam Stotz, Jared Holsopple, Moises Sudit and Michael E. Kuhl. High level information fusion for tracking and projection of multistage cyber attacks. Information Fusion, vol. 10, no. 1, pages 107–121, 2009. (Cited on pages 84, 89, 93 and 256.)

[Yen 2010] John Yen, Michael McNeese, Tracy Mullen, David Hall, Xiaocong Fan and Peng Liu. RPD-based hypothesis reasoning for cyber situation awareness. In Cyber situational awareness, pages 39–49. Springer, 2010. (Cited on pages 105 and 109.)

[Yu 2004] Dong Yu and Deborah Frincke. A novel framework for alert correlation and understanding. In International Conference on Applied Cryptography and Network Security, pages 452–466. Springer, 2004. (Cited on pages 39, 88 and 256.)

[Yu 2007] Dong Yu and Deborah Frincke. Improving the quality of alerts and predicting intruder's next goal with Hidden Colored Petri-Net. Computer Networks, vol. 51, no. 3, pages 632–654, 2007. (Cited on pages 22, 39, 43, 88, 111, 170 and 256.)

[Zali 2012] Zeinab Zali, Massoud Reza Hashemi and Hossein Saidi. Real-Time Attack Scenario Detection via Intrusion Detection Alert Correlation. In 9th International ISC Conference on Information Security and Cryptology (ISCISC), pages 95–102. IEEE, 2012. (Cited on pages 43, 56, 86 and 255.)

[Zali 2013] Zeinab Zali, Massoud Reza Hashemi and Hossein Saidi. Real-time intrusion detection alert correlation and attack scenario extraction based on the prerequisite-consequence approach. The ISC International Journal of Information Security, vol. 4, no. 2, 2013. (Cited on pages 86, 171, 229 and 255.)

[Zargar 2014] Saman T. Zargar. ONTIDS: A highly flexible context-aware and ontology-based alert correlation framework. In 6th International Symposium on Foundations and Practice of Security, La Rochelle, France, October 21-22, 2013, volume 8352, page 161, 2014. (Cited on pages 34, 87, 172, 187 and 255.)

[Zhai 2006] Yan Zhai, Peng Ning and Jun Xu. Integrating IDS alert correlation and OS-level dependency tracking. In International Conference on Intelligence and Security Informatics, pages 272–284. Springer, 2006. (Cited on pages 43, 79, 80, 93, 105 and 250.)

[Zhang 2007] Aifang Zhang, Zhitang Li, Dong Li and Li Wang. Discovering Novel Multistage Attack Patterns in Alert Streams. In International Conference on Networking, Architecture, and Storage (NAS), pages 115–121. IEEE, 2007. (Cited on pages 86, 89, 93 and 254.)

[Zhang 2008] Shaojun Zhang, Jianhua Li, Xiuzhen Chen and Lei Fan. Building network attack graph for alert causal correlation. Computers & Security, vol. 27, no. 5, pages 188–196, 2008. (Cited on pages 85 and 253.)

[Zhang 2009] Zonghua Zhang and Pin-Han Ho. Janus: A dual-purpose analytical model for understanding, characterizing and countermining multi-stage collusive attacks in enterprise networks. J. Netw. Comput. Appl., vol. 32, no. 3, pages 710–720, 2009. (Cited on page 138.)

[Zhang 2015] Yang Zhang, Tingwen Liu, Jinqiao Shi, Panpan Zhang, Haoliang Zhang and Jing Ya. An Automatic Multi-Step Attack Pattern Mining Approach for Massive WAF Alert Data. In 36th IEEE Symposium on Security and Privacy (S & P), 2015. (Cited on pages 43, 46, 50, 52, 76, 121, 227 and 249.)

[Zhang 2016] Yongtang Zhang, Xianlu Luo and Haibo Luo. A multi-step attack-correlation method with privacy protection. Journal of Communications and Information Networks, vol. 1, pages 133–142, 2016. (Cited on pages 48, 52, 53, 83 and 252.)

[Zhang 2017] Daojuan Zhang, Kexiang Qian, Peng Zhang, Shu Mao and Hongbin Wu. Alert correlation analysis based on attack path graph. In IEEE Conference on Energy Internet and Energy System Integration (EI2). IEEE, 2017. (Cited on pages 79, 187 and 250.)

[Zhao 2014] Wentao Zhao, Pengfei Wang and Fan Zhang. Extended Petri Net-Based Advanced Persistent Threat Analysis Model. In Computer Engineering and Networking, pages 1297–1305. Springer, 2014. (Cited on pages ix and 40.)

[Zhe 2011] Gong Zhe, Li Dan, An Baoyu, Ou Yangxi, Cui Wei, Niu Xinxin and Xin Yang. An analysis of ant colony clustering methods: models, algorithms and applications. International Journal of Advancements in Computing Technology, vol. 3, no. 11, pages 112–121, 2011. (Cited on page 138.)

[Zhong 2013] Chen Zhong, Deepak S Kirubakaran, John Yen, Peng Liu, Steve Hutchinson and Hasan Cam. How to use experience in cyber analysis: An analytical reasoning support system. In IEEE International Conference on Intelligence and Security Informatics (ISI), pages 263–265. IEEE, 2013. (Cited on pages ix and 106.)

[Zhong 2014] Chen Zhong, Deepak Samuel, John Yen, Peng Liu, Robert Erbacher, Steve Hutchinson, Renee Etoty, Hasan Cam and William Glodek. RankAOH: Context-driven similarity-based retrieval of experiences in cyber analysis. In IEEE International Inter-Disciplinary Conference

on Cognitive Methods in Situation Awareness and Decision Support (CogSIMA), pages 230–236. IEEE, 2014. (Cited on page 106.)

[Zhong 2015] Chen Zhong, John Yen, Peng Liu, Rob Erbacher, Renee Etoty and Christopher Garneau. An integrated computer-aided cognitive task analysis method for tracing cyber-attack analysis processes. In Proceedings of the 2015 Symposium and Bootcamp on the Science of Security, page 9. ACM, 2015. (Cited on pages ix, 106 and 107.)

[Zhong 2017] Chen Zhong, John Yen, Peng Liu, Rob F Erbacher, Christopher Garneau and Bo Chen. Studying analysts' data triage operations in cyber defense situational analysis. In Theory and Models for Cyber Situation Awareness, pages 128–169. Springer, 2017. (Cited on page 107.)

[Zhou 2007] Jingmin Zhou, Mark Heckman, Brennen Reynolds, Adam Carlson and Matt Bishop. Modeling network intrusion detection alerts for correlation. ACM Transactions on Information and System Security (TISSEC), vol. 10, no. 1, page 4, 2007. (Cited on pages 38, 80, 171 and 250.)

[Zhou 2010] Chenfeng Vincent Zhou, Christopher Leckie and Shanika Karunasekera. A survey of coordinated attacks and collaborative intrusion detection. Computers & Security, vol. 29, pages 124–140, 2010. (Cited on pages 22 and 87.)

[Zhu 2006] Bin Zhu and Ali A. Ghorbani. Alert correlation for extracting attack strategies. IJ Network Security, vol. 3, no. 3, pages 244–258, 2006. (Cited on pages 31, 43, 48, 49, 60, 71, 73, 83, 92, 93, 111, 170 and 248.)

[Zuech 2015] Richard Zuech, Taghi M. Khoshgoftaar and Randall Wald. Intrusion detection and big heterogeneous data: a survey. Journal of Big Data, vol. 2, no. 1, page 3, 2015. (Cited on pages 4 and 22.)

Julio NAVARRO

Modelization and Identification of Multi-step Cyberattacks in Sets of Events

Résumé

Une cyberattaque est considerée comme *multi-étapes* si elle est composée d'au moins deux actions différentes. L'objectif principal de cette thèse est aider l'analyste de sécurité dans la création de modèles de détection à partir d'un ensemble de cas alternatifs d'attaques multi-étapes. Pour répondre à cet objectif, nous présentons quattre contributions de recherche.

D'abord, nous avons réalisé la première bibliographie systématique sur la détection d'attaques multi-étapes. Une des conclusions de cette bibliographie est la manque de méthodes pour confirmer les hypothèses formulées par l'analyste de sécurité pendant l'investigation des attaques multi-étapes passées.

Ça nous conduit à la deuxième de nos contributions, le graphe des scénarios d'attaques abstrait ou AASG. Dans un AASG, les propositions alternatives sur les étapes fondamentales d'une attaque sont répresentées comme des branches pour être évaluées avec l'arrivée de nouveaux événements.

Pour cette évaluation, nous proposons deux modèles, Morwilog et Bidimac, qui font de la détection au même temps que l'identification des hypothèses correctes. L'évaluation des résultats par l'analyste permet l'évolution des modèles.

Finalement, nous proposons un modèle pour l'investigation visuel des scénarios d'attaques sur des événements non traités. Ce modèle, qui s'appelle SimSC, est basé sur la similarité entre les adresses IP, en prenant en compte la distance temporelle entre les événements.

Mots-clés : Cybersécurité, attaque multi-étapes, correlation d'événements, détection d'attaques

Summary

A cyberattack is considered as *multi-step* if it is composed of at least two distinct actions. The main goal of this thesis is to help the security analyst in the creation of detection models from a set of alternative multi-step attack cases. To meet this goal, we present four research contributions.

First of all, we have conducted the first systematic survey about multi-step attack detection. One of the conclusions of this survey is the lack of methods to confirm the hypotheses formulated by the security analyst during the investigation of past multi-step attacks.

This leads us to the second of our contributions, the Abstract Attack Scenario Graph or AASG. In an AASG, the alternative proposals about the fundamental steps in an attack are represented as branches to be evaluated on new incoming events.

For this evaluation, we propose two models, Morwilog and Bidimac, which perform detection and identification of correct hypotheses. The evaluation of the results by the analyst allows the evolution of the models.

Finally, we propose a model for the visual investigation of attack scenarios in non-processed events. This model, called SimSC, is based on IP address similarity, considering the temporal distance between the events.

Keywords: Cybersecurity, multi-step attack, event correlation, attack detection

Résumé de thèse

Modélisation et identification de cyberattaques multi-étapes dans des ensembles d'événements

Julio Navarro

Soutenu le : 14 mars 2019

Ce document constitute le résumé en français de la thèse "Modelization and Identification of Multi-step Cyberattacks in Sets of Events", de Julio Navarro, écrite en anglais. La recherche pour cette thèse a été fait au sein de l'équipe CSTB du Laboratoire Icube à l'Université de Strasbourg. Elle a été financée par un projet BPI qui s'appelle HuMa.

1 Introduction

Dans un monde dans lequel les réseaux d'ordinateurs sont devenus indispensables dans le quotidien, une multitude de mécanismes ont été mises en place pour protéger ces réseaux des possibles cyberattaques. Ces mécanismes sont basés sur l'utilisation d'outils et processus gérés par des analystes de sécurité [Sundaramurthy 2015]. Une des sources d'information utilisées par l'analyste de sécurité vient des événements enregistrés par chaque dispositif, service ou application du réseau.

Dans ce thèse, nous nos sommes intéressé à la modélisation et identification des cyberattaques à partir de l'information contenue dans ces événements. La modélisation est le processus d'abstraction des conclusions déduites par l'analyste à partir d'un ensemble d'événement où les actions d'une attaque sont représentées. Le résultat de ce processus sont des modèles qui représentent chacune des hypothèses posés par l'analyste. L'identification est la confirmation de ces modèles auprès de la arrivée de nouvelles attaques qui les correspondent. L'identification correcte de modèles est fondamentale pour éviter des attaques similaires dans le futur.

Nous nous sommes focalisé sur un type spécifique de cyberattaque : les attaques multi-étapes. Ce terme désigne une cyberattaque composée de plusieurs phases ou actions différentes, chacune représentée par une ou plusieurs événements. Les actions exactes des attaquants sont souvent difficile d'analyser quand on fait l'analyse d'attaques multi-étapes. Un des problèmes est que les actions qui composent une attaque multi-étapes n'ont pas nécessairement une corrélation causale forte avec les autres, et plusieurs combinaisons d'étapes sont possibles. Il y a aussi des actions qui peuvent sembler de faire partie de l'attaque mais qui ne sont pas des étapes fondamentales qui doivent se répéter dans une future occurrence de l'attaque.

La question de recherche à laquelle répond cette thèse est :

Comment peut on aider l'analyste de sécurité à décider les plus probables parmi plusieurs hypothèses des attaques multi-étapes ? Peut on réaliser la détection en même temps que ce processus d'identification a lieu ?

L'objectif de cette thèse est la création de modèles pour :

- Exprimer les différents hypothèses sur une attaque multi-étape.
- Identifier les hypothèses correctes en faisant au même temps de la détection.

On va voir que ce problème n'a pas été abordé par la littérature sur les attaques multi-étapes.

2 Définitions préliminaires

Dans cette section, nous présentons quelques définitions qui sont clés pour comprendre les contributions de la recherche.

2.1 Cyberattaque

Le premier concept basique à définir est la *cyberattaque*. Une cyberattaque est un acte qui compromet la confidentialité, l'integrité ou la disponibilité d'un réseau d'ordinateurs ou de l'information qu'il contient.

Dans le cadre de cette recherche, on se focalise exclusivement sur ceux qui ont lieu au *Cyberspace*, qui est l'environment abstrait où la communication des ordinateurs a lieu. Nous ne considérons pas de cyberattaques qui affectent le monde physique.

Dans ce contexte, on peut considerer deux acteurs principaux, l'attaquent, qui est l'agent qui exécute une cyberattaque, et le défenseur, qui est l'entité en charge de protéger le réseau attaqué. Nous prenons la place du défenseur, qui est personnifié dans la figure de l'analyste de sécurité. Elle va être la responsable de développer les modèles de détection pour détecter les cyberattaques.

2.2 Trace

Pour savoir qu'est-ce qui se passe au réseau, l'analyste compte sur les traces, qui enregistrent de l'information sur les actions faites. Il y a deux grands types de traces très utilisés (Figure 1) :

- **Les paquets**. Ils sont l'unité minimale d'exchange d'information dans un protocol de communication.
- **Les événements**. Ils sont des actions préservées dans les actifs du réseau dans un format de texte qui s'appelle *log*. Il y a un type spécial d'événements, les alertes, qui sont des événements qui signalent sur une action malveillante ou une faille.

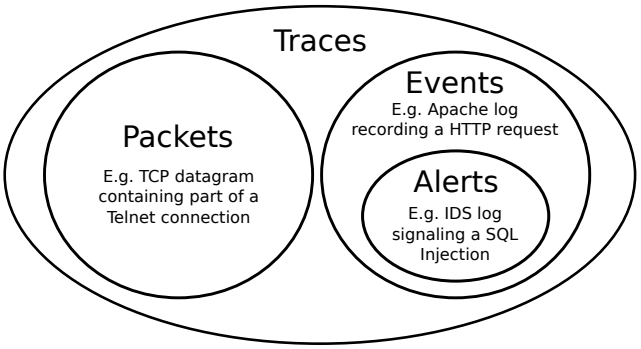


FIGURE 1 – Représentation des traces

Dans cette recherche, nous nous sommes focalisé sur les événements parce qu'ils sont des traces spécifiquement créé pour représenter des actions, pas comme les paquets, qui sont des unités d'exchange d'information qui éventuellement contiennent de l'information sur des actions.

2.3 Le processus de cybersécurité

L'analyste est une pièce centrale [Caulkins 2018] qui est capable d'incorporer au processus de détection la créativité humaine et un ensemble des connaissances construit à partir de son expérience.

Les réseaux d'ordinateurs sont devenus indispensables dans le quotidien. Ils nous permettent de communiquer avec des personnes distantes, partager des fichiers ou simplifier les tâches administratives. Malgré ses avantages, il y a aussi des risques dans l'interconnexion des ordinateurs. Par exemple, la cyberattaque WannaCry a provoqué des ravages en 2017, en affectant plus de 230.000 ordinateurs dans 150 pays, comme la Russie ou l'Espagne [Ehrenfeld 2017]. WannaCry est un exemple de *cyberattaque multi-étapes*, un terme qui désigne une cyberattaque composée de plusieurs phases ou étapes différentes.

Ces actions sont enregistrées dans les *traces* du réseau, présentées dans la section 2.2. Un exemple de trace sont les fichiers de *logs*, des journaux des événements qui sont sauvegardés dans les dispositifs connectés au réseau. Ils existent trois activités majeures d'exploitation de ces fichiers : l'investigation, la détection et la prédiction. Chacune est liée à un moment de temps : le passé, le présent et le futur, respectivement. La détection des attaques en temps réel et la prédiction des attaques futures sont basés sur modèles construits grâce aux conclusions obtenues après l'investigation des attaques exécutées dans le passé. Notre question de recherche fait référence au processus d'abstraction de ces conclusions, ou *modélisation*, et à la confirmation des modèles construits, ou *identification*.

2.4 Attaques multi-étapes

Les nombre des actions contenues dans une attaque nous permet de distinguer deux types d'attaques : mono-étape et multi-étape. Dans un côté, les attaques *mono-étape* sont composées par seulement une action. Une injection de SQL et une dénégation de service sont des examples d'attaques mono-étapes. Si l'attaque est composé de plusieurs actions différentes, on l'appelle attaque *multi-étape*.

Nous nous sommes focalisé sur les attaques multi-étapes par son grande répercussion actuelle. Un indicateur de ça c'est le longue temps moyenne qui passe entre le début d'une cyberattaque et la découverte de l'intrusion, qui permet aux attaquants de developper des attaques avec plusieurs étapes (Table 1).

[Fire Eye 2018]	101 days of median dwell time
[CrowdStrike 2018]	86 days of average dwell time
[Trustwave 2018]	83 days of average dwell time

TABLE 1 – Temps qui passe entre le début d'une cyberattaque et la découverte de l'intrusion (dwell time) pour l'année 2017.

Un example concret d'une attaque multi-étape récente est WannaCry, apparue en mai 2017. Elle est probablement la pire attaque de l'histoire des réseaux informatiques. Elle a affecté des centaines de milliers d'ordinateurs dans le monde. L'attaque est basée sur un malware qui crypte l'information de la machine infectée et se propage en profitant d'une faille du protocol SMB v1. La mitigation de cette attaque a été pas facile : même si de signatures simples pour détecter le malware ont été développé le même jour de l'infection, les analystes de sécurité ont passé au moins un mois a dévoiler tous les étapes de l'infection (Figure 2). La mise en place des modèles de détection focalisés sur les étapes intermédiaires et pas seulement sur la fin de l'attaque (l'infection de malware) aurais pu réduire l'impact de l'infection.

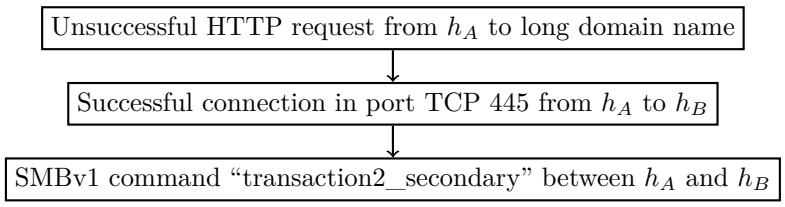


FIGURE 2 – Une séquence d'actions appartenant à WannaCry.

3 Présentation des contributions

La littérature scientifique se focalise seulement sur des modèles des detections finales, construits à partir des événements qui appartiennent à une attaque concrete. L'analyste de sécurité modélise directement l'attaque trouvé pour l'incorporer au processus de détection sur les événements à venir.

Ce modèle ne laisse pas de la place pour les possibilités alternatives de l'attaque. Ces possibilités viennent des questions qui se pose l'analyste : quelles sont les actions exactes qui participent à l'attaque, est-ce qu'elles vont se répéter, est-ce que le même ordre va être respecté, etc. En plus, si l'attaque est

ciblé, l'analyste a seulement une example d'attaque à étudier et les questions ne peuvent pas être résolu.

Ça ramène à l'analyste à poser une série des hypothèses alternatives sur l'attaque, qui donnent comme résultat des modèles alternatifs. Notre idée est de préserver ces modèles pour le processus de détection donc on peut détecter des sequences qui correspondent aux modèles et au même temps faire un processus d'identification des modèles correctes pour consolider un modèle de détection.

Nous proposons un nouveau modèle pour confronter les attaques. Si dans le modèle classique on a l'analyste seulement dans un côté du processus, avec les alarmes générés par la détection envoyées à lui dans un seul sense, notre modèle place l'analyste dans le centre du processus, en lien avec la philosophie suivie par tous les partenaires du projet HuMa, qui a financé cette recherche.

On introduit un nouveau processus que c'est l'identification, grace auquel on va pouvoir déterminer quels sont les modèles identifiés et plus pertinents pour créer un modèle de détection définitif. Dans notre modèle, l'analyste modélise tout un ensemble des hypothèses alternatives au lieu d'un modèle de détection unique. Pour faire ça, Nous devons d'abord proposer un modèle pour que l'analyste puisse exprimer ses hypothèses. Ces hypothèses doivent être représentés dans une structure commun pour exécuter le processus de détection. Nous proposons comme structure commun le modèle AASG, une des contributions de cette recherche.

L'AASG est fait pour représenter les hypothèses comme des chemins qui peuvent être parcouru par des algorithmes de recherche. L'objectif des algorithmes est faire de la détection et identification d'attaques multi-étapes sur le modèle AASG. Il faut donc développer des algorithmes qui travaillent sur les événements qui arrivent au système et génèrent des alarmes qui peuvent être évalués par l'analyste de sécurité. Elle pourra donc donner de l'information à l'algorithme pour son apprentissage. Nous proposons deux algorithmes pour ce propos : Morwilog, basé sur le comportement des fourmis, et Bidimac, basé sur la statistique Bayésienne.

Pour arriver aux conclusions qui nous ont permis de développer ces contributions, on a fait la première recherche bibliographique qui existe sur la détection des attaques multi-étape, qui représente aussi une contribution de cette thèse.

Les contributions mentionnés sont présentés brièvement dans la liste suivante :

- **Bibliographie systématique sur la détection d'attaques multi-étapes**. Dans les publications traitant le problème de recherche des attaques multi-étapes dans des jeux d'événements il n'y a pas de distinction claire entre les attaques passées (investigation sur des fichiers de traces statiques) et présentes (détection sur des données en temp réel). En fait, la plupart des méthodes de détection sont évaluées sur des jeux estatiques, car il est difficile de générer des attaques multi-étapes et les analyser en temps réel. On parle donc de *méthodes de détection* pour parler des méthodes de recherche des attaques qui peuvent éventuellement être utilisées aussi pour l'investigation. Des méthodes spécifiques pour la détection d'attaques multi-étapes ont été développées depuis le dernier changement de siècle, mais jusqu'à maintenant il n'existait pas d'une analyse du domaine, ses défis et ses perspectives. Une de nos contributions est le premier travail de bibliographie systématique sur la détection d'attaques multi-étapes dans des enjeux de traces.
- **Graphes des scénarios d'attaques abstraits (AASG)**. L'ensemble des hypothèses proposées par l'analyste de sécurité sur la manière dont une attaque multi-étape peut se répéter doit être modélisé pour effectuer le processus d'identification des celles qui sont avérées. Nous proposons un nouveau modèle, le AASG (Abstract Attack Scenario Graph en anglais), où les propositions alternatives sont représentées pour être évaluées avec l'arrivée de nouveaux événements. Au contraire de la représentation concrète d'une attaque multi-étape, que nous appelons CASG (Concrete Attack Scenario Graph), un AASG peut capturer l'abstraction nécessaire pour la définition des cas futurs de l'attaque. Les cas alternatifs sont organisés dans les différentes branches de l'AASG, toujours à partir d'un nœud racine qui permet l'identification du premier événement de la séquence et la sélection de l'AASG adéquat. Chaque nœud contient un événement abstrait qui représente les événements qui peuvent être identifiés dans cette étape de l'attaque en utilisant un ensemble de fonctions.
- **Algorithmes pour la détection et l'identification de scénarios**. Une autre contribution de cette thèse est la proposition de deux algorithmes pour faire la détection et l'identification d'attaques multi-étapes en utilisant les AASGs : Morwilog et Bidimac. Dans les deux, l'analyste est directement impliqué dans le processus, en envoyant l'information sur la nature malicieuse des attaques détectés. Morwilog est un algorithme basé sur le comportement des fourmis où une fourmi artificielle, appelé *morwi*, est créé chaque fois qu'un événement qui correspond au nœud racine d'un AASG est detecté dans le système. En

utilisant des niveaux de phéromones assignés à chaque arc du graphe, les *morwis* sont guidées vers la solution plus probable en termes d'historique du réseau. Une alerte est envoyée à l'analyste chaque fois qu'une *morwi* arrive à un nœud sans descendance, pour permettre l'apprentissage du système à partir de l'information donnée par l'expert. D'autre part, Bidimac est une adaptation des méthodes classiques d'inférence bayésienne. Le graphe devient un AASG bayésien, avec des paramètres de probabilité assignés à chaque arc. Vu le caractère nouveau des AASGs, Bidimac offre une façon de comparer Morwilog avec une méthode bayésienne classique.

- **Investigation visuel des scénarios d'attaques**. Nous proposons une contribution additionnelle qui contribue au contexte de notre recherche principale. Cette contribution est SimSC, un modèle pour l'identification de scénarios d'attaques dans des sous-ensembles de logs de formats hétérogènes. Il est basé sur la similarité entre les adresses IP, en prenant en compte la distance temporelle entre les logs.

4 Étude bibliographique sur la détection d'attaques multi-étapes

La première contribution est une recherche systematic qui a été publié en 2018 [Navarro 2018]. Nous avons défini deux critères d'inclusion : on considère seulement des méthodes de détection qui travaille sur des traces réels qui arrive au système, et la structure complète de l'attaque doit être considéré. Ça veut dire qu'on ne considère pas des méthodes, par example, de détection de malware, qui sont capable de détecter la présence de l'attaque mais seulement par une étape.

Le champ de détection d'attaques multi-étape est relativement nouveau, de début des années 2000. Le résultats du survey est un corpus de 201 papiers qui présentent 138 méthodes différents.

Une des conclusions de l'étude bibliographique est sur les set de données utilisés dans les expériences. On voit que la plupart des publications travaillent avec au moins un set de données publique (Figure 3).

Par contre, nous avons remarqué qu'il existe trop peu des set de données publiques. En plus, la plupart des publications utilisent le set DARPA 2000, qu'il a été crée il y a 19 ans.

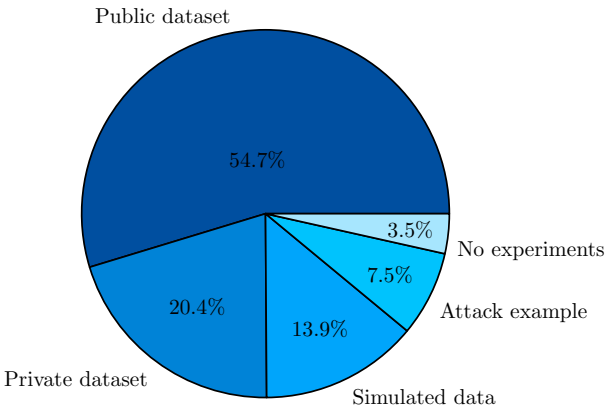


FIGURE 3 – Distribution des publications par le type des éxperiences.

Pour notre recherche, aux sets de données publique trouvés dans l'étude bibliographique on doit ajouter le set HuMa, crée en 2015 par un des partenaires membre du projet qui a financé cette thèse. Malheureusement, même s'il y a quelque datasets disponibles, aucun d'entre eux a ses donnés éttiquetés pour indiquer où sont les attaques. Seulement trois viennent avec des informations sur les attaques, donc on peut étiqueter les données à la main. Nous avons donc choisi ces trois pour évaluer nos contributions : ISCX, HuMa et DARPA 2000. Il faut savoir que chaque dataset choisi contient seulement une attaque par ensemble de données, et ça pose des difficultés pour l'évaluation.

Une autre conclusion est que les attaques multi-étape sont généralement modélisé comme des graphes acycliques dirigés (DAGs). Dans ces modèles, le nœuds contiennent des traces qui représentent les actions ou étapes de l'attaque. Les arêtes représentent la relation entre les actions, comme par example une relation de similarité. Vu que chaque auteur nomme ce structure d'une façon différent, on lui a donné le nom de Graphe de Scénario d'Attaques Concrets (CASG). On voit que ces modèles sont utilisés seulement pour la modélisation des attaques concrets : la modélisation des scénarios alternatives n'est pas considérés dans la littérature.

5 Graphes des scénarios d'attaques abstraits (AASG)

L'idée du CASG a été prise pour développer un modèle où les hypotheses alternatives de l'attaque puisse être modélisés. Un CASG permet de repré-

senter seulement une attaque. On a du donc faire un processus d'abstraction qui nous a amené au nouveau modèle.

Ce nouveau modèle s'appelle Graphe de Scénario d'Attaque Abstrait (AASG), et c'est la prochaine contribution qu'on va analyser. Les CASGs representent un résultat de la détection, et l'AASG est vraiment un modèle pour faire de la détection des hypothèses alternatives. Pour définir l'AASG il faut deux pas : la modélisation des hypothèses et la définition de la structure du modèle.

5.1 Modélisation des hypothèses

On peut commencer par la modélisation des hypothèses de l'analyste sur des attaques multi-étapes. Cette idée des hypothèses alternatives a été récupéré d'un papier de Pirolli et Card de 2005. Ces auteurs posent l'idée mais les hypothèses ne sont pas formellement définies.

Pour les définir formellement, il faut que chaque hypothèse represent une possible voie d'action d'une attaque multi-étape, donc le modèle doit être composé par chacune des étapes présentes dans l'attaque. Chaque fragment du modèle doit représenter un ensemble des événements, qui éventuellement vont se correspondre à chacune des étapes de l'attaque.

Pour représenter l'ensemble des événements qui correspondent à chaque étape, nous avons défini un événement abstrait comme un ensemble de conditions. On propose deux types de conditions : des *conditions absolus*, qui sont basées sur la comparaison avec des valeurs prédéfinies, et les *conditions relatives*, qui sont basées sur la comparaison avec les étapes précédentes de l'hypothèse.

Nous avons établi 10 conditions, 5 absolus et 5 relatives, qui sont définis par une série de functions mathématiques. Tous les conditions sont listés dans la Table 2.

5.2 Définition de la structure d'un AASG

Une fois les hypothèses sont modélisées, il faut définir la structure de l'AASG. Nous nous sommes inspirés dans le modèle AOH, développé pour l'analyse du comportement des analystes pendants la résolution de conflits. Dans ce modèle, les hypothèses sont simplement définies en format de texte libre, mais nous voulons les exprimer dans une façon mathématique pour les définir formellement et pour permettre aux algorithmes d'utiliser le modèle.

Type	Name of function	Short name	Formula
Absolute conditions	Equality	EQL	$g_1(n_a, r) = \begin{cases} 1, & \text{if } n_a = r \\ 0, & \text{otherwise} \end{cases}$
	Inequality	NEQ	$g_2(n_a, r) = \begin{cases} 1, & \text{if } n_a \neq r \\ 0, & \text{otherwise} \end{cases}$
	Prefix similarity	PFX	$g_3(n_a, r, \lambda) = \begin{cases} 1, & \text{if } \frac{l}{L} > \lambda \\ 0, & \text{otherwise} \end{cases}$
	Textual similarity	TXT	$g_4(n_a, r, \lambda) = \begin{cases} 1, & \text{if } jac(n_a, r) > \lambda \\ 0, & \text{otherwise} \end{cases}$
	Set-based	SET	$g_5(n_a, R) = \begin{cases} 1, & \text{if } n_a \in R \\ 0, & \text{otherwise} \end{cases}$
Relative conditions	Equality	SIM_EQL	$f_1(n_a, m_b) = \begin{cases} 1, & \text{if } n_a = m_b \\ 0, & \text{otherwise} \end{cases}$
	Common element	SIM_COM	$f_2(N_a, M_b) = \begin{cases} 1, & \text{if } N_a \cap M_b \neq \varnothing \\ 0, & \text{otherwise} \end{cases}$
	Prefix similarity	SIM_PFX	$f_3^*(n_a, m_b, \lambda) = \begin{cases} 1, & \text{if } \frac{l}{L} > \lambda \\ 0, & \text{otherwise} \end{cases}$
	Textual similarity	SIM_TXT	$f_4^*(n_a, m_b, \lambda) = \begin{cases} 1, & \text{if } jac(n_a, m_b) > \lambda \\ 0, & \text{otherwise} \end{cases}$
	Inequality	SIM_NEQ	$f_{neq}(n_a, m_p) = \begin{cases} 1, & \text{if } n_a \neq m_p \\ 0, & \text{otherwise} \end{cases}$

TABLE 2 – Fonctions utilisées pour le développement des événements abstraits dans les AASGs.

Nous arrivons à donner un caractère mathématique au modèle en utilisant les concepts de la Théorie de Graphes. Un autre condition pour le modèle est qu'il soit fini, parce que les hypothèses définies par l'analyste vont avoir un nombre limités des étapes. C'est pour ça que nous avons utilisés des graphes acycliques dirigés (DAG), sans boucles, pour éviter des possibles chemins infinis. Et finalement, le modèle doit être unique, pour permettre aux algorithmes de pouvoir choisir un AASG à parcourir pour chaque événement qui arrive au système. On peut avoir ça en utilisant un graphe avec une seule source.

En prenant en compte tous ces conditions, nous avons défini le AASG comme un graphe avec les hypothèses représentés dans chacune des branches. Chaque AASG va être identifié par la source, appelé nœud racine. Dans ce modèle, si on prends deux nœuds connectés, on appelle parent au nœud les plus proche au nœud racine et enfant au nœud le plus loin.

5.3 Example d'AASG

Comme example de modélisation des hypothèses et d'AASG on va prendre l'attaque contenue dans le dataset ISCX, représenté dans la Figure 4. Dans cette attaque, l'attaquant commence par capturer un des ordinateurs du réseau pour après sauter dans un autre. Depuis cette deuxième machine, un scan est fait contre le réseau de serveurs interne. Une fois la cible finale est trouvé, un SQL injection permets l'accès à la machine et l'accès est consolidé par l'installation d'un backdoor.

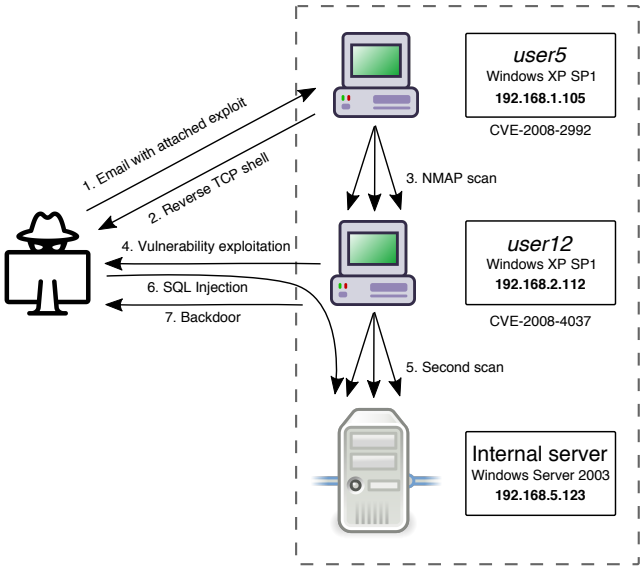


FIGURE 4 – Attaque UNB ISCX island-hopping [Zargar 2014].

Nous nous focalisons seulement dans les trois dernières pas de l'attaque, qu'on peut modéliser en utilisant des conditions absolus et relatives. D'abord le scan IP, après l'injection SQL et finalement le Backdoor. Mais l'order des deux derniers pas peut être renverser, donc l'analyste peut considérer aussi une deuxième hypothèse face à une nouvelle répétition de l'attaque (Figure 5).

En partant de cette attaque on a développé une structure d'attaque multi-étape représenté dans un AASG. Cette AASG s'appelle S2A et correspond à un scan suivi de deux attaques exécutées depuis des addresses IP différentes (Figure 6). Nous considérons deux types de scans, IP et un de service. Après les scans, nous considérons trois types d'attaques : une injection SQL, l'exécution d'un exploit et un bruteforce SSH. Ce type de représentation d'un

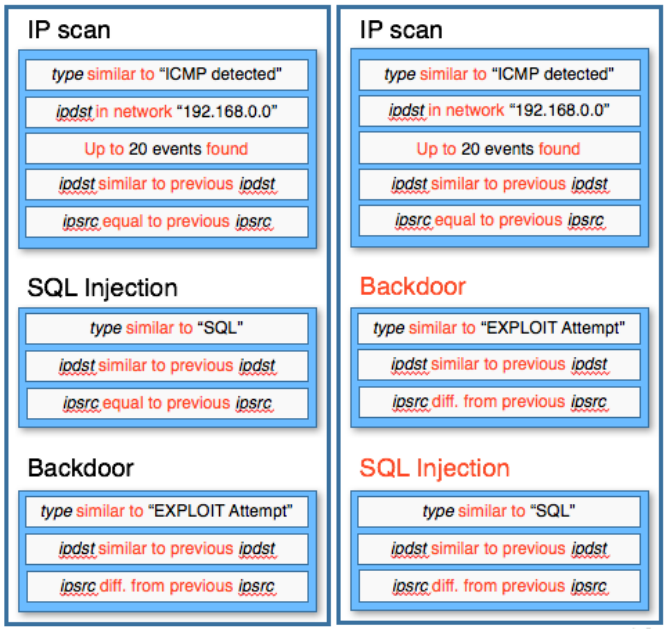


FIGURE 5 – Modélisation de deux hypothèses alternatives sur l'attaque UNB ISCX island-hopping

AASG est appelé la représentation fonctionnelle, et c'est simple à representer et à lire, donc l'analyste peut l'utiliser confortablement.

5.4 Représentation JSON

Une autre contribution lié aux AASGs est la représentation dans un format que les machines puissent lire et utiliser. Dans les AASGs, chaque nœud contient une représentation abstrait d'un log. On a défini un format JSON pour codifier les AASGs. Le format de ce JSON est assez simple :

- Le fichier représente un ensemble des arbres, qui sont contenus dans une liste sous la rubrique « trees ».
- Chaque arbre contient un identifiant (« id ») et une liste de nœuds (« nodes »).
- Chaque nœud contient quatre éléments :
  - Un identifiant (« id »)
  - Un set de comparaisons à faire (« e_star ») pour identifier ce nœud à une classe de logs. Par défaut les comparaisons sont faites par égalité. Le pair champ/value (« field » et « value ») in-

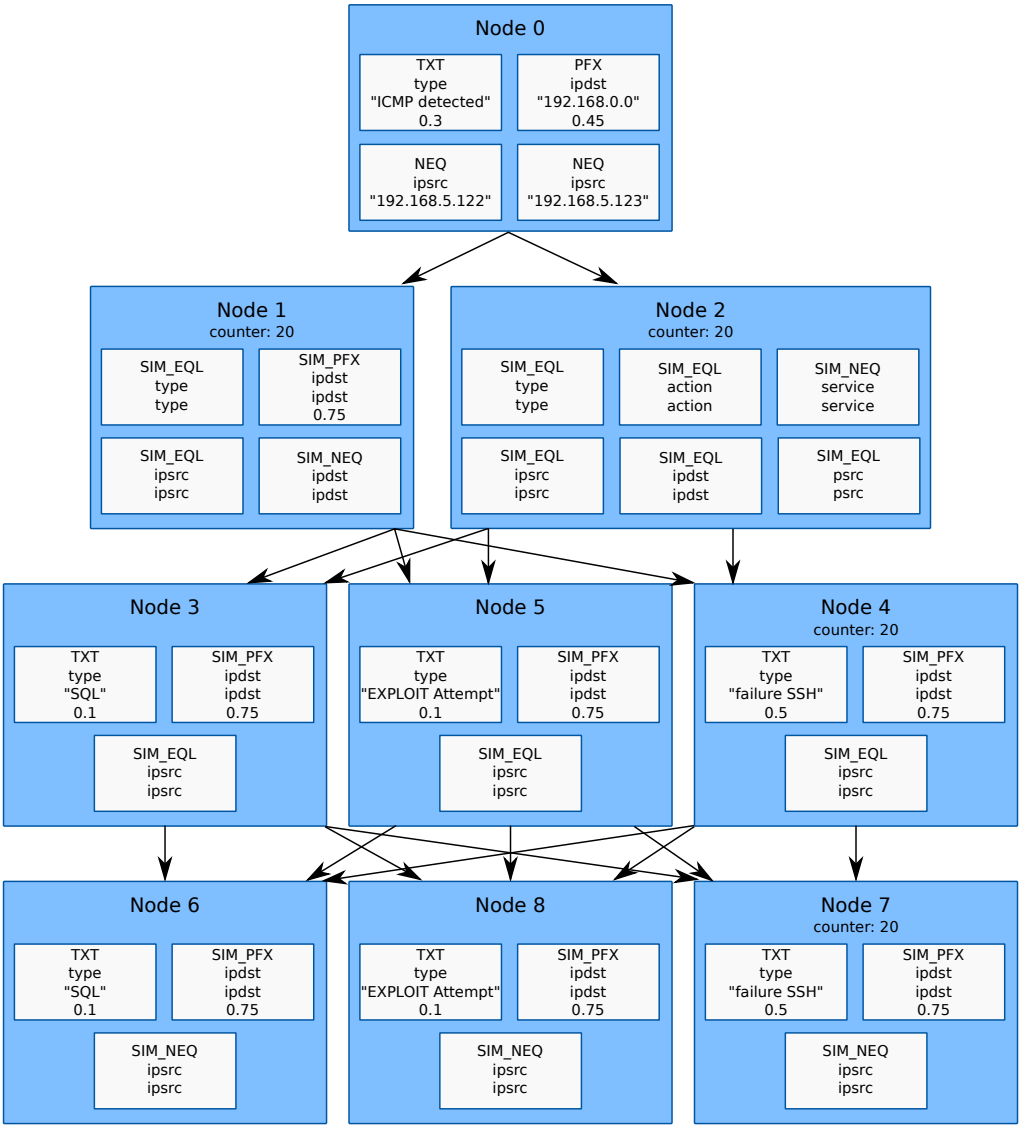


FIGURE 6 – AASG S2A (représentation fonctionnelle).

dique que tous les logs pour lesquels le champ indiqué dans le nœud (« type », par exemple) a la valeur indiquée dans le nœud (« Stream_DoS », par exemple) se correspondent à ce nœud.

- Une liste des fils du nœud (« children ») dans la structure de l'arbre.
- Un niveau de phéromones initial (« ph ») qui est utilisé par l'algorithme Morwilog (section 6).

FIGURE 7 – Exemple AASG dans le format JSON.

On montre un exemple d'un arbre simple (nœud racine et deux enfants) dans la Figure 7.

En utilisant ce format, l'analyste peut partager ses arbres avec d'autres analystes. Ce format est conçu pour pouvoir être interprété par l'humain mais surtout pour être facilement adopté par les outils d'analyse et détection.

5.5 Conclusions AASG

Pour conclure cette section sur les AASGs, on a développé une représentation abstrait pour exprimer les hypothèses de l'analyste qui contraste avec le caractère concret du modèle CASG, qui représente seulement une attaque multi-étape spécifique.

Ce modèle peut être utilisé pour faire de la détection, en lançant de proces-

sus de détection sur chacune des hypothèses contenu dans l'AASG. Et aussi pour faire de l'identification, pour signaler les hypothèses correcte trouvés, vu qu'on peut assigner des poids aux arrêtes de l'AASG pour enregistrer l'apprentissage sur la pertinence de chacune des hypothèses. Ils nous faut des algorithmes pour faire ces détection et identification. C'est pour ça que nous avons développé Morwilog et Bidimac, deux autre contributions de cette thèse, présenté dans la prochaine section.

6 Algorithmes pour l'exploitation d'AASG

Chacun des algorithmes développés suit une approach complètement différente. D'abord, Morwilog suit une recherche stochastique où seulement une branche (une hypothèse) est cherchée chaque fois. Il y aura donc des choix aléatoire des chemins dans l'AASG.

De son côté, Bidimac suit une recherche déterministe où tous les branches de l'AASG sont cherchées. Le statistique des sequences d'entré est donc préservée.

6.1 Morwilog

Morwilog est basé sur l'optimization par colonie de fourmis (ACO), une metaheuristique d'optimization inspirée par le comportement des fourmis quand elles cherchent de la nourriture. Une colonie des fourmis converge toujours au chemin le plus court vers la nourriture grâce au dépôt de phéromones (Figure 8). Cette idée est directement pris par ACO, où les plus haut concentration de pheromones corresponds avec le chemin le plus court dans un graphe d'optimisation. Mais en Morwilog, l'increment de pheromones va signaler les branches dans l'AASG qui sont trouvés et confirmés par l'analyste.

Plus précissement, Morwilog est basé sur le moteur des hommilières (manhill) développé au sein d'ICube. L'idée principale est la génération de fourmis artificielles après l'arrivage d'un log au système. Ces fourmis s'appellent « morwis », nom qui vient du préfix Proto Indo-Européen pour fourmi.

Le comportement de ces morwis est basé sur le comportement des fourmis quand elles sont à la recherche de nourriture. Similairement, les morwis traversent les AASGs, qui représentent séquences d'évènements, à la recherche des hypothèses. Pour pouvoir utiliser l'AASG dans ce contexte il faut ajouter

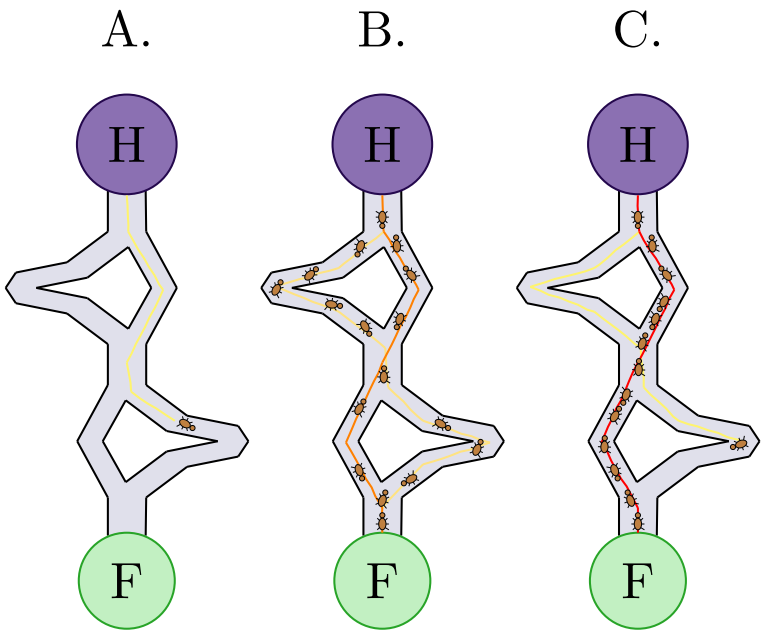


FIGURE 8 – Exemple de la convergence des fourmis au chemin le plus court.

le niveau de pheromones aux arrêtes. L'AASG resultant s'appelle AASG stigmergique. Et l'algorithme va faire évoluer ces niveaux dès que des hypothèses sont trouvées.

Quand un risque est identifié par un morwi après parcourir un AASG et l'expert humain confirme la détection, les morwis déposent des phéromones virtuelles dans les arbres, donc des autres morwis peuvent les suivre. Le niveau de phéromones artificiels déposé par les morwis dépend de cet analyse et va déterminer que chemin de l'arbre sera favorisé par rapport à d'autres.

Dans la Figure 10, on montre un exemple de fonctionnement de Morwilog avec un AASG précis, basé sur le médiatique attaque WannaCry. La correspondance entre chaque nœud et les étapes dans l'attaque sont montrées dans la Figure 9.

Les pas suivis par l'algorithme sont les suivantes :

1. Chaque fois qu'un événement suspicieux arrive à Morwilog, une morwi est générée et elle cherche l'arbre d'événements avec un nœud racine qui se correspond (fait du « match ») avec l'événement. Si aucun arbre a été trouvé, a nouveau arbre avec une séquence d'actions aléatoire est créé.

Level	Node	Matched event	Origin
0	κ_0	Unsuccessful HTTP request from a host, called A, to long domain name	Proxy
1	κ_1	Successful connection in port TCP 445 from A to any another endpoint (B) in the local network	Internal firewall
1	κ_2	Successful HTTP request to the same domain name as in κ_0	Proxy
2	κ_3	SMBv1 communication between A and other machine using command 'transaction2_secondary'	Internal Firewall
2	κ_4	Malformed SMB headers for the NEGOTIATE PROTOCOL REQUEST from A to B (CVE-2009-3103)	Endpoint B
2	κ_5	SQL Injection alert coming from A	IDS

FIGURE 9 – Événements dans le hypothèses sur WannaCry.

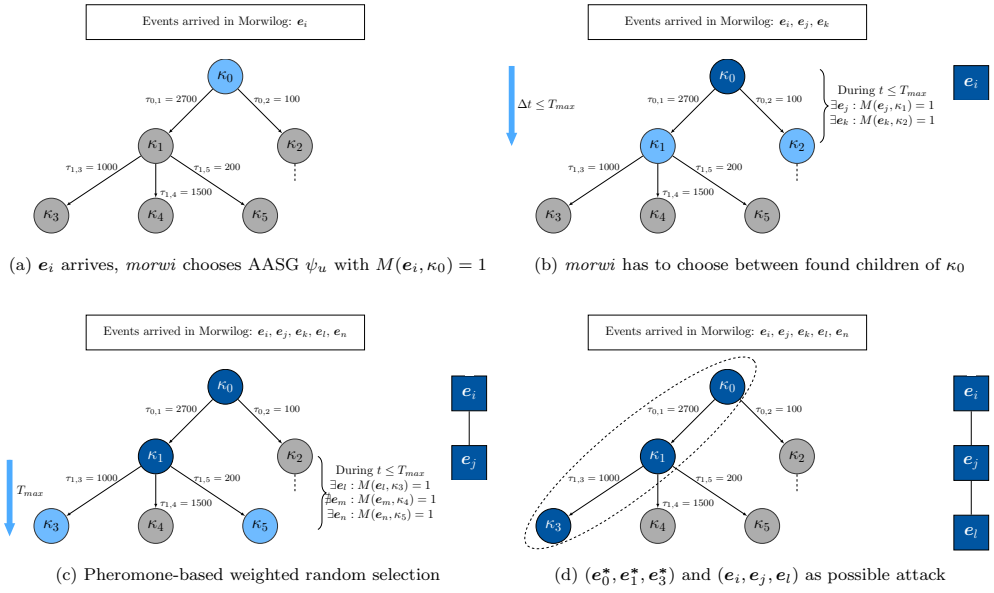
2. La morwi commence le parcours de l'arbre dans le nœud racine et elle cherche des subséquents événements qui correspond avec les nœuds du niveau deux de l'arbre. Le temps de recherche des événements est limité pour ne pas saturer le système d'exploitation. Dans cet exemple, les deux événements e_j et e_k qui correspondent avec les nœuds k_1 et k_2 ont été trouvés. Maintenant, la morwi doit décider un d'entre ces nœuds pour continuer son voyage dans l'arbre. La sélection est faite en suivant une sélection aléatoire pondéré par les poids des phéromones dans chaque chemin.
3. Dans cet exemple, l'option la plus probable est k_1. Imaginez que la morwi choisit ce nœud. Maintenant elle doit répéter le même procédé pour le niveau suivant. Le procédé est répété jusqu'à la morwi arrive sur un nœud sans enfants.
4. Une fois le parcours est fini, la morwi rendre comme résultat la séquence complète choisie si le taux de phéromones est plus grand qu'un certain seuil fixé.

Le prochain pas est l'évaluation de l'expert en sécurité de la séquence proposée par l'algorithme. L'expert va étudier la séquence d'événements et va indiquer au système s'il s'agit ou pas d'une attaque. Si c'est une attaque, le taux de phéromones de tout le chemin est incrémenté (voir Figure 11). Si ce n'est pas considéré comme une attaque, il y a un décrément du niveau de phéromones (voir Figure 12).

L'evolution de pheromones chaque fois qu'une alarme est évaluée se fait par deux pas. D'abord, il y a un processus d'évaporation dans tous les arrêtes de l'AASG :

$$\tau[t+1] = (1-\rho) \cdot \tau[t] \tag{1}$$

Après, il y a l'increment ou le décrément dans les arrêtes qui sont dans la



(a) e_i arrives, *morwi* chooses AASG ψ_u with $M(e_i, \kappa_0) = 1$

(b) *morwi* has to choose between found children of κ_0

(c) Pheromone-based weighted random selection

(d) (e_0^*, e_1^*, e_3^*) and (e_i, e_j, e_l) as possible attack

FIGURE 10 – Exemple de fonctionnement de Morwilog.

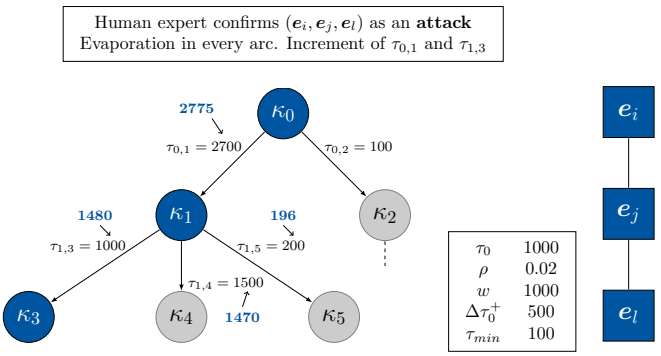


FIGURE 11 – Exemple d'évolution de l'AASG si la séquence est confirmée comme attaque.

branche qui a généré l'alarme ($\Delta\tau^+ = -\Delta\tau^-$) :

$$\Delta\tau^+(\tau[t]) = \Delta\tau_0^+ e^{-\frac{(\tau[t]-\tau_0)^2}{2w^2}} \tag{2}$$

Les formules pour les deux opérations donne les paramètres de Morwilog, auxquelles il faut ajouter le temps maximum de recherche et le niveau minimum

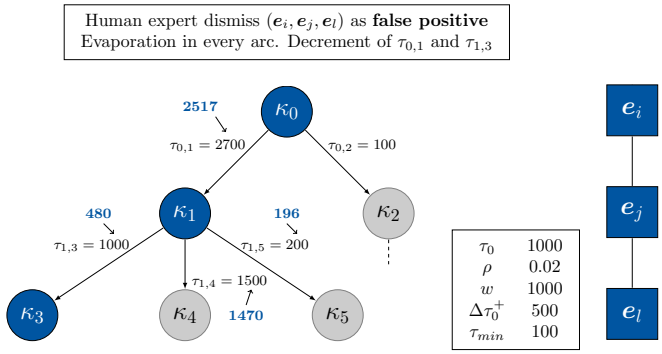


FIGURE 12 – Exemple d'évolution de l'AASG si la séquence n'est point confirmée comme attaque.

Name	**Description**	**Ref. value**
T_{max}	Maximum search time	62 s
τ_0	Initial number of pheromones	1000
ρ	Evaporation rate	0.02
w	Spreading of $\Delta\tau^{+,-}$ function	1000
$\Delta\tau_0^+$	Change of pheromones when $\tau[t] = \tau_0$	500
τ_{min}	Minimum level of pheromones	100

TABLE 3 – Paramètres de Morwilog

de pheromones (vois Table 3). Ce niveau sert à empêcher les pheromones d'aller aux valeurs negatives.

En ACO classique, l'increment et decrement est une quantité fixe. Nous avons décidé d'implementer l'increment/decrement par une fonction Gaussiane pour forcer un changement plus rapide dans les moments initiaux.

Les conclusions de Morwilog est qu'on peut faire de la détection des hypothèses, parce qu'une alarme est lancé quand des séquences qui correspond au branches sont trouvés. Et on peut aussi faire de l'identification, parce que le pheromones vont s'incrémenter pour le cas trouvés et confirmés par l'analyste.

Comme c'est le premier algorithme développé pour travailler dans la détection et identification des hypothèses alternatives d'attaques multi-étapes, on n'a pas de références pour comparer. Ça nous a poussé à développer Bidimac, basé sur l'inference Bayésienne.

6.2 Bidimac

Mais pour quoi l'inference Bayésienne ? D'abord, c'est une méthode assez étendue et bien étudié. Mais aussi la définition de l'AASG stigmergique correspond avec la logique Bayésienne. Quand on prend un AASG et on le transforme dans une AASG stigmergique, les niveaux de pheromones sont toujours initialisés au même valeurs, τ_0, un des paramètres de Morwilog.

On peut définir la probabilité de choix comme la probabilité de choisir un nœud depuis son parent quand tous les enfants ont été trouvé. Si on change le niveau de pheromones par la probabilité de choix, on obtient directement un modèle Bayésien. On a donc l'AASG stigmergique, où les pheromones sers à sélectionner aléatoirement un chemin, et l'AASG Bayésien, où les paramètres des probabilités vont refléter le statistic du système et tous le branches sont cherchées.

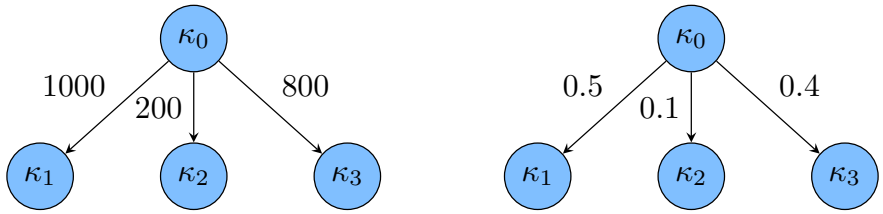
En Bidimac, le AASG Bayésien va être divisé dans tous ses branches et une séquence d'événements qui correspond à chacune des branches va être cherchée. Tous les branches trouvés vont générer des alarmes pour l'analyste. Une fois les alarmes sont confirmées, tous les paramètres de probabilités de l'AASG Bayésien vont évoluer par rapport un certain taux d'apprentissage η.

6.3 Comparaison Morwilog et Bidimac

Les similarités entre Morwilog et Bidimac sont :

- Le deux suivent la logique Bayésienne, où des probabilités de parcours de l'arbre sont connues a priori. Mais la méthode Bayésienne, comme son nom l'indique, suit une actualisation des probabilités Bayésienne classique : la probabilité associée à chaque lien est obtenue à partir de l'observation directe des séquences des événement. Morwilog par contre fonctionne à partir de la métaphore biologique des fourmis artificielles, et incorpore des mécanismes comme l'évaporation de phéromones ou l'élection des chemins qui ne sont pas supportés par la statistique des modèles Bayésiennes.
- Le deux utilisent des Graphes Dirigés et Acycliques (DAG, Directed Acyclic Graphs) pour représenter les relations entre les différents événements.
- Les nœuds dans les DAGs sont liés par des lien avec une probabilité

associée. La probabilité dans Morwilog est codée sous forme de phéromones (Figure 13).
- Les graphes résultant de l'exécution des deux méthodes donnent de l'information sur la distribution statistique des séquences représentant des attaques possibles.



(a) AASG stigmergique (phéromones) (b) AASG Bayésien (param. prob.)

FIGURE 13 – Comparaison entre un AASG utilisé par Morwilog et un AASG Bayésien, qui codent la même information

Même s'il existe des similitudes, les différences sont remarquables :

- **Choix entre les options**. En Morwilog, quand deux ou plusieurs séquences avec des événements en commun correspondent à un AASG, une seule option est choisie dans chaque étape. Dans la méthode Bayésienne, toutes les séquences trouvées vont contribuer à la mise à jour des probabilités dans l'AASG et il n'y a pas de processus d'élection.
- **Phéromones pour l'évaluation des choix**. Le niveau de phéromones ne correspond pas exactement à la probabilité d'apparition d'une séquence dans le set de données, même s'il peut nous en donner une idée approximative. Les phéromones fonctionnent pour le choix de chemins par les fourmis à chaque bifurcation.
- **Mécanisme pour pénaliser ou renforcer des chemins**. Dans Morwilog, il y a un mécanisme pour incrémenter ou décrémenter spécifiquement les phéromones d'une séquence. Pour la méthode Bayésienne le changement des probabilités est basé seulement sur la statistique de l'attaque.
- **Décrément de probabilité dans tous les liens**. Après chaque phase d'exécution de Morwilog il y a une phase d'évaporation pour décrémenter le niveau des phéromones sur tous les liens. En Bayésien on n'a pas ce mécanisme. Ça apporte une convergence plus rapide vers le chemin correct entre tous les hypothèses proposées par l'expert et évite la stagnation de l'algorithme.

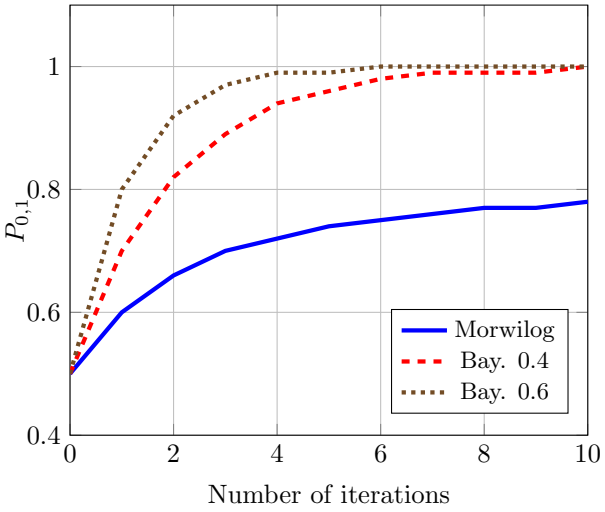


FIGURE 14 – Évolution de la probabilité d'élection.

On a défini la probabilité d'élection comme la probabilité assignée à un chemin de l'AASG. Dans le cas de Morwilog, la probabilité dépend des phéromones, et on peut la calculer en divisant le niveau de phéromones du chemin par l'addition des niveaux de phéromones de tous les chemins partant du même nœud (celui-ci compris). Si on fait une étude théorique de l'évolution de la probabilité d'élection dans les deux méthodes, on constate que Morwilog nous donne plus de liberté pour faire converger l'algorithme au chemin de l'arbre qui représente l'attaque. Si seulement une entre les hypothèses proposées est correcte, l'expert va être notifié plus rapidement avec Morwilog qu'avec la méthode Bayésienne (Figure 14).

7 Évaluation

Les résultats de l'évaluation des AASGs et les algorithmes a été soignoisement présentée dans la thèse et on ne va pas reproduire ici les résultats. La conclusion principale est que le choix entre Morwilog et Bidimac depend de l'application, des nombres des événements attendu et de l'expectative de l'analyste par rapport aux AASGs construits.

La détection a été évalué sur trois datasets : ISCX (développé par UNB), HuMa (le dataset développé au sein du projet) et DARPA 2000 (le dataset classique pour la détection d'attaques multi-étapes). Comme ces datasets contiennent seulement une instance d'attaque multi-étape et on a besoin de

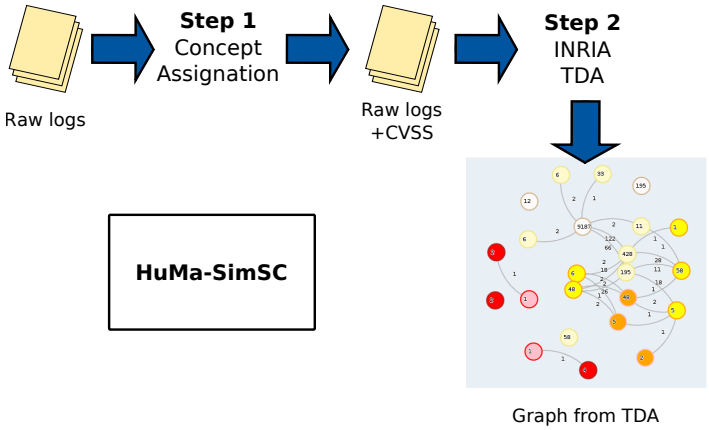


FIGURE 15 – Intégration de SimSC dans l'architecture HuMa.

faire évoluer les AASG, pour l'identification on a généré un set de datasets artificielles en utilisant Splunk Event Generator.

8 Investigation visuel des scénarios d'attaques

Une autre contribution développé dans le cadre de cette thèse est SimSC, un modèle qui sert pour l'investigation visuel des scénarios d'attaques.

8.1 SimSC pour HuMa

La version originale de SimSC travaille comme un « zoom » sur les nœuds des graphes crées par TDA, un des autres outils du projet HuMa, le projet qui a financé cette thèse. Ces graphes ont un ensemble de logs représentés dans chaque nœud. Deux nœuds sont liés s'ils ont au moins un log en commun.

Si l'analyste veut mieux explorer le contenu d'un nœud, il ou elle va le sélectionner et lancer l'outil SimSC (Figure 15). La sortie de SimSC est un graphe de comportement où chaque nœud représente un seul log.

On résume ici le processus suivi par SimSC pour la création de ces graphes de comportement. Deux logs sont liés s'ils ont un certain nombre d'adresses IP en commun et que la distance temporelle entre eux est sous un certain seuil. Ces relations entre logs permettent à l'expert de se focaliser dans des logs similaires, qui appartiennent au même scenario. Les logs qui appartiennent

au même scenario ont typiquement des adresses IP en commun et ils sont temporellement proches.

SimSC a été conçu pour tenir en compte des autres relations entre les attributs des logs pour faire les liens entre les logs. Cependant, pour travailler avec les résultats de HuMa on ne peut qu'utiliser les adresses IP et le temps de chaque log, parce que les logs sont présentés en format de texte, sans aucune différentiation entre les champs des logs. Ceci est expliqué par le fait que les logs sont des logs massifs (Big Data) desquels on ne connaît pas son format spécifique.

L'implémentation de SimSC fonctionne de la façon suivante quand il traite les graphes provenant de l'algorithme TDA :

- D'abord, on extrait automatiquement la liste des adresses IP présentes dans le log. On peut les extraire facilement car elles suivent un format normalisé X.X.X.X, avec X un octet. Par contre, il n'est pas possible de faire la distinction entre les adresses IP qui correspondent aux sources et à celles qui correspondent aux destinations.
- La date correspondant à la génération du log est aussi extraite. On cherche des formats de dates dans le log, on transforme en timestamp POSIX (millisecondes) et on prend la valeur minimum comme correspondant au timestamp du log. La valeur minimum permet d'identifier le plus vieux timestamp associé au log. On a fait la comparaison entre cette valeur minimum extraite automatiquement et le timestamp réel dans un jeu de logs étiqueté avec des timestamps corrects contenant plus de 3 million de logs. Le résultat a été toujours le même : le plus bas d'entre tous les timestamps automatiquement extraits du log correspond à la valeur du timestamp réel.
- Une fois que l'information est extraite depuis les logs, l'algorithme compare tous les logs entre eux et crée un lien entre deux logs si a) la différence entre 2 timestamps ne surpasse pas un certain seuil et b) s'il y a au moins un certain nombre d'adresses IP en commun dans la liste. Les métriques basées sur la différence de timestamp et la similitude entre des adresses IP sont les métriques les plus utilisés dans la reconnaissance des attaques multi-étapes (plus de détails dans l'article A Systematic Survey on Multi-step Attack Detection, publié dans le cadre de cette thèse et référencé à la fin de ce document).

TDA a pour but de générer des représentations agrégées à partir de la projection des logs sur de nombreuses dimensions. Comme les logs sont de pièces d'information multi-dimensionnels (ils sont plusieurs champs ou attributs), ils

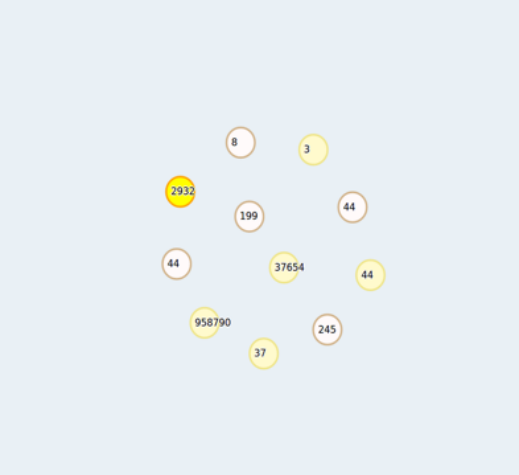


FIGURE 16 – Graphe fourni par l'algorithme TDA et utilisé dans l'exemple de SimSC.

peuvent être représentés comme vecteurs. Chaque nœud du graphe d'entrée représente un cluster de vecteurs généré par une fonction filtrage appliquée sur les logs. La fonction filtrage est basée sur le score CVSS associé à chaque log. L'étiquette CVSS (Common Vulnerability Scoring System) a été assignée aux logs dans une phase de prétraitement antérieur en fonction de la dangerosité de chacun par rapport à la potentielle exploitation de vulnérabilités. Les liens entre les nœuds sont construits lorsque deux nœuds possèdent un ou plusieurs vecteurs en commun. Cette extraction est illustrée avec un exemple de graphe fourni par TDA. On utilise le graphe de la Figure 16. Le chiffre contenu dans chaque nœud indique la quantité de logs qui y est associée. Deux nœuds sont connectés s'ils ont des logs en commun, mais dans cet exemple il n'y a pas de logs connectés.

L'algorithme SimSC est maintenant appliqué sur chacun des nœuds du graphe de la Figure 16. Pour visualiser les résultats nous avons utilisé une interface graphique basée sur une application web, développée en JavaScript, HTML et Python. C'est une plateforme interactive dans laquelle l'expert peut cliquer sur les nœuds des graphes pour afficher l'information des logs et même entraîner les nœuds pour explorer la structure des scenarios de comportement. Les couleurs assignées aux nœuds sont choisies en fonction de l'étiquette CVSS qui correspond à chaque log. Visuellement il donne à l'analyste une idée des logs plus dangereux.

L'analyste peut analyser le contenu de chaque nœud du graphe issue de l'algorithme TDA en cliquant sur lui et en appliquant l'algorithme SimSC sur

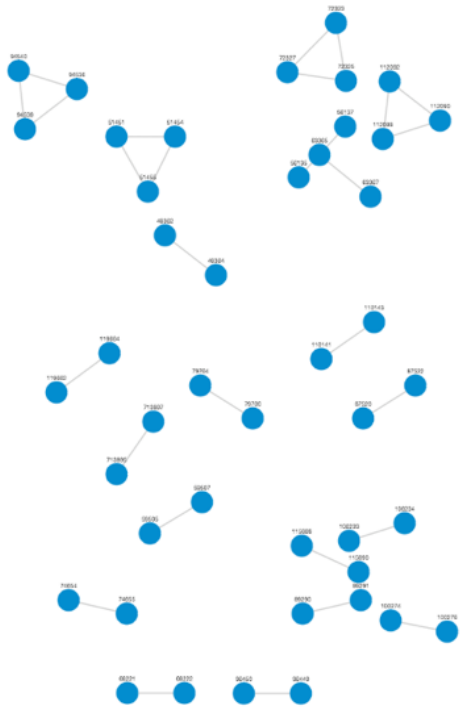


FIGURE 17 – SimSC sur le nœud 4 de la Figure 16.

les adresses IP et les timestamps. Le bénéfice de l'analyse SimSC par rapport à l'analyse faite avec TDA est que maintenant l'analyste va avoir les détails de chaque log individuel et sa connexion avec des autres logs en termes d'adresses IP et de différence temporelle.

Quelques exemples de scénarios d'usage anormaux découverts par l'analyse sont maintenant présentés. On considère 2 nœuds représentatifs du graphe de la Figure 16, qui a 11 nœuds :

- **Nœud 4** (Figure 17) : Il contient des logs réguliers générés par le daemon CROND périodiquement. Son agrégation en couples ou trio est très particulière et donne à l'analyste une vision du temps dans lequel les tâches sont exécutées. Comme ils sont des tâches périodiques, les liens sont créés seulement quand la différence de temps entre les logs surpasse certain seuil et pour tous les logs au même temps. Par exemple, dans le graphe de la Figure 17, fait avec une différence de temps maximum pour construire chaque lien de 59 secondes, on a observé que si on monte la différence de temps à 60 secondes, tous les logs sont liés par des liens. Ça veut dire que la différence de temps entre les logs suit un pattern établi entre chaque couple de logs.

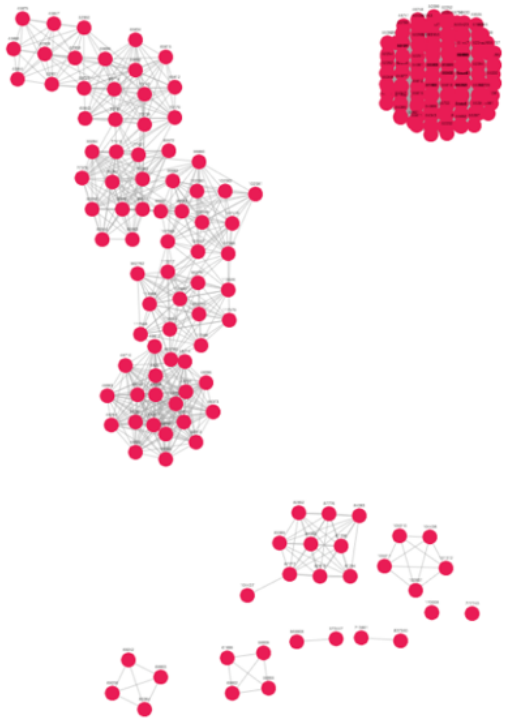


FIGURE 18 – SimSC sur le nœud 6 de la Figure 16.

- **Nœud 6** (Figure 18) : Le cluster des logs que l'on voit indique un scan : l'adresse IP source est la même tout le temps et il y a une grande quantité de paquets de connexion en direction du même serveur dans un espace de temps réduit, en changeant les numéros de ports. Un scan est utilisé par un attaquant pour trouver de l'information nouvelle sur les systèmes à attaquer. Le plus intéressant est que le fournisseur de ces données n'avait pas prévenu de ce scan, et on l'a trouvé grâce à SimSC.

8.2 Logan : une adaptation de SimSC

Le laboratoire ICube a aussi développé Logan, une implémentation de SimSC qui démontre les capacités de ce modèle. Le but de cette implémentation est de tester SimSC, développé pour HuMa, dans un environnement réel afin de supporter les tests à faire par les partenaires industriels. L'interface montre les résultats de SimSC appliqué à l'analyse des logs Web générés par Genida, une plateforme de centralisation, stockage et coordination de donnés de cohortes de maladies génétiques rares développée au sein d'ICube.

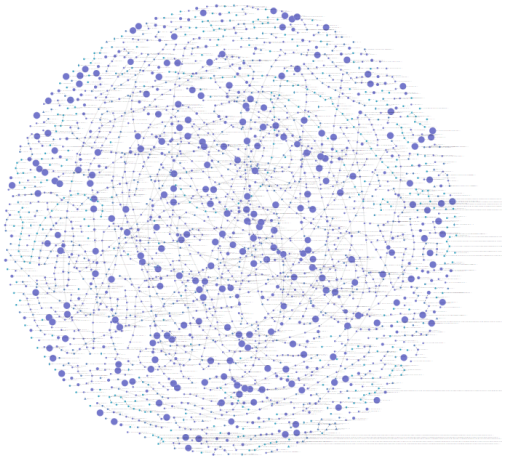


FIGURE 19 – Vue globale d'une analyse de SimSC dans l'implémentation Logan.

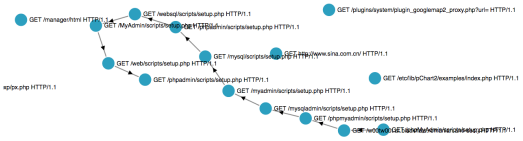


FIGURE 20 – Test de scripts par un utilisateur malveillant détecté en utilisant Logan (SimSC)

La plateforme est complètement opérationnelle pour l'analyse des logs issus de l'application. L'analyste doit sélectionner la période de temps à afficher et SimSC réalise l'analyse sur cette période. Le résultat est un graphe interactif où les nœuds représentent des logs, qui sont liés pour former les séquences de comportement des utilisateurs. Si on montre tous les logs au même temps dans le graphe, on obtient un graphe assez difficile à analyser, comme on peut voir dans la Figure 19.

Mais, en filtrant les logs de haut risque (code http 400-500), l'analyste peut facilement détecter les tentatives d'intrusion. Si on ne prend qu'un type de logs, on peut facilement les chemins qui décrit le comportement d'un utilisateur. Ça permet à l'analyste d'identifier visuellement des attaques comme des injections SQL ou des tentatives d'exécution de scripts (Figure 20). Les attaques trouvées sont transmises aux développeurs du système de stockage des données pour prévoir les vecteurs d'attaque possibles lors du développement.

9 Conclusion

Le problème qu'on a affronté dans ce thèse est la non consideration dans le modèle classique de détection d'attaques multi-étape des hypothèses posés par l'analyste. Nous avons proposé pour ça un modèle où l'analyste est placé dans le centre pour modéliser ses hypothèses sur les attaques multi-étapes et pouvoir les détecter et identifier les hypothèses correctes.

L'étude bibliographique, un des contributions de cette thèse car elle est la première étude complète fait sur la détection d'attaques multi-étape, nous a montré que dans le modèle classique, on doit délivrer des modèles de détection définitifs.

Nous avons développé une série d'outils pour donner à l'analyste la possibilité de proposer des modèles intermédiaire comme des hypothèses. D'abord, une structure, l'AASG, pour les modéliser. Ensuite, deux algorithmes, Morwilog et Bidimac, pour faire de la détection et de l'identification des hypothèses sur les AASGs.

Avec ce nouveau modèle, une fois la présence d'une attaque comme WannaCry, la pire attaque que les réseaux informatiques ont souffert, est découverte, l'analyste peut commencer depuis sa réception à coder ces hypothèses dans un ou plusieurs AASG. Ces AASGs pourront être envoyer à des algorithmes comme Bidimac ou Morwilog pour la détection et l'identification des hypothèses correctes et, entre temps, l'analyste peut continuer son processus d'investigation sur l'attaque pour arriver à un modèle définitif de détection.

Publications

Articles de journal :

- Navarro, J. ; Deruyver, A. & Parrend, P. **A systematic survey on multi-step attack detection**. Computers & Security, Elsevier, 2018, 76, 214-249
- Navarro, J. ; Legrand, V. ; Deruyver, A. & Parrend, P. **OMMA : open architecture for Operator-guided Monitoring of Multi-step Attacks**. EURASIP Journal on Information Security, Nature Publishing Group, 2018, 2018, 6
- Parrend, P. ; Navarro, J. ; Guigou, F. ; Deruyver, A. & Collet, P. **Foundations and applications of artificial Intelligence for zero-day**

and multi-step attack detection. EURASIP Journal on Information Security, Nature Publishing Group, 2018, 4

Articles de conférence :

- Parrend, P. ; Guigou, F. ; Navarro, J. ; Deruyver, A. & Collet, P. **For a refoundation of Artificial Immune System research : AIS is a Design Pattern**. IEEE Symposium Series on Computational Intelligence (SSCI), 2018, 1122-1129
- Parrend, P. ; Guigou, F. ; Navarro, J. ; Deruyver, A. & Collet, P. **Artificial Immune Ecosystems : the role of expert-based learning in artificial cognition**. Journal of Robotics, Networking and Artificial Life, 2018, 4, 303-307
- Navarro, J. ; Legrand, V. ; Lagraa, S. ; François, J. ; Lahmadi, A. ; De Santis, G. ; Festor, O. ; Lammari, N. ; Hamdi, F. ; Deruyver, A. & others. **HuMa : A multi-layer framework for threat analysis in a heterogeneous log environment**. International Symposium on Foundations and Practice of Security, 2017, 144-159
- Navarro-Lara, J. ; Deruyver, A. & Parrend, P. **Morwilog : an ACO-based system for outlining multi-step attacks**. IEEE Symposium Series on Computational Intelligence (SSCI), 2016, 1-8

Références

[Caulkins 2018] Bruce Caulkins, Tiffani Marlowe and Ashley Reardon. *Cybersecurity Skills to Address Today's Threats*. In International Conference on Human Factors in Cybersecurity, pages 187–192. Springer, 2018.

[CrowdStrike 2018] CrowdStrike. *Global Threat Report : Blurring the Lines Between Statecraft and Tradecraft*, 2018.

[Ehrenfeld 2017] Jesse M. Ehrenfeld. *Wannacry, cybersecurity and health information technology : A time to act*. Journal of medical systems, vol. 41, no. 7, page 104, 2017.

[Fire Eye 2018] Fire Eye. *M-Trends 2018*, 2018.

[Navarro 2018] Julio Navarro, Aline Deruyver and Pierre Parrend. *A systematic survey on multi-step attack detection*. Computers & Security, vol. 76, pages 214–249, July 2018.

[Sundaramurthy 2015] Sathya Chandran Sundaramurthy, Alexandru G. Bardas, Jacob Case, Xinming Ou, Michael Wesch, John McHugh and S. Raj Rajagopalan. *A Human Capital Model for Mitigating Security Analyst Burnout*. In Symposium on Usable Privacy and Security (SOUPS), pages 347–359, 2015.

[Trustwave 2018] Trustwave. *Global security report*. Technical report, Trustwave, 2018.

[Zargar 2014] Saman T. Zargar. *ONTIDS : A highly flexible context-aware and ontology-based alert correlation framework*. In Foundations and Practice of Security : 6th International Symposium, FPS 2013, La Rochelle, France, October 21-22, 2013, Revised Selected Papers, volume 8352, page 161, 2014.