A New Approach of Network Simulation for Data Generation in Evaluating Security Products

Pierre-Marie Bajan

University of Paris-Saclay
and IRT SystemX
France
Email: {first.last}@irt-systemx.fr

Christophe Kiennert
and Herve Debar

Telecom SudParis
France
Email: {first.last}@telecom-sudparis.eu

***Abstract*—Evaluating a security product requires the ability to conduct tests to assert that the product reacts as expected, both in terms of scalability and semantics. However, the production of evaluation data at a large scale with a high semantic is very costly with current methods. Load tests are semantically poor and semantic tests require a testbed environment to be deployed at a large scale. Evaluation data from real world activity need to be anonymized and a compromise must be made between the request of the evaluator and the interest of the real world organization. Moreover, to evaluate the full scope of a security product, the evaluator needs multiple test methods. In this paper, we describe a new methodology to produce evaluation data with a customizable level of realism and the possibility to be deployed at a large scale with lower resource requirements for a network support than a testbed environment. Our prototype relies on this method to generate realistic activity for up to 250 simulated users interacting with a real-world webmail server.**

Keywords–cybersecurity; simulation; evaluation.

I. INTRODUCTION

Security products can be defined as all 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. Tests can be divided 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 amount of data.

With current testing methods [1], *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 several properties of the product and 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. 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 define two concepts. First, a *Data reproducing function* represents the level of realism of the simulation (the property of the data to reproduce) and decides the level of control over the data. Second, *elementary actions* correspond to the most atomic actions the evaluator can simulate, the basis upon which the experimental scenario will be built. We also develop a prototype of our methodology and validate our approach with 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 defines the concepts of our methodology and uses those concepts to introduce our model. Section IV 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 V.

II. RELATED WORK

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. One of the approaches to obtain data with the highest semantics is to use real world data, which can come from several sources: provided by organizations doing real world productions, or obtained from honeypots where actors from the real world were tricked into interacting with a recording system to learn about the current trends (ex: generation of intrusion detection signatures using a honeypot [2]).

However, the evaluator does not have a complete knowledge of the content of the data. Some of it 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 [3] and desanonymization techniques [4]). As for honeypots, the evaluator can never know beforehand how many 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

 ISBN: 978-1-61208-652-1

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 [5], file copy operations [6], compiling the Linux kernel [7], 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 [8], Nikto [9], w3af [10], Nessus [11]) also offers good control.

However, those methods are quite time-consuming or require in-depth knowledge of the evaluated product. Moreover, the available tools are not necessarily appropriate for customized generation.

B. Load tests

Load tests create stress on the tested product [12]. The most common tests use workload drivers like SPEC CPU2000 [7], ApacheBench [13] [14], iozone [14], LMBench [15] [7], 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 [13] [15], files download [16], or execution of Linux commands [16]). 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 semantic tests at a large scale

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 [17]. Methods for 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 [18], CAIDA [19], DEFCON [20], MawiLab [21], etc.).

However, publicly available datasets, on top of often containing errors [1], 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, those large-scale experiments produce one-time datasets that can be quickly outdated.

III. Our evaluation data production method

Our goal is to generate evaluation data at a large scale with a customizable level of realism and semantic richness. The main weakness of large scale semantic test's method lies in the testbed environment. A testbed environment requires large resources and a lot of contributors to set up, use, maintain and return to a previous state. The virtual machines used to support the data generation methods are costly and the light virtual machines are currently too limited for the requirements of semantic methods. We propose a new production method that generates controlled activity data from short traces independently of the network support. 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. In our methodology, the simulated data is not produced by the execution of activity functions of the host system but by a generic *Data generating function* that represents a level of realism required of the simulated data. A simulated activity is comprised of a single *Data generating function* that is provided with a set of *Model data* extracted from the execution of specific *Elementary actions* and used to create a *Script* of the activity. In the following section, we define concepts on which we build a formal description of our methodology.

A. Concepts and definitions

1) Elementary action: In our methodology, we distinguish real activity – not issued from our simulation method – ($\mathcal{R}$) and simulated activity – issued from our simulation method – ($\mathcal{S}$) in *Elementary actions*. We call *Elementary action* (A) a short ordered set of interactions that represents an action between two actors of the activity. Those actors are 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.

For each *Elementary action*, we acquire *Model data* that are the captured data of the execution of this *Elementary action* during real activity. 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.

Model data take different forms (traces, logs, values, etc.), and to label the ground truth, *Model data* are classified by the evaluator. The evaluator can use that classification to label the resulting *Simulation data*. For example, the evaluator can create two classes of *Model data* to represent malicious activity and benign activity, respectively. In other contexts, such as generating activity 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: A *Data generating function* (f) creates *Simulation data* from *Model data*. *Simulation data* ($d^{simulation}$) is created from the execution by a *Host* (H) of an *Elementary action* (A) during a simulated activity. It is the output of a *Data generating function* and we express our demands for *Simulation data* as *Equivalence*.

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

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

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. 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.

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

The evaluator can select the *Data generating function* that is useful to his *Model data* with *Simulation parameters* ($p^{simulation}$) and provide 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* to match a larger dataset of *Model data* (e.g. services accepting the same credentials but with different identifiers, such as "id=" and "_id=") or provide a finer control (e.g. possibility to change the credentials in the submit form).

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: We finally define a *Script*, which is the representation of a realistic behavior of a *Host*. A *Script* is an ordered set of actions coupled with *Elementary action parameters*. These actions can be *Elementary actions* or actions that do not generate activity data (e.g. "wait X seconds"). 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.

4) Our model: 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):

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

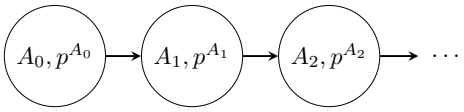

Figure 1. Example of a *Script*

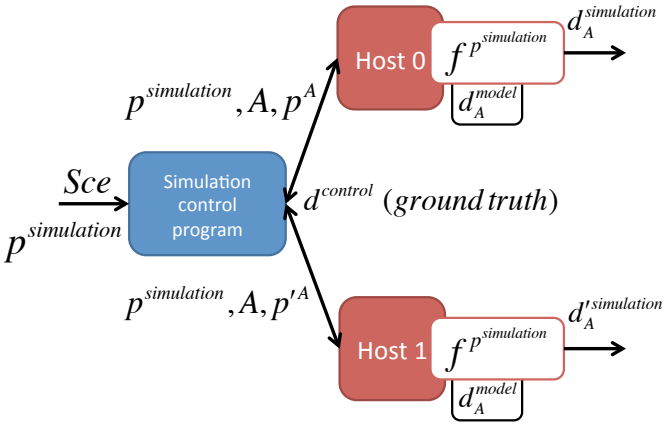

Figure 2. Generation of simulated activity from short traces

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 is 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:

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

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 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 ground truth is built upon the *Control data* of *Hosts* simulated by our model. Therefore, no data from *Hosts* unrelated to the simulation are processed.

Lastly, we must present one of the major issues of our model: the parameterization of the *Elementary actions*. Indeed, in accordance with the required level of realism of the simulation, the parameterization must allow the *Data generating function* to preserve various data properties. The higher the level of realism, the more complex the reproduction of *Elementary actions* becomes. Therefore, designing a *Data*

 ISBN: 978-1-61208-652-1

generating function for a highly realistic simulation, where not only packet size is preserved but also data acknowlegement, 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 real-life network reproduction, the semantic and scalability issues can be solved by identifying and using inputs with a high semantic value recorded during a long *Model data* acquisition period.

IV. Implementation of the Proposed Method

In this section, we describe the implemention of a prototype that follows the requirements of our model, and we show that simulations based on that prototype are both scalable and realistic.

This prototype uses Mininet [22] 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 conserve 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 3 for all the *Hosts* of the simulation. In Figure 3, 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.

To make sure that our method improves the existing methods, it must meet these two requirements:

- **Scalability:** ability to generate evaluation data proportionally to the scale of the simulation, up to a few hundreds of hosts.
- **Realism:** closeness of the generated data to the the referential *Model data*.
  Our *Data generating functions* are designed to preserve specific data properties (cf. Property 1), thus matching different levels of realism. The following experiments aim at proving that these properties are still preserved in the context of a *Scenario* with an increasing number of hosts.

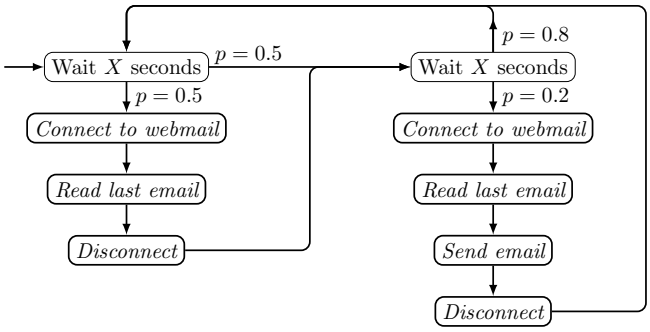

Figure 3. Generation of simulated activity from short traces

B. Experiments on the prototype

The validation of our prototype is conducted with two separate experiments, which aim to prove that our model leads to both scalable and realistic data generation.

The first experiment is a control experiment. We deploy 5 virtual machines on the network simulator Hynesim [23] and make them generate the activity of our simulation. We script the *Elementary actions* of the *Script* described in Figure 3 with the web driver Selenium [24] and make the virtual machines use their browser to interact with the webmail server. This experiment provides referential values for our second experiment. To prove the scalability of our prototype, we expect proportionality between these values and the results of our simulation, with respect to the number of *Hosts*.

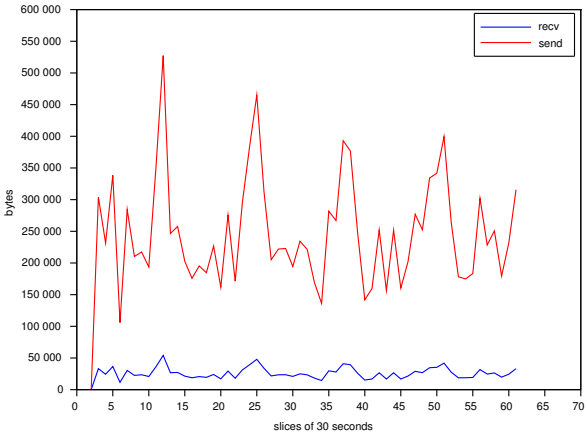

Figure 4. Network traffic of the webmail server for a single experiment (50 *Hosts*)

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

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

server hosting the simulation. Every 30 seconds, we measure four parameters: CPU usage, memory usage, network I/O, and disk I/O. Figure 4 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 3, 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.

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 results aims at proving that the data generated in the second experiment are consistent with those obtained from the first experiment, in terms of both quantity and semantics.

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.

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 5 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.

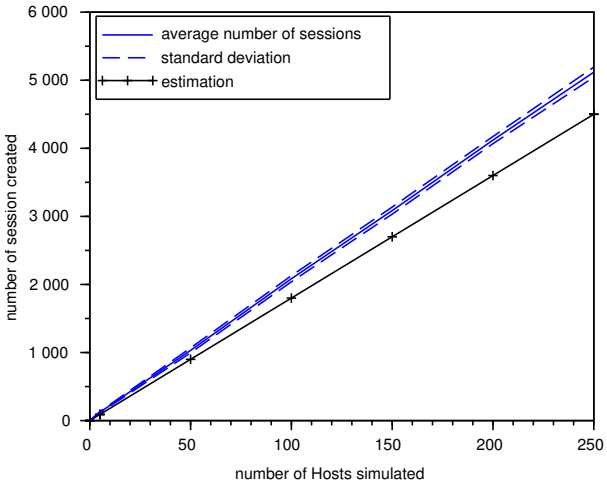

Figure 5. Number of sessions created during simulation (blue) compared to estimation (black)

These results establish that the simulated activity produces a consistent amount of logs. In Figure 6, 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 III. 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.

 ISBN: 978-1-61208-652-1

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

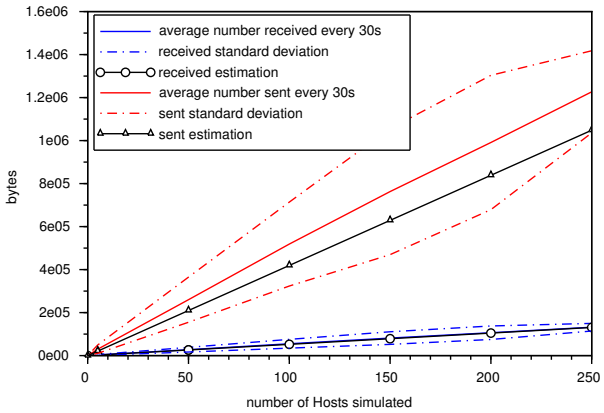

Figure 6. Network traffic of the webmail server

For each *Elementary action* of the activity *Script*, we look for log entries that could act as signatures for the action. 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 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. Moreover, the webmail server produces the appropriate number of logs reflecting the correct semantics. Therefore, our prototype succeeds in providing scalable and realistic data generation, thus validating our model.

V. 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 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. Therefore, it will be possible to use this prototype to develop more complex activity scenarios dedicated to the evaluation of security products such as intrusion detection systems.

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

 ISBN: 978-1-61208-652-1

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] 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.

[2] 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.

[3] 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.

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

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

[6] 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.

[7] 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.

[8] 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.

[9] 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.

[10] 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.

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

[12] 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.

[13] 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.

[14] 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.

[15] 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.

[16] 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.

[17] 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.

[18] 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.

[19] 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.

[20] 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.

[21] 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.

[22] 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.

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

[24] 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.

 ISBN: 978-1-61208-652-1