Evaluating Security Products:

Formal Model and Requirements of a New Approach

Pierre-Marie Bajan* Christophe Kiennert† Hervé Debar†
*Université Paris-Saclay and Institut de Recherche Technologique SystemX (Saclay, France),
Email: {first.last}@irt-systemx.fr
†Télécom SudParis (Evry, France),
Email: {first.last}@telecom-sudparis.eu

***Abstract*—In a previous paper, we presented a new method to generate evaluation data for the evaluation of security products and services. That approach tackles the issues of producing a workload with a rich semantic at a large scale. Testbed environments are the most appropriate tool for such task but induce a lot of effort and costs to implement. We presented a model to produce semantic data that can be implemented on light virtual networks and thus deployed at a large scale. This paper is an extension of our complete formal model. In this extension, we identify additional requirements for our model and define our ambitions. We translate those ambitions in verifiable properties of our model. Our prototype, although currently limited, provides the basis for an evaluation method that is customizable, reproducible, realistic, accurate and scalable. We generate realistic activity for up to 250 simulated users interacting with a real-world webmail server in an experiment to verify the properties of our model.**

Keywords–cybersecurity; simulation; evaluation; formal method.

I. Introduction

Security products are composed of services and products designed to protect a service, machine or network against attacks. Like other products, they must be tested to guarantee adherence to specifications. In a previous paper [1] published at ICIMP 2018, we divided evaluation tests into two categories: *semantic tests* – tests of capability that require data with a high-level of semantic; and *load tests* – tests that subject the product to a large workload.

With current testing methods [2], *load tests* are semantically poor, thus not realistic. Meanwhile, *semantic tests* either require vast amount of resources to reach large scales (e.g., testbed environments), or rely on real life captures with their own set of challenges (e.g., elaboration of the ground truth and privacy concerns). Moreover, a complete evaluation of a security product tests different properties of the product. Thus the evaluator needs to select different methods with the right granularities. The granularity of interactions of the data corresponds to the level of control or precision of the data. For an evaluator, the right granularity for a testing method is a granularity that is fine enough to test specific vulnerabilities or properties. A granularity too large does not match the need of the evaluator and a granularity too fine may result in a drastic increase of the preparation burden of the evaluator for little to no improvement of the results. Rather than relying on several methods with different granularities, we aim to elaborate a method to produce data with a customizable granularity and the possibility to achieve large scale generation with appropriate semantic.

In this paper, we present a methodology to produce simulated evaluation data with different granularities independently of the network support. To achieve variable granularity of our model, we formally presents the concepts of our simulation and define properties that must be respected and verify. We also extract five requirements from our analysis of existing methods to determine the criteria of an ideal method: customizable, reproducible, realistic, accurate and scalable. To ensure that the method we propose is as ideal as possible, we convert those requirements in additional properties of our method. We then ensure that the developed prototype of our method respect all the raised properties in a series of experiments.

The remainder of this paper is organized as follows. Section II reviews the related work on the production of evaluation data and their limits. Section III is our analysis of current methods compiled into five criteria for an data generating method. Section IV defines the concepts of our methodology and uses those concepts to introduce our model. Section V explains the different choices we made for the implementation of our prototype and shows the experiment results to validate our model. Finally, we conclude our work in Section VI.

II. Related work

We first present existing related to the generation of semantic tests.

A. Semantic tests

Semantic tests generate evaluation data with high semantic value. Their goal is to generate realistic workloads to produce real-life reactions of the security product or to test specific functionalities and vulnerabilities of the product. There are two approaches to those tests: take data from the real world (the highest level of semantic), or execute specialized tools and homegrown scripts.

Real world data can come from several sources: provided by real world organizations, or obtained from honeypots where attackers were tricked into interacting with a recording system

to learn about the current trends (ex: generation of intrusion detection signatures using a honeypot [3]). However, the evaluator does not have a complete knowledge of the content of the data. Some of them can be misidentified or the intent behind some actions misinterpreted. Moreover, real world data are difficult to obtain. Organizations are reluctant to provide data that can damage their activity, and data anonymization has the drawback of deleting relevant information (e.g., challenges of anonymization [4] and desanonymization techniques [5]). As for honeypots, the evaluator can never know beforehand the amount of data he can obtain or what kind of data he will gather.

Another way to obtain high semantic data is to generate them according to a defined scenario, relying on tools and scripts to produce specific and calibrated data. Those scripts can be homegrown scripts, exploits, or software testing scripts that try every function of a software to validate its specifications. Manually generating the data (e.g., video transcoding [6], file copy operations [7], compiling the Linux kernel [8], etc.) offers the greatest control over the interactions inside the data, but the automation of the activity generated through scripts with tools like exploit databases (Metasploit [9], Nikto [10], w3af [11], Nessus [12]) also offers good control. However, those methods are quite time-consuming or require in-depth knowledge of the evaluated product. Moreover, the granularity of control for varied for each tool and may not be appropriate for the evaluator.

B. Load tests

Load tests create stress on the tested product [13]. The most common tests use workload drivers like SPEC CPU2000 [8], ApacheBench [14] [15], iozone [15], LMBench [16] [8], etc. They produce a customizable workload with a specific intensity. The evaluator can also manually start tasks or processes known to stimulate particular resources (e.g., kernel compilation [14] [16], files download [17], or execution of Linux commands [17]). Those methods are designed to test particular resources of a system (like I/O, CPU and memory consumption) or produce large amount of workload of a specific protocol. For example, SPEC CPU 2017 generates CPU-intensive workloads while ApacheBench generates intensive HTTP workloads. However, the semantics of the workloads are low: the generated data are characteristic of the driver used and do not closely resemble real life data.

C. Deployment of scalable semantic tests

Evaluators prefer tests that are both intensive and with a high semantic, as the performance of security products like intrusion detection systems often deteriorate at high levels of activity [18]. The prefered method is to deploy semantic tests at a large scale. However, the deployment of tests with a realistic semantic at a large scale provides its own challenge.

Semantic tests are deployed on a large scale network support like a testbed environment where a large amount of resources and contributors are gathered to create a large-scale test. Evaluators must either have access to a testbed environment with enough resources to deploy large-scale experiments or use the results of other organizations that conducted large-scale experiments and made their data publicly available for the scientific community (DARPA/KDD-99 [19], CAIDA [20], DEFCON [21], MawiLab [22], etc.).

However, publicly available datasets, on top of often containing errors [2], are not designed for the specific needs of each evaluator. The evaluator needs to have an in-depth knowledge of the characteristics of the activities recorded in the dataset to avoid having an incorrect interpretation of the results of studies using those datasets. Finally, there is the issue of freshness of the traces. Those large-scale experiments produce one-time datasets that are quickly outdated.

III. Our analysis

From the analysis of existing method, we took into consideration the strengths and weaknesses of each method and came to the conclusion that, among the current existing methods, testbed environments are the best tool available for an evaluator to properly evaluate products. They allow the evaluator to have a single evaluation environment that can perform semantic tests that doubles as load tests. The large scale of the environments allows the evaluator to use a wide range of tools in a single experiment. With a proper activity model to conduct the evaluation, the evaluator can have an evaluation traffic close to reality with full control over parameters like intensity of the workflow or the introduction of incidents. The evaluator has full knowledge of the activity of the simulation and does not have to face issues from real world data like anonymisation.

However, testbeds environments have a serious flaw: the prohibiting cost of setting up, maintaining and generating large-scale experiments. We analyzed the reasons of that cost to find a way to make evaluation data with the same semantic richness than testbed environments more easily available. We came to the conclusion that one of the main culprit of the cost of such environments is the network infrastructures.

In this section we offer our analysis on how to reduce the cost of network infrastructure by crafting a method compatible with lower-end network simulator. We also share the analysis of the criteria of an ideal method to generate evaluation data. During our study of related work, we observed that the strengths and and weaknesses of all the methods mostly revolved over five major requirements: customization, reproducibility, realism, accuracy and scalability. We explain in this section those requirements in further details.

A. Using process virtual machines

One of the reasons for that cost is that the network infrastructures for testbed environments are composed of either physical machines or virtual machines (VMs) that create simulated hardware upon which the evaluator installs a complete operating system. There are virtual network infrastructures that use lighter VMs that do not simulate the hardware but only primary functions of the operating system (Wine, Mininet,

IMUNES, etc.). However, evaluators do not use those network infrastructures because those light VMs do not support a wide range of real-world application and programs.

For example, IMUNES generates cloned kernels coupled with Jail in FreeBSD systems to use them as virtual machines [23]. Those cloned kernels do not handle a graphical interface and cannot launch a browser. Thus, using this type of virtual network for a testbed environment seriously limits the possible actions taken by the VMs, even if a lot more cost-effective for large scale virtual network.

Rather than working on improving the virtualization tools, we decided to take another approach on the issue and to improve the method to generate evaluation data with a semantic richness. We propose a new production method that generates controlled activity data from short traces independently of the network support. This method consists in a single program reproducing the simulation data of a large variety of programs with a level of quality sufficient for an evaluator need. This program does not complete the tasks of the real-world programs but only reproduce model data of such programs. Even if the reproduced data are not perfect, they must at least comply with the evaluation requirements of the evaluator.

Thus, rather than deploying VMs that can handle commonly used real-world programs, we can deploy a large number of light VMs that can run our program producing evaluation data from model data. It can be implemented on a testbed environment or on a network support with lower requirements like a lower end network simulator. We also want our method to meet the need of evaluators to generate tests with a rich variety (different systems, properties of the data, etc.) and to devise hybrid tests, both semantic and load oriented.

B. Essential requirements for an data generation method

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

In a cross analysis of what was presented as strengths or weaknesses of other methods, we highlight five goals, or requirements for an ideal method, that represent the most relevant aspect of the current state of evaluation data method.

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 a victim 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 method to be able to provide realistic evaluation data to the level of publicly available traces provided with a realistic activity model.
- **Accurate**: one of the main weakness of traces obtained from real-world production and high level 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 requirements are the ambition of our evaluation data method. Even if our method ends up not being as ideal, we want our approach to respect as much as possible those requirements.

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.

IV. DATA PRODUCTION METHOD MODEL

In this section, we present a formal model for our data production method. A formal representation of our model allows us to present a generic approach that is not restrained to a single implementation. Network simulators do not use similar simulation techniques, even among the same category of network simulators. The support of different network infrastructures provides different strengths that may be of interest to the evaluators. In the same way, implementation choices can also impact the properties and capacity of our model. A formal model presents a generic model that allows for different implementation approaches. 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 four 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 add several properties to our model to ensure to respect as closely as possible the criteria of an ideal method.

A. Concepts and definitions

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

The goal of *Elementary actions* is to divide the activity we simulate in 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. Therefore, 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 short action like "connection to service S" or "adding an entry to service S'".

For each *Elementary action*, we acquire *Model data* that are the captured data of the execution of this *Elementary action* during real activity (activity not issued from our simulation). *Model data* take different forms (traces, logs, values, etc.) according to the nature of the data the evaluation target can handle.

Furthermore, the evaluator can classify the *Model data*. The evaluator uses 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. For example, the evaluator can create two classes of *Model data* to represent malicious activity and benign activity, respectively. 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 to a *Data generating function*.

2) Data generating function: We 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*.

The properties of the *Simulation data* and *Model data* represent the level of realism as seen by the evaluator. They correspond to a set of specific features of the data, either qualitative or quantitative. 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. and they represent the level of realism chosen by the evaluator.

Our definition of *Data generating function* does not include 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 have the same properties.

$$d_A^{activity} \sim d'^{activity}_A$$
$$\Longleftrightarrow$$
$$Properties(d_A^{activity}) = Properties(d'^{activity}_A)$$

It is important to specify that two data are only equivalent in the eyes of evaluator. 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 also means that two equivalent data are not necessarily identical. For example, if the evaluator is only interested in the volume of the traffic and the size of the packets, two packets with the same size but different content will be equivalent.

A *Data generating function* can produce *Simulation data* that are not identical but equivalent for the same inputs but executed at different times. For instance, if the generation of the *Simulation data* includes the addition of randomly generated parameters like tokens. However, we do not consider times as an input of our formalism of *Data genration functions*. We do not include time as an input because, although time can impact if the *Simulation data* are identical, it does not impact the equivalence of *Simulation data*.

The evaluator selects a set of *Elementary actions* to decide the finesse 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 1: 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.

$$\forall\ p^A, \forall\ d \in \mathcal{D}\ and\ f\ /f(d, p^A) = d^{simulation}$$
$$f\ is\ useful\ to\ \mathcal{D} \Leftrightarrow\ \forall d \in \mathcal{D}, d \sim d^{simulation}$$

The evaluator can select the *Data generating function* among a pool of *Data generating functions* with *Simulation parameters* ($p^{simulation}$). The evaluator chooses the function that is useful to his *Model data* and 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* (f) to produce *Simulation data* that can impact positively the usefulness of the *Data generating function*.

$$\forall p^A, \quad \left.\begin{array}{l} f(d_A, p^A) \; is \; useful \; to \; \mathcal{D} \\ f(d_A, \emptyset) \; is \; useful \; to \; \mathcal{D}' \end{array}\right\} \Rightarrow \mathcal{D}' \subseteq \mathcal{D}$$

For example, we take a *Data generating function* that preserves the property "acknowledgement of the data by the *Service*" of traces given to it. That function, in order to be useful to the dataset that has the same property, will change some time-sensitive information on the input data like a cookie or a session ID. However, different *Services* may mark this 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 relying on a different *Data generating function* for every service due to those insignificant differences, we want to parameter 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 being force to provide 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*.

The *Data generating functions* are controlled and fed by a simulation control program. This program gives order to virtual hosts to execute specific *Data generating functions* with specific inputs. The orders are issued on a separate virtual network. The resulting data exchanged between the simulation control program and the virtual hosts are called *Control data*.

To sum up, *Data generating functions* are selected with *Simulation parameters* by the evaluator for the properties they preserve and the evaluator adapts or controls the *Simulation data* with *Elementary action parameters*. The data exchanged by the program that controls the simulation and the *Host* that runs a *Data generating function* is the *Control data* ($d^{control}$) and is 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) Scenario and Scripts: A *Script* is the representation of a realistic behavior of a *Host*. We define a *Script* as an ordered set of *Elementary actions* coupled with *Elementary action parameters*. A *Script* ($Script_H$) is defined for each individual *Host* and describes the activity it must generate during the simulation. The set of defined *Scripts* is called the *Scenario* (Sce) of the simulation.

A *Script* can be represented as a graph of actions, as illustrated in Figure1.

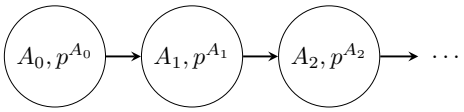

Figure 1. Example of a *Script*

B. Our model

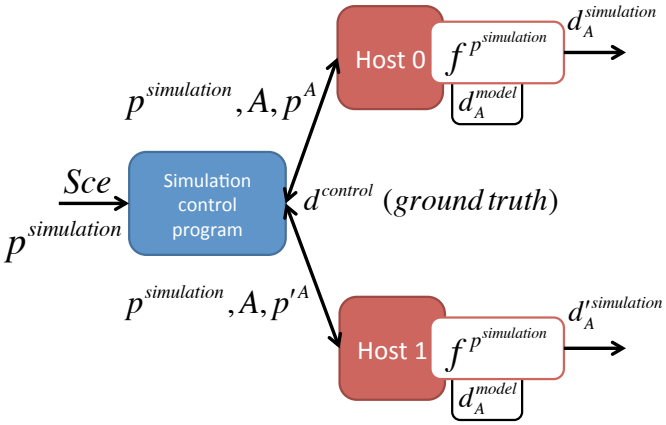

Figure 2. Generation of simulated activity from short traces

Figure 2 is a representation of our model. In that figure, the evaluator provides the simulation control program with the *Simulation parameters* ($p^{simulation}$) and the *Scenario* (Sce):

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

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* and executes the *Data generating function* (f) selected in the *Simulation parameters*. That function produces *Simulation data*, 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:

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

However, while those two *Simulation data* may not be equal, they are equivalent.

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 some *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 generates 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 for instance not only packet size is preserved but also data acknowledgment, requires to consider three main aspects:

- **typing**: 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 inputs in the *Model data* with a high semantic value for the simulation.

C. Formalisation of data generation method requirements

Here, 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.

1) Reproducibility: Reproducibility concerns two different aspects: an experiment carried out several times in the same condition must produce the same results, and a previous experiment should not impact a new experiment – which comes down to 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 as a property on *Data generating functions*:

Property 2 (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 any instant is equivalent to any *Simulation data* previously produced with the same input data.

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

$$f\ is\ reproducible \ \Leftrightarrow\ d^{simulation} \sim d'^{simulation}$$

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 of 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.

2) Realism: The realism of our simulation depends on two factors:

- The *Data generating function* and *Model data*: the *Simulation data* produced are only as realistic as the *Data generating function* and the input data allow it. The verification property of the Data generating function (cf. Property 1) 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 the evaluator considers 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 3 (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 what 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 produced *Simulation data* will present a volumetric network activity as realistic as the *Scenario* of the simulation and the *Model data*.

3) Adaptability: The evaluator can adapt a simulation on our model with several aspects.

Firstly, the *Scenario*. 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 more atomic the *Elementary actions* are, the more precisely the evaluator can control the simulation and create *Scenarios* adapted to his needs.

Secondly, the *Elementary action parameters*. They 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.).

Lastly, the *Data generating functions*. 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 IV-C2. 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.

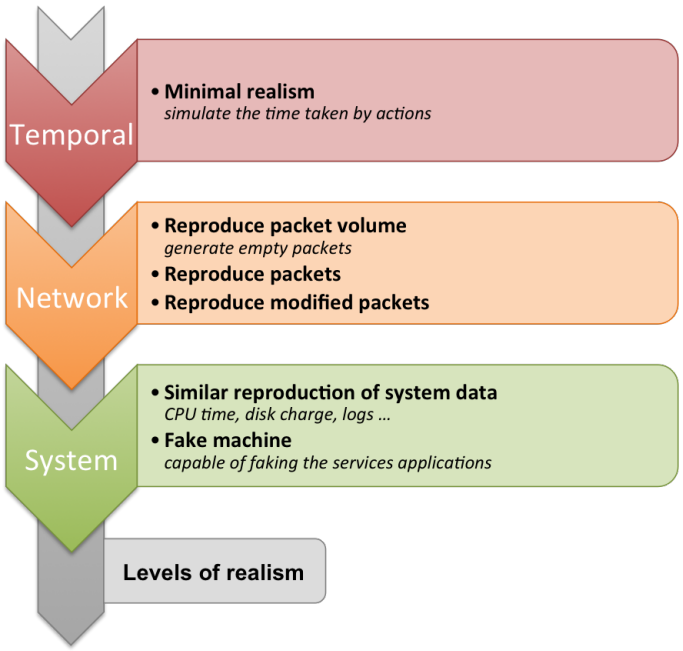

Figure 3. Levels of realism

In Figure 3, we exhibit 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 of 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 imagined 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 last 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 **Fake machine**.

All those different levels of realism correspond to different *Data generating functions* that the evaluator can select for his 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.

To be noted, the nature of the *Model data* described in each type of level of realism is different. **Temporal** uses measure values as *Model data*, **Network** has 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.

4) 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 to do it and what the result was (success, failure, error, etc.).

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

Property 4 (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* an *Host* took
- 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 risk 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 and can label the *Simulation data* produced. While our model also receives data, it does not know if the data was correctly processed by external components nor does it have 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 this 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 at the charge of the evaluator.

5) 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.

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

$$d_{H_0}^{simulation} \cup d_{H_1}^{simulation} \neq d_{H_0}^{simulation} * 2$$

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 1:

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

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

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

  $$\Rightarrow \quad d_{H_0}^{simulation} \sim d\ and\ d_{H_1}^{simulation} \sim d$$

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

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

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

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

$$\iff d_{H_0}^{simulation} \sim d_{H_1}^{simulation}$$

- So, we have:

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

$$\Rightarrow \quad d_{H_0}^{simulation} \sim d_{H_1}^{simulation}$$

We also 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:

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

$$d_{H_0}^{simulation} \sim d_{H_1}^{simulation} \; and \; d_{H_0}^{simulation} \neq d_{H_1}^{simulation}$$

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

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

$$d_{H_0}^{simulation} \cup d_{H_1}^{simulation} \neq d_{H_0}^{simulation} * 2$$

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 requirements on the network support:

Property 5 (Implementation requirement 2): The network infrastructure supporting the simulation model must be favorable to 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. We detail the implementation of a prototype in the following section.

V. Implementation of the proposed method

In this section, we describe the implemention of a prototype that follows the requirements of our model. This prototype uses Mininet [24] as the network support of our simulation. Mininet is an open-source network simulator that deploys lightweight virtual machines to create virtual networks, and able to create hundreds of lightweight virtual machines in a short amount of time.

A. Model of the prototype

Our prototype contains several *Data generating functions* that preserve each of these properties: execution time, packet size, acknowledgement of the data by the *Service*. Based on these *Data generating functions* we simulate the activity of 50 to 200 *Hosts* representing regular employees of a small company interacting with the *Service* of a webmail server Roundcube on a Postfix mail server. A simulation control program follows the *Script* described in Figure 4 for all the *Hosts* of the simulation. In Figure 4, the *Elementary actions* are in italics 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. The intensity of the *Script* can be modulated by modifying the value of X.

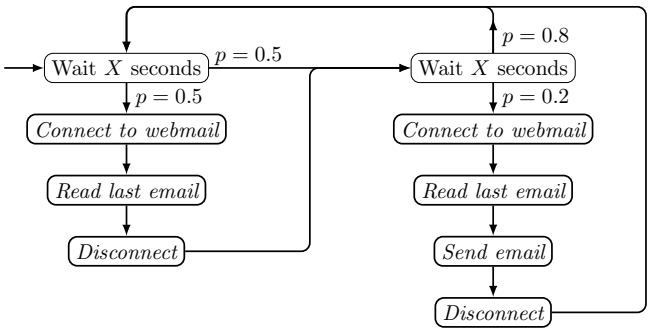

Figure 4. Generation of simulated activity from short traces

To make sure that our method improves the existing methods, we must verify that our implementation respects the properties we highlighted:

- the verification property of *Data generating functions* (c.f. Property 1)
- the reproducibility property of *Data generating functions* (c.f. Property 2)
- the verification property of the *Scenario* (c.f. Property 3)
- the implementation requirement on *Control data* (c.f. Property 4)
- the implementation requirement for scalability (c.f. Property 5)

TABLE I. NUMBER OF LINES IN THE WEBMAIL LOG FILES.

	5 VMs		5 Hosts		50 Hosts		100 Hosts		150 Hosts		200 Hosts		250 Hosts	
Filenames	**avg**	**stdev**	**avg**	**stdev**	**avg**	**stdev**	**avg**	**stdev**	**avg**	**stdev**	**avg**	**stdev**	**avg**	**stdev**
userlogins	90	9	112	10	1032	36	2084	45	3085	52	4121	53	5118	74
imap	43245	5070	57775	5306	487883	22742	984642	28820	1450507	27792	1933823	21117	274825	235985
sql	4955	525	6703	563	56081	1886	113031	2452	167138	2964	223427	2906	265354	4688

B. Verification of the implementation requirements

The verification of the implementation requirements (Properties 4 and 5) are the easiest to verify.

We select the network simulator Mininet in accordance to Property 5: it uses lightweight virtual machines and can create a large amount of virtual hosts (around a thousand) connected with virtual links in a matter of 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 the Property 4, we simply ask the simulation control program to write in a file the information required by that property.

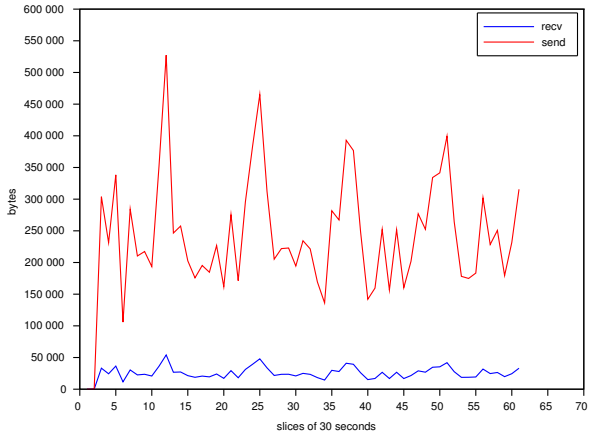

Figure 5. Network traffic of the webmail server for a single experiment (50 *Hosts*)

C. Experiments on the prototype

We verify the other properties with two separate experiments.

The first experiment is a control experiment. We deploy 5 virtual machines on the network simulator Hynesim [25] and make them generate the activity of our simulation. We script the *Elementary actions* of the *Script* described in Figure 4 with the web driver Selenium [26] 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) 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 5 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 4, with $X = 30$.

We also retrieve the logs produced by the webmail server during both experiments. The quantity and content of the logs is analyzed in Table I and Table II. Both experiments are done twenty times for each set of parameters to ensure the consistency of the results (Property 2). In the results we express the average value and standard deviation of those twenty experiments.

In the second experiment, we use the *Data generating function* with the highest level of realism: the adapted replay. That *Data generating function* preserves the data acknowledgement by the *Service* and allows *Elementary action parameters* to modify the inputs of submitted forms. Concretely, it means that a server cannot distinguish the adapted replay from an interaction with a real user. Also, with the help of *Elementary action parameters*, the evaluator can freely change the credentials replayed to the webmail server.

The analysis of the logs aims to verify the Properties 1 and 3. We verify that the Postfix server correctly accepted the data generated by the *Data generating function* for each *Elementary actions* (Property 1) and that the logs of the simulation reflects our estimation from the logs of the control experiment (Property 3). We also verify that the results are consistent over multiple instances (Property 2).

Table I represents the quantity of logs produced by the webmail server during both experiments. We express the average number of lines in the log files of the webmail and 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 control experiment while the other entries are the results of the simulation experiment.

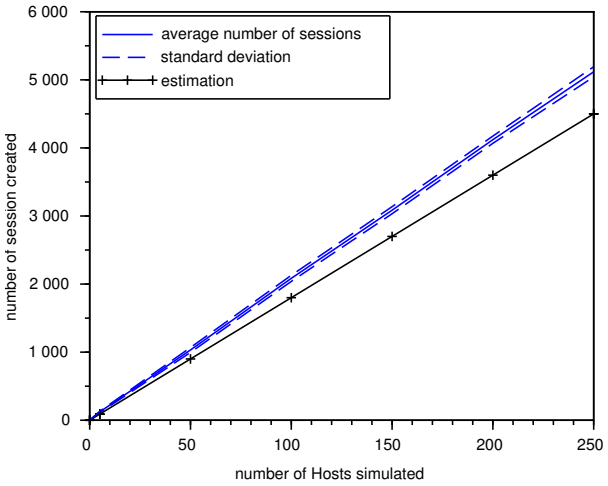

Figure 6. Number of sessions created during simulation (blue) compared to estimation (black)

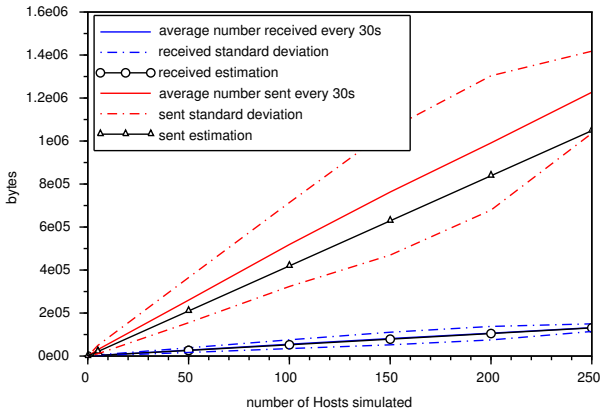

Figure 7. Network traffic of the webmail server

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 6 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 but it can be explained by 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. In Figure 7, we examine the network traffic produced by our simulated activity. The blue and red parts represent the average number of bytes, respectively, received and sent by the webmail server every 30 seconds, along with the standard deviation. For comparison, the black lines correspond to the estimation of the expected results 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 the 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 cached data into account. Therefore, our simulated connections request more data from the webmail server than estimated. This observation is part of the parametrization issues of the *Data generating function* raised at the end of Section IV. Adding *Elementary action parameters* to modify the behavior of the function can solve this issue as we did for previous typing and semantic issues. However, the addition of new *Elementary action parameters* is made from empiric observation and could be improved by adding new methods to our model like machine learning.

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 now focus on proving that the activity semantics was also preserved.

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 signature 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 action ”read the email” for a different email that the one in the *Model data*. Or to do the action ”connect to the webmail” with purposely wrong credentials. In both cases, the manual analysis of the simulated data showed 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 II.

From Table II, the following observations can be made:

- the number of signatures for the ”connect” *Elementary*

TABLE II. SIGNATURE LOG ENTRIES.

		5 VMs		5 Hosts		50 Hosts		100 Hosts		150 Hosts		200 Hosts		250 Hosts	
Signatures	**Actions**	**avg**	**stdev**	**avg**	**stdev**	**avg**	**stdev**	**avg**	**stdev**	**avg**	**stdev**	**avg**	**stdev**	**avg**	**stdev**
imap.sign1	connect	90	9	122	10	1032	36	2079	44	3075	55	4118	54	4874	87
imap.sign2	connect	90	9	122	10	1032	36	2079	44	3075	55	4118	54	4874	87
imap.sign3	connect	90	9	122	10	1032	36	2079	44	3075	55	4118	54	4874	87
imap.sign4	connect	90	9	122	10	1032	36	2079	44	3075	55	4118	54	4874	87
imap.sign5	connect	90	9	122	10	1032	36	2079	44	3075	55	4118	54	4874	87
imap.sign6	connect	90	9	122	10	1032	36	2079	44	3075	55	4118	54	4874	87
imap.sign7	connect	90	9	122	10	1032	36	2079	44	3075	55	4118	54	4874	87
imap.sign8	connect	90	9	122	10	1032	36	2079	44	3075	55	4118	54	4874	87
imap.sign9	connect	90	9	122	10	1032	36	2079	44	3075	55	4118	54	4873	88
imap.sign10	connect	90	9	122	10	1032	36	2079	44	3075	55	4118	54	4873	88
sql.sign1	connect	90	9	122	10	1032	36	2079	44	3075	55	4118	54	4874	87
sql.sign2	disconnect	90	9	122	10	1032	36	2079	44	3085	52	4121	53	5118	74
imap.sign11	open	89	9	122	10	1028	36	2069	44	3059	52	4090	53	4808	91

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 I.
- 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 II 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 3). 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 1). Each result was verified twenty times (Property 2). Therefore, our prototype succeeds in providing scalable and realistic data generation, thus validating our model.

VI. CONCLUSION

In this paper, we establish a new methodology to generate realistic evaluation data on a network support (Mininet) with far fewer requirements than the common network testbeds. This methodology takes into consideration the need for an evaluator to test different properties and evaluate different vulnerabilities in a security product. Therefore, an evaluator can select the *Data generation function* that matches the properties of the product that need to be tested. The evaluator also has a control on the granularity of the activity *Elementary actions*. The finesse of the simulated activity can be improved by introducing new *Elementary actions* or adding *Elementary action parameters* to the *Data generation function*.

We add to our previous published paper several aspects. First, we elaborate the ambition of our method into five identified requirements: customizability, reproducibility, realism, accuracy, and scalability. Second, we explain with greater details the different concepts of our methods and translate our added requirements into verifiable properties of our model. It allows us to introduce the concepts of levels of realism with clear example of the current capacities of our model and its future potential. We validate our model with a prototype able to generate realistic activity up to 250 users interacting with a webmail server. The traffic can be customized in terms of *Hosts* numbers as well as *Scripts* content. With our added verifiable properties of the five requirements of our model, we define experimental tests to make sure our prototype complies with our ambitions. In another article [27], we use this prototype to define evaluation methodologies and evaluate a security product: the intrusion detection system Suricata.

However, our prototype still has a few limitations. The existing *Data generating functions* mostly focus on the creation of network activity and does not generate system activity for host-based security products. The parametrization for more realistic *Data generating functions* also raises additional issues that need to be addressed with further work. Finally, our prototype is currently limited to the simulation of *Hosts*. In parallel with the testing of network-based intrusion detection

systems based on our prototype, the next steps of our work will focus on extending our prototype to include the simulation of *Services* and develop new *Data generating functions* that focus on the generation of system data rather than network data.

REFERENCES

[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] A. Milenkoski, M. Vieira, S. Kounev, A. Avritzer, and B. D. Payne, "Evaluating computer intrusion detection systems: A survey of common practices," ACM Computing Surveys (CSUR), vol. 48, no. 1, 2015, p. 12.

[3] C. Kreibich and J. Crowcroft, "Honeycomb: creating intrusion detection signatures using honeypots," ACM SIGCOMM computer communication review, vol. 34, no. 1, 2004, pp. 51–56.

[4] 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. IEEE, 2007, pp. 385–390.

[5] S. E. Coull et al., "Playing devil's advocate: Inferring sensitive information from anonymized network traces." in NDSS, vol. 7, 2007, pp. 35–47.

[6] A. Srivastava, K. Singh, and J. Giffin, "Secure observation of kernel behavior," Georgia Institute of Technology, Tech. Rep., 2008.

[7] F. Lombardi and R. Di Pietro, "Secure virtualization for cloud computing," Journal of Network and Computer Applications, vol. 34, no. 4, 2011, pp. 1113–1122.

[8] 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, vol. 5, no. 2, 2012, pp. 74–83.

[9] K. Nasr, A. Abou-El Kalam, and C. Fraboul, "Performance analysis of wireless intrusion detection systems," in International Conference on Internet and Distributed Computing Systems. Springer, 2012, pp. 238–252.

[10] K. Ma, R. Sun, and A. Abraham, "Toward a lightweight framework for monitoring public clouds," in Computational Aspects of Social Networks (CASoN), 2012 Fourth International Conference on. IEEE, 2012, pp. 361–365.

[11] J.-K. Ke, C.-H. Yang, and T.-N. Ahn, "Using w3af to achieve automated penetration testing by live dvd/live usb," in Proceedings of the 2009 International Conference on Hybrid Information Technology. ACM, 2009, pp. 460–464.

[12] F. Massicotte, M. Couture, Y. Labiche, and L. Briand, "Context-based intrusion detection using snort, nessus and bugtraq databases." in PST, 2005.

[13] N. J. Puketza, K. Zhang, M. Chung, B. Mukherjee, and R. A. Olsson, "A methodology for testing intrusion detection systems," IEEE Transactions on Software Engineering, vol. 22, no. 10, 1996, pp. 719–729.

[14] 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. Springer, 2008, pp. 1–20.

[15] H. Jin et al., "A vmm-based intrusion prevention system in cloud computing environment," The Journal of Supercomputing, vol. 66, no. 3, 2013, pp. 1133–1151.

[16] J. Morris, S. Smalley, and G. Kroah-Hartman, "Linux security modules: General security support for the linux kernel," in USENIX Security Symposium, 2002, pp. 17–31.

[17] M. Laureano, C. Maziero, and E. Jamhour, "Protecting host-based intrusion detectors through virtual machines," Computer Networks, vol. 51, no. 5, 2007, pp. 1275–1283.

[18] P. Mell, V. Hu, R. Lippmann, J. Haines, and M. Zissman, "An overview of issues in testing intrusion detection systems," NIST Interagency, Tech. Rep., 2003.

[19] R. K. Cunningham, R. P. Lippmann, D. J. Fried, S. L. Garfinkel, I. Graf, K. R. Kendall, S. E. Webster, D. Wyschogrod, and M. A. Zissman, "Evaluating intrusion detection systems without attacking your friends: The 1998 darpa intrusion detection evaluation," Massachusetts Inst. of Tech. Lexington Lincoln Lab, Tech. Rep., 1999.

[20] T. V. Phan, N. K. Bao, and M. Park, "Distributed-som: A novel performance bottleneck handler for large-sized software-defined networks under flooding attacks," Journal of Network and Computer Applications, vol. 91, 2017, pp. 14–25.

[21] C. Cowan, S. Arnold, S. Beattie, C. Wright, and J. Viega, "Defcon capture the flag: Defending vulnerable code from intense attack," in DARPA Information Survivability Conference and Exposition, 2003. Proceedings, vol. 1. IEEE, 2003, pp. 120–129.

[22] R. Fontugne, P. Borgnat, P. Abry, and K. Fukuda, "Mawilab: combining diverse anomaly detectors for automated anomaly labeling and performance benchmarking," in Proceedings of the 6th International COnference. ACM, 2010, p. 8.

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

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

[25] "Homepage of Hynesim," 2018, URL: https://www.hynesim.org [accessed: 2018-04-09].

[26] R. A. Razak and F. R. Fahrurazi, "Agile testing with selenium," in Software Engineering (MySEC), 2011 5th Malaysian Conference in. IEEE, 2011, pp. 217–219.

[27] 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 Fith International Conference on Information Systems Security and Privacy, 2019.