Simulation of activities and attacks : application to cyberdefense

Pierre-Marie Bajan

université PARIS-SACLAY

NNT : 2019SACLL014

Thèse de doctorat

Simulation d'activités et d'attaques : application à la cyberdéfense

Thèse de doctorat de l'Université Paris-Saclay
préparée à l'IRT SystemX avec Télécom SudParis

Ecole doctorale n°580 Sciences et technologies de l'information et de la communication (STIC)
Spécialité de doctorat : Réseaux, Information et Communications

Thèse présentée et soutenue à Evry, le 5 Juillet 2019, par

PIERRE-MARIE BAJAN

Composition du Jury :

Mr Christophe Bidan Professeur, Centrale Supélec	Rapporteur
Mr Michaël Hauspie Maître de Conférences HDR, Université de Lille	Rapporteur
Mr Gaël Thomas Professeur, Télécom SudParis	Examinateur
Mme Isabelle Chrisment Professeur, Télécom Nancy	Examinateur
Mr Hervé Debar Professeur, Télécom SudParis	Directeur de thèse
Mr Christophe Kiennert Maître de Conférences, Télécom SudParis	Encadrant / Invité
Mr Makhlouf Hadji Chercheur, IRT SystemX	Encadrant / Invité

Contents

List of Figures

List of Tables

Abstract

The evaluation of security products is a key issue in cybersecurity. Numerous tools and methods can evaluate the properties of services (compliance with the specifications, workload processing capacity, resilience to attacks) and security products (policy accuracy, attack coverage, performances overhead, workload processing capacity).

Most existing methods only evaluate some of those properties. Methods, like testbed environments, that can cover all aspects are costly in resources and manpower. Few structures can afford the deployment and maintenance of those testbed environments. In this thesis, we propose a new method to generate at a large scale evaluation data that match the evaluator's evaluation requirements.

We base our method on the deployment of a small program on a lightweight virtual network. That program reproduces model data according to the need of the evaluator. Those needs are translated into levels of realism. Those levels match the characteristics of the model data preserved by the simulation program.

We formally present our method and introduce additional requirements (customization, reproducibility, realism, accuracy, scalability) as properties of our model. We also explain the step by step construction of our prototype along with the experimental validation of our method.

Although our prototype's functions are currently limited, we can still use our prototype to evaluate a security product. We first introduce a methodology to apply our method to the evaluation of services and security products. We then conduct a series of experiments according to the methodology to evaluate an intrusion detection system.

Our evaluation of an intrusion detection system illustrates the advantages of our method but it also underline the current limitation of our prototype. We propose a series of improvements and development to conduct to transform our current limited prototype into an efficient evaluation tool that can evaluate services and security products alike.

Thanks

I would like to especially thanks:

- Hervé Debar without whom this thesis would have not be possible;
- Christophe Kiennert for his patience and wondrous corrections in all my papers;
- IRT SystemX for financing and providing support for my thesis project;
- Flavien Quesnel for always helping me in all my difficulties;
- Jérôme Letailleur and Isabelle Hirayama for their genuine interest in my work;
- Nada Essaouini for her support and encouragement even though she never read my papers;
- Philippe Wolf for helping me work in better conditions;
- The whole team EIC;
- my family and friends for their continuous support and patience;

CHAPTER 1

Introduction

1.1 Context

Information systems in companies and governments heavily depend on security products to protect information and infrastructure. There exists a wide variety of security products: firewall, IDS, antivirus, WAF (Web Application Firewall), SBC (Session Border Control), DLP (Data Leak Prevention), etc. to protect their system and assets. It is difficult for the administrator of a system to select the optimal products, organize them in a functioning topology and properly supervise all those different products. It is crucial for the administrator to have a thorough, fair comparison of the security products.

The evaluation of security products covers a wide range of considerations that need to be tested: the security properties of the product (security policy, attack coverage,workload processing capacity, etc., [Milenkoski et al. 2015]), the impact on the system (performances overhead and overhead for the users [Kainda et al. 2010]), and the impact on already deployed security products (avoid inconsistencies in the deployment of configuration policies [Garcia-Alfaro et al. 2011]). As such, we require reliable evaluation tools that can adapt to the needs of the evaluator.

A large variety of solutions exists to evaluate the different aspects of security products. They can broadly be divided into semantic tests that evaluate specific properties of the product, and load tests that evaluate the workload processing capacity of the product. The semantic tests are mostly home-grown, targeting specific functionalities or vulnerabilities of the evaluation target and are often not scalable. To deploy those tests at a large scale, the evaluator needs a large network infrastructure like a testbed environment. On the other hand, intensive tests highly stress resources of the evaluation target and can be used at a large scale. However, they solely target some resources (memory, CPU, I/O, etc.) and their executions are easily detectable. So, security products that are not evaluated in a testbed are exposed to two separate tests that neither accurately reflect the real-world environment. In an operational context, security products must ensure specific functionalities or protect against a variety of attacks while under the stress of a large amount of regular traffic. Current evaluations mostly test those two aspects separately and do not take into account the impact of the stress regular traffic on the functionalities of security products. That problem was already highlighted in 2003 by the U.S. Department of Commerce in [Mell et al. 2003].

To properly evaluate security products in an operational context close to real world, apart from providing traces from the real world, evaluators employ testbed environments. Testbed environments provide an extensive network infrastructure that can launch a large variety of semantic tests to generate evaluation data whose content is fully known and that is close to real-world activity. To reduce deployment and maintenance costs, testbed environments often turn to virtual network infrastructure. Despite that, the main drawback is the resource and time costs needed to set up, configure and maintain the testbed. While being an efficient solution to evaluate security products in an operational context, the cost is too high for most structures. We need an affordable evaluation tool or environment that can provide evaluation data with a high level of semantic at large scale. The evaluation data must be adaptable to the needs of the evaluator and offer the possibility to cover all kinds of evaluation.

1.2 Objectives and contributions

The goal of this thesis is to conceive and validate a method to generate evaluation data that provide specific and controlled interactions with the evaluation target of semantic tests at a large scale. To do that, we propose a new network simulation method that is independent of the nature of the the virtual network infrastructure and can work on a virtual environment with lesser cost requirements than the virtual networks used in a testbed environment.

Our method must also adapt to the evaluator's requirements. Evaluations do not all have the same goals and needs. Some evaluations might need to control the data at the application level while others may only require data with much lower criteria (e.g., a volumetric evaluation). Evaluators should be able to choose a tradeoff between the level of control over the data and the resources they are willing to spend. In this thesis, our method offers several functions to generate data with different criteria, with different *levels of realism*. The higher the level of realism the evaluator needs, the more stringent the requirements for the input of the associated function will be.

Our contributions are of several types:

- We conceive a formal model of our method and extract several properties of our model to ensure we are as close as possible to ideal data generation method (reproducible, realistic, scalable, accurate, adaptable).
- From that model, we implement a prototype of our method on a lightweight simulator and simulate the network activity of a small company. We also present the experimental verification data and identify the strengths and weaknesses of our implementation [Bajan et al. 2018].
- We also devise a methodology to apply our method for the evaluation of security products and services. We illustrate this methodology with the evaluation of an open-source security product with our network simulation prototype[Bajan et al. 2019].

1.3 Outline of the thesis

Chapter 2 presents a general overview of the evaluation of security products and services. This overview starts with the presentation of fundamental knowledge of the evaluation of services and security products as they require the verification of different properties. This chapter also presents the different tools available to the evaluation of products and the different types of existing virtual networks.

Chapter 3 presents the formal model of our simulation. It begins with the definition of our different concepts. We deduce the properties of our model to attain an ideal data generation method for evaluation.

Chapter 4 describes the simulation network we developed according to our model. We give the steps necessary for the prototype along with the procedure to obtain the required inputs. We also explain the choice we made for the network infrastructure and the performance results we obtained.

Chapter 5 presents the application of our method. We explain the evaluation methodology to follow in the evaluation of security products and services and apply it to the evaluation of a security product. We analyze the results and explain what the current shortcomings of our method and prototype are and propose some improvements and future developments.

Finally, we conclude our work and discuss perspectives in Chapter 6.

CHAPTER 2

Evaluation of security products: a broad overview

Security products are meant to be deployed in an operational, real-life context. The managers of computer systems must be assured that the product they select can:

- handle an operational context: in real-life use, the product must process large volumes of requests from users and accomplish all its functionalities. If the product does not handle the required volume of requests or does not comply with its specifications, it can not be used in an operational context.
- provide accurate results: the manager of the network needs to know if he can trust the result of the products that monitor and secure the network. For example, if the product is a firewall, the manager needs to know that the security rules are correctly applied.
- provide exploitable results: the evaluated target must provide results that can be understood and used by the manager. If important information is overwhelmed by irrelevant or less urgent information, the manager would not be able to appropriately react.

Those requirements are difficult to evaluate outside of an evaluation context close to an operational environment. In that context, the evaluation of security products is also linked to the performance and behavior of the services present on the network, especially for a security product like an intrusion detection system (IDS) or a firewall. In-line security products are placed on the path of network activity between users and services on a computer system. Those security products take measures (authorize, block, alert, etc.) according to the data that pass through them. In consequence, the evaluation of such products requires that the evaluation target analyzes exchanges between users and services. The consideration of the evaluation of services is thus relevant to the evaluation of security products. Those two types of evaluation targets, security products and services, present similar concerns and rely on similar tools. Moreover, it is valuable to know the impact that the insertion of a security product has over the performance of services.

Thus, we consider as our evaluation targets the combination of both services and security products. Even if the evaluation of services and security products presents some similarities, the properties to verify in their evaluation are different. Therefore, before going into the details of our proposed method and approach to network simulation, there is a need to explain the fundamentals of the evaluation of security products and services, and to describe the evaluation tools and current methods for creating virtual networks.

2.1 Fundamental knowledge of the evaluation of security products and services

Figure 2.1 illustrates the different elements involved in an evaluation. The evaluation target can be either a service or a security product. Their evaluation requires an environment that provides a network and actors. The actors can be regular *users* or *attackers*. Depending on the nature of the evaluation target, it can require the presence of both types of targets. For example, security products can be in-line products like a firewall, or they can be out-of-band, like an authentication server.

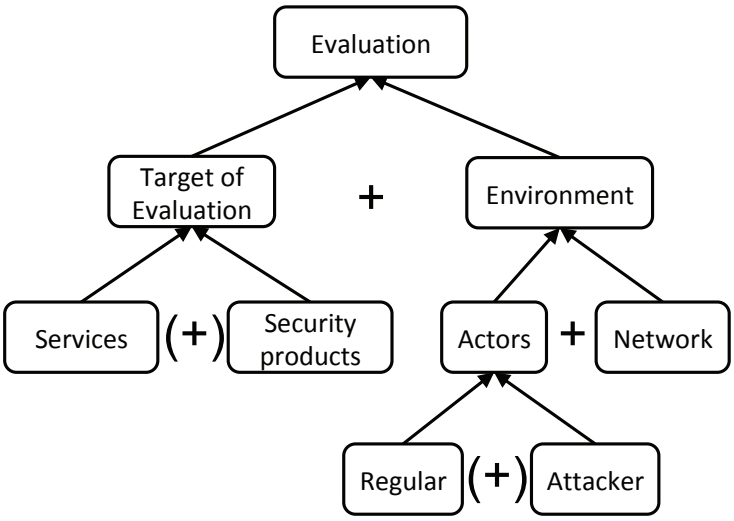

Figure 2.1 – Elements of an evaluation

In-line security products are placed in the network so that incoming traffic has to go through them before reaching its destination. They only work when placed in the middle of the traffic between actors and services. They can have an action on that traffic. Thus the evaluation of such products requires the presence of services alongside the security product. On the other hand, the evaluation of most services does not require the presence of a security product. Similarly, an evaluation that verifies the compliance of a service with specifications does not require the presence of an attacker.

Services and security products are the targets of a large variety of evaluations, each with different goals in mind. Such evaluations aim to verify specific properties that can

be different for services and security products. In this section, we describe the main properties for services and security products.

2.1.1 Evaluation of services

A service must be able to meet the needs of the users (supporting enough requests for operational use, compliance to the specifications, etc.) with a reasonable performance overhead (memory, CPU, I/O, energy, etc.). It must also be able to resist the actions of an attacker in the case a security product does not adequately protect it. Thus, the evaluation of services must verify the following properties:

- compliance with the specifications
- workload processing capacity
- resilience to attacks

The first goal of the evaluation of a service is to verify that the service complies with its design. For services produced with model-driven engineering (UML, SysML, etc.), there are two steps to this evaluation: a validation of the model (does this service meet the needs?) and a verification of the model (was this service correctly built?). In [Gogolla and Hilken 2016], the authors propose essential use cases for model exploration, validation, and verification of a structural UML model enriched by OCL (Object Constraint Language) invariants and demonstrate them on a model validator.

The workload processing capacity is an evaluation of the capacity of the product to handle a large number of requests and significant stress. The goal of such an evaluation is to ensure that the service will be resilient when in operation and can meet the demands of users. This evaluation can determine if enough resources were allocated to the service, or if it can provide inputs to improve the performances of the service. For example, [Nahum et al. 2007] studied the performance of a SIP server according to the scenario and use of the protocol.

Lastly, a service can be evaluated based on its resilience to attacks. Services need to be able to resist attacks on their own as much as possible, even before additional protection and detection layers are added. The goal of such an evaluation is to determine the scope of the attacks the service can resist or not. This evaluation aims to find vulnerabilities due to configuration flaws and vulnerabilities specific to the service itself. According to the complexity and widespread use of the products, a consequent number of vulnerabilities can be discovered each year. The product vendor must then plan to fix the vulnerabilities as soon as possible while those using or depending on that product must check if their own version is vulnerable and act in consequence. The more vulnerabilities a product has, the more consequent is the security risk of using this product. To help predict the discovery of vulnerability in the two major HTTP servers (Apache Server and IIS), [Alhazmi and Malaiya 2006] carried out a detailed analysis of the prediction capabilities of two models for the discovery of vulnerabilities.

2.1.2 Evaluation of Security Products

Many types of security products exist: firewall, IDS, antivirus, WAF (Web Application Firewall), SBC (Session Border Control), DLP (Data Leak Prevention), etc. It is necessary to know how well they detect and/or block attacks (accuracy, attack coverage, workload processing capacity) and how much overhead they impose on the system.

Therefore, the evaluation of a security product aims at validating the following properties:

- policy accuracy
- attack coverage
- performance overhead
- workload processing capacity

Policy accuracy regards the correctness of the judgement of the security product. That judgement can take different forms: detecting an attack, accepting credentials, detecting abnormal behavior, etc. The evaluation of this property requires the security product to judge a mixed set of interactions (attack/regular, accepted/rejected, etc.). The correctness of that judgement is based on the policy of the security product that can be flawed or impacted by configuration issues. In [Garcia-Alfaro et al. 2011], the authors present a management tool that analyzes configuration policies and assists in the deployment of configuration policies of network security components. The goal is to ensure the absence of inconsistencies in the policies of security products on the network. This property is also linked to the issue of the base-rate fallacy[Axelsson 2000] in in-line security products like IDSs, where too many false positive alerts(falsely raised alerts) make the security product unusable for the supervisor of the system. In the evaluation of this property we want to know the optimal configuration of the security product that minimize the false-positive while maximizing the true-positive rate.

The evaluation of attack coverage aims to determine the range of attacks that can impact the security product, the range of attacks successfully blocked by the security product and those that are not handle by it. It allows the evaluator to know the strengths and weaknesses of the security products in the range of known attacks or vulnerabilities. This help the administrator of a network to know an appropriate combination of security products for its system. The coverage limit of the security product can be the direct results of its scope and level of layer examination. Some products examine only the transport layers while others examine up till the application layer. The attack coverage of a security product can also be impacted by the configuration of the product.

Finally, the evaluation of performance overhead deals with the resources consumed by the security product. However, performance overhead usually focuses on the impact of the addition of a security product on the performances of the network. An inline security product analyzes the network traffic that passes through it. Therefore, if the

security product has poor performances, it can slow down the overall traffic on the network. Similarly, if an out-of-band security product replies to requests slowly, it can also impact the performances of services that depend on that security product. For example, the use of DNSSEC to secure DNS against cache poisoning attacks has a negative impact on the performances of networks, as pointed out in [Migault et al. 2010].

2.2 Overview of evaluation tools

The evaluation targets, like services and security products, must be verified in all kinds of properties. A large variety of tools exists to evaluate those aspects. They can be divided into two types: executable workloads (2.2.1) and traces (2.2.2)[Milenkoski et al. 2015]. Executable workloads are meant to generate evaluation data during live testing and traces are records of real-life activity or artificial activity that are used as inputs in offline testing. Trace are records of an activity. They can be records from real world activity or an artificial activity generated by executable workloads. Those two types of methods are not completely separated from each other but represent two different types of testing: *live* and *offline*.

2.2.1 Executable workloads

Executable workloads are employed for live testing a security product. They run on a physical machine, a virtual machine or a virtual network to generate data that will be fed to a tested security product. The support on which the executable workload runs depends on the nature of the tested product (network-based or host-based product) or the resources available to evaluators (physical network, emulated virtual network, or simulated virtual network). The differences between the various networks used for support are explained in Section 2.3.

Executable workloads allow the test of security products in a context close to an operational context and are often used to create traces for other structures that cannot handle the investment of live testing. This investment has numerous forms: malicious workloads often work on a specific victim environment that needs to be reproduced and set up. That environment needs to be restored to its previous state after each experiment. The malicious workload might also crash the victim environment and render it unstable. All those tasks are time-consuming and expensive.

The survey by [Milenkoski et al. 2015] on common practices of IDS evaluation describes four kinds of executable workloads. **Workload drivers** and **manual generation** are used to generate *pure benign* workloads (with no attack). **Exploit database** and **vulnerability and attack injection** are used to generate pure malicious workloads (contains only attacks). *Mixed* executable workloads, or workloads that contain both malicious and legitimate traffic, are a mix of different executable workloads generating pure benign and pure malicious workloads. We present summary of the survey

on the different types of executable workloads in the following tables: Table 2.1, Table 2.2, Table 2.3 and Table 2.4.

Description	Drivers designed to generate artificial pure benign evaluation data with different characteristics (e.g., CPU-intensive, I/O intensive, ...)
Types of evaluation data	Pure benign evaluation data
Types of security products	Works for network-based and host-based security products
Advantages	+ Customization of the intensity behavior of the workload
Disadvantages	- Often, it does not resemble real life data
Examples	SPEC CPU2006, iozone, Postmark, httpbench, UnixBench, ...

Table 2.1 – Summary of the analysis of workload drivers

In Table 2.1, we see that workload drivers are drivers designed to produce a customizable pure benign workload with a specific intensity of one characteristic of the workload. Those workloads are used to evaluate the performances of security products for intensive benign workloads. However, the generated workloads are characteristic to the driver and do not closely mimic real-life workloads. Workload drivers are mainly used to evaluate the workload processing capacity of evaluation targets. They do not evaluate all functionalities of the evaluation targets making them unsuitable for the verification of compliance to the specifications. Also, as the interactions of workload drivers with evaluation targets do not resemble the operational use in real-life, they can be detrimental to the evaluation of products with a learning phase, like anomaly-based IDS. Examples of such drivers are the SPEC CPU 2006 [Henning 2006][Ming et al. 2017] that generates CPU-intensive workloads, ApacheBench [Riley et al. 2008][Jin et al. 2013] that generates intensive HTTP workloads. and iozone [Jin et al. 2013] that generates intensive I/O.

Description	The execution of real system users' tasks known to exercise system resources
Types of evaluation data	Pure benign evaluation data
Types of security products	Often used for host-based security products
Advantages	+ With a realistic activity model, it resembles real-life workloads
	+ Suitable for traces in a recording testbed environment
Disadvantages	- Temporal and intensity characteristic cannot be customized
	- Require substantial amount of manpower
Examples	[Srivastava et al. 2008], [Lombardi and Di Pietro 2011], [Reeves et al. 2012]

Table 2.2 – Summary of the analysis of manual generation

The second method to create benign executable workloads is manual generation. Table 2.2 presents the strengths and weaknesses of manually generating traffic where the evaluator executes user tasks on real systems that use a specific portion of the resources. Examples of such tasks are copying large files to force a large quantity of I/O on the system, or file encoding to emulate CPU-intensive tasks. With a realistic activity model, workloads generated that way will closely resemble those found in real life. That method is preferred for the generation of workloads on testbeds. However, the intensity and time-control of those workloads cannot be customized like in workload

drivers as they are difficult to control and predict. Moreover, this method may require consequent manpower and resources. We also consider in that category homegrown scripts that can either execute consuming tasks or verify the specification of a service or a security product. Those scripts are highly specific and difficult to generalize to a large scale. Thus this method is suited to verify the compliance with the specifications of the evaluation target but challenging to apply in the evaluation of the workload processing capacity.

	Manual assembly	**Readily available**
Description	Security researchers typically use an exploit database	
	Evaluators normally obtain attack scripts from public exploit repositories	Many researchers employ penetration testing tools as a readily available exploit database
Types of evaluation data	Pure malicious evaluation data	
Types of security products	Used for network-based and host-based security products	Of limited use for host-based security products
Advantages		+ Freely available tools like Metasploit enables a customizable and automated platform exploitation with an up-to-date exploit database + Many of the exploits can be used without crashing the victim environment
Disadvantages	- Locating and adapting exploits is time-consuming and requires in-depth knowledge of the inner-working and architecture of the product - Public exploit repositories do not feature techniques for evaluating the detection of evasive attacks	- Most exploits are executed from remote sources making them useful for network-based products but of limited use to host-based product
Examples	[Mell et al. 2003], [Lombardi and Di Pietro 2011]	Metasploit, Nikto, w3af, Nessus

Table 2.3 – Summary of the analysis of exploit databases

To evaluate the resilience to attacks on services or the properties of security products, the evaluator can manually attack the target, make his own attack script (exploit) or use an exploit database. We describe that method in Table 2.3. To generate pure malicious workload, evaluators commonly use an exploit database, a database of scripted attacks. It is also possible to assemble databases with the help of public exploit repositories manually. However, manually assembling an exploit database requires consequent of human resources and time to research and adapt the exploits. Many researchers prefer to use penetration testing tools such as Metasploit, a readily available exploit database. It is kept up-to-date and is freely available. However, many of the available exploits focus on network-based security products.

We present the last executable form of workloads in Table 2.4. It uses the principle of software fault injection to inject vulnerabilities and attacks. It injects exploitable

vulnerable code in a victim platform in order to attack the platform. It is useful in the case where an exploit is not a viable option. However, it requires in-depth knowledge of the architecture of the security product and its inner-working.

Description	Artificial injection of exploitable vulnerable code in a target platform and then attacking the platform. It relies on the principles of the more general research area of fault-injection.
Types of evaluation data	Pure malicious evaluation data
Types of security products	Used for network-based and host-based security products
Advantages	+ Although not yet mature, it is useful in cases when collection of attack scripts that exploit vulnerabilities is unfeasible
Disadvantages	- The injection of attacks such that the sensors of a security product under test are exercised requires in-depth knowledge of the architecture and mechanism of the product
Examples	[Fonseca et al. 2014]

Table 2.4 – Summary of the analysis of vulnerability and attack injection

Workload drivers, manual generation, exploit databases and vulnerability and attack injection are the four kinds of executable workloads commonly used for the evaluation of security products. Those methods can evaluate the different properties of services and security products separately. There also exists tools specialized in looking for known vulnerabilities – vulnerability scanners (e.g., Arachni, OWASP, OpenVAS, etc.) – or unknown vulnerabilities – fuzzing tools (Antiparser, Peach, Wapiti, Webfuzzer, etc.). They are specialized in evaluating the resilience to attacks of services. All those methods need an execution environment specific to the evaluation target and the malicious workloads used. That specific environment is often the barrier, along with the manpower and resources needed, that prevents evaluators from using this kind of methods, especially in the case of network-based security products that often require a testbed.

2.2.2 Traces

We call a *trace* the capture of activity. Depending on the needs of the tested security product, it could be packet captures, log files, resource consumption, etc. Traces are replayed with tools (ex: TCPreplay for network traces) and do not require a specific environment. Evaluators do not spend many resources to feed the traces to the security product, but the challenges of using traces often lie in the constitution of the ground truth.

The ground truth is an exact representation of the content of the traces. The entries in the ground truth is linked to the granularity of control of the evaluation. An entry can be at the level of logs (a line or a group of lines), at the level of packets (a packet, a segment, a flow), or combination of both. The inputs that informed each entry of the ground truth depends on the evaluator's level of control on the evaluation, the granularity of the simulation. It contains labels indicating the classification of each entry (for example: benign/attack) along with other information relevant to the

evaluation of the performances of security products (for example which type of attack that entry is). The accuracy of the evaluation depends on the correct composition of the ground truth. However, the ground truth is not always easy to compose, either because some information is missing (anonymized data) or because we are not sure of the delimitation between attacks and regular traffic (real-world production trace) or with other attacks (honeypots). Moreover, traces become quickly outdated as trends and attacks evolve fast.

In the survey [Milenkoski et al. 2015] on common practices of IDS evaluation, traces methods are split into two approaches: trace acquisition and trace generation. Trace acquisition concerns the use of traces made available to the evaluator from real-world activity or a fabricated activity modeled as a reference for the community. However, the evaluator does not always have access to traces adapted for his evaluation. In that case, the evaluator generates traces to be reused multiple times or shared with others: it is trace generation. The evaluator often captures traces in an isolated environment like a testbed, or with a honeypot. We summarized the information of the survey on trace acquisition in Table 2.5 and on trace generation in Table 2.6.

	Real-world production trace	**Publicly available trace**
Description	The process of obtaining real-world production traces from an organization (i.e., proprietary traces) or obtaining publicly available traces that are intended for use in security research	
Types of evaluation data	Pure benign, pure malicious and mixed evaluation data	
Types of security products	Used for network-based and host-based security products	
Advantages	+ Real-life workload	+ No legal restraints to use it + The ground truth is included with the traces
Disadvantages	- Difficult to obtain due to privacy concerns - The anonymization of the data may remove relevant data for the evaluation - The ground truth is difficult to construct	- It often contains errors and is quickly outdated - In-depth knowledge of the characteristics of the recorded activities to avoid inaccurate interpretation of results of studies
Examples	[Seeberg and Petrovic 2007], [Coull et al. 2007]	DARPA/KDD-99, CAIDA, DEFCON, ITA, LBNL/ICSI, MawiLab

Table 2.5 – Summary of the analysis of trace acquisition

Table 2.5 presents the two main type traces that an evaluator can acquire: real-world production traces and publicly available traces. The real-world production traces are traces that an organization has captured from its activity and made publicly available or available to a group of evaluators. Those traces are as close to reality as evaluation data can be but are also a source of concern for the organization that distributed them. They contain a lot of private data of the organization that can be damaging for the organization or its employees. The traces are anonymized to prevent hurting the interests of the organization. However, those techniques often either delete a significant amount of relevant data for the evaluator or not hide enough private information. A balance between those two conflicting interests must be found, and that conflict often

renders companies and other organizations unwilling to share traces of their activity. Moreover, as the traces are from real-world production, they are not labeled. The constitution of the ground truth is another challenge of using real-world production traces. A lot of time and human resources are needed to describe the traces entirely and, even then the correctness of the ground truth cannot be fully guaranteed. Traces from the real world can evaluate the workload processing capacity and resilience to attacks of services but cannot guarantee an evaluation of the compliance with the specifications unless the traces are specifically designed for the evaluation target. Traces are more convenient to use in the evaluation of in-line security products as the traces do not have to adapt to the security product. However, the uncertainty in the reliability of the ground truth does not make it a practical method to evaluate the policy accuracy of security products.

The other kind of trace acquisition presented in Table 2.5 is publicly available traces. It consists in datasets that are made available to the public, often from large-scale experiments made to generate traces close to the real world without having to worry about privacy and anonymization. There are no legal constraints to use those traces, but they have their risks. They often contain errors and are quickly outdated. They do not always match the victim environment of the evaluator, so the evaluator needs to have an in-depth knowledge of the dataset, the conditions of generation of the dataset and the ground truth. Otherwise, it could lead to an inaccurate interpretation of the results of the study of the evaluator.

When an evaluator does not have access to the appropriate traces, he can generate his own traces. Once generated, those traces are reused for other evaluations or by other evaluators of his research group. One of the possible methods for generating traces consists in using executable workloads (like the manual generation) and capturing traces in an isolated environment: a testbed environment. A testbed environment is large scale network environment where an evaluator can execute workloads (homegrown scripts, drivers, exploits, etc.) on a large number of machines (physical or virtual) to produce an artificial, but close to reality, activity. However, as we highlight in Table 2.6, building and maintaining a testbed is costly, especially for large-scale network activity, and the methods used for generating the workloads may be faulty or too simplistic. Moreover, even if a realistic model of activity is used to generate accurate and complex workloads, it will produce one-time datasets. The model of activity will become outdated and needs to be changed after a given period of time. The generation of the traces has to be redone periodically, or with every significant change to the activity model.

In Table 2.6, the other method that an IDS evaluator can use to generate his own traces is honeypots. Honeypots reproduce the interactions of real targets and trick real-world attackers into thinking that they face a real vulnerable system or services. Then the honeypots capture the actions performed by the attacker and the results. They are classified according to the level of interaction they can have with the attacker. Low-level interactions honeypots are easy to maintain and can generate a ground truth without difficulty, but they are also easy to detect for an attacker. On the other hand, honeypots with a high level of interactions are difficult to detect for the attacker and

	Testbed environment	Honeypots
Description	To avoid issues with acquiring traces, the evaluator generates its own traces to share.	
	Traces are generated in a recording testbed environment that executes executable workloads	By mimicking real systems and/or vulnerable services, honeypots record malicious activity performed by an attacker without his knowledge
Types of evaluation data	Pure benign, pure malicious and mixed evaluation data	Pure malicious evaluation data
Types of security products	Used for network-based and host-based security products	
Advantages	+ Used to generate publicly available traces to be used as a reference by the community	+ Ideal for generation of realistic pure malicious traces + Low level of interaction honeypots are easy to maintain and the activity is easy to label + High level of interaction honeypots are more flexible and hard to detect
Disadvantages	- It is costly to build a testbed that scales to realistic production environments - The method used to generate an activity may use faulty or simplistic methods - With a realistic activity model, the realistic traces are still one-time datasets	- The outcome of a trace generation campaign is uncertain. - Low level of interaction honeypots are easy to detect - High level of interaction honeypots are expensive to maintain and require time-consuming analysis of the recorded activity
Examples	[Cunningham et al. 1999], [Shiravi et al. 2012]	Sebek, Argos, honeybrid, HoneySpider, honey, nepenthes, ...

Table 2.6 – Summary of the analysis of trace generation

are more flexible than low-level ones. However, they are expensive to maintain, and the composition of the ground truth and the analysis of the attacker's activity is time-consuming.

To sum it up, traces are great tools for evaluators to feed a practical activity to a security product at a low cost. Evaluators can either obtain traces from the real-world or from publicly available source that are used as reference datasets by the community. However, those traces can often be outdated or not target the evaluator use case. According to the evaluation target of the evaluator, the traces might need to be extremely specific. In consequence, evaluators can generate their own traces to share with other evaluators and reuse in other evaluations. They capture the activity on a testbed environment or attacks from honeypots with various levels of interaction.

2.2.3 Summary and analysis

Executable workloads and traces have different strengths and weaknesses. They can be used broadly or limited to the evaluation of specific properties. In Table 2.7, we sum up the evaluation properties of those different methods.

Compliance (Comp.), Processing (Proc.), Resilience (Resi.)

	Services			Security products			
	Comp.	Proc.	Resi.	Policy	Coverage	Overhead	Proc.
drivers		X				X	X
manual generation	X		X	X	partial		
exploit database			X	X	X		
V&A injection			X	X	partial		
real-world production	partial	X	partial	partial	partial	X	X
publicly available	partial	X	X	X	partial	X	X
testbed environments	X	X	X	X	X	X	X
honeypots			partial	partial	partial		

Table 2.7 – Targets of the evaluation tools

Drivers can generate a large number of predetermined traffic that exercise the workload processing capacity of evaluation targets. They can also target specific resources of the evaluation targets to evaluate the performance overhead of a security product.

Homegrown scripts or manually generated activity can verify every functionality of an evaluation target. It is very suitable to evaluate the compliance with the specifications of an evaluation target, whether it is to test the functionalities of a service or the correctness of the policy of a security product. Homegrown scripts or manual attacks can be made to target specific vulnerabilities or try complex attack vectors. However, a lot of time and work is needed to make scripts and to generate a large number of attacks manually. While it can be useful in the resilience evaluation of services, homegrown scripts and manual attacks can limit the coverage of the evaluation of a security product.

Exploit databases are a practical choice to generate a large variety of attacks in an attack coverage evaluation. They are tools that can test a large variety of known vulnerabilities of services and security products. Exploit databases are used in the evaluation of the resilience of services and the accuracy of the policy of security products.

Vulnerability and attacks injections are a complementary solution to exploits. They can provide a way to generate attacks that can not be made with exploit scripts. However, the scope of current tools that rely on this technique is limited. They can handle a smaller attack coverage than exploit databases. This method can cover the same aspects than exploit databases but with more limitation.

Traces from real-world production can be used to evaluate the compliance with the specifications of services but are not practical. The traces need to target the specific product and be large enough to cover all the functionalities of the evaluation target. Traces from real-world production offer large samples of production data. Once replayed to services with a proper program, those traces can generate the stress of production

use. It assures that the services and security products can handle real-life production workloads. However, this method has limits for evaluation with attacks. An evaluator cannot be sure to correctly identify all attacks in the traces.

Honeypots are tools that solely focus on attacks. The attacks captured are real-world attacks and can be known or unknown attacks. However, like for real-world production traces, it is difficult for the evaluator to know when a specific attack begins or ends. Moreover, the evaluator cannot predict the attacks that would be captured by the honeypots. Thus, complete coverage of attacks is a challenge.

Testbed environments are the most appropriate methods to evaluate services and security products. Those environments use different executable workloads on a large virtual network. Testbed environments apply homegrown scripts, manual generation of activity and attacks and exploit databases at a large scale. Thus, this method is proficient in every type of evaluation. The main drawbacks of that method are the resource costs and manpower needed to set up, maintain and use the environment.

Publicly available traces are the referential method to compare security products. They are records of large scale experiment on testbed environments. It is affordable method for the scientific community to have a high quality evaluation of their targets. The goal of publicly available traces is to produce a large dataset to others evaluators that might not have the resources to create that dataset. It was designed for the test of the processing capacity of services and security products. Moreover, the attacks in the traces are clearly identified and the ground truth is reliable. In consequence they are one of the most used evaluation tool of the scientific community with some traces being famously used for decades like DARPA / KDD 99.

However, publicly available traces present different issues. As for real-world production traces it is not adapted to the evaluation of services and out-of-band security products that requires an exact match between the evaluation target and the content of the traces. But beyond that issue, publicly available traces are quickly outdated and the framework of the experiment behind them are flawed. [McHugh 2000] was the first to propose a critical analysis of the experimental framework of DARPA. Since then multiple new datasets were proposed and employed by the community while taking into consideration of the critics raised by McHugh: NSL-KDD[Brown et al. 2009], DEFCON[Nehinbe 2009], CAIDA[Yavanoglu and Aydos 2017], LBNL[Nechaev et al. 2004], etc. The framework and record process of those datasets were in turn also criticized by the community and some offered new frameworks ([Bhuyan et al. 2015], [Moustafa and Slay 2016], [Gharib et al. 2016], [Sharafaldin et al. 2018], etc.). However, those datasets faced the same issues: benchmark datasets need to be constantly updated due to the rapid evolution of malwares and attack strategies. The effort of the community to improve the evaluation methods of security products has been to either offer frameworks with criteria on how to produce quality evaluation datasets on testbed environments[Gharib et al. 2016], or to propose tools to generate synthetic data with a realistic statistical distribution[Vasilomanolakis et al. 2016][Shiravi et al. 2012].

Those efforts improve the quality of the traces publicly available to the community but depend on testbed environments for their capture and must be continuously updated. They do not tackle the issue of the cost that such an environment require. We want to look into the different kinds of network environments to identify virtualization tools that can reduce that cost.

2.3 Virtual network infrastructures

As previously described, a lot of different tools exist to evaluate services and security products. Among them, executable workloads require an isolated and controlled network environment to generate evaluation data without outside interferences or leaks. There are two main methods to create the network structure of such an environment: a physical network or a virtual network. A physical network build solely for testing is expensive and lack flexibility. Changing the topology of a network for every evaluation is costly. A more realistic solution is to use a virtual network. However, the term virtual network is used to describe several types of networks with each a different meaning and definition.

A general definition of a virtual network is the following:

> *In essence, a Virtual Network, in its elementary form, consists of a group of Virtual Nodes interconnected via dedicated Virtual Links.[Houidi et al. 2009]*

A virtual node is the virtual equivalent of every network equipment and machines involved in a network: routers, switch, end-points machines, servers, etc. A virtual link is a link that connects the virtual network interfaces of the virtual nodes. A single virtual link can stretch over several physical links or connect two virtual interfaces on the same server.

Among the products called network simulators are simulators based on a mathematical representation of the behavior of network equipments. It is not the kind of virtual network we consider as network virtualization. A virtual network is a network where at least part of its components is virtualized, in which a software version of the components replaces hardware components. According to if the endpoint nodes of the virtual network are simulated or not, we can classify the virtual networks into two types: network simulation and network emulation.

There is no clear consensus on the difference between network simulation and network emulation. We consider virtualized networks like overlay networks as belonging to network emulation and simulated network as belonging to network simulation.

2.3.1 Network emulation

We call network emulation a virtual network created on top of one or several physical networks with various features and capacities. A virtual network emulator virtualizes

the links and the nodes in between the links of the physical networks and creates a virtual network that possesses the features and capabilities of the physical networks. The topology of that virtual network will be independent of the topologies of the physical networks. And thus, it can be configured in an entirely different way from the physical networks to comply with the needs of the user for that virtual network. Several virtual networks can coexist on top of one or several physical networks. They can be quickly redesigned to fit different needs. The goal of virtual network emulation is to aggregate resources from multiple physical networks and offer end-to-end services to end users. It is also possible for the virtual network to serve as the basis of another virtual network created wholly or partly on top of it as illustrated in Figure 4.4 extracted from a survey on network virtualization[Chowdhury and Boutaba 2010].

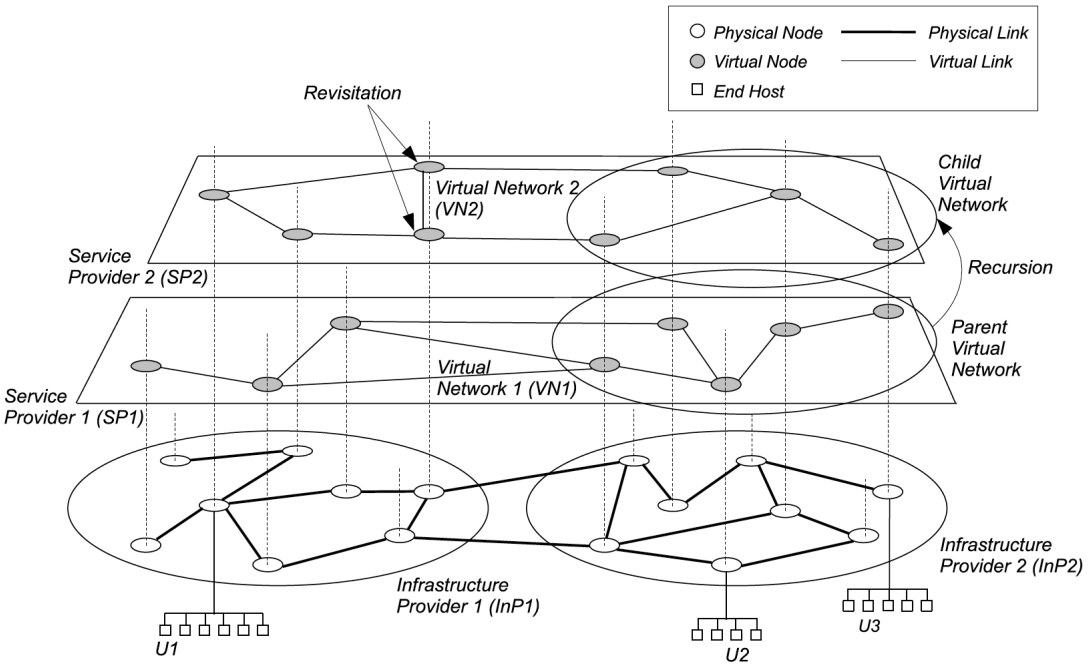

Figure 2.2 – Representation of network virtualization architecture in [Chowdhury and Boutaba 2010]

In Figure 4.4, we have two infrastructure Providers (InP1 and InP2) who provide the physical network that they manage to a Service Provider (SP1). SP1 uses the two networks to create his virtual network (VN1), which allows user U1, connected to the first network of InP1, and user U2 connected to the second network of InP2, to communicate. The other Service Provider (SP2) uses the physical network of InP1 and part of the virtual network VN1, created by SP1, to create its virtual network (VN2). U1, U2, and U3 can choose either VN1 or VN2 to communicate. The two virtual networks VN1 and VN2 are independent and do not interfere with each other.

Network emulation requires a physical network to sustain its virtual network and only offers to virtualize the links and nodes in between. To test network security products we chose not to use a physical network because it would be too costly. Furthermore,

even if we constructed a physical network to support our virtual network, we still need active endpoint nodes that create the activity and attacks required for the tests. A virtual network set up by a network emulator does not virtualize the endpoints of a network. Endpoints like physical machines or virtual machines can be connected to the virtual network but handling those endpoints is beyond the scope of the emulator.

2.3.2 Network Simulation

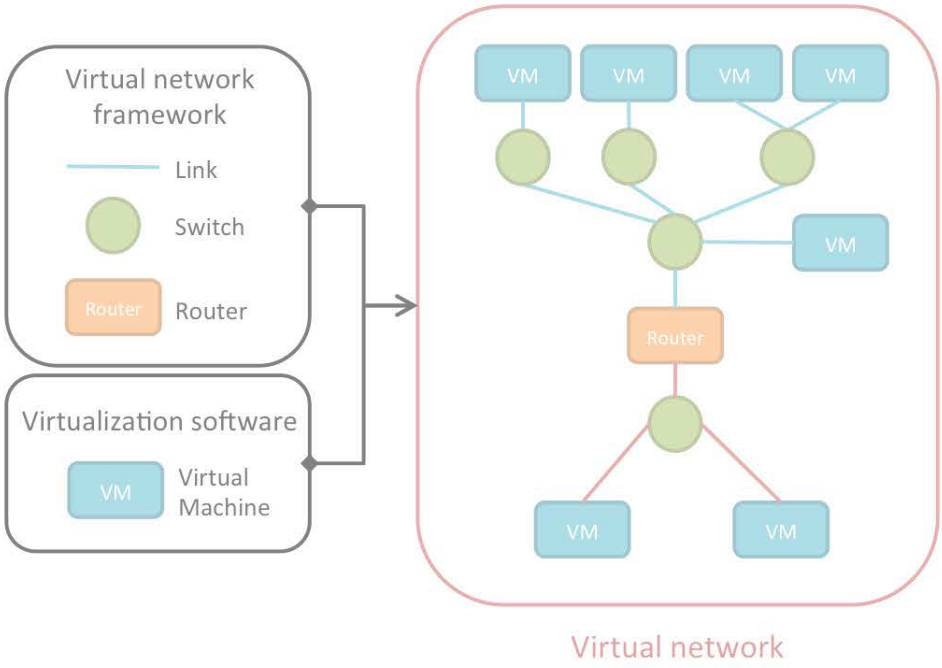

Figure 2.3 – Working principle of a network simulator

Network simulation entails the virtualization of links and in-between nodes like network emulation, but it also virtualizes endpoint nodes. It is more relevant for testing and experiments than network emulation. The most common way to create a virtual network in network simulation is to create virtual machines as endpoints and connect them with virtual links and virtualized network equipment like switches and routers.

Figure 4.5 shows the general principle of how a network simulator commonly works. A network simulator uses two programs to create a virtual network:

- A virtualization software: it creates and manages Virtual Machines (VMs) –the endpoints of our virtual network (example: VirtualBox)
- A virtual network framework: it creates network specific in-between nodes (switch and routers) and virtual links between the virtual interfaces of all nodes (example: VDE[Davoli 2005])

In most cases, it also includes a management component to quickly set up and configure the nodes of the network and spare the user from having to deal with all the compatibility details between the program managing the virtual machines and the program managing the network elements.

2.3.3 Virtual Machines

The virtualization software is the main distinguishing factor for network simulators available on the market. The program used by the simulator has particularities that result in various types of VM and types of network simulators. In general, we can divide the VMs into two main types: the system VMs and the process VMs. These two types of VMs were defined by James E. Smith and Ravi Nair as follows:

> "A *process* VM is a virtual platform that executes an individual process. This type of VM exists solely to support the process; it is created when the process is created and terminates when the process terminates. In contrast, a *system* VM provides a complete, persistent system environment that supports an operating system along with its many user processes. It provides the guest operating system with access to virtual hardware resources, including networking, I/O, and perhaps a graphical user interface along with a processor and memory."[Smith and Nair 2005]

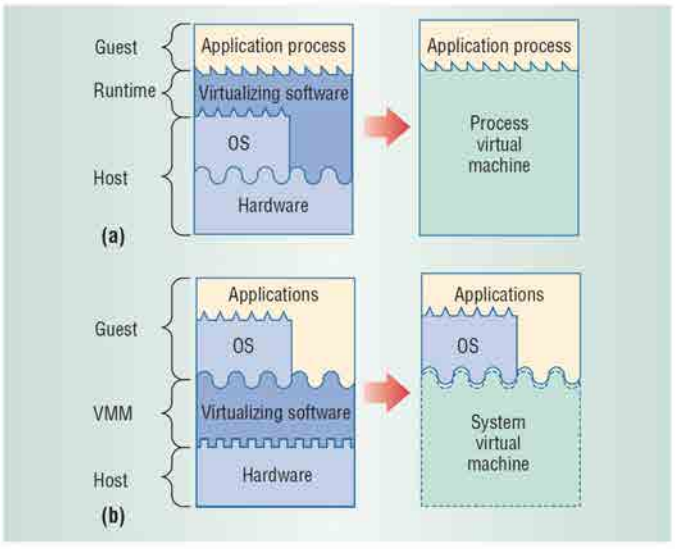

Figure 3. Process and system VMs. (a) In a process VM, virtualizing software translates a set of OS and user-level instructions composing one platform to those of another. (b) In a system VM, virtualizing software translates the ISA used by one hardware platform to that of another.

Figure 2.4 – Description of process VM and system VM from [Smith and Nair 2005]

Figure 2.4 illustrates several differences between process VM and system VM. A system VM virtualization software virtualizes the hardware of a machine and lets the user install an OS on that virtualized hardware. An example of system VM virtualization software is VirtualBox. In a process VM virtualization software, the user can only run processes on the virtualized OS forced on by the virtualization software. The imposed virtualized OS is dependent on the method used by the virtualization software. An example of process VM virtualization software is Wine. Containers are also process VMs.

A user using a system VM virtualization software can choose the OS to install, but the user needs to configure it himself in most cases. Moreover, the process can be lengthy and the launch of dozens of system VMs at the same time costs a lot in term of computational resources. A process VM is designed to start quickly and does not need much configuration effort. Moreover, the resources required for a process VM are a lot cheaper. Scalability is also one of the main features of process VMs, making them interesting for building significant virtual networks. The drawback of process VMs is that virtualization softwares use different methods to simulate the OS (clonable kernel, bundles of commonly used libraries, etc.) and those methods limit the capabilities of the process VM.

The method employed by the virtualization software to create a VM can further divide the process VMs and system VMs into more types of process VMs and system VMs. Each of these methods has a different approach to tackle the virtualization challenge peculiar to either system VMs – intercept the system call of guest OS – or process VMs – virtualizing the guest OS. Different methods create different results, and network simulators then choose different virtualizing software according to the features those methods provide. It leads to various network simulators that do not have the same purpose.

For example, Hynesim handles system VMs and relies on virtualizing softwares such as VirtualBox, QEMU/KVM, LXC, and VMware. Meanwhile, IMUNES creates virtual networks with process VMs using the method described in[Zec 2003]. Another example of process VM network simulator is Mininet. Mininet uses namespaces to create different VMs and can easily create more than a hundred VMs in a virtual network in under a minute on an average computer. However, because Mininet uses namespaces, all the created VMs use a Linux kernel, and they share the same file system. Not only is there no isolation between the namespaces but it also creates problems for services as all services share the same configuration files.

Overall, network simulators using system VMs can create very realistic virtual networks with a large variety of virtual hosts that are not dependent on the OS of the host machine. However, that virtual network consumes many resources, and each virtual host needs a specific configuration. Network simulators using process VMs can create a massive number of lightweight VMs rapidly with different topologies. The virtual hosts do not require an individual configuration. They can use the programs installed on the host system and possess a file system where any change is discarded at the end of the simulation. However, on the downside, the virtual host OS depends on

the network simulator, and the virtual machines can do limited actions on the virtual networks.

In a nutshell, a system VM virtual network allows diversity but is resource expensive, while a process VM virtual network is easy to create and less costly, but the virtual hosts' capacities are limited. Table 2.8 presents a classification of a few existing network simulators:

System network simulator	Process network simulator
Cloonix[Peach et al. 2016]	Marionnet[Loddo and Saiu 2008]
Hynesim[Prigent et al. 2005]	Mininet[De Oliveira et al. 2014]
GNS3[Welsh 2013]	MLN[Begnum 2006]
Unified Networking Lab[Vlasyuk et al. 2016]	Netkit[Rimondini 2007]
	IMUNES[Puljiz and Mikuc 2006]
	CORE[Ahrenholz et al. 2008]

Table 2.8 – **Examples of network simulators**

At this point, choosing a network simulator requires selecting the proper trade-off: either simulating a complete network capable of diversity, but at a high cost, or simulating a network quickly and at a larger scale, but with restrictions on what the virtual hosts can do. Most virtual networks used in the evaluation of security products and services, like testbed environments, are built by network simulator using system VMs or an overlay network connected to system VMs. Both solutions are expensive in resources but necessary to execute the workload drivers or script that generate the evaluation data.

2.4 Conclusion

The evaluation of services and security products consists in the verification of several heterogenous properties (compliance with the specifications, processing capacity, resilience, policy accuracy, etc.). Those properties are verified with different methods and tools. The role of the evaluator is to select compatible tools together, compose them on a network infrastructure and orchestrate them to work together to obtain a reliable evaluation of the target. Such tasks consume time and resources, especially to evaluate all properties of services and security products.

So far, only testbed environments can test all properties of services and security products. They can deploy at a large scale homegrown scripts or involve a large number of evaluator to manually generate activity. Contrary to real-world traces, the evaluator has full knowledge of the data produced. However, the resource and time needed to set up and maintain such an environment are high, especially when the scale is significant. One of the reasons for that cost is the virtual infrastructure of testbed environments. Testbed environments generally relies on system VMs. Such VMs properly support the application and programs generating data.

Other virtual networks that relies on process VMs cannot support those programs. Those virtual networks are a lot more cost efficient at a large scale. The only thing limiting their use is the lack of method to generate reliable evaluation data with the available resources. The goal of this thesis is to propose an alternative method to effectively reduce the effective costs of virtual networks in an evaluation context. We aim to propose a new method of generation of data that is independent of the virtual infrastructure and can function on less expensive infrastructure like a virtual network using process VMs.

CHAPTER 3

Formal Model of Simulation

Looking at the current state of the art on evaluation tools and virtualization techniques, we realized that one of the limitations for a commonly used and broad evaluation tool is the cost and human resources of the solution that can cover all the aspects of an evaluation. Rather than working on improving the virtualization tools, we decided to take another approach on the issue.

From the observation of testbed environments, we devise a method to produce evaluation data with a lower infrastructure requirement. By looking at the advantages of the different evaluation tools, we also identify five properties that an ideal evaluation method should have. An ideal evaluation tool is an affordable tool that can be : reproducible, realistic, adaptable, accurate and scalable. Then, we elaborate a formal model of our approach with which we could express those requirements.

A formal representation of our model allows us to present a generic approach of our intuition that is not restrained to a single implementation. We are conscious that network simulators, although they can belong to the same categories, do not use similar simulation techniques. The support of different network infrastructure can give to our model different properties that may interest evaluators. In the same fashion, implementation choices can also impact the properties and capacity of our model. The formal model present a generic model that allows for different implementation approaches. Moreover, the formalisation of the properties of our model provide a concrete verification of those properties. By clearly defining the properties of our model, we can devise tests to verify that the implementation is in accordance with the model.

In our model, we define several core concepts: elementary actions, data generating functions, scenarios, and scripts. After those definitions and the presentation of our simulation model, we deduce formal requirements on our model according to the five ideal criteria. In short, we added several properties to our model to ensure to respect as closely as possible the criteria of an ideal method.

3.1 Observations & Intuition

As we previously mentioned, testbed environments are the current most adequate solution in the evaluation of security tools and services. However, the cost of setting up, maintaining, and using those environments are too expensive for most structures. One of the reasons for that cost is the network infrastructure that can be composed of physical machines or system VMs. Evaluators must use machines that can support a wide range of application and programs and lighter VMs like process VMs do not support those applications.

We propose to change the way to produce evaluation data on a simulation network infrastructure. Rather than deploying a machine that runs all the real-life programs (browser, pdf readers, mail clients, office programs, etc.), we have a single simulation program that can reproduce the data of those programs. We do not need to have an actual browser making requests in VMs to generate traffic for a network-based security product or a web client. We can run a program that does not require a GUI (contrary to a browser) and produces requests similar to a browser. As long as the generated data is accepted the same way by the services, there is no need for the real-world programs and applications.

Thus, from that idea, we develop a method to reproduce model data that can be accepted by services. We present that method and our concepts in Section 3.2.

3.1.1 Our requirements for an evaluation data method

Our ambition is to propose an evaluation data method that can incorporate most of the strengths of the existing while leaving their weaknesses.

To accomplish that, we highlight five goals, or requirements for an ideal method, that represent the most relevant aspect of the current state of the art.

Ideally, an evaluation data method must be:

- **Customizable**: one of the main strengths of executable workloads is that the evaluators can customize the generated activity to match their needs. It allows them to use the same tools to test a security product with different metrics. Meanwhile traces only allow the test of one scenario per trace. We want our method to be able to offer a wide variety of parameters to modulate and produce evaluation data according to the needs of the evaluator. We also want our method to avoid the usual issue of the freshness of traces.
- **Reproducible**: one of the biggest weakness of executable workloads is that it is often time-consuming to restore the victim environment to its previous state, especially in a sophisticated setting like a testbed. A significant advantage of traces is that it is easy to feed to a security product. In consequence, traces are used as a standard for the community. We want a method that can provide evaluation data with little to no overhead to restore an evaluation environment.

- **Realistic**: real-world production traces and honeypots allow evaluators to confront security products and services with real-life data and attacks. With a realistic activity model, manual generation can produce a close to real life evaluation data. Its capture, after deployment at a large scale in a testbed environment, are the publicly available traces. The community considers that method of generating data as sufficiently realistic to be used as a reference. As such, we want our ideal method to be able to provide realistic evaluation data to the level of publicly available traces provided with a realistic activity model. We also want our ideal method to avoid the same issues with privacy as real-world production traces.

- **Accurate**: one of the main weakness of traces obtained from real-world production and high-level of interaction honeypots is the difficulty to generate the ground truth of traces. The evaluator cannot guarantee the accuracy and correctness of the generated ground truth. Isolating attacks from one another or legitimate traffic is challenging. An ideal evaluation data method should have full knowledge of the activity generated.

- **Scalable**: large-scale testbeds allow the evaluators to generate complex and realistic traffic despite the large cost of doing so. Publicly available traces aim to provide that complex traffic at structures that cannot afford that high cost. We want our method to be able to generate the evaluation data of large scale networks while offering the possibility to choose the size of this network.

These five criteria are the goals we fix for our evaluation data method. Even if our method ends up not being as ideal, we want our approach to respect as much as possible the requirements of an ideal method.

We also want to be able to provide evaluation data for live testings and offline testings. As such, we want our method to be close to executable workloads. Indeed, this type of evaluation data method that allows live testings and generates traces for offline testings.

3.2 Concepts and Definitions

The evaluation data method we propose is similar to workload drivers but with key differences. We execute small programs to generate evaluation data on the hosts of the network environment. However, those programs do not execute tasks known to exercise specific system resources, like SPEC CPU2000, or a series of empty traffic targeting a specific protocol, like httpbench. Those programs replay the data of traces corresponding to small actions on virtual hosts. The host's actions coordinate with the actions of the other hosts in an activity scenario. The generated data are not identical to the traces but preserve the relevant features, in accordance with the needs of the evaluator.

We produce simulation data identically to the model data up to the limit of the need of the evaluator. For example, if the evaluator is only interested in the data up to the transport layer, we do not need to produce realistic application layer data.

We explain our model further in this section. We define concepts on which we build a formal description of our methodology.

3.2.1 Elementary actions

We call *Elementary action* (A_H^S) a sequence of interactions that represents an action between two actors of the activity. Those actors are either a *Host* – a source of generated data (H) – or a *Service* – a set of functionalities available to a *Host* (S). A *Service* can be an external server or an internal service.

$$A_H^S = (I_{H,0}^S, \ldots, I_{H,n}^S) \quad where \quad I_{H,i}^S = interaction\ i\ between\ the\ host\ H\ and\ the\ service\ S$$

To simplify the notation, we note A_H^S as A_H or, if there is no ambiguity on the *Host*, A.

The goal of *Elementary actions* is to divide the activity we simulate in individual actions that correspond to an entry of the ground truth, such as "connection to the web interface of a webmail server". The ground truth is an exact representation of the activity generated. So a finer set of *Elementary actions* for an activity means a finer representation of the simulated activity and a finer control of the activity model for the evaluator. Roughly translated, even if the model does not forbid it, an *Elementary action* is not meant to be a large set of interactions like "a day of activity of user U". An *Elementary action* intends to represent a small action like "connection to service S" or "adding an entry to service S'".

In our methodology, we distinguish real activity ($\mathcal{R}$) – not issued from our simulation method – and simulated activity ($\mathcal{S}$) – issued from our simulation method.

For each *Elementary action*, we acquire *Model data* that are the captured data of the execution of this *Elementary action* during real activity ($A^{\mathcal{R}}$). *Model data* take different forms (traces, logs, values, etc.) according to the nature of the data the evaluation target can handle.

$$d^{model} = \{Capture(A_0^{\mathcal{R}}),\ \ldots, Capture(A_n^{\mathcal{R}})\}$$

Furthermore, the evaluator can classify the *Model data*. The evaluator relies on that classification to help label the resulting *Simulation data* and create a labeled ground truth. However, the evaluator is in charge of deciding a classification as it can change according to the need of the evaluator, or to the target of the evaluation. For example, the evaluator can create two classes of *Model data* to represent malicious activity and benign activity, respectively.

$$d^{malicious} = \{Capture(A_0^{\mathcal{R}}), \ldots, Capture(A_m^{\mathcal{R}})\} \quad where\ \forall i \in [\![0,\ m]\!],$$
$$A_i\ is\ a\ malicious\ elementary\ action$$

$$d^{benign} = \{Capture(A_0^{\mathcal{R}}), \ldots, Capture(A_b^{\mathcal{R}})\} \quad where\ \forall i \in [\![0,\ b]\!],$$
$$A_i\ is\ a\ benign\ elementary\ action$$

In other contexts, the evaluator can define other classes. For the evaluation of administration tools of a network, the actors to consider are different (security: attacker/user, administration: admin/user/client) and the evaluator will have to define classes of data accordingly.

After capturing *Model data* for every *Elementary action* relevant for the evaluation, the resulting set of *Model data* is then given as an input to a *Data generating function*.

3.2.2 Data generating function

We simply define a *Data generating function* (f) as a function that creates *Simulation data* from *Model data*. *Simulation data* ($d^{simulation}$) is the execution of an *Elementary action* (A) during a simulated activity.

A *Data generating function* (f) takes two inputs: *Model data* (d^{model}) and a set of *Elementary action parameters* (p^A). We define later the *Elementary action parameters*. To simplify the notation, when the set of *Elementary action parameters* is empty – $f(d^{model}, \emptyset)$ – we note it $f(d^{model})$.

Our definition of *Data generating function* does not includes requirements on the output, the *Simulation data*. We express our demands for *Simulation data* as *Equivalence*.

We call *Equivalence* ($\sim$) the fact that two activity data (*Model data* or *Simulation data*) have the same properties.

$$d^{activity} \in \{d^{model}, d^{simulation}\}$$
$$d^{activity} \sim d'^{activity} \iff Properties(d^{activity}) = Properties(d'^{activity})$$

It is important to specify that the only identical properties of two equivalent data are the properties of interest to the evaluator. So if not all properties are necessarily identical, it means that two equivalent data are not necessarily identical. For example, if the evaluator has interest only in the size of the packets, two packets with the same size but different content will be equivalent.

Property 1 *Two equivalent data are not necessarily identical:*

$$\forall d, d' \in \mathcal{D}, \quad d \sim d' \quad \not\Rightarrow \quad d = d'$$

The fact that two equivalent data are not necessarily identical is especially true when we consider *Data generating function* that generates data with time-sensitive fields (for example timestamp or random numbers). However, we do not consider the time as an input of the *Data generating function* because we are not concern by the strict identicallity of the data but by its equivalence. And the equivalence of the data is not impacted by the time.

The properties of the data are of different forms: acknowledgement of the data by the *Service*, size of sent packets, value of a measure, etc. They represent the level of realism chosen by the evaluator. Different *Data reproducing functions* can conserve different properties from the same *Model data*. The *Simulation data* produced by different *Data generating functions* are not necessarily equivalent even if they used the same *Model data*.

$$\left.\begin{array}{l} f(d^{model}) = d^{simulation} \\ f'(d^{model}) = d'^{\,simulation} \end{array}\right\} \quad \not\Rightarrow d^{simulation} \sim d'^{\,simulation}$$

The evaluator selects a set of *Elementary actions* to decide the granularity of control over the simulation and he chooses a *Data generating function* to reproduce the properties of the data he requires. If the *Data generating function* that produces *Simulation data* from a dataset of *Model data* cannot produce data with the same properties, it is useless for the evaluator. Thus, we define the following verification property of *Data generating functions*:

Property 2 *A* Data generating function f *is said to be* ***useful to a set of*** **Model data** $\mathcal{D}$ *if all* Simulation data *generated by* f *from any* Model data *that belong to* $\mathcal{D}$ *is equivalent to the data used as model.*

$$\forall\ d \in \mathcal{D}, f(d) \sim d \quad \Leftrightarrow f\ is\ useful\ to\ \mathcal{D}$$

The evaluator has to choose a *Data generating function* according to the properties he wants to simulate in his evaluation. The evaluator makes that choice with *Simulation parameters*.

We call *Simulation parameters* ($p^{simulation}$) a set of parameters given by the evaluator that defines the choice of a *Data generating function* f. $f^{p^{simulation}}$ is the *Data generating function* selected by the *Simulation parameters* among a pool of n functions ($\{f^0, \ldots, f^n\}$).

$$p^{simulation} = \{p_0, \ldots, p_n\} \quad / f^{p^{simulation}}(d^{model}) = d^{simulation}$$

The evaluator selects a *Data generating function* with *Simulation parameters* for the properties preserved by Property 2. The evaluator also provides the *Data generating function* with additional parameters called *Elementary action parameters* (p^A).

Elementary action parameters allow the evaluator to modify the behavior of the *Data generating function*. It can be to match a larger dataset of *Model data* or to provide a finer control.

We call *Elementary action parameters* (p^A) associated to an *Elementary action* A a set of parameters provided to a *Data generating function* to produce *Simulation data* that can impact positively the usefulness of the *Data generating function*.

$$p^A = \{p_0^A, \ldots, p_n^A\} \quad / \quad \left.\begin{array}{l} f(d_A, p^A) \; is \; useful \; to \; \mathcal{D} \\ f(d_A, 0) \; is \; useful \; to \; \mathcal{D}' \end{array}\right\} \quad \Rightarrow \mathcal{D}' \subseteq \mathcal{D}$$

For example, we take a *Data generating function* that conserves the property "acknowledgement of the data by the *Service*" of traces given to it. That function will change time-sensitive information on the input data, like a cookie or a session ID. However, different *Services* may mark those information differently. A service S_1 might mark the session ID with the tag "&session_id=" while a service S_2 will mark it with the tag "&_session=". Some services might also have other additional change in their interactions that other services do not have. To avoid having a different *Data generating function* for every service due to those insignificant differences, we want to parametrize the transformations applied by the *Data generating function*.

Moreover, the evaluator might want to produce *Simulation data* from the same *Model data* but with different results. For example, the evaluator wants to reproduce the *Elementary action* "connect to a webmail" with the conservation of the property "acknowledgement of the data by the *Service*". However, left as it is, the *Simulation data* always present the same credentials. The evaluator may want to simulate connection to the webmail with different IDs without having to have a different *Model data* for every set of credentials. It is necessary to be able to parameter the function to make small modifications rather than having a lot of *Model data* for the same *Elementary action*.

To avoid having to make a *Data generating function* that conserves the property "acknowledgement of the data by the *Service*" for each *Service* of the simulation, or having a lot of different data as model for a same *Elementary action*, we provide parameters to *Data generating functions* to modify the *Simulation data* produced, while conserving the properties of the data.

3.2.3 Scenario and scripts

A *Script* is the representation of a realistic behavior of a *Host*. We define a *Script* as a sequence of *Elementary actions* coupled with *Elementary action parameters*.

$$Script_H = ([A_{0,H}, p^{A_0}], \ldots, ([A_{n,H}, p^{A_n}])$$

A *Script* ($Script_H$) is defined for each individual *Host*. It describes the activity it must generate during the simulation. The set of *Scripts* is called the *Scenario* (Sce) of the simulation.

$$Sce = \{Script_{H_0}, \ldots, Script_{H_n}\}$$

A *Script* can be represented as a graph of actions, as illustrated in Figure3.1.

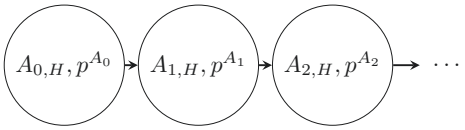

Figure 3.1 – Example of a *Script*

Finally, the data exchanged by the program that controls the simulation and the *Host* that runs a *Data generating function* are the *Control data* ($d^{control}$) and are essentially the ground truth of the simulation. The compilation of the *Control data* informs us of all the actions taken during the simulated activity.

3.3 Model and concept application

We presented the basic concepts of our model. Now we can apply those concepts to explain our model. We also translate criteria for an ideal method into properties of our model and constraints on our implementation.

3.3.1 The model

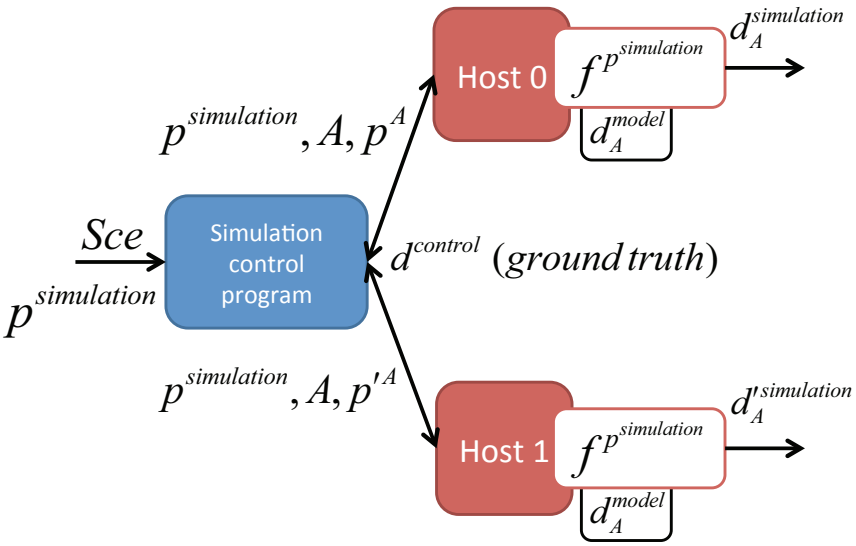

Figure 3.2 – Generation of simulated activity from short traces

Figure 3.2 is a representation of our model. The evaluator provides the simulation control program with the *Simulation parameters* ($p^{simulation}$) and the *Scenario* (*Sce*):

$$Sce = \{Script_{H_0}, Script_{H_1}\} = \{([A, p^A], \ldots), ([A, p'^A], \ldots)\}$$

The simulation control program interprets the *Scenario* and the *Simulation parameters* and deduces the number of *Hosts* in the current simulation. It instructs the *Hosts* H_0 and H_1 to reproduce the *Elementary action* (A) with the parameters $p^{simulation}$ and p^A. Then, each *Host* retrieves the *Model data* associated to the *Elementary action* (d_A^{model}) and executes the *Data generating function* (f) selected by the *Simulation parameters*. That function produces *Simulation data* representing the *Elementary action* $(d_A^{simulation})$, which are sent to a *Service*. The use of different *Elementary action parameters* by H_0 and H_1 results in the generation of different *Simulation data* even when the *Data generating function* and the *Model Data* are the same:

$$\left.\begin{aligned} d_A^{simulation} &= f^{p^{simulation}}(d_A^{model}, p^A) \\ d'^{simulation}_A &= f^{p^{simulation}}(d_A^{model}, p'^A) \end{aligned}\right\} \quad \not\Rightarrow d_A^{simulation} = d'^{simulation}_A$$

After the *Hosts* inform the simulation control program that they finished simulating the *Elementary action A*, they await the next simulation orders from the simulation control program.

The model we presented is the situation where all the *Hosts* are simulated and the *Services* are real services. If a subset of the *Hosts* also acted as *Services*, they could also initiate the generation of *Simulation data* according to requests received from other *Hosts* in the form of other *Simulation data*.

In our model, the processing of the *Control data* of *Hosts* simulated by our model makes the ground truth of the simulation. Therefore, we do not process data from *Hosts* unrelated to the simulation (external hosts connected to the simulation). The evaluators must incorporate those elements to their ground truth.

Lastly, we must present one of the significant issues of our model: the parameterization of the *Elementary actions*. The parameterization is the addition of *Elementary action parameters* to extend the scope and variability of the *Data generating function* while still preserving various data properties. However, the higher level the preserved properties are, the more complex the reproduction of *Elementary actions* becomes. Therefore, designing a *Data generating function* for a highly realistic simulation, where not only packet size is preserved but also data acknowledgment, requires to consider three main aspects:

- **types**: identification and generation of short-lived data like tokens, identifiers of session, etc.
- **semantics**: modification of inputs with a high semantic value in the *Model data*: credentials, mail selection, mail content, etc.
- **scalability**: a large scale execution of the *Data generation function* can have consequences on the previous aspects and requires additional changes (e.g., creation of multiple user accounts in the *Service* database).

These three aspects are integrated to the *Elementary action parameters*. However, a few in-depth issues still require further consideration and development in order to

elaborate a model able to adapt to various test situations without the intervention of the evaluator. The typing issue can be solved with methods based on machine learning, but others may require specific methodologies according to the context of the evaluation. For example, in the case of the reproduction of a real-life network, the semantic and scalability issues can be solved with analysis of an extended *Model data* acquisition period. The evaluator can identify and highlight in the *Model data* inputs with a high semantic value for the simulation.

3.3.2 Ideal criteria: application of our concepts

An ideal evaluation data method must be *reproducible*, *realistic*, *customizable*, *accurate* and *scalable*. We want our method for evaluation data to respect as many of those requirements as possible. In this subsection, we discuss how to translate the requirements into properties of the model of our method or, in case it is not possible, into implementation requirements.

Reproducibility

Reproducibility concerns two different aspects: an experiment realized several times in the same condition must produce the same results, and a previous experiment should not impact a new experiment – the ability to restore the simulation environment to a starting state.

The first requirement of reproducibility means that when reproducing a similar event in the same context, our simulation must provide a similar result. We translate it in a property on *Data generating functions*:

Property 3 (Reproducibility property of Data generating function) *A* Data generating function *f is said to be* ***reproducible*** *if, for any* Model data*, the resulting* Simulation data *generated at every instant is equivalent to any* Simulation data *previously produced with the same input data.*

$$\forall d \in \mathcal{D},\ \forall t' \neq t \quad \left.\begin{array}{l} at\ instant\ t, f(d) = d^{simulation} \\ at\ instant\ t', f(d) = d'^{simulation} \end{array}\right\}$$
$$f\ is\ reproducible \ \Leftrightarrow\ d^{simulation} \sim d'^{simulation}$$

The second requirement of reproducibility is not a requirement on our model but on the virtual infrastructure of our simulation. The implementation of our model must allow the creation networks in similar conditions without any lasting impact from any previous use of the simulation. This requirement will impact the choice we make for the network support of our simulation implementation.

Realism

The realism of our simulation depends on two factors:

- The *Data generating function* and *Model data*: the *Simulation data* produced is only as realistic as the *Data generating function* and the input data allow. The verification property of the Data generating function (cf. Property 2) ascertains that the *Simulation data* produced is as realistic as the *Model data* used as input of the *Data generating function*. Thus, for *Simulation data* to be realistic, the *Model data* must also be considered realistic for the same criteria of realism. Moreover, the *Data generating function* must preserve the properties of the *Model data* that one considered as realistic.
- The *Scenario*: the activity our simulation produce is only as realistic as the *Scenario*. At its core, the *Scenario* is the activity model of the evaluator for the simulation. A realistic activity model with realistic *Model data* will result in a realistic activity. We want our model to verify that property.

Property 4 (Verification property of the Scenario) *The* Model data *generated by a real activity according to a* Scenario *must be equivalent to the* Simulation data *generated by a simulated activity of the same* Scenario.

In short, our simulation model can generate an activity according to a *Scenario* with the guarantee that the properties of the *Simulation data* are as realistic as the input data. As to which properties are guaranteed, it will depend on the *Data generating function* chosen by the evaluator. In a more concrete example, if our method makes a simulation of a network activity with a *Data generating function* that preserves the packet size of the model data, we can guarantee that the *Simulation data* produced will present a volumetric network activity as realistic as the *Scenario* of the simulation and the *Model data*.

Adaptability

The *Scenario* allows the evaluator to generate the activity of different activity models. With a large variety of *Elementary actions*, the evaluator can create complex *Scenarios* and obtain realistic *Hosts*' behavior during the simulation. Moreover, the smaller the *Elementary actions* are, the more precisely the evaluator can control the simulation and create *Scenarios* adapted to its needs.

The *Elementary action parameters* can preserve the realism of the *Simulation data* while offering another degree of customization. The evaluator can significantly improve the semantic value of the *Simulation data* by using those parameters to modify customizable inputs in the *Model data* (credentials, POST form inputs, etc.).

The variety of *Data generating functions* to select from is one strength of our model. Each *Data generating function* offers a guarantee of realism as seen in Section 3.3.2. We

can define levels of realism that correspond to the selection of different *Data generating functions*. As the realism property of our model partly depend on the *Data generating function*, a customizable *Data generating function* means an adaptable realism of the simulation.

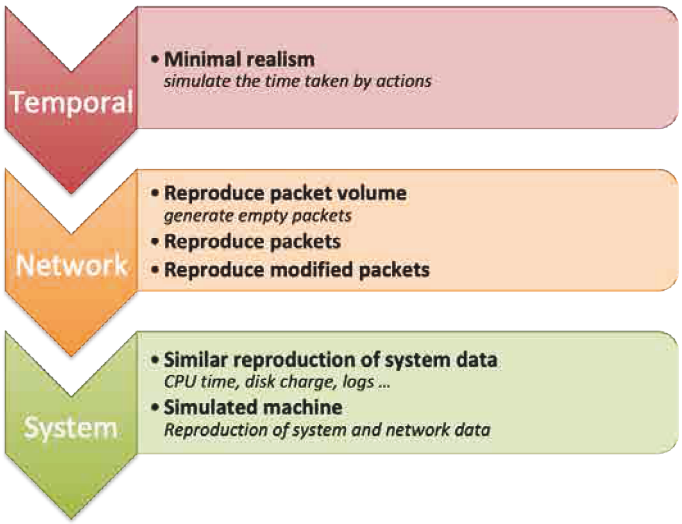

Figure 3.3 – Levels of realism

In Figure 3.3, we describe several levels of realism associated with several *Data generating functions*. We divide these levels into three main types: **Temporal**, **Network** and **System**.

The first type, **Temporal**, is the minimal level of realism possible and is useful only when the evaluator is solely interested in the elapsed time of different *Scenarios*. In this case, the simulation of activity consists of *Hosts* waiting the average time of each *Elementary action*.

The second type of levels of realism, **Network**, is chosen when an evaluator is solely interested in the simulated network traffic. We identify three levels for this type:

- **Reproduce packet volume**: if the evaluator has an interest in the network charge of the simulation, with no interest for the content. From the *Model data* of each *Elementary action*, the *Data generating function* reproduces all the packets up to the transport layer before padding the payload with random bytes to match the size of the input data's packets.

- **Reproduce packets**: if the evaluator is interested in reproducing the input data with the full payload. The *Data generating function* reproduces the packets while changing the appropriate fields (tokens, session IDs, etc.) for a correct exchange between the *Host* and the *Service*.

- **Reproduce modified packets**: if the evaluator needs more precise control than the previous **Reproduce packets** level of realism. The *Data generating function* produces the packets as described above, but *Elementary action parameters*

customize the output. For example, if the *Data generating function* reproduces the *Model data* to connect to the webmail server, the evaluator also modify the credentials used to connect to the *Service* by the *Model data*.

Finally, the third type of realism is **System**. It is the highest level of realism we describe for the simulation. With **Reproduction of system data**, we want to reproduce the system data measured on the *Hosts* so that it matches the *Model data*. For the highest degree of realism we want the *Data generating function* to reproduce the data of service applications – to provide network and system data without executing the applications. It is, in essence, a **Simulated machine**.

All those different levels of realism correspond to different *Data generating functions* that the evaluator can select for its evaluation simulation. It is also possible to imagine and add other levels of realism or *Data generating functions* with different uses. Those levels of realism represent our implementation goal for a customizable realism of our model.

The nature of the *Model data* described in each type of level of realism is different. **Temporal** uses measure values as *Model data*, **Network** have network traces, and the *Model data* of **System** are a combination of system data and network traces.

To sum it up, the evaluators can customize the simulation of our model according to:

- The scenario and network topology (size, topology, connexion to external networks, ...)
- The *Model data*: the scope of *Elementary actions* (smaller or larger number of interactions)
- The *Elementary action parameters*: customize inputs with semantic value (credentials, information fields, submission contents, etc.)
- The *Data generating function*: the evaluator can select a level of realism useful for a specific dataset or the preservation of specific properties.

Accuracy

The constitution of the ground truth is deduced from the analysis of the *Control data*. We want the *Control data* of our model to contain the information on the input of every event $(p^{simulation}, A, p^A)$, to know when the *Host* has reproduced the *Elementary action*, how long it took him to do it and what was the result (success, failure, error, etc.).

Our simulation must guarantee that information. We translate it into an implementation requirement of our model:

Property 5 (Implementation requirement) *The* Control data *of the simulation must include the following information for every entry of the* Scenario *associated to an* Elementary action*:*

- *The* Elementary action *taken by an* Host
- *The timestamp of that action*
- *The* Elementary action parameters *sent with the action*
- *The result of that* Elementary action
- *The time it took to carry it*

By combining this information with the traces used for each *Elementary action*, it is possible to label a record of the activity generated by the simulation of our model quite simply.

In our model, each *Host* only receives instructions to replay one *Elementary action* at a time. Only the simulation control program knows the full *Scenario* and knows which *Elementary action* and *Elementary action parameter* any given *Host* will replay next. Each *Host* is in a stateless situation at any given point of the simulation. We could choose to give the *Script* of the *Host* to each of them at the start of the simulation and let them deal with the execution of the *Script*. However, it would introduce the risk of having the *Hosts* act out of sync with each other. Especially when introducing elements of randomness in the *Scripts* of the *Hosts*.

With a centralization of the decision-making process of the simulation, we have a central point with full knowledge of the situation of the simulation at any given point in time, which facilitates the constitution of the ground truth of the simulation.

However, our model is not connected to the output of the tested product. Our model solely handles the data sent to the tested product. It is able to label the *Simulation data* produced. While our model also received data, it does not know if the data was correctly processed by external components. For example, it has no access to their logs. Those elements are necessary for a complete ground truth. For example, if our simulation asks an *Host* to reproduce the *Elementary action* "connect to the webmail server of the company", we will know what *Data generating function* it used, when and how long it took for the *Host* to do it and if the execution of the function went well. However, we do not have access to the log of the webmail server telling us if the connection was successful. That information is not present in the *Control data*. The evaluator would have to provide that information to the ground truth generated by the simulation. In the previous example, we assumed that we are in the case where we simulate only the clients of real-life services. If we also simulated the *Services*, then the *Control data* would also contain the information on the output of the tested product.

The scope we guarantee for the constitution of an accurate ground truth only goes as far as the elements simulated by our simulation goes. The constitution of the ground truth with the data of external elements connected to our simulation is left to the evaluator.

Scalability

Scalability is an essential issue for evaluation data. It is a property that rapidly increases the difficulty and cost of most methods to generate evaluation data. However, it is an interesting property because it allows the constitution of complex and large-scale activity close to a real-world environment. One of the interesting property of having a scalable network simulation is that the *Simulation data* generated by two *Hosts* (H_0 and H_1) is not the same thing as the *Simulation data* of a single *Host* H_0 doing the same action twice.

$$\exists\{d, f^{p^{simulation}}, p^A\}/ \quad \left.\begin{array}{l} d_{H_0}^{simulation} = f_{H_0}^{p^{simulation}}(d, p^A) \\ d_{H_1}^{simulation} = f_{H_1}^{p^{simulation}}(d, p^A) \end{array}\right\} \quad d_{H_0}^{simulation} \cup d_{H_1}^{simulation} \neq d_{H_0}^{simulation} * 2$$

In the same way, having 50 *Hosts* making one connection is not the same thing that having one *Host* making 50 connections at the same time. The impact on the tested product is also not the same, especially if that security product must follow the activity of each user separately.

Our model guarantees that if two *Hosts* use the same useful *Data generating function* on the same *Model data*, then the resulting *Simulation data* are equivalent.

Proof.

- According to Property 2:

$$f \ is \ useful \ to \ \mathcal{D}, \ if \ \forall d \in \mathcal{D}, d \sim d^{simulation} = f(d)$$

- So, if the *Hosts* H_0 and H_1 produce *Simulation data* with f then:

$$\forall\{d, p^A\}/ \quad \left.\begin{array}{l} d_{H_0}^{simulation} = f_{H_0}(d, p^A) \\ d_{H_1}^{simulation} = f_{H_1}(d, p^A) \end{array}\right\} \quad \Rightarrow \quad d_{H_0}^{simulation} \sim d \ and \ d_{H_1}^{simulation} \sim d$$

- The definition of equivalence is that two data are equivalent if and only if their properties are the same. So:

$$\left.\begin{array}{l} d_{H_0}^{simulation} \sim d \iff Properties(d_{H_0}^{simulation}) = Properties(d) \\ d_{H_1}^{simulation} \sim d \iff Properties(d_{H_1}^{simulation}) = Properties(d) \end{array}\right\}$$

$$\Rightarrow \ Properties(d_{H_0}^{simulation}) = Properties(d_{H_1}^{simulation})$$

$$and \quad Properties(d_{H_0}^{simulation}) = Properties(d_{H_1}^{simulation}) \iff d_{H_0}^{simulation} \sim d_{H_1}^{simulation}$$

- So, we have:

$$\forall\{d, p^A\}/ \quad \left.\begin{array}{l} d_{H_0}^{simulation} = f_{H_0}(d, p^A) \\ d_{H_1}^{simulation} = f_{H_1}(d, p^A) \end{array}\right\} \quad \Rightarrow \quad d_{H_0}^{simulation} \sim d_{H_1}^{simulation}$$

Moreover, according to Property 1, we know that two equivalent data are not necessarily equal to each other ($d \sim d' \quad \not\Rightarrow \quad d = d'$). Therefore, it exists a case where two *Model data* are equivalent and not equal. So:

$$\exists\{d, f^{p^{simulation}}, p^A\}/ \quad \left.\begin{array}{l} d_{H_0}^{simulation} = f_{H_0}^{p^{simulation}}(d, p^A) \\ d_{H_1}^{simulation} = f_{H_1}^{p^{simulation}}(d, p^A) \end{array}\right\}$$
$$d_{H_0}^{simulation} \sim d_{H_1}^{simulation} \; and \; d_{H_0}^{simulation} \neq d_{H_1}^{simulation}$$

So, if we produce $d_{H_0}^{simulation}$ once more on each side we obtain the following result on our model:

$$\exists\{d, f^{p^{simulation}}, p^A\}/ \quad \left.\begin{array}{l} d_{H_0}^{simulation} = f_{H_0}^{p^{simulation}}(d, p^A) \\ d_{H_1}^{simulation} = f_{H_1}^{p^{simulation}}(d, p^A) \end{array}\right\} \quad d_{H_0}^{simulation} \cup d_{H_1}^{simulation} \neq d_{H_0}^{simulation} * 2$$

■

We now have proof that one of the main interesting aspects of a scalable simulation is not in opposition to our model. With our model, we can produce the activity of n *Hosts* that would be different from doing the same activity n times simultaneously on a single *Host*.

The other aspect to consider with scalability is the network infrastructure. To achieve a scalable simulation, we must impose some requirement on the network infrastructure:

Property 6 (Implementation requirement 2) *The network infrastructure supporting the simulation model must be capable of scalability. It should:*

- *Use virtual hosts: it is not possible to have a scalable network with physical hosts so our network support must use virtual hosts.*
- *Have low setup time and resources requirements: resources consumption and the workforce required for setting up a vast network are what often prevent evaluation data methods from being scalable.*

Following the second requirement, we aim to make our simulation model work on a virtual network using process VMs.

3.4 Conclusion

Most of the resources of the system VMs are not used during the simulation on testbed environments. Lighter VMs would significantly reduce the cost and maintenance of testbeds environments, but they are unable to execute real-world production applications and programs. We solve that issue by replacing the execution of those applications

with the reproduction of their data. From that point, we defined a series of concepts to present a formal presentation of our simulation model. We developed that formal model for two reasons: to have a model independent from implementation choices and allow the proposition of several implementation approach, and to formally express properties that can be validated in a concrete fashion. In this model, we instruct virtual hosts to reproduce *Model data* with a specific level of realism (*Data generating function*). The hosts will follow a *Script* from the *Scenario* defined by the evaluator. This *Script* defines a series of Elementary actions to reproduce.

From our analysis of existing methods, we also presented five criteria of what we consider as an ideal method to generate evaluation data: reproducibility, realism, adaptability, accuracy, and scalability. From our formal model, we extracted several properties and implementation requirements. That analysis also highlighted the strengths (customizable level of realism, scalability, centralization of the decision-making process, etc.) and weaknesses (ground truth of external components, the complexity of high-level *Data generating function*, parametrization issue, etc.) of our current model.

We now have to select a network simulator platform that can fill the requirements of our method and to propose an implementation of our model. The resulting prototype will serve to validate the strengths highlighted by our model but also look for development possibilities to compensate and reduce the weaknesses of our formal model. Mainly, we can look for possible tools that can reduce the preparation and configuration burden of the evaluator.

CHAPTER 4

Simulation plateform

In this chapter, we present the implementation choices and issues we encountered in the making of a prototype for our model. The conception of our prototype followed in several steps: the selection and capture of *Model data*, the development of *Data generating functions*, and the deployment of the prototype on a simulated network. We made choices on the *Model data* to capture (what *Service* to use as a model, how to capture it, the *Elementary actions* to reproduce, etc.), on our approach for *Data generating functions*, and the network infrastructure to use. The prototype serves as an illustration of our model. Its functionalities are limited as its goal is to prove the feasibility and to gauge the potential of our method along with identifying the current issues of our model. Though the current preparatory steps of our prototype may burden the evaluator, we also propose different approaches to mitigate the issues and transform the prototype into a functional evaluation tool.

4.1 Capture of model data

The *Model data* of our model is the referential data used by *Data generating functions* to produce *Simulation data*. We define the realism of our simulation on how close the *Simulation data* are to the *Model data* in the eyes of the evaluator. The *Model data* is the measuring stick of the quality of our simulation as the *Simulation data* of our model can at most be only as realistic as the *Model data*. As such, the capture of *Model data* is an important step of the implementation of our method.

Before the actual capture of *Model data*, we have to decide on a reference network. That network will serve as a source of *Model data* and as a comparison with the simulated data. For that network, we did not have a physical network available with full control and access to capture the data. We chose a target topology that we reproduced in a virtual network with system VMs. As presented in Section 2.3.3, a virtual network with system VMs is the optimal virtual environment. However, system VMs have large resources costs. Testbed environments often use virtual network with system VMs and the publicly available traces captured on those networks serve as a reference for the research community. As such, we make the assumption that, with appropriately configured services, the data captured on such a network are realistic enough to be used as *Model data*.

4.1.1 Reference network

We use Hynesim to create the virtual reference network. It is an open-source network simulator that connects system VMs in a virtual network. It handles a variety of virtualization software like QEMU, VirtualBox, and VMware.

We chose to reproduce the network of a small company with typical services. In that network:

- most services are external: webmail, website, data storage, etc. We assume that a small company would mostly use cloud services
- some services are internal: printer, DNS, DHCP, LDAP. We use the DNS server to refer for the local resolution of the domain name of the company (mycompany.com) toward the external services.
- the company has a single connection point to services, such as a regular Internet Box.

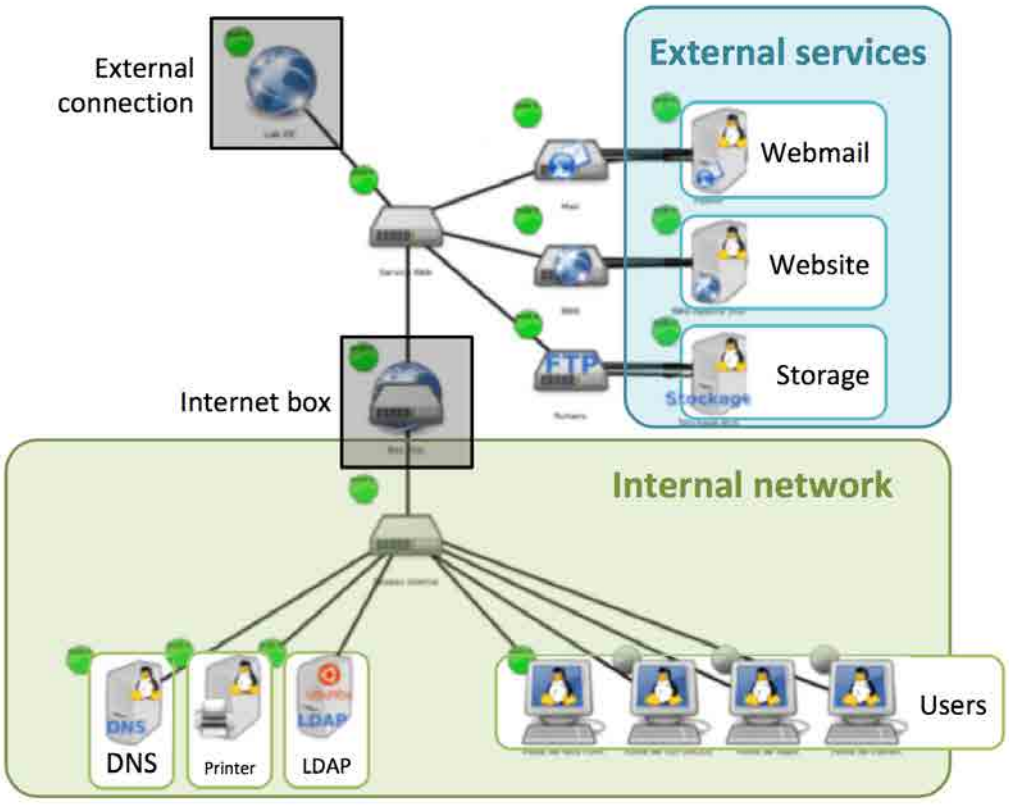

Figure 4.1 – Topology of our reference network

Figure 4.1 shows the reference network created on Hynesim. For each type of external service, we instantiated several servers on VMs. For example, for the website server we set up an Apache server VM, a NGIX server VM, and a LiteSpeed server

VM. For the webmail server, we create a VM for a Postfix server, a UbuntuServer Mail, a Zimbra webmail, and a Sendmail server each. The goal of that approach is to provide a variety of sources for the *Model data*.

In a further development stage of the prototype, we mainly focused on the *Model data* of a Postfix server with the addition of the open-source Roundcube webmail layer on top of it. We focused on a single type of service because of time constraints. However, one of the potential development axes of our prototype is to develop a database of *Model data*. One of the limitations of the current model is the amount of preparation work that falls upon the shoulders of the evaluator. Indeed, the evaluator must capture his own *Model data*, which means instantiating a reference network. As such, to transform the current prototype in a practical tool, we want the prototype to provide *Model data* from a wide range of services. With such a practice, the evaluator would not have to configure and install services on system VMs to generate *Model data*. The evaluator could therefore directly use a database of elementary action for a set of services.

For each type of services (mail, website, storage, etc.), this database would offer the *Model data* of:

- different programs with the same functionalities (for example Apache, NGINX, LiteSpeed, etc. for webservers)
- different versions of the same program (for the reproduction of attacks specific to a version of a product)
- the interactions with different types of clients (the requests can be different according to the client's OS or browser)

The database does not need to include *Model data* for all those situations. If the difference in the *Model data* is relatively minor (change over a small number of entries), those small changes can be implemented in the *Data generating function* thanks to *Elementary actions parameters*.

4.1.2 Definition of the sets of Elementary actions

Once the reference network is set up, we can start the process of actually capturing the *Model data*. We define the *Model data* as the captured data of the execution of *Elementary actions*. Thus, for each service, we need a set of *Elementary actions* to execute.

In our preliminary definition of *Elementary actions*, we also consider the role of the simulated actors in our simulation. The idea is that each action is available to one or several actors of the simulation. An evaluator decides different roles for the actors of its evaluation (examples: employees, attackers, administrator, commercial, CEO, clients, etc.). Those roles are attributed differently according to the goal of the evaluation and highlight categories of simulated users. Those categories can have different criteria. For example, users that are legitimate or not on the network (employees, clients, attackers),

that may have access to different resources (employees, administrator, commercial, clients) or that are priority targets (administrator, CEO). It may be useful for the evaluator to separate the actions that are specific to some categories.

Here, we consider categories of authorized actors: employee (regular user), administrator and commercial employee. Each category of actor can execute different sets of actions. If a user from a specific category executes an *Elementary action* from another category of users, it should be considered as abnormal behavior.

We devise the set of *Elementary actions* into two subsets. The first subset is comprised of the *Elementary actions* that every actor can execute. This subset is defined as follows:

- **connection to the webmail**: the user opens a browser (Firefox) and loads the url of the webmail server (http://mail.mycompany.com/~roundcube/). The user then inputs his credentials and clicks the "Connect" button.
- **open an email**: the user continues on the browser page opened by the "connection to the webmail". The user clicks on the first email of the list of emails in the reception box. The user stays on that page for a few seconds and then returns to the homepage of the webmail.
- **delete an email**: the user continues on the browser page opened by the "connection to the webmail". The users looks in the homepage for the email from a specified address with a specified subject. If that email is in the list, the user selects that email and clicks the "Delete" button.
- **write an email**: the user continues on the browser page opened by the "connection to the webmail". The user clicks on the "New message" button. Then the user fills the form with the specified destination address and content before he clicks the "Send" button.
- **disconnection from the webmail**: the user continues on the browser page opened by the "connection to the webmail". The user clicks the "Log out" button of the webmail and closes the browser page.

Those *Elementary actions* are not the full extent of the functionalities of the webmail server. A large variety of more complex *Elementary actions* can be imagined. For example, we could have *Elementary actions* to open specific emails, write an email to several destination addresses, put in copy addresses, delete several emails at the same time, open the attachments of emails, add addresses to contacts, etc. We limit ourselves to what we considere the most basic need to simulate the regular activity of an employee (the regular user of our system).

The second subset of *Elementary actions* are the actions specific to a single category of actors (the administrators and commercial users):

- **add an account**: the administrator opens a browser page and loads the administration page of the database of the webmail (http://mail.mycompany.com/

phpmyadmin/). The administrator inputs administrative credentials and clicks on the "Connect" button. The user then a SQL command to add an account on the database. The users logs out and closes the browser page.

- **delete an account**: same as the previous *Elementary action*, except that the administrator enters a SQL command to delete an account.
- **connect to "contact@mycompany.com"**: the same as "connection to the webmail" except that the commercial employee enters the credentials of the contact address of the company.

Based on these *Elementary actions*, the process to capture *Model data* is defined as follows. First, we create a Selenium script for each *Elementary actions*. The *Model data* of those actions are the PCAP files captured from the execution of the Selenium script. We capture each *Model data* several times.

We then compare those files together to be sure to not employ an outlier. By doing that comparison, we noticed a small issue for the *Elementary actions* that connect to the webmail. The first *Model data* captured of the action was significantly bigger that the following captures. That difference in size in the *Model data* captured for the same *Elementary action* is due to the browser that loaded the page for the first time and cache some data (fonts, logos, etc.). When that connection is made a second time, the browser does not request those data a second time and thus significantly reduce the size of captured *Model data*.

Elementary actions can be context-sensitive and provide different *Model data* for the same interaction. In our formalism, the evaluator can either make an *Elementary action* for each context, or – if the change in the *Model data* is small – adapt the *Simulation data* with *Elementary actions parameters*. In our case, we decide on the first solution and use a longer PCAP file as *Model data*. Those model data are more diverse in content than the second set of data that consist almost only of HTML code requests.

4.2 Data generating functions

At this point, we have set up a reference network, defined a number of *Elementary actions*, and we are able to capture *Model data* for each of those *Elementary actions*. Now, we need to define *Data generating functions* and verify that they possess the properties we defined in our model.

We present the *Data generating functions* introduced in the previous chapter, in Figure 3.3. We explain our implementation of the *Data generating functions* for the **Temporal** and **Network** levels of realism. We did not have the time to implement the *Data generating functions* of the **System** level of realism but we present our intuition on how to accomplish it. Also, we experimentally test those functions and analyze the results.

We also expand on the impact of *Elementary action parameters* on *Data generating function* and the tools we can develop to help in the creation of other functions.

4.2.1 Simulation control program

To deploy and test the *Data generating functions*, we created a simulation program to produce *Elementary actions* – for real and simulated activity – and to capture different types of *Model data*. This program serves as the basis of the simulation control program of our simulation. We present the program architecture in Figure 4.2.

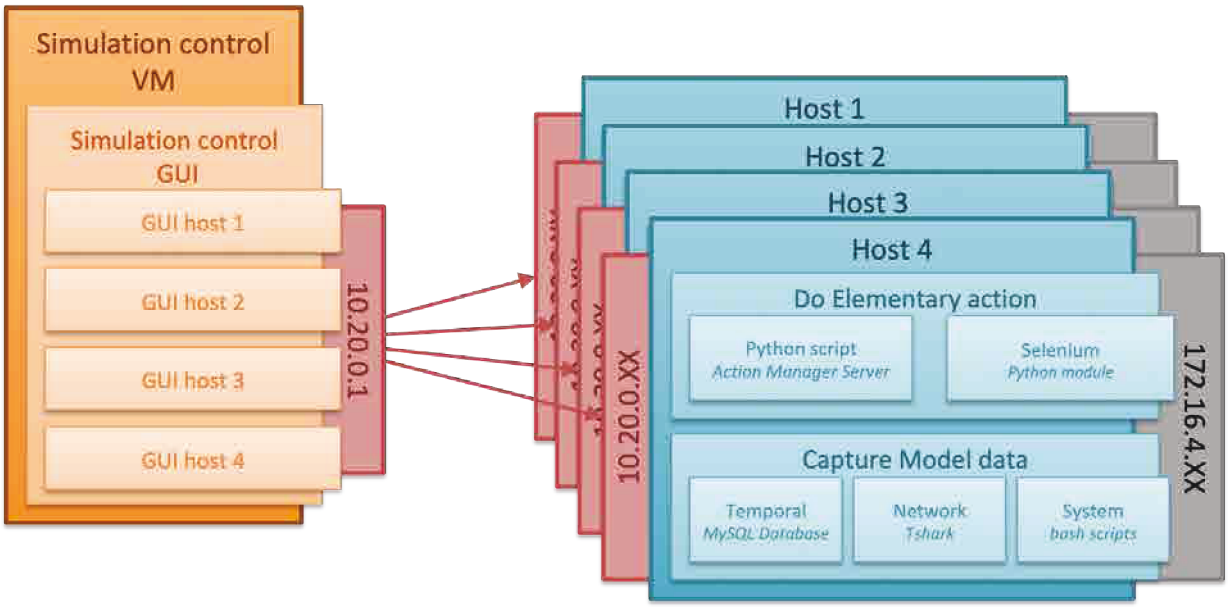

Figure 4.2 – Architecture of our simulation program

The simulation program in Figure 4.2 is deployed on the reference network built on Hynesim and serves as the basis for the simulation control program of our network simulation with Mininet. The simulation control VM is a QEMU VM added to the reference network shown in Figure 4.1. That VM is connected with a different virtual network with the users VMs. The user VMs (Host 1 to 4) are connected to two separated virtual networks: the internal network in black (on the right side of the hosts – 172.16.4.0/24) and a simulation control network in red (on the left side of the hosts – 10.20.0.0/24). In the internal network, only simulated data or real data are exchanged, while in the simulation control network there are only *Control data*.

For the interactions between the simulation control GUI and the hosts, we use a XML RPC server provided by Hynesim called the Action Manager server. That server is listening on the simulation control server and waits for a message telling it to execute a Python script with the provided arguments. Each Python script executed by the Action Manager server is a *Data generating function* and the arguments are *Elementary action parameters*.

The host VMs wait for instructions from the simulation control VM to do one or both of the following actions:

- Execute an *Elementary action*:
  - As real activity: the VM will execute a Selenium script to open a browser and do a series of actions as if it were a real user. For now, we only implemented "real" actions that happen through a browser because of our current limited simulation target. However we could easily imagine using a bash script instead and execute more complex actions.
  - As simulated activity: the VM will execute a Python script of a *Data generating function* with *Elementary action parameters*. The action is executed in the reference network and not in the simulated network. That function of the simulation control program serves to verify the proper working of the *Data generating function* and will be the basis of the simulation control program on the simulation network.
- Capture *Model data*:
  - We ask the host VM to capture the *Model data* of one or several types (temporal, network, or system) through different means (register execution time in MySQL, create a PCAP file with tshark, capture logs, etc.).
  - The capture can acquire from real or simulated activities. Captured data from simulated activity can then be compared to the data from real activity to verify that their properties are the same.
  - Each captured data file indicates the name of the *Elementary action*, the type of activity (real or simulated), and a timestamp.

Each of those actions can be manually decided by the evaluator in the simulation control GUI or automatically executed by the simulation program with a provided script.

This simulation control program and the listening servers on the host VMs serve as the basis for our development of *Data generating function* with a **Network** level of realism.

4.2.2 Levels of realism

As explained in the previous chapter, we call level of realism a set of *Data generating functions* that preserves properties of the *Model data*. We categorized those levels into three main types according to the type of *Model data*: temporal (uses time measurements as *Model data*, network (uses network traces), and system (uses several system data like logs, resource usages, etc.). The division into those types is not strict as more complex *Data generating functions* of higher levels of realism may require the data of lower level (for example, the *Simulated machine* level – the highest level we presented in Section 3.3.2 – simulates system and network data).

Temporal

The goal of this level of realism is to have virtual hosts wait the average time of execution of an action. This level of realism does not have an application use. We devised this level as a starting point and simple example of a *Data generating function*.

We created a Selenium script of each *Elementary action* we chose. On a QEMU VM, we ran that script ten times for each action and registered in a database the execution time of each action. Then we created a Python script that upon receiving an average time and standard deviation will wait for this average time more or less the standard deviation before sending an action-completed message.

Network

We defined three *Data generating functions* for the Network level of realism. The first *Data generating function* is the reproduction of solely the volume of packets. For this function we chose to keep the packet headers identical to *Model data* up to the transport layer included. The sole modification we made inside the header layers are changing the addresses of the packets to match simulation target.

The second *Data generating function* for this level of realism is the reproduction of packets. The packets that are reproduced must be accepted by the service as the *Model data* would be. In short, the *Data generating function* conserves the "acknowledgment by the service" property of the *Model data*. Our approach to that *Data generating function* is to have a main script that handles the actual reproduction of packets but calls a specialized script to handle key moments of the process that may differ from service to service. These key moments are the following:

- The initial state: does the service requires specific variables? A token? A cookie? etc.
- Before sending a HTTP request: should the variables in the header change, be removed or should new ones be added? It also adapt the request content according to the evaluator instructions with *Elementary action parameters*.
- When receiving a response: should we look for specific outputs in the answer?
- Retrieval process of the token: with what keyword can we identify the token field in the packet? Does that token require particular treatment before sending?
- The first connection: what variables should be removed or added from the header once the first connection is successful?
- Cookie: which model should it follow? What should it contain?

Although the main script is generic for all services, other scripts are service-specific and the identification of the changes to make for the key moments and the creation of the specialized script can be long and difficult according to the targeted service. The

creation of the specialized script for this *Data generating function* imposes another heavy burden to the evaluator who wants to use our method with this function.

However, there are several ways to avoid and/or solve that issue:

- To avoid this issue, the evaluator can use a service simulated by our method with a similar *Data generating funtion*. If the evaluation target of the evaluator is not a service but a security product, the evaluator may not require the verification of a valid cookie or token. Reproducing the *Model data* with the same (or random) token and cookies has little impact on the evaluated security product. There is no need for specialized scripts.
- To solve that issue, we can add to the simulation program presented in Figure 4.2 an identification and learning process (machine learning or otherwise) to automatically identify and deduce how parameters of the *Model data* change over several instances for the exact same action. The simulation program we presented allows for an easy replication and capture of a same *Elementary action*. An addition to that program could deduce the specialized script adapted to the service.
- Another method to solve the issue would be to execute in the virtual hosts the portion of the code that indicates the variables required by the service and what treatment should be applied. Even if the process VMs we use do not have the resources to handle any GUI like a browser, it should have enough resources to apply a simple portion of code. However, it might impact the performances.

We implemented the first solution and create an hybrid of a simulated service. The approach is quite simple: we use the exact same *Data generating function* with a global setting and put together all the packets of the *Model data* of each *Elementary action*. When the simulated service receives a request, it reproduces the packet from its pool of packets that answered that request. When a request is present inside several *Model data*, thus leading to several possible answers, the simulated service replies with the first one on the list. The obtained results were promising but inconsistent. However, we worry that our approach might be flawed. The more *Elementary actions* a service contains, the higher the chance of request packets being used in different *Elementary actions* but with different results. Therefore, the higher the number of *Elementary actions* on a service, the higher the risk of failure.

We can improve our method by determining which *Elementary action* is behind each request. We can do that by examing past requests of the user and guessing which *Elementary action* that request is most likely associated with. We left those proposals for further development of the presented model.

The third and last level of realism to consider is an improved version of the previous *Data generating function*. We call that function **adaptive replay**. While creating Selenium scripts to execute *Elementary actions* with a real browser, we realized that there was a series of user inputs that would differ from user to user or from session to session, such as credentials, email content, unread emails or nearly every input in a POST form. The previous *Data generating function* simply uses the exact same user

inputs as the *Model data* for every instance. Rather than having *Model data* for every set of user inputs, we proposed a *Data generating function* that would preserve the same properties but can modify user inputs through *Elementary action parameters*.

The proper modification of those inputs is easier for user inputs than hidden parameters like tokens or cookies. The identification and modification of those parameters can be not trivial and require trials and errors to manually deduce the proper handling of those parameters. A possible solution would be to have an identification and learning process to automatically identify and deduce how parameters of the *Model data* change over several instances. The comparison of several instances of the *Model data* of a same *Elementary action* is an efficient tool to identify those parameters. To deduce the modifications, we can try several basic transformations (copy, incrementation, addition of time, etc.) of the identified fields or compare with *Model data* of the same *Elementary action* with different user inputs.

System

The reproduction of system *Model data* is interesting for the evaluation of system security products. System security products analyze system data of the server or computer upon which they are install to raise or correct issues on that system. A good example would be an anti-virus. To evaluate a single anti-virus, a regular system VM is far more efficient that our proposed evaluation environment. However, those products often report to a central supervision server any attack they detect. A large scale deployment of an environment with a reporting of system security products is a necessity in the study of propagation effects, weak signals, detection of compromised machines in a sane environment, and others current attacks that are difficult to study without a large amount of ressources.

The first *Data generating function* on the System level of realism is a function whose sole focus is to provide the simulation data for system security products. It has no need of network *Model data*. The first difficulty we predict for that level of realism is the identification of the *Model data*. System security products use a far larger variety of data than network security products. They can analyze logs, critical files, resources consumption, etc. Different system security products may analyze different files and logs for their reporting or use different programs to fetch resource consumption information.

We can take several approaches for our simulations:

- On the evaluation target:
  - If the evaluator wants to solely evaluate the supervision service: our *Data generating function* can be a function that simulates the report data of the system security products. It is an easier function to implement as there is only one type of *Model data* to simulate. The principle is similar to our approach for the Network level of realism However, it does not evaluate the supervision system as a whole, just the supervision server in charge of collecting and analyzing the reports of security products.

  - If the evaluator wants to also evaluate the system security products: the task is more complex and must first start by an identification of the inputs of the system security products. We may need to change the network simulator we are using (Mininet) for another network simulator that provides stronger isolation (c.f. Section 4.3). Indeed, in Mininet all VMs share the same file system, making every VM have the same changes when modifying logs and important files.

- On the *Data generating functions*:
  - Have a *Data generating function* for each source of *Model data*. According to that approach, the evaluator has a panel of *Data generating functions* to execute for each *Elementary action*. The evaluator must use one for each input of the system security products. That approach can cause timing issues as the *Data generating functions* may not be executed at the same time, especially if there is a large number of inputs to consider. However, this may not be an issue with system security product with a periodic activation (for example, if the analysis of logs and files only occurs every XX minutes). This approach is very interesting if we are to consider partial signal of an attack (some of the *Simulated data* can be reproduced from an attack while the others are not).
  - Have a *Data generating function* for each system security product. The simulation is then tailored for a specific security product and does not require the evaluator to know all the inputs, if provided with the *Data generating function*. We also avoid an accumulation of delays between the inputs due to the execution of successive *Data generating functions* for one *Elementary action*. However, it limits the degree of control over the inputs of the security product and the *Data generating function* can not be reused for other security products. This approach is more appropriate for security products with a quicker response and an evaluation system with a variety of system security products.

- On the simulation of files and logs:
  - The *Data generating function* can directly modify the files and logs that the system security product uses. However, this may pose an issue if some of the impacted files are important for the proper activity of the VM.
  - To avoid the previous issue we may redirect the system security product towards a copy of all the contentious files and logs. But it is possible that we have to directly modify the security product if we can not redirect it through aliases.

- On the simulation of resources consumption:
  - The simulation of resources consumption can be tricky. We can try to intercept or replace the programs in charge of collecting the resources consumption to report the simulated value to the security product.

– An even trickier solution would be to execute tasks with similar resource consumption. The main point of our simulation method is to be able to capitalize on light network simulation that can not handle the execution of the actual task. So, to preserve that aspect, if we are to leave the programs in charge of collecting resource consumption, we can determine a series of tasks that can be executed on a process VM with a predetermined consumption of resources. However, it is difficult to have an exact prediction of the resources consumed by the task and it may not properly reflect the consumption on real machines. The best we can possibly do with such method is to have tasks that generate a wide range of resource consumption (low, medium, high) and use those levels to create abnormal consumption. We do not have tasks that create resources consumption similar to the model but only proportionally similar.

In short, we can either simulate the reports sent by actual security products to the supervision server, or try to extend our model to the more complex tasks of simulating system activity on process VMs. The first one is easier to implement and can be based on the *Data generating functions* of the Network level of realism as we just have to reproduce the packets sent by security products to the supervision server. However, this does not allow our model to evaluate actual system security products. The second approach presents multiple technical challenges on the reproduction of simulated files and resources consumption and imposes more stringent requirements on the selection of the network infrastructure of our simulation. But it allows us to have a more complex simulation system that can evaluate all types of security products and supervision systems.

The focus of our development has mostly been in the design of the model, the conception of the prototype, *Data generating functions* of the Network level of realism, and conducting validation experiments.

4.3 Network infrastructure

For the network infrastructure of our simulation, we must select a network simulator that matches the criteria we raised for our simulation. The traditional approach for a network simulator is to recreate a functional virtual network that could, in its ideal form, completely replace the original network but with the added advantage that the virtual network is easier to recreate and can be returned to its original configuration.

Our needs for a network simulator are different. We do not want to create a functional virtual network capable of replacing the original network. We want to create a network simulator that does not reproduce end-points nodes with all their functions but reproduces only the adaptable behavior of those nodes.

Figure 4.3 illustrates how our simulator works. Our simulator controls process VMs as end-point nodes, uses them to produce the *Simulation data* needed for our specific

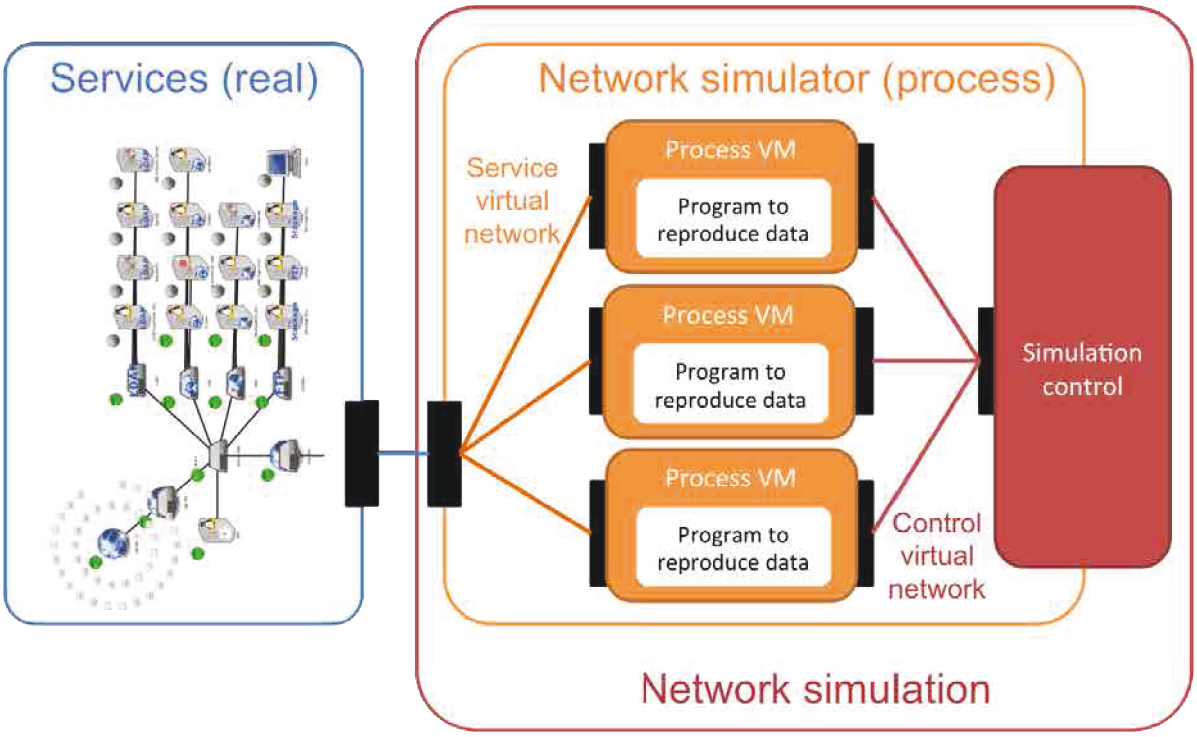

Figure 4.3 – Architecture of our network simulator

test, and make them interact with real services that are not aware of the difference. The services responses can also be simulated by our simulator.

4.3.1 Comparison between Mininet and IMUNES

To select the network simulator for our networking support, we must clearly identify our needs. We reduce our selection of network simulators from the taxonomy of virtualization products in Appendix A to two network simulators, namely Mininet and IMUNES, according to those criteria:

- a network simulator that uses process VMs
- open-source
- updated in the recent years
- easy to use
- used by the community

Mininet

Mininet[De Oliveira et al. 2014] is an open source network simulator which describes itself as a network emulation orchestration system. It is able to simulate all the elements of a network (switches, routers, links, hosts) on a single Linux kernel by promptly creating numerous *namespaces* that will be the domains of the Mininet hosts. These domains can behave like real machines and it is possible to run arbitrary programs installed on the host system on those domains and even start SSHD to access them. Each Mininet host acts as if they have their own kernel without any difference with a real machine. However each virtual host, or namespace, created by Mininet shares the same host file system and PID space. There is very little isolation between the virtual hosts created by Mininet and this is a major drawback as illustrated in Figure 4.4.

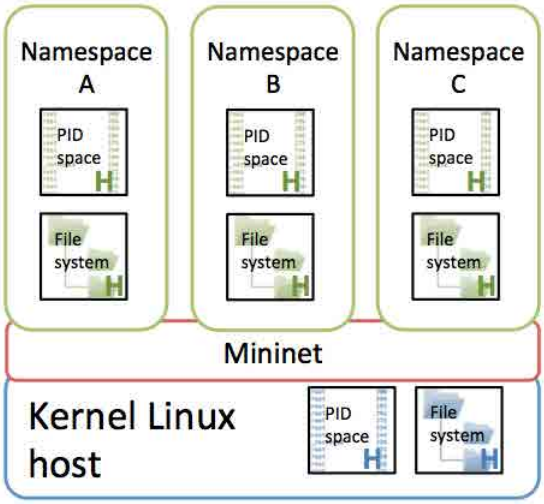

Figure 4.4 – Creation of namespaces by Mininet

Figure 4.4 illustrates that all virtual hosts created by Mininet share the file system and PID space of the host system. This process creates limitation. Software that depends on other operating system kernels cannot be executed and programs running into each namespace will share the same configuration files and PID space.

Mininet can efficiently generate a large quantity of virtual hosts sharing the same kernel and connect them together with virtual links, switches and routers according to either a predefined structure in the command line or by the use of CLI or python script to generate a customized network.

The virtual networks created by Mininet can be connected to external networks by adding an external interface to a virtual host and virtual hosts are able to write on virtual Ethernet interfaces in a manner close to real interface. The sent packets are subject to delays and link speed, and are processed by the virtual switches and routers as they would be by real hardware. The performances when two programs communicate through Mininet are quite close to those of native machines.

As a test, we use Mininet as the network backbone of our approach to network simulation in a nominal case with 5 virtual hosts. The results are close to what we expected. We observe that it effectively creates the needed virtual network, and the virtual hosts can correctly create packets like a real machine. The creation of virtual hosts is quite easy and an individual configuration for each virtual host is not needed. By using Python scripts, Mininet can be used to assure the scalability features of our network simulator. The lack of isolation between the virtual hosts, while not an obstacle for our simulation, can potentially impact the final results and the validity of the simulation at a large scale.

To conclude our evaluation of Mininet, we found out the following advantages at using Mininet:

+ Mininet complies with our needs for the network simulator.
+ It is easy to use through Python scripts. It makes the creation of customized virtual network easier.
+ Mininet can be deployed on any Linux kernel.

But we also noticed some drawbacks to Mininet:

- There is little isolation between Mininet hosts as they share the host file system and PID space.
- Mininet is not distributed and can only work on a single system. It creates limitation in resources.
- Mininet hosts can only execute programs on Linux kernel.

IMUNES

IMUNES[Puljiz and Mikuc 2006] is another open source network simulator that uses process VMs for virtual hosts. IMUNES also works on a single kernel but, contrary to Mininet, it does not create virtual hosts with a Linux kernel but a FreeBSD kernel. IMUNES uses a different method for virtualization. Those virtual hosts are not *namespaces* like in Mininet but *vimages*, virtual hosts constructed using *Jails* and clonable network stack in the FreeBSD kernel[Zec 2003].

Mininet and IMUNES hosts have similar capabilities as IMUNES hosts are also capable of executing programs on FreeBSD kernel and they can be accessed by SSH if the daemon is started.

However contrary to namespaces used in Mininet, vimages have a more thorough isolation between them. They no longer share the same PID space and file system as the host system. Each virtual host in IMUNES has the same file system at start but it evolves independently afterwards. It means that a new file created in the file system of one virtual host is not created in the file system of other virtual host like it was in

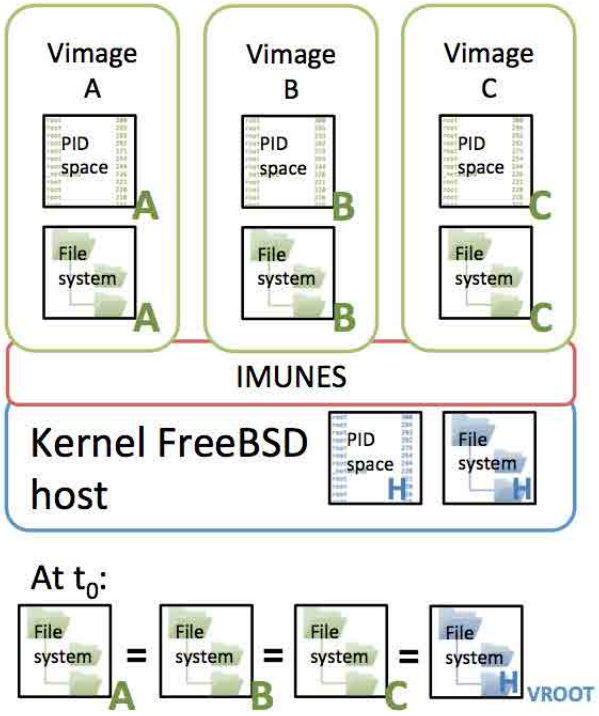

Figure 4.5 – Creation of vimages by IMUNES

Mininet. The file system used as reference for the virtual hosts when they are created is the `/var/imunes/vroot` repertory of the host file system. This repertory is represented by `File system` H_{VROOT} in Figure 4.5.

At the end of the simulation, any change added to the reference file system is discarded and the next simulation will start anew with the reference file system for every host. Another difference between Mininet hosts and IMUNES hosts is that the PID space of each created virtual host is independent from the start in IMUNES.

In the end, as illustrated in Figure 4.5, *vimages* and *namespaces* are working in a similar way but *vimage* provides better isolation and actually has a separate file system and PID space for each virtual host. As for the virtual network connecting the virtual host, it stays similar to Mininet. The virtual hosts write on virtual Ethernet interfaces as they would on a real one and the packets are processed by switches, routers and links like in a real network with the same limitations that could be customized (delay, limit speed, queuing, bandwidth, duplicate packets, lost packets...).

As we did with Mininet, we also tested IMUNES as the networking support of our approach to network simulation in a nominal case with 5 virtual hosts. The results were similar to those of Mininet but we noticed a few differences in concept. One difference is the concept of custom-configuration that does not exist in Mininet. A custom-configuration is a script, possibly in bash, that the virtual host will execute at its creation. It can enable virtual hosts to start specific services from the very beginning.

IMUNES also differentiates itself from Mininet in its use. Mininet can either be used with the CLI or launch predefined architecture, in command line or create a customized network with a Python script. With IMUNES there is no CLI but a GUI to create an architecture and save that architecture in a custom file describing the nodes of the network and the links connecting them. Simulations are started in IMUNES from the GUI or in a command line with a custom file. However, a new network from scratch can not be created with a command line. This detail has its importance because using a GUI is ill-versed for scalability purpose and being able to immediately add a large number of hosts is important for scalability.

Our analysis of IMUNES showed that:

+ IMUNES complies with our network simulator criteria and presents features similar to Mininet.
+ It has the added bonus of providing isolation between the virtual hosts
+ IMUNES can run on any FreeBSD kernel.
- IMUNES is not distributed
- IMUNES only executes FreeBSD programs
- The GUI is ill-versed for scalability

4.3.2 Justifications for the choice of Mininet as the network infrastructure

Although IMUNES has the added advantage of having an isolation between its virtual hosts, it also has a lot of implementation difficulties.

We implemented two versions of our simulation prototype using Mininet and IMUNES respectively as the network infrastructure but IMUNES needed far more efforts to implement and carried issues when trying to create a larger scale simulation. It took far more time than Mininet to launch a similarly scaled simulation and the results obtained were inconsistent as we upped the scale. Moreover, it created various compatibility issues as we developed our prototype on a Linux OS rather than FreeBSD. The results obtained at a small scale where IMUNES was stable were not conclusive enough to convince us to invest more effort into the development of a prototype with IMUNES.

Although the technical challenges provided by IMUNES were not impossible to overcome, the fact that the results obtained were at best equal to Mininet pushed to cut short our losses and select the working prototype with Mininet.

However, the goal of our prototype is to be able to conduct experiments to validate our proposed model of generation of evaluation data. More work and improvements are needed to make it a performant tool and to that end a different choice for network

infrastructure might emerge. Indeed, our model aims to be independent of the network infrastructure and our choice of network infrastructure for our prototype should not limit the future development and evaluation tools based on our method. We do not claim that Mininet is the best network infrastructure for our method but it was the most appropriate network structure for the development of our prototype.

4.4 Experimental validation

In this section, we present the validation experiments and results of the highest level of realism we implemented: reproducing modified packets. It is the highest Network level of realism where the *Data generating function* preserves the "acknowledgement by the service" property while allowing to modify user inputs with *Elementary action parameters*.

In Section 3.2 and Subsection 3.3.2, we identified a series of properties that our model must respect to work properly and to be as close as possible to our criteria of a data generation method:

- the verification property of *Data generating functions* (c.f. Property 2)
- the reproducibility property of *Data generating functions* (c.f. Property 3)
- the verification property of the *Scenario* (c.f. Property 4)
- the implementation requirement on *Control data* (c.f. Property 5)
- the implementation requirement for scalability (c.f. Property 6)

In our experiment we must confirm that those properties are verified by our prototype and our *Data generating function*. We developed our prototype taking the implementation requirements (Properties 5 and 6) into consideration.

We selected the network simulator Mininet in accordance to Property 6: it relies on lightweight virtual machines and can create a large amount of virtual hosts (around a thousand) connected with virtual links in a few minutes on a regular computer. Mininet uses the lightweight virtualization mechanisms built into the Linux OS: processes running in network namespaces, and virtual Ethernet pairs. Mininet can emulate links, hosts, switches, and controllers at a very low resource cost.

Concerning Property 5, we simply ask the simulation control program to write in a file the information required by that property.

Thus, the challenge of those experiments is to validate the verification property of *Data generating functions* (Property 2), the reproducibility property of *Data generating functions* (Property 3), and the verification property of the *Scenario* (Property 4).

4.4.1 Experiments context

For our experiments, we simulate the activity of 50 to 200 *Hosts* representing regular employees of a small company interacting with the *Service* of a Roundcube webmail server on a Postfix mail server. A simulation control program follows the *Script* described in Figure 5.5 for all the *Hosts* of the simulation. In Figure 5.5, the *Elementary actions* are in bold while actions that do not generate activity data are in a regular font. The *Host* can simulate two different series of *Elementary actions* after a waiting period of X seconds each time. Therefore, the intensity of the *Script* can be modulated by modifying the value of X.

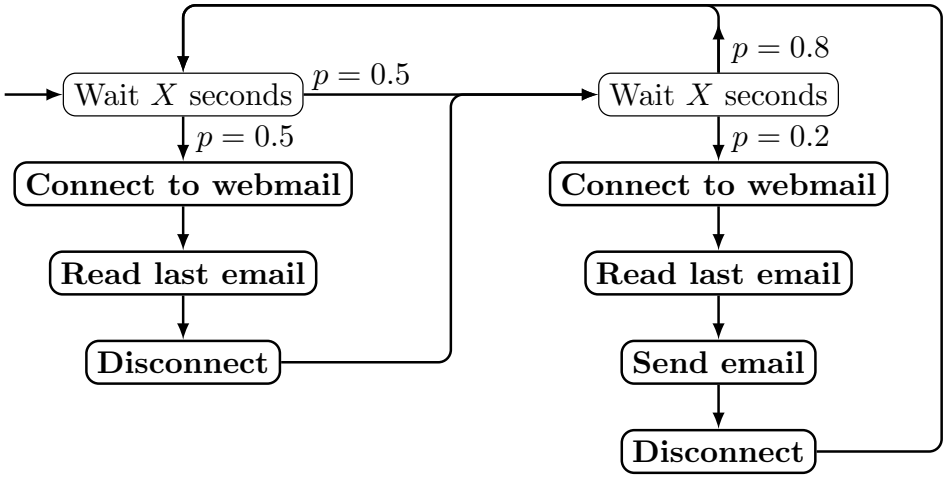

Figure 4.6 – Generation of simulated activity from short traces

We verify the properties with two separate experiments. The first experiment is a control experiment. We deploy 5 virtual machines on the network simulator Hynesim and make them generate the activity of our simulation. We script the *Elementary actions* of the *Script* described in Figure 5.5 with the web driver Selenium and make the virtual machines use their browser to interact with the webmail server. This experiment provides referential values for our second experiment. We expect proportionality between these values and the results of our simulation, with respect to the number of *Hosts*.

In the second experiment, we simulate different number of *Hosts* (5, 50, 100, 150, 200 and 250) in Mininet and make them generate the activity of regular users using a webmail service for 30 minutes. We measure the activity at three different points: the webmail server, the network simulator Mininet and the server hosting the simulation. Every 30 seconds, we measure four parameters: CPU usage, memory usage, network I/O, and disk I/O. Figure 4.7 is an example of the measured activity. It represents the network traffic received and sent by the webmail server with 50 simulated *Hosts*. Each *Host* follows the *Script* described in Figure 5.5, with $X = 30$. The webmail server sent several times more packets than it received. This difference is consistent with the fact that content requests take a lot less packets than sending that content.

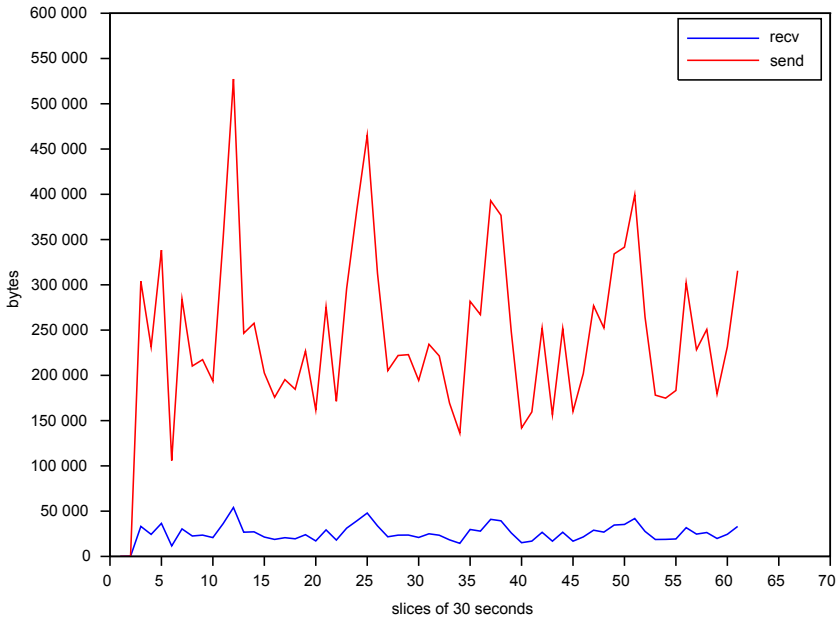

Figure 4.7 – Network traffic of the webmail server for a single experiment (50 *Hosts*)

We also retrieve the logs produced by the webmail server during both experiments. The quantity and content of the logs is analyzed in Table 4.1 and Table 4.2. Both experiments are done twenty times for each set of parameters to ensure the consistency of the results (Property 3). In the results, we display the average value and standard deviation of those twenty experiments.

4.4.2 Performance results

To verify the verification property of *Data generating functions* (Property 2), we must check that the simulated data had the same impact on the service as the *Model data*. It entails that the log entries of the service from the second experiment must be similar to the log entries of the referential experiment. The verification property of the *Scenario* (Property 4) indicates that the results should also be proportional to the results from real activity. Thus the simulated activity must be equivalent to the referential experiment in a similar context but it also must match our expectation in different contexts (different number of simulated hosts). Lastly, we verify the reproducibility property of *Data generating functions* (Property 3) by doing 20 instances for each set of parameters of the second experiment. Each instance must be equivalent to the other instances for the same set of parameters (modulo the randomness factor in our *Scenario*). Thus the standard deviation of our results must be in an acceptable range.

Table 4.1 represents the quantity of logs produced by the webmail server during both experiments. We display the average number of lines in the log files of the webmail and

	5 VMs		5 Hosts		50 Hosts		100 Hosts	
Filenames	**avg**	**stdev**	**avg**	**stdev**	**avg**	**stdev**	**avg**	**stdev**
userlogins	90	9	112	10	1032	36	2084	45
imap	43245	5070	57775	5306	487883	22742	984642	28820
sql	4955	525	6703	563	56081	1886	113031	2452

	150 Hosts		200 Hosts		250 Hosts	
Filenames	**avg**	**stdev**	**avg**	**stdev**	**avg**	**stdev**
userlogins	3085	52	4121	53	5118	74
imap	1450507	27792	1933823	21117	2748252	35985
sql	167138	2964	223427	2906	265354	4688

Table 4.1 – Number of lines in the webmail log files.

their standard deviation. The first column is the name of the main log files produced by the server: "userlogins" logs every connection (successful or not), "imap" logs every instruction from the server that uses the IMAP protocol, and "sql" logs every interaction between the server and its database. The entries under the name "5 VMs" correspond to the results of the referential experiment while the other entries are the results of the simulation experiment.

The analysis of the number of entries into each log files serves as a rough indicator to known if the simulated traffic matches our expectation. By comparing the number of entries during the second experiment (increasing number of hosts simulated by our prototype) with the number of entries in the control experiment, we can determine if those entries meet our expectations.

The number of lines in "userlogins" represents the number of connections during the experiments (one line per connection) and can be used to calculate the number of sessions created during both experiments. Figure 4.8 shows the average number of sessions created during the second experiment and its standard deviation according to the number of simulated *Hosts*. We also estimate the average number of sessions inferred from the results of the control experiment, based on proportionality (avg("5 VMs") $\times$ $\frac{number\ of\ Hosts}{5}$).

We observe that the number of sessions created during the second experiment is close to our estimation. Our simulation produces more sessions than expected. This is due to the fact that our *Data generating function* reproduces the *Model data* of an *Elementary action* faster than the browser of the virtual machines. Hence, in a period of 30 minutes, the simulated activity has gone through more cycles of the *Script* than the control experiment. A projection of the number of lines of the other log files ("imap" and "sql") displays similar results. These results establish that the simulated activity produces a consistent amount of logs.

Another rough indicator is the quantity of traffic produced by both experiments. In Figure 4.9, we examine the network traffic produced by our simulated activity. The lower plain (blue) and upper plain (red) lines represent the average number of bytes, respectively received and sent by the webmail server every 30 seconds, along with the standard deviation in dashed lines. For comparison, the black lines with respectively circles and triangles correspond to the estimation of the expected results for received

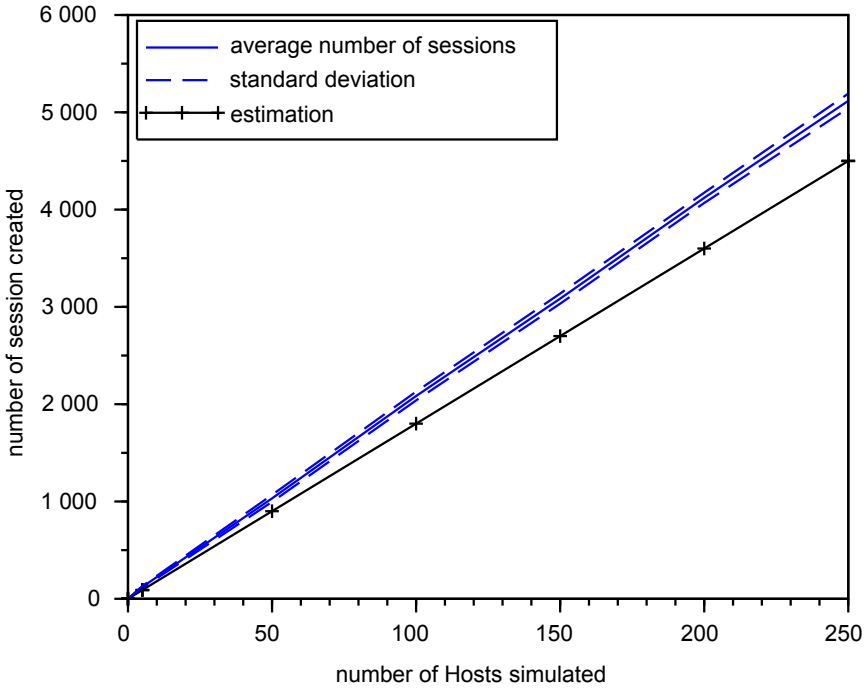

Figure 4.8 – Number of sessions created during simulation (plain blue line) compared to estimation (crossed black line)

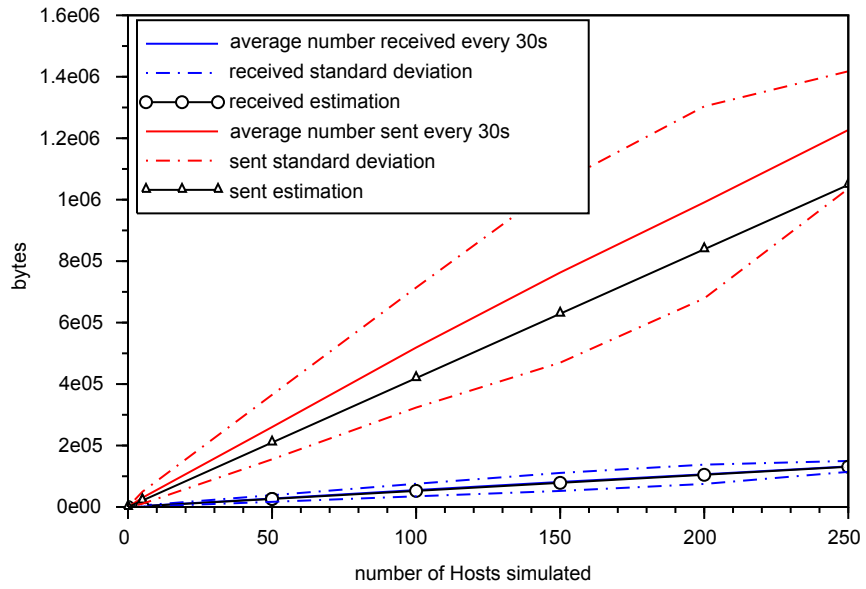

Figure 4.9 – Network traffic of the webmail server

and sent traffic based on the control experiment. As before, the results of the second experiment are close to our estimation. The deviation can be justified with the same explanation regarding the activity speed difference. This deviation is also partly due to cached data. Since these data are stored on the host after the first connection, the amount of exchanged data during the first connection is higher than during subsequent sessions.

However, our *Data generating function* does not take caching mechanisms into account. Therefore, our simulated connections request more data from the webmail server than estimated. Adding *Elementary action parameters* to modify the behavior of the function can solve this issue as we did for previous typing and semantic issues. We previously discussed (Subsection 4.2.2) methods to improve the addition of *Elementary action parameters* in *Data generating functions*.

Despite those issues, we have shown that the simulated activity of the second experiment generated a large network activity proportionally to the number of simulated *Hosts*, as expected. We respect our expectation of scalability. We now focus on proving that the activity semantics was also preserved.

4.4.3 Semantic results

For each *Elementary action* of the activity *Script*, we look for log entries that could act as signatures for the action. We select those signatures by comparing the logs of the different *Elementary actions*. The log entries that appeared for only one *Elementary action* are selected as signatures of that action.

Those log entries are used to verify that the server acknowledges the simulation data as it would real actions. We also manually verified the correctness of the *Data generating function* for some *Elementary actions*. For example, we ask the simulation to do the "read the email" action for a different email that the one in the *Model data*. Or to do the "connect to the webmail" action with purposely wrong credentials. In both cases, the manual analysis of the simulated data showes that the webmail server properly acknowledged the simulated data.

Signatures from log entries is a more global form of verification. By comparing these signatures in both experiments, we obtain the results displayed in Table 4.2.

From Table 4.2, the following observations can be made:

- the number of signatures for the "connect" *Elementary action* is slightly inferior to the number of sessions (the number of lines from "userlogins") observed for 150 *Hosts* and above. It is explained by the fact that the signatures correspond to the number of successful connections to the webmail server. If we remove the number of lines in the "userlogins" file that correspond to failed connections, we find the exact number of signatures for the "connect" *Elementary action*.
- the number of signature for the "disconnect" *Elementary action* corresponds to the exact number of sessions observed in Table 4.1.

		5 VMs		5 Hosts		50 Hosts		100 Hosts	
Signatures	**Actions**	**avg**	**stdev**	**avg**	**stdev**	**avg**	**stdev**	**avg**	**stdev**
imap.sign1	connect	90	9	122	10	1032	36	2079	44
imap.sign2	connect	90	9	122	10	1032	36	2079	44
imap.sign3	connect	90	9	122	10	1032	36	2079	44
imap.sign4	connect	90	9	122	10	1032	36	2079	44
imap.sign5	connect	90	9	122	10	1032	36	2079	44
imap.sign6	connect	90	9	122	10	1032	36	2079	44
imap.sign7	connect	90	9	122	10	1032	36	2079	44
imap.sign8	connect	90	9	122	10	1032	36	2079	44
imap.sign9	connect	90	9	122	10	1032	36	2079	44
imap.sign10	connect	90	9	122	10	1032	36	2079	44
sql.sign1	connect	90	9	122	10	1032	36	2079	44
sql.sign2	disconnect	90	9	122	10	1032	36	2079	44
imap.sign11	open	89	9	122	10	1028	36	2069	44

		150 Hosts		200 Hosts		250 Hosts	
Signatures	**Actions**	**avg**	**stdev**	**avg**	**stdev**	**avg**	**stdev**
imap.sign1	connect	3075	55	4118	54	4874	87
imap.sign2	connect	3075	55	4118	54	4874	87
imap.sign3	connect	3075	55	4118	54	4874	87
imap.sign4	connect	3075	55	4118	54	4874	87
imap.sign5	connect	3075	55	4118	54	4874	87
imap.sign6	connect	3075	55	4118	54	4874	87
imap.sign7	connect	3075	55	4118	54	4874	87
imap.sign8	connect	3075	55	4118	54	4874	87
imap.sign9	connect	3075	55	4118	54	4873	88
imap.sign10	connect	3075	55	4118	54	4873	88
sql.sign1	connect	3075	55	4118	54	4874	87
sql.sign2	disconnect	3085	52	4121	53	5118	74
imap.sign11	open	3059	52	4090	53	4808	91

Table 4.2 – Signature log entries.

- the number of signatures for the "open" *Elementary action* is slightly inferior to the number of signatures for the "connect" *Elementary action* for 50 *Hosts* and above. It is likely due to the experiment ending before the last *Script* cycle ended for a few *Hosts*.
- no characteristic entry for the "send an email" *Elementary action* could be found in the "userlogins", "imap" and "sql" log files.

The failure of several connections in our simulation may also be due to the parameterization of the *Data generating function*. The adapted replay *Data generating function* was designed to modify short-lived information from the *Model data* like the token or the session identifier according to the server reply from the requests. However, such modification was not included in the first request. The webmail server possibly refused some connections because they contained the same information at the same time. Therefore, an improvement of the typing of the adapted replay *Data generating function* should raise the number of successful connections with a high number of simulated *Hosts*. Table 4.2 shows that for each successful session in our simulated activity, the webmail server correctly interpreted the *Elementary actions*.

To sum up the results analysis, our prototype generates a simulated activity that produces a realistic amount of network traffic and logs from the webmail server (Property 4). Moreover, the webmail server produces the appropriate number of logs reflecting the correct semantics. Each simulated *Elementary action* resulted in the same log entries as a real one (Property 2). Each result was verified twenty times (Property 3). Therefore, our prototype succeeds in providing scalable and realistic data generation, thus validating our model.

4.5 Conclusion

In this chapter, we discussed the issues we encountered in the establishment of a prototype of our method: the acquisition of *Model data*, the development of *Data generating functions*, the selection of a network infrastructure, and the experimental verification of our prototype.

For each of those issues, we explained how we treated the issue and proposed methods for better solutions to the issues we could not solve. We also explained the future directions for our prototype: more *Elementary actions*, improved acquisition of *Model data* coupled with the automatic treatment of *Elementary action parameters*, the System level of realisms, a prototype with isolation between virtual hosts, etc. Despite not attaining all our ambitions, we developed a prototype that verified all the properties and requirements of our simulation method. Although the results of this prototype tend to degrade after a certain point in scaling, the results are good enough to attempt to apply our prototype to an actual evaluation of a security product or service. We attempt such evaluation in the next chapter.

We proved that our method, despite all its requirements and imposed properties, is implementable and shows a lot of potential. Currently, a lot of preparation work is required from the evaluator to use our prototype but the proposed improvements can greatly reduce that burden. If in future development we can create stable and complete tools that can support multiple network infrastructures, we would have a powerful evaluation tool and process that can adapt to the available resources of the evaluator, the evaluation target requirements and goal of the evaluator.

CHAPTER 5

Evaluation of services and security products

In the previous chapter, we discussed the prototype we developed based on our model and the requirements we established in Chapter 3. The scope of the prototype is still limited, and improvements are possible. However, the experimental results of the verification experiments showed that the prototype could be used for the evaluation of services and security products as long as we stay inside the operational range of the prototype.

To that effect, we discuss in this chapter of the methodologies to apply to evaluate security products and service. We describe the place of our network simulation in the topology of an evaluation. According to the type of products evaluated and the amount of preparation work the evaluator is willing to do, our simulation can be one element among several tools or the sole tool of the evaluation. Indeed, our simulation can be connected to external components to reduce the preparation burden of the evaluator. However, the addition of external components implies more effort to be made on the composition of the ground truth.

To illustrate our methodology and present a use case of our simulation, we evaluate an open-source IDS called Suricata. For that evaluation, we choose to follow a methodology that uses external components as the source of attacks and malicious traffic. Although the current capacity of the prototype limits the obtained results, it shows some interesting results that confirm the needs for this type of simulation.

5.1 Methodologies

5.1.1 Objectives of the methodology

Before explaining how to apply our proposed simulation method to the evaluation method, it is good to recall some of the notions we presented in the evaluation of services and security products in Chapter 2, Section 2.1.

As illustrated in Figure 5.1, an evaluation requires two elements: an evaluation target and an evaluation environment. The evaluation targets we consider are either

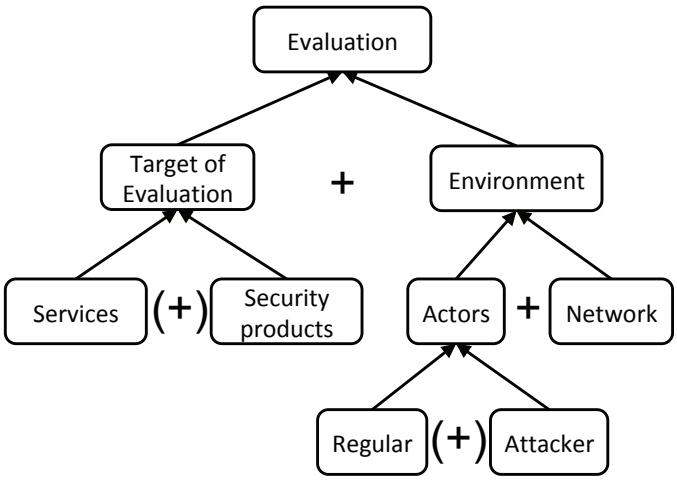

Figure 5.1 – Elements of an evaluation

services or security products. However, according to the type of target, our evaluation may include both services and security products. For example, in-line security products like IDSs or firewalls require other services in the evaluation. In-line security products analyze traffic between two actors (two services, a service and a user, or two users) and deduce the action to take. Unlike out-of-band security products like an authentification server, in-line security products are not security products that are directly requested by services or users. Thus, we cannot evaluate their actions without generating exchanges between services and users.

An evaluation also requires an evaluation environment that provides a network and actors. We can further divide the actors according to the need of the evaluator. In a security evaluation of security products, we divide those actors into regular users and attackers. In the performance evaluation of services, the evaluator can decide to divide the actors according to their role in the evaluation (employees, clients, administrator, developers, etc.). The categories of actors in a simulation are not fixed and can change according to the need of the evaluator. Our simulation model does not have to provide all those roles. We choose a network infrastructure that can connect external components to the simulation, and the evaluator can delegate some of those roles to external components.

In this chapter, we develop an evaluation methodology based on our method that takes into consideration the type of evaluation target and the diversity of roles of the actors of the simulation. We propose methodologies to evaluate services and security products that can either solely rely on our simulation or that employ external components for some roles to lighten the burden on the evaluator. We explain the advantages and drawbacks of each choice.

5.1.2 Evaluation of services

Figure 5.2 shows the general topology of the network simulation. The virtualized structure of the simulation creates the hosts and connects them in a simulated network. We

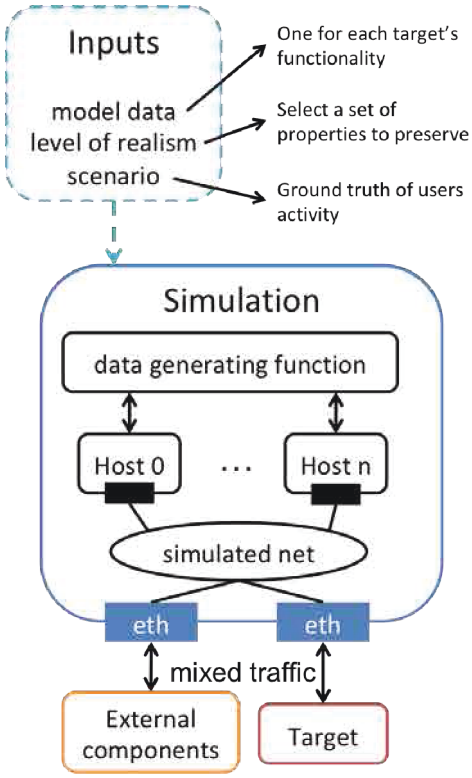

Figure 5.2 – Simulation general topology

connect the evaluated target to that network along with other external components that the evaluator may use. To use the simulation, the evaluator must provide three inputs: the *Model data* that the simulation will reproduce, the level of realism of the simulation, and the *Scenario* of the simulation.

The evaluator must provide *Model data* for each *Elementary action*. Those *Elementary actions* will correspond to entries of the ground truth of the actions taken by the users. To ensure the compliance with the specifications, we select as *Elementary actions* each functionality of the evaluation target. The evaluator must manually execute each functionality of the service and record the resulting data, which form the *Model data*. The nature of that data can change according to the input requirements of the data generating function of the simulation. It can be values, logs, network traces, or otherwise.

The level of realism is not an actual input of the simulation model, but one of the simulation parameters. It corresponds to a *Data generating function* that reproduces the model data while preserving one or several properties of the model data (size of packets, acknowledgment by the service, waiting time, etc.). We discussed the several levels of realism we implemented in Section 4.2.2. The choice of the level of realism is a critical step to produce *Simulation data* that are realistic to the eyes of the evaluator.

The third input required for the simulation is the *Scenario*. This *Scenario* is composed of sets of *Elementary actions* and *Elementary action parameters*. Those *Elementary actions* are the functionalities of the services. Just like the functionalities might

require parameters, the evaluator has to provide *Elementary action parameters* necessary for the compliance to the specification. Those parameters may be requirements for the proper working of the *Elementary actions* (e.g., credentials) or inputs to improve the variety of the *Simulation data* (e.g., contents of an email).

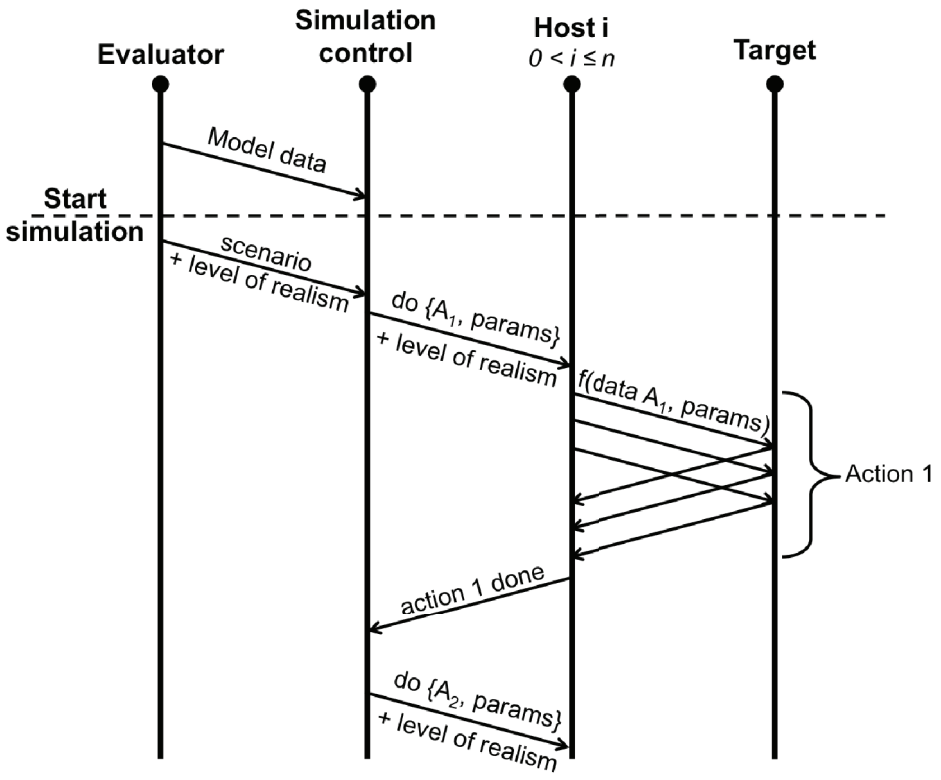

Figure 5.3 – Sequential diagram of the simulation

Figure 5.3 illustrates the sequential process of the simulation with the elements previously explained. The figure displays only one of the virtual hosts of the n hosts that interact simultaneously with the target. In this example, the evaluator gives a scenario that required the host H_i to simulate the *Elementary action* A_1 followed by the action A_2. The virtual host generates the data on the client side of A_1 with the data generating function f corresponding to the selected level of realism. The host gives f the model data of the action A_1 and the *Elementary action parameters* in the scenario. With an appropriate scenario, the network simulation can make a large number of clients use all the functionalities of the service, verifying both the compliance to the specifications and the workload processing capacity.

For the production of attacks, there are two solutions: the evaluator can either provide inputs for attacks as he would for other elementary actions or connect an external source of attacks (exploit database, physical attacker, vulnerability scanner, etc.) to the simulated network. The simulation model uses the same data generating function to produce malicious or benign data. Thus, the generation of malicious attacks requires the same inputs as benign data. It can be burdensome for the evaluator to capture sufficient *Model data* to reach a decent variety of attacks to evaluate the attack coverage of the services. The use of an external component like a vulnerability scanner

can be more appropriate for such tasks. If the evaluator only needs to cover a small variety of attacks, the burden on the evaluator would be much smaller.

The last step of every evaluation is the analysis of the experimental results. This step is a comparison of the ground truths of the different components of the evaluation: services, security products, network, and users. The evaluator compares the ground truth of the evaluated target with the ground truth of other components. Thus, in the study of discrepancies, we can ascertain if the mistake comes from the evaluated product or the evaluation method.

The network simulation provides the network environment and actors (regular users and attackers). The ground truth for the actors are the *Scenario* and *Control data* generated by the simulation. They display which *Elementary actions* were simulated by which host with which parameters. For the network, an analysis of the network traces can constitute a ground truth. Indeed, by identifying specific URLs or specific packets, or if the attacker comes from an external component, we can identify the network traces based on the IP addresses. We then compare that ground truth to the simulation scenario or control data.

To sum up, in addition to the necessary inputs for the simulation we presented in the previous chapter, the evaluator that evaluates a service must:

- capture *Model data* for each functionality of the service
- decide if the simulation simulates all actors (regular users and attackers) or relies on external components for some roles
- have a ground truth of a component of the simulation other than our simulation for comparison. It can be the analysis of network traces. The ground truth of other actors must complete the ground truth of the simulation if external components are involved.

5.1.3 Security Products

When evaluating an in-line security product, the evaluator must configure the simulation to generate traffic between users and services. Contrary to the evaluation of services, that traffic does not need to cover every functionalities of the services. So long that the evaluator constructs a realistic activity model of the interaction with the services (the *Scenario*) it will be enough.

In the evaluation of an out-of-band security product, the evaluator is no longer required to generate a simulated activity between users and services. Like the evaluation of services, the evaluator must first identify the actors interacting with the functionalities of the security product and reproduce these interactions. They can be interactions between the security product and users or between the security product and services or both. The evaluator must then capture the model data of those interactions.

The evaluator may construct different *Scenarios* according to different situations (regular use of services, overload of requests, under attack, etc.) and generate *Simula-*

tion data of several services protected by the same security product. The attacks can be *Elementary actions* of that *Scenario* or external components. Like in the evaluation of services, to ensure a proper evaluation of the attack coverage, we advise the use of an external component as a source of attacks. However, the addition of the ground truth of the external component is left to the evaluator.

We mentioned in Chapter 2 that the evaluation of security products consists of the verification of four properties: policy accuracy, attack coverage, workload processing capacity and performance overhead. The network simulation can verify these properties on a security product.

The policy accuracy represents the accuracy of the judgment of the security product, and its evaluation depends on the configuration of the security products. To properly evaluate this property a reliable ground truth is required. Its difficulty depends on the topology chosen, more precisely if it includes external components or not. If the evaluation includes external components, additional work must be done to generate a complete ground truth of the evaluation, but the external network interface of the components make their traffic easily identifiable.

The "attack coverage" property also generates additional work for the evaluator according to the topology choice. If the simulation simulates the attackers, the evaluator must capture a large amount of the model data added to match the variety of attacks of an external attacker.

The "workload processing capacity" property is inherent to our simulation method. The method aims to generate consequent benign traffic along with attacks for an evaluation closer to production context.

The last property of the "performance overhead" evaluation requires an additional step to the experiment. The network simulation can be supported on a single server. It is quite easy to measure the performances of the server as a whole for two experiments. The first experiment is the simulation without the security product, and the second experiment consists of the same simulation with the security product. The difference between the two should provide the performance overhead due to the security product.

5.2 Evaluation of an IDS

In this section, we present an application of the previous methodology for the evaluation of a security product, focusing on the Suricata IDS. The goal of this evaluation is to verify the impact of a consequent volume of benign data on the detection rate of Suricata. We also want to look for differences in the analysis by Suricata between live and offline traffic. We call live analysis the alerts generated by Suricata during the simulation and offline analysis the alerts generated by Suricata from the network traces captured from the simulation. The main difference between the two analysis is that during the live analysis, Suricata is under stress to keep up with the flow of data passing through it, while in the offline analysis Suricata reads the traces at its own pace.

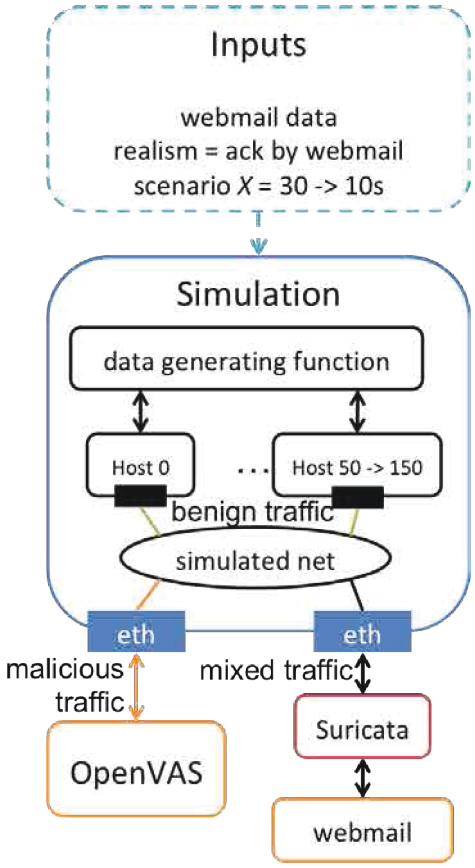

Figure 5.4 – Topology of our evaluation of Suricata

Figure 5.4 describes the topology of our evaluation of Suricata. The simulation is created and managed by the prototype described in Chapter 4. We chose to have an external service and an external attacker to evaluate the IDS. We set up Suricata with the rules of EmergingThreat as available in January 2018. We left the basic configuration of Suricata. The external service is a webmail server (Postfix + Roundcube) that we set up to look like the webmail server of a small company. The external attacker is OpenVAS, a vulnerability scanner. We used the scan named "Full and fast" that test 62 families of vulnerability. We scan the external server, and the probe passes through the simulated network. This scan lasts an average of 23 minutes and starts 3 minutes after the start of the simulation scenario of 30 minutes of benign activity.

The simulated hosts are the employees of a small company. The employees all follow the same script of activity shown in Figure 5.5.

Figure 5.5 represents the decision graph of the *Elementary actions* performed by each host. The host waits X seconds before deciding with a probability of 0.5 to perform the first series of *Elementary actions* (connect $\rightarrow$ read email $\rightarrow$ disconnect). It then waits X seconds again and decides with a probability of 0.2 to perform the second series of *Elementary actions* (connect $\rightarrow$ read mail $\rightarrow$ write mail $\rightarrow$ disconnect). Each simulated host repeats that script for the whole duration of the simulation.

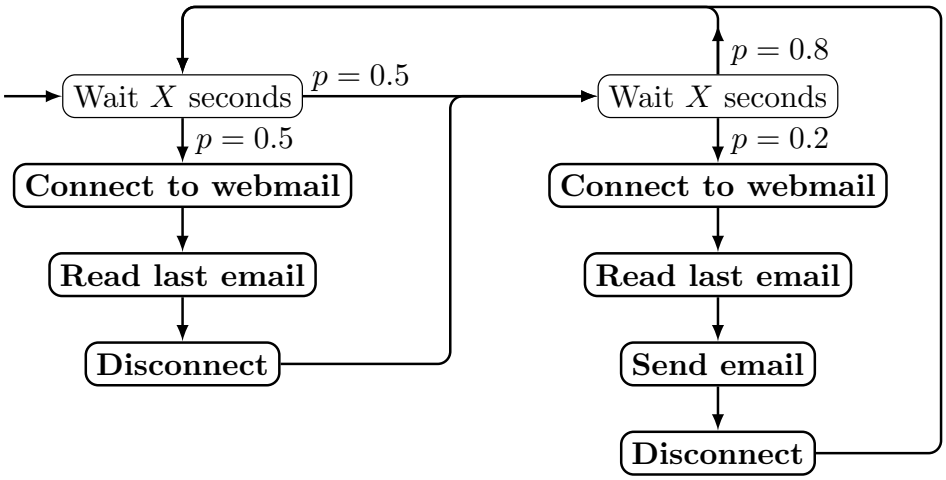

Figure 5.5 – Script of the benign activity

We regulate the intensity of the generated benign traffic with the number of hosts simulated during the experiment (50, 100, 150, 200 or 250 hosts), and we control the intensity of the script by changing the waiting period X between each series of *Elementary actions* (10 or 30 seconds). The data generating function (selected by the level of realism) that we choose for this evaluation is a function that generates the same packets as the model data but modifies some elements so that the webmail server may accept them. They are two types of modifications: modifications for the sake of variability and modification for the sake of functionality. The first kind modifies a part of the content of the model data that may differ from user to user (examples: credentials, email content, etc.). The second modifications focus on elements that are time sensitive and, therefore, must be changed to be accepted by the service (e.g., the token ID, the cookies, the IP addresses, etc.). They do not affect the variability but maintain the consistency of the data generation function.

5.2.1 Evaluation with benign traffic

Before starting the test with mixed traffic, we need to have a reference point for benign traffic and malicious traffic. We start with benign traffic. We do the evaluation without the external attacker OpenVAS. The goal is to obtain a reference point for the quantity of simulated data generated solely by the simulation and see if these benign data raise any alert on the IDS. We test different numbers of hosts and different waiting periods. We do each experiment for a set of parameters (number of hosts, waiting period X) 20 times.

Figure 5.6 shows the average number of bytes received (blue with a left arrow) and sent (red with a right arrow) every 30 seconds by the simulation from the webmail server during the experiment. The solid lines are the experiments where the scenario has a waiting period of 10 seconds between each series of simulated actions while the dotted lines are the experiments where the waiting period is 30 seconds. The simulated hosts request a variety of data for each page from the service webmail (HTML pages,

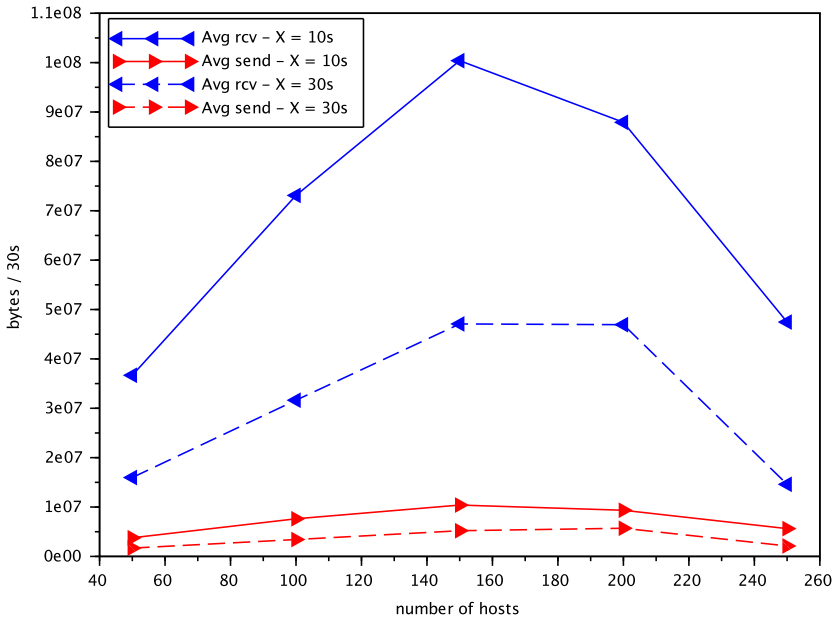

Figure 5.6 – Network traffic of the evaluation with benign traffic.

fonts, links, logos, etc.) which results in the simulation receiving a lot more data than it sends. Figure 5.6 shows that a more intensive scenario results in a significantly increased amount of exchanged data.

This figure also shows that our current implementation prototype has difficulties handling more than 150 hosts. The amount of generated data is proportional to the number of hosts simulated on the first part of the graph up to 150 hosts. Beyond that point the simulation encounters difficulties. The cause of such difficulties is still unidentified and is the target of improvement of our current prototype. It can be due to limitations of the network support (Mininet) or limitations on the service webmail that may not be able to handle so many simultaneous connection requests for a set of only five credentials. However, previous experiments tend to point a limitation on the handling number of connections (around 6000) during the experiment, which would be a limitation of our network support.

Despite that technical difficulty, we observe interesting results from this evaluation of the generation of benign data. Coincidentally, this evaluation of the generation of benign data also ends up being an evaluation of the external service of our simulation. The graph in Figure 5.6 represents four days and 20 hours of continuous activity between the simulation and the webmail server. It evaluates the capacity of the webmail server to process a consequent workload.

The experiments also reveal that our benign traffic is raising an alert on Suricata. Indeed, we deliberately chose not to use encryption in the simulated traffic to limit the issues in our evaluation. Thus it raises a Suricata alert when our simulated hosts

connect to the service with credentials in clear text. It generates one alert each time the "connect to the webmail" elementary action is simulated.

	$X = 10$s		$X = 30$s	
Nbr hosts	avg	stdev	avg	stdev
50	2433	82	1044	33
100	4864	64	2065	50
150	5611	197	3077	65
200	5078	192	3287	452
250	3091	139	1227	285

Table 5.1 – Number of "clear password" alerts from Suricata

Table 5.1 shows the average number and standard deviation of alerts generated by Suricata during the generation of benign traffic. We calculate the average and standard deviation out of the twenty experiments made for each set of parameters. These numbers are also representative of the number of sessions created with the webmail server during the experiment. The standard deviation of the number of alerts is relatively high when one or several experiments encountered difficulties, especially when the simulation reaches 150 hosts or more.

5.2.2 Evaluation with malicious traffic

After evaluating the benign data, we now focus on the malicious data. In the topology of Figure 5.4, we only launch the simulated network without starting the activity of the hosts. We then launch a scan of the service with the vulnerability scanner OpenVAS. We observe the alerts raised by Suricata in Table 5.2.

Alert ID	Live analysis	Offline analysis
2006380	1	1
2012887	6	6
2016184	2	2
2019232	780	780
2019239	260	260
2022028	1040	1040
2220007	2	2
2220018	1	1
2221002	1	1
2221007	57	57
2221013	1	1
2221014	1	1
2221015	2	2
2221016	1	1
2221018	1	1
2221028	6	6
2230010	34	34

Table 5.2 – Number of alerts from Suricata (malicious only)

Table 5.2 shows the number of alerts raised by Suricata, classified by alert ID. The alert ID 2012887 corresponds to the alert that warns about the transmission of a password in clear text, similar to the benign traffic analysis.

From Table 5.2, we can infer that the traffic solely generated by OpenVAS does not create enough stress on Suricata to generate a difference between the offline and live analysis.

5.2.3 Evaluation with mixed traffic

Lastly, we evaluate Suricata with mixed traffic generated by our simulation and OpenVAS at the same time. We start the generation of benign traffic first then, after two minutes, we start the vulnerability scan. We generate different levels of traffic intensity (number of hosts + X), and we compare the alerts raised during the live and offline analysis of Suricata. We expect the resulting alerts to be equal to the number of alerts found for the same intensity of benign traffic plus the alerts raised with the malicious traffic. To be consistent, we order OpenVAS to do the same vulnerability scan.

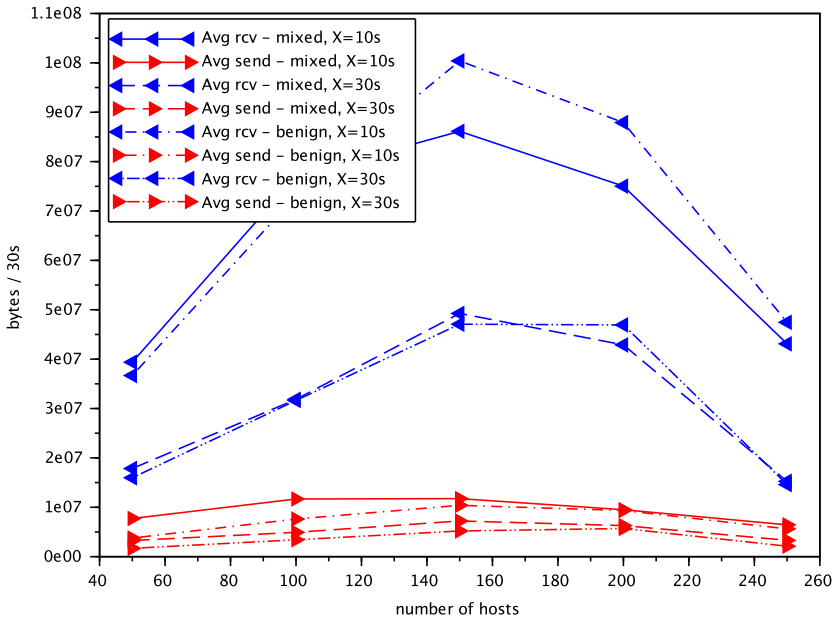

Figure 5.7 – Network traffic of the evaluation with mixed and benign traffic

Figure 5.7 presents the average number of bytes received (blue with a left arrow) and sent (red with a right arrow) every 30 seconds by the simulation from the webmail server during the experiments with mixed and benign traffic. For reference, we display the results of the benign traffic in Figure 5.6 along with the results of the mixed traffic. In this figure, the lines without dots represent the data exchanged during the mixed traffic

	50 Hosts		100 Hosts		150 Hosts		200 Hosts		250 Hosts	
Alert ID	live	off.	live	off.	live	off.	live	off.	live	off.
2006380	2	2	2	2	2	2	2	2	2	2
2012887	2460	2465	4813	4821	5526	5541	5069	5075	3044	3046
2016184	2	2	2	2	2	2	2	2	2	2
2019232	780	780	780	780	780	780	780	780	780	780
2019239	260	260	260	260	260	260	260	260	260	260
2022028	1040	1040	1040	1040	1040	1040	1040	1040	1040	1040
2220007	2	2	2	2	2	2	2	2	2	2
2220018	1	0	1	0	1	0	1	0	1	0
2221002	2	1	2	1	2	1	2	1	2	1
2221007	57	56	57	56	57	56	57	56	57	56
2221013	1	1	1	1	1	1	1	1	1	1
2221014	1	1	1	1	1	1	1	1	1	1
2221015	2	2	2	2	2	2	2	2	2	2
2221016	1	1	1	1	1	1	1	1	1	1
2221018	2	0	2	0	2	0	2	0	1	0
2221028	6	5	6	5	6	5	6	5	6	5
2230010	33	33	32	32	33	33	32	32	33	32

Table 5.3 – Number of alerts from Suricata (mixed, $X = 10$s).

and the lines with dots (single or double) are the data exchanged during the benign traffic generation. A solid line corresponds to mixed traffic with $X = 10$s, a dotted line to mixed traffic with $X = 30$s, a dotted line with single dots to benign traffic with $X = 10$s and finally a dotted line with double dots to benign traffic with $X = 30$s. As in Figure 5.6, less intense scenarios ($X = 30$s) exchange a lower number of bytes than more intense scenario ($X = 10$s). Moreover, apart from the mixed experiment launched with limit parameters (150 hosts, $X = 10$s), the behavior of the mixed experiment is close to the benign traffic experiment, as we expected.

Table 5.3 represents the average number of alerts raised by Suricata by alert ID for the most intense scenario ($X = 10$s). For each number of hosts, we show the average number of alerts raised live and offline by Suricata. For the most part, the results are pretty similar to the profile of alerts showed in Table 5.2 except for alert 2012887 that was raised both during the benign traffic and the mixed traffic. The range of variation of this specific alert matches the results obtained for the evaluation with benign traffic in Table 5.1.

However, Table 5.3 also shows that differences appear between offline and live analysis. Those differences are proof that Suricata operates slightly differently between a live analysis and an offline analysis due to the stress inflicted on the IDS during the simulation. With more substantial stress (more intensive scenario, more hosts and different configurations of Suricata) we can expose even more alerts that the IDS detects in offline analysis but misses in the live analysis.

Table 5.4 shows the preliminary results of on-going experiments with a higher intensity scenario ($X = 5$s), a more in-depth vulnerability scanning and more rules activated on Suricata. The analysis of benign traffic shows none of those alerts. In these experiments, a few interesting results appear. Alert 2200069 shows that there are attacks

			Mixed					
	OpenVAS		50 Hosts		100 Hosts		150 Hosts	
Alert ID	live	off.	live	off.	live	off.	live	off.
2200069		7395		7617		7611		7252
2200074		20		60		79		71
2230010	44	44	32	31	33	33	33	33

Table 5.4 – Number of interesting alerts from Suricata ($X = 5$s)

that Suricata does not detect during a live analysis. Alert 2200074 produces more alerts in mixed traffic than in an attack without benign traffic. On the contrary, we detect fewer alerts 2230010 and 2230015 than expected with the addition of benign traffic. The analysis of those experiments is still on-going, but an improvement of the current limitations of the prototype could allow a deeper understanding of the behavior of the IDS.

5.3 Conclusion

In this chapter, we defined a methodology for the evaluation of services and security products. We can design an experiment using our network simulation that respects all the properties of the evaluation of services and security products. However, our methodology requires some preparation efforts from the evaluator. The evaluator needs to provide model data and scenarios and must choose an appropriate data generating function. He must also make topology choice (the use of external components, selection, and composition of ground truth, actors interacting with the target, etc.) according to his goals and his evaluation target. This method is still at its initial stage. We can significantly reduce the preparation effort of the evaluator with the development of further data generating functions, tools to identify time-sensitive inputs, and the accumulation of model data.

To illustrate the proposed evaluation methodology, we present the experimental results of an evaluation of a network-based IDS. We evaluate this IDS with our network simulation using only benign traffic, only malicious traffic, and mixed traffic. After incidentally evaluating the workload processing capacity of the external service of our topology, we observed that the separate evaluation of benign traffic and malicious traffic gave slightly different results than with mixed traffic. In particular, we observe a difference in behavior between a live and offline analysis most likely due to the stress of consequent benign traffic.

However, a more advanced prototype of the simulation could provide more development of the model and the broaden scope of possible evaluations. It would also be interesting to extend the experimental results to similar security products and products of different types.

CHAPTER 6 Conclusion

We started this thesis from the observation that the evaluation of security products is a large task with multiple aspects. A complete evaluation is not only an evaluation of the properties of the security product but also of its impact on its operational environment (actors, services, other security products). As such, evaluators require tools or an environment that can test all kind of properties at an affordable operational cost. The environment must be able to concurrently evaluate functionalities with a fine comb (semantic tests) and evaluate at a scale that matches the operational context of the evaluation target (load tests).

To that end, we established the current state of the evaluation of security products and services. We identified the properties to evaluate for services (compliance with the specifications, workload processing capacity, resilience to attacks) and security products (policy accuracy, attack coverage, performance overhead, workload processing capacity). We presented the tools employed by the community for those evaluations. Load tests tools stress specific resources like workload drivers (HTTPBench, UnixBench, iozone, etc.). Other methods allow for semantic tests that verify specific functionalities or vulnerabilities like manually generating activity, developing homegrown scripts, or exploit databases. To obtain a combination of those tests and create an operational context with the targeting of specific vulnerabilities and productive use of functionalities with realistic intent, the evaluator must turn to traces from other sources (real-world production, publicly available traces or honeypots) or testbed environments that allow for the large scale deployment of semantic tests. However, all those methods can either target parts of the properties that interest us, may not be adapted to specific needs of the evaluator, may not guarantee full knowledge of its content, or requires a vast amount of resources and time. The method that ended up as the closest to our goal is a testbed environment, a method that requires many resources to set up and operate. Much of that cost is due to the network infrastructure required to support that environment, leading to few being able to afford that cost.

In this thesis, we proposed a new approach to generate evaluation data at a large scale that can rely on lower-cost network infrastructures like virtual networks that use process virtual machines. The idea is to execute a program reproduce model data to a level of realism acceptable to the evaluator. The simulation program does not reproduce the model interaction (*Elementary action*) of the actor (user, attackers or services) but solely the data of that interaction. This simulation program can then

function on a lightweight network infrastructure at a large scale. The idea is to propose a method that can exploit the advantages of its virtual network infrastructure. We presented a formal model of our approach where we defined key concepts of our model (*Data generating functions*, *Model data*, *Elementary action parameters*, etc.) along with methods to judge if the produced data was acceptable to the evaluator (levels of realism and notion of equivalent data over identical data). We imposed several requirements over our evaluation method (reproducibility, realism, adaptability, accuracy, scalability) and transformed them into verifiable properties of our model.

After proposing a formal model of our approach, we aimed to verify our model through the implementation of a prototype. The construction of the prototype requires several intermediary steps like the construction of the referential network, the definition of *Elementary actions* and capture of the *Model data*, the creation of *Data generating functions* that correspond to several levels of realism, the experimental validation of those functions, the selection of the network infrastructure, etc. In this stage of development, the preparation burden on the evaluator is quite consequent, especially in the conception of *Data generating functions* and the incorporation of *Elementary action parameters*. The identification of problematic elements to accomplish the preservation of high-level properties (for example the "acknowledgment by the service" property) is currently made by hand. We must manually inspect network traces to identify user inputs nomenclature and hidden parameters of the services (token, identifiers, cookies, etc.). However, we propose a method to deduce the parameters and try basic transformations through the comparison of several identical instances of the same *Elementary action*, then the comparison of instances with different user inputs. We believe we can implement a program to automatically deduce the vast majority of the processing of those parameters. Our prototype can currently only reproduce network data, but we presented our ideas to *Data generating functions* for the evaluation of system security products. Our method may not be suited for the evaluation of a single system security product, a regular system VM being appropriate for such task. The advantages of our method appears in the evaluation of a security system that relies on system and network security products with a supervision center.

Although the presented prototype is limited, it is enough to use for an example application to the evaluation of a security product. We first define a methodology to apply our data generation method to the evaluation of services and security products. We offer to the evaluator the architectural choice of using our method solely to evaluate their target or use external tools to reduce the preparation burden. We mostly advise connecting an external component when evaluating the attack coverage property of security product as the evaluator would have to capture the *Model data* for a large number of attacks to have a decent coverage of possible attacks. We then applied the proposed methodology in the example use case of an IDS. In the process, we conducted several experiments: the generation of solely benign traffic, solely malicious traffic and mixed traffic. We also wanted to evaluate the difference in behavior fo the IDS when stressed in a live analysis of the traffic or when analyzing offline traffic. We showed that our prototype generated enough stress on the IDS to generate a difference in analysis between live traffic and offline traffic. We also highlighted several interesting data points that showed the worth of further improving the development of our prototype. Serious

work is still required to transform our limited prototype into an effective evaluation tool. However, we showed through this thesis that our approach presents much potential and offers a level of adaptability (scale, type of simulated data, different level of realism, etc.) that is not offered by existing data generation method dedicated to the evaluation of security products and services.

List of publications

1. P.-M. Bajan, H. Debar, and C. Kiennert. A new approach of network simulation for data generation in evaluating security products. In *ICIMP 2018, The Thirteenth International Conference on Internet Monitoring and Protection, 2018.*

2. P.-M. Bajan, H. Debar, and C. Kiennert. Methodology of a network simulation in the context of an evaluation: Application to an ids. In *ICISSP 2019, The Fifth International Conference on Information Systems Security and Privacy, 2019.*

3. P.-M. Bajan, H. Debar, and C. Kiennert. Evaluating security products: Formal model and requirements of a new approach. In *International Journal On Advances in Security,* volume 12, 2019.

APPENDIX A

Taxonomy of virtualization software

In this appendix we present a taxonomy of virtualization softwares we compiled. For each software, we put together the following information:

- The name of the software
- If the software is obsolete
- The type of virtualization software:
  - a regular virtualization software: this software creates and handles virtual machines. VirtualBox is a good example.
  - a network simulator: this software creates a virtual network connecting together virtual machines. We differentiate those softwares from network emulator that virtualizes the resources of a physical network to create one or several virtual networks. We explained more that distinction in Section 2.3.
  - a network emulator
  - a model simulator: we did not mention this type of software previously as it is a type of virtualization outside our scope of interest. The type of virtual network produced is a mathematical model of the network and the software generates artificial network frames based on that model. This type of simulator is mostly used for studying and developing network protocols. It can stop the network at any time and analyze any frame exchange.
- The type of generated virtual machines: this entry is for virtualization softwares and network simulators that uses virtual machines
- The latest release date we can find of the software
- If the software is open source or a commercial product

Name	Obsolete	Software type	VM type	Last Release	Availability
Bochs		Virtualization Software	system	2017	open-source
Cloonix		Network Simulator	system	2016	open-source
Cooperative Linux	X	Virtualization Software	process	2011	open-source
CORE		Network Simulator	process	2018	open-source
CrossOver		Virtualization Software	process	2019	commercial
DOSBox		Virtualization Software	system	2018	open-source
DOSEMU	X	Virtualization Software	process	2007	open-source
Dosemu2		Virtualization Software	process	2017	open-source
Einar	X	Network Simulator	system	2006	open-source
Emulab		Network Emulator	-	2018	plateform
EVE-NG		Virtualization Software	process	2019	open-source
FAUmachine	X	Virtualization Software	system	2012	open-source
GloMoSim		Model Simulator	-	2016	open-source
GNS3		Network Simulator	system	2019	open-source
Hynesim		Network Simulator	system	2018	open-source
IMUNES		Network Simulator	process	2016	open-source
LINE Network Emulator	X	Network Simulator	process	2014	open-source
Marionnet		Network Simulator	process	2018	open-source
Mininet		Network Simulator	process	2017	open-source
MLN	X	Network Simulator	process	2009	open-source
Modelnet	X	Network Emulator	-	2005	open-source
NCTUns / Estinet	X	Network Emulator	-	2009	open-source
Netkit		Network Simulator	process	2016	open-source
NetSim		Network Emulator	-	2019	commercial
ns-2 / ns-3		Model Simulator	-	2011 / 2018	open-source
Omnet++		Model Simulator	-	2019	open-source
OFNet	X	Network Simulator	process	?	open-source
OPNET modeler		Model Simulator	-	2016	commercial
Parallels		Virtualization Software	system	2018	commercial
PearPC		Virtualization Software	system	2015	open-source
Planetlab		Network Simulator	process	?	plateform
Plex86	X	Virtualization Software	process	2003	open-source
Psimulator2		Network Simulator	process	2017	open-source
Q	X	Virtualization Software	system	2008	open-source
QEMU/KVM		Virtualization Software	system	2019	open-source
Shadow		Model Simulator	-	2018	open-source
TRANGO	X	Virtualization Software	system	?	commercial
User-Mode Linux (UML)		Virtualization Software	process	2016	open-source
UMLMON	X	Virtualization Software	process	2006	open-source
Unified Networking Lab		Network Simulator	system	2016	open-source
VDE		Network Emulator	-	2014	open-source
Virtual PC	X	Virtualization Software	system	2011	freeware
VirtualBox		Virtualization Software	system	2019	open-source
Virtuozzo (OpenVZ)		Virtualization Software	process	2019	open-source
VMware		Network Simulator	system	2019	commercial
VNUML		Network Simulator	process	2009	open-source
Win4Lin	X	Virtualization Software	system	2008	commercial
Wine		Virtualization Software	process	2019	open-source
XtratuM		Virtualization Software	system	2017	open-source
Xen		Virtualization Software	system	2019	open-source

Figure A.1 – Taxonomy of virtualization softwares

Bibliography

[Ahrenholz et al. 2008] J. Ahrenholz, C. Danilov, T. R. Henderson, and J. H. Kim. Core: A real-time network emulator. In *MILCOM 2008-2008 IEEE Military Communications Conference*, pages 1–7, 2008. IEEE.

[Alhazmi and Malaiya 2006] O. H. Alhazmi and Y. K. Malaiya. Measuring and enhancing prediction capabilities of vulnerability discovery models for apache and iis http servers. In *Software Reliability Engineering, 2006. ISSRE'06. 17th International Symposium on*, pages 343–352, 2006. IEEE.

[Axelsson 2000] S. Axelsson. The base-rate fallacy and the difficulty of intrusion detection. *ACM Transactions on Information and System Security (TISSEC)*, 3(3):186–205, 2000.

[Bajan et al. 2018] P.-M. Bajan, H. Debar, and C. Kiennert. A new approach of network simulation for data generation in evaluating security products. In *ICIMP 2018, The Thirteenth International Conference on Internet Monitoring and Protection*, 2018.

[Bajan et al. 2019] P.-M. Bajan, H. Debar, and C. Kiennert. Methodology of a network simulation in the context of an evaluation: Application to an ids. In *ICISSP 2019, The Fifth International Conference on Information Systems Security and Privacy*, 2019.

[Begnum 2006] K. M. Begnum. Managing large networks of virtual machines. In *LISA*, volume 6, pages 205–214, 2006.

[Bhuyan et al. 2015] M. H. Bhuyan, D. K. Bhattacharyya, and J. K. Kalita. Towards generating real-life datasets for network intrusion detection. *IJ Network Security*, 17(6):683–701, 2015.

[Brown et al. 2009] C. Brown, A. Cowperthwaite, A. Hijazi, and A. Somayaji. Analysis of the 1999 darpa/lincoln laboratory ids evaluation data with netadhict. In *2009 IEEE Symposium on Computational Intelligence for Security and Defense Applications*, pages 1–7, 2009. IEEE.

[Chowdhury and Boutaba 2010] N. M. K. Chowdhury and R. Boutaba. A survey of network virtualization. *Computer Networks*, 54(5):862–876, 2010.

[Coull et al. 2007] S. E. Coull, C. V. Wright, F. Monrose, M. P. Collins, M. K. Reiter, et al. Playing devil's advocate: Inferring sensitive information from anonymized network traces. In *NDSS*, volume 7, pages 35–47, 2007.

[Cunningham et al. 1999] MASSACHUSETTS INST OF TECH LEXINGTON LINCOLN LAB. Evaluating intrusion detection systems without attacking your friends: The 1998 darpa intrusion detection evaluation. Technical report, 1999.

[Davoli 2005] R. Davoli. Vde: Virtual distributed ethernet. In *First International Conference on Testbeds and Research Infrastructures for the Development of Networks and Communities*, pages 213–220, 2005. IEEE.

[De Oliveira et al. 2014] R. L. S. De Oliveira, C. M. Schweitzer, A. A. Shinoda, and L. R. Prete. Using mininet for emulation and prototyping software-defined networks. In *2014 IEEE Colombian Conference on Communications and Computing (COLCOM)*, pages 1–6, 2014. IEEE.

[Fonseca et al. 2014] J. Fonseca, M. Vieira, and H. Madeira. Evaluation of web security mechanisms using vulnerability and attack injection. *IEEE Transactions on Dependable and Secure Computing*, (1):1, 2014.

[Garcia-Alfaro et al. 2011] J. Garcia-Alfaro, F. Cuppens, N. Cuppens-Boulahia, and S. Preda. Mirage: a management tool for the analysis and deployment of network security policies. In *Data Privacy Management and Autonomous Spontaneous Security*, pages 203–215. Springer, 2011.

[Gharib et al. 2016] A. Gharib, I. Sharafaldin, A. H. Lashkari, and A. A. Ghorbani. An evaluation framework for intrusion detection dataset. In *2016 International Conference on Information Science and Security (ICISS)*, pages 1–6, 2016. IEEE.

[Gogolla and Hilken 2016] M. Gogolla and F. Hilken. Model validation and verification options in a contemporary uml and ocl analysis tool. *Modellierung 2016*, 2016.

[Henning 2006] J. L. Henning. Spec cpu2006 benchmark descriptions. *ACM SIGARCH Computer Architecture News*, 34(4):1–17, 2006.

[Houidi et al. 2009] I. Houidi, W. Louati, D. Zeghlache, and S. Baucke. Virtual resource description and clustering for virtual network discovery. 2009.

[Jin et al. 2013] H. Jin, G. Xiang, D. Zou, S. Wu, F. Zhao, M. Li, and W. Zheng. A vmm-based intrusion prevention system in cloud computing environment. *The Journal of Supercomputing*, 66(3):1133–1151, 2013.

[Kainda et al. 2010] R. Kainda, I. Flechais, and A. Roscoe. Security and usability: Analysis and evaluation. In *2010 International Conference on Availability, Reliability and Security*, pages 275–282, 2010. IEEE.

[Loddo and Saiu 2008] J.-V. Loddo and L. Saiu. Marionnet: a virtual network laboratory and simulation tool. In *First International Conference on Simulation Tools and Techniques for Communications, Networks and Systems*, 2008.

[Lombardi and Di Pietro 2011] F. Lombardi and R. Di Pietro. Secure virtualization for cloud computing. *Journal of network and computer applications*, 34(4):1113–1122, 2011.

[McHugh 2000] J. McHugh. Testing intrusion detection systems: A critique of the 1998 and 1999 darpa intrusion detection system evaluations as performed by lincoln laboratory. *ACM Trans. Inf. Syst. Secur.*, 3(4):262–294, November 2000.

[Mell et al. 2003] P. Mell, V. Hu, R. Lippmann, J. Haines, and M. Zissman. An overview of issues in testing intrusion detection systems. 2003.

[Migault et al. 2010] D. Migault, C. Girard, and M. Laurent. A performance view on dnssec migration. In *Network and Service Management (CNSM), 2010 International Conference on*, pages 469–474, 2010. IEEE.

[Milenkoski et al. 2015] A. Milenkoski, M. Vieira, S. Kounev, A. Avritzer, and B. D. Payne. Evaluating computer intrusion detection systems: A survey of common practices. *ACM Comput. Surv.*, 48(1):12:1–12:41, September 2015.

[Ming et al. 2017] J. Ming, Z. Xin, P. Lan, D. Wu, P. Liu, and B. Mao. Impeding behavior-based malware analysis via replacement attacks to malware specifications. *Journal of Computer Virology and Hacking Techniques*, 13(3):193–207, 2017.

[Moustafa and Slay 2016] N. Moustafa and J. Slay. The evaluation of network anomaly detection systems: Statistical analysis of the unsw-nb15 data set and the comparison with the kdd99 data set. *Information Security Journal: A Global Perspective*, 25(1-3):18–31, 2016.

[Nahum et al. 2007] E. M. Nahum, J. Tracey, and C. P. Wright. Evaluating sip server performance. In *ACM SIGMETRICS Performance Evaluation Review*, volume 35, pages 349–350, 2007. ACM.

[Nechaev et al. 2004] B. Nechaev, M. Allman, V. Paxson, and A. Gurtov. Lawrence berkeley national laboratory (lbnl)/icsi enterprise tracing project. *Berkeley, CA: LBNL/ICSI*, 2004.

[Nehinbe 2009] J. O. Nehinbe. A simple method for improving intrusion detections in corporate networks. In *International Conference on Information Security and Digital Forensics*, pages 111–122, 2009. Springer.

[Peach et al. 2016] S. Peach, B. Irwin, and R. Van Heerden. An overview of linux container based network emulation. In *15th European Conf. Cyber Warfare and Security*, pages 253–259, 2016.

[Prigent et al. 2005] G. Prigent, F. Harrouet, J. Tisseau, and F. Paul. Simulation hybride de la sécurité des systemes d,Äôinformation. 2005.

[Puljiz and Mikuc 2006] Z. Puljiz and M. Mikuc. Imunes based distributed network emulator. In *Software in Telecommunications and Computer Networks, 2006. SoftCOM 2006. International Conference on*, pages 198–203, 2006. IEEE.

[Reeves et al. 2012] J. Reeves, A. Ramaswamy, M. Locasto, S. Bratus, and S. Smith. Intrusion detection for resource-constrained embedded control systems in the power grid. *International Journal of Critical Infrastructure Protection*, 5(2):74–83, 2012.

[Riley et al. 2008] R. Riley, X. Jiang, and D. Xu. Guest-transparent prevention of kernel rootkits with vmm-based memory shadowing. In *International Workshop on Recent Advances in Intrusion Detection*, pages 1–20, 2008. Springer.

[Rimondini 2007] M. Rimondini. Emulation of computer networks with netkit. *Dipartimento di Informatica e Automazione, Roma Tre University, http://www. netkit. org/, RT-DIA-113-2007*, 2007.

[Seeberg and Petrovic 2007] V. E. Seeberg and S. Petrovic. A new classification scheme for anonymization of real data used in ids benchmarking. In *Availability, Reliability and Security, 2007. ARES 2007. The Second International Conference on*, pages 385–390, 2007. IEEE.

[Sharafaldin et al. 2018] I. Sharafaldin, A. H. Lashkari, and A. A. Ghorbani. Toward generating a new intrusion detection dataset and intrusion traffic characterization. In *ICISSP*, pages 108–116, 2018.

[Shiravi et al. 2012] A. Shiravi, H. Shiravi, M. Tavallaee, and A. A. Ghorbani. Toward developing a systematic approach to generate benchmark datasets for intrusion detection. *computers & security*, 31(3):357–374, 2012.

[Smith and Nair 2005] J. E. Smith and R. Nair. The architecture of virtual machines. *Computer*, 38(5):32–38, 2005.

[Srivastava et al. 2008] Georgia Institute of Technology. Secure observation of kernel behavior. Technical report, 2008.

[Vasilomanolakis et al. 2016] E. Vasilomanolakis, C. G. Cordero, N. Milanov, and M. Mühlhäuser. Towards the creation of synthetic, yet realistic, intrusion detection datasets. In *NOMS 2016-2016 IEEE/IFIP Network Operations and Management Symposium*, pages 1209–1214, 2016. IEEE.

[Vlasyuk et al. 2016] G. Vlasyuk, O. Starkova, K. Herasymenko, and M. Zemlianyi. Approaches and algorithms of virtual telecommunication networks analysis in unetlab environment. In *2016 Third International Scientific-Practical Conference Problems of Infocommunications Science and Technology (PIC S&T)*, pages 181–184, 2016. IEEE.

[Welsh 2013] C. Welsh. *GNS3 network simulation guide*. Packt Publ., 2013.

[Yavanoglu and Aydos 2017] O. Yavanoglu and M. Aydos. A review on cyber security datasets for machine learning algorithms. In *2017 IEEE International Conference on Big Data (Big Data)*, pages 2186–2193, 2017. IEEE.

[Zec 2003] M. Zec. Implementing a clonable network stack in the freebsd kernel. In *USENIX Annual Technical Conference, FREENIX Track*, pages 137–150, 2003.

Titre : Simulation d'activités et d'attaques : application à la cyberdéfense

Mots clés : simulation, évaluation, méthodologie, cybersécurité

Résumé : L'évaluation de produits de sécurité est un enjeu crucial de la cybersécurité. De nombreux produits et méthodes existent pour les propriétés des services (conformité aux spécifications, traitement de la charge et résistance aux attaques) et des produits de sécurité (justesse de la décision, variété d'attaques supportées, impact sur les performances et traitement de la charge).
La plupart des méthodes existantes ne peuvent évaluer qu'une partie de ces propriétés. Les méthodes pouvant couvrir toutes ces propriétés, comme les bancs de tests, nécessitent un fort coût de ressources et main d'oeuvre. Peu de structures peuvent se permettre de déployer et maintenir des bancs de tests complets avec les outils actuels. Dans cette thèse, nous proposons une nouvelle approche pour générer des données d'évaluation à grande échelle en respectant les exigences et besoins de l'évaluateur.
Notre méthode est basée sur le déploiement d'un simple programme capable de reproduire des données modèles sur un réseau virtuel léger. Les exigences de l'évaluateur sont traduites en différents niveaux de réalisme correspondant à la préservation de différentes caractéristiques de la donnée modèle sur la donnée simulée.
Nous présentons en détails le formalisme de notre méthode et imposons des critères d'exigences (adaptabilité, reproductibilité, réalisme, précision et passage à l'échelle) sur notre méthode. Nous expliquons également les étapes du développement d'un prototype de cette méthode et les validations expérimentales de nos exigences.
Bien que les fonctionnalités du prototype présentées soient limitées, nous pouvons néanmoins utiliser ce prototype pour faire une première évaluation d'un produit de sécurité. Nous introduisons d'abord une méthodologie pour évaluer des services et produits de sécurité avec notre méthode puis nous faisons une série d'expérimentations pour évaluer un outil de détection d'intrusion.
Cette évaluation nous permet de souligner l'intérêt et les avantages de notre méthode mais également de présenter les limitations actuelles de notre prototype. Nous proposons également un ensemble d'axes d'amélioration pour développer notre prototype en un outil d'évaluation efficace.

Title : Simulation of activities and attacks: application to cyberdefense

Keywords : simulation, evaluation, methodology, cybersecurity

Abstract : The evaluation of security products is a key issue in cybersecurity. Numerous tools and methods can evaluate the properties of services (compliance with the specifications, workload processing capacity, resilience to attacks) and security products (policy accuracy, attack coverage, performances overhead, workload processing capacity).
Most existing methods only evaluate some of those properties. Methods, like testbed environments, that can cover all aspects are costly in resources and manpower. Few structures can afford the deployment and maintenance of those testbed environments. In this thesis, we propose a new method to generate at a large scale evaluation data that match the evaluator's evaluation requirements.
We base our method on the deployment of a small program on a lightweight virtual network. That program reproduces model data according to the need of the evaluator. Those needs are translated into levels of realism. Those levels match the characteristics of the model data preserved by the simulation program.
We formally present our method and introduce additional requirements (customization, reproducibility, realism, accuracy, scalability) as properties of our model. We also explain the step by step construction of our prototype along with the experimental validation of our method.
Although our prototype's functions are currently limited, we can still use our prototype to evaluate a security product. We first introduce a methodology to apply our method to the evaluation of services and security products. We then conduct a series of experiments according to the methodology to evaluate an intrusion detection system.
Our evaluation of an intrusion detection system illustrates the advantages of our method but it also underline the current limitation of our prototype. We propose a series of improvements and development to conduct to transform our current limited prototype into an efficient evaluation tool that can evaluate services and security products alike.

Université Paris-Saclay
Espace Technologique / Immeuble Discovery
Route de l'Orme aux Merisiers RD 128 / 91190 Saint-Aubin, France