Metrics for security activities assisted by argumentative logic

Tarek Bouyahia

IMT Atlantique
Bretagne-Pays de la Loire
École Mines-Télécom

UNIVERSITE BRETAGNE LOIRE

THÈSE / IMT Atlantique

sous le sceau de l'Université Bretagne Loire

pour obtenir le grade de

DOCTEUR DE IMT Atlantique

Mention : Informatique

École Doctorale Matisse

Présentée par

Tarek Bouyahia

Préparée au département Logique des Usages, Sciences Sociales et de l'Information
Laboratoire Labsticc

Métriques pour le déclenchement des évènements de sécurité assistées par la logique argumentative

Metrics for security activities assisted by argumentative logic

Thèse soutenue le 29 mars 2017

devant le jury composé de :

Mohamed Mosbah
Professeur, Institut Polytechnique de Bordeaux / Président

Joaquin Garcia-Alfaro
Professeur, Université de Paris-Saclay – Evry / Rapporteur

Zonghua Zhang
Maître de conférences (HDR), IMT Lille Douai - Villeneuve - d'Ascq / Rapporteur

Fabien Autrel
Ingénieur de Recherche, IMT Atlantique / Examinateur

Frédéric Cuppens
Professeur, IMT Atlantique / Examinateur

Nora Cuppens
Directeur de Recherche, IMT Atlantique / Directrice de thèse

Abstract

The growth and diversity of services offered by modern systems make the task of securing these systems a complex exercise. On the one hand, the evolution of the number of system services increases the risk of causing vulnerabilities. These vulnerabilities can be exploited by malicious users to reach some intrusion objectives. On the other hand, the most recent competitive systems are those that ensure a certain level of performance and quality of service while maintaining the safety state. Thus, modern security systems must consider the user requirements during the security process.

In addition, reacting in critical contexts against an attack after its execution can not always mitigate the adverse effects of the attack. In these cases, security systems should be in a phase ahead of the attacker in order to take necessary measures to prevent him/her from reaching his/her intrusion objective.

To address those problems, we argue in this thesis that the reaction process must follow a smart reasoning. This reasoning allows the system, according to a detected attack, to preview the related attacks that may occur and to apply the best possible countermeasures.

On the one hand, we propose an approach that generates potential attack scenarios given a detected alert. Then, we focus on the generation process of an appropriate set of countermeasures against attack scenarios generated among all system responses defined for the system. A generated set of countermeasures is considered as appropriate in the proposed approach if it presents a coherent set (i.e., it does not contain conflictual countermeasures) and it satisfies security administrator requirements (e.g., performance, availability). We argue in this thesis that the reaction process can be seen as two agents arguing against each other. On one side the attacker chooses his arguments as a set of actions to try to reach an intrusion objective, and on the other side the agent defending the target chooses his arguments as a set of countermeasures to block the attacker's progress or mitigate the attack effects.

On the other hand, we propose an approach based on a recommender system using Multi-Criteria Decision Making (MCDM) method. This approach assists security administrators while selecting countermeasures among the appropriate set of countermeasures generated from the first approach. The assistance process is based on the security administrator decisions historic. This approach permits also, to automatically select appropriate system responses in critical cases where the security administrator is unable to select them (e.g., outside working hours, lack of knowledge about the ongoing attack). Finally, our approaches are implemented and tested in the automotive system use case to ensure that our approaches implementation successfully responded to real-time constraints.

Résumé

L'accroissement et la diversification des services offerts par les systèmes informatiques modernes rendent la tâche de sécuriser ces systèmes encore plus complexe. D'une part, l'évolution du nombre de services système augmente le nombre des vulnérabilités. Ces vulnérabilités peuvent être exploitées par des utilisateurs malveillants afin d'atteindre certains objectifs d'intrusion. D'autre part, un système de sécurité moderne est considéré comme étant un système compétitif s'il assure un certain niveau de performance et de qualité de service tout en maintenant l'état de sécurité. Ainsi, les systèmes de sécurité modernes doivent tenir compte des exigences de l'utilisateur au cours du processus de sécurité.

En outre, la réaction dans des contextes critiques contre une attaque après son exécution ne peut pas toujours remédier aux effets néfastes de l'attaque. Dans certains cas, il est essentiel que le système de sécurité soit en avance de phase par rapport à l'attaquant et de prendre les mesures nécessaires pour l'empêcher d'atteindre son objectif d'intrusion.

Pour faire face à ces problèmes, nous soutenons dans cette thèse que le processus de sécurité doit suivre un raisonnement intelligent qui permet au système, selon une attaque détectée, de prévoir les attaques qui peuvent se produire par corrélation et d'appliquer les meilleures contre-mesures possibles.

D'abord, nous proposons une approche qui génère des scénarios potentiels d'attaque qui correspondent à une alerte détectée. Ensuite, nous nous concentrons sur le processus de génération d'un ensemble approprié de contre-mesures contre les scénarios d'attaque générés. Un ensemble généré des contre-mesures est considéré comme approprié dans l'approche proposée s'il présente un ensemble cohérent (il ne contient pas des contre-mesures conflictuelles) et il satisfait les exigences de l'administrateur de sécurité (par exemple, la performance, la disponibilité). Nous soutenons dans cette thèse que le processus de réaction peut être considéré comme un débat entre deux agents. D'un côté, l'attaquant choisit ses arguments comme étant un ensemble d'actions pour essayer d'atteindre un objectif d'intrusion, et de l'autre côté l'agent défendant la cible choisit

ses arguments comme étant un ensemble de contre-mesures pour bloquer la progression de l'attaquant ou atténuer les effets de l'attaque.

D'autre part, nous proposons une approche basée sur un système de recommandation en utilisant une méthode multicritère de la prise de décision MCDM (Multi Criteria Decision Making). Cette approche assiste l'administrateur de sécurité lors de la sélection des contre-mesures parmi l'ensemble approprié des contre-mesures générées à partir de la première approche. Le processus d'assistance est basé sur l'historique des décisions de l'administrateur de sécurité. Cette approche permet également de sélectionner automatiquement des réponses appropriées du système dans les cas critiques où l'administrateur de sécurité est incapable de les sélectionner (par exemple, en dehors des heures de travail, par manque de connaissances sur l'attaque en cours). Enfin, notre approche est implémentée et testée dans le cadre des système automobiles afin de vérifier si les approches proposées satisfont bien les contraintes de temps réel.

Acknowledgement

First and foremost, I would like to express all my gratitude to my parents for their continuous support and encouragement. My undying gratitude goes to my wife, **Asma**, for her love, understanding and support. I wish to thank all members of my family to whom I dedicate this thesis.

I am forever grateful to my thesis supervisors, Professors **Nora Cuppens-Boulahia**, **Frédéric Cuppens** and Dr **Fabien Autrel** for priceless advice, invaluable guidance and for their support and encouragement during my Ph.D study and thesis research.

I am thankful to the members of my supervisory committee, the reviewers Professor **Joaquin Garcia-Alfaro** and Dr **Zonghua Zhang** for the time taken to review my thesis and to give advice to improve its content. Thanks a lot to Professor **Mohamed Mosbah** for being part of the jury. It is a great honor for me to have them evaluate my thesis.

Finally, I would like to thank all my friends and colleagues in IMT Atlantique with whom I passed pleasant moments for their encouragement and scientific support, **Anis**, **Mariem**, **Tarik**, **Reda**, **Said**, **Vivien**, **Patrick**, **Samiha**, **Eric**, **Khaoula**, **Yanhuang**, **Edwin**, **Thomas**, **Simon**, **Xiaoshu**. A special thank to **Lyes**, **Nada** and **Safaa** who supported me so much. They have all been my family in France.

Contents

CHAPTER 1

Introduction

1.1 Motivation

Designing a secure system has always been a complex exercise. In practice, much of the focus of designers and developers being on delivering a working system in the first place; on the other hand, security concerns have long been considered only in retrospect, especially after serious flaws are discovered. Security experts are thus generally confronted with an existing system, whose architecture might actually hamper the deployment of security mechanisms that would prevent the occurrence of the attacks they envision. From the embedded system viewpoint, enforcement of security requirements becomes even more challenging and more critical. These challenges stem from the tight relationship between architecture design and its functional, and non-functional requirements as well as their impact on one another. For instance, if the system architecture design changes or evolves, these requirements should meet the new architecture design objectives and choose the best countermeasure that can be applied in this specific context or situation. This is especially true in safety-critical systems such as automotive systems [Bar-El 2009, Ruddle et al. 2010], where attacks may be devastating, but where security functions overhead may also result in an absolutely useless system. In such a context, designing a secure system has always been a complex exercise. Indeed, security is a functionality that is difficult to specify and implement because it is not modular: modifications to one part of an application may interact strongly with the security properties of other parts of the same application.

On the other side, reacting in a critical context against an attack after his execution can not always mitigate the attack damages. In these cases it is essential to anticipate the attacker's intentions and to take precautionary measures to prevent the attacker from reaching his/her intrusion objective. For instance, we consider a single physical server hosting a set of services such as an HTTP server, an SSH server and a database server. An attacker can detect those services by scanning the open ports for example, then try to fingerprint those services to check if a known vulnerable version is running.

From the detection point of view, detecting the port scanning and fingerprinting can be used to formulate hypothesis on the future attacks the attacker may perform on those services and select appropriate countermeasures against the inferred attack scenario. However the attacker may decide to modify his/her intrusion objectives, because he/she does not have the tools to attack the detected services for instance. In such case the attacker may execute new attacks corresponding to a new intrusion objective. From the detection point of view, this means that the reaction plan inferred from the first attack is no longer valid and must be revised in the light of the newly detected attacks.

Moreover, an appropriate countermeasure should depend on the context in which the system is operating. For instance, when considering a database server in the company private network, the availability criterion should be favored during work time when employees are using the system database, but the performance criterion should be preferred outside of working hours when database backups are created. In less critical situations, the system can afford to prioritize countermeasures that ensure certain user requirements instead of taking strict measures that affect the service availability for example. For mobile systems, such as those which are present in vehicles, the environment in which they operate can evolve as they move and impact the reasoning process. We also claim that the security analysis should also play an important role with respect to convincing the designer of increasingly complex embedded systems of the consistency and exhaustivity of his reasoning and selection of security measures, at least with respect to the identified threats. The use of argumentative logic [Dung 1995] driven reasoning engine can help in dynamic enforcement of security mechanisms through the introduction of non-monotonic reasoning capabilities. This non-monotonic logic provides a smart reasoning that allows to reason on the cost of applying a countermeasure and to minimize the set of generated system responses given a detected attack. These capabilities open up the door to the dynamic selection and enforcement of security mechanisms performed statically only today.

1.2 Contributions

The challenges of modern security tools is to keep the system in a safe state while satisfying the system different requirements (e.g., maintaining the best possible level of performance and quality of service). Thus, we argue that the security process must follow a smart reasoning that allows the system, according to a detected attack, to prevent the related and potential attacks that may occur and to choose the best possible set of countermeasures. To meet this objective, we propose the following contributions:

- *Contribution 1.* We introduce an approach [Bouyahia et al. 2014] for efficient enforcement of security requirements, this approach is driven by argumentative logic (AL). It describes a structured collaboration and interrelationship between the system architecture design and security requirements to support the long-term needs of the system. The purpose of security activities assisted by argumentative logic is to bring into focus the key areas of concern, highlighting the decision criteria and security context for each system aspect that has direct or indirect value for a stakeholder.

- *Contribution 2.* In modern attacks, the attacker can execute several actions in order to make the execution of other actions possible until reaching a certain intrusion objective. For this purpose, we provide an efficient method [Bouyahia et al. 2015] allowing to instantiate actions hypothesis correlated to the detected malicious action. Doing so, security administrator becomes aware about the potential attacker's intentions, which provide a better system reaction against intrusion. Given an attack against a specific system, the best countermeasure to apply depends on the context in which the system is operating. For example, in the case of an automotive system, the fact that the vehicle is operating downtown or on a freeway changes the impact an attack may have on the system. Thus, the system must take into account the set of active contexts when generating system responses. For this purpose, we show how to improve the existing argumentation framework by defining the Contextual Value-based Argumentation Framework ($CVAF$) [Bouyahia et al. 2015] . $CVAF$ presents a dynamic framework that allows to consider the current set of active contexts while generating system responses against intrusion.

- *Contribution 3.* We propose a content-based recommendation approach [Bouyahia et al. 2016] using Multi-Criteria Decision Making (MCDM) for efficient security administrator assistance when selecting the appropriate countermeasures, given a specific attack scenario. To learn more about the security administrator way of reacting, we propose a learning module which provides an idea about the security administrator preferences and requirements according to his/her decisions historic. This approach considers the different effects a countermeasure could have on the system (e.g., performance, availability) as criteria to be considered when selecting the appropriate system responses. This approach permits also, to automatically select appropriate countermeasures in critical cases where the system security administrator is unable to select them by his/her self.

- *Contribution 4.* We apply our approaches on an automotive system as an example of a case study to explore the issues that can meet complex systems during the reac-

tion process. This use case illustrates the potential need for dynamic enforcement of security requirements to control the various security activities. We present some experimental results concerning the execution costs of our implemented approach. These results allow the evaluation of the approach in terms of performance and time required for the system to react against different attack scenarios detected at the same time. Based on these results, we show how our approach implementation successfully responded to real-time constraints, since the responses of critical systems, such as automotive system, must be instantly provided especially in critical contexts.

1.3 Organization of the dissertation

This dissertation is organized as follows:

Chapter 2 – System Response against Intrusion Detection: State of the Art – we discuss in this chapter research investigations and technologies aiming to assist system response against intrusion detection. It depicts the main results published in the field of reaction against intrusion, focusing on the proposed automated approaches.

Chapter 3 – Introduction to the Argumentation Logic – presents a general introduction for Argumentation logic. It defines Argumentation logic frameworks while highlighting works using this logic in the security field.

Chapter 4 – Context-aware System Response against Intrusion Detection – introduces a novel approach which uses an argumentative logic framework to reason and select the most appropriate countermeasures given an attack and its context. This approach allows also to anticipate the attacker's intentions.

Chapter 5 – Multi-Criteria Recommender Tool for Supporting Intrusion Response System – proposes an approach based on content-based recommendation for efficient security administrators assistance in the context of reaction against intrusion detection.

Chapter 6 – Implementation and Evaluation – presents some experimental results concerning the execution costs of our implemented approaches. In this chapter, we consider the automotive system as a case study to evaluate the implementation of our approaches in an embedded system where real-time constraints must be satisfied.

Chapter 7 – Conclusions and Perspectives – this Chapter concludes the dissertation by summarizing the contributions and presenting the perspectives for future work.

CHAPTER 2

Automated System Response against Intrusion: State of the Art

2.1 Introduction

Recent security concerns related to future computer systems make enforcement of security requirements one of the most critical phases when designing such systems. Traditionally, reasoning about the best intrusion response to apply has always been a part of the security administrator responsibilities. In recent years, attackers changed their way to infiltrate computer systems and use more sophisticated attacks to reach their intrusion objectives. To cope with such modern attacks, Intrusion Response Systems (IRSs) must provide modern techniques that can maintain monitored systems in safe conditions while causing the minimum damage. In this chapter, we discuss the IRSs architectures and functionalities. We specify their corresponding characteristics, descriptions and existing approaches. Finally, we broach Attack Description Languages and we discuss their benefits and drawbacks in the attack modeling field.

2.2 Intrusion Response System Definition

Existing Intrusion Response Systems (IRS) are mainly divided into three approaches [Stakhanova et al. 2007b]: notification systems, manual response systems and automatic response systems.
Notification systems represent the majority of IRS. They are systems that just inform the security administrator about detected intrusions by generating reports and alarms

like Snort [Roesch 1999]. These systems require that the security administrator has a special knowledge about the various threats in order to select appropriate countermeasures.
Manual response systems are systems that notify the security administrator about ongoing detected intrusion and assist him/her while selecting system responses. These systems provide response alternatives to the security administrator, these alternatives are a preprogrammed set of responses corresponding to the reported attack.
Automatic response systems are systems that immediately provide a response to the detected intrusion. These systems do not need a human interaction.
From the security administrator's point of view, the main metric that differentiates these three approaches is the delay between the intrusion detection and the system response selection. In notification and manual system, this delay can be extended to hours and days (i.e., in week-end and outside of working hours when no security administrator is available). Even when an intrusion occurs during working hours, the time required for the security administrator to reason about the best system responses to apply provides a window of opportunity for the attackers, especially in notification systems where no assistance is provided. The author in [Cohen 1999] presents a study concerning the impact of reaction delay on the attack success rate. This study is based on simulations and shows that for ten hours of delay between intrusion detection and response, the attack success rate is 80%. This success rate increases to 95% when the response delay is twenty hours. For thirty hours as response delay, the attacker never fails to achieve his/her intrusion objective.

Now that we highlighted the importance of time delay between the intrusion detection and the system response on the efficiency of an IRS, we argue that Automated IRSs are by far the most suitable approaches for designing an efficient Intrusion Response System.

2.3 Automated Intrusion Response Systems (AIRSs)

AIRSs are especially exploited when designing a critical secure system where responses against intrusion must be provided in real-time. For instance, delayed system responses against intrusions in automotive systems are not acceptable, especially in critical contexts.
Automated intrusion response systems can be classified according to four characteristics as shown in Figure 2.1:

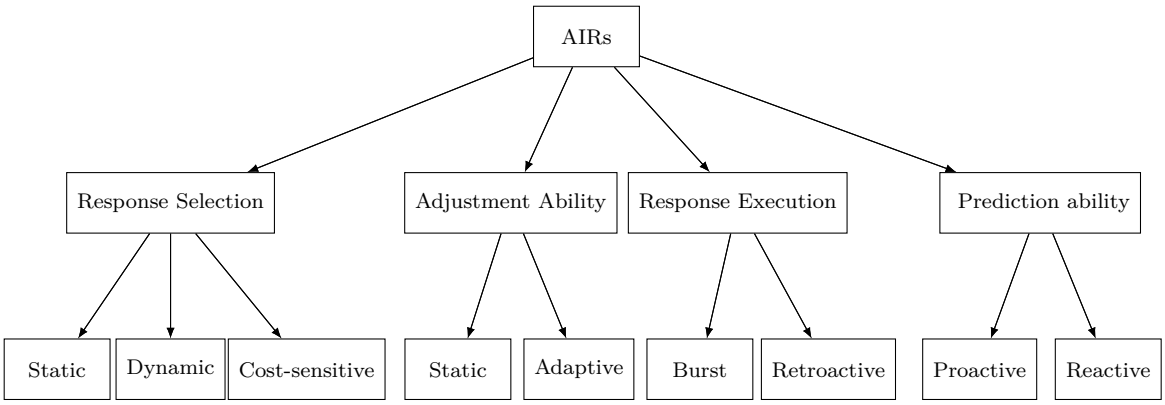


Figure 2.1: Taxonomy of Automated Intrusion Response Systems

2.3.1 Response Selection

We can classify AIRSs into three types of response selection models:

Static model

In static models, alerts are assigned to predefined countermeasures. In [Bruschi and Rosti 2001], the authors propose a Linux kernel module able to detect attacks through signatures. This module analyzes system calls to recognize attacks performed by the monitored host to block them. For instance, the module can detect that a process is launched with a shellcode as parameter and decide to stop the process. The authors observe that this type of automatic response is aggressive: a legitimate user, unconscious that his/her host is infected can be blocked. The authors in [Chen and Yang 2004] proposed a framework called Attack-Response Matrix (ARM) whose role is to allow policies to dictate actions that must be taken given a specific detected attack. This framework maps attack types to system responses. Static mapping models are the majority of existing approaches. The main limitation of such approaches is that the system responses are predictable, thus attackers can consider system counter measures while planning their attack scenarios. In addition, static model approaches do not take into account the different security administrator requirements and the costs of selecting countermeasures.

Dynamic model

Response selection in dynamic models is based on multiple factors, including system state (e.g., existing vulnerabilities, service implications), attack metrics (e.g., severity, confidence, frequency) and administrator security requirements (e.g., security policy

constraints, response goals). In other words, dynamic models are approaches where the system response against an attack is not always the same and it depends on multiple factors. In [Kiriansky et al. 2002], the authors propose a dynamic code analyzer driven approach. They propose an interpreter whose role is to inspect the code to be executed by the processor. If the inspected code is not considered as malicious, then it is stored in a buffer including the checked and authorized code. Otherwise, the code execution is then blocked. Reaction against intrusion in this approach is based on blocking the attacks based on injecting malicious code. In [Ragsdale et al. 2000], the authors propose an approach based on an agent architecture called Adaptive Agent based Intrusion Response System (AAIRS). Once an alert is detected, the AAIRS generates a response plan based on some factors (e.g.,response goal, attack type, attack implication). Figure 2.2 presents the response decision-making model of the AAIRS approach. The authors in [Porras and Neumann 1997] present an architecture called

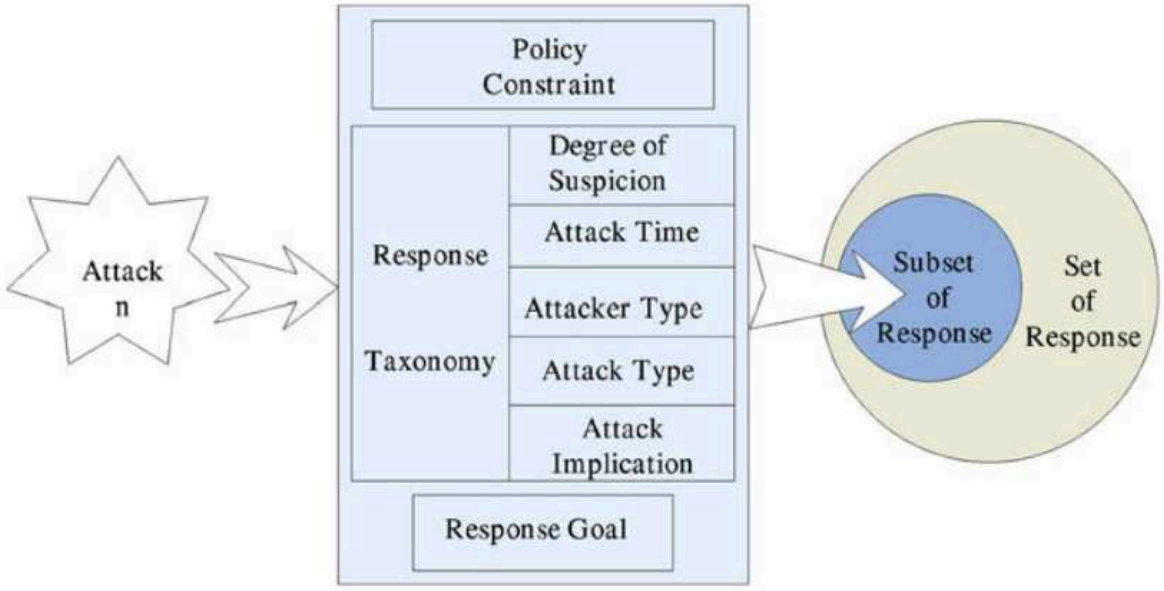


Figure 2.2: Response decision-making model of AAIRS

Event Monitoring Enabling Responses to Anomalous Live Disturbances (EMERALD). In EMERALD, countermeasures selection is based on attack evaluation to determine its severity and its impact on the monitored system. A resolver part is included in EMERALD architecture and whose role is to combine attack metrics (e.g., severity, confidence, frequency) to formulate the monitor's response policy.

The data used for the analyzes in EMERALD monitor are provided from event streams derived from a variety of sources. These sources can be audit data, network datagrams, SNMP traffic, application logs, and analysis results from other IDSs. Event

flows are inherently heterogeneous and must be formatted before the analyze phase. The statistical component of the monitor (Profiler Engine) performs an analysis based on the same approach as NIDES from an event stream [Anderson et al. 1994]. The component performing a signature-based analysis (Signature Engine) uses a variant of the P-Best expert system [Lindqvist and Porras 1999]. The analyzes results of both systems as well as those of other analyzers interfaced to the monitor are subjected to the resolver to be correlated. The resolver is the component of the monitor that is responsible for implementing the response policy. The countermeasures are defined in the $response - methods$ field of a reaction model. The major weakness in this work is that the authors do not provide description of the architecture capabilities and its application. Proposed dynamic models such as EMERALD or the model proposed in [White and Pooch 1996] do not focus on the damage caused by an intrusion. The response selection process considers only the responses evaluation and its impact on the monitored system.

Cost-sensitive model

Cost-sensitive model is a technique that attempts to balance intrusion damage and response cost [Lee et al. 2002, Mu and Li 2010]. These models provide a risk assessment component in order to measure intrusion damage. In [Balepin et al. 2003], the authors used the SHIM architecture [Laboratories 2003] to detect Linux hosting machine processes used to perform a specified attack. They propose to supervise all resources (e.g., files, connections) related to the malicious processes. Entities to supervise and countermeasures available for each entity are specified by the user. Countermeasures against malicious processes are automatically selected after evaluating the available countermeasures cost and choosing the most appropriate one. For this purpose, the authors propose a "gain matrix", which formulates the effects of selecting a specific response on the system. This formulation is based on two metrics; the probability of the system to be on a specific step from the attack scenario and the benefit from applying a system response on a specific system state. We note that this work considers only the detection of attacks executed on a single machine. The authors in [Soojin et al. 2006] propose an approach basically depending on the probabilistic correlation. This approach considers three main factors for system response against an intrusion detection: 1) operational cost, costs concerning the IDS time process of events; 2) damage cost, costs concerning the attack impact; 3) response cost, which refers to the cost of applying a system response against a detected attack. The disadvantage of this solution is that it induces a considerable traffic in the network. The authors in [Toth and Krügel 2002] propose a cost-sensitive intrusion response approach that evaluates the response effect on the

system using a dependency tree structure. This approach allows to select the response which has the minimal negative effect on the system. However, the work presented in [Toth and Krügel 2002] is not providing countermeasures that depend on the context in which the system is operating.

2.3.2 Adjustment Ability

The adjustment of an AIRS is its ability to revise and to adjust system responses selection based on previous responses analysis. The majority of AIRSs are non-adaptive (or static) system: response selection process remains the same during the attack detection. Few are the approaches that use adaptive systems. The authors in [Foo et al. 2005] propose an adapted approach called ADEPTS. This approach considers the response history by evaluating responses selected in the past as success of failure in order to improve the current responses selection. The AAIRS approach provides an adapted response against a detected intrusion by prioritizing responses that have been successfully applied over less successful responses. The adjustment ability is based on a learning process that evaluates historic selected responses, so that the AIRS selects only the succeeded countermeasures selected in the past. This learning is not automatically performed, it requires human intervention to evaluate previous decisions taken.

2.3.3 Response Execution

The main limitation of previous approaches is the large number of countermeasures to be selected especially when the corresponding attack scenario is constituted of a large number of attack steps. The challenge in AIRSs is to select the optimal set of candidate responses in real time. There are two types of AIRSs according to the type of response execution [Shameli-Sendi et al. 2012]: burst and retroactive.

Burst

Burst approaches present AIRSs that do not consider a mechanism to measure the risk index of the monitored host once the countermeasure has been applied. In this model, monitored system can apply a large set of responses, given a detected intrusion, while a subset of these responses may be enough to stop the attack. The major weakness of burst approaches is the high cost in system performance and quality of service. In other words, the only objective of such approaches is to mitigate the attack without considering the nominal system functional behaviors (e.g., performance, availability).

For instance, when considering two responses against intrusion where the first is to filter the suspect host and the second is to block all the connected hosts. The second countermeasure causes availability loss and can be avoided when applying filtering suspect host is enough to mitigate the attack.

Retroactive

Retroactive approaches provide a feedback mechanism that measures the countermeasure effect based on the responses history. This approach was first proposed in [Mu and Li 2010]. The authors presented a response measure decision-making model that optimizes the generated responses set by avoiding unnecessary responses and reducing the risk of false positive response. Existing works [Foo et al. 2005, Stakhanova et al. 2007b, Lee et al. 2002, Shameli-Sendi et al. 2013] propose approaches that rely on heuristics to reduce the size of candidate responses given a detected attack scenario. In order to limit the size of the system responses set, ADEPTS considers only the countermeasures that are applicable in the sites where the detected alert was generated. ADEPTS uses a graph of intrusion goals called $I - Graph$. It provides a semi-automated method called $PortableI - Graph$ (PIG) that determine the possible path of spread of the intrusion, appropriate services where to deploy the response, and appropriately choose the response. ADEPTS is not evaluating the candidate system responses according to the response effect on the overall system. The proposed approach considers only the system response effect on the specific service where it is deployed. The authors in [Stakhanova et al. 2007b] presented a retroactive AIRS based on a confidence level threshold; if the selected countermeasure mitigates the attack, its success factor is increased by one, and it is decreased by one on the contrary.

2.3.4 Prediction Ability

From the prediction ability point of view, AIRSs can be classified into two categories: Reactive and Proactive.

Reactive

In reactive approaches, system responses are applied only after an intrusion objective is achieved. Most existing AIRSs use this approach (e.g., [Papadaki and Furnell 2006], [Strasburg et al. 2009]), although this approach is not useful in critical systems where

high security is required. For instance, suppose that an attacker steals confidential and critical information. In this case, a reactive response is not useful since the confidential information has already been disclosed. In [Anuar et al. 2010], the authors present the drawbacks of using reactive approach, which are the following:

- System responses are applied after an intrusion detection, so the system remains in a vulnerable state until the reactive response is applied.
- It is difficult to return the system to the safety state.
- The attacker has a delay between intrusion detection and system response, this delay provides a window of opportunity for the attacker to be exploited.
- From the monitored system point of view, it is easier to maintain system in safe condition than returning it from an unhealthy state to the normal conditions.
- Systems are exposed to an important risk of damage, since responses are applied after an incident is detected.

Proactive

Proactive response system allows to prevent a malicious action before it happens. Existing AIRSs use a prediction phase in the detection component. The authors in [Yu and Frincke 2007] proposed the Hidden Colored Petri-Net (HCPN). This approach can describe the relationship between the intrusion different steps, alerts and actions. HCPN associates system states with a confidence level. This approach is called "hidden" because actions are not observable by IDS but can be inferred through alerts generation. In [Sendi et al. 2012], the authors proposed Alert Severity Modulating to anticipate the attacker's intentions. This approach is based on Hidden Markov Model (HMM) to extract interactions between attackers and targets. The main limitation of this approach is that it is not evaluating attack scenarios in term of risk and impact. The proposed approach does not consider as well the impact of applying a countermeasure given a detected attack scenario.

We previously showed approaches and architectures whose main role is to provide an automated system response against intrusion detection. A complete list of approaches and research studies on IRSs is given in Table 2.1. In light of the presented approaches, we argue that an ideal intrusion response system should satisfy the following features: Proactive, Adaptable, Cost-sensitive and Retroactive. We will now discuss the existing languages that model events detected on the system.

AIRS	Response Selection	Adjustment Ability	Response Execution	Prediction Ability
DC&A [Fisch 1996]	Dynamic	Non-adaptive	Burst	Reactive
CSM [White and Pooch 1996]	Dynamic	Non-adaptive	Burst	Reactive
EMERALD [Porras and Neumann 1997]	Dynamic	Non-adaptive	Burst	Reactive
BMSL-based response [Bowen et al. 2000]	Static	Non-adaptive	Burst	Reactive
SoSMART [Musman and Flesher 2000]	Static	Non-adaptive	Burst	Reactive
PH [Somayaji and Forrest 2000]	Static	Non-adaptive	Burst	Reactive
AAIRS [Ragsdale et al. 2000]	Dynamic	Adaptive	Burst	Reactive
SARA [Lewandowski et al. 2001]	Dynamic	Non-adaptive	Burst	Reactive
CITRA [Schnackengerg et al. 2001]	Dynamic	Non-adaptive	Burst	Reactive
TBAIR [Wang et al. 2001]	Dynamic	Non-adaptive	Burst	Reactive
Network IRS [Toth and Krügel 2002]	Cost-sensitive	Non-adaptive	Burst	Reactive
Lee's IRS [Lee et al. 2002]	Cost-sensitive	Non-adaptive	Burst	Reactive
Tanachaiwiwat's IRS [Tanachaiwiwat et al. 2002]	Cost-sensitive	Non-adaptive	Burst	Reactive
Specification-based IRS [Balepin et al. 2003]	Cost-sensitive	Non-adaptive	Burst	Reactive
ADEPTS [Foo et al. 2005]	Cost-sensitive	Adaptive	Burst	Proactive
FAIR [Papadaki and Furnell 2006]	Cost-sensitive	Non-adaptive	Burst	Reactive
Stakhanova's IRS [Stakhanova et al. 2007a]	Cost-sensitive	Adaptive	Burst	Proactive
DIPS [Haslum et al. 2007]	Cost-sensitive	Non-adaptive	Burst	Proactive
Jahnke [Jahnke et al. 2007]	Cost-sensitive	Non-adaptive	Burst	Reactive
Strasburg's IRS [Strasburg et al. 2009]	Cost-sensitive	Adaptive	Burst	Reactive
IRDM-HTN [Mu and Li 2010]	Cost-sensitive	Non-adaptive	Retroactive	Reactive
OrBAC [Kanoun et al. 2010]	Cost-sensitive	Adaptive	Burst	Proactive
Kheir's IRS [Kheir et al. 2010]	Cost-sensitive	Non-adaptive	Burst	Proactive
Shameli's IRS [Shameli-Sendi et al. 2013]	Cost-sensitive	Adaptive	Retroactive/Burst	Reactive

Table 2.1: Classification of existing AIRSs based on proposed taxonomy

2.4 Attack Description Languages

IDSs are responsible for detecting the occurrence of certain events in the monitored system. These events correspond to actions performed by the attacker, these actions being part of its attack strategy. An alert corresponding to the detection of a certain action contains information on involved machines (i.e., the action source and targeted hosts), and provides the name of the associated action. The amount and type of information transported by the alert are directly dependent on the detection technique used. These information do not provide actions semantics; trying to reason on such information is therefore very difficult. In order to reason on a set of alerts and draw conclusions from these observations, it is necessary to model the detectable actions to associate a semantic alerts. In the following, we present existing attack description languages, and we show how they are used.

2.4.1 CAML

This language is developed and used in [Cheung et al. 2003] as part of *Correlated Attack Modeling* (CAM) project [CAM 2003]. The purpose was to define a high level language so that it can be used by different correlation modules. CAML permits to model the steps of intrusion scenarios. An action is represented by a CAML module and its links with other modules are expressed by the specification of a *pre-condition* and a post-condition field. CAML language is accompanied by a predicates library representing vocabulary allowing to describe system properties according to an action model. A CAML module consists of three sections:

- *activity*: specifies events list to observe in order to instantiate an action model represented by a module. CAML events are based on the IDMEF format [Debar et al. 2007].
- *pre-condition*: defines the system state required for the execution of the action. This field defines, in addition, required conditions of other events already observed. For instance, as shown in Figure 2.3, the *pre-condition* field require that $r1$ must be observed before $r2$.
- *post-condition*: defines a list of predicates and events inferred once activity and *pre-condition* fields was satisfied.

Figure 2.3 presents a modeling of action that execute locally a code allowing the attacker access to confidential information.

```
module Remote-Exec-Access-Violation-2-Data-Theft
(
   activity:
      r1: Event(
            Source(Node(Address(a: address)))
            Target(Node(Address(b: address)))
            Classification(origin == "vendor-specific"
                           name == "CAM-Remote-Exec"))
      r2: Event(
            Source(Node(Address(address == b)))
            Target(Node(Address(c: address)))
            Classification(origin == "vendor-specific"
                           name == "CAM-Access-Violation"))
   pre:
      StartsBefore(r1, r2)
   post:
      Event(
         starttime == r1.starttime
         endtime == r2.endtime
         Source(Node(Address(address == a)))
         Target(Node(Address(address == c)))
         Classification(
            origin == "vendor-specific"
            name == "CAM-Data-Theft"))
)
```

Figure 2.3: CAML module: Remote execution and access violation to data theft

It is important to note that a CAML module does not correspond necessarily to an event detectable by an Intrusion Detection System (IDS). As shown in the example in Figure 2.3, we can see that the *post-condition* field is validated once both events $r1$ and $r2$ was observed in the good order. A correlation module using CAML as language of attacks description was implanted using the inference engine P-BEST [Lindqvist and Porras 1999] and by converting CAML models to P-BEST rules. Note that this conversion phase has to be manually performed. The implementation of this correlation engine revealed a combinatorial explosion problem in the P-BEST inference engine. This issue had not been highlighted in previous P-BEST engine applications because rules used in [Lindqvist and Porras 1999] have a limited effect on its antecedents.

Translating CAML models into P-BEST rules generates P-BEST rules with complex antecedents. The authors did not define a syntax or a grammar for the language. We observe that this language is largely based on the IDMEF structure, which means that its use is restricted in architectures using only this alert format.

2.4.2 ATiKi

In [Steffan and Schumacher 2002], Steffan and Schumacher present an attack scenario discovery tool. They provide both a representation of scenarios by Petri networks and a modeling for actions constituting the scenarios. More precisely, they used the "Attack Net" modeling presented in [McDermott 2000] to model attack scenarios. ATiKi modeling consists of two main elements:

> **Brute-force guess password**
> **Preconditions**: [→ read access to /etc/passwd], [→ account with weak password]
> **Postconditions**: [→ knowledge of password]
> **Contexts**: [→ UNIX-like system], [→ Linux system]: most modern Linux systems use shadow passwords, so /etc/passwd does not contain password hashes.
> **Description**: A password can be guessed if it is included in a reasonably small search space, such as all combinations of lowercase letters or lists of English words or names. See [→ account with weak password] for more cases of weak passwords. If the hash value of the password is known, an attacker can do the password guessing off-line by generating a hash value for each candidate in the search space and comparing it with the known hash.

Figure 2.4: Transition Brute-force guess password with *pre-* and *postconditions* and context. [→ ...] denotes hyperlinks in the ATiki system.

- Conditions: It describes informally the system properties (e.g., Unlimited failed logins are allowed) and the attacker's capabilities (e.g., valid password is known) by logical predicates. A true/false label should be assignable to each predicate.

- Transitions: It describes action *pre-condition* and *post-condition*. The semantics of transition is that all of the *pre-condition* have to occur in order to enable the transition to the *post-condition*

Conditions and transitions are associated with Wiki pages [Wik] to provide a better navigation in attack graphs. A wiki page is an HTML page that can be modified through a browser providing a simplified syntax, so that users can participate in web site construction. Attacks graphs are automatically generated starting from hyperlinks situated in models *pre-condition* and *post-condition*. Figure 2.4.2 presents an ATiKi model transition corresponding to a guess weak password attack. We can see that correlation links between system properties conditions and the attacker are explicitly specified by using hyperlinks toward conditions. **Contexts** field in Figure 2.4.2 defines the context of the guess weak password action. This field facilitates the navigation in system Wiki pages. Once the set of transitions and conditions was defined, a search is done to determine the set of attacks graphs.

Figure 2.5 presents an example of attack graph generated from a small set of transitions and conditions. This tool is intended to explore attacks graphs, but it is not intended to intrusion detection. However, although the authors do not mention it, it would be possible to use the generated graphs as a scenario models base for IDS. In addition, intrusion objectives cannot be modeled using Atiki modeling.

2.4.3 ADeLe

As part of MIRADOR project [Cuppens 2001], the authors in [Michel and Mé 2001] propose an Attack Description Language (ADeLe). The aim of this language is to specify a database of attacks to configure a set of IDS. ADeLe is a procedural language, we present in the following the structure of an ADeLe model. An ADeLe model consists of three parts:

– EXPLOIT: This part specifies required conditions to perform the attack, attack description (i.e., the attack code or its different steps) and the attack effects on the system. This part is composed of three parts: *pre-condition*, attack code, *post-condition*. There are no proposed languages expressing *pre-condition* and *post-condition* fields. The attack code field allows to specify the nature of the language used in the attack. It is thus possible to include a C++ function or to specify informally the attack and stating that this is a text. The language specification is used to provide a selection of the best interpreter or compiler when reading the file.

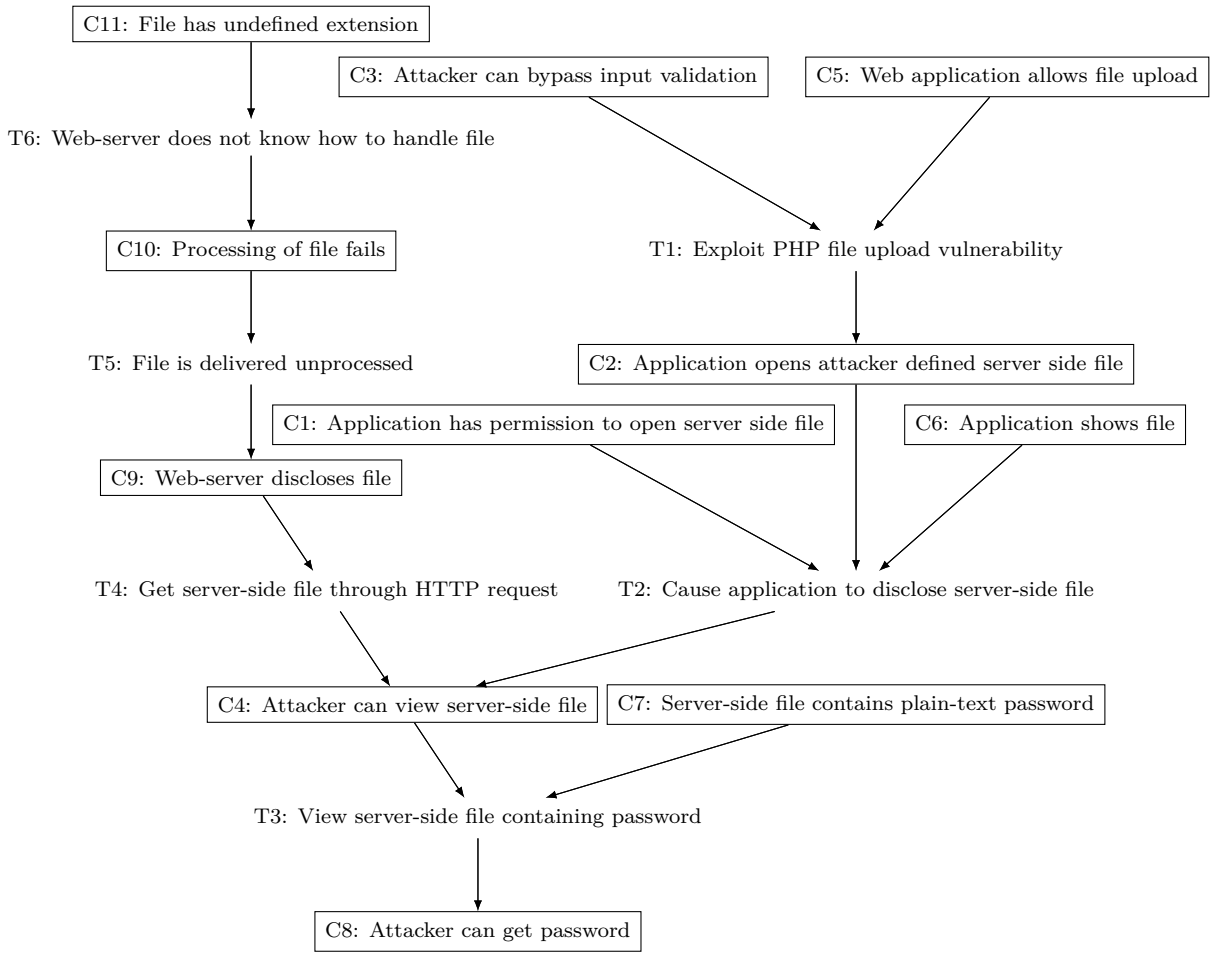


Figure 2.5: Example of Attack Graph discovered by Atiki system

– DETECTION: This part specifies how to detect the attack. A high level language is proposed to express the hash allowing to detect a low level event. This language permits also to express complex scenarios including know attacks. This part is composed of three sub parts:

  – DETECT: This part specifies the alerts and events expected once the attack is performed. This part specifies also the time and contextual constraints between alerts/events. These specifications allow to define how alerts/events must be sequenced to detect a full attack scenario.

  – CONFIRM: This part expresses elements that should be checked on the monitored system to confirm or deny the success of the attack. A set of functions is provided for this purpose. For instance, the function `Unreachable_Machine(<IP_address>)` returns the boolean value `true` if the specified machine is unavailable.

  - REPORT: This part specifies how generated alerts fields when detecting attack modeled via ADeLe language.

- RESPONSE: This part specifies countermeasures to be executed given a detected attack. Many functions are provided, for instance the function `Kill_Process(target_ip,user_name,process_id|"ALL")` permits to stop a process in a specific machine and the function `Script_Exec(script_name)` permits to execute a script.

We argue that the RESPONSE field do not respond to modern system requirements, since it proposes a simple countermeasure for each attack. However, for a given attack there may be several responses, more or less effective and whose relevance may vary depending on the context in which the system is operating or on the security administrator preferences and objectives. We judge important that the security system must be able to provide multiple system responses to the security administrator, leaving him the choice to select the most appropriate countermeasure. Since this language is used to model entire scenarios, system users are obliged to update the whole scenarios library each time a new elementary attack is introduced.

2.4.4 LAMBDA

The authors in [Cuppens and Ortalo 2000] present an attack description language. This language is based on logic and uses a declarative approach. It was also developed as part of MIRADOR project. The aim of this language is to define attacks independently from the technique used or the type of the targeted host. LAMBDA permits to model malicious actions (i.e., actions that violates security policy) as well as suspects actions (i.e., actions that do not violate the security policy but allow the execution of a malicious action). A LAMBDA model describes the attack from different point of views. We first give an informal description of a LAMBDA model, then we will discuss the different languages used in this modeling. A LAMBDA model describes an attack from two perspectives: the attacker perspective and the detection point of view. From the perspective of the attacker the model specifies three components:

- A set of conditions that must be satisfied in the targeted system, so that the attacker could perform his/her attack.

- The attack execution effects on the targeted system. These effects could be a modification on the system state (e.g., performing a Deny Of Service (DOS),

opening a connection) or could concern the user or the attacker itself (e.g., a knowledge acquisition or obtaining privileges reserved for the administrator)

– The scenario corresponding to the attack. An attack scenario can be composed of multiple actions.

From the detection point of view, a LAMBDA model specifies how to detect an attack. This description consists of three parts:

– The actions to be taken on the monitored system to detect the attack.

– How these detection actions must be combined to detect the attack.

– A set of verification measures to quantify the impact of the attack on the system.

LAMBDA is a modular language, which allows to describe an attack starting from other attack models. The modularity of a modeling language is an important aspect. Indeed, it facilitates the maintenance of an attack base and allows to describe attacks that can be used afterward in more complex scenarios. We present now the LAMBDA language structure as well as some LAMBDA model examples. The model adopted for the system representation is presented in Figure 2.6. The knowledge of the system

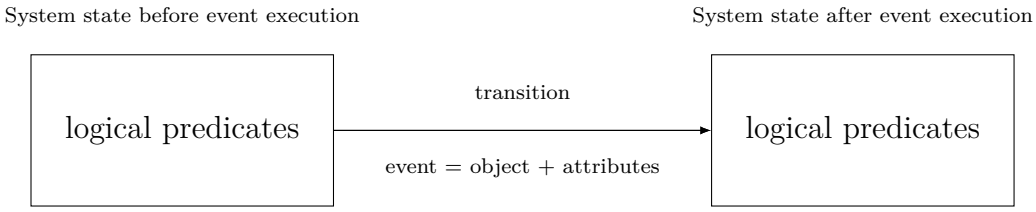


Figure 2.6: Monitored system model

state is represented in first order logic using logical predicates. Three languages are used in a LAMBDA model:

– State description, Language L1: This language is based on predicate logic. It allows to describe the system state via the *pre-condition* and *post-condition* fields of a given attack. For instance, the $use\_service(Address, netBios)$ predicate specifies that the host corresponding to $Address$ variable uses the $netBios$ service. This predicate can be used, for example, in an attack model where its execution requires that the targeted host uses the $netBios$ service. In order to combine multiple predicates in an expression, the logical operators $\wedge$, $\vee$ and $\neg$ are used in attack modeling. The effects of an attack does not always result in a change in the system

state. The execution of an attack can allow the attacker to get information about the targeted system without violating security policy. A meta predicate *knows* is defined to represent this acquisition of knowledge by the attacker. For instance, the predicate $knows(User, mountedpartition(Address, Partition))$ specifies that $User$ knows that the partition $Partition$ of the host corresponding to the address $Address$ is mounted. This predicate can be used for example in the *post-condition* of *shwomount* LAMBDA action.

– Transitions description, Language L2: In LAMBDA modeling, transitions are associated with events. Language L2 is based on two operators ($\wedge$ and $=$) and a set of attribute names. The set of considered attributes are *action*, *actor*, *date*. In order to compare temporal information, L2 includes also the operators $<$ and $\leq$. The transition associated with an event e can be formalized in L2 as follows: $action(e) = a \wedge actor(e) = u \wedge date(e) = [t1, t2]$.

– Events combination, Language L3: This language provides operators, allowing to combine events. Operators allowing to combine two events $e1$ and $e2$ are:

  – e_1 ; e_2: sequential composition of e_1 then e_2.
  – $e_1 \mid e_2$: parallel execution of e_1 and e_2.
  – $\bar{e_1}[t_1, t_2]$: absence of e_1 in the events flow between t_1 and t_2.
  – e_1 ? e_2: represents the non-deterministic choice between e_1 and e_2.
  – e_1 & e_2: synchronized execution of e_1 and e_2.

The full description of an attack is modeled using these three languages. An attack model described in LAMBDA is represented as follows:

attack: $attack\_name(arg_1, arg_2, ...)$

pre: $cond \in L_1$

post: $cond \in L_1$

scenario: $expr \in L_3$

where $cond \in L_2$

detection: $expr \in L_3$

where $cond \in L_2$

verification: $expr \in L_3$

where $cond \in L_2$

In a LAMBDA model, variables are represented by terms starting with a capital letter and constants by terms starting with a lowercase letter. We note that variables are used locally in a LAMBDA model. On the other hand, we cited the modularity of language: it is actually possible to reference an attack model as an action associated with an event, using the clause **where**. We present an example of attack modeled via LAMBDA. This attack consists of six steps and allows to exploit a bad configuration of security policy in order to access to a host partition. The attack's steps are modeled as follows:

1. **rpcinfo -p** *Target-IP*
   This command allows the attacker to know whether the *portmapper* service and the NFS daemon are started on the target host.

2. **showmount -e** *Target-IP*
   The attacker obtains the list of exportable hard drive partitions.

3. **showmount -a** *Target-IP*
   The attacker obtains the list of mount points.

4. **finger @** *Target-IP*
   The attacker obtains a host user ID and knows that the devil **finger** is started on the target host.

5. **adduser** –**uid** *Userid Username*
   This step is performed on the attacker's host. He/she adds a user account on his/her host by specifying the parameters acquired through the previous steps.

6. **mount -t** *Target-partition* **mnt**
   This step corresponds to the execution of action violating the security policy. In fact, the attacker obtains an illegal access to a mounted partition.

attack: $NFS\_abuse(IP - cible)$

pre: $remote\_access(A, H) \wedge ip\_address(H, IP - cible)$
$\wedge use\_service(H, portmapper) \wedge use\_service(H, mountd)$
$\wedge exported\_partition(H, P) \wedge mounted\_partition(H, P)$
$\wedge connected\_user(U, H) \wedge userid(U, H, Userid)$
$\wedge use\_service(H, fingerd) \wedge root\_user(A, H_A)$
$\wedge connected\_user(A, H_A) \wedge owner(Directory, U)$

post: $can\_access(A, Directory)$

scenario: $((E_1; (E_2 \& E_3)) \& E_4 \& E_5); E_6$

where $action(E_1) =$**rpcinfo -p** *IP-cible*
$\wedge action(E_2) =$**showmount -e** *IP-cible*
$\wedge action(E_3) =$**showmount -a** *IP-cible*
$\wedge action(E_4) =$**finger @** *IP-cible*
$\wedge action(E_5) =$**adduser** $-$**uid** *Userid Username*
$\wedge action(E_6) =$**mount -t** *Target-partition* **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: $((F_1; (F_2 \& F_3)) \& F_4); F_5$

where $action(F_1) = detect(E_1)$
$\wedge action(F_2) = detect(E_2)$
$\wedge action(F_3) = detect(E_3)$
$\wedge action(F_4) = detect(E_4)$
$\wedge action(F_5) = detect(E_6) \wedge date(F_5) = t$

verification: W_1

where $action(W_1) = foreign\_mount() \wedge date(W_1) = t'$
$\wedge t' \leq t$

Figure 2.7: Model of the attack exploiting the NFS service.

The Figure 2.7 represents a LAMBDA modeling of the full attack. The first four steps are part of the acquiring knowledge phase about the targeted machine. We did not represent elementary attack models in Figure 2.7 but the overall scenario. Note that the *detection* field makes no mention of the step executed locally on the attacker machine. In fact, this step is not detectable. The *detection* and $verification$ fields are intended to specify the operations required to detect and verify the success of the attack in the IDS. The separate specification of tasks to detect and verify the attack allows greater flexibility in specifying the attacks as these operations are largely dependent on the used platform. The $verification$ field specifies a function that checks if someone tried to mount a partition from a foreign host to the monitored network. Whereas the *detection* field specifies the attacker hash. The logical information that contain the *pre-condition* and *post-condition* models fields allow to consider automatic generation of complex scenarios using elemental action models.

We will see in the following chapters that LAMBDA language was considered to specify the elementary actions, to take advantage from its modularity. LAMBDA well models complex scenarios and facilitates the update process of attack base. Once these actions are specified, it is possible to find logical links between these models in order to correlate alerts for instantiating these models (see chapter 4). This makes the proposed AIRS in a phase advance with respect to attackers.

2.5 Conclusion

This chapter introduced Attack Description Languages. We discussed the benefits and drawbacks of most important languages and we judged LAMBDA language as the most appropriate attack description language to our approach. This chapter introduced as well the Automated Intrusion Response Systems and their main features. We explored existing approaches and their limitations. We showed that Proactive property should be satisfied so that the AIRS could anticipate the attackers intentions. We saw that Cost-sensitive and Adaptive properties of an AIRS should be satisfied as well in order to ensure the best level of performance and quality of service and to maintain system in safe conditions while respecting the security administrator requirements. Finally, we showed that it is important to satisfy the Retroactive property in order to satisfy the real-time constraints by generating an optimal set of countermeasures. For this purpose, we present in this thesis an approach based on the argumentative logic. This non-monotonic logic provides a smart reasoning that allows to reason on the cost of applying a countermeasure and to minimize the set of generated system responses given

a detected attack. Next chapter presents a background to the argumentative logic, we introduce basic notions that will be used in the rest of the thesis.

CHAPTER 3

Introduction to the Argumentation Logic

3.1 Introduction

When a system reasons and interacts with external elements, it may face different inconsistencies (e.g., unreliable observations, conflict between information exchanged with other systems). Thus, a smart system should have a reasoning tool that allows to manage those inconsistencies. We believe intuitively that argumentation is an appropriate candidate, since humans use it as a way to reason and to cope with conflicts. Argumentation presents an adapted model for the cognitive process of an AIRS to manage interactions between countermeasures that will be considered as arguments in the rest of the thesis. In this chapter, we present a definition of Argumentation Logic (AL), then, we explore different argumentation frameworks and we discuss the advantages and drawbacks of each existing framework. Finally, we explore existing works and approaches related to the security field using AL.

3.2 Motivation

Classical deduction is a consequence relation (denoted $\vdash$) that links premises with proofs. The example presented below shows how to reach a conclusion from a finite number of premises:

`John is 88 years old.`
`John is a man.`
`All men over 80 are old.`
$88 \geq 80$

`John is old.`

Classical deduction ensures the monotonic property: if Θ is a consequence of Γ then it is also a consequence of each set containing Γ:

$$\text{if } \Gamma \vdash \Theta \text{ and } \Gamma \subset \Delta \text{ then } \Delta \vdash \Theta$$

In other words, adding a new premises to a set Γ has no effects on conclusion deduced from Γ. Thus, when we accept proof premises, we are forced to accept its conclusion. Here we are talking about a closed universe and a monotonic logic. This logic is not adapted for intrusion detection context, since attacker actions modify the state of the system on which we reason.

Unlike classical logic, argumentation allows us to draw conclusions reserving the right to withdraw them in light of new information. We build and compare arguments that defend a conclusion as well as counter-arguments against the same conclusion. For instance, the following argument allows to reach the same conclusion of the example previously defined:

`John is old because he is octogenarian`

Contrary to monotonic logic, arguments can be defeated by other counter-arguments. Arguments are open to objections. Here we are talking about an open universe and a non-monotonic logic. An argument is accepted only when all its objections are defeated. We can distinguish different types of objections:

- Arguments that leave some implicit premises assuming that the audience accept it. For example, we presuppose that John is a man.
- Arguments that use vague, imprecise or open information. For example, no age threshold is mentioned.
- Arguments that admit objections in exceptional cases. For example, John is perhaps immortal.
- Arguments that can be introduced even when there are doubts about certain facts. For example, I am not sure that John is octogenarian.

This list is not exhaustive but it allows to distinguish between arguments and proofs. To summarize, when information are incomplete, uncertain or unclear and when the universe is open, it is better to use an argumentation system to model the reasoning rather than a proof system. This is the case of an autonomous and social agent. In the next section, we present existing argumentation frameworks.

3.3 Argumentation frameworks

In this section, we first present the AAF proposed by Dung [Dung 1995]. This work has highly inspired all argumentation systems proposed in last two decades. Thereafter, we present two extensions of the AAF: The framework proposed by Amgound and Cayrol [Amgoud and Cayrol 1998, Amgoud and Cayrol 2002] introduce the notion of preference and the framework proposed by Bench-Capon [Bench-Capon 2003] introduce the notion of value.

3.3.1 Abstract Argumentation Framework (AAF)

Phan Minh Dung proposes in [Dung 1995] to model the human way of argumentation during problems solving. In this framework, arguments are abstracted into entities whose role is solely determined by their relation to other arguments. That is why we talk about an abstract argumentation framework. To illustrate this principle, Dung presents an example of argumentation between two persons **I** and **A**, whose countries are at war, about who is responsible for blocking negotiation in their region.

Example

I: My government can not negotiate with your government because your government does not even recognize my government.
A: Your government does not recognize my government either.

The explicit content of **I**'s utterance is that the failure of **A**'s government to recognize **I**'s government blocks the negotiation. This establishes the responsibility of **A**'s government for blocking the negotiation by an implicit appeal to the following commonsense interpretation rule:

Responsibility attribution: If an actor performs an action which causes some state of affairs, then the actor is responsible for that state of affairs unless its action was justified.

A uses the same kind of reasoning to counter argue that **I**'s government is also responsible for blocking the negotiation as **I**'s government does not recognize **A**'s government either. At this point, neither arguer can claim "victory" without hurting his own position. Consider the following continuation of the above arguments:
I: But your government is a terrorist government.

This utterance justifies the failure of **I**'s government to recognize **A**'s government. Thus the responsibility attribution rule cannot be applied to make **I**'s government responsible for blocking the negotiation. So this represents an attack on **A**'s argument. If the exchange stops here, then **I** clearly has the "last word", which means that he has successfully argued that **A**'s government is responsible for blocking the negotiation.

Definition 1. *An Abstract Argumentation Framework is a pair* $\langle AR, attacks \rangle$ *Where:*

- AR *is a finite set of arguments*
- $attacks$ *a relationship over* $AR \times AR$.

$attacks(A_1, A_2)$ *means that* A_1 *represents an attack on* A_2.

Similarly, we say that a set S of arguments attacks A, if A is attacked by an argument from S. Argumentation systems according to AAF can be modeled using oriented graphs where nodes present arguments and arcs the *attack* relationship. For example, let $AS = \langle AR, attacks \rangle$ be an argumentation system defined as follows:

- $P, Q, R, S \in AR$
- $attacks(S, Q)$ $attacks(R, Q)$ $attacks(Q, P)$

AS argumentation system can be represented using oriented graph as shown in Figure 3.1. Arguments are represented by Letters and arcs represent the *attacks* relationship between them.

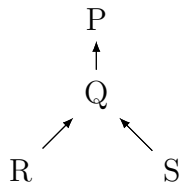


Figure 3.1: AS system representation in oriented graph

The objective of AAF is to determine if an argument is acceptable. The semantics of this framework assign a status to the arguments in a given set by the relationships that link on one another. An argument is said to be acceptable if it can be supported by other acceptable arguments despite the critics. The acceptability of an argument depends on the arguments (respectively counter arguments) that defend it (respectively attack it). It is in this perspective that the following definitions are introduced.

Definition 2. *Let $AS = \langle AR, attacks \rangle$ be an AAF, $S \subseteq AR$ and $P \in AR$.*
*S is said to be **conflict-free** iff $\forall P, Q \in S, \neg attacks(P, Q)$.*

Definition 3. *Let $AS = \langle AR, attacks \rangle$ be an AAF, $S \subseteq AR$ and $P \in AR$.*
*P is **acceptable** with respect to S iff $\forall Q \in AR, attacks(Q, P) \Longrightarrow attacks(S, Q)$*

Definition 4. *Let $AS = \langle AR, attacks \rangle$ be an AAF, $S \subseteq AR$.*
*A conflict-free set of arguments S is **admissible** iff each argument in S is acceptable with respect to S.*

In classical logic, a theory has a unique extension (argument set) that is a fixed point of the deduction operation. In non-monotonic logic, a theory can admit several extensions. Argumentative logic is a non-monotonic reasoning model whose semantics are defined by different extensions. For this purpose, three extension types are considered: ground extension, preferred extension and stable extension.

To define the ground extension, we introduce the characteristic function of an AAF.

Definition 5. *The characteristic function, denoted by FAF, of an argumentation system $AS = \langle AR, attacks \rangle$ is defined as follows:*
$F_{AS} : 2^{AR} \longrightarrow 2^{AR}$
$F_{AS}(S) = \{P \in AR \mid P \text{ is acceptable with respect to } S \}$

Definition 6. *Let $AS = \langle AR, attacks \rangle$ be an AAF. The grounded extension of AF, denoted by GE_{AS} is the least fixed point of F_{AS}.*

Figure 3.2: AS1 and AS2 systems representations in oriented graph

An argumentation system has always a unique ground extension that can be empty. For example, we consider two argumentation system ($AS1$) and ($AS2$) as shown in Figure 3.2. Each of these argumentation systems has a unique ground extension: $GE_{AS1} = \{R, S, P\}$ and $GE_{AS2} = \{\varnothing\}$ respectively for $AS1$ and $AS2$.

To enrich the semantics of an argumentation system, we define a preferred extension as the maximum admissible set.

Definition 7. *A **preferred extension** of an argumentation framework AF is a maximal (with respect to set inclusion) admissible set of AF.*

A preferred extension represents a consistent position that can defend itself against any attack and that can not be extended without introducing conflicts. The argumentation system $AS2$ presented in Figure 3.2 has a unique ground extension $GE_{AS2} = \{\varnothing\}$ and two preferred extensions: $pref_{AS2} = \{P, R\}$ and $pref_{AS2} = \{P, S\}$ An argumentation system does not necessarily have a unique preferred extension. We can then distinguish several classes of acceptability. A rationality which accepts only arguments that are in all preferred extensions is called skeptical. If it accepts the arguments that are at least in one preferred extension, it is qualified as credulous.

The ground extension is included in all preferred extensions. In addition, a preferred extension can be empty. Thus, the stable extension was introduced and defined as follows:

Definition 8. *Let $AS = \langle AR, attacks\rangle$ be an AAF, $S \subseteq AR$ and S is conflict-free. S is a stable extension iff $\forall P \in AR - S, attacks(S, P)$*

In other words, a conflict-free set S is called a stable extension iff S attacks each argument which does not belong to S.

An argumentation system does not necessarily have a stable extension. However, when it exists, it is not empty. For instance, we consider two argumentation system ($AS2$) and ($AS3$) as shown in Figure 3.3. Both preferred extensions: $pref_{AS2} = \{P, R\}$ and $pref_{AS2} = \{P, S\}$ are also stable extensions. $AS3$ do not has stable extension.

Figure 3.3: AS2 and AS3 systems representations in oriented graph

In addition, every stable extension is a preferred extension, but not vice versa. If this condition is verified, we talk about a coherent argumentation system.

Definition 9. *Let $AS = \langle AR, attacks\rangle$ be an AAF, AS is said to be **coherent** if each preferred extension of AS is stable.*

Unlike a preferred extension, a stable extension is never empty but it does not always exist. The notion of preferred extension is more interesting than ground extension since it allows to introduce different interpretations: skepticism and credulity.

Theorem 1. *Let $AS = \langle AR, attacks \rangle$ be an AAF, if the associated graph contains no cycles then this system has a unique non-empty preferred extension which is also a ground extension and stable extension.*

In abstract frameworks, attacking arguments have the same force. When two arguments attack each other, it is not possible to decide which one should be preferred. This fact has been addressed in [Amgoud and Cayrol 1998] where the authors extend *ARs* to define Preference-based Argumentation Framework (PAF) and Value-based Argumentation Framework (VAF). The authors argue that in many contexts the soundness of an argument is not the only consideration and that arguments have also a force.

3.3.2 Preference-based Argumentation Framework (PAF)

The authors in [Amgoud and Cayrol 1998, Amgoud and Cayrol 2002] extend the AAF. They claim that the acceptability of an argument depends on the arguments (respectively counter arguments) that defend it (respectively attack it) as well as on their force. Thus, they introduce a preference relationship on the argumentation framework (see Definition 1).

Definition 10. *A preference-based argumentation framework is a triplet $\langle AR, attacks, pref \rangle$ Where:*

- *AR and attacks have the same definitions as in Definition 1*
- *$pref$ is a preference relationship (i.e., $pref$ is a strict order relationship in AR.*

Arguments in PAF are linked by a preference relationship. In this way, an attack from argument P against argument Q may fail if Q is preferred over P. That is why the notion of $defeat$ has been introduced.

Definition 11. *Let $PAS = \langle AR, attacks, pref \rangle$ be a preference-based argumentation system and $P, Q \in AR$. P $defeats$ Q (denoted $defeats(P, Q)$) iff $attacks(P, Q) \wedge \neg pref(Q, P)$.*

In other words, P defeats Q if Q is not preferred over P.

Similarly, we say that a set S of arguments defeats A, if A is defeated by an argument from S. Therefore, Definition 2, Definition 3 and Definition 4 are modified as following:

Definition 12. *Let $PAS = \langle AR, attacks, pref \rangle$ be an AAF, $S \subseteq AR$ and $P, Q \in AR$.*

- *S is said to be **conflict-free** iff $\forall P, Q \in S, \neg defeats(P, Q)$.*
- *P is **acceptable** with respect to S iff $\forall Q \in AR, defeats(Q, P) \Longrightarrow defeats(S, Q)$*
- *A conflict-free set of arguments S is **admissible** iff each argument in S is acceptable (using PAF acceptability definition) with respect to S.*

As presented in Definition 7, a preferred extension of an argumentation framework PAF is a maximal (with respect to set inclusion) admissible set of PAF. We can represent a PAF using oriented graph models. Arguments are represented by nodes and arcs represent the $defeat$ relationship between them (i.e., $A \rightarrow B$ means that A defeats B). The preference relationship being a strict order relationship, it is asymmetric and transitive. Therefore, oriented graphs associated to PAF contain no cycle. According to the Theorem 1, PAF systems have a unique non-empty preferred extension which is also a ground extension and stable extension. For example, Figure 3.4 presents a PAF system ($AS4$). In the left side, $AS4$ $attacks$ relationships are represented using oriented graph. Preference relationships between $AS4$ arguments are listed in the middle of the figure. $AS4$ is represented in oriented graph model in the right side. $pref_{AS4}\{P, R\}$ is a non-empty and unique preferred extension and it is as well a ground extension and a stable extension.

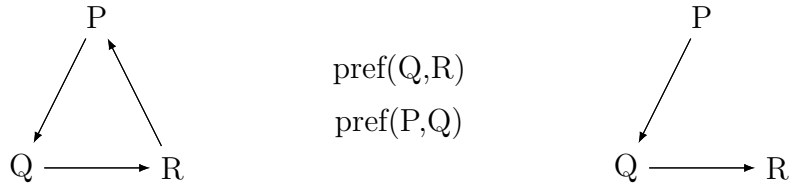


Figure 3.4: AS4 PAF system representation in oriented graph

The extension of the AAF to a PAF provides the use of the notion of preference. We can therefore model the choices made by an audience [Chaim and Lucie 1969]. However, this approach does not allow to consider several audiences simultaneously. This is the purpose of the work presented in the next section.

3.3.3 Value-based Argumentation Framework (VAF)

Trevor Bench-Capon also extends [Bench-Capon 2003] the AAF proposed by Dung in order to assign forces to arguments. The proposed argumentation framework allows

to consider several audiences simultaneously via multiple acceptability classes which enrich the semantics of the argumentation system. The proposed argumentation framework includes a set of values and a set of audiences. Multiple arguments can promote the same value.

Definition 13. *A Value-based Argumentation Framework (VAF) is a 5-tuple* $\langle AR, attacks, V, val, P\rangle$ *Where:*

- *AR and attacks have the same definition as in Definition 1*
- V *is a finite set of values*
- *val is a function which maps elements from AR to elements of* V
- P *is a set of audiences*

An audience corresponds to the audience concept proposed in [Chaim and Lucie 1969]. Audiences are differentiated from each other by preferences that they affect to values. There are potentially as many audiences as strict order relationships on the set of values. We can associate an argumentation system to each audience.

Definition 14. *An audience-specific value-based argumentation framework is a 5-tuple* $VAF_a = \langle AR, attacks, V, val, Valpref_a\rangle$ *Where:*

- *AR, attacks,* V *and val have the same definition as in Definition 13*
- a *is an audience,* $a \in P$
- $Valpref_a$ *is a transitive, irreflexive and asymmetric preference relationship,* $Valpref_a \subseteq V \times V$

Bench Capon redefines the notion of $defeat$ (Definition 11) to include the concerned audience.

Definition 15. *Let* $VAS_a = \langle AR, attacks, V, val, Valpref_a\rangle$ *an audience-specific value-based argumentation system and* $P, Q \in AR$.
$A\ defeats\ B$ *for audience a iff* $atacks(P, Q) \wedge \neg Valpref_a(Q, P)$

Similarly, we say that a set S of arguments defeats A for an audience a, if A is defeated by an argument from S. In other words, an argument Q is preferred over an argument P if its value is greater than P value. We notice that the success of an attack

is guaranteed when arguments have the same value. When all arguments in a VAF have the same value, the argumentation system can be seen as an AAF (Definition 1). An argumentation system can be seen as a PAF (Definition 11), when each argument from a VAF has a different value. Definition 12 was also adapted as follows:

Definition 16. *Let* $VAS_a = \langle AR, attacks, V, val, Valpref_a \rangle$ *an audience-specific value-based argumentation system,* $P, Q \in AR$ *and* $S \subseteq AR$.

- S *is said to be* ***conflict-free*** *for audience* a *iff* $\forall P, Q \in S, \neg defeats_a(P, Q)$.
- P *is* ***acceptable*** *for audience* a *with respect to* S *iff* $\forall Q \in AR, defeats_a(Q, P) \Longrightarrow defeats_a(S, Q)$
- *A conflict-free set of arguments* S *is* ***admissible*** *for audience* a *iff each argument in* S *is acceptable (using* VAF *acceptability definition) with respect to* S.

As presented in Definition 7, a preferred extension for audience a is the maximal admissible set for audience a denoted $preferred_a$. We can represent an audience-specific value-based argumentation system using oriented graph. Arguments are represented by nodes and arcs represent the $defeat$ relationship (for audience a) between them (i.e., $A \rightarrow B$ means that $A\ defeats_a\ B$). The preference relationship being a strict order relationship, it is asymmetric and transitive. Therefore, oriented graphs associated to VAF_a contain no cycle. According to the Theorem 1, VAF_a systems have a unique non-empty preferred extension which is also a ground extension and stable extension.

For example, Figure 3.5 presents a VAF_a system ($AS5$). If the shadowed nodes have a preferred value for audience a, then the preferred extension is $pref_{AS5}\{S, Q\}$ which is by the way unique and non-empty and it is also a ground and stable extension. Otherwise, the preferred extension is $pref_{AS5}\{P, R\}$ which is also a ground and stable extension, unique and non-empty.

Bench Capon enriches the proposed argumentation framework by introducing the notion of objective and subjective acceptance of an argument as well as the notion of indefensible argument as follows: as follows:

Definition 17. *Given a* VAF, $\langle AR, attacks, V, val, P \rangle$, *an argument* $A \in AR$ *is* ***objectively acceptable*** *iff* $\forall p \in P,\ A \in preferred_p$.

Definition 18. *Given a* VAF, $\langle AR, attacks, V, val, P \rangle$, *an argument* $A \in AR$ *is* ***subjectively acceptable*** *iff* $\exists p \in P,\ A \in preferred_p$.

Definition 19. *Given a* VAF, $\langle AR, attacks, V, val, P \rangle$, *an argument* $A \in AR$ *is* ***indefensible*** *iff* $\forall p \in P,\ A \notin preferred_p$.

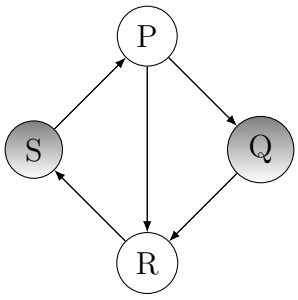


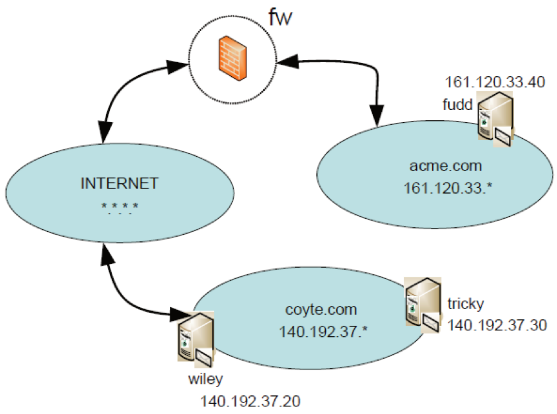
Figure 3.5: AS5 audience-specific value-based argumentation system representation in oriented graph

3.4 Related Work

The application of argumentation in Cybersecurity related issues is not deeply investigated and it presents a new research field. A first preliminary work addressing Cybersecurity issues using argumentation was proposed in [Rowe et al. 2012]. In this paper, the authors suggest the use of argumentation to provide automated support for security decisions and in reconfiguration problem, to diagnose the root cause of security attack, and to set security policies.

3.4.1 Argumentation logic for firewall policy specification

Argumentation was applied to assist firewall configuration management in [Bandara et al. 2006, Bandara et al. 2009, Applebaum et al. 2012].
Firewall is a tool that specifies which traffic types should be permitted or denied based on IP addresses. This tool is configured through an ordered set of rules. The author in [Wool 2004] presents a study concerning firewall configurations. This study shows that the average of rules in firewall configuration is 144 rules and can reach thousand rules. This large number of rules presents a constraint when specifying and maintaining firewall configurations. In addition, configuring firewall is difficult a task since many changes can be required during firewall configurations. Theses changes are made by different security administrators. Thus, such changes can induce anomalies since rules order is crucial when specifying firewall configurations. For instance, consider the example presented in Figure 3.6. The *acme.com*'s security administrator specifies the following high-level requirement: "*allow FTP connections from all hosts in the coyote.com network except for the host called tricky.coyote.com*". This requirement is implemented in this example by rule 5 and 6. However, if the security administrator inverts these rules order, FTP connections from all hosts in *coyote.com* will be



(a) Network topology

Order	Ptrcl	Src IP	Src Port	Dst IP	Dst Port	Action
1	tcp	140.192.37.20	any	*.*.*.*	80	block
2	tcp	140.192.37.*	any	*.*.*.*	80	allow
3	tcp	*.*.*.*	any	161.120.33.40	80	allow
4	tcp	140.192.37.*	any	161.120.33.40	80	block
5	tcp	140.192.37.30	any	*.*.*.*	21	block
6	tcp	140.192.37.*	any	*.*.*.*	21	allow
7	tcp	140.192.37.*	any	161.120.33.40	21	allow
8	tcp	*.*.*.*	any	*.*.*.*	any	block
9	udp	140.192.37.*	any	161.120.33.40	53	allow
10	udp	*.*.*.*	any	161.120.33.40	53	allow
11	udp	*.*.*.*	any	*.*.*.*	any	block

(b) Firewall rules

Figure 3.6: Example firewall configuration [Al-Shaer and Hamed 2003]

allowed including the host *tricky.coyote.com*. This constitutes an anomaly in firewall configuration called "shadowing" anomaly. Other anomaly types can be detected when analyzing a firewall configuration such as redundancy, generalization and correlation anomalies. In [Bandara et al. 2006], the authors treat the problem of resolving the possible anomalies in firewall policies using argumentation for Logic Programming with Priorities (LPP) [Kakas et al. 1994, Dimopoulos and Kakas 1995]. The use of this framework allows preferences to be encoded, thus allowing complex reasoning over the relative priorities between rules. For instance, the security administrator specifies in this example the following high-level requirement: "*block any traffic except HTTP requests to fudd.acme.com*".
This type of requirement includes a permission and a denial requirements which are the following: "*block any traffic*" and "*allow HTTP requests to fudd.acme.com*". These two requirements are specified by the security administrator, respectively by rule 8 and 3 and are implemented as follows:

```
req(block_any, block, Pkt):
action(block, Pkt) ←
packetFrom(any, Pkt), packetTo(any, Pkt), traffic(any, Pkt).

req(allow_http_fudd, allow, Pkt):
action(allow, Pkt) ←
packetFrom(any, Pkt), packetTo(fudd, Pkt), traffic(http, Pkt).
```

To avoid generating a shadowing anomaly when configuring the firewall, the

authors propose the `prefer` predicate:

```
order(allow_http_fudd, block_any):
prefer(req(allow_http_fudd, allow, Pkt), req(block_any, block, Pkt)).
```

This predicate is used to ensure precedence relationship between these requirements to avoid the shadowing anomaly. The notion of attack between arguments is introduced in this approach through the predicate `complement`:

$\texttt{complement(L}_1\texttt{, L}_2\texttt{)} \leftarrow \texttt{B}$

This predicate ensures that two conflictual rules ($\texttt{L}_1$, $\texttt{L}_2$) can not be part of the same admissible extension under some conditions `B`. The `complement` predicate includes the incompatibility between two opposite preference rules (i.e., `prefer(rule1,rule2)` is always incompatible with `prefer(rule2,rule1)`). The complementarity between rules allowing the traffic and others denying the same traffic is defined through `complement` predicate as follows:

```
complement(action(allow,_), action(block,_)).
```

In [Bandara et al. 2009], the authors extend their previous work. The proposed approach generates automatically firewall policies from higher-level requirements. The extended approach supports generation of the low-level rules from high-level policies.

The framework presented by Applebaum in [Applebaum et al. 2012] differentiates itself from these last two approaches through introducing the notion of rationales. The authors take advantage from the Value-based Argumentation Framework by assigning a rationale (value) to each argument (rule) in the policy. Applebaum proposes to resolve conflictual situations in firewall policies by defining a potential ordering of the rationales behind each argument (rule). The firewall can then resolve anomalies and conflictual rules through this order of priorities. We consider the firewall policy example in Table 3.1 proposed by the authors.
The proposed firewall policy contains anomalies. The authors present in Table 3.2 all anomalies in the firewall policy example.

Argumentation is incorporated in this approach in order to avoid these anomalies. The *attacks* relationship between arguments is used to define anomalies over rules. For instance, rules 5 shadows rule 11, therefore rule 5 is said to be in *attack* relationship

order	action	protocol	source IP	port
1	allow	*	*	20
2	allow	*	*	80
3	block	*	123.456.78.90	*
4	allow	*	*	21
5	block	*	*	53
6	allow	TCP	123.456.78.11	23
7	block	*	123.456.78.*	*
8	allow	UDP	123.456.78.11	5027
9	allow	UDP	*	*
10	block	*	*	6969
11	allow	*	75.75.75.75	53
12	block	*	*	*

Table 3.1: An example firewall policy [Applebaum et al. 2012]

shadowing	Correlation	Generalization	Redundancy
(5,11)	(1,3) (1,7)	(1,12) (2,12)	
(7,8)	(2,3) (2,7)	(4,12) (6,7)	
	(3,4) (3,9)	(6,12) (8,12)	
	(4,7) (5,9)	(9,12) (11,12)	
	(7,9) (9,10)		

Table 3.2: All anomalies in the firewall policy example. Each pair (x,y) is an anomaly

with rule 11. Table 3.3 summarizes all attack relationships between the firewall policy rules.

To resolve conflicts in firewall policy, the authors use the VAF and introduce the notion of "reasons". They propose to assign high-level reasons (values) to rules (arguments) to determine which rules should be preferred given an anomaly caused by two conflictual rules. They propose the following high-level reasons: accessibility, prophylaxis, legitimate/malicious sender, enable/disable a protocol, and enable/disable a program. According to the proposed firewall policy example, the authors assign reasons to the firewall rules as showed in Table 3.4

The security administrator specifies an ordering for these reasons according to his/her preferences and requirements. For instance, the authors consider the reasons ordering presented in Table 3.5.

Once the reasons ordering is established, the proposed approach provides recommen-

rule	attacks
1	3, 7, 12
2	3, 7, 12
3	1, 2, 4, 9
4	3, 7, 12
5	9, 11
6	7, 12
7	1, 2, 4, 6, 8, 9
8	7, 12
9	3, 5, 7, 10, 12
10	9
11	5, 12
12	1, 2, 4, 6, 8, 9, 11

Table 3.3: All attack relationships within the example firewall policy

rule	values	attacks
1	enable protocol	3, 7, 12
2	enable protocol	3, 7, 12
3	malicious sender	1, 2, 4, 9
4	enable protocol	3, 7, 12
5	disable protocol	9, 11
6	legitimate sender, enable program	7, 12
7	malicious sender	1, 2, 4, 6, 8, 9
8	legitimate sender, enable program	7, 12
9	accessibility	3, 5, 7, 10, 12
10	disable protocol	9
11	legitimate sender, enable protocol	5, 12
12	prophylaxis	1, 2, 4, 6, 8, 9, 11

Table 3.4: Overlap of rules in example policy, the center column gives the reason behind the rule

dations to the security administrator in order to assist him/her to deal with existing anomalies. Based on the ordering in Table 3.5, a recommendation is generated for each anomaly. Table 3.6 shows the different recommendations to avoid the firewall policy anomalies and the justification of each recommendation.
The main limitation of this approach is that the security administrator defines a static

order	value name
1	allow programs
2	block malicious senders
3	allow legitimate senders
4	block protocols
5	block programs
6	allow protocols
7	prophylaxis
8	accessibility

Table 3.5: Potential ordering of the ground-based values with lower order meaning higher priority

rules in conflict	anomaly name	recommendation	justification
(5,11)	shadowing	place 11 before 5	allow sender > block protocol
(7,8)	shadowing	place 8 before 7	allow program > block sender
(1,3),(1,7)	correlation	place 3,7 before 1	block sender > allow protocol
(2,3),(2,7)	correlation	place 3,7 before 2	block sender > allow protocol
(3,4)	correlation	ignore	block sender > allow protocol
(3,9)	correlation	ignore	block sender > accessibility
(4,7)	correlation	place 7 before 4	block sender > allow protocol
(5,9)	correlation	ignore	block protocol > accessibility
(7,9)	correlation	ignore	block sender > accessibility
(9,10)	correlation	place 10 before 9	block protocol > accessibility
(1,12),(2,12),(4,12)	generalization	ignore	allow protocol > prophylaxis
(6,7)	generalization	ignore	allow program > block sender
(6,12),(8,12)	generalization	ignore	allow program > prophylaxis
(9,12)	generalization	remove 9	prophylaxis > accessibility
(11,12)	generalization	ignore	allow sender > prophylaxis

Table 3.6: Anomalies and their corresponding recommendation based on the ordering in Table 3.5

order of priorities for the reasons behind the firewall rules. However, administrators can decide in specific cases to change the order of rationales priority. For instance, the security administrator can decide to give "prophylaxis" reason a higher priority than "allow protocols" in critical contexts. In such cases, firewall administrators are forced to update the firewall configuration for each required change in priority order.

3.4.2 Argumentation logic for access control

There are very few proposed approaches [Boella et al. 2005, Dijkstra et al. 2005, Doutre et al. 2007]dealing with access control management based on argumentation. In [Boella et al. 2005], the authors proposed an argument based approach for access control. Classical access control tools are based on the identity of the client through checking if the client profile complies with the security policy. However, the required credentials to access to a specific resource are not always known to the client. Therefore, interactive access control was introduced in [Koshutanski and Massacci 2004a, Koshutanski and Massacci 2004b], in which the process of accessing a resource can be seen as an interaction between the client and the resource owner. This process constitutes a negotiation about the credentials required for accessing the required resource until reaching an agreement. Authors defines the *objective-policy description* as a tuple $\langle O, P, K\rangle$ where O is a set of conditional objectives, P a set of policy rules and K a set of integrity constraints. The access control example presented in [Boella et al. 2005] illustrates these three types of rules in an $objective - policy\ description$. The authors consider a digital library case of study to explore how argumentation can be used for managing control access. Requests submitted by clients concern getting access to an article or an mp3 file. Authorization to get an electronic copy of an article is only given once the subscription to the library has been payed with the client e-money. This authorization can as well been given once the client shows a university employee pass. Once the client got the paper, the system collects a survey as a questionnaire to be filled and send by the client. Once the client got access to the requested mp3 file, the system improves its bandwidth. The authors distinguish among credentials (C) and state variables (S) in the logical language. The credentials and variables set corresponding to the example are the following:

$C = \{es, sp, el, sr, em, ra, f, rm, ds, sm\}$ and $S = \{a, al, cr, m, cs, ib\}$, where:

a : access article from digital library
al : authorized to access library
es : subscribe to library with e-money
sp : send university employee pass
cr : comply with digital rights
el : pay library with e-money
sr : signed order by employer
m : access mp3 file
em : pay mp3 file with e-money
ra : request article
cs : collect survey
f : fill in questionnaire
rm : request mp3 file
ib : improve bandwidth
ds : decrease download speed
sm : share downloaded mp3 file

Consider the following *objective-policy description*:

$O = \{ra \rightarrow a, a \rightarrow cs, rm \rightarrow m, m \rightarrow ib\}$
$P = \{al \wedge cr \rightarrow a, sp \rightarrow al, es \rightarrow al, sr \rightarrow cr, el \rightarrow cr, f \rightarrow cs, em \rightarrow m, sm \rightarrow ib, ds \rightarrow ib\}$
$K = \{ra, rm, em \rightarrow \neg es, em \rightarrow \neg el\}$

The authors introduce the notion of goal set and candidate goal argument for a goal set. A goal set is derived from the system objectives O and it represents the security administrator goals and requirements. These notions are defined formally as follows:

Definition 20. *Let $\langle O, P, K\rangle$ be an objective-policy description.*

- *A goal set G is a set of literals.*
- *A candidate goal argument for goal set G, written $c(G)$, is a finite linear tree consisting of pairs of sets of literals with its unique leave (B, G) or any B, such that for each node (B, H) there exists a conditional objective $l_1 \wedge .... \wedge l_n \rightarrow l \in O$ such that:*
  - *(a) $B = \{l_1, ..., l_n\} \subseteq Cl(K, U)$, where U is the union of all literals occurring in the ancestors of (B, H).*
  - *(b) if (B, H) is the root, then $H = \{l\}$, otherwise $H = \{l\} \cup H'$ when the unique parent of (B, H) is (B', H') for some B'.*
- *A goal argument for goal set G, written $g(G)$, is a candidate goal argument $c(G)$ such that there is no set of goal sets $\{G_1, ..., G_n\}$ with each $G_i \neq G$ and $G = G_1 \cup ... \cup G_n$. A maximal goal set is a goal set which has a goal argument and which is maximal with respect to set inclusion.*
- *We say that two goal arguments conflict if they contain nodes $\langle B_1, H_1\rangle$ and $\langle B_2, H_2\rangle$ such that $Cl(K \cup B1 \cup H1 \cup B2 \cup H2, \varnothing) \vdash \perp$, where $\perp$ stands for any contradiction.*

According to the example previously presented, the goals sets are $\{a\}$, $\{a, cs\}$, $\{m\}$, $\{m, ib\}$. We note that the set $\{a, cs, m, ib\}$ is not a goal set, since it can be splitted in $\{a, cs\}$ and $\{m, ib\}$. The plan arguments generated for the example are represented in Figure 3.7.

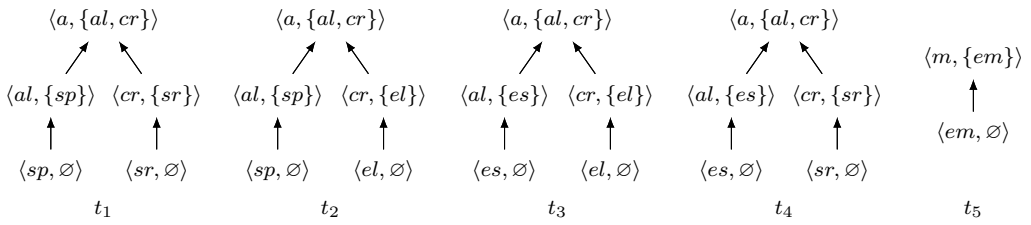


Figure 3.7: The plan arguments for the proposed example

The authors used Dung's abstract framework to formalize the proposed argumentation theory. As seen in Section 3.3.1, an Abstract Argumentation Framework is a pair $\langle AR, attacks \rangle$. In this approach, the authors consider the AR set as goal-plan arguments, and the conflict relationship between goal arguments as the *attacks* relationship. We explored in this section the existing argumentation frameworks and we showed the semantics provided by each framework. We use the argumentation in the context of intrusion detection to take profit of the smart reasoning that provides this non-monotonic logic.

3.4.3 Argumentation logic for network security analysis

More related to the topic of this thesis, the authors in [Martinelli and Santini 2014] propose a preliminary argumentation approach for assisting security administrators when selecting countermeasures against a specific detected attack. This approach takes advantage of the Abstract Argumentation Framework. The authors consider a decision system as a pair (D, F) where D is the set of security decisions and F is an AAF. The authors distinguish between two kinds of arguments:

- epistemic arguments (A_e): arguments based only on beliefs
- practical arguments (A_p): arguments built from beliefs and preferences or goals

Given a specific decision $d \in D$, the authors define two subsets of practical arguments:

- arguments in favor of d and denoted $\mathcal{F}_f(d)$
- arguments against d and denoted $\mathcal{F}_c(d)$

The authors propose a way to prefer ($\succcurlyeq$) a decision instead of another using *unipolar* principle defined as follows:

Definition 21. *Let $DS = (D, F)$ be a decision system, where F is an AAF, and $Acc_{stb}(F)$ collects the sceptically accepted arguments of a framework F under the stable semantics. Let $d_1, d_2 \in D$, then:*
$d_1 \succcurlyeq d_2 \Leftrightarrow \mid \mathcal{F}_f(d_1) \cap Acc_{stb}(F) \mid \geqslant \mid \mathcal{F}_f(d_2) \cap Acc_{stb}(F) \mid$

In other words, this principle specifies the arguments in favor of and against a decision and select the most supported decision.
The authors present the following example for adopting countermeasures:
Consider an AAF, $F_{worm} = \langle \{a, b, c, d, e, f, g\}, \{b \mapsto a, c \mapsto b, d \mapsto b, e \mapsto d, f \mapsto a, g \mapsto f\} \rangle$[1] where:

a. A worm is attacking our web server.

b. Disabling Web traffic mitigates worms.

c. Web traffic can be blocked only if loss $> 70\%$.

d. Traffic should not be blocked if the alarm is faulty.

e. Evidence shows that the alarm is reliable.

f. The antivirus has been recently updated.

g. Virus definitions are no longer maintained.

This AAF is represented in Figure 3.8 Two decisions are considered in this example, $D = \{disable\text{-}port80, \neg disable\text{-}port80\}$. The first decision represents the action of disabling traffic through port 80 whereas the second one represent the action of not disabling it. Arguments b, c and d are directly related to decisions in D, therefore A_p is $\{b, c, d\}$. We distinguish $\mathcal{F}_f(disable\text{-}port80) = \{b\}$ and $\mathcal{F}_f(\neg disable\text{-}port80) = \{c, d\}$. The proposed example has only one stable extension which is $stb(F_{worm}) = \{a, c, e, g\}$ thus $Acc(F_{worm}) = \{a, c, e, g\}$ (represented in gray in Figure 3.8. According to Definition 21, $\mid \mathcal{F}_f(disable\text{-}port80) \cap Acc_{stb}(F_{worm}) \mid = 0$ and $\mid \mathcal{F}_f(\neg disable\text{-}port80) \cap Acc_{stb}(F_{worm}) \mid = 1$. Consequently, $\neg disable\text{-}port80 \succcurlyeq disable\text{-}port80$, and the recommended countermeasure is to not disable traffic on port 80. The main limitation of the proposed approach is that it does not consider the topology of the network. The approach proposed by the authors in [Martinelli et al. 2015] takes into account this limitation and it considers the dependencies between the network different components when reasoning about the best countermeasure to recommend.

[1] The authors denote by ($\mapsto$) the attack relationship between arguments

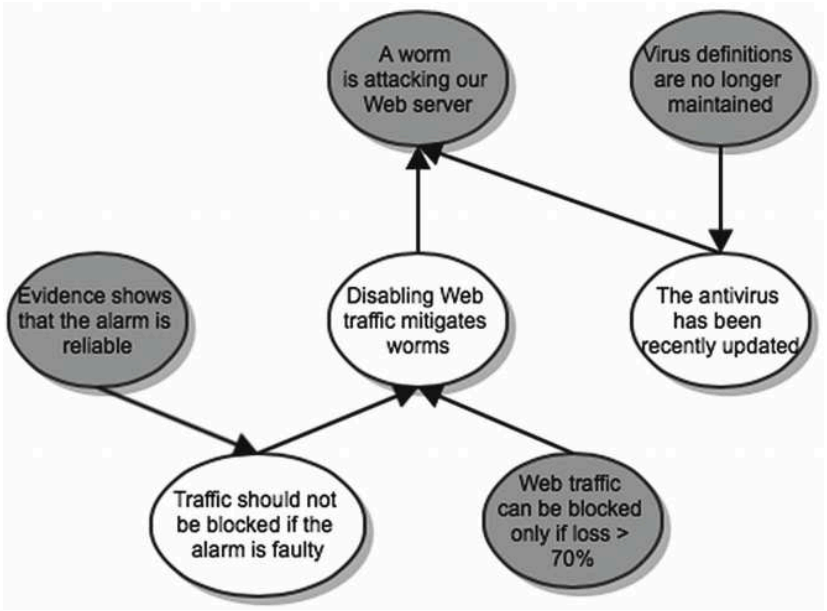


Figure 3.8: *AAF* representation: shadowed nodes represent arguments of the unique stable extension

3.5 Conclusion

As mentioned in the previous chapter, an AIRS must have a reasoning model that manages inconsistencies between countermeasures from its reaction policy and to reason on information transmitted by IDSs. In this chapter we have shown that argumentation presents an adapted model to the process of an AIRS to manage the interactions between internal arguments that presents the reaction policy and external arguments that presents attack detection. This chapter introduced the Argumentation Logic and its different frameworks. We showed the main advantages that provide each existing framework as well as the issues that can meet security approaches based on argumentation logic. The next chapter presents an approach for monitoring systems that allows the security administrator to choose the best intrusion response among all appropriate intrusion response possibilities. This approach is driven by argumentative logic and it considers the current context on which the system is operating. The purpose of this approach is to give a dynamic aspect to the intrusion response process of attacked systems that may operate while different contexts are active.

CHAPTER 4

Context-aware Response against Intrusion Detection

4.1 Introduction

Automatic response in an intrusion detection process is a difficult problem. Indeed activating an inappropriate countermeasure for a given attack can have deleterious effects on the targeted system. In some cases the countermeasure can be more harmful than the attack it is targeted against. Moreover, given an attack against a specific system, the best countermeasure to apply depends on the context in which the system is operating. For example, in the case of an automotive system, the fact that the vehicle is operating downtown or on a freeway changes the impact an attack may have on the system. In this chapter, we present an approach for monitoring systems that allows the security administrators to choose an intrusion response that satisfies security administrator requirements and which considers the current context on which the system is operating. This approach is driven by Argumentation Logic. The purpose of this approach is to give a dynamic aspect to the intrusion response process of attacked systems that may operate while different contexts are active. The security system will be able to take the suitable decision, against a specific attack scenario, and which ensure the system safety while satisfying the prioritized system requirements.

This chapter explains how we model the intrusion process and presents our approach to construct the set of arguments set corresponding to an attack scenario. It defines a new argumentation framework which is an extension of the Value-based Argumentation Framework (VAF). We present deployment scenarios highlighting how our approach is applied in the use case of automotive systems.

4.2 Modeling the intrusion processes

The attacker is modeled as an agent who can choose from a set of actions a subset to execute in order to reach one or several intrusion objectives. The set of actions executed by an attacker can be organized in a scenario of correlated actions. Informally, by correlated actions we mean that in an intrusion scenario, some action effects makes other actions possible. From the attacked system point of view, given a set of observed actions organized into an attack scenario, reacting against an ongoing attack consists in selecting a set of countermeasures which modifies the system state to stop the attack progression or mitigate its effects.

4.2.1 Modeling the attacker

We use the LAMBDA formalism [Cuppens and Ortalo 2000, Cuppens et al. 2006] to model the intrusion process. In our context, we consider that several probes are distributed in the monitored system to generate events which corresponds to actions executed by the agents acting on the system. Those agents can be legitimate users as well as malicious agents. The probes can be intrusion detection users [Axelsson 2000] or programs monitoring system logs for interesting events.

Actions

We generalize the notion of LAMBDA attack to the notion of LAMBDA action as some of the actions an attacker execute do not have malicious effects on the attacked system, i.e., their effects do not violate the security policy of the system. In our approach, a LAMBDA action description is composed of the following elements:

Definition 22. ***LAMBDA action***
name: *the action name*
pre-condition: *defines the state of the system required for the execution of the action.*
post-condition: *defines the state of the system after the successful execution of the action.*
detection: *is a description of the expected alert corresponding to the detection of the attack.*

The detection attribute may be empty as some actions cannot be detected by probes. For example, the modification of a file in the file system may not be logged. Although it is technically possible, for example on Linux systems using the audit infrastructure,

logging such events for every component of a system may result in generating too many events and flood the process responsible for reasoning on possible intrusions on the system.

The pre-condition and post-condition are written using conjunctions of literals, a literal being defined as follows:

- a constant is a string starting with a lower case character or a number
- a variable is a string starting with an upper case letter
- a term is either a constant, a variable or a functional symbol over a list of terms
- a literal is a predicate symbol over a list of terms expressing a boolean property

The detection attribute is written as a list of affectations of values to a subset of the free variables in the pre-condition. An action model is instantiated, i.e., values are assigned to the list of free variables in the pre-condition, when a new alert is generated. Predefined function symbols corresponding to alert attributes are used to specify the detection field. The alert contains the name of the LAMBDA model to instantiate if such model exists.

Literals with specific semantics are defined. The literal *not*, of arity one, models the negation. The literal *knows* of arity two models the fact that an agent has some knowledge: $knows(a, b)$ means that agent a knows that b is true. The set of LAMBDA action models is called $\mathcal{A}$.

Correlation between actions

An attacker may execute several actions to modify the system state in order to reach a state where the security policy is violated. Some actions are executed in order to make the execution of other actions possible. When the effects of an action are a subset of the pre-conditions of another action, we say that the two actions are correlated. More formally, the notion of correlation between two LAMBDA actions is defined as follows:

Definition 23. *Let a and b be two LAMBDA descriptions of actions, $post(a)$ is the set of literals of the post-condition of a and $pre(b)$ is the set of literals of pre-condition of b.*
Correlation: *a and b are correlated if the following condition is satisfied:*
$\exists E_a$ and E_b such that

- $(E_a \in post(a) \wedge E_b \in pre(b)) or (not(E_a) \in post(a) \wedge not(E_b) \in pre(b))$

- *E_a and E_b are unifiable through a most general unifier u*

Given a set of LAMBDA action models, searching for such correlation links between action models results in a set of correlation rules. A correlation rule is a triple $\{A, B, u\}$, which represents the fact that model A is correlated with model B through the most general unifier u.

Correlation between instantiated actions

An action model is instantiated by assigning values to the free variables in the pre-condition. This is done by evaluating the detection field, but this may not be sufficient to assign values to all variables. The rest of the free variables are instantiated through unification with the system state, the system state being represented by a conjunction of literals with no free variables. More formally, an action model instance is a couple composed of the instantiated model and a finite substitution σ. The instance number i of action model A, denoted A_i, is $A_i = \{A, \sigma\}$. An instantiated action A_1 instance of model A and an instantiated action B_1 instance of model B are correlated iff A and B are correlated and if the corresponding unifier is satisfied. Correlated actions, models or instances, can be represented as graphs [Benferhat et al. 2003]. The set of action instances is denoted $\mathcal{A}_i$.

Intrusion objectives

An intrusion objective represents a state in which the system security policy is violated. An intrusion objective model description is composed of the following elements:

Definition 24. ***LAMBDA intrusion objective***
name: *the objective name*
condition: *defines the state of the system in which the system security policy is violated*

The condition is a conjunction of literals representing a violation of the security policy. Actions can be correlated with intrusion objectives using the same principle as in Definition 23 by replacing the $pre(b)$ set of literals by the intrusion objective condition.

4.2.2 Anticipating the attacker's intentions

Anticipating the intentions of the attacker consists in generating sequences of virtual action instances, i.e., actions not instantiated from alerts, so that the set of action instances created from alerts unified with the set of virtual action instances is correlated with an intrusion objective. The generation of such virtual actions is described in [Benferhat et al. 2003].

4.2.3 Intrusion Scenario

An intrusion scenario is defined as a set of action instances correlated with an intrusion objective as defined in [Benferhat et al. 2003]. The set of scenarios constructed from the set of action instances is denoted $\mathcal{S}$. We define the $hyp : \mathcal{S} \rightarrow \mathcal{A}_i$ function which returns the set of hypothesis in a scenario.

4.2.4 Modeling countermeasures

Countermeasures are actions which are executed to mitigate the effects of an attack or prevent the execution of other attacks. More generally, they are actions which have a negative effect on the execution of other actions. More formally, we model a countermeasure the same way an action is modeled except that its detection field is empty. A countermeasure is not instantiated from an alert, values are assigned to its free variables by examining the effects it must have on the system state in order to mitigate the effects of an attack or prevent the execution of other attacks. The notion of anti-correlation formalizes the notion of negative effect.

Anti-correlation between actions

The notion of anti-correlation between two LAMBDA actions is defined as follows:

Definition 25. *Let a and b be two LAMBDA descriptions of actions, $post(a)$ is the set of literals of the post-condition of a and $pre(b)$ is the set of literals of pre-condition of b.*
Anti-correlation: *a and b are anti-correlated if the following condition is satisfied: $\exists E_a$ and E_b such that*

- $(E_a \in post(a) \wedge not(E_b) \in pre(b)) or (not(E_a) \in post(a) \wedge E_b \in pre(b))$
- *E_a and E_b are unifiable through a most general unifier u*

Given a set of LAMBDA action models, searching for such anti-correlation links between action models results in a set of anti-correlation rules. An anti-correlation rule is a triple $\{A, B, u\}$, which represents the fact that model A is anti-correlated with model B through the most general unifier u.

We also define the notion of anti-correlation between an action and an intrusion objective by replacing the $pre(b)$ set of literals by the intrusion objective condition in definition 25. We introduce the *anticor* binary relationship to express that two action models or action instances are anti-correlated. If A and B are anti-correlated, then $(A, B) \in anticor$, which can also be represented by the fact that $anticor(A, B)$ is true.

Instantiating countermeasures

Given a scenario of instantiated actions correlated to an intrusion objective, we can create a set of countermeasures instances anti-correlated with either virtual actions or virtual intrusion objectives. Since a countermeasure is not instantiated from an alert, the free variables in its precondition and postcondition are instantiated using the unifier in the anti-correlation rule used to select the countermeasure.

4.3 Argumented intrusion response against attacks

Argumentation Frameworks (AFs) have been applied successfully to formalize non monotonic reasoning among other forms of reasoning. In the context of intrusion detection, the process of reacting against attacks can be seen as a form of non monotonic reasoning. Actually, given a set of detected attacks, it is possible to reason on the state of the system, described as in section 4.2.1, to choose countermeasures to apply among a set of possible system responses [Cuppens et al. 2006, Samarji et al. 2013]. As shown in the attack example presented in section 4.1, the set of countermeasures selected for an attack can evolve as new attacks are observed or if the system state evolves. In the context of intrusion detection, we believe that the process of reasoning on the observed attacks against a system to select the most adapted countermeasure for a given attack can be modeled as an argumentation process using a VAF.

We claim that modeling the attack and reaction processes using the semi-explicit correlation approach can be seen as two agents arguing against each other. On one side the attacker chooses his arguments, a set of actions, to try to reach an intrusion objective, and on the other side the agent defending the target chooses his arguments, a set of countermeasures, to block the attacker's progress or mitigate the attack effects.

We argue that the anti-correlation relationship between two LAMBDA model instances can be seen as an attack relationship over arguments.

Since we model the argumentation process using a VAF, a force, called here rationale, is associated with every action model. We add an extra element to an action model, the *rationale* attribute. This attribute models the reason motivating the execution of an action. From the attacker's point of view, this reason is related to the success of the attack. For example, the reason associated with the action of fingerprinting an operating system is to find vulnerabilities. From the point of view of the agent defending the system, the reason associated with the execution of a countermeasure is related to restoring some properties of the system. For example, adding a filtering rule to a firewall to block a host which connected to a server containing sensitive information is associated with the $confidentiality$ reason.

4.3.1 Constructing the set of arguments

The set of LAMBDA models Λ is the union of the LAMBDA action models set, $\mathcal{A}$ and the LAMBDA intrusion objective models set, $\mathcal{O}$, i.e., $\Lambda = \mathcal{A} \cup \mathcal{O}$. We denote by $\mathcal{A}_i$ and $\mathcal{O}_i$ respectively the sets of action instances and intrusion objective instances. Then $\Lambda_i = \mathcal{A}_i \cup \mathcal{O}_i$ is the set of all LAMBDA model instances. We denote by $\mathcal{L}$ the logic of predicates which is used to express the pre-condition, post-condition and system state condition of the action, reaction and intrusion objective models. The function $model : \Lambda_i \rightarrow \Lambda$ returns the LAMBDA model corresponding to a LAMBDA model instance. For an intrusion objective, the function $cond : \Lambda \rightarrow \mathcal{L}$ returns its system state condition.

Given an intrusion scenario S, constructed as specified in section 4.2.3, and the set of countermeasures C computed for S, we build the set of arguments used to reason as the union of the two sets:

Definition 26. ***Argument set:*** *the set $\mathcal{AR}(S)$ of arguments corresponding to an intrusion scenario S contains all the LAMBDA model instances of S plus all the LAMBDA countermeasures instances anti-correlated with the hypothesis of S:* $\mathcal{AR}(S) = S \cup \{cm \mid \forall h \in hyp(S), (anticor(post(cm), pre(h)) \vee anticor(post(cm), cond(h))) \wedge model(cm) \in (\mathcal{A} \cup \mathcal{O})\}$

Now that we know how to build the set of arguments corresponding to an intrusion scenario, we define the attack relationship *attacks* between arguments:

Definition 27. ***Attack relationship:*** *let S be an intrusion scenario and $\mathcal{AR}(S)$ the corresponding set of arguments. Let $a_1 \in \mathcal{AR}(S)$, $a_2 \in \mathcal{AR}(S)$ be two arguments. $attacks(a_1, a_2)$ is true iff $anticor(post(a_1), pre(a_2)) \vee anticor(post(a_1), cond(a_2))$*

Actually, in our model the effects of a countermeasure are characterized by its effects on the system through the specification of its post-condition, but it does not represent the reason why a countermeasure should be chosen. For instance, some countermeasures may enhance the performance of an attacked system to the detriment of the availability of some services. If for some reason the performance of the system should be favored over the availability of the services it provides, then we can choose the countermeasure associated to the favorite reason.

According to this modeling, we think that $VAFs$ are well-suited for our problematic since they allow to associate a value to each argument. However, in our case, the order relationship over the rationales associated with each argument is highly dependent on the context in which an attack is detected. In the next section we extend $VAFs$ to take into account the contextual aspect of our reasoning.

4.3.2 Extending value-based argumentation frameworks

Due to the dynamic nature of information systems, we argue that using a static preference relation $valpref$ is not adapted. We extend the definition of VAF to that of a Contextual VAF.

Definition 28. *A Contextual Value-based Argumentation Framework, denoted $CVAF$, is a 6-tuple $\langle AR, attacks, V, val, \mathcal{C}, ContPref \rangle$ where:*

- *AR, $attacks$, V and val have the same definition as in $VAFs$ (see Definition 13)*
- *$\mathcal{C}$ is a set of contexts. A context is either active or inactive. At a given time multiple contexts can be active*
- *$ContPref$ is a transitive, irreflexive and asymmetric preference relation on $V \times V$ which depends on the set of active contexts in $\mathcal{C}$*

Here a context represents a subset of the system state. More formally, the $hold(S, C)$ relationship is used to represent that some context C is active for some agent S. This relationship is inferred from the system state through derivation rules. For a context c, such derivation rule is defined as follows :

$$\bigwedge_{j=1}^{n} P_j \rightarrow holds(S, c)$$

where P_j is an n-ary predicate. Such derivation rule is called a context definition. For example the following context definition expresses that if an agent is on holidays, the $c_{holidays}$ context is active for this agent:

$on\_holidays(S) \rightarrow holds(S, c_{holidays})$

In this model we do not explicit the activation condition for each context in $\mathcal{C}$, we consider that this set is extracted from a contextual security policy specification, such as an OrBAC [Cuppens and Cuppens-Boulahia 2008] policy for example. The $ContPref$ relation is not defined for every possible combination of active contexts. A default order relation is defined and other definitions are specified for some active context combinations. If no definition is given for some combination of active contexts, then the default order relation applies. $ContPref$ has the same definition as the $valpref$ used in VAF except that it allows to generate the preference between the arguments forces depending on the current context configuration. To define exceptions in priority order for a subject (s) when a combination of (n) contexts is active, the system user defines the necessary updated preferences as following:

$$\bigwedge_{j=1}^{n} holds(s, c_j) \rightarrow ContPref(v_1, v_2) \wedge ... \wedge ContPref(v_{m-1}, v_m)$$

Where $c_j \in C$ and $v_1, v_2 ... v_{m-1}, v_m \in V$. We consider in this chapter that the elements of V are rationales that describe the application effect of the countermeasure on the system state.

4.3.3 Managing contexts

In our approach, values from the V set are interpreted as properties of the system which are favored by the associated arguments. These properties may be desirable in some context and should be avoided in some other context. For example, the availability property of some server may be desirable if the context $C_{high\_traffic}$ is active for this server, where $C_{high\_traffic}$ abstracts the fact that some server is under high network load.

The $favor(C, V)$ relationship expresses the fact that property V is desirable for context C. Conversely, the $avoid(C, V)$ relationship expresses the fact that property V is not desirable for context C.

Those relationships should be used by expert to express for each context which values from the V set are desirable or should be avoided.

The following derivation rules are used to derive which values are desirable or unwanted for an agent for which some contexts are active:

$holds(S, C) \wedge favor(C, V) \rightarrow wanted(S, C, V)$

$holds(S, C) \wedge avoid(C, V) \rightarrow unwanted(S, C, V)$

For each value from V, we define two sets containing respectively the couples (s, c) from the derived *wanted* and *unwanted* relationships:

$\forall v \in V, W_v = \{(s, c) \mid wanted(s, c, v)\}$

$\forall v \in V, U_v = \{(s, c) \mid unwanted(s, c, v)\}$

W_v contains the set of agents for which the same value v from V is a desirable property for all the active contexts for those agents. U_v contains the set of agents for which the same value v from V is not a desirable property for all the active contexts for those agents.

From those sets, we can compute a score s for each value v from V:

$\forall v \in V, s(v) = W_v - U_v$

This score is used to define an order relationship over the values in V. This order is used to derive the $contextualPref$ relationship:

$\forall v_1 \in V, \forall v_2 \in V, v_1 \neq v_2, s(v_1) > s(v_2) \rightarrow contextualPref(v_1, v_2)$

If two values from V have the same score, the default order relationship is used:

$\forall v_1 \in V, \forall v_2 \in V, v_1 \neq v_2, s(v_1) = s(v_2) \wedge defaultContextualPref(v_1, v_2) \rightarrow contextualPref(v_1, v_2)$

In the next section, we present how the system can take into account the context change during the reaction process.

4.3.4 Argumented and context aware reaction mechanism

Given an intrusion scenario, from the point of view of the agent defending the system under attack by another malicious agent, the reaction process consists in choosing among the possible countermeasures the best subset according to his/her preferences, those preferences being encoded in the $ContPref$ relation. According to our approach, this consists in using the attack relationship we have defined to build admissible sets of arguments, each set representing a coherent set of candidate countermeasures. Note that we only consider the argumentation process from the defending agent point of view, we do not try to construct extensions corresponding to actions the attacker could

make. Taking into account a context change, the system recomputes the admissible extensions according to the updated $ContPref$ relation. Building admissible extensions ensures us that they do not contain conflictual countermeasures, which would make the execution of the corresponding set of countermeasure impossible. The system operator manages the detected attacks and chooses the reaction that suits the best the current security state. The operator can choose the reaction among the stable extension of the arguments set $\mathcal{AR}(\mathcal{S})$ corresponding to the considered scenario S which offers the set of countermeasure that mitigate all the possible attacks that may occur on the system. Preferred extensions are maximal sets of arguments (with respect to set inclusion) that defend themselves against all attacks. The preferred extension of $\mathcal{AR}(\mathcal{S})$ is the maximal admissible set of arguments. According to the Theorem 1, $CVAF$ systems have a unique non-empty preferred extension which is also a ground extension and stable extension, since the oriented graphs associated to the $CVAF$ contain no cycle.

The process of countermeasures selection depends on the type of reasoning used by the system operator: The credulous reasoning consists in the selection of countermeasures (arguments) appearing in at least one preferred extension, this offers the system more intrusion response possibilities that may be defeated by other countermeasures. Whereas the skeptic reasoning consists in selecting countermeasures from the grounded extension which present the least (with respect to set inclusion) complete extension. In this kind of reasoning, the selected countermeasure will not be defeated by any other reaction model. We summarize our approach in Algorithm 1 and Algorithm 2 where we show how the security system generates the arguments set corresponding to a detected attack, and how it constructs the preferred extension.

Theorem 2. *Given a set of n actions and m intrusions objectives and p reactions in a an attack scenario, the complexity of the algorithm 1 is $\mathcal{O}((n+m)^2 + p \times n)$ in time.*

Proof. According to algorithm 1, the loop from line 3 to line 13 costs $\mathcal{O}((n+m)^2)$. The nested loops from line 14 to line 20 costs $\mathcal{O}(p \times n)$. Therefore, the overall time complexity of algorithm 1 is $\mathcal{O}((n+m)^2 + p \times n)$. □

Theorem 3. *Given an argument set of n arguments, the complexity of the algorithm 2 is $\mathcal{O}(n^3)$ in time.*

Proof. According to algorithm 2, the nested loops from line 2 to line 8 costs $\mathcal{O}(n^2)$. The nested loops from line 9 to line 15 costs $\mathcal{O}(n \times m)$ where m is the number of arguments in the preferred extension. In the worst case the number of arguments in the preferred extension is equal to the number of arguments in the argument set (i.e., the preferred extension in this case is the argument set). Thus, we consider the complexity in time

1: $current\_action \leftarrow detected\_action$
2: $\mathcal{S} \leftarrow current\_action$; $intrusion\_objective\_found \leftarrow false$
3: **do**
4: **for all** $model \in \mathcal{A} \cup \mathcal{O}$ **do**
5: **if** $correlated(current\_action, model) == true$ **then**
6: **if** $model \in \mathcal{O}$ **then**
7: $\mathcal{S} \leftarrow \mathcal{S} \cup \{model\}$; $intrusion\_objective\_found \leftarrow true$
8: **else**
9: $\mathcal{S} \leftarrow \mathcal{S} \cup \{model\}$; $current\_action \leftarrow model$
10: **end if**
11: **end if**
12: **end for**
13: **while**$(intrusion\_objective\_found == false)$
14: **for all** $action \in \mathcal{S}$ **do**
15: **for all** $reaction \in \mathcal{A}$ **do**
16: **if** $anticorrelated(action, reaction) == true$ **then**
17: $\mathcal{C} \leftarrow \mathcal{C} \cup \{reaction\}$
18: **end if**
19: **end for**
20: **end for**
21: $\mathcal{AR} \leftarrow \mathcal{C} \cup \mathcal{S}$
22: $GenerateArgumentsSet(detected\_action) \leftarrow \mathcal{AR}$

Algorithm 1: GenerateArgumentsSet(detected_action)

of this nested loops as $\mathcal{O}(n^2)$. Therefore, these two nested loops cost $\mathcal{O}(n^2)$. Finally, the main loop from line 1 to line 19 will be executed in the worst case n times since the argument set will be decremented in the worst case by one argument according to line 16. Therefore, the overall time complexity of algorithm 2 is $\mathcal{O}(n^3)$. □

4.3.5 Avoiding unexpected side effects of countermeasures

Now that we have built an exhaustive list of coherent and appropriate countermeasures, we focus on this section on how to avoid generating countermeasures having unexpected side effects. For this purpose, we propose the following VAF defined as follows:
$\langle Pref_{ext}, side\_effects\_attack, V, val, valpref \rangle$
Where $side\_effects\_attack$ is defined as follows:

1: **do**
2: **for all** $argument1 \in \mathcal{AR}$ **do**
3: **for all** $argument2 \in \mathcal{AR}$ **do**
4: **if** $\neg defeats(argument2, argument1)$ **then**
5: $PreferredExt \leftarrow PreferredExt \cup \{argument1\}$
6: **end if**
7: **end for**
8: **end for**
9: **for all** $argument1 \in \mathcal{AR}$ **do**
10: **for all** $argument2 \in PreferredExt$ **do**
11: **if** $defeats(argument2, argument1)$ **then**
12: $\mathcal{R} \leftarrow \mathcal{R} \cup \{argument1\}$
13: **end if**
14: **end for**
15: **end for**
16: $\mathcal{AR}' \leftarrow \mathcal{AR}/(PreferredExt \cup \mathcal{R})$
17: $attacks' \leftarrow attacks/((PreferredExt \times \mathcal{R}) \cup (\mathcal{R} \times \mathcal{AR}) \cup (\mathcal{AR} \times \mathcal{R}))$
18: **return** $PreferredExt \cup ConstructPreferredExtension(\mathcal{AR}', attacks')$
19: **while**$(PreferredExt! = \emptyset)$

Algorithm 2: ConstructPreferredExtension(AR,attacks)

Definition 29. ***Attack relationship:*** *let IS be an intrusion scenario and $Pref_{ext}(IS)$ the corresponding preferred set. Let $r_1 \in Pref_{ext}(IS)$, $a_1, a_2 \in \mathcal{AR}(IS)$ be two arguments. $side\_effects\_attack(r, a_2)$ is true iff:*
$anticor(post(r_1), pre(a_1)) \wedge cor(post(r_1), pre(a_2))$

In other words, two arguments $Arg_1 = \langle Cm_1, a_1, R_j \rangle$ and $Arg_2 = \langle Cm_1, a_2, R_k \rangle$ are attacking each other if a countermeasure Cm_1 generated among the preferred extension (from the previous Section) against an attack a_1 can help an attacker in another attack scenario to meet his/her intrusion objective by making the execution of action a_2 possible. In our approach, we consider that a countermeasure having side effects can be proposed to the security administrator only when the attack scenario that it is targeted against is more risky than the attack scenario that it can help (i.e., the attack scenario having the lower risk value is defeated by the other one).

The generated preferred extension according to the VAF presented in this section presents the final countermeasures list to be proposed to the security administrator and which satisfy the following conditions (i) no conflicts between parallel responses,

and (ii) no unexpected side effects of responses on the system. We consider in this section, the use case presented in [Samarji et al. 2015].

Use case

We consider two simultaneous threats led by two attack entities (A1 and A2) as shown in the generated intrusion scenario of Figure 4.1. In the initial system state, A1 has already infected machine M1 and actively scanned user U. In parallel, A2 has already infected machine M2 which belongs with M1 to the same Ethernet network (machines are reachable via $Switch12$). It is predicted for A1 to crack the password of U's account and highjack it in order to do a toll fraud which induces economic losses to U. Besides, a likely scenario for A2 is predicted starting by discovering M1 and then poisoning it with ARP messages, in order to spoof its address later on and make calls or inject packets as if they were sent by M1. We consider the response plan generated against threat A1 and described below:

$t1$: [[$passCrack(A1, server, u)$, $discovermacaddress(A2, M2, M1)$];
$t2$: [**disconnect(M1)**];
$t3$: [**install(SecurityPatch,M1)**, $injectRTPpackets(A2, M2, M1)$];
$t4$: [**connect(M1)**]]

The above sequence, presented in the graph of Figure 4.2, designs a response R3 against threat A1. R3 consists in patching the vulnerability of M1, and blocking thereby A1. By launching R3, thus disconnecting M1, after that A2 discovers the address of M1, A2 does no more need to perform ARP poisoning. Indeed, disconnecting a machine is like inducing a denial of service on this machine. Consequently, A2 can directly spoof the address of M1 and fulfill its attack objective. Consequently, R3 has a side effect on the system, by increasing the risk of threat A2. This side effect is generated since the following logical predicate returns true:

$$corr(connect(M1), Bot\_infection(M1))$$

We consider in this experimentation that R3 is the only candidate system response against A1. Two arguments are generated for this IS:

$$\mathcal{AR}(IS) = \{r1, r2\}$$

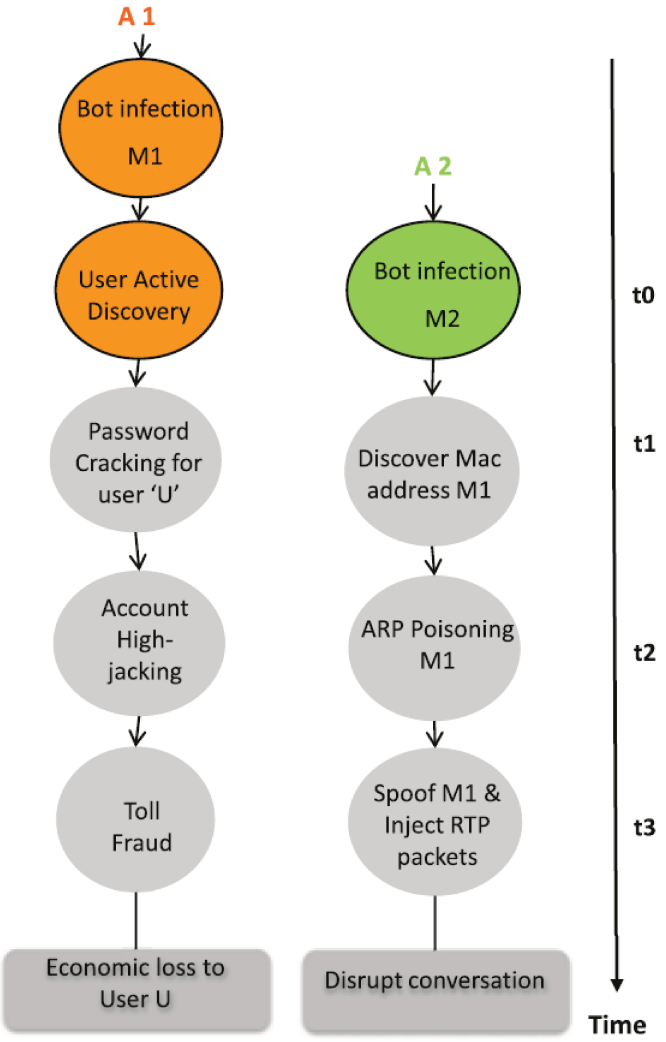


Figure 4.1: Intrusion Scenario : System threatened by two attack entities A1 and A2.

Where $r1 =< R3, spoof, Risk_{A1} >$ and $r2 =< not(R3), Bot\_infection, Risk_{A2} >$. R3 belongs in the preferred extension as a system response only if A1 threat is more risky than A2 threat.

4.3.6 Architecture

As shown in Figure 4.3, our system consists of three main parts involved in the generation process of system response against a detected alert:

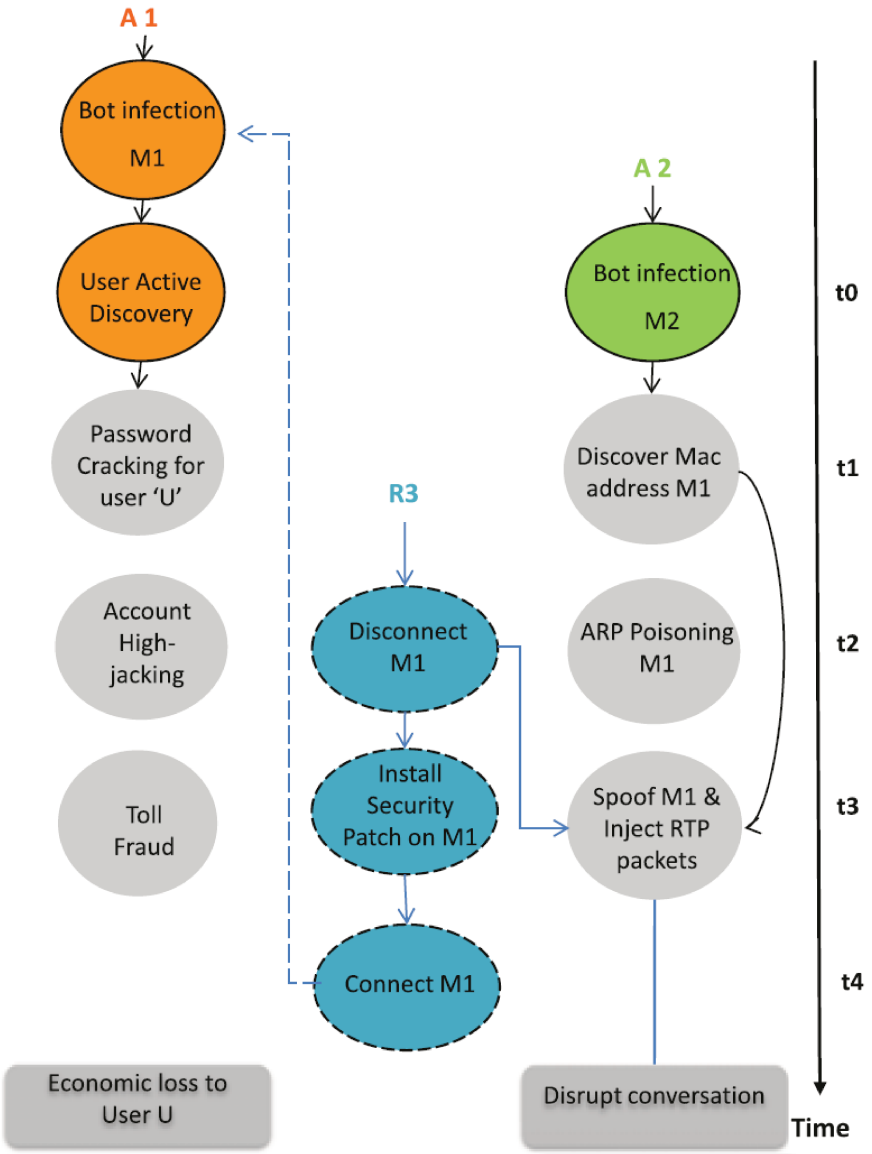


Figure 4.2: Response against A1, having side effects on A2.

Models

We consider three types of models: actions, intrusion objectives, countermeasures. For the description of the models we use the XML standard as follows:

- Actions: consists of four XML attributes (name, pre, post, detection)
- Intrusion objectives: consists of two XML attributes (name, condition)
- Countermeasures: consists of four XML attributes (name, pre, post, rationale)

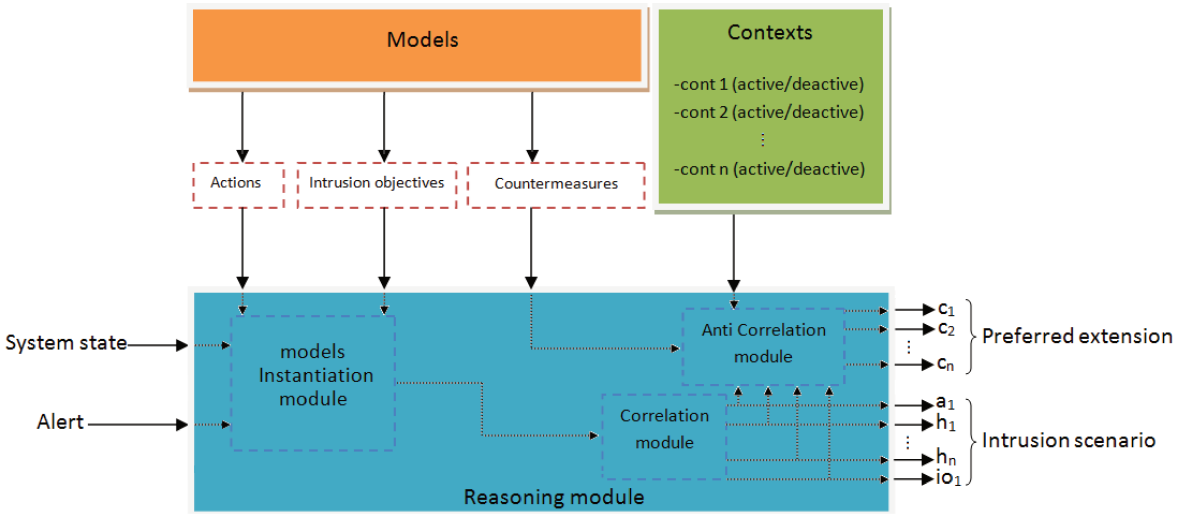


Figure 4.3: Overall system architecture

Models must be loaded just once by the system throughout its operation. A new load is required at each model update or when adding new models.

Contexts

In this part, the system manages the activation of the contexts in which it operates. At each context change, this module automatically updates the list of active contexts. This list will then be covered by the reasoning module to update the order of system priorities. The module detects the context change through the information reported by the various system components. For instance, in our case study (automotive system), several components allow to report on the different contexts in which the vehicle can operate (e.g., rain detection sensors, the clock, GPS).

Reasoning module

When loading an alert, the reasoning module loads the system state file to extract information concerning the current system state in order to instantiate the action corresponding to the detected alert. Then the instantiated action will be the input of the correlation module whose role is to generate and instantiate action hypothesis correlated to the detected action. This module allows to anticipate the attacker's intentions by predicting malicious actions that the attacker will be able to perform until reaching some intrusion objective. Once an alarm is loaded, an intrusion scenario is generated by the correlation module as an output. The generated scenario will be

an input for the anti-correlation module whose role is to generate a consistent set of countermeasures that prevent the intrusion scenario. Finally, the anti-correlation module refers to the "context" part to update the priority order by extracting the list of active contexts. This enables the reasoning module to generate the best set of countermeasures (i.e., the set that best fits the phase and the user requirements).

In the next section, we show the deployment of our approach in an automotive system using the credulous reasoning.

4.4 Reaction process in an automotive context

We apply our approach on an automotive system as an example of a case study to explore the issues that can meet complex systems during the reaction process.

4.4.1 Automotive system

In order to give an example of potential need for dynamic enforcement of security requirements to control different security activities, we consider the following abstract example of the automotive on-board system. A modern automotive on-board network interconnects a hundred of microcontrollers, termed Electronic Control Units (ECUs) organized into application-specific domains bridge by gateways, as shown in Figure 4.4. Each ECU is responsible for a basic functionality of the vehicle (e.g., Brake, direction, GPS signal). Thus, ensuring the security for these components presents a vital need for automotive systems.

Attacks have been shown to be quite feasible [Koscher et al. 2010] by bypassing the filtering performed between domains or by brute-forcing ECU cryptography-based protection mechanisms. Security vulnerabilities can be exploited to affect automotive system different components (e.g., lock/unlock car wheel at speed, disable brakes, kill engine, disable cylinders) [Koscher et al. 2010, Checkoway et al. 2011]. Such attacks may in practice originate from the Internet connection increasingly available in vehicles or even from the Bluetooth pairing of a compromised mobile phone to the vehicle on-board network. Further attacks are anticipated in upcoming Car2X applications, which will feature vehicle-to-vehicle (V2V) or vehicle-to-infrastructure (V2I) communications. Many security attacks and vulnerabilities are due to the fact that either security policy is not well specified and enforced or system-wide security policies (dependencies between different security policies) are too weak. Automotive on-board architectures

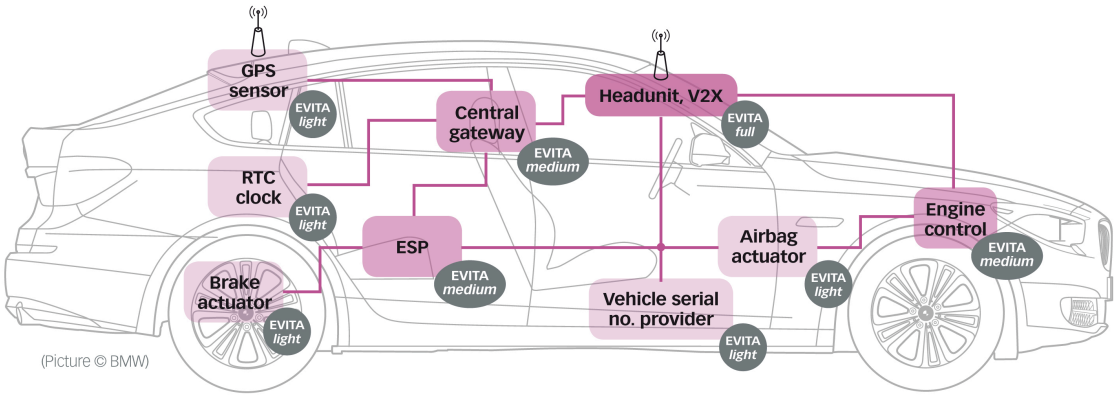


Figure 4.4: Automotive on-board network architecture [Project]

do not only rely on the simple enforcement of security rules, but also involve multiple enforcement points, especially when the underlying platforms and infrastructures are providing services themselves, like HSM, or middleware layers. For instance, the security policy to be applied in a vehicle is the combination of an invariant policy for the usage control of cryptographic credentials of Electronic Control Units (ECU), and a flexible networking security policy. The credential usage control policy is enforced by the HSM and possibly through the virtualization of the ECUs if applications on the same ECU have to be segregated. In contrast, the networking security policy is enforced by all network elements.

Moreover, the access control architecture must also allow enforcement of rules that limit the traffic on the buses under consideration, based on trusted authentication or other security mechanisms like traffic filtering or secure logging. However, as highlighted in the previous section, the enforcement of these different security mechanisms depends on a specific event or situation. For instance, while communicating with external entities like vehicle-to-infrastructure, it is preferable to apply the traffic filtering rules to limit the computation load on the HSM, which is responsible for the verification of cryptographic operations. Applying such rules will eventually increase the performance of on-board system. However, always applying such kind of rules is not desirable, as the enforcement of rules requires that the vehicle is in a specific context as well as a specific security event is active. To dynamically enforce these different sets of security policies, we call these policies as reaction policies [Autrel et al. 2009]. In an on-board architecture, we need a system in which policy enforcement decisions are based on specific arguments in order to attain more fine grained enforcement of

security policies.
Three main contexts are considered in this approach:

- *in_car*: Context defined to activate or deactivate specific activity in the vehicle. This context is defined to manage the vehicle different services (e.g., ESP, GPS sensor, Engine control) by controlling the ECU arranged in the architecture specific areas and connected by bridges as shown in Figure 4.4.
- $V2V$: Context defined to manage communication within vehicles. In the $V2V$ context, when two or more vehicles or roadside stations are in radio communication range, they connect automatically and establish a network enabling the sharing of position, speed, and direction data. Every vehicle is also a router and allows sending messages to more distant vehicles and roadside stations.
- $V2I$: Context defined to manage communication between vehicle and infrastructure. In $V2I$, the infrastructure plays a coordination role by gathering global or local information on traffic and road conditions and then suggesting or imposing certain behaviors on a group of vehicles. One example is ramp metering, already widely used, which requires limited sensors and actuators (measurements of traffic density on a highway and traffic lights on ramps).

Extra contexts are considered as well, describing the environment on which the vehicle may operate (e.g., highway, parking, rainy day, night).

4.4.2 Attack modeling

We consider in this section, that the automotive system detects a malicious action (a1) consisting in cracking the wifi passkey modeled as follows:

$$\begin{aligned} name: &\quad wifi\_passkey\_crack(A,T) \\ pre: &\quad role(T, wifi\_gateway) \wedge is\_on(T) \\ post: &\quad network\_access(A,T,wifi) \end{aligned}$$

The system generates the attacks that may be performed by correlation to reach a potential intrusion objective. In the following, we consider the “manipulation of relayed message” as a potential intrusion objective (io1) that the attacker can achieve through the “Message saturation" attack. Relayed messages are open to manipulation in an ITS-S (Vehicle) en route. Received messages that are intended for relaying can be withheld. An ITS-S is unable to determine quickly whether a received message is valid and from a legitimate user and then acts on information received in the message.

This intrusion objective is modeled as follows:

$$\begin{array}{ll} name: & manipulation\_relayed\_messages \\ condition: & manipulate(A,T) \wedge saturated\_server(T) \end{array}$$

Thus the system considers the "message saturation" as an attack hypothesis (h1) which consists in overflowing the ITS server with messages (Denial of Service attack). We consider several implementations of this attack: through wifi connection, Bluetooth connection or direct connection to the system bus. This attack is modeled as follows:

$$\begin{array}{ll} name: & message\_saturation(A,T,M) \\ pre: & network\_access(A,T,M) \wedge role(T, its\_server) \wedge is\_on(T) \\ post: & saturated\_server(T) \end{array}$$

Where the access type is held by the M variable. The $role(T, its\_server)$ means that the entity T acts as an ITS server

4.4.3 Response model

Once the attack scenario is generated, the system selects anti-correlated models that remedy the detected attack, the intrusion objective and all the attack hypothesis. For instance, the system has two intrusion responses against the passkey crack attack in the network: $disable\_wifi$ and $filter\_host$. We model these two countermeasures as the following:

$$\begin{array}{ll} name: & disable\_wifi(A,T) \\ pre: & is\_on(T) \wedge network\_access(A,T,wifi) \wedge is\_on(wifi) \\ post: & not(network\_access(A,T,wifi)) \wedge not(is\_on(wifi)) \\ rationale: & precaution \end{array}$$

The $disable\_wifi$ countermeasure is usually used when critical contexts are active and when we cannot predict the level and the current impact of the detected attack.

$$\begin{array}{ll} name: & filter\_host(A,T) \\ pre: & is\_on(T) \wedge network\_access(A,T,wifi) \wedge is\_on(wifi) \\ post: & not(network\_access(A,T,wifi)) \\ rationale: & availability \end{array}$$

Once the $filter\_host$ countermeasure is applied, the attacker, identified by his/her IP

address, cannot access the service. Here, we can identify a relation of anti-correlation between these countermeasures ($filter\_host$ requires, in its preconditions, that wifi must be on whereas $disable\_wifi$ turns the wifi off).

4.4.4 Rationales

We consider three main contexts in the automotive system:

- in_car: context defined to activate or deactivate specific activity in the vehicle
- V2V: context defined to manage communication within vehicles
- V2I: context defined to manage communication between vehicle and infrastructure

The rationales order is initially defined for the three main contexts. For instance, the rationales order in the in_car context is defined as follows:

$1.confidentiality, 2.performance, 3.availability, 4.integrity, 5.precaution.$

This rationales order depends on the context on which we reason. For instance, when communicating with external entities like vehicle to infrastructure ($V2I$), it is preferable to apply the traffic filtering rules to limit the computational load which ensures precaution for the system. However, applying such kind of countermeasures in the in_car context reduce the performance of the vehicle.
Functional experts in automotive systems define "performance" rationale as the different parameters allowing the vehicle to execute correctly its multiple functions. They define as well the rationale "precaution" as the capacity of the countermeasure to mitigate the corresponding detected attack. In other words, "precaution" reflects the severity level of countermeasures.
When reasoning in a specific main context, extra contexts can become active, which may change the rationales order. For instance, in the in_car context and the vehicle (a) is in $highway$ context the “precaution" rationale becomes more prioritized than “availability" and “integrity", because we become reasoning in a critical context. In a formal way, this exception is defined as follows:

$$holds(a, highway) \wedge holds(a, in\_car) \rightarrow$$
$$ContPref(precaution, availability) \wedge ContPref(availability, integrity)$$

4.4.5 Intrusion response selection

In this section, we consider the scenario of attack described in Section 4.4.2 we denote it $s1$. Once the system generates the attack scenario $s1$, it selects the appropriate intrusion response for $a1$ and also for potential attacks that may be triggered in coordination [Samarji et al. 2013] with it. The system generates the set of admissible arguments (countermeasures) for the detected and potential attacks. Here, the arguments set $\mathcal{AR}(s1)$ content is $\mathcal{AR}(s1) = \{a1, h1, io1, r1, r2, r3, r4, r5, r6, r7, r8\}$. Where:
$r1$ *filter host*: This countermeasure proposes to filter the suspect host and to isolate it from the ITS infrastructure.

$r2$ *disable wifi*: This countermeasure blocks the wifi connection inside the concerned vehicle to stop any suspected activities. It is a strict system response since it causes a loss of system availability.

$r3$ *reduce frequency*: This countermeasure proposes to reduce the frequency of the beacon and other safety-of-life messages from 10 Hz to a lower number to reduce congestion. An alternative solution is to use adaptive frequency control where messages would be sent at different frequencies depending upon the nature of the message, the availability of 5.9 GHz bandwidth, and potentially other local conditions.

$r4$ *add source identification*: A source address added to a $V2V$ message must be identifiable by the ITS receiving station and non-forgeable so that the receiving station can trust that the source address has not been modified between the time of message origination and the time the message was received.

$r5$ *limit message traffic*: An ITS-S is required to register (and authenticate) to the ITS infrastructure either when it enters an administrative region or at each roadside unit that comes into range if the roadside infrastructure is not extensive. Once registered, the vehicle accepts and processes only messages received from the ITS infrastructure while it is in radio range. When no roadside unit is in range then the ITS-S will receive and process ITS messages from other vehicles.

$r6$ *digitally sign each message*: The recipient of a message can gain confidence in the message's origin, the permissions of the originator, and its integrity against changes in transit if the message includes a digital signature or other form of cryptographic checksum and the recipient has the means to check that the checksum is valid.

$r7$ *include a non_cryptoghraphic checksum*: A simple approach to protecting the contents of a transmitted ITS message is to include a checksum computed from the original contents. The receiving ITS-S is then able to calculate the checksum itself and compare it with the value included in the incoming message. If the checksum values do not match, the received message can be rejected. A simple longitudinal parity check would probably be insufficient for the purpose of establishing the integrity of a received ITS message, but the more reliable Fletcher or Adler algorithms would provide the necessary protection. These algorithms, unfortunately, require more processing resources in both the sending and receiving ITS-S.

$r8$ *remove requirements for message relay*: The propagation of ITS messages to emulate a wide-area broadcast (particularly in an emergency situation) is achieved by allowing an ITS-S (Vehicle) to re-broadcast any received message that has not reached the edge of its relevance area. Removing this capability makes it impossible for a message to be modified en route. This can only be achieved if the roadside infrastructure is sufficient to receive the original message and to transmit it across the whole of the relevance area.

The intrusion response ($r1$, $r2$) and ($r2$, $r8$) are attacking each other according to the anti-correlation definition ($r1$ and $r8$ need that the wifi connection must be on in their preconditions). $r1$ and $r8$ defeat $r2$ when the system is reasoning in *in_car* context defined by default as shown in Figure 4.5 where defeated countermeasures are presented in grey.

The system generates the preferred extension $Pref_{ext}$=$\{r1, r3, r4, r5, r6, r7, r8\}$ which presents the maximal (with respect to set inclusion) admissible set of $\mathcal{AR}(s1)$. The generation of the preferred extension depends on the current active contexts. For instance, when reasoning in the *highway* context, the system updates the rational order as described in the previous section. Thus, the $disable\_wifi$ countermeasure defeats $filter\_host$ and *no_cryp_cheksum* since the rationale of $r2$ which is "precaution" becomes more prioritized than "performance" (the rationale behind $r1$ and $r8$) as shown in the Figure 4.6. The updated preferred extension becomes $Pref_{ext}$=$\{r2, r3, r4, r5, r6, r7\}$.

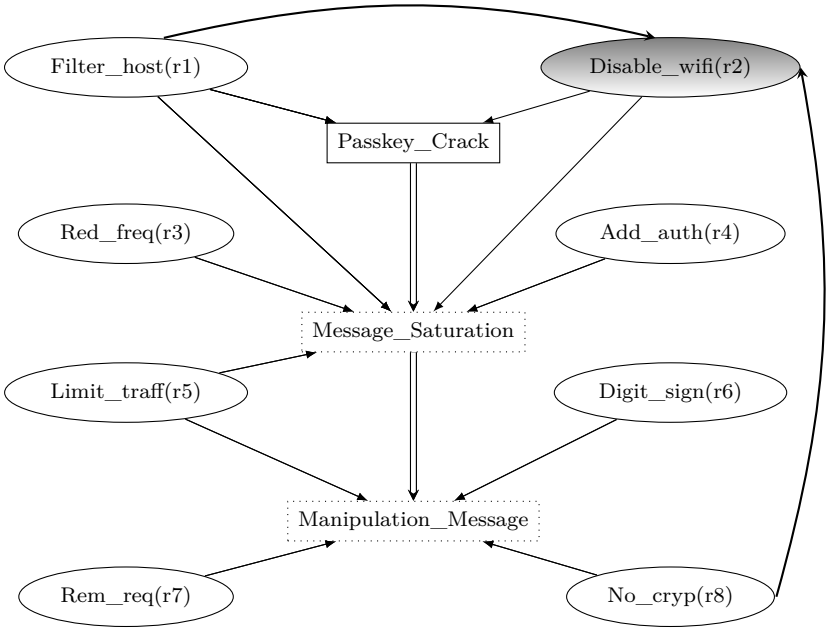


Figure 4.5: System response against crack passkey attack in {in_car} context

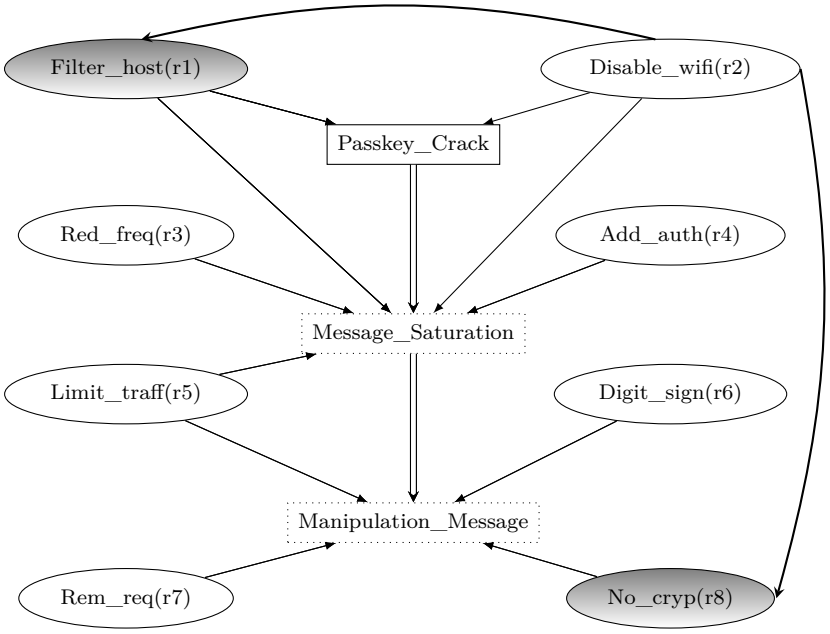


Figure 4.6: System response against crack passkey attack in {in_car,high_way} context

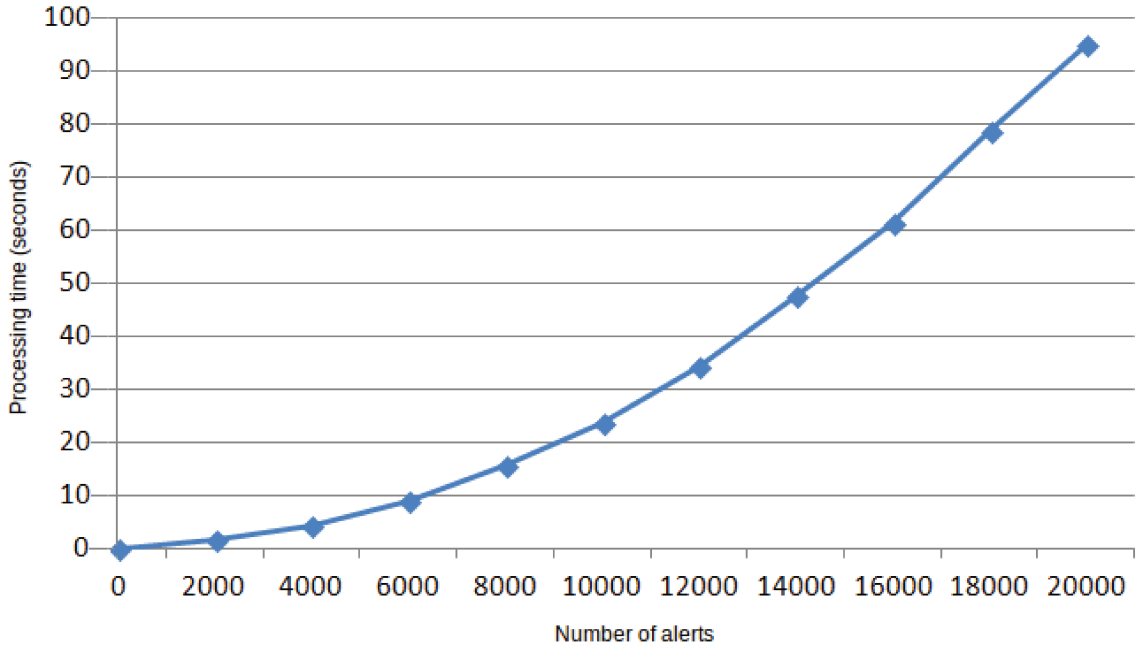


Figure 4.7: Preferred extension generation: Execution time per number of detected alerts

4.4.6 Performance evaluation

To evaluate the performance of our approach, we tested for different number of detected alerts, the time required for the system response. All experimentations in this chapter have been performed on a four-core server (Intel(R) Core(TM) i5-3230M CPU @ 2.60GHz) with 8 GB of RAM and running Ubuntu Linux 14.04. Figure 4.7 shows the preferred extension generation costs per number of detected attacks. According to the results, the system can load and process 2138 alerts in one second. We judge that one second is a reasonable time interval for processing alerts since the role of the response module is not to remedy a detected intrusion objective, but rather to prevent a potential one (i.e., the response module is considered in a phase advance with respect to the attacker). Thus, we consider that the response module we implemented can process 2138 alerts in real-time when considering the same experimental conditions.

4.5 Conclusion

In order to protect a system from modern attacks, it is necessary to have a dynamic and intelligent enforcement of security policies. We consider the argumentative logic driven

system the most appropriate to achieve this objective. We have proposed an approach based on the argumentative logic and modeled via LAMBDA models which ensures the automatic intrusion responses for security system. This approach considers the different active contexts on which the system is operating. In this chapter, we showed how to improve the existing argumentation framework. We proposed a new approach that allows us to to take the suitable and dynamic decisions that maintain the system in safe conditions while satisfying the prioritized system requirements. In the proposed approach, the reasoning module refers to a rational order which is manually predefined by the system expert. The next chapter will focus on the integration of a recommender module to the current architecture. The main role of the recommender module is to provide pertinent decisions among those proposed by the generated preferred extension. Next chapter presents a more enhanced approach where the rational order could be automatically established by referring to the user historic.

CHAPTER 5

Multi-Criteria Recommender Tool for Supporting Intrusion Response System

5.1 Introduction

This chapter introduces an approach based on a recommender system for efficient security administrator's assistance in the context of reaction against intrusion detection. Recommender systems are tools for processing and organizing information in order to give assistance to the system users. This assistance is provided by analyzing their own preferences or the preferences of their community. The proposed methodology considers the set of active contexts while analyzing the security administrator decisions historic. It provides better recommendation depending on the contexts in which the system is operating. We propose in this chapter an approach based on a recommender system using Multi-Criteria Decision Making (MCDM) method for assisting system security administrators while selecting the appropriate countermeasures against a specific attack scenario. This approach considers the different effects a countermeasure could have on the system as criteria to be considered when selecting the appropriate countermeasures. The objective of this approach is not to replace the security administrator during the countermeasures selection process, but rather to recommend system responses based on the security administrator decisions historic. This approach permits also, to automatically select appropriate countermeasures in critical cases where the system security administrator is unable to select them.

This chapter introduces an MCDM approach for security administrator assistance. It shows as well how to integrate the MCDM module into system response against intrusion detection approach presented in the previous chapter. Finally, it presents

deployment scenarios highlighting how our approach is applied in the use case of automotive systems.

5.2 Related work

The aim of this section is to address the recommendation problem from the MCDM perspective and to demonstrate the interest of applying MCDM methods to design multi-criteria recommender systems. There are three basic approaches for recommender systems: the content-based recommendation [Pazzani and Billsus 1997], collaborative filtering [Resnick et al. 1994] and a hybrid approach [Balabanovic and Shoham 1997] that combines collaborative and content-based methods. The collaborative filtering approach consists in collecting evaluations about the different contents and generating predictions for the user about a specific content by comparing them with the evaluations done by users with similar tastes and preferences.
The Content-based approach focuses only on the user evaluations to generate recommendations. This approach consists in analyzing the user evaluations historic to identify the user common features of interest. The work done in [Pazzani and Billsus 1997] presents an approach that collects user evaluations of the interest of visited pages on the World Wide Web. The authors show that a user profile can learn from this information and use it to recommend other pages that may interest the user.

There exists several contributions showing recommender systems that engage some MCDM methods as presented in [Manouselis and Costopoulou 2007, Adomavicius et al. 2011]. The authors in [Montibeller and Franco 2010] propose a framework to support strategic decision making in an organization. The proposed framework employs Multi-Criteria Decision Analysis to support decision making in strategy workshops. This framework takes into account the organization modern nature which is less hierarchic and more participative with a more distributed knowledge and decision taking. The approach proposed in [Montibeller and Franco 2010] considers the multiple objectives aspect that must satisfy the organization strategic decision. However, this framework presents a high level of uncertainty. In [Zeleny 1982], Zeleny proposes to increase the decider confidence and to limit to the post-decision regrets. Zeleny proves how pre-decision and post-decision steps are interdependent. In [Chiprianov et al. 2013], the authors propose to model the MCDM process using Model Driven Engineering approaches. The proposed approach offers a guidance for the analyst and improves the communication between deciders and analysts. More related to the security field, the authors in [Oglaza et al. 2014] propose a novel approach that combines an MCDM approach called KAPUER with classic access control

tools to assist users while writing high level permission rules. This approach includes algorithms that converge after the first phase of initializing user preferences.

In this chapter, we propose a recommender system based on the content-based approach to assist the system security administrator in choosing the most appropriate countermeasures according to his/her requirements, given a specific attack scenario. Using a simple recommender system is not appropriate in the context of system response against intrusion detection. It cannot take into account the multiple dimensions of the impact that a countermeasure could have on the system state. The recommended tool proposed in this work apply MCDM methods to consider the multiple criteria nature of countermeasures. There exists several MCDM methods for calculating alternatives scores:

Technique for Order Preferences by Similarity to Ideal Solutions (TOPSIS) [Hwang et al. 1993]

In this method two artificial alternatives are defined:

- Ideal alternative: alternative having the best level for all attributes considered.
- Worst alternative: alternative having the worst attribute values.

TOPSIS selects the alternative that is the closest to the ideal solution and farthest from negative ideal alternative. TOPSIS assumes that we have m alternatives and n criteria and we have the score of each option with respect to each criterion. Let x_{ij} score of option i with respect to criterion j We have a matrix $X = (xij)$, $m \times n$ matrix. Let J be the set of benefit criteria (more is better). Let J' be the set of negative criteria (less is better). First, we construct a normalized decision matrix. This step transforms various attribute dimensions into non-dimensional attributes, which allows comparisons across criteria. Normalized scores are calculated as follows:
$r_{ij} = \frac{x_{ij}}{\sqrt{\sum_{i=1}^{m} x_{ij}^2}}$, $i = 1, .., m$; $j = 1, .., n$
Then, we construct the weighted normalized decision matrix. Assume we have a set of weights for each criteria w_j for $j = 1, .., n$. We multiply each column of the normalized decision matrix by its associated weight. An element of the new matrix is: $v_{ij} = w_j \times r_{ij}$.
After that, we determine the ideal and worst solutions as follows:
Ideal solution: $A_i = v_1*, .., v_n*$, $where\ v_j* = \{max(v_{ij})\ if\ j \in J\ ;\ min(v_{ij})\ if\ j \in J'\}$
Worst solution: $A_w = v'_1, .., v'_n$, $where\ v'_j = \{min(v_{ij})\ if\ j \in J\ ;\ max(v_{ij})\ if\ j \in J'\}$
In the next step, we calculate the distance d_{jw} between the target alternative j and the worst solution A_w, as well as the distance d_{ji} between the target alternative j and

the ideal solution A_i. Then, we calculate relative closeness to the ideal solution S as follows:
$S = \frac{d_{jw}}{d_{jw}+d_{ji}}$ Finally, we select the Alternative with S closest to 1.

ELECTRE [Bouyssou and Roy 1993]

The ELECTRE methods are based on the following four preference situations concerning the comparison of two decisions:

- I (Indifference): it corresponds to a situation where there are clear and positive reasons that justify an equivalence between the two actions (it leads to a reflexive and symmetric binary relation).
- P (Strict Preference): it corresponds to a situation where there are clear and positive reasons in favor of one (identified) of the two actions (it leads to a non-reflexive and asymmetric binary relation).
- Q (Weak Preference): it corresponds to a situation where there are clear and positive reasons that invalidate strict preference in favor of one (identified) of the two actions, but they are insufficient to deduce either the strict preference in favor of the other action or indifference between both actions, thereby not allowing either of the two preceding situations to be distinguished as appropriate (it leads to a non-reflexive and asymmetric binary relation).
- R (Incomparability): it corresponds to an absence of clear and positive reasons that would justify any of the three preceding relations (it leads to a non-reflexive and symmetric binary relation).

There are two main parts of an ELECTRE application: first, the construction of one or several outranking relations, which aims at comparing in a comprehensive way each pair of decisions; second, an exploitation procedure that elaborates on the recommendations obtained in the first phase. The nature of the recommendation depends on the problem being addressed: choosing, ranking or sorting. Usually the Electre Methods are used to discard some alternatives to the problem, which are unacceptable. After that we can use another MCDM to select the best one. Criteria in ELECTRE methods have two distinct sets of parameters: the importance coefficients and the veto thresholds.

AHP [Saaty]

AHP uses a hierarchical structure and pairwise comparisons. An AHP hierarchy has at least three levels:

- The main objective of the problem at the top.
- Multiple criteria that define alternatives in the middle (m).
- Competing alternatives at the bottom (n).

In AHP method, criteria weighting must be determined using $(m*(m-1))/2$ pairwise comparisons. Alternatives scoring using $m*((n*(n-1))/2)$ pairwise comparisons between alternatives for each criteria. After completing pairwise comparisons, AHP is just the hierarchical application of SAW method.

SAW [Afshari et al. 2010]

Alternatives scores are calculated using SAW method as follows:

Definition 30. *Simple Additive Weighting (SAW) method*

$$\forall i \in \{1, N\}, S_i = \sum_{j=1}^{M} w_j \times r_{ij}$$

Where:
S_i *is the overall score of the* i^{th} *alternative,*
r_{ij} *is the rating of the* i^{th} *alternative for the* j^{th} *criterion,*
w_j *is the weight (importance) of the* j^{th} *criterion,*
N *the number of alternatives and* M *the number of criteria.*

Based on the literature reviewed [Aruldoss et al. 2013], the observed advantages and disadvantages of the MCDM methods previously introduced are summarized in Table 5.2.

Method	Advantages	Disadvantages
TOPSIS	Has a simple process; easy to use and program; the number of steps remains the same regardless of the number of attributes.	Its use of Euclidean Distance does not consider the correlation of attributes; difficult to weight and keep consistency of judgment.
ELECTRE	Takes uncertainty and vagueness into account.	Its process and outcome can be difficult to explain in layman's terms; outranking causes the strengths and weaknesses of the alternatives to not be directly identified.
AHP	Easy to use; scalable; hierarchy structure can easily adjust to fit many sized problems; not data intensive.	Problems due to interdependence between criteria and alternatives; can lead to inconsistencies between judgment and ranking criteria; rank reversal.
SAW	Ability to compensate among criteria; intuitive to decision makers; calculation is simple does not require complex computer programs.	Estimates revealed do not always reflect the real situation; result obtained may not be logical.

Table 5.1: Summary of MCDM methods

5.3 Multi-Criteria Decision Making module

We designed an MCDM module to support security administrator during system response against intrusion detection. The recommender system we designed follows an iterative process as shown in Figure 5.1. This iterative process is triggered when the system detects an intrusion and generates its preferred extension. The security administrator selects countermeasures among recommended ones generated from the preferred extension. The system consults selected countermeasures evaluations according to a predefined criteria list. The criteria list presents a list of nominal system functional behavior (e.g., availability, integrity, performance). The evaluation consists in assigning to each criterion a mention that describes the impact level the countermeasure could have on the system state. The possible evaluations are (Very Low(0), Low(1),

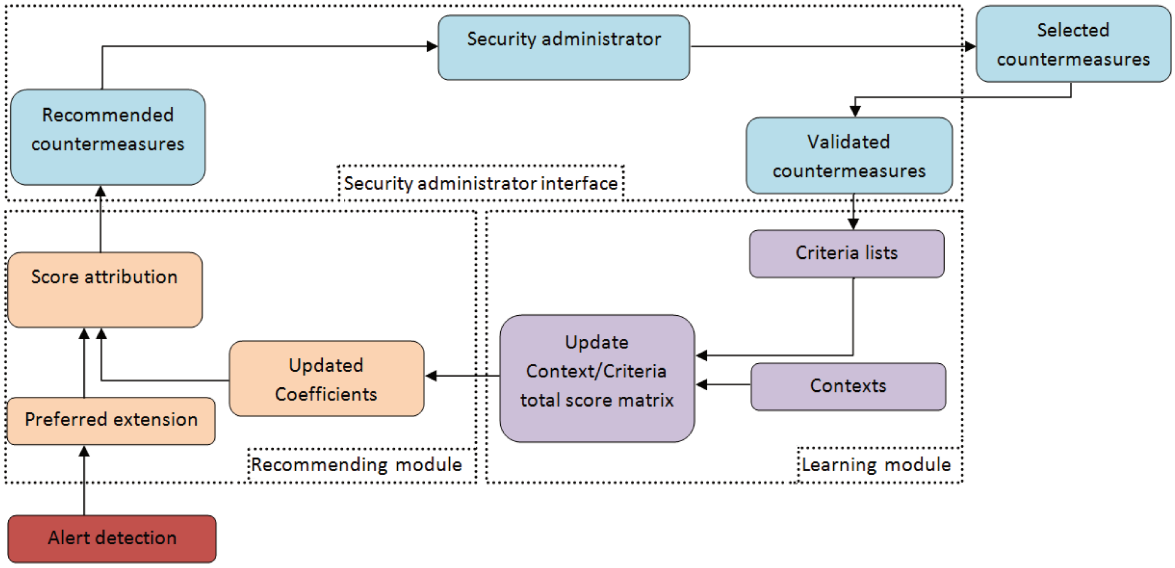


Figure 5.1: Recommender system architecture

Medium(2), High(3), Very High(4)). "Very High" implies that the countermeasure has the highest impact level on the system according to the considered criterion (e.g., countermeasures that disable wifi connection ensure a very high level of precaution). This evaluation is done by functional experts that study the different effects a countermeasure could have on the system state (e.g., level of availability loss). Then, the system checks the list of active contexts during countermeasures selection time by the security administrator.

To learn more about the security administrator way of reacting, we propose a learning module which learns about the security administrator preferences and requirements according to his/her decisions historic. Preferences in intrusion response context are considered in our approach as the criteria having the highest score according to a specific combination of active contexts. We define the Context/Criteria matrix, which constitutes the core of the learning phase and which allows to analyze choices made by the security administrator. It is often called an options matrix or a decision table. This matrix is updated when countermeasures are selected by the security administrator and depending on the context on which the system is operating. For instance, when a security administrator always selects countermeasures having a high score on "precaution" criterion in a critical context, Criteria/Context matrix generates "precaution" as the most preferred criterion by the security administrator in this critical context. Thus, the recommendation of some countermeasures among all the preferred extension refers to each criterion score provided by the Context/Criteria matrix.

Different parts are involved in the Multi-Criteria Decision Making process such as the learning module, the recommending module, and the security administrator interface.

5.3.1 Learning module

We define the learning part as the different process allowing to give a visibility about the security administrator way of reacting and the different parameters influencing his/her decisions. This part is supplied by the security administrator decisions historic by analyzing the score of different criteria. Each time the security administrator validates a decision, the Context/Criteria matrix is updated with the selected countermeasure values according to each criterion. The update process of the Context/Criteria matrix according to n selected countermeasures is established as follows:

Definition 31. *Let Mat be a Context/Criteria matrix, Mat is a matrix of integers, $j \in card(criteria)$ and $criteria(CM_i, j)$ a function returning the j^{th} criterion evaluation corresponding to the CM_i countermeasure.*

$$Mat_{updated}[contconfig][j] = Mat_{current}[contconfig][j] + \sum_{i=1}^{n} criteria(CM_i, j)$$

Where $contconfig$ represents the current combination of active contexts, and $criteria(CM_i, j) \in [0, 4]$ as described in Section 5.3

5.3.2 Recommending module

The recommendation phase is based on the Context/Criteria matrix to determine the decider favored criteria per context. The recommending module calculates the j^{th} coefficient criterion as follows:

Definition 32. *Let Mat be a Context/Criteria matrix, $n = card(criteria)$ and $j \in \{1..n\}$*

$$coeff(j, contconfig) = \frac{Mat[contconfig][j]}{\sum_{i=1}^{n} Mat[contconfig][i]}$$

Where $coeff(j, contconfig) \in [0, 1]$

Coefficients are then used in the score assignment phase where candidate countermeasures are evaluated based on their value per criterion and the criterion coefficient.

The sum of all coefficients must be 1. The score of each candidate countermeasure must be calculated upon dynamic criteria coefficients to reflect a score compatible with the importance the security administrator assigned to each criterion according to each set of active contexts.

The score assignment presents the last phase of recommendation, which consists in a dynamic assignment of score to each proposed countermeasure so that the system can compare them and recommend the most relevant ones.

In this approach, we opted for SAW (Simple Additive Weighting) method which evaluates alternatives based on two metrics: the performance value of the alternative in term of a specific criterion, and the relative weight of importance of this criterion. SAW method is applicable only when all alternatives are evaluated in the same unit. Otherwise, other methods, such AHP for example, that allow to standardize alternatives evaluations, should be applied. As presented in this section, possible evaluations of all countermeasures are standardized (i.e., possible countermeasures evaluations according to each criterion are: $1, 2, 3$ and 4). Thus, SAW method is applicable in our case of study. In addition, SAW is known as the simplest and the fastest MCDM method. This will be helpful when designing system's response against intrusion detection since such systems must respond to real-time constraints especially in critical contexts. The score of each proposed countermeasure is calculated using the SAW method as follows:

Definition 33. *Let CM be a countermeasure, $n = card(criteria)$, $j \in \{1..n\}$ and $criteria(CM, j) \in [0, 4]$*

$$Score(CM, contconfig) = \sum_{j=1}^{n} (criteria(CM, j) \times coeff(j, contconfig))$$

5.3.3 Security administrator interface

The aim of the recommending system is to assist security administrators and show them the points that alone they are not able to see. As explained in Section 5.3.1, the security administrator provides information that supply the learning module. Each time the security administrator selects some countermeasures, he/she is asked to validate his/her decision. The validation phase allows the learning module to consider only the decisions that satisfy the security administrator, the learning module does not consider the administrator regrettable countermeasures. Once the administrator validates his/her decisions, the learning module updates the Context/Criteria matrix,

to take into account the new decisions.

5.4 MCDM module integration with intrusion response system

The overall system architecture is described in Figure 5.2, where the MCDM module is integrated with the intrusion response architecture presented in the previous chapter. As presented in the previous chapter, the reasoning module refers to a criteria order called "rational order" which is manually predefined by the system expert. In

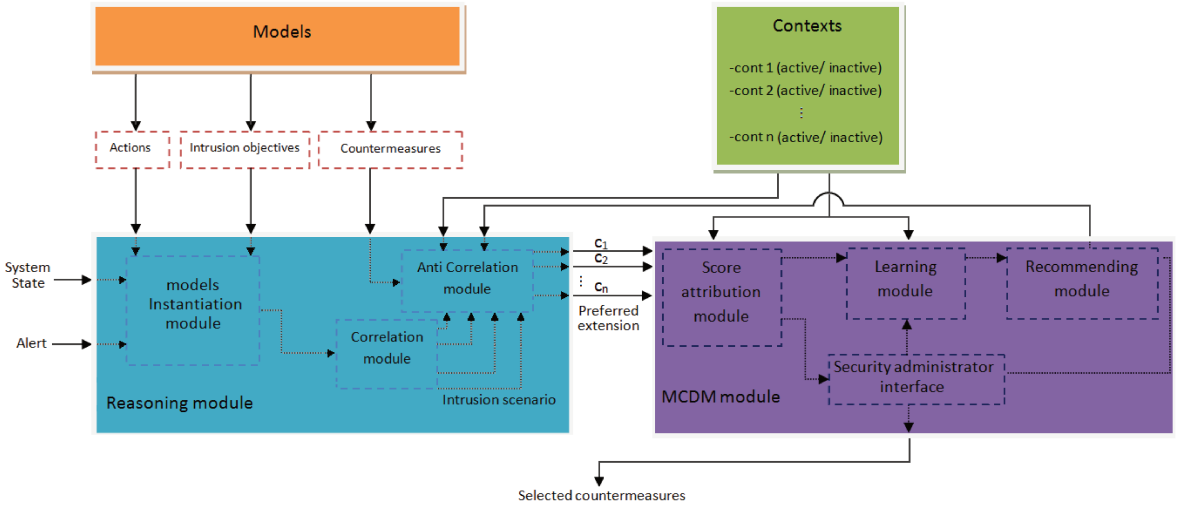


Figure 5.2: Overall system architecture

this chapter, we integrate the MCDM module to the existing architecture. In this approach, the criteria order will be automatically established by referring to the security administrator decisions historic.
The overall system process can be summarized in four steps.

5.4.1 Prediction phase

Reacting in a critical context against an attack after its execution can not always mitigate the adverse effects of the attack. In these cases it is essential to anticipate the attacker's intentions and to take precautionary measures to prevent the attacker from reaching his/her intrusion objective. For this pur-

pose, the approach proposed in previous chapter consists on instantiating actions hypothesis correlated to the detected malicious action. We consider the example where the system detects an action consisting on cracking the wifi passkey. The system instantiates correlated attack hypothesis that the attacker may execute. The system considers *message_saturation* as a correlated action. *Message_saturation* is considered as a correlated action, since *wifi_passkey_crack* postconditions are a subset of *message_saturation* preconditions. The system generates as well *manipulation_relayed_messages* as a potential intrusion objective correlated to *message_saturation*. *Manipulation_relayed_messages* is then considered as the intrusion objective that the attacker can achieve starting from *wifi_passkey_crack* and through the *message_saturation* attack. According to the example, an attack scenario {*wifi_passkey_crack*, *message_saturation*, *manipulation_relayed_messages*} is to be considered in the responses generation phase and not only the *wifi_passkey_crack*.

5.4.2 System response generation phase

In this phase, the Anti Correlation module selects countermeasures that are anti correlated to the generated attack scenario. To avoid generating a response set containing conflictual countermeasures, we proposed in the previous chapter an approach that refers to a mono criterion evaluation to determine which countermeasure should be selected. In this chapter, we propose a more enhanced approach by integrating the MCDM module. This module allows a more adaptive evaluation between countermeasures by comparing their scores. This allows an evaluation that covers all the countermeasures impacts on the system, rather than considering the main effect of a countermeasure on the system as the only criterion to be considered. MCDM module intervenes in the proposed framework at two levels:

- It automatically updates the criteria order according to the set of active contexts in the preferred extension generation phase.
- It generates the recommended countermeasures among the preferred extension.

5.4.3 Recommendation phase

Once a preferred extension is generated according to a specific attack scenario, all generated countermeasures are subdivided into criteria evaluations. The recommender system refers to the current Context/Criteria matrix to generate the criteria order and

coefficients as described in Section 5.3. Then, the system calculates countermeasures scores starting from countermeasures criteria evaluations and the criteria order and coefficients as defined in Definition 33. Countermeasures having highest scores will be recommended over the other system responses from the preferred extension. Security administrators are asked to select countermeasures that satisfy their preferences and requirements according to the current set of active contexts. This module is summarized in Algorithm 3.

1: $Sum\_per\_context = 0$
2: **for all** $criterion \in Criteria$ **do**
3: $Sum\_per\_context \leftarrow Sum\_per\_context + Mat[contconfig][criterion]$
4: **end for**
5: **for all** $criterion \in Criteria$ **do**
6: $coeff(criterion, contconfig) \leftarrow Mat[contconfig][criterion]$ / $Sum\_per\_context$
7: **end for**
8: **for all** $argument \in PreferredExt$ **do**
9: $Score(argument, contconfig) = 0$
10: **for all** $criterion \in Criteria$ **do**
11: $Score(argument, contconfig) \leftarrow Score(argument, contconfig) + criteria(argument, criterion) \times coeff(criterion, contconfig)$
12: **end for**
13: **end for**
14: $Recommended\_Countermeasure\_List \leftarrow order(PreferredExt, contconfig)$
15: **return** $Recommended\_Countermeasure\_List$

Algorithm 3: ConstructRecommendedList(PreferredExt,contconfig)

Theorem 4. *Given a set of N generated countermeasures and M criteria, the complexity of the algorithm 3 is $\mathcal{O}(N \times M + Nlog(N))$ in time.*

Proof. According to algorithm 3, the loop from line 2 to line 4 costs $\mathcal{O}(M)$. The second loop (from line 5 to line 7) costs also $\mathcal{O}(M)$. The nested loops (from line 8 to line 13) costs $\mathcal{O}(N \times M)$. Finally, the execution of the function *order* (line 14)costs in the worst case $\mathcal{O}(Nlog(N))$, since it uses merge sort. Therefore, the overall time complexity of algorithm 3 is $\mathcal{O}(N \times M + Nlog(N))$. □

5.4.4 Matrix update phase

The Context/Criteria matrix is updated when countermeasures are selected by the security administrator and depending on the context in which the system is operating. When the security administrator selects countermeasures from the preferred extension, the Context/Criteria matrix is updated by adding selected countermeasures evaluations according to each criterion to the matrix current scores. This phase provides information concerning the security administrator preferences and requirements using his/her personal decisions historic organized according to the different criteria. It provides criteria order according to each set of active contexts as well as the coefficient of importance to give to each criterion. In this approach, the Context/Criteria matrix supports the generation of preferred extension by updating the criteria order and coefficients, which allows to apply the $defeat$ notion between countermeasures by comparing countermeasures scores.

The inputs of the overall architecture, as shown in Figure 5.2 are:

- System state: presented as a set of literals describing the current system state
- Alert: corresponds to an action detection
- Models: actions, intrusion objectives, countermeasures models
- Contexts: supplies the reasoning module and the MCDM module with the current set of active contexts

The overall system main outputs are countermeasures selected by the security administrator.

5.5 Application to the automotive case of study

We show in this section, the deployment of our approach in the automotive system as an example of a case study and we present the evaluation of the proposed approach.

5.5.1 Deployment scenario

We consider in this section, the attack scenario S presented in the previous chapter and described below:

- The automotive system detects a malicious action consisting in cracking the wifi passkey $Wifi\_passkey\_crack$
- The system generates $message\_saturation$ as an attack hypothesis correlated to $Wifi\_passkey\_crack$
- The system generates $manipulation\_relayed\_messages$ as an intrusion objective for the attacker

As an output of the reasoning module presented in the previous chapter, the system generates a preferred extension that consists in a coherent set of candidate countermeasures. The generation process of the preferred extension depends on the current active contexts. The Figure 4.5 and Figure 4.6 presented in the previous chapter show that the preferred extension corresponding to S scenario are $Ex\_pref_{\{in\_car\}}$ and $Ex\_pref_{\{in\_car,high\_way\}}$ respectively, for $\{in\_car\}$ and $\{in\_car, high\_way\}$ context.

Contexts	Integrity	Availability	Confidentiality	Performance	Precaution
$\{in\_car\}$	64	88	**110**	104	22
$\{in\_car, high\_way\}$	81	72	**127**	112	103
$\{V2V\}$	77	**102**	97	67	54

Table 5.2: Context/Criteria matrix

We consider the Context/Criteria matrix example presented in Table 5.2. Once the security administrator selects and validates countermeasures, the Context/Criteria matrix is updated by evaluations corresponding to each criterion. Values in bold present the security administrator most preferred criterion according to the different combination of active contexts. The recommending module generates the criteria order and coefficients using Definition 32 in Section 5.3.2. Table 5.3 presents the criteria order and coefficients according to three contexts configurations ($\{in\_car\}$,$\{in\_car,high\_way\}$,$\{V2V\}$). This table is provided by the Context/Criteria matrix. The criteria coefficients reflect the importance to be attached to each criterion at the recommendation phase.

Once the system generates a preferred extension corresponding to a specific attack scenario, the system refers to the criteria order and coefficients tables to calculate the score of each proposed countermeasure. For instance, the preferred extension generated in in_car context and corresponding to S, contains two countermeasures:

(a) $\{in\_car\}$ context

Order	Criteria	Coeff
1	Confidentiality	0.284
2	Performance	0.268
3	Availability	0.227
4	Integrity	0.165
5	Precaution	0.057

(b) $\{in\_car, high\_way\}$ context

Order	Criteria	Coeff
1	Confidentiality	0.257
2	Performance	0.226
3	Precaution	0.208
4	Integrity	0.164
5	Availability	0.145

(c) $\{V2V\}$ context

Order	Criteria	Coeff
1	Availability	0.257
2	Confidentiality	0.244
3	Integrity	0.194
4	Performance	0.169
5	Precaution	0.136

Table 5.3: Criteria order and coefficients provided by the Context/Criteria matrix depending on the active contexts

- $Reduce\_frequency$
- $Add\_ source\_ auth$

(a) $Reduce\_frequency$

Criteria	**Value**
Integrity	High (3)
Availability	Very Low (0)
Confidentiality	Medium (2)
Performance	Very Low (0)
Precaution	Very High (4)

(b) Add_source_auth

Criteria	**Value**
Integrity	High (3)
Availability	High (3)
Confidentiality	Medium (2)
Performance	Medium (2)
Precaution	Low (1)

Table 5.4: Examples of countermeasures values per criteria

In the following, we present the recommendation process for both countermeasures. We denote by Rf and Asa respectively, $Reduce\_frequency$ and $Add\_ source\_ auth$ countermeasure. Evaluations done by functional experts corresponding to both countermeasures are presented in Table 5.4.

To determine which countermeasures should be recommended, the system calculates the score of each countermeasure using the Definition 33 in Section 5.3.2. For instance,

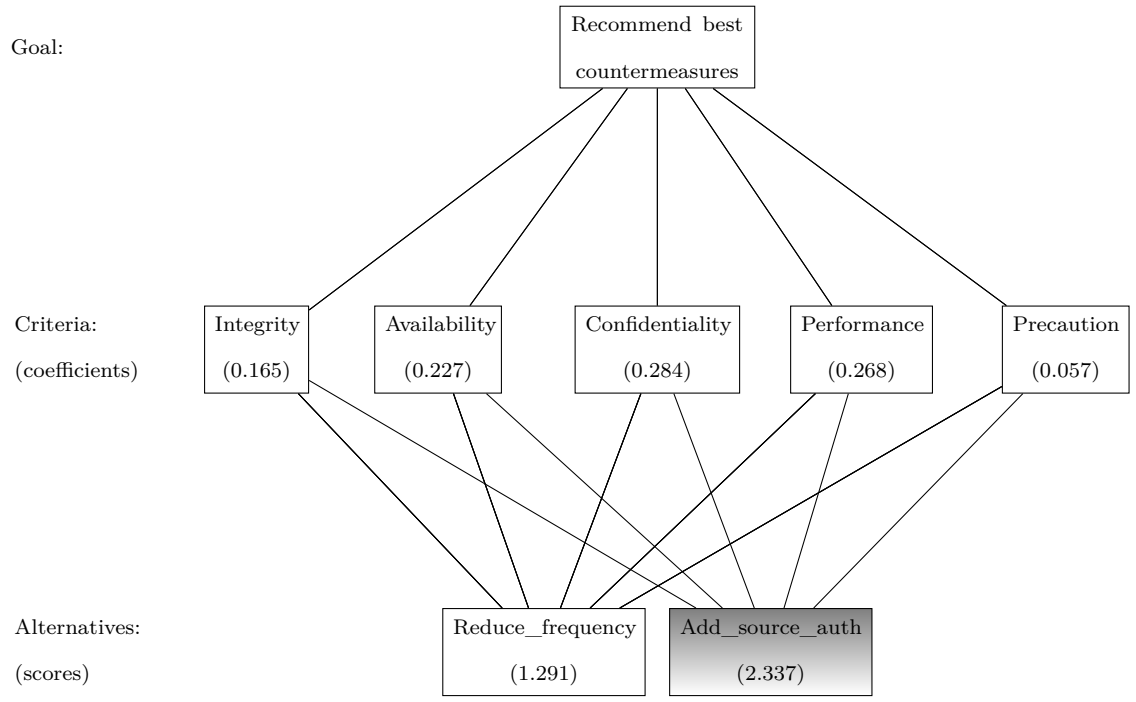


Figure 5.3: Score assignment using SAW method for reduce_frequency and add_source_authentification coutnermeasures in $\{in\_car\}$ context

the score attribution of $Reduce\_frequency$ and Add_source_auth is calculated as follows:

$$Score(Rf, in\_car) = criteria(Rf, Integrity) \times Coeff(Integrity, in\_car)$$
$$+criteria(Rf, Confidentilaity) \times Coeff(Confidentilaity, in\_car)$$
$$+criteria(Rf, Precaution) \times Coeff(Precaution, in\_car)$$

$$Score(Rf, in\_car) = 3 \times 0.165 + 2 \times 0.284 + 4 \times 0.057$$
$$Score(Rf, in\_car) = 1.291$$

The Add_source_auth score in in_car context is calculated with the same formula as $Reduce\_frequency$, we obtain $Score(Asa, in\_car) = 2.337$.
Thus, Add_source_auth countermeasure will be recommended over $Reduce\_frequency$ countermeasure. Figure 5.3 summarizes the score assignment process for both countermeasures. The shadowed node represents the recommended countermeasure. The main goal of our approach being to assist the decider rather than replacing him, the user can select the recommended countermeasures as well as other proposed countermeasures from the preferred extension.

Once the decider selects a countermeasure (i.e., the recommended countermeasure or another proposed one), the system updates the Context/Criteria matrix as presented

in Table 5.5 (when the decider selects *Add_source_auth* countermeasure) and calculates the new criteria coefficients per contexts. This constitutes the learning phase of the recommendation process.

Contexts	Integrity	Availability	Confidentiality	Performance	Precaution
$\{in\_car\}$	67	91	**112**	106	23
$\{in\_car, high\_way\}$	81	72	**127**	112	103
$\{V2V\}$	77	**102**	97	67	54

Table 5.5: Updated Context/Criteria matrix

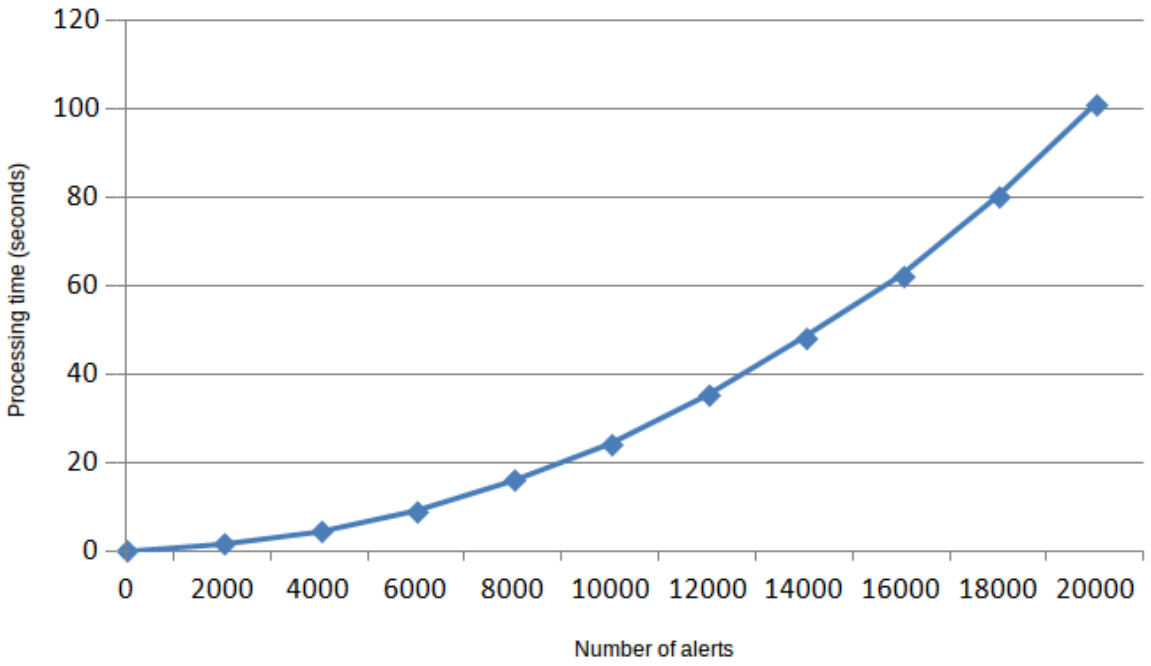


Figure 5.4: System response selection: Execution time per number of detected alerts

5.5.2 Performance evaluation

The Figure 5.4 shows the system response selection costs per number of detected attacks. The measure confirms that a $o(N \times M + Nlog(N)))$ complexity is achieved, where N is the number of generated countermeasures and M is the number of criteria. According to the experimental results, we consider that the proposed approach can

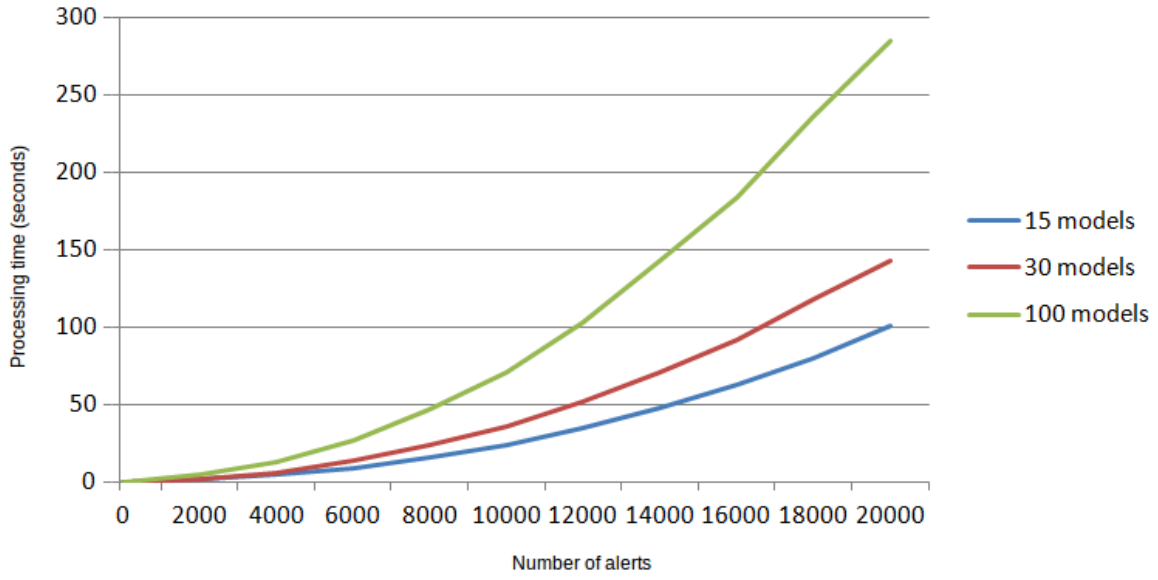


Figure 5.5: System response selection: Execution time per number of models number according to three different model numbers

process 1959 alerts in real-time (i.e., the system can generate preferred extension and automatically select system responses for 1959 alerts in real-time).
We note that all results provided in this chapter consider 15 loaded models (i.e., models include actions, intrusion objectives and countermeasures). Real-time constraints are satisfied with this approach when considering a reasonable number of models. Figure 5.5 shows the execution costs per number of detected alerts for three different model numbers. The measure confirms that a $o(n^3)$ complexity is achieved.

5.6 Conclusion

In order to assist security administrators when selecting countermeasures, it is necessary to have a recommender system that analyzes the security administrator decisions historic to determine his/her different preferences and requirements. We consider the content-based approach the most appropriate to achieve this objective. The content-based approach considers the user decisions historic and analyses them to provide appropriate recommendations. In this work, we proposed an approach based on content-based recommendation for efficient security administrator assistance when selecting the appropriate countermeasures, given a specific attack scenario. This approach considers the set of active contexts in different steps of generation system response as well as in

the recommendation phase. We opted for a SAW method for its low computational costs. Indeed, we showed that the complexity in time of the proposed algorithm is polynomial. Thus, our approach provides recommendations of system responses in real-time for a reasonable number of criteria and countermeasures included in the generated preferred extension. We can notice that the proposed approach does not provide the same results after a finite number of executions. Once a countermeasure is applied and validated by the security administrator, the Context/Criteria matrix is updated by the selected countermeasure evaluations. This update induces a change in criteria coefficients and may as well change the criteria order, which may provide different results in next executions.

In the following chapter, we will focus on the evaluation of the approach efficiency by testing different scenarios and checking if this approach is applicable in the context of real-time system constraints.

CHAPTER 6

Implementation and Evaluation

6.1 Introduction

This chapter presents the implementation of modules presented in chapter 4 and chapter 5. We integrate to CRIM [Autrel and Cuppens 2006] these modules to ensure automatic system responses selection against intrusion detection. We start in this chapter by presenting CRIM tool and its main features. We consider also in this chapter the automotive system as a case of study for our approach. We show how the preferred extension generation module and the MCDM module are integrated into CRIM tool. Furthermore, we present some experimental results to show the efficiency and the performance of our approach.

6.2 CRIM

CRIM is an intrusion detection tool that allows several Intrusion Detection Modules (IDMs) to cooperate in order to provide an accurate alert. CRIM is implemented in C++ using the Qt library [Qt] for graphics.

6.2.1 Features and architecture

We briefly describe in this section the main features provided by CRIM. These features are described as follows:

- Alerts management: this function manages alerts generated by the different IDMs in a relational database.

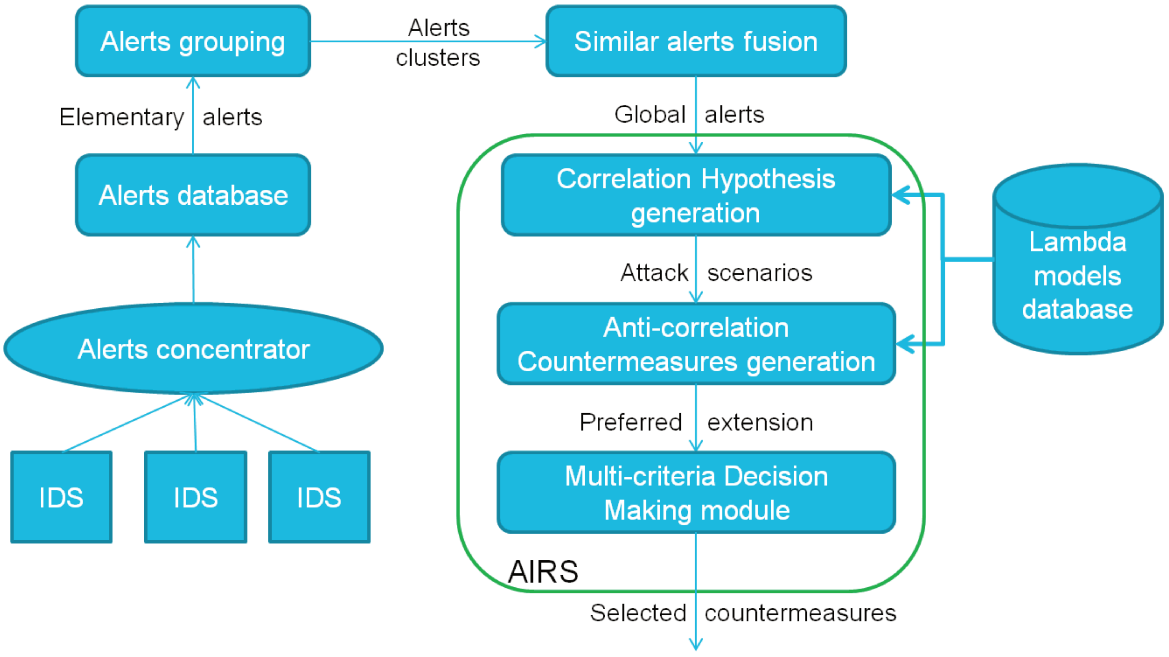


Figure 6.1: CRIM architecture

- Alerts grouping: this function generates packages grouping alerts that match the same attack occurrences.
- Alerts fusion: a global alert is generated for each package identified by the grouping function.
- Alerts correlation: this function permits to identify the different steps of a more global attack scenario (function implemented in Section A.1 in Appendix A).
- Intentions recognition: this function analyses the different attack scenarios generated by the correlation module in order to anticipate the attacker's intentions. It generates a set of attack scenarios according to the detected alerts and it sorts them in order of plausibility.
- System reaction: this function activates the most appropriate countermeasure by considering the results provided by the previous functions (functions implemented in Sections A.2 and A.3 in Appendix A).

Figure 6.1 summarizes the principles that have been proposed to develop CRIM.

6.2.2 Models

We present in this section the implementation of Lambda models (actions, intrusion objectives, countermeasures) as well as alerts models. In addition, we summarize in this section the composition of each model and the role of each model field.

Actions

The action model consists of four XML fields:

- Name: specifies the action name.
- Pre: specifies the state of the system required for the execution of the action.
- Post: specifies the state of the system after the successful execution of the action.
- Detection: describes the expected alert corresponding to the detection of the attack.

Figure 6.2 shows an example of an action model corresponding to *GNSS_spoofing* action model.

```
<action>
<name> GNSS_spoofing </name>
<pre> injection_false_messages(A,T)</pre>
<post> spoofing(T)</post>
<detection> A=//Source/Node/Geographical_localization/text(),
             T=//Target/Node/Geographical_localization/text(),
             classification=R89
</detection>
</action>
```

Figure 6.2: CRIM: Action model example

Intrusion objectives

The intrusion objective model consists of two XML fields:

- Name: specifies the intrusion objective name.
- Condition: specifies the state of the system in which the system security policy is violated.

Figure 6.3 shows an example of an intrusion objective model corresponding to *manipulation_relayed_messages* intrusion objective model.

```
<objective>
<name> manipulation </name>
<condition> manipulate(A,T) </condition>
</objective>
```

Figure 6.3: CRIM: Intrusion objective model example

Countermeasures

The countermeasure model consists of four XML fields:

- Name: specifies the action name.
- Pre: specifies the state of the system required for the execution of the action.
- Post: specifies the state of the system after the successful execution of the action.
- Rationale: describes the expected alert corresponding to the detection of the attack.

Figure 6.4 shows an example of a countermeasure model corresponding to $disable\_wifi$ countermeasure model. As shown in Table 5.4 in the previous chapter, countermea-

```
<reaction>
<name> Disable_wifi </name>
<pre> network_access(A,T,wifi),is_on(T),is_on(wifi)</pre>
<post> not(is_on(wifi)) </post>
<rationale> rationale(1,1,2,1,4) </rationale>
</reaction>
```

Figure 6.4: CRIM: Countermeasure model example

sures are evaluated by functional experts according to a predefined list of criteria. Rationale predicate consists in evaluations corresponding to the five predefined criteria and where possible evaluations are (Very Low(0), Low(1), Medium(2), High(3), Very High(4)). Evaluations inside rationale predicate are respectively for integrity, availability, confidentiality, performance, precaution. For instance, $disable\_wifi$ countermeasure presented in this example has the following evaluations :

- Integrity: Low (1)
- Availability: Low (1)
- Confidentiality: Medium (2)
- Performance: Low (1)

- Precaution: Very High (4)

Alerts

IDMEF (Intrusion Detection Message Exchange Format) is the only alert format supported by CRIM. This alert format was developed as part of the IETF (Internet Engineering Task Force) workshop. The IDMEF as been designed as a format for reporting information related to the observation of a suspicious event. IDMEF transports data between IDS and an alerts management console for example. It can be used for dialogue between two alerts processing modules. Considering the cooperative intrusions detection context, IDMEF designs an architecture where an IDS can be easily removed or added and where new alert processing modules can be easily introduced as well. The IDMEF model is object oriented so it can be extended through defining new sub classes or creating new aggregation relationships with new classes. Once the model is extended, an application that was able to handle instantiated alerts from the non-extended model will be able to handle extended alerts model without considering new data. The IDMEF model is specified by XML DTD (Document Type Definition). The IDMEF is used to transmit the alerts generated from IDS. It is also used to transmit fusion alerts resulting from the aggregation of multiple alerts. Finally, IDMEF is also used in CRIM to model attack scenario alerts and alerts that require applying countermeasures. Figure 6.5 shows an example of an IDMEF alert modeled in XML file.

6.3 Implementation

6.3.1 Preferred extension generation

We implemented a module in CRIM [Autrel and Cuppens 2006] that integrates the notion of rationales in countermeasures. When detecting a hostile action, this module generates the possible scenarios which can be performed by an attacker until he/she reaches his/her intrusion objective. CRIM calculates and generates the largest set (preferred extension) which contains the largest number of coherent countermeasures able to avoid the generated scenario. The generation process of the preferred extension is based on the recursive Algorithm 2 presented in Chapter 4. We show in this section, how CRIM generates preferred extensions according to intrusion detection by considering that for a given detected alert the attacker might have several intrusion objectives. For instance, we consider the scenario presented in the previous chapters

```
<?xml version="1.0" encoding="UTF-8"?>
   <IDMEF-Message version="1.0"
                        xmlns:idmef="http://iana.org/idmef">
<Alert messageid="01">
       <Analyzer analyzerid="01">
       </Analyzer>
       <CreateTime ntpstamp="0xc56b24dc.0x6b7d9557">2015-12-15T21:02:20Z</CreateTime>
       <Source >
         <Node >
           <Geographical_localization>10.10.0</Geographical_localization >
            <Description_node >
            </Description_node>
         </Node>
       </Source>
       <Target >
         <Node>
           <Geographical_localization>10.11.0</Geographical_localization  >
           <Description_node >
             <residual_energie>0.3</residual_energie>
           </Description_node>
         </Node>
       </Target>


    <Classification origin="specific">
        <name>Intrusion</name>
        <url>none</url>
    </Classification>
     </Alert>
   </IDMEF-Message>
```

Figure 6.5: IDMEF alert example

where the system detects that an attacker cracks the passkey of the network. CRIM explores all the attack models to find the actions that the attacker may perform by correlation to the detected attack to reach some intrusion objectives. CRIM generates the potential attacks as well as potential intrusion objectives as shown in Figure 6.6. Attack scenarios proposed in this chapter are the following:

- **Scenario 1:**
  - Wifi_passkey_crack
  - Message_saturation
  - Manipulation_relayed_messages
- **Scenario 2:**
  - Wifi_passkey_crack
  - Message_saturation
  - DDOS

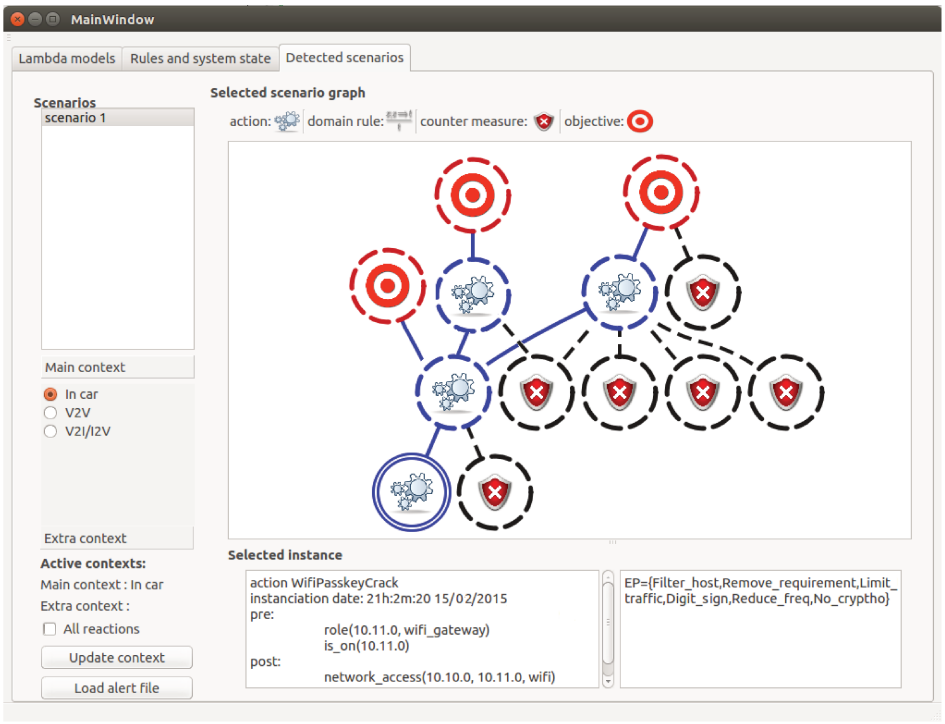


Figure 6.6: CRIM: Preferred extension according to WifiPasskeyCrack attack detection in *in_car* context

- **Scenario 3:**
  - Wifi_passkey_crack
  - False_message_injection
  - GNSS_spoofing

Now that all the potential attacks are generated, CRIM explores the existing countermeasures models that mitigate the generated potential attacks and instantiate them as shown in Figure 6.6. We active the highway context and we run to recompute the updated preferred extension for the same detected alert. As we have seen before, the context change updates the rationale order and coefficients which may change the accepted arguments in the preferred extension as shown in Figure 6.7.

6.3.2 MCDM integration

We integrated in CRIM the MCDM module presented in Chapter 5. CRIM generates automatically countermeasures against the generated attack scenarios from the preferred extension. The countermeasures selection process is based on the proposed

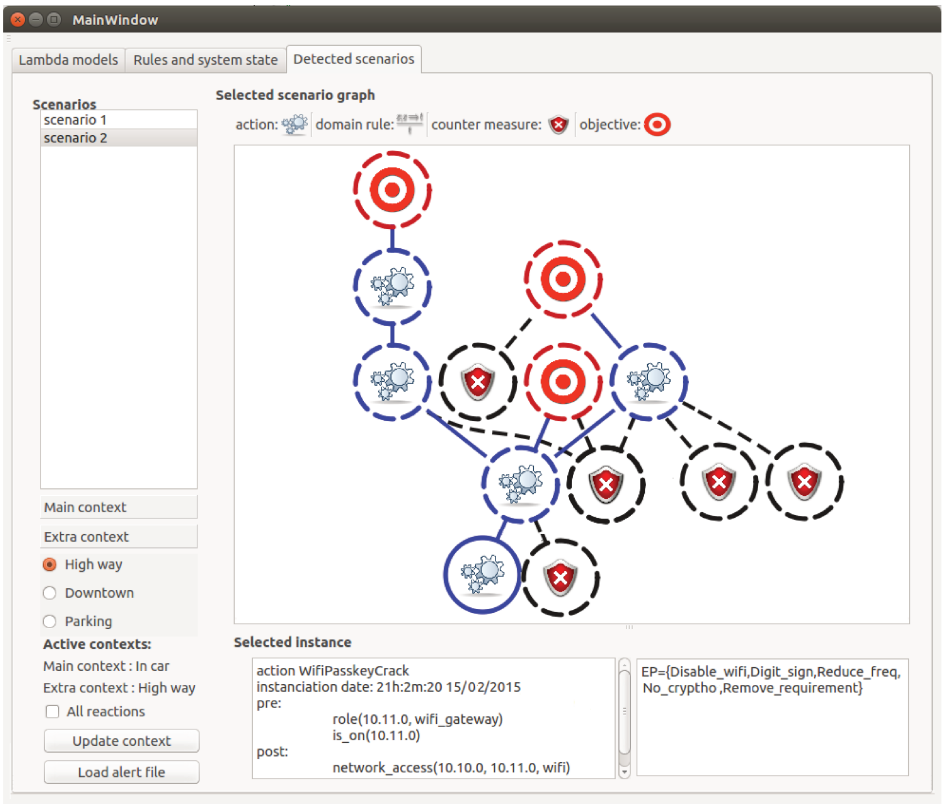


Figure 6.7: CRIM: Preferred extension according to WifiPasskeyCrack attack detection in *in_car* and *highway* context

countermeasures scores according to the current active contexts as shown in the previous chapter. Figure 6.8 shows the system responses against $Wifi\_passkey\_crack$ attack detection in *in_car* and *highway* context. Countermeasures having the best scores from the preferred extension are:

- Disable_wifi
- Remove_requirement
- No_cryptographic_checksum

Once the context in which the automotive system is operating changes, CRIM updates the new coefficients to generates an updated preferred extension then it selects appropriate system responses. For instance, as presented in Figure 6.9 system responses against the same alert detection in $V2I$ context are:

- Filter_host
- Digit_sign

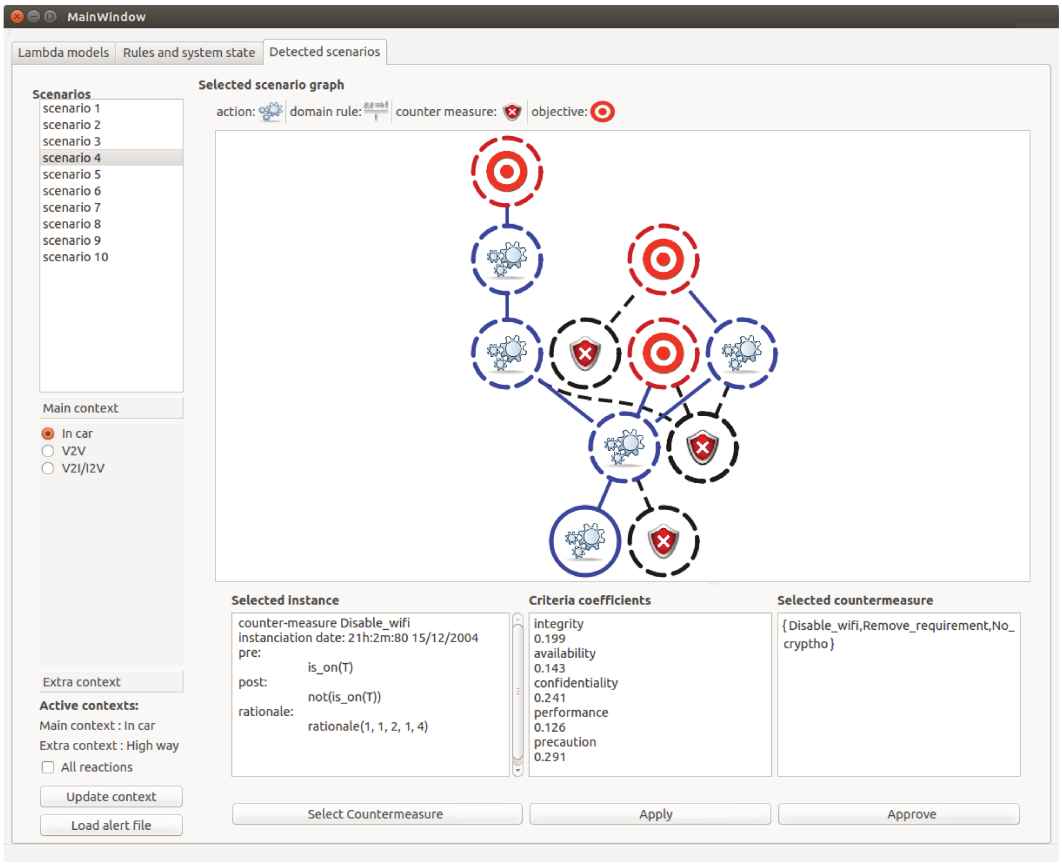


Figure 6.8: System response selection according to WifiPasskeyCrack attack detection in *in_car* and *highway* context

- No_cryptographic_checksum

In less critical contexts and less critical detected alerts, the security administrator can select countermeasures from all existing countermeasures as shown in Figure 6.10. As presented in Figure 6.10 security administrator can select countermeasures that do not belong to the preferred extension by checking "All reactions" check box. In these cases, the security administrator must ensure that there is no conflict between selected countermeasures. Otherwise, conflictual countermeasures can block each other once the selected countermeasures are executed.

6.3.3 Learning file

As explained in the previous chapter, countermeasures scores are calculated based on the Context/Criteria matrix. This matrix provides the different criteria scores and coefficients according to a specific set of active contexts. The Context/Criteria matrix is modeled in our implementation as a text file that saves criteria scores and coefficients.

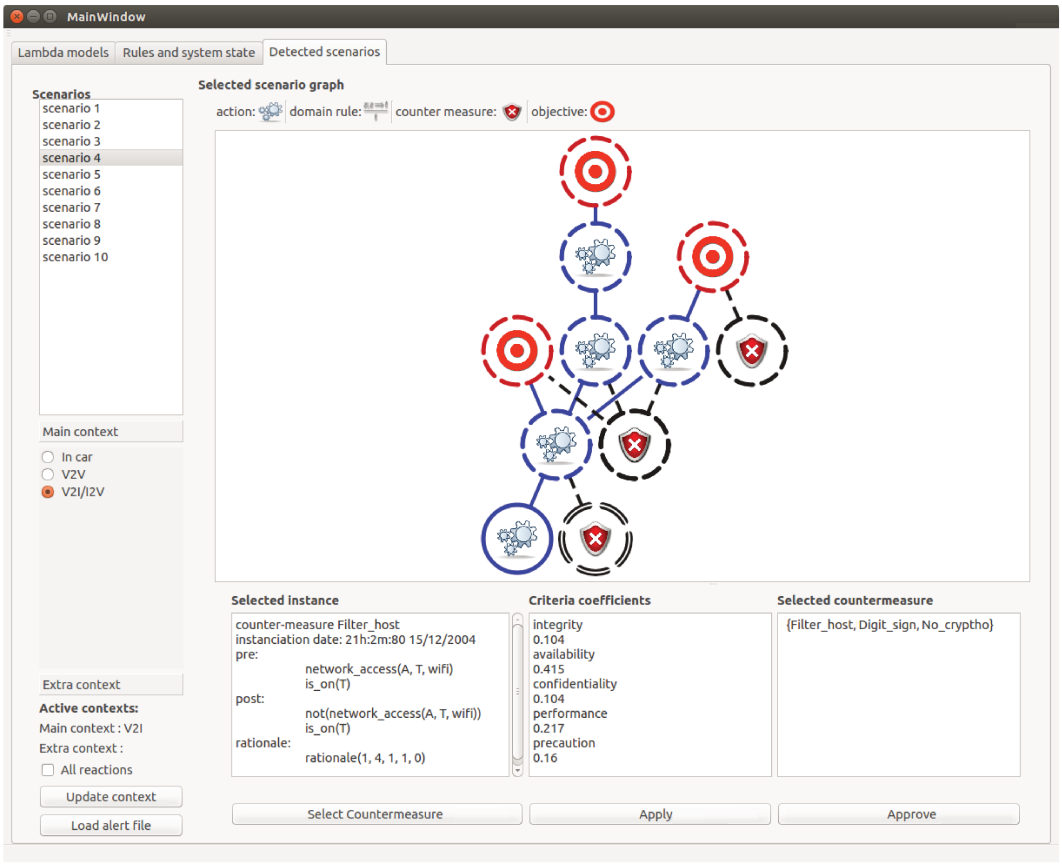


Figure 6.9: System response selection according to WifiPasskeyCrack attack detection in $V2I/I2V$ context

This file is updated automatically once the security administrator validates the selected countermeasures. Figure 6.11 shows an example of the Context/Criteria matrix file corresponding to *in_car* and *high_way* context.

6.4 Evaluation

Alerts considered in Figure 4.7 are alerts corresponding to real attacks. However, security systems are always confronted with false positives. We study the evolution time required for the process of three different total number of alerts (5 000, 10 000, 15 000), and each time we change the percentage of real alerts to test the process costs. To get false positives, we generate IDMEF alert files that do not correspond to existing action models. The Figure 6.12 shows the execution costs per percentage of real alerts from the total detected alerts. We can notice that the execution time when the total number of detected alert contains only false alerts is almost null. This offers an advantage because the time required for processing real alerts will not be adversely

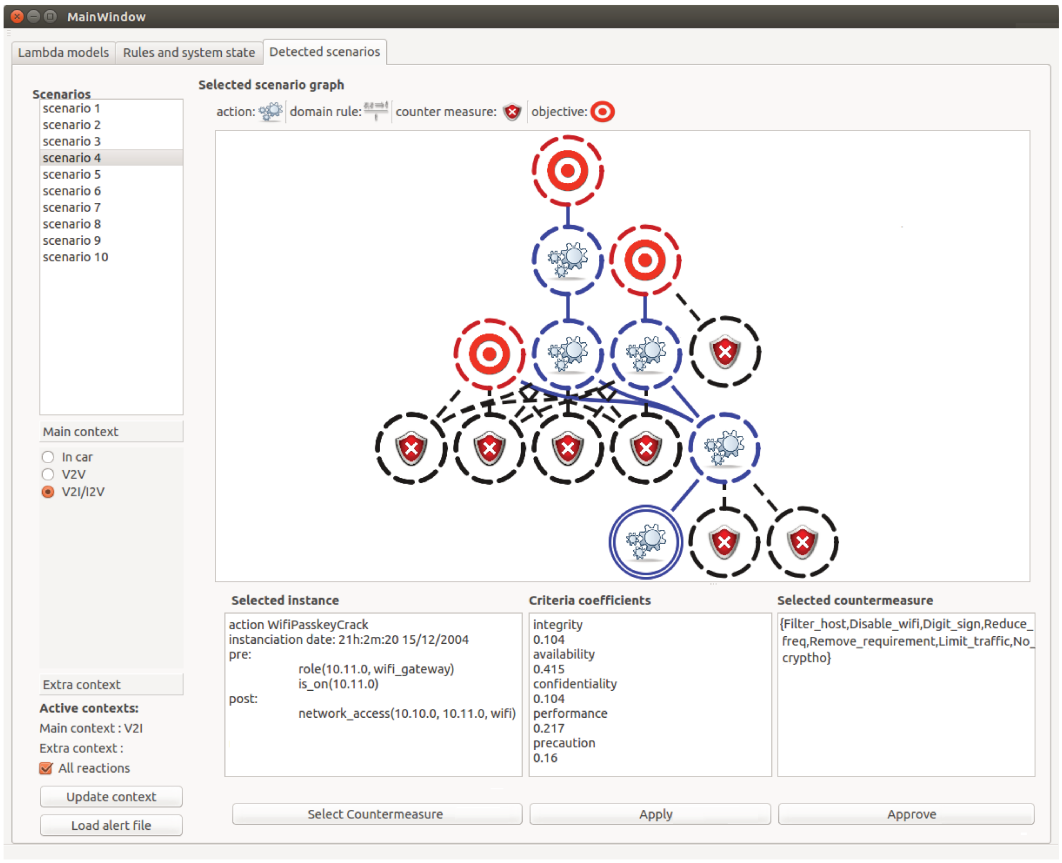


Figure 6.10: All existing countermeasures according to WifiPasskeyCrack attack detection in $V2I/I2V$ context

```
In car - High way
rationale(integrity)
181
rationale(availability)
130
rationale(confidentiality)
265
rationale(performance)
115
rationale(precaution)
220
coefficients
rationale(integrity)
0.199
rationale(availability)
0.143
rationale(confidentiality)
0.291
rationale(performance)
0.126
rationale(precaution)
0.241
```

Figure 6.11: Learning file corresponding to *in_car* and *high_way* context

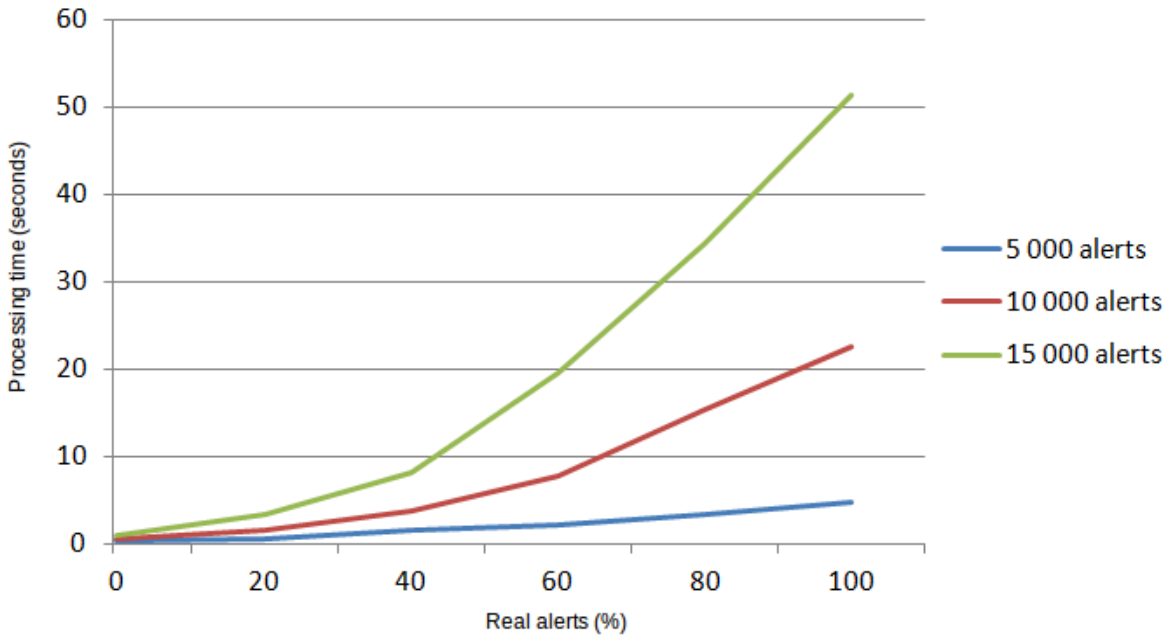


Figure 6.12: Preferred extension generation: Execution time per percentage of real alerts from the detected alerts

affected whatever the number of false alerts detected. The overall AIRS complexity is $o(P^3 + N \times M + Nlog(N)))$ where P is the number of models (i.e., actions, intrusion objectives and countermeasures), N is the number of countermeasures generated in the preferred extension and M is the number of criteria. According to the experimental results, the proposed approach does not satisfy real-time constraints when considering more than 350 models. The proposed approach can process many alerts detected simultaneously in real-time when considering a reasonable number of loaded models.

6.5 Conclusion

The current implementation of the approach successfully responded to real-time constraints, since the responses of critical systems, such as automotive system, must be instantly provided especially in critical contexts. We presented some experimental results concerning the execution costs of our implemented approach. These results allow the evaluation of the approach in terms of performance and time required for the system to response against different attack scenarios detected at the same time. We showed that the proposed approach can process 1959 alerts in real-time. By implementing our approach, we successfully protect critical systems and infrastructures by generating

potential attack scenarios and selecting the countermeasures that satisfy the security administrator preferences and requirements.

CHAPTER 7

Conclusions and Perspectives

In this chapter, we present an overview on how the different thesis objectives outlined in the first chapter have been followed. Afterwards, we show how our approaches can be improved and open up the door to a new research focuses.

7.1 Contributions

Our main objective in this thesis was to provide a tool that keeps the system in a safe state while maintaining the best possible level of performance and quality of service. We present in this section different contributions proposed to meet this objective.

Our first contribution is introduced in Chapter 4 where we proposed an approach [Bouyahia et al. 2014] for an efficient enforcement of security requirements. This approach is based on AL and it describes a collaboration between the system architecture design and security requirements to support the long-term needs of the system. In [Bouyahia et al. 2015], we proposed a more enhanced approach that allows to instantiate actions hypothesis correlated to the detected malicious action. This provides a better system reaction against intrusion by providing the security administrator a better view about the potential attacker's intentions. Moreover, we showed in [Bouyahia et al. 2015] how the system takes into account the set of active contexts when generating system responses. For this purpose, We showed how to improve the existing argumentation framework by defining the Contextual Value-based Argumentation Framework (CVAF) which presents a dynamic framework that consider the current set of active contexts while generating system responses against intrusion.

Second, we presented in Chapter 5 a content-based recommendation approach [Bouyahia et al. 2016] using MCDM method which provides an efficient security administrator assistance when selecting the appropriate countermeasures, given a specific

attack scenario. This approach considers the attack scenario generated from the first approach as well as the corresponding preferred extension in order to assist security administrators to select countermeasures satisfying the prioritized system requirements. For this purpose, we proposed a learning module which allows to learn more about the security administrator way of reacting and have an idea about the security administrator preferences and requirements according to his/her decisions historic. The proposed approach considers the countermeasures impact on the nominal system functional behaviors (e.g., integrity, performance) as criteria to be considered when selecting system responses. Moreover, the learning module permits to automatically select appropriate system responses in critical cases where no security administrator is available to select them by his/her self (e.g., outside working hours).

We applied our approaches on an automotive system as an example of a case study to explore the capability of the proposed approaches to provide system responses for complex system architecture. We did some experimentation concerning the execution costs in time of the implemented approaches. The obtained results show that our approaches implementation successfully respond to real-time constraints, since the responses of critical systems such as automotive system must be instantly provided especially in critical contexts.

7.2 Perspectives

We give in this section some future research directions that could be investigated as a continuation of the results presented in this thesis:

7.2.1 Coordinated attack

The proposed approaches do not consider a possible cooperation between different detected attacks. From the security system point of view, two detected alerts corresponding to different attacker addresses are always considered as two individual attackers. Whereas, several attacker sources may cooperate in order to meet a common intrusion objective. A more enhanced approach can be proposed by considering the coordinated attacks [Samarji et al. 2015] in the response generation process. A Simultaneous Attack Graphs (SAG) can be an input to our Intrusion Response System. A SAG is a graph that contains the most risky attack scenarios hypothesis given a detected alert.

7.2.2 Extending the formal model

We defined five nominal system functional behaviors: Availability, confidentiality, performance, integrity and precaution. Whereas, other security and functional requirements such as traceability and privacy might be requested by security functional experts. Thus, our formal model can be extended to allow security administrator to specify new security and functional requirements. Moreover, the given criteria are limited to high-level security properties, a more enhanced approach can be proposed by considering more detailed and low-level criteria. All attack scenarios generated by the correlation module are treated by the anti correlation module in the same way. However, many parameters can be considered to prioritize some attack scenarios over other ones such as the scenario difficulty level, time required and likelihood. One possible perspective is to consider theses parameters while generating system responses given multiple generated attack scenarios. Finally, the proposed approaches consider only countermeasures impact on the system (e.g., ensuring precaution, ensuring availability). A possible perspective is to consider the countermeasure applying costs in time, i.e., a more enhanced approach can consider countermeasures restore time as an argument to be considered during reasoning process.

7.2.3 The attacker's point of view

We think that it would be interesting as a perspective to try to plan more precisely the attacker future actions by applying the same approach of defense. This would make it possible to build preferred extensions from the attacker's point of view. An attacker chooses the actions in order to reach some goal. A reason representing the negative effects on the security properties of the attacked system can be associated with each action. For example, when an attacker is scanning a target to find its open ports, it is not an attack on itself, but it provides the attacker information to plan his next steps. The reason associated with such an action could be *target discovery* for example. If we consider a denial of service attack implemented through the action consisting in flooding a target with UDP packets, the associated reason could be *availability loss*.

CHAPTER 8

Résumé en français

8.1 Introduction

Le défi des outils de sécurité modernes est de garder le système dans un état de sécurité tout en maintenant le meilleur niveau possible de performance et de qualité de service. Ainsi, nous soutenons que le processus de sécurité doit suivre un raisonnement dynamique et intelligent qui permet au système lors de la détection d'une action malveillante, de prévoir les potentielles actions qui peuvent se produire en corrélation et de générer le meilleur ensemble possible de contre-mesures. Pour atteindre cet objectif, nous proposons les contributions suivantes :

- Nous introduisons une approche [Bouyahia et al. 2014] pour une application efficace des exigences de sécurité, cette approche est supportée par la logique argumentative. Elle décrit une collaboration structurée entre l'architecture du système et des exigences de sécurité pour répondre aux besoins du système durant son fonctionnement. Le but des activités de sécurité assistées par une logique argumentative est de mettre en évidence les critères de décision de contexte pour assurer le maintien du système en condition de sécurité.

- Dans les attaques modernes, l'attaquant peut exécuter plusieurs actions afin de rendre l'exécution d'autres actions possibles jusqu'à atteindre un certain objectif d'intrusion. Pour cela, nous proposons une méthode efficace [Bouyahia et al. 2015] permettant d'instancier les hypothèses d'actions corrélées à l'action malveillante détectée. Cela permet à l'administrateur de sécurité de se rendre compte des intentions potentielles de l'attaquant, ce qui offre une meilleure réaction du système contre les intrusions.

- Compte tenu d'une attaque contre un système donné, la meilleure contre-mesure à appliquer dépend du contexte dans lequel le système est entrain d'opérer. Par exemple, dans le cas des systèmes automobiles, le fait que le véhicule circule au centre ville ou sur une autoroute change l'impact qu'une attaque peut avoir sur

le système. Ainsi, le système doit prendre en compte l'ensemble des contextes actifs lors de la génération des contre-mesures contre une attaque détectée. Pour cela, nous montrons comment adapter les systèmes d'argumentation existantes en définissant le système d'argumentation contextuelle à base des valeurs (CVAF) [Bouyahia et al. 2015].

- Nous proposons une approche de recommandation basée sur le contenu [Bouyahia et al. 2016] en utilisant les méthodes de prise de décision multicritères (MCDM) pour une meilleure assistance de l'administrateur de sécurité lors de la sélection des contre-mesures appropriées, compte tenu d'un scénario d'attaque spécifique. Pour en savoir plus sur la façon dont l'administrateur de sécurité réagit, nous proposons un module d'apprentissage qui permet de former une idée sur les préférences de l'administrateur de sécurité et ses exigences en fonction de son historique de décisions. Les préférences dans notre approche sont considérées comme étant les critères ayant le score le plus élevé selon une combinaison spécifique de contextes actifs. Cette approche considère les différents impacts qu'une contre-mesure pourrait avoir sur le système (par exemple, la performance, la disponibilité) comme étant des critères à prendre en considération lors de la sélection des réponses appropriées du système. Cette approche permet également de sélectionner automatiquement des contre-mesures appropriées dans les cas critiques où l'administrateur de sécurité du système n'est pas en mesure de prendre des décisions.

- Nous appliquons nos approches sur les systèmes automobiles comme exemple de cas d'étude pour explorer les difficultés que peuvent rencontrer les systèmes complexes au cours du processus de la réaction. Ce cas d'étude illustre le besoin potentiel d'une application dynamique des exigences de sécurité pour contrôler les diverses activités de sécurité. Nous présentons des résultats expérimentaux concernant les coûts d'exécution de nos approches implémentées. Ces résultats permettent l'évaluation de l'approche en termes de performance et de temps de réaction requis du système contre les différents scénarios d'attaque détectés simultanément. En se basant sur ces résultats, nous montrons comment notre implémentation satisfait les contraintes de temps réel, car les réponses des systèmes critiques, tels que le système automobile, doivent être immédiatement fournies en particulier dans les contextes critiques.

8.2 Génération des scénarios d'attaques et des contre-mesures sensibles au contexte

Le but principale de cette approche est de générer un ensemble cohérent de contre-mesures pour chaque alerte détectée. L'ensemble généré doit satisfaire les exigences de l'administrateur de sécurité et doit aussi tenir en compte des contextes actifs lors de la détection de l'alerte. Nous estimons dans cette approche que le système de sécurité doit être toujours en avance de phase par rapport à l'attaquant. Pour ce faire, nous générons les scénarios potentiels d'attaque qui correspondent à une alerte détectée.

8.2.1 Génération des scénarios d'attaques

Dans notre approche, nous considérons que plusieurs sondes sont distribuées dans le système contrôlé pour générer des événements qui correspond à des actions exécutées par les agents agissants sur le système. Ces agents peuvent être des utilisateurs légitimes ainsi que des agents malveillants. Les sondes peuvent être des utilisateurs de détection d'intrusion ou les fichiers logs des programmes de surveillance. A chaque fois que les sondes de détections détectent une alerte, le module de corrélation génère et instancie des hypothèses d'actions corrélées à l'action qui correspond à l'alerte détectée. Ce module explore tous les objectifs d'intrusion possibles que l'attaquant peut y parvenir. On a opté pour la modélisation des modèles de sécurité (actions, objectifs d'intrusion et contre-mesures) via le langage LAMBDA [Cuppens et al. 2006]. Nous définissions une action LAMBDA comme suit:

LAMBDA action
name: le nom de l'action
pre-condition: définie l'ensemble de conditions devant être satisfaites sur le système visé par l'action pour qu'elle puisse être réalisée
post-condition: définie Les effets de l'exécution de l'attaque sur le système visé.
detection: définie l'alerte associée à la détection de l'action.

Un attaquant peut exécuter plusieurs actions pour modifier l'état du système afin d'atteindre un état où la politique de sécurité est violée. Certaines actions peuvent être exécutées par l'attaquant afin de rendre l'exécution d'autres actions possibles. Lorsque les effets (post-condition) d'une action sont un sous-ensemble des pré-conditions d'une autre action, on considère que les deux actions sont corrélées.
Un scénario d'attaque est définie dans notre approche comme étant :

- l'action qui correspond à l'alerte détectée,
- les possibles hypothèses d'actions et
- un potentiel objectif d'intrusion

8.2.2 Systèmes d'argumentation à base de valeurs étendue

Bench Capon définit un système d'argumentation à base de valeurs [Bench-Capon 2003] comme suit :

$$VAF = \langle AR, attacks, V, val, Valpref_a \rangle$$

- AR est l'ensemble des arguments et $attacks$ est une relation binaire entre $AR \times AR$.
- V est un ensemble fini de valeurs.
- val est une fonction qui mappe les éléments de AR avec les éléments de V
- $Valpref$ est une relation de préférence entre les valeurs de V.

En raison de la nature dynamique des systèmes d'information, nous soutenons que l'utilisation d'une relation de préférence statique $Valpref$ n'est pas adaptée. Nous étendons la définition de VAF à celle d'un système d'argumentation contextuel $CVAF$ définit comme suit :

$$CVAF = \langle AR, attacks, V, val, \mathcal{C}, ContPref \rangle$$

- AR, $attacks$, V and val ont la même définition que celle d'un $VAFs$
- $\mathcal{C}$ est un ensemble de contextes. A un moment donné plusieurs contextes peuvent être actifs.
- $ContPref$ est une relation de préférence transitive, irreflexive et asymétrique dans $V \times V$ et qui dépend de l'ensemble de contextes actifs dans $\mathcal{C}$

8.2.3 Génération des contre-mesures sensibles au contexte

Les contre-mesures sont des actions qui sont exécutées pour remédier aux effets d'une attaque ou pour bloquer l'exécution d'autres attaques. Plus généralement, ce sont des

actions qui ont un effet négatif sur l'exécution d'autres actions. Plus formellement, nous modélisons une contre-mesure de la même manière qu'une action est modélisée à l'exception que son champ de détection est remplacé par le champ "rationale". Ce champ permet de définir les raisons qui motivent l'exécution de l'action. En fait, nous représentons la raison pour laquelle une contre-mesure doit être choisi. Par exemple, certaines contre-mesures peuvent améliorer la performance du système attaqué au détriment de la disponibilité de certains services. Si pour une raison quelconque la performance du système devrait être favorisée par rapport à la disponibilité des services qu'il fournit, alors nous pouvons choisir la contremesure associée au motif favori. Les contre-mesure ne s'instancient pas à partir d'une alerte, les valeurs sont affectées à ses variables libres en examinant les effets qu'elle doit avoir sur l'état du système, afin d'atténuer les effets d'une attaque ou d'empêcher l'exécution d'autres attaques. La notion d'anti-corrélation formalise cette notion d'effet négatif, Formellement elle est définie comme suit:

Anti-correlation: Soient a et b deux modèles LAMBDA d'actions, il sont anti-corrélées si la condition suivante est vérifiée :
$\exists E_a$ et E_b tel que

- $(E_a \in post(a) \wedge not(E_b) \in pre(b)) or (not(E_a) \in post(a) \wedge E_b \in pre(b))$
- E_a et E_b sont unifiables à travers un unificateur plus général u

Tel que $post(a)$ est l'ensemble des prédicats du champ post-condition de a et $pre(b)$ est l'ensemble des prédicats du champ pré-condition de b.
Compte tenu d'un scénario d'intrusion S, construit comme spécifié dans la Section 8.2.1, et l'ensemble des contre-mesures C calculées pour S, nous construisons l'ensemble des arguments $\mathcal{AR}(S)$ utilisé pour la phase de raisonnement comme étant l'union de ces deux ensembles.
Maintenant que nous avons montré comment construire l'ensemble des arguments correspondant à un scénario d'intrusion, nous définissons la relation d'attaque entre ces arguments :

Relation d'attaque: Soit S un scénario d'intrusion et $\mathcal{AR}(S)$ l'ensemble des arguments correspondant. Soit $a_1 \in \mathcal{AR}(S)$, $a_2 \in \mathcal{AR}(S)$ deux arguments. $attacks(a_1, a_2)$ est vrai ssi $anticor(post(a_1), pre(a_2)) \vee anticor(post(a_1), cond(a_2))$

Dans notre approche, nous ne faisons pas explicitement la condition d'activation pour chaque contexte dans C, nous considérons que cet ensemble est extrait d'une

spécification de politique de sécurité contextuelle, comme une politique OrBAC [Cuppens and Cuppens-Boulahia 2008] par exemple. $ContPref$ a la même définition que celle utilisé dans valpref VAF sauf qu'il permet de générer la préférence entre les forces (valeurs) des arguments en fonction de la configuration des contextes actifs.

Compte tenu d'un scénario d'intrusion, du point de vue de l'agent défendant le système attaqué par un autre agent malveillant, le processus de la réaction consiste à choisir parmi les contre-mesures possibles le meilleur sous-ensemble selon ses préférences. Ces préférences étant codées par la relation $ContPref$. Selon notre approche, cela consiste à utiliser la relation d'attaque que nous avons défini pour construire des ensembles admissibles d'arguments (extensions préférées), chaque ensemble représentant un ensemble cohérent de contre-mesures.

8.3 Une approche multicritère pour assister la prise de décision par l'administrateur de sécurité

Pour éviter de générer des contre-mesures conflictuelles, nous avons proposé dans la section précédente une approche qui procède à une évaluation monocritère pour déterminer quelle contre-mesure doit être sélectionnée. Dans cette section, nous proposons une approche plus renforcée qui intègre un module d'aide à la décision (MCDM). Ce module permet une évaluation plus adaptative entre les contre-mesures en comparant leurs scores. Ceci permet une évaluation qui couvre tous les impacts des contre-mesures sur le système, plutôt que de considérer l'effet principal d'une contre-mesure sur le système comme étant le seul critère à prendre en considération. Le module MCDM intervient dans l'approche proposée sur deux niveaux:

- Il met à jour automatiquement l'ordre des critères selon l'ensemble des contextes actifs dans la phase de génération d'extension préférée.
- Il génère les contre-mesures recommandées à partir de l'extension préférée.

Nous distinguons trois parties principales dans le système de recommandation proposée dans cette approche :

- Module d'apprentissage : Nous définissons la partie d'apprentissage comme étant le processus permettant de donner une visibilité sur la façon dont l'administrateur de sécurité réagit et les différents paramètres influençant ses décisions. Cette partie est alimentée par l'historique des décisions de l'administrateur de sécurité

en analysant le score de différents critères. A chaque fois que l'administrateur de sécurité valide une décision, la matrice d'apprentissage est automatiquement mise à jour avec les valeurs de contre-mesures sélectionnées en fonction de chaque critère.

- Module de recommandation : Ce module calcule le coefficient de chaque critère en se basant sur son score par rapport à la somme totale des critères par contexte. Ces coefficients sont utilisés pour calculer le score des différents contre-mesures au moment de la recommandation. Le score de chaque contre-mesure doit être calculée en se basant sur les coefficients dynamiques des critères pour refléter un score compatible avec l'importance qu'accorde l'administrateur de sécurité pour chaque critère en fonction de chaque ensemble de contextes actifs. Nous utilisons dans notre approche la méthode de décision multicritère SAW [Afshari et al. 2010] pour le calcul des scores des contre-mesures. SAW est connu comme étant la méthode MCDM la plus simple et la plus rapide. Cela sera utile lors de la conception d'un système de réaction contre la détection d'intrusion, car ces systèmes doivent satisfaire les contraintes de temps réel en particulier dans des contextes critiques.

- L'interface de l'administrateur de sécurité : Le but du système de recommandation proposé est non seulement de remplacer les administrateurs de sécurité et de prendre des décisions à leur place, mais aussi de les assister et leur montrer les points que eux seuls ils ne sont pas en mesure de les voir. L'administrateur de sécurité fournit les informations qui alimentent le module d'apprentissage. Chaque fois que l'administrateur de sécurité sélectionne certaines contre-mesures, il/elle est invité(e) à valider sa décision. La phase de validation permet au module d'apprentissage de considérer que les décisions qui satisfont l'administrateur de sécurité, le module d'apprentissage ne considère pas les choix regrettables des contre-mesures. Une fois que l'administrateur valide ses décisions, le module d'apprentissage met à jour la matrice Contexte/Critères pour qu'elle prend en compte les nouvelles décisions.

8.4 Application sur les systèmes automobiles

Afin de donner un exemple pratique sur le besoin potentiel d'une application dynamique des exigences de sécurité pour contrôler les différents activités de sécurité, nous considérons l'exemple du système automobile. Un système moderne d'automobile à bord relie une centaine de microcontrôleurs, appelés unités de commande électronique (ECU) organisés en domaines spécifiques de l'architecture et reliés par des passerelles, comme

le montre la Figure 8.1. Les attaques ont été montré très fréquentes dans les systèmes

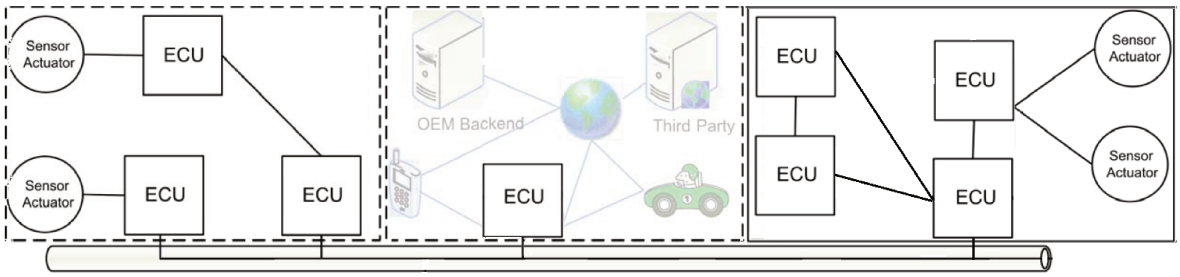


Figure 8.1: Architecture d'un système automobile

automobiles [Koscher et al. 2010] en contournant le filtrage effectué entre les domaines ou par des attaques de type brute-force sur les mécanismes de protection des ECU à base de cryptographie. Ces failles de sécurité peuvent être exploitées pour affecter les différents composants du système automobile (par exemple, verrouillage/déverrouillage des roues de la voiture en pleine la vitesse, désactiver les freins, arrêter moteur, désactiver les cylindres) [Koscher et al. 2010, Checkoway et al. 2011]. De telles attaques peuvent dans la pratique provenir de la connexion Internet qui est devenue de plus en plus disponibles dans les véhicules ou même à travers une connexion Bluetooth d'un téléphone portable compromis situé à l'intérieur d'une véhicule.

Les systèmes automobiles ne comptent pas uniquement sur la simple application des règles de sécurité, mais impliquent également plusieurs points d'application, en particulier lorsque les plates-formes sous-jacentes et les infrastructures fournissent eux-mêmes des services, comme HSM, ou les couches de middleware. La politique de sécurité à appliquer dans un véhicule est la combinaison d'une politique statique pour le contrôle d'usage des informations d'identification cryptographiques des ECU, et une politique dynamique de sécurité de réseau.

La politique du contrôle d'usage des informations d'identification cryptographiques est appliquée par le HSM. En revanche, la politique de sécurité du réseau est appliquée par tous les éléments du réseau. En outre, l'architecture de contrôle d'accès doit également permettre l'application des règles qui limitent le trafic sur le réseau, en se basant sur une authentification de confiance ou d'autres mécanismes de sécurité comme le filtrage du trafic. Cependant, comme a été souligné dans la section précédente, l'application de ces différents mécanismes de sécurité dépend d'un événement ou d'une situation spécifique. Par exemple, lors de la communication avec des entités externes comme véhicule à infrastructure (V2I), il est préférable d'appliquer les règles de filtrage du trafic pour limiter la charge de calcul sur le HSM, qui est responsable de la vérification des opérations cryptographiques. L'application de ce genre de règles

permet d'améliorer la performance du système automobile. Cependant, toujours appliquer ce genre de règles n'est pas souhaitable, car l'application des règles doit dépendre du contexte dans le quel le système est entrain d'opérer.

8.5 Implémentation et évaluation

Cette section présente l'implémentation des modules présentés dans les sections précédentes. Nous intégrons à CRIM[Autrel and Cuppens 2006] ces modules pour assurer la sélection automatique des contre-mesures du système contre la détection d'intrusion.

8.5.1 CRIM

CRIM est un outil de détection d'intrusion qui permet plusieurs modules de détection d'intrusion (MDIs) de coopérer afin de fournir une alerte précise. Les IDMs en CRIM sont basés sur une approche hybride qui combine l'approche comportementale [Anderson 1980] et l'approche fondée sur les scénarios d'attaque. Spécifier plusieurs MDIs en collaboration permet ainsi de réduire la génération des fausses alertes positives et négatives. L'architecture de CRIM adaptée à notre approche est représentée par la Figure 8.2. Nous avons implémenté un module qui intègre la notion de "raison" dans la modélisation des contre-mesures. Ce module permet, lors de la détection d'une action hostile de générer les scénarios d'actions possibles que l'attaquant peut enchaîner jusqu'à ce qu'il atteint un certain objectif d'intrusion. Ce simulateur permet de calculer et de générer l'extension préférée qui contient le plus grand nombre de contre-mesures cohérentes capables de remédier au scénario généré.

8.5.2 Evaluation

Génération des extensions préférées

Selon les résultats expérimentaux, nous constatons que le système peut charger et traiter 2138 alertes en une seconde. Nous estimons qu'une seconde est un intervalle de temps raisonnable pour le traitement des alertes car le module de réaction est considéré en avance de phase par rapport à l'attaquant vu qu'il anticipe les intentions de l'attaquant. Ainsi, nous considérons que le module de réaction que nous avons implémenté peut traiter 2138 des alertes en temps réel.

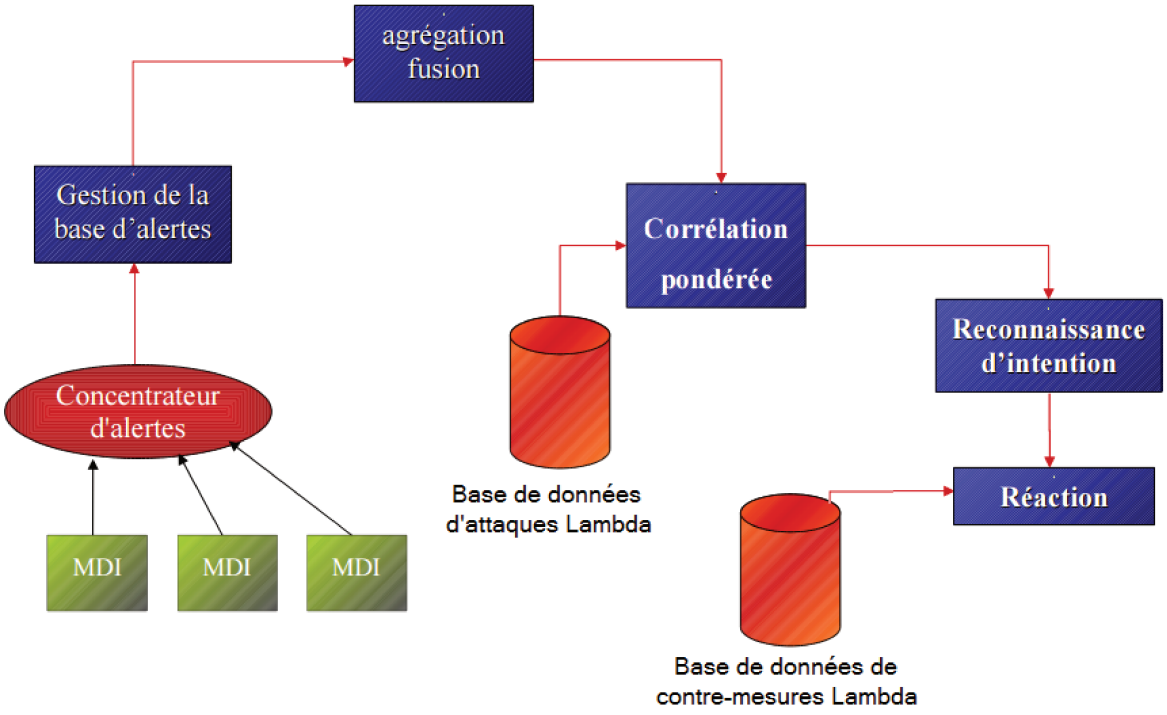


Figure 8.2: Architecture du CRIM

Intégration du module MCDM

Nous avons intégré dans CRIM le module MCDM présenté dans la Section 8.3. CRIM génère automatiquement des contre-mesures contre les scénarios d'attaque générés à partir de l'extension préférée. Le processus de sélection des contre-mesures est basé sur les scores des contre-mesures proposées en fonction des contextes actifs. Selon les résultats de l'expérimentation, nous considérons que l'approche proposée peut traiter 1959 alertes en temps réel (le système génère l'extension préférée et sélectionne automatiquement les réponses du système pour 1959 alertes en temps réel).

8.6 Conclusion

Afin de protéger un système contre les attaques modernes, il est nécessaire de disposer d'une application dynamique et intelligente des politiques de sécurité. Nous considérons que la logique argumentative est l'outil le plus appropriée pour atteindre cet objectif. Nous avons proposé une approche basée sur la logique argumentative et la modélisation via le langage Lambda. Cette approche assure la réaction automatique pour les systèmes de sécurité contre les intrusions détectées. L'approche proposée tient

compte des différents contextes actifs durant lesquels le système peut opérer. Dans ce travail, nous avons montré comment améliorer les systèmes d'argumentation existants. Nous avons proposé aussi une nouvelle approche qui assiste l'administrateur de sécurité pour sélectionner des contre-mesures appropriées et dynamiques qui maintiennent le système dans des conditions de sécurité, tout en satisfaisant les exigences de l'administrateur de sécurité.

Les travaux de recherches menées dans cette thèse peuvent être étendues dans plusieurs directions. Notre modèle formel peut être étendu pour permettre à l'administrateur de sécurité de spécifier des nouvelles exigences de sécurité (par exemple, la traçabilité). L'approche proposée dans cette thèse ne considère pas une éventuelle coopération entre les différents attaques détectées. Du point de vue du système de sécurité, deux alertes détectées correspondantes à deux adresses d'attaquants différentes sont considérées dans notre approche comme deux attaques individuelles. Alors que plusieurs attaquants peuvent coopérer afin d'atteindre un objectif d'intrusion commun. Une approche plus renforcée peut être proposée en considérant les attaques coordonnées [Samarji et al. 2015] dans le processus de génération des contre-mesures.

APPENDIX A

CRIM modules Source Code

A.1 Generation of attack scenarios

```
#include "crim_core.h"
#include "mainwindow.h"
#include <sstream>

//
...
//

void CCrimCore::ComputeCorrelationRules()
{
        messageOutputInterface->OutputMessage("Computing correlation rules...");
        // Clear rules
        ClearCorrelationRules();

        // first compute direct correlation rules between actions
        // we iterate through the loaded action models
        vector < CActionModel * >::iterator first, second, end;
        first = actionModels.begin();
        end = actionModels.end();

        while ( first != end )
        {
                second = actionModels.begin();
                modelA = *first;

                while ( second != end )
                {
                        modelB = *second;

                        // avoid checking for correlation between
                        // a model and itself
                        if ( (*first)->GetName() == (*second)->GetName() )
                        {
                                second++;
                                continue;
                        }

                        // get the pre-condition and post-condition
```

```
                        vector < CFirstOrderPredicate * > post, pre;
40                      vector < CFirstOrderPredicate * >::iterator ca, cb, ea, eb;
                        post = (*first)->GetPostCondition();
                        pre = (*second)->GetPreCondition();
43                      ea = post.end();
                        eb = pre.end();

46                      for (ca = post.begin(); ca!=ea; ca++)
                        {
                                for (cb = pre.begin(); cb!=eb; cb++)
49                      CheckExpressionsForCorrelation(*ca, *cb);
                        }
                        second++;
52              }
                first++;
        }
55
        // compute direct correlation rules between actions and objectives
        // we iterate through the loaded action models
58      vector < CIntrusionObjectiveModel * >::iterator currObj, endObj;
        endObj = objectiveModels.end();
        first = actionModels.begin();
61
        while ( first != end )
        {
64              currObj = objectiveModels.begin();
                modelA = *first;

67              while ( currObj != endObj )
                {
                        modelB = *currObj;
70
                        // get the pre-condition and system state condition
                        vector < CFirstOrderPredicate * > post, cond;
73                      vector < CFirstOrderPredicate * >::iterator ca, cb, ea, eb;
                        post = (*first)->GetPostCondition();
                        cond = (*currObj)->GetPreCondition();
76                      ea = post.end();
                        eb = cond.end();

79                      for (ca = post.begin(); ca!=ea; ca++)
                        {
                                for (cb = cond.begin(); cb!=eb; cb++)
82                      CheckExpressionsForCorrelation(*ca, *cb);
                        }
                        currObj++;
85              }
                first++;
        }
88
        // compute direct correlation rules between actions and domain rules
        // and between domain rules and actions
91      // we iterate through the loaded action models and domain rules
        vector < CDomainRuleModel * >::iterator currRule, endRule;
        endRule = domainRuleModels.end();
94
        first = actionModels.begin();
        while ( first != end )
```

```
97          {
                currRule = domainRuleModels.begin();

100             while ( currRule != endRule )
                {
                        vector < CFirstOrderPredicate * > post, pre;
103                     vector < CFirstOrderPredicate * >::iterator ca, cb, ea, eb;

                        // correlation check between action and rule
106                     modelA = *first;
                        modelB = *currRule;
                        // get the pre-condition and post-condition
109                     post = (*first)->GetPostCondition();
                        pre = (*currRule)->GetPreCondition();
                        ea = post.end();
112                     eb = pre.end();

                        for (ca = post.begin(); ca!=ea; ca++)
115                     {
                                for (cb = pre.begin(); cb!=eb; cb++)
                        CheckExpressionsForCorrelation(*ca, *cb);
118                     }

                        // correlation check between rule and action
121                     modelA = *currRule;
                        modelB = *first;
                        // get the pre-condition and post-condition
124                     post = (*currRule)->GetPostCondition();
                        pre = (*first)->GetPreCondition();
                        ea = post.end();
127                     eb = pre.end();

                        for (ca = post.begin(); ca!=ea; ca++)
130                     {
                                for (cb = pre.begin(); cb!=eb; cb++)
                        CheckExpressionsForCorrelation(*ca, *cb);
133                     }
                        currRule++;
                }
136             first++;
        }

139     // compute direct correlation rules between domain rules and objectives
        // we iterate through the loaded domain rules and objectives
        currRule = domainRuleModels.begin();
142
        while ( currRule != endRule )
        {
145             currObj = objectiveModels.begin();
                modelA = *currRule;

148             while ( currObj != endObj )
                {
                        modelB = *currObj;
151
                        // get the post-condition and system state condition
                        vector < CFirstOrderPredicate * > post, cond;
154                     vector < CFirstOrderPredicate * >::iterator ca, cb, ea, eb;
```

```
                        post = (*currRule)->GetPostCondition();
                        cond = (*currObj)->GetPreCondition();
157                     ea = post.end();
                        eb = cond.end();

160                     for (ca = post.begin(); ca!=ea; ca++)
                        {
                                for (cb = cond.begin(); cb!=eb; cb++)
163                     CheckExpressionsForCorrelation(*ca, *cb);
                        }
                        currObj++;
166             }
                currRule++;
        }
169
        // compute direct correlation rules between domain rules
        // we iterate through the loaded domain rules
172     vector < CDomainRuleModel * >::iterator currRuleRight;
        currRule = domainRuleModels.begin();

175     while ( currRule != endRule )
        {
                currRuleRight = domainRuleModels.begin();
178             modelA = *currRule;

                while ( currRuleRight != endRule )
181             {
                        modelB = *currRuleRight;

184                     // get the pre-condition and post-condition
                        vector < CFirstOrderPredicate * > post, pre;
                        vector < CFirstOrderPredicate * >::iterator ca, cb, ea, eb;
187                     post = (*currRule)->GetPostCondition();
                        pre = (*currRuleRight)->GetPreCondition();
                        ea = post.end();
190                     eb = pre.end();

                        for (ca = post.begin(); ca!=ea; ca++)
193                     {
                                for (cb = pre.begin(); cb!=eb; cb++)
                        CheckExpressionsForCorrelation(*ca, *cb);
196                     }
                        currRuleRight++;
                }
199             currRule++;
        }
```

A.2 Anti-correlation between countermeasures and other models

```
1   #include "crim_core.h"
    #include "mainwindow.h"
    #include <sstream>
```

```
4
        // now compute anti-correlation rules between reaction models and other models
7       vector < CReactionModel * >::iterator currReaction, endReaction;
        endReaction = reactionModels.end();

10      // anti-correlation rules between reactions
        currReaction = reactionModels.begin();

13      while ( currReaction != endReaction )
        {
            vector < CReactionModel * > :: iterator currReaction2 ;
16
            currReaction2 = reactionModels.begin();
            modelA = *currReaction;
19
            while ( currReaction2 != endReaction )
            {
22              modelB = *currReaction2;

                // get the reaction1 post-condition and reaction2 pre-condition
25              vector < CFirstOrderPredicate * > post, pre;
                vector < CFirstOrderPredicate * >::iterator ca, cb, ea, eb;
                post = (*currReaction)->GetPostCondition();
28              pre = (*currReaction2)->GetPreCondition();
                ea = post.end();
                eb = pre.end();
31              if (modelA!=modelB)
                {for (ca = post.begin(); ca!=ea; ca++)
                {
34                  for (cb = pre.begin(); cb!=eb; cb++)
                    CheckExpressionsForAntiCorrelationReaction(*ca, *cb);
                }
37
                }
                currReaction2++;
40          }
            currReaction++;
        }
43
            // anti-correlation rules between reactions and actions
            currReaction = reactionModels.begin();
46          while ( currReaction != endReaction )
            {
                    first = actionModels.begin();
49                  modelA = *currReaction;

                    while ( first != end )
52                  {
                            modelB = *first;
                            // get the reaction post-condition and action pre-condition
55                          vector < CFirstOrderPredicate * > post, pre;
                            vector < CFirstOrderPredicate * >::iterator ca, cb, ea, eb;
                            post = (*currReaction)->GetPostCondition();
58                          pre = (*first)->GetPreCondition();
                            ea = post.end();
                            eb = pre.end();
61
```

```
                        for (ca = post.begin(); ca!=ea; ca++)
                        {
                                for (cb = pre.begin(); cb!=eb; cb++)
                                CheckExpressionsForAntiCorrelation(*ca, *cb);

                        }
                        first++;
                }
                currReaction++;
        }

    // anti-correlation rules between reactions and objectives
        currReaction = reactionModels.begin();
        while ( currReaction != endReaction )
        {
                currObj = objectiveModels.begin();
                modelA = *currReaction;

                while ( currObj != endObj )
                {
                        modelB = *currObj;

                        // get the pre-condition and system state condition
                        vector < CFirstOrderPredicate * > post, cond;
                        vector < CFirstOrderPredicate * >::iterator ca, cb, ea, eb;
                        post = (*currReaction)->GetPostCondition();
                        cond = (*currObj)->GetPreCondition();
                        ea = post.end();
                        eb = cond.end();

                        for (ca = post.begin(); ca!=ea; ca++)
                        {
                                for (cb = cond.begin(); cb!=eb; cb++)
                                CheckExpressionsForAntiCorrelation(*ca, *cb);
                        }
                        currObj++;
                }
                currReaction++;
        }

        // anti-correlation rules between reactions and rules
        currReaction = reactionModels.begin();
        while ( currReaction != endReaction )
        {
                currRule = domainRuleModels.begin();
                modelA = *currReaction;

                while ( currRule != endRule )
                {
                        modelB = *currRule;

                        // get the reaction post-condition and rule pre-condition
                        vector < CFirstOrderPredicate * > post, pre;
                        vector < CFirstOrderPredicate * >::iterator ca, cb, ea, eb;
                        post = (*currReaction)->GetPostCondition();
                        pre = (*currRule)->GetPreCondition();
                        ea = post.end();
                        eb = pre.end();
```

```
121                         for (ca = post.begin(); ca!=ea; ca++)
                            {
                                    for (cb = pre.begin(); cb!=eb; cb++)
124                                 CheckExpressionsForAntiCorrelation(*ca, *cb);
                            }
                            currRule++;
127                 }
                    currReaction++;
            }
130

        stringstream n1, n2;
133         n1 << CLambdaModel::GetCorrelationRulesNumber();
            n2 << CLambdaModel::GetAntiCorrelationRulesNumber();
            messageOutputInterface->OutputMessage("done.\n");
136         messageOutputInterface->OutputMessage("Total number of correlation rules:"
             + n1.str() + "\n");
            messageOutputInterface->OutputMessage("Total number of anti-correlation rules:"
139          + n2.str() + "\n");
    }

142 void CCrimCore::CheckExpressionsForCorrelation( CFirstOrderLogicExpressionNode * first,
                                                  CFirstOrderLogicExpressionNode * second)
    {
145         // check if two expressions are predicates
            if ( (first->GetType() != CFirstOrderLogicExpressionNode::NODE_PREDICATE) &&
                     (second->GetType() != CFirstOrderLogicExpressionNode::NODE_PREDICATE))
148                 return;

            CFirstOrderPredicate * predicateNode1 = (CFirstOrderPredicate *)first;
151         CFirstOrderPredicate * predicateNode2 = (CFirstOrderPredicate *)second;
            bool knows1,knows2=false;
            string fi,se="";
154         string val1 = first->GetValue();
            string val2 = second->GetValue();
            //cout<<"Dans CheckExpressionsForCorrelation, first="+val1+" et second="
157                 +val2<<endl;

            // skip special modalities
160         if ( val1 == "knows" )
              {knows1=true;
              vector <CFirstOrderLogicExpressionNode*> args = predicateNode1->GetArguments();
163           first = args[1];
              //cout<<"first args[0]="<<args[0]->ToString()<<endl;
              fi=args[0]->ToString();
166           //cout<<"first=knows : "<<(args[1])->ToString()<<endl;

            }
169         if ( val2 == "knows" )
              {knows2=true;
              vector <CFirstOrderLogicExpressionNode*> args = predicateNode2->GetArguments();
172           second = args[1];
              //cout<<"second args[0]="<<args[0]->ToString()<<endl;
              se=args[0]->ToString();
175           //cout<<"second=knows : "<<(args[1])->ToString()<<endl;
            }
            // check for negation, both predicates must be negated or not
```

```
178         if ( (val1 == "not") &&
               (first->GetType() == CFirstOrderLogicExpressionNode::NODE_PREDICATE) &&
               (val2 == "not") &&
181         (second->GetType() == CFirstOrderLogicExpressionNode::NODE_PREDICATE) )
            {//cout<<"NOT case !"<<endl;
                    predicateNode1 = (CFirstOrderPredicate *)first;
184                 predicateNode2 = (CFirstOrderPredicate *)second;
                    vector < CFirstOrderLogicExpressionNode * > args;
                    args = predicateNode1->GetArguments();
187                 first = args.front();
                    args = predicateNode2->GetArguments();
                    second = args.front();
190         }

            // check if two expressions are predicates
193         if ( (first->GetType() != CFirstOrderLogicExpressionNode::NODE_PREDICATE) &&
                     (second->GetType() != CFirstOrderLogicExpressionNode::NODE_PREDICATE))
            return;
196
            predicateNode1 = (CFirstOrderPredicate *)first;
            predicateNode2 = (CFirstOrderPredicate *)second;
199
            val1 = predicateNode1->GetValue();
            val2 = predicateNode2->GetValue();
202
            if ( val1 != val2 ) {
            //cout<<"( val1 != val2 ), val1="<<val1<<" , val2="<<val2<<endl;
205         return;}

            // reset current rule
208         currentRule.Clear();
            currentRule.SetLinkingPredicate(predicateNode1->GetValue());
            currentRule.SetFirst(modelA);
211         currentRule.SetSecond(modelB);
            currentRule.SetModelA(modelA->GetName());
            currentRule.SetModelB(modelB->GetName());
214
            // check if the two expressions have the same
            // number of arguments
217         vector <CFirstOrderLogicExpressionNode*> args1 = predicateNode1->GetArguments();
            vector <CFirstOrderLogicExpressionNode*> args2 = predicateNode2->GetArguments();

220         if ( args1.size() != args2.size() ) return;
            if(knows1 && knows2)
                    {
223                 //cout<<"check unification avec 2 knows !"<<endl;
                    //cout<<"pred1 : knows ("<<fi<<","<<val1<<"(arguments)"<<endl;
                    //cout<<"pred1 : knows ("<<se<<","<<val2<<"(arguments)"<<endl;
226                 CFirstOrderLogicExpressionNode * k1=
                            new CFirstOrderLogicExpressionNode(fi);
                    args1.push_back(k1);
229                 CFirstOrderLogicExpressionNode * k2=
                            new CFirstOrderLogicExpressionNode(se);
                    args2.push_back(k2);
232                 }
            // check the predicates attributes
            // check for unification
235         for (unsigned int i = 0; i < args1.size(); i++)
```

```
        {//cout<<"check unification !"<<endl;
                if ( CheckUnification(args1[i], args2[i]) == false ) return;
        }
        // the two expressions can be unified,
        // add rule to both models
    modelA->AddCorrelationRuleRight(currentRule);
    modelB->AddCorrelationRuleLeft(currentRule);
        cout<<" modeles lies : "<<modelA->GetName()<<" et "<<modelB->GetName()<<endl;
}
```

A.3 Attack relation between countermeasures

```
#include "crim_core.h"
#include "mainwindow.h"
#include <sstream>

//check for anti-correlation relation between reactions
void CCrimCore::CheckExpressionsForAntiCorrelationReaction(CFirstOrderLogicExpressionNode
                                    * first, CFirstOrderLogicExpressionNode * second )
{
    // check if two expressions are predicates
    if ( (first->GetType() != CFirstOrderLogicExpressionNode::NODE_PREDICATE) &&
         (second->GetType() != CFirstOrderLogicExpressionNode::NODE_PREDICATE) )
        return;

    CFirstOrderPredicate * predicateNode1 = (CFirstOrderPredicate *)first;
    CFirstOrderPredicate * predicateNode2 = (CFirstOrderPredicate *)second;

    string val1 = first->GetValue();
    string val2 = second->GetValue();

    // check for negation, one expression must be the negation of the other to
    // be anti-correlated
    if ( ((val1 == "not") && (val2 == "not")) ||
         ((val1 != "not") && (val2 != "not")) ) return;

    // skip special modalities
    if ( val1 == "not" )
    {
       vector <CFirstOrderLogicExpressionNode *> args = predicateNode1->GetArguments();
       first = args.front();
    }
    if ( (val2 == "not") )
    {
       vector <CFirstOrderLogicExpressionNode *> args = predicateNode2->GetArguments();
       second = args.front();
    }
    if ( val1 == "knows" )
    {
       vector <CFirstOrderLogicExpressionNode *> args = predicateNode1->GetArguments();
       first = args[1];
    }
    if ( val2 == "knows" )
    {
```

```
        vector <CFirstOrderLogicExpressionNode *> args = predicateNode2->GetArguments();
        second = args[1];
    }

    // check if two expressions are predicates
    if ( (first->GetType() != CFirstOrderLogicExpressionNode::NODE_PREDICATE) &&
       (second->GetType() != CFirstOrderLogicExpressionNode::NODE_PREDICATE) ) return;

    predicateNode1 = (CFirstOrderPredicate *)first;
    predicateNode2 = (CFirstOrderPredicate *)second;

    val1 = predicateNode1->GetValue();
    val2 = predicateNode2->GetValue();

    if ( val1 != val2 ) return;

    // reset current rule
    currentRule.Clear();
    currentRule.SetLinkingPredicate(first->GetValue());
    currentRule.SetFirst(modelA);
    currentRule.SetSecond(modelB);
    currentRule.SetModelA(modelA->GetName());
    currentRule.SetModelB(modelB->GetName());

    // check if the two expressions have the same
    // number of arguments
    vector <CFirstOrderLogicExpressionNode *> args1 = predicateNode1->GetArguments();
    vector <CFirstOrderLogicExpressionNode *> args2 = predicateNode2->GetArguments();

    if ( args1.size() != args2.size() ) return;

    // check the predicates attributes
    // check for unification
    for (unsigned int i=0; i<args1.size(); i++)
    {   if ( CheckUnification(args1[i],args2[i]) == false ) return;
    }
    // the two expressions can be unified,
    // add rule to both models


    modelA->AddAntiCorrelationRuleRight(currentRule);
    modelB->AddAntiCorrelationRuleLeft(currentRule);

    // check which countermeasure defeats the other one,
    // according to the rationale order

    scoreA=0;scoreB=0;
    for(int i=0; i<5 ;i++)
    { //float score1;
        //float score2;
        string ch1="",ch2="",evaluation1="",evaluation2="",coeffs;
        evaluation1=modelA->GetRationaleString().substr(10,13).c_str();
        evaluation2=modelB->GetRationaleString().substr(10,13).c_str();
        ch1=evaluation1.substr(i*3,1).c_str();
        ch2=evaluation2.substr(i*3,1).c_str();
        coeffs= coeff[i].substr(2,3).c_str();

        coefficient= atoi(coeffs.c_str()) * 0.001;
```

```
102         scoreA=  scoreA + (atoi(ch1.c_str()) * coefficient);
            scoreB=  scoreB + (atoi(ch2.c_str()) * coefficient);

105     }

        if (scoreB<=scoreA) {modelB->GotDefeated();
108     }
        else {modelA->GotDefeated();
        }
111
}
```

APPENDIX B

Glossary

AAIRS. Adaptive Agent based Intrusion Response System.

AAF. Abstract Argumentation Framework.

ADeLe. Attack Description Language.

ADEPTS. Adaptive Intrusion Response using Attack Graphs in an E-Commerce Environment.

AF. Argumentation Framework.

AHP. Analytic Hierarchy Process.

AI. Artificial Intelligence.

AIRS. Automated Intrusion Response System.

AL. Argumentation Logic.

ARM. Attack Response Matrix.

ATiKi. ATiki is a tool for a Web-based system that supports collection and sharing of security-related knowledge.

Availability. The property of ensuring timely and reliable access to and use of information.

BMSL. Behavioral Monitoring Specification Language.

C++. It is a general-purpose programming language. It has imperative, object-oriented and generic programming features, while also providing facilities for low-level memory manipulation.

CAM. Correlated Attack Modeling.

CAML. Correlated Attack Modeling Language.

CITRA. Cooperative Intrusion Traceback and Response Architecture.

Confidentiality. The property that sensitive information is not disclosed to unauthorized individuals, entities or processes. It includes means for protecting personal privacy and proprietary information.

CRIM. Corrélation et Reconnaissance d'Intentions Malveillantes (Correlation and Recognition of Malicious Intentions).

CSM. Cooperating Security Managers.

CVAF. Contextual Value-based Argumentation Framework.

DDoS. Distributed Denial of Service.

DIPS. Distributed Intrusion Prevention System.

DoS. Denial of Service.

DTD. Document Type Definition.

ECU. Electronic Control Unit.

ELECTRE. ELimination Et Choix Traduisant la REalité (ELimination and Choice Expressing REality).

EMERALD. Event Monitoring Enabling Responses to Anomalous Live Disturbances.

FAIR. Flexible Automated Intelligent Responder.

FTP. File Transfer Protocol.

HCPN. Hidden Colored Petri-Net.

HMM. Hidden Markov Model.

HSM. Hardware Security Module.

HTML. Hypertext Markup Language.

HTTP. Hypertext Transfer Protocol.

IDM. Intrusion Detection Module.

IDMEF. Intrusion Detection Message Exchange Format.

IDS. Intrusion Detection System.

IETF. Internet Engineering Task Force.

Integrity. The property that sensitive data has not been modified or deleted in an unauthorized and undetected manner while in storage, during processing or in transit.

IRDM-HTN. Intrusion Response Decision-Making model based on Hierarchical Task Network planning..

IRS. Intrusion Response System.

ITS. Intelligent Transport System.

LAMBDA. LAnguage to Model a dataBase for Detection of Attacks.

LPP. Logic Programming with Priorities.

MCDM. Multi-Criteria Decision Making.

MIRADOR. Mécanismes de détection d'Intrusion et de Réaction aux Attaques en DOmaine militaiRe (Intrusion Detection Mechanisms and Attack Response in military Field).

OrBAC. Organization-Based Access Control.

PAF. Preference-based Argumentation Framework.

P-BEST. Production-Based Expert System Toolset.

Performance. The property of taking countermeasures that ensure the best quality of service.

PH. Process Homeostasis.

Precaution. The property of taking precautionary countermeasures to avoid system damage in critical contexts.

Qt. It is a cross-platform application framework that is widely used for developing application software that can be run on various software and hardware platforms with little or no change in the underlying codebase, while still being a native application with the capabilities and speed thereof.

SARA. Survivable Autonomic Response Architecture.

SAW. Simple Additive Weighting.

SHIM. System Health and Intrusion Monitoring.

SoSMART. System or Security Managers Adaptive Response Tool.

SSH. Secure SHell.

TBAIR. Tracing-Based Active Intrusion Response.

TOPSIS. Technique for Order of Preference by Similarity to Ideal Solution.

V2I. Vehicle to Infrastructure.

V2V. Vehicle to Vehicle.

VAF. Value-based Argumentation Framework.

XML. eXtensible Markup Language.

List of Publications

International Conferences

- T. Bouyahia, M.S. Idrees, N. Cuppens-Boulahia, F.Cuppens and F.Autrel. Metric for Security Activities Assisted by Argumentative Logic. In DPM/SETOP/QASA 2014, Volume 8872 of Lecture Notes in Computer Science., Springer (2014) 183-197, Wroclaw, Poland, September 10-11,2014.
- T. Bouyahia, F. Autrel, N. Cuppens-Boulahia, and F. Cuppens. Context aware intrusion response based on argumentation logic. In C.Lambrinoudakis and A. Gabillon, editors, Risks and Security of Internet and Systems - 10th International Conference, CRiSIS 2015, Mytilene, Lesbos Island, Greece, July 20-22, 2015, Revised Selected Papers, volume 9572 of Lecture Notes in Computer Science, pages 91-106, 2015. Springer.
- T. Bouyahia, N. Cuppens-Boulahia, F. Cuppens, F. Autrel : Multi-criteria recommender approach for supporting intrusion response system. In Foundations and Practice of Security - 9th International Symposium, FPS 2016, Québec City, QC, Canada, October 24-25, 2016, Revised Selected Papers. Volume 10128 of Lecture Notes in Computer Science, Springer (2016) 51-67

Bibliography

[Wik] The simplest online database that could possibly work. `http://wiki.org`. 19

[CAM 2003] Correlated attack modeling (cam) project. 2003. 16

[Adomavicius et al. 2011] G. Adomavicius, N. Manouselis, and Y. Kwon. Multi-criteria recommender systems. In *Recommender Systems Handbook*, pages 769–803. 2011. 80

[Afshari et al. 2010] A. Afshari, M. Mojahed, and R. M. Yusuff. Simple additive weighting approach to personnel selection problem. *International Journal of Innovation, Management and Technology*, 1(5):511, 2010. 83, 123

[Al-Shaer and Hamed 2003] E. S. Al-Shaer and H. H. Hamed. Firewall policy advisor for anomaly discovery and rule editing. In G. S. Goldszmidt and J. Schönwälder, editors, *Integrated Network Management VII, Managing It All, IFIP/IEEE Eighth International Symposium on Integrated Network Management (IM 2003), March 24-28, 2003, Colorado Springs, USA*, volume 246 of *IFIP Conference Proceedings*, pages 17–30, 2003. Kluwer. 40, 161

[Amgoud and Cayrol 1998] L. Amgoud and C. Cayrol. On the acceptability of arguments in preference-based argumentation. In G. F. Cooper and S. Moral, editors, *UAI '98: Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence, University of Wisconsin Business School, Madison, Wisconsin, USA, July 24-26, 1998*, pages 1–7, 1998. Morgan Kaufmann. 31, 35

[Amgoud and Cayrol 2002] L. Amgoud and C. Cayrol. A reasoning model based on the production of acceptable arguments. *Ann. Math. Artif. Intell.*, 34(1-3):197–215, 2002. 31, 35

[Anderson et al. 1994] D. Anderson, T. Frivold, A. Tamaru, A. Valdes, and update B. Release. Next generation intrusion detection expert system (nides), software users manual. 1994. 11

[Anderson 1980] Technical report, James P. Anderson Company, Fort Washington, Pennsylvania. Computer security threat monitoring and surveillance. Technical report, 1980. 125

[Anuar et al. 2010] N. B. Anuar, M. Papadaki, S. Furnell, and N. L. Clarke. An investigation and survey of response options for intrusion response systems (irss). In H. S. Venter, M. Coetzee, and M. Loock, editors, *Information Security South Africa Conference 2010, Sandton Convention Centre, Sandton, South Africa, August 2-4, 2010. Proceedings ISSA 2010*, 2010. ISSA, Pretoria, South Africa. 14

[Applebaum et al. 2012] A. Applebaum, K. N. Levitt, J. Rowe, and S. Parsons. Arguing about firewall policy. In B. Verheij, S. Szeider, and S. Woltran, editors, *Computational Models of Argument - Proceedings of COMMA 2012, Vienna, Austria, September 10-12, 2012*, volume 245 of *Frontiers in Artificial Intelligence and Applications*, pages 91–102, 2012. IOS Press. 39, 41, 42, 165

[Aruldoss et al. 2013] M. Aruldoss, T. M. Lakshmi, and V. P. Venkatesan. A survey on multi criteria decision making methods and its applications. *American Journal of Information Systems*, 1(1):31–43, 2013. 83

[Autrel and Cuppens 2006] F. Autrel and F. Cuppens. Crim: un module de corrélation d'alertes et de réaction aux attaques. *Annales Des Télécommunications*, 61(9-10):1172–1192, 2006. 99, 103, 125

[Autrel et al. 2009] F. Autrel, N. Cuppens-Boulahia, and F. Cuppens. Reaction policy model based on dynamic organizations and threat context. In E. Gudes and J. Vaidya, editors, *DBSec*, volume 5645 of *Lecture Notes in Computer Science*, pages 49–64, 2009. Springer. 69

[Axelsson 2000] Intrusion detection systems: A survey and taxonomy. Technical report, 2000. 52

[Balabanovic and Shoham 1997] M. Balabanovic and Y. Shoham. Content-based, collaborative recommendation. *Commun. ACM*, 40(3):66–72, 1997. 80

[Balepin et al. 2003] I. Balepin, S. Maltsev, J. Rowe, and K. N. Levitt. Using specification-based intrusion detection for automated response. In G. Vigna, E. Jonsson, and C. Krügel, editors, *Recent Advances in Intrusion Detection, 6th International Symposium, RAID 2003, Pittsburgh, PA, USA, September 8-10, 2003, Proceedings*, volume 2820 of *Lecture Notes in Computer Science*, pages 136–154, 2003. Springer. 11, 15

[Bandara et al. 2006] A. K. Bandara, A. C. Kakas, E. C. Lupu, and A. Russo. Using argumentation logic for firewall policy specification and analysis. In R. State, van der S. Meer, D. O'Sullivan, and T. Pfeifer, editors, *Large Scale Management of Distributed Systems, 17th IFIP/IEEE International Workshop on Distributed Systems: Operations and Management, DSOM 2006, Dublin, Ireland, October 23-25, 2006, Proceedings*, volume 4269 of *Lecture Notes in Computer Science*, pages 185–196, 2006. Springer. 39, 40

[Bandara et al. 2009] A. K. Bandara, A. C. Kakas, E. C. Lupu, and A. Russo. Using argumentation logic for firewall configuration management. In *Integrated Network Management, IM 2009. 11th IFIP/IEEE International Symposium on Integrated Network Management, Hofstra University, Long Island, NY, USA, June 1-5, 2009*, pages 180–187, 2009. IEEE. 39, 41

[Bar-El 2009] H. Bar-El. Intra-Vehicle Information Security Framework. In *Proceedings of the 7th escar Conference*, Düsseldorf, Germany, 2009. 1

[Bench-Capon 2003] T. J. M. Bench-Capon. Persuasion in practical argument using value-based argumentation frameworks. *J. Log. Comput.*, 13(3):429–448, 2003. 31, 36, 120

[Benferhat et al. 2003] S. Benferhat, F. Autrel, and F. Cuppens. Enhanced correlation in an intrusion detection process. In V. Gorodetsky, L. Popyack, and V. Skormin, editors, *Computer Network Security*, volume 2776 of *Lecture Notes in Computer Science*, pages 157–170. Springer Berlin Heidelberg, 2003. 54, 55

[Boella et al. 2005] G. Boella, J. Hulstijn, and van der L. W. N. Torre. Argumentation for access control. In *AI*IA*, pages 86–97, 2005. 45

[Bouyahia et al. 2015] T. Bouyahia, F. Autrel, N. Cuppens-Boulahia, and F. Cuppens. Context aware intrusion response based on argumentation logic. In C. Lambrinoudakis and A. Gabillon, editors, *Risks and Security of Internet and Systems - 10th International Conference, CRiSIS 2015, Mytilene, Lesbos Island, Greece, July 20-22, 2015, Revised Selected Papers*, volume 9572 of *Lecture Notes in Computer Science*, pages 91–106, 2015. Springer. 3, 113, 117, 118

[Bouyahia et al. 2016] T. Bouyahia, N. Cuppens-Boulahia, F. Cuppens, and F. Autrel. Multi-criteria recommender approach for supporting intrusion response system. 2016. 3, 113, 118

[Bouyahia et al. 2014] T. Bouyahia, M. S. Idrees, N. Cuppens-Boulahia, F. Cuppens, and F. Autrel. Metric for security activities assisted by argumentative

logic. In *DPM/SETOP/QASA 2014*, volume 8872 of *Lecture Notes in Computer Science*, pages 183–197, 2014. Springer. 3, 113, 117

[Bouyssou and Roy 1993] D. Bouyssou and B. Roy. Aide multicritere a la decision: Methodes et cas. *Economica, Paris*, 1993. 82

[Bowen et al. 2000] T. Bowen, D. Chee, M. Segal, R. Sekar, T. Shanbhag, and P. Uppuluri. Building survivable systems: an integrated approach based on intrusion detection and damage containment. In *DARPA Information Survivability Conference and Exposition, 2000. DISCEX '00. Proceedings*, volume 2, pages 84–99 vol.2, 2000. 15

[Bruschi and Rosti 2001] D. Bruschi and E. Rosti. Angel: a tool to disarm computer systems. In V. Raskin, S. J. Greenwald, B. Timmerman, and D. M. Kienzle, editors, *Proceedings of the New Security Paradigms Workshop 2001, Cloudcroft, New Mexico, USA, September 10-13, 2001*, pages 63–69, 2001. ACM. 9

[Chaim and Lucie 1969] P. Chaim and O.-T. Lucie. *The New Rhetoric*. University of Notre Dame Press, Notre Dame, IN, 1969. 36, 37

[Checkoway et al. 2011] S. Checkoway, D. McCoy, B. Kantor, D. Anderson, H. Shacham, S. Savage, K. Koscher, A. Czeskis, F. Roesner, and T. Kohno. Comprehensive experimental analyses of automotive attack surfaces. In *20th USENIX Security Symposium, San Francisco, CA, USA, August 8-12, 2011, Proceedings*, 2011. USENIX Association. 68, 124

[Chen and Yang 2004] Y. Chen and Y. Yang. Policy management for network-based intrusion detection and prevention. In *Managing Next Generation Convergence Networks and Services, IEEE/IFIP Network Operations and Management Symposium, NOMS 2004, Seoul, Korea, 19-23 April 2004, Proceedings*, pages 219–232, 2004. IEEE. 9

[Cheung et al. 2003] S. Cheung, U. Lindqvist, and M. W. Fong. Modeling multistep cyber attacks for scenario recognition. In *3rd DARPA Information Survivability Conference and Exposition (DISCEX-III 2003), 22-24 April 2003, Washington, DC, USA*, pages 284–292, 2003. IEEE Computer Society. 16

[Chiprianov et al. 2013] V. Chiprianov, P. Meyer, and J. Simonin. Towards a model-based multiple criteria decision aid process. 2013. 80

[Cohen 1999] F. Cohen. Simulating cyber attacks, defences, and consequences. *Computers & Security*, 18(6):479–518, 1999. 8

[Cuppens 2001] F. Cuppens. Managing alerts in a multi-intrusion detection environmen. In *17th Annual Computer Security Applications Conference (ACSAC 2001), 11-14 December 2001, New Orleans, Louisiana, USA*, pages 22–31, 2001. IEEE Computer Society. 19

[Cuppens et al. 2006] F. Cuppens, F. Autrel, Y. Bouzida, J. García, S. Gombault, and T. Sans. Anti-correlation as a criterion to select appropriate countermeasures in an intrusion detection framework. *Annales des Télécommunications*, 61(1-2):197–217, 2006. 52, 56, 119

[Cuppens and Cuppens-Boulahia 2008] F. Cuppens and N. Cuppens-Boulahia. Modeling contextual security policies. *Int. J. Inf. Sec.*, 7(4):285–305, 2008. 59, 122

[Cuppens and Ortalo 2000] F. Cuppens and R. Ortalo. LAMBDA: A language to model a database for detection of attacks. In H. Debar, L. Mé, and S. F. Wu, editors, *Recent Advances in Intrusion Detection, Third International Workshop, RAID 2000, Toulouse, France, October 2-4, 2000, Proceedings*, volume 1907 of *Lecture Notes in Computer Science*, pages 197–216, 2000. Springer. 21, 52

[Debar et al. 2007] H. Debar, D. A. Curry, and B. S. Feinstein. The intrusion detection message exchange format (idmef). 2007. 16

[Dijkstra et al. 2005] P. Dijkstra, F. Bex, H. Prakken, and de K. Vey Mestdagh. Towards a multi-agent system for regulated information exchange in crime investigations. *Artif. Intell. Law*, 13(1):133–151, 2005. 45

[Dimopoulos and Kakas 1995] Y. Dimopoulos and A. C. Kakas. Logic programming without negation as failure. In J. W. Lloyd, editor, *Logic Programming, Proceedings of the 1995 International Symposium, Portland, Oregon, USA, December 4-7, 1995*, pages 369–383, 1995. MIT Press. 40

[Doutre et al. 2007] S. Doutre, P. McBurney, L. Perrussel, and J. Thévenin. Arguing for gaining access to information. In E. H. Durfee, M. Yokoo, M. N. Huhns, and O. Shehory, editors, *6th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2007), Honolulu, Hawaii, USA, May 14-18, 2007*, page 20, 2007. IFAAMAS. 45

[Dung 1995] P. M. Dung. On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming and n-person games. *Artificial Intelligence*, 77(2):321 – 357, 1995. 2, 31

[Fisch 1996] E. A. Fisch. A taxonomy and implementation of automated responses to intrusive behaviour. *Ph.D thesis, Texas A&M University*, 1996. 15

[Foo et al. 2005] B. Foo, Y. Wu, Y. Mao, S. Bagchi, and E. H. Spafford. ADEPTS: adaptive intrusion response using attack graphs in an e-commerce environment. In *2005 International Conference on Dependable Systems and Networks (DSN 2005), 28 June - 1 July 2005, Yokohama, Japan, Proceedings*, pages 508–517, 2005. IEEE Computer Society. 12, 13, 15

[Haslum et al. 2007] K. Haslum, A. Abraham, and S. J. Knapskog. DIPS: A framework for distributed intrusion prediction and prevention using hidden markov models and online fuzzy risk assessment. In N. Zhang and A. Abraham, editors, *Proceedings of the Third International Symposium on Information Assurance and Security, IAS 2007, August 29-31, 2007, Manchester, United Kingdom*, pages 183–190, 2007. IEEE Computer Society. 15

[Hwang et al. 1993] C. Hwang, Y. Lai, and T. Liu. A new approach for multiple objective decision making. *Computers & OR*, 20(8):889–899, 1993. 81

[Jahnke et al. 2007] M. Jahnke, C. Thul, and P. Martini. Graph based metrics for intrusion response measures in computer networks. In *32nd Annual IEEE Conference on Local Computer Networks (LCN 2007), 15-18 October 2007, Clontarf Castle, Dublin, Ireland, Proceedings*, pages 1035–1042, 2007. IEEE Computer Society. 15

[Kakas et al. 1994] A. C. Kakas, P. Mancarella, and P. M. Dung. The acceptability semantics for logic programs. In P. V. Hentenryck, editor, *Logic Programming, Proceedings of the Eleventh International Conference on Logic Programming, Santa Marherita Ligure, Italy, June 13-18, 1994*, pages 504–519, 1994. MIT Press. 40

[Kanoun et al. 2010] W. Kanoun, N. Cuppens-Boulahia, F. Cuppens, and S. Dubus. Risk-aware framework for activating and deactivating policy-based response. In Y. Xiang, P. Samarati, J. Hu, W. Zhou, and A. Sadeghi, editors, *Fourth International Conference on Network and System Security, NSS 2010, Melbourne, Victoria, Australia, September 1-3, 2010*, pages 207–215, 2010. IEEE Computer Society. 15

[Kheir et al. 2010] N. Kheir, N. Cuppens-Boulahia, F. Cuppens, and H. Debar. A service dependency model for cost-sensitive intrusion response. In D. Gritzalis, B. Preneel, and M. Theoharidou, editors, *Computer Security - ESORICS 2010, 15th European Symposium on Research in Computer Security, Athens, Greece, September 20-22, 2010. Proceedings*, volume 6345 of *Lecture Notes in Computer Science*, pages 626–642, 2010. Springer. 15

[Kiriansky et al. 2002] V. Kiriansky, D. Bruening, and S. P. Amarasinghe. Secure execution via program shepherding. In D. Boneh, editor, *Proceedings of the*

11th USENIX Security Symposium, San Francisco, CA, USA, August 5-9, 2002, pages 191–206, 2002. USENIX. 10

[Koscher et al. 2010] K. Koscher, A. Czeskis, F. Roesner, S. Patel, T. Kohno, S. Checkoway, D. McCoy, B. Kantor, D. Anderson, H. Shacham, and S. Savage. Experimental security analysis of a modern automobile. In *Security and Privacy (SP), 2010 IEEE Symposium on*, pages 447 –462, May 2010. 68, 124

[Koshutanski and Massacci 2004a] H. Koshutanski and F. Massacci. An interactive trust management and negotiation scheme. In T. Dimitrakos and F. Martinelli, editors, *Formal Aspects in Security and Trust: Second IFIP TC1 WG1.7 Workshop on Formal Aspects in Security and Trust (FAST), an event of the 18th IFIP World Computer Congress, August 22-27, 2004, Toulouse, France*, volume 173 of *IFIP*, pages 115–128, 2004. Springer. 45

[Koshutanski and Massacci 2004b] H. Koshutanski and F. Massacci. A system for interactive authorization for business processes for web services. In N. Koch, P. Fraternali, and M. Wirsing, editors, *Web Engineering - 4th International Conference, ICWE 2004, Munich, Germany, July 26-30, 2004, Proceedings*, volume 3140 of *Lecture Notes in Computer Science*, pages 521–525, 2004. Springer. 45

[Laboratories 2003] N. A. Laboratories. Secure execution environments/generic soft- ware wrappers for security and reliability. `http://www.networkassociates.com`, 2003. 11

[Lee et al. 2002] W. Lee, W. Fan, M. Miller, S. J. Stolfo, and E. Zadok. Toward cost-sensitive modeling for intrusion detection and response. *Journal of Computer Security*, 10(1/2):5–22, 2002. 11, 13, 15

[Lewandowski et al. 2001] S. Lewandowski, D. Van Hook, G. O'Leary, J. Haines, and L. Rossey. Sara: Survivable autonomic response architecture. In *DARPA Information Survivability Conference amp; Exposition II, 2001. DISCEX '01. Proceedings*, volume 1, pages 77–88 vol.1, 2001. 15

[Lindqvist and Porras 1999] U. Lindqvist and P. A. Porras. Detecting computer and network misuse through the production-based expert system toolset (P-BEST). In *1999 IEEE Symposium on Security and Privacy, Oakland, California, USA, May 9-12, 1999*, pages 146–161, 1999. IEEE Computer Society. 11, 17

[Manouselis and Costopoulou 2007] N. Manouselis and C. Costopoulou. Analysis and classification of multi-criteria recommender systems. *World Wide Web*, 10(4):415–441, 2007. 80

[Martinelli and Santini 2014] F. Martinelli and F. Santini. Debating cybersecurity or securing a debate? - (position paper). In F. Cuppens, J. García-Alfaro, A. N. Zincir-Heywood, and P. W. L. Fong, editors, *Foundations and Practice of Security - 7th International Symposium, FPS 2014, Montreal, QC, Canada, November 3-5, 2014. Revised Selected Papers*, volume 8930 of *Lecture Notes in Computer Science*, pages 239–246, 2014. Springer. 47

[Martinelli et al. 2015] F. Martinelli, F. Santini, and A. Yautsiukhin. Network security supported by arguments. In *Thirteenth Annual International Conference on Privacy, Security and Trust (PST). IEEE*, 2015. 48

[McDermott 2000] J. P. McDermott. Attack net penetration testing. In M. E. Zurbo and S. J. Greenwald, editors, *Proceedings of the 2000 Workshop on New Security Paradigms, Ballycotton, Co. Cork, Ireland, September 18-21, 2000*, pages 15–21, 2000. ACM. 18

[Michel and Mé 2001] C. Michel and L. Mé. *Trusted Information: The New Decade Challenge*, chapter ADeLe: An Attack Description Language for Knowledge-Based Intrusion Detection, pages 353–368. Springer US, Boston, MA, 2001. 19

[Montibeller and Franco 2010] G. Montibeller and A. Franco. Multi-criteria decision analysis for strategic decision making. In *Handbook of multicriteria analysis*, pages 25–48. Springer, 2010. 80

[Mu and Li 2010] C. Mu and Y. Li. An intrusion response decision-making model based on hierarchical task network planning. *Expert Syst. Appl.*, 37(3):2465–2472, 2010. 11, 13, 15

[Musman and Flesher 2000] S. Musman and P. Flesher. System or security managers adaptive response tool. *DARPA Information Survivability Conference and Exposition,*, 2:1056, 2000. 15

[Oglaza et al. 2014] A. Oglaza, R. Laborde, and P. Zaraté. Kapuer: un assistant à l'écriture de politiques d'autorisation pour la protection de la vie privée. *Ingénierie des Systèmes d'Information*, 19(6):91–115, 2014. 80

[Papadaki and Furnell 2006] M. Papadaki and S. Furnell. Achieving automated intrusion response: a prototype implementation. *Inf. Manag. Comput. Security*, 14(3):235–251, 2006. 13, 15

[Pazzani and Billsus 1997] M. J. Pazzani and D. Billsus. Learning and revising user profiles: The identification of interesting web sites. *Machine Learning*, 27(3):313–331, 1997. 80

[Porras and Neumann 1997] P. A. Porras and P. G. Neumann. EMERALD: event monitoring enabling responses to anomalous live disturbances. In *1997 National Information Systems Security Conference*, oct 1997. 10, 15

[Project] E. Project. E-safety Vehicle InTrusion protected Applications. `http://www.evita-project.org`. 69, 162

[Qt] Qt. `http://www.qt.io/`. 99

[Ragsdale et al. 2000] D. Ragsdale, C. Carver, J. Humphries, and U. Pooch. Adaptation techniques for intrusion detection and intrusion response systems. In *Systems, Man, and Cybernetics, 2000 IEEE International Conference on*, volume 4, pages 2344–2349 vol.4, 2000. 10, 15

[Resnick et al. 1994] P. Resnick, N. Iacovou, M. Suchak, P. Bergstrom, and J. Riedl. Grouplens: An open architecture for collaborative filtering of netnews. In *CSCW '94, Proceedings of the Conference on Computer Supported Cooperative Work, Chapel Hill, NC, USA, October 22-26, 1994*, pages 175–186, 1994. 80

[Roesch 1999] M. Roesch. Snort: Lightweight intrusion detection for networks. In D. W. Parter, editor, *Proceedings of the 13th Conference on Systems Administration (LISA-99), Seattle, WA, November 7-12, 1999*, pages 229–238, 1999. USENIX. 8

[Rowe et al. 2012] J. Rowe, K. N. Levitt, S. Parsons, E. Sklar, A. Applebaum, and S. Jalal. Argumentation logic to assist in security administration. In R. Ford, M. E. Zurko, C. Herley, and T. Whalen, editors, *The New Security Paradigms Workshop, NSPW '12, Bertinoro, Italy - September 18 - 21, 2012*, pages 43–52, 2012. ACM. 39

[Ruddle et al. 2010] EVITA Project. Security Requirements for Automotive On-Board Networks based on Dark-side Scenarios. Technical Report 2.3, 2010. 1

[Saaty] T. Saaty. The analytic hierarchy process, mcgraw-hill, new york, 1980. *There is no corresponding record for this reference.* 83

[Samarji et al. 2013] L. Samarji, F. Cuppens, N. Cuppens-Boulahia, W. Kanoun, and S. Dubus. Situation calculus and graph based defensive modeling of simultaneous attacks. In G. Wang, I. Ray, D. Feng, and M. Rajarajan, editors, *Cyberspace Safety and Security*, volume 8300 of *Lecture Notes in Computer Science*, pages 132–150. Springer International Publishing, 2013. 56, 73

[Samarji et al. 2015] L. Samarji, N. Cuppens-Boulahia, F. Cuppens, S. Papillon, W. Kanoun, and S. Dubus. On the fly design and co-simulation of

responses against simultaneous attacks. In G. Pernul, P. Y. A. Ryan, and E. R. Weippl, editors, *Computer Security - ESORICS 2015 - 20th European Symposium on Research in Computer Security, Vienna, Austria, September 21-25, 2015, Proceedings, Part II*, volume 9327 of *Lecture Notes in Computer Science*, pages 642–661, 2015. Springer. 64, 114, 127

[Schnackengerg et al. 2001] D. Schnackengerg, H. Holliday, R. Smith, K. Djahandari, and D. Sterne. Cooperative intrusion traceback and response architecture (citra). In *DARPA Information Survivability Conference amp; Exposition II, 2001. DISCEX '01. Proceedings*, volume 1, pages 56–68 vol.1, 2001. 15

[Sendi et al. 2012] A. S. Sendi, M. Dagenais, M. Jabbarifar, and M. Couture. Real time intrusion prediction based on optimized alerts with hidden markov model. *JNW*, 7(2):311–321, 2012. 14

[Shameli-Sendi et al. 2012] A. Shameli-Sendi, N. Ezzati-Jivan, M. Jabbarifar, and M. Dagenais. Intrusion response systems: Survey and taxonomy. *International Journal of Computer Science and Network Security*, 12(1):1–14, 2012. 12

[Shameli-Sendi et al. 2013] A. S. Shameli-Sendi, J. Desfossez, M. R. Dagenais, and M. Jabbarifar. A retroactive-burst framework for automated intrusion response system. *Journal Comp. Netw. and Communic.*, 2013:134760:1–134760:8, 2013. 13, 15

[Somayaji and Forrest 2000] A. Somayaji and S. Forrest. Automated response using system-call delay. In S. M. Bellovin and G. Rose, editors, *9th USENIX Security Symposium, Denver, Colorado, USA, August 14-17, 2000*, 2000. USENIX Association. 15

[Soojin et al. 2006] Soojin, B. Chung, H. Kim, Y. Lee, C. Park, and H. Yoon. Real-time analysis of intrusion detection alerts via correlation. *Computers & Security*, 25(3):169–183, 2006. 11

[Stakhanova et al. 2007a] N. Stakhanova, S. Basu, and J. Wong. A cost-sensitive model for preemptive intrusion response systems. In *21st International Conference on Advanced Information Networking and Applications (AINA 2007), May 21-23, 2007, Niagara Falls, Canada*, pages 428–435, 2007. IEEE Computer Society. 15

[Stakhanova et al. 2007b] N. Stakhanova, S. Basu, and J. Wong. A taxonomy of intrusion response systems. *IJICS*, 1(1/2):169–184, 2007. 7, 13

[Steffan and Schumacher 2002] J. Steffan and M. Schumacher. Collaborative attack modeling. In G. B. Lamont, H. Haddad, G. A. Papadopoulos, and B. Panda, editors, *Proceedings of the 2002 ACM Symposium on Applied Computing (SAC), March 10-14, 2002, Madrid, Spain*, pages 253–259, 2002. ACM. 18

[Strasburg et al. 2009] C. Strasburg, N. Stakhanova, S. Basu, and J. S. Wong. A framework for cost sensitive assessment of intrusion response selection. In S. I. Ahamed, E. Bertino, C. K. Chang, V. Getov, L. Liu, H. Ming, and R. Subramanyan, editors, *Proceedings of the 33rd Annual IEEE International Computer Software and Applications Conference, COMPSAC 2009, Seattle, Washington, USA, July 20-24, 2009. Volume 1*, pages 355–360, 2009. IEEE Computer Society. 13, 15

[Tanachaiwiwat et al. 2002] S. Tanachaiwiwat, K. Hwang, and Y. Chen. Adaptive intrusion response to minimize risk over multiple network attacks. *ACM Trans on Information and System Security*, 2002. 15

[Toth and Krügel 2002] T. Toth and C. Krügel. Evaluating the impact of automated intrusion response mechanisms. In *18th Annual Computer Security Applications Conference (ACSAC 2002), 9-13 December 2002, Las Vegas, NV, USA*, pages 301–310, 2002. IEEE Computer Society. 11, 12, 15

[Wang et al. 2001] X. Wang, D. S. Reeves, S. F. Wu, and J. Yuill. Sleepy watermark tracing: An active network-based intrusion response framework. In M. Dupuy and P. Paradinas, editors, *Trusted Information: The New Decade Challenge, IFIP TC11 Sixteenth Annual Working Conference on Information Security (IFIP/Sec'01), June 11-13, 2001, Paris, France*, volume 193 of *IFIP Conference Proceedings*, pages 369–384, 2001. Kluwer. 15

[White and Pooch 1996] G. B. White and U. W. Pooch. Cooperating security managers: Distributed intrusion detection systems. *Computers & Security*, 15(5):441–450, 1996. 11, 15

[Wool 2004] A. Wool. A quantitative study of firewall configuration errors. *IEEE Computer*, 37(6):62–67, 2004. 39

[Yu and Frincke 2007] D. Yu and D. A. Frincke. Improving the quality of alerts and predicting intruder's next goal with hidden colored petri-net. *Computer Networks*, 51(3):632–654, 2007. 14

[Zeleny 1982] M. Zeleny. *Multiple Criteria Decision Making*. McGraw-Hill, New York, 1982. 80

List of Figures

List of Tables

Résumé

L'accroissement et la diversification des services offerts par les systèmes informatiques modernes rendent la tâche de sécuriser ces systèmes encore plus complexe. D'une part, l'évolution du nombre de services système accroît le nombre des vulnérabilités qui peuvent être exploitées par des attaquants afin d'atteindre certains objectifs d'intrusion. D'autre part, un système de sécurité moderne doit assurer un certain niveau de performance et de qualité de service tout en maintenant l'état de sécurité. Ainsi, les systèmes de sécurité modernes doivent tenir compte des exigences de l'utilisateur au cours du processus de sécurité.

En outre, la réaction dans des contextes critiques contre une attaque après son exécution ne peut pas toujours remédier à ses effets néfastes. Dans certains cas, il est essentiel que le système de sécurité soit en avance de phase par rapport à l'attaquant et de prendre les mesures nécessaires pour l'empêcher d'atteindre son objectif d'intrusion.

Nous soutenons dans cette thèse que le processus de sécurité doit suivre un raisonnement intelligent qui permet au système de prévoir les attaques qui peuvent se produire par corrélation à une alerte détectée et d'appliquer les meilleures contre-mesures possibles.

Nous proposons une approche qui génère des scénarios potentiels d'attaque qui correspondent à une alerte détectée. Ensuite, nous nous concentrons sur le processus de génération d'un ensemble approprié de contre-mesures contre les scénarios d'attaque générés. Un ensemble généré des contre-mesures est considéré comme approprié dans l'approche proposée s'il présente un ensemble cohérent et il satisfait les exigences de l'administrateur de sécurité (par exemple, la disponibilité). Nous soutenons dans cette thèse que le processus de réaction peut être considéré comme un débat entre deux agents. D'un côté, l'attaquant choisit ses arguments comme étant un ensemble d'actions pour essayer d'atteindre un objectif d'intrusion, et de l'autre côté l'agent défendant la cible choisit ses arguments comme étant un ensemble de contre-mesures pour bloquer la progression de l'attaquant ou atténuer les effets de l'attaque.

D'autre part, nous proposons une approche basée sur une méthode d'aide à la décision multicritère. Cette approche assiste l'administrateur de sécurité lors de la sélection des contre-mesures parmi l'ensemble approprié des contre-mesures générées à partir de la première approche. Le processus d'assistance est basé sur l'historique des décisions de l'administrateur de sécurité. Cette approche permet également de sélectionner automatiquement des contre-mesures appropriées lorsque l'administrateur de sécurité est dans l'incapacité de les sélectionner (par exemple, en dehors des heures de travail, par manque de connaissances sur l'attaque). Enfin, notre approche est implémentée et testée dans le cadre des systèmes automobiles.

Mots-clés: Logique argumentative, Réponses à l'intrusion, Sélection des contre-mesures, Multi-Criteria Decision Making, Systèmes véhiculaire, AIRS, Langage de Description d'Attaque, Anti-corrélation.

Abstract

The growth and diversity of services offered by modern systems make the task of securing these systems a complex exercise. On the one hand, the evolution of the number of system services increases the risk of causing vulnerabilities. These vulnerabilities can be exploited by malicious users to reach some intrusion objectives. On the other hand, the most recent competitive systems are those that ensure a certain level of performance and quality of service while maintaining the safety state. Thus, modern security systems must consider the user requirements during the security process.

In addition, reacting in critical contexts against an attack after its execution can not always mitigate the adverse effects of the attack. In these cases, security systems should be in a phase ahead of the attacker in order to take necessary measures to prevent him/her from reaching his/her intrusion objective.

To address those problems, we argue in this thesis that the reaction process must follow a smart reasoning. This reasoning allows the system, according to a detected attack, to preview the related attacks that may occur and to apply the best possible countermeasures.

On the one hand, we propose an approach that generates potential attack scenarios given a detected alert. Then, we focus on the generation process of an appropriate set of countermeasures against attack scenarios generated among all system responses defined for the system. A generated set of countermeasures is considered as appropriate in the proposed approach if it presents a coherent set (i.e., it does not contain conflictual countermeasures) and it satisfies security administrator requirements (e.g., performance, availability). We argue in this thesis that the reaction process can be seen as two agents arguing against each other. On one side the attacker chooses his arguments as a set of actions to try to reach an intrusion objective, and on the other side the agent defending the target chooses his arguments as a set of countermeasures to block the attacker's progress or mitigate the attack effects.

On the other hand, we propose an approach based on a recommender system using Multi-Criteria Decision Making (MCDM) method. This approach assists security administrators while selecting countermeasures among the appropriate set of countermeasures generated from the first approach. The assistance process is based on the security administrator decisions historic. This approach permits also, to automatically select appropriate system responses in critical cases where the security administrator is unable to select them (e.g., outside working hours, lack of knowledge about the ongoing attack). Finally, our approaches are implemented and tested in the automotive system use case to ensure that our approaches implementation successfully responded to real-time constraints.

Keywords: Argumentative logic, Intrusion response, Countermeasures selection, Multi-Criteria Decision Making, Automotive system, AIRS, Attack description language, Anti-correlation.

IMT Atlantique
Bretagne-Pays de la Loire
École Mines-Télécom

N° d'ordre : 2017IMTA0013

IMT Atlantique Bretagne-Pays de la Loire - www.imt-atlantique.fr

Campus de Brest
Technopôle Brest-Iroise
CS 83818
29238 Brest Cedex 03
T +33 (0)2 29 00 11 11
F +33 (0)2 29 00 10 00

Campus de Nantes
4, rue Alfred Kastler - La Chantrerie
CS 20722
44307 Nantes Cedex 3
T +33 (0)2 51 85 81 00
F +33 (0)2 51 85 81 99

Campus de Rennes
2, rue de la Châtaigneraie
CS 17607
35576 Cesson Sévigné Cedex
T +33 (0)2 99 12 70 00
F +33 (0)2 99 12 70 08