Semi-Synthetic Data Set Generation for Security Software Evaluation

Florian Skopik, Giuseppe Settanni, Roman Fiedler, Ivo Friedberg
Safety and Security Department
AIT Austrian Institute of Technology, Austria
firstname.lastname@ait.ac.at

***Abstract*—Threats to modern ICT systems are rapidly changing these days. Organizations are not mainly concerned about virus infestation, but increasingly need to deal with targeted attacks. This kind of attacks are specifically designed to stay below the radar of standard ICT security systems. As a consequence, vendors have begun to ship self-learning intrusion detection systems with sophisticated heuristic detection engines. While these approaches are promising to relax the serious security situation, one of the main challenges is the proper evaluation of such systems under realistic conditions during development and before roll-out. Especially the wide variety of configuration settings makes it hard to find the optimal setup for a specific infrastructure. However, extensive testing in a live environment is not only cumbersome but usually also impacts daily business. In this paper, we therefore introduce an approach of an evaluation setup that consists of virtual components, which imitate real systems and human user interactions as close as possible to produce system events, network flows and logging data of complex ICT service environments. This data is a key prerequisite for the evaluation of modern intrusion detection and prevention systems. With these generated data sets, a system's detection performance can be accurately rated and tuned for very specific settings.**

***Keywords*-anomaly detection evaluation, synthetic data set generation, scalable system behavior model**

I. Introduction

The security landscape has massively changed in the recent years. The ever increasing complexity of network systems, software and services, as well as their increasing integration and dependencies has led to novel forms of cyber security attacks. While just some years ago, the deployment of general cyber defense systems, such as firewalls and virus scanners, was mostly sufficient to protect against the most common threats, the situation is different today. Organizations are not so much concerned about spreading viruses, which can be properly detected using signature-based virus scanners, but have to fight more and more against quite targeted attacks. Vulnerability black markets [1], APTs [2] and the wide usage of malware generation toolkits [3] has led to a completely changed threat landscape. Therefore new defense technologies are being developed, such as the application of various promising heuristics in novel intrusion detection systems, and log-level event correlation across systems for advanced anomaly detection.

However, the challenge is, since attacks can be quite complex and deeply buried in the usual network traffic, how to comprehensively test and evaluate such systems in both phases (i) during *development* to enhance their effectiveness towards new attack methods/vectors through continuous algorithmic improvements, and then (ii) before *deployment* in order to tune configurations and adapt them to particular environments, e.g. to meet performance criteria. Due to the lack of data sets from realistic attack scenarios, an easy performance evaluation *and* comparison is much harder than in other computer science domains, e.g. image categorization or semantic text analysis.

Raw data sets from real systems, such as log files or packet captures, offer only limited utility for a comprehensive evaluation. Mostly the data flows and behavior of real systems are not fully known, and do not consist of the kind of attack signatures that the system under test is supposed to discover. Additionally, privacy concerns mostly hinder the direct adoption of real data sets. On the other hand, using purely synthetic data does not meet realistic test conditions, since the dynamics and complexity of real world systems are virtually impossible to be modeled in data generators. Eventually, we motivate the need for a reliable approach to merge real log data/network data with attack traces.

We therefore propose an approach to 'semi-synthetic' log data generation that consists of a wide variety of components to generate realistic and scalable data sets for the evaluation of novel anomaly detection mechanisms. This approach uses virtual infrastructure setups and artificial human stimuli to generate log and network flow data that comply to real data (regarding frequency of events, dynamics, variability etc.). Additionally, numerous kinds of attacks can be performed on these virtual environments whose occurrence and properties are well known. We foresee our system for the generation of reference data sets, with well known characteristics that are used to test and compare the efficiency and effectiveness of different security systems.

In this work, we discuss the following contributions:

- *Semi-synthetic Data Generation* deals with the combined application of virtual/artificial components and real user input models.
- *System Design and Architecture* highlights our reference design for such a system, including the underlying design decisions.
- *Implementation and Evaluation* outlines the applicability of our approach and, in particular, shows the usage

of an open source solution that has been evaluated and successfully applied in course of a research project.

The remainder of the paper is organized as follows. Related work is covered in Section II. Section III shows challenges and the problem statement as well as highlights the overall concept and architecture. Then, Section IV introduces the current implementation and the accompanying ready-to-use software package. Section V deals with the applicability of the proposed solutions. Finally, Section VI concludes the paper.

II. Background and Related Work

Today, more and more network security solutions rely on the timely event correlation and analysis of system log files from all various sources, not only network components, but also higher level application severs and the like. This approach is especially promising to detect quite complex attacks, such as APT attacks (advanced persistent threats) [2]. Therefore we need a testbed that is not only capable of emulating the network layer but also these higher level services and an integrated logging infrastructure.

Unfortunately the availability of testbeds with the required properties, including realistic scale, infrastructure and user behavior, combined with a holistic knowledge and control on occurring attacks (e.g., SQL injection attacks [4]) is quite limited. The paper on ViSe [5] describes a virtual security testbed, which is meant to make security testing significantly easier, as a virtual environment allows to switch back to previous snapshots once a malware has infiltrated and/or destroyed parts of a system. This paper further surveys similar approaches. The Lincoln Adaptable Real-time Information Assurance Testbed (LARIAT) [6], represents one of the first major attempt to create a comprehensive platform for testing intrusion detection systems. Lincoln Laboratory SIMulator (LLSIM) [7] is a fully virtualized descendant of LARIAT. Written in Java, it provides a highly-customizable environment capable of simulating hundreds of nodes on commodity hardware. TIDeS, the Testbed for evaluation of Intrusion DEtection Systems [8], seeks to quantify the evaluation process when determining the appropriate type of IDS to use given a specific network topology. The Cyber DEfense Technology Experimental Research testbed (DETER) [9] is one of the leading testbeds for security researchers, which emerged from a partnership project between the National Science Foundation, Department of Homeland Security, USC, UC Berkeley, and McAfee Research. Furthermore, there are dedicated security testbeds for infrastructures different from regular office networks, e.g., SCADA in the domain of automation systems [10].

While some of these testbeds have attracted thousands of researchers to evaluate their prototype systems, we argue that all these approaches are quite network-centric. They are the perfect foundational basis to test pure netflow analysis capabilities or study how a worm behaves in a certain network topology and how to counter related attacks. However, since other approaches, e.g. (AECID [11], operate on a higher log level layer, we need also facilities to include the virtual human users as an essential component in the whole socio-technical system. This is the strength of our testbed design.

III. Problem Statement and Contributions

A. Problem Statement

In order to test novel monitoring and security systems, such as anomaly detection algorithms and IDSs, we need an approach to create realistic test data with a wide variety of properties in context of different scenarios. Essentially, this data to be used could origin from three different sources, being either synthetically produced, from a real system, or from a semi-synthetic approach. In the latter case, a real software infrastructure is utilized, that however runs in a virtual environment and is operated by virtual users, instead of real ones. We argue that this approach offers a reasonable balance in terms of pros and cons between purely synthetic data and real data case, as further summarized in Table I.

So, finally the questions we are dealing with is *how can we efficiently generate test data suitable for security systems evaluation that is produced by real systems but with least possible human effort*?

data origin	*advantage*	*disadvantage*
synthetic	easy to (re-)produce, has desired properties, no unknown properties	no realistic 'noise' mostly simplified situations
real	realistic test basis	bad scalability (user input, varying scenarios), privacy issues, attack on own system needed
semi-synthetic	more realistic than synthetic data, easier to produce than real data	simplified and biased if an insufficient synthetic user model applied

Table I
TEST DATA AND THEIR PROS AND CONS.

B. Application of Semi-Synthetic Data

Among the potential application use cases in the security domain that rely on strong and sophisticated data input in their development, testing and deployment phase and for which we consider our work highly relevant, are the following:

- *Log-Level Anomaly Detection*, that runs on a consolidated log base.
- *Network Flow Level Inspection*, as performed by many security monitors in corporate networks.
- *ICT System Event Correlation*, e.g., typical security incident and event management (SIEM) [12] solutions.

In order to fit to the aforementioned application areas, the generated data set must fulfill a set of criteria as summarized in Table II. Due to these requirements we conclude that it is not feasible to create a realistic data set synthetically only. We rather propose the application of a real setup, however

in virtual environments for easy adaptability, that is driven by simulated human user input. We elaborate on the details in the next section.

data property	*explanation*
large size	scale and amounts as they similarly occur in real networks
realistic	feasible in terms of distribution, variability, random noise, complexity etc.
easy to generate	time efficient testing of numerous different setups
easy to adapt	fits to various test conditions and purposes
configurable complexity	applicable for basic scenario testing and detailed evaluations
extensible generation	extensible to new features in future systems

Table II
REQUIRED DATA PROPERTIES.

C. Approach Outline to Test Set Generation

Eventually, we chose an approach that instantiates the building blocks given in Fig. 1 to generate semi-synthetic data sets with the required properties:

- *Regular User Input Simulation* are virtual users which cause the ICT infrastructure components of the environment to generate logging information with properties as close as possible to real legitimate users.
- *Anomaly Injection* deals with the injection of deliberate failures e.g., caused by a maliciously acting user.
- a customizable *Services and Network Infrastructure*, e.g., a network simulation such as Mininet[1] and some typical Web based system on top.
- *Logging Facilities* to store the produced log files.
- an interface to the *System under Test*, such as an intrusion detection system, and anomaly detection algorithm or event correlation engine.

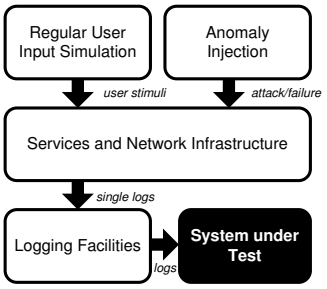

Figure 1. Approach Outline.

D. Approach Novelties

Additionally to numerous testbeds discussed in Section II, there are also some ready-to-use data sets available (e.g., the popular DARPA set [13] or [14]), which however are either outdated or intended for a different use (i.e., netflow analysis and packet filtering mechanisms). Besides these data sets, also available testbeds are not immediately applicable in our case. The reasons for that as well as our contributions are as follows:

- *Service Complexity:* in contrast to typical infrastructure testbeds that focus rather on the correct simulation of large-scale ICT networks, our approach is designed to run higher level (Web) services, integrates a working logging infrastructure and enables the simulation of not hundreds similar hosts but just a few, but complex, individual systems.
- *User Behavior:* Additionally, the approach foresees to enable simulated user inputs/interactions, and thus to model real user behavior that is usually not in focus of network simulation testbeds.
- *User Interface Automation:* Finally, due to the user-centric view, our approach integrates modules for simulating user interface inputs, i.e., defining scripts that trigger actions in complex Web based interfaces (e.g., via a standard Web Browser).

IV. IMPLEMENTATION AND SOFTWARE PACKAGE

After outlining the basic motivation and concept for semi-synthetic log data generation, we are now going to demonstrate a concrete instance of this idea here.

A. AECID Data Generation Framework

For our own research in the field of automatic event correlation of incident detection (short: AECID [11]) we set up an extensible architecture as given in Fig. 2, which is distributed over several virtual machines. In particular this architecture implements the three layer model from the previous section in the following way:

1) *Virtual User Layer:* A virtual user is simply realized by using the Selenium Framework[2], which is a browser automation tool. Usually applied for Web application testing, this tool is the perfect match to also simulate real user input. A configurable script issues carefully pre-modeled user actions in varying time intervals. The occurence of these actions follow a realistic distribution.
2) *Infrastructure Layer:* The Web-based system that is utilized by the virtual users resides in a separate virtual machine consisting of the Mantis Bug Tracker[3] and an underlying Apache Web server, a backend database, and support services, e.g mail server, backup, monitoring, firewall. This simple setup can reasonably cover a wide variety of typical use cases, e.g., users create new issues, check for updates etc.
3) *Logging Layer:* The bottom layer consists of a typical corporate logging infrastructure, again on a separated machine, where all log messages from the bug tracker,

[1] http://mininet.org

[2] http://www.seleniumhq.org
[3] http://www.mantisbt.org

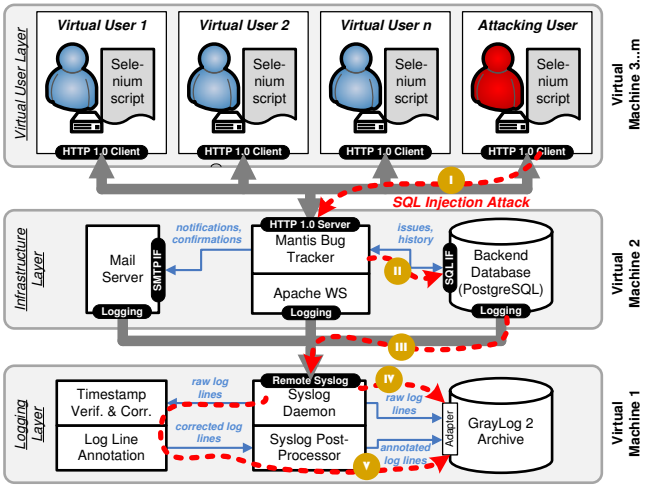

Figure 2. Implemented architecture for semi-synthetic data generation for a typical Web System (here: Mantis Bug Tracker).

the database and the support services are collected using rsyslog[4]. Some facilities to annotate log messages, e.g., add identifiers for the log source, and correct timestamps in case of drifting clocks are additionally provided. Finally, the consolidated logs are managed by a GrayLog2[5] database.

Run-Time Perspective. Virtual users utilize the system according to predefined scripts. These scripts describe the typical actions that are issued by real users with certain probability and in certain order. An overview, just to describe the possible complexity, is given in Fig. 3. All of these use cases cause typical events on the Mantis platform, the mail server, firewalls and queries to the SQL database. These events are logged via rsyslog.

The virtual users cause an adequately realistic load on the system (in terms of frequency and complexity) which in turn leads to the generation of dependent log events. Additionally to this synthetic data of known-good situations, we can further inject anomalies to stress the system under test. These anomalies are created by an attacker. In case of Fig. 2, this means a malicious user performs an SQL injection attack [4] (#I) which is enabled by a vulnerability of the Mantis Bug Tracker. Through this vulnerability a non-legitimate user is able to issue SQL queries and modify entries he has no privileges for (#II). This behavior causes a number of anomalies in the log files (#III), such as unusual SQL queries on the database, abnormal order of commands issued on the bug tracker which differ from typical use cases, or missing notification mails on the Web server for certain changes on the bug tracker. All these information is stored in a GrayLog2 instance both in raw format (#IV) and in a pre-processed form (#V), e.g.,with an additional timestamp of recording. Finally, the whole logging data is extracted from the Graylog2 archive and used to evaluate the detection rate of the injected anomalies with IDSs and algorithms to-be-tested.

[4]http://www.rsyslog.com
[5]http://graylog2.org

B. User Interaction Behavior

Creating log files that consist of events caused by realistic user behavior is of paramount importance to ensure solid data sets. We therefore carefully set up user behavior simulations with respect to issued actions and paths (cf. Fig. 3), as well as interaction rates, amounts and temporal properties. For that purpose we studied the typical usage behavior of Mantis on a real system. Some examples are given in Table III. Each row shows the current status of a user in the first field and potential follow-up actions along with a probability in the second field. This data was obtained from a Mantis instance in a productive environment over a long-term investigation of 3 months. Notice that determined probabilities of course vary depending on the main occupation of the user, e.g. developer, software tester, project manager. In order to simplify the situation, we determined values for a ”virtual average user”.

C. Log Data Handling and Extraction

The correct handling of the recorded log data is crucial for sound evaluations of the system under test. Clock drifts, different time zones between network components or even swapped log lines can render the results useless. For performance tests a minimum latency when accessing the newest log data is also important. This latency is not only influenced by the required pre-processing of the incoming lines. Access time at the log archive is also a critical parameter.

Therefore, we propose a two-level log archive infrastructure as depicted in Fig. 4. Log data is collected at the network nodes using remote syslog connectors. It is then received by a central *Timestamp Validation and Correction Unit*. The corrected log lines are sent once to a persistent storage implemented by MongoDB and Graylog2 and a second time to an in-memory cache for the most recent lines realized with HSQLDB. A *Source Adapter* then provides an interface for the system under test to access the log lines.

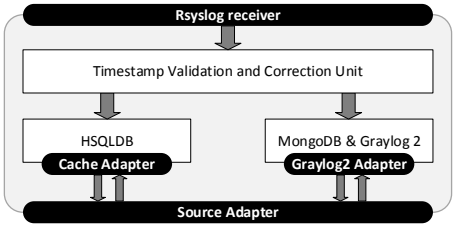

Figure 4. Two-Level log archive solution.

Every log source is identified by a unique id (basically a hierarchically organized url). Messages can be retrieved by either specifying the desired id, a time span or both (see interface description in Listing 1).

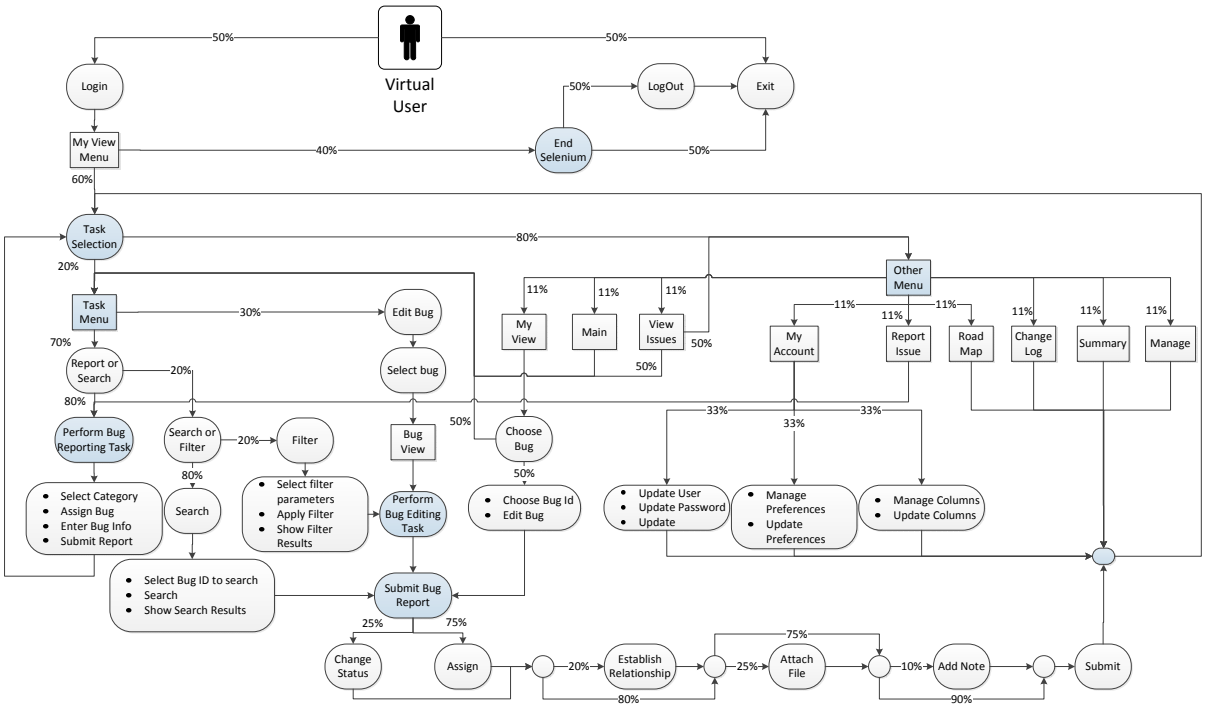

Figure 3. Mantis use cases of virtual users and the respective likelihoods to follow a specific path through the system.

```
String[] getSourceIDs();
ResultSet getMessagesAfterTime
  (long timestamp, String[] sourceIDs);
ResultSet getMessages
  (long startTime, long endTime, String[] sourceIDs);
ResultSet getMessagesAfterId
  (long messageID, String[] sourceIDs);
```

Listing 1. Interface provided by the Source Adapter.

Every request is first handled by the *Cache Adapter*. The cache contains all lines received in the last x seconds (where x can be configured). If the query cannot be answered with cached data only (e.g. the selected time window exceeds the caching time), the request is instead processed by the *Graylog2 Adapter*. This adapter provides access to Graylog2 and MongoDB which implement high-performance indexing and searching on vast amounts of data.

D. Software Package

Supporting this paper we provide an example infrastructure package. It consists of three extensible and independently replaceable virtual images. All software packages needed are already pre-installed.

1) Network Image: The first image contains a virtual network setup. Various services (mail daemon, rsyslog, database, two independent mantis instances) are already pre-installed and configured. The remote syslog daemon is used to extract the log information generated by the services and to forward them to the analysis image.

2) Analysis Image: The analysis image provides the log handling infrastructure as described in Section IV-C. The provided interface is one way to extract the generated semi-synthetic log data and perform custom evaluations.

3) User Simulation Image: The third and last image is used to simulate one user on the test network. Running this image multiple times is a convenient way to simulate users with different clients and IP addresses. With small adaptations it would also be possible to simulate multiple browser agents, or completely different statistical user behavior.

V. Evaluation and Discussions

The evaluation is performed by the means of (i) run-time performance measurements, (ii) qualitative analysis of the results, and (iii) a critical review on the applicability.

A. Scenario Description and Test Setup

The virtual user images are entirely deployed on a i7-3540M @3GHz machine with 8GB of RAM running Windows 7 Enterprise 64 bit. The scenario is composed by N ($N <= 25$) VirtualBox (linked copies) virtual machines running Ubuntu 13.04 and OpenBox. Each virtual machine has a base memory of 256 MB, its own IP address and executes at start-up the Semi-synthetic log data generator tool which emulates the behavior of a single user on Mantis using Google Chrome Web browser. The execution time is set to 15 minutes. When the execution time is over, all the virtual machines are simultaneously shut down. The log messages analyzed in the test are collected from three sources: the Web server, the SQL database, and a reverse proxy. In the described setup we have been able to run tests with up to 25 simultaneous virtual users on a single machine.

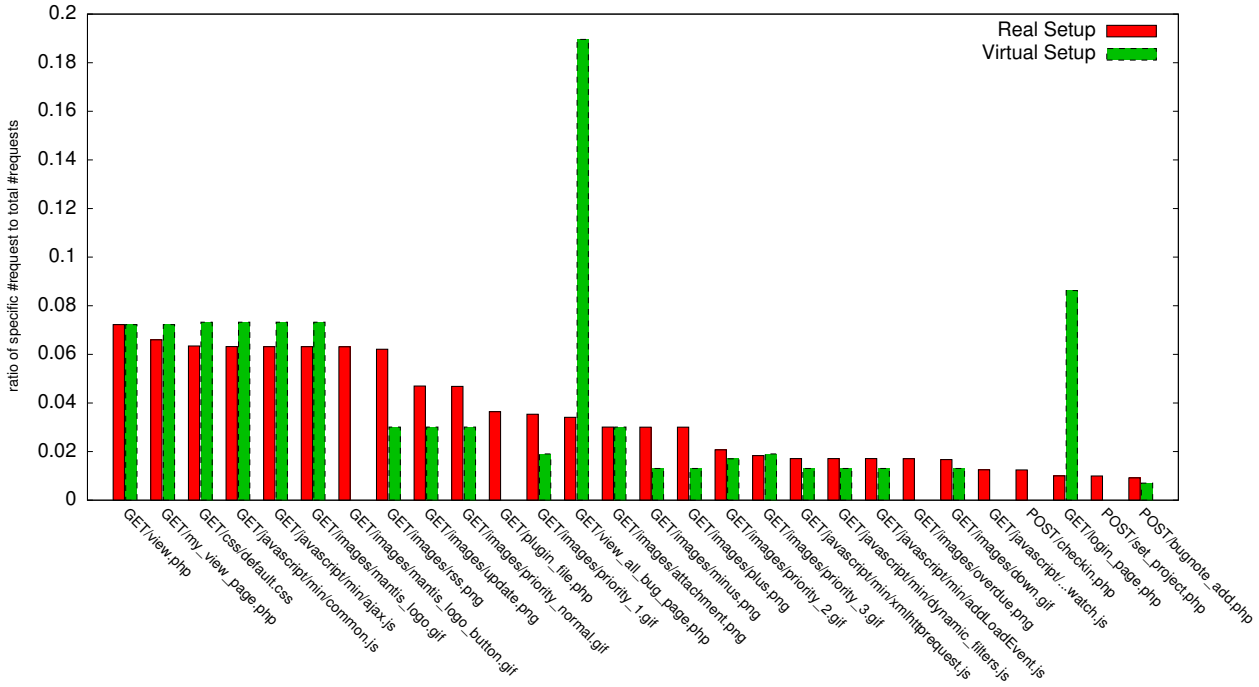

Figure 5. Distribution of user requests extracted from a data set from a real productive environment compared to request occurring in the simulation environment with virtual users. Notice: some user actions are not implemented in the virtual user model (e.g., `GET/plugin_file.php`). Moreover, the two outliers result from the fact that the simulated user does not use cookies to keep logged-in and does not use bookmarks to go to specific views directly, thus requests `GET/view_all_bug_page.php` much more frequently.

B. Run-Time Performance

Since, the implemented system does not work deterministically, we performed all experiments at least 5 times in order to investigate the variability of results. Furthermore, in order to generate some realistic load on the test system (Mantis and periphery), we set up 5 to 25 virtual users and utilize different functions, such as browsing through the entered bug list, viewing details, update details of a bug and so on. In this setup the backend produces massive amounts of logging data, e.g., each single Web server request is logged and each database query issued. Due to the nature of the log data, the relation of *number of users* to *produced data* is virtually linear. Table IV shows some details about the scale of data that we produced on a single machine within 15 minutes.

C. Qualitative Analysis

Real Mantis Usage Reference Data set. To rate the accuracy of our virtual user behavior model, i.e., the order and frequencies of virtual user actions, we studied the logged action of a Web server, which runs a Mantis instance that is used by a real software development team. We extracted the most important data properties of this real world example and compared them with the data properties of the logs produced with our approach.To sum up, the real data set consists of 291 different types of (Apache) Web server requests (sites and resources, posting data etc.) and a total of 492832 web server requests, produced by 25 simultaneously active real users over a period of one regular working day. The distribution of the request frequencies is strongly exponential, which means that the 10 most often issued requests account for half of all requests (approx 250000), and half of all request types occur less than 30 times.

Semi-Synthetic Data Set. The resulting data set from our test runs distinguishes only 88 types of requests (compared to 291 from the real example), since we did not implement all different actions a real user could perform on the platform. However, we implemented the most frequent ones to reach a user behavior and system utilization that is quite similar to the real platform. In total, with 25 simulated users, we produced between 3063 and 4308 requests (multiple test runs with varying behavior). Notice that we took care not to overload the Mantis host system and to implement request rates that are realistic. This means that the virtual user remains on Web sites for several minutes like a real user usually does. However, if one needs to produce large amounts of log files and realistic request rates are not an issue, e.g., for retrospective anomaly detection, one could simply turn up the request rate by shorten the wait-states of the user behavior model. This would affectively allow to produce much more data in much shorter periods as a real system could deliver.

Comparison. Eventually, the quality of the produced log data sets and the feasibility for a proper security system evaluation depends on its similarity with real world data.

status	*follow-up action*
LoggedIn	user performed login because he wants to operate on bugs (TaskSelection 60%) or he wants to analyze his issues (MyViewMenu) and then exit (EndSelenium 40%).
TaskSelection	work on bugs (TaskMenu 20%) or manage account and settings (OtherMenu 80%).
TaskMenu	report or search a bug (ReportOrSearch 70%) or work on an existing bug (EditBug 30%).
ReportOrSearch	enter information (category, assigner, description) and report a new bug (PerformBugReportingTask 80%), or search for a bug using its bug ID or via filtering options (SearchOrFilter 20%).
SearchOrFilter	search a bug by using the bug ID (Search 80%), or by specifying filtering options (Filter 20%). On the obtained bug editing operations are performed and the bug report is then submitted (SubmitBugReport).
EditBug	select a bug using the bug ID, view the bug details, perform editing operations and then submit the bug report (SubmitBugReport).
SubmitBugReport	change status (ChangeStatus 25%), or assign bug (Assign 75%). In any case further operations on the bug are performed. A relationship between bugs is established (20%) or not (80%), a file is attached (25%) or not (75%), a note is added (10%) or not (90%). The bug report is at this point submitted and the user, in case the timeout is not expired yet, selects a new task (TaskSelection) .
OtherMenu	the user can perform, with the same probability (11%), one of the following operations: go to the user view and edit a single bug (MyView), go to the main page and select a new task (Main), view all the issues (ViewIssues), go on the account page (MyAccount), report a new issue (ReportIssue), go to the road map page (RoadMap), go to the logging page (ChangeLog), go to the summary page (Summary), or go to the management page (Manage).

Table III
USER ACTION PATHS AND PROBABILITIES.

	5 users		25 users	
Log Source	*min.*	*max.*	*min.*	*max.*
Web Sever	926	1029	3063	4308
Database	72150	78589	240889	305068
Reverse proxy	916	1018	3022	4217
Total log lines	74072	80521	247231	313593
avg.lines/minute	4938	5368	16482	20906

Table IV
LOG DATA PRODUCTION PERFORMANCE DURING A 15 MINUTES RUN.

For that purpose, we compared (i) the data set from a real productive environment with (ii) our produced set. Figure 5 visualizes the relative request rate for the real setup and the virtual setup for the most common types of request. As one can see, even the quite simple virtual user behavior model which only implements 88 of 291 request types can already simulate a realistic load on the platform. Notice that a 100% accurate virtual user behavior is not required since the purpose of the produced requests is just to create a realstic load and background noise on the system when an actual attack is carried out to evaluate security systems (e.g., IDSs). Hence, it is just important that the virtual setup does not ease the detection of anomalies and attacks.

Two outliers are significant, which we left in our model on purpose. The resource `view_all_bug_page.php` is the starting point for all simulated user actions and is requested therefore much more often, while in reality people may use direct links to an issue (directly request `view.php`). Moreover, the simulated user does not use cookies to keep logged-in. These outliers are not a weakness of our approach or implemented system, but just show the importance of an accurate user model to get realistic data and the consequences if certain aspects are not considered appropriately.

D. Critical Review

After demonstrating the successful application we like to highlight the following notable advantages of our approach:

- Not only log-data but also reaction of components, e.g. IDS detection, blocking attack, etc. can be simulated.
- The browser/server interaction for user input simulation triggers completely realistic effects, e.g. loading of style sheets, javascript-processing, form-processing etc.
- The interactions of the server components and their temporal order and complexity are completely realistic.
- The whole test-ecosystem allows evaluation of a wide variety of real attacks on the interlinked server/client system, e.g. spreading of drive-by downloads.

Nevertheless, there are a few inherent disadvantages that applicants need to be aware of. In our future work, we are going to address some these:

- The reproducibility of data sets is limited. Even if we seed random generators of the user behavior model with a predefined value, depending on the complexity of the system the log output might be slightly different in each run.
- The user interaction model based on real browser input results in much more realistic load on the infrastructure than simple command line based scripts, however there might still be artifacts due to oversimplification in the data sets.
- The usability in terms of speed, ease of use and system scalability, is considerably worse compared to a data generation application that is running on a single machine.

VI. CONCLUSION AND FUTURE WORK

In this paper, we described an approach to realistic (log) data set generation of Web-based infrastructures. This is essential in order to evaluate security systems in a realistic manner. In contrast to many existing approaches, we do not focus on large-scale networks with hundreds of machines where we perform net flow analysis. We rather deal with simulating systems and service, on a higher level, including lifelike user input in order to challenge systems close to

reality. The results of this effort are reliable data sets for system testers, and algorithm developers.

Future work deals with reducing the complexity of application for users, the improvement of the existing software bundle, especially in terms of modularity and extensibility with third party software, and the incorporation of critical feedback from the research community.

ACKNOWLEDGEMENTS

This work was partly funded by the Austrian FFG research program KIRAS in course of the project CIIS (840842) and the European Union FP7 project ECOSSIAN (607577).

REFERENCES

[1] A. Ozment, "Bug auctions: Vulnerability markets reconsidered," in *Third Workshop on the Economics of Information Security*, 2004.

[2] C. Tankard, "Advanced persistent threats and how to monitor and deter them." *Network Security*, vol. 2011, no. 8, pp. 16–19, 2011.

[3] M. E. Karim, A. Walenstein, A. Lakhotia, and L. Parida, "Malware phylogeny generation using permutations of code." *J. in Comp. Virology*, vol. 1, no. 1-2, pp. 13–23, 2005.

[4] W. G. Halfond, J. Viegas, and A. Orso, "A classification of SQL-Injection attacks and countermeasures," in *IEEE Int'l Symposium on Secure Software Engineering*, 2006.

[5] M. Richmond, "Vise: The virtual security testbed," University of California, Santa Barbara, Tech. Rep., 2005.

[6] L. M. Rossey, R. K. Cunningham, D. J. Fried, J. C. Rabek, R. P. Lippmann, J. W. Haines, and M. A. Zissman, "Lariat: Lincoln adaptable real-time information assurance testbed," in *In IEEE Proc. Aerospace Conference*, 2001, pp. 2671–2682.

[7] X. Jiang, D. Xu, H. J. Wang, and E. H. Spafford, "Virtual playgrounds for worm behavior investigation." in *RAID*, vol. 3858. Springer, 2005, pp. 1–21.

[8] G. Singaraju, L. Teo, and Y. Zheng, "A testbed for quantitative assessment of intrusion detection systems using fuzzy logic," in *IEEE Int'l Inf. Assurance Workshop*, 2004.

[9] T. Benzel, "The science of cyber security experimentation: the deter project." in *ACSAC*. ACM, 2011, pp. 137–148.

[10] T. Morris, R. Vaughn, and Y. S. Dandass, "A testbed for scada control system cybersecurity research and pedagogy," in *Workshop on Cyber Security and Information Intelligence Research*, 2011, pp. 27:1–27:1.

[11] F. Skopik and R. Fiedler, "Intrusion detection in distributed systems using fingerprinting and massive event correlation," in *43. Jahrestagung der Gesellschaft fuer Informatik e.V.*, 2013, pp. 2240 – 2254.

[12] D. R. Miller, S. Harris, A. Harper, S. VanDyke, and C. Blask, *Security Information and Event Management (SIEM) Implementation*. McGraw-Hill, 2010.

[13] DARPA Intrusion Detection Data Sets, http://www.ll.mit.edu/mission/communications/cyber/CSTcorpora/ideval/data/, 2000.

[14] J. Sommers and P. Barford, "Self-configuring network traffic generation." in *Internet Measurement Conference*. ACM, 2004, pp. 68–81.