LESS Is More: Host-Agent Based Simulator for Large-Scale Evaluation of Security Systems

John Sonchack[1] and Adam J. Aviv[2]

[1] University of Pennsylvania, Philadelphia, PA 19104, USA
[2] United States Naval Academy, Annapolis, MD 21402, USA

Abstract. Recently proposed network security systems have demonstrated the benefits of *scale* for achieving many security goals, including the detection of worm outbreaks, botnets, and denial of service attacks. However, scale is also a barrier to further advancement of such systems: obtaining and working with appropriately large data sets is difficult, and existing simulation techniques are ill suited for this domain. To overcome these challenges, we propose a host behavior simulator, LESS, designed for evaluating large scale network security systems. LESS build and automatically configures the behaviors of host agents using background traffic samples and malicious traffic models. In turn, host agents communicate with each other throughout a simulation, generating traffic records. We demonstrate the applicability and benefits of LESS by tuning it with publicly available traces, and then using generated records to *reproduce* results from several recently proposed systems. We also used LESS to *extend* the evaluations of these systems, highlighting dimensions of large scale security system performance that would be difficult to study without simulation.

Keywords: Data Challenges, Large Scale Security, Simulation, Agent Based, Stochastic.

1 Introduction

Many recently proposed network security systems leverage *scale* [1, 2, 3, 4, 5]. By analyzing diverse data sets gathered from many vantage points, a network security system can identify macroscopic host behavior patterns that cannot be observed at the local network level. Although beneficial, the scale of such proposed techniques also results in a high bar for advancement and investigation from the general research community [6, 7]. Internet- or ISP- scale data sets appropriate for studying large-scale security systems are difficult to acquire and share. Furthermore, data sets of sufficient scale that are publicly available often lack context that is important for meaningful evaluation, such as ground truth about which traffic is malicious. These data limitations affect the ability of researchers to reproduce existing results, and limit the dimensions in which they can analyze their systems.

In this paper, we argue that novel simulation techniques specifically designed to analyze large scale security systems can directly address these challenges. In

M. Kutyłowski and J. Vaidya (Eds.): ESORICS 2014, Part II, LNCS 8713, pp. 365–382, 2014.

other areas of network research, simulation has proven an effective tool to overcome similar data-related challenges. However, available simulation tools do not model the usage and behavior patterns in a way well suited for evaluating large scale security systems. We propose an alternate simulation strategy that is agent based: the **L**arge-scale **E**valuation for **S**ecurity **S**imulation (LESS). LESS generates host agents that communicate and generate traffic based on behavior defined by tunable stochastic processes. LESS automatically tunes these processes using small samples of real traffic and user provided models of well defined network security threats.

We implemented the LESS simulator, and assessed its effectiveness by using it to re-evaluate four large scale network security systems. These systems were originally evaluated on real traffic traces, inaccessible to the public. However, using publically available network traffic samples, intuitive malicious traffic models, and previously published traffic parameters, LESS generated traces that *reproduced* results from the original evaluations, *confirming prior results with access to just public data*. Furthermore, LESS allowed us to *extend* the original evaluations, *studying prior systems in new dimension that could not be achieved with the original data*. These results suggest that domain-specific simulation, and LESS in particular, is a vital tool in studying large scale collaborative security techniques.

In summary, our contributions are:

- The design and implementation of LESS: an agent based simulator for evaluating large scale network security systems.
- A tunable, stochastic host-behavior process that allows hosts-agents to generate background traffic based on parameters from real traffic samples, and malicious traffic based on parameters from intuitive threat models.
- Simulation based *reproductions* and *extensions* of results from four large scale security system evaluations.

In the remainder of this paper, we examine the intersection of large scale security systems and simulation based analysis. First, in Section 2, we discuss related simulation techniques. Next, in Section 3, we summarize four recently proposed large scale security systems, and their evaluations. We discuss the implementation of LESS in Section 4. Then, in Section 5, we use LESS, along with publicly accessible data sets, to reproduce and extend the evaluations of the surveyed security systems. Finally, we discuss limitations and future work in Section 6, and conclude in Section 7.

2 Related Work

Simulation offers many benefits for computer network related research [8]. Discrete event simulators [9], and virtual networks [10] provide frameworks for simulating many network connected devices on a single machine, and are useful for evaluating distributed systems. However, these tools require users to completely specify the actions of each simulated device, making them ill-suited for evaluating large scale security systems, which are designed to detect or prevent specific

types of malicious activity based on usage patterns. Our work is complementary: LESS models inter-host communication patterns and behaviors, but *not* the networks that connect hosts.

Traffic generators, such as [11] generate synthetic packets to be carried on real or virtual networks. These systems model packet and payload level features, and are used to evaluate techniques that are sensitive to these features, such as congestion control [12]. However, large scale network security systems are sensitive to higher level features, such as traffic dispersion or host communication patterns. LESS is orthogonal to synthetic traffic generators, in that it models these higher level network-flow properties and is better suited for evaluating large scale security systems.

User behavior modeling has been proposed to generate specific classes of non-malicious Internet traffic, such as HTTP traffic [13]. LESS differs in two important ways from these systems: first, it models behavior patterns at the *host* level; second, it allows users to augment hosts with *malicious* behaviors that model large scale security threats such as worm outbreaks [14], correlated attackers [4], and bot communication networks [15]. Thus, LESS applies similar behavior modeling techniques, but in a way better suited for evaluating large scale security systems.

Simulation techniques tailored for specific classes of security systems have been proposed; Sommers et al. [16] craft flow payloads to evaluate and tune intrusion detection systems, and Chen et al. [17] propose a simulator for evaluating worm outbreaks in a peer-to-peer network. These techniques are complementary to LESS in scope and scale, but motivated by the common observation that other network simulation techniques are ill-suited for security research.

Testbed systems, such as Lariat [18], are platforms that use real or virtualized networks along with complex models of host behaviors to evaluate security systems of varying scale. Testbeds require careful tuning: the authors of Lariat, for example, estimated that configuring their system to test a new intrusion detection system takes approximately four months [18]. In contrast, LESS uses more general models that can be automatically fit to and validated against real traffic traces. Simpler models also lead to quicker run times: each simulation that we ran to generate experimental results for Section 5 executed in one hour or less. LESS and security testbeds could be used complementary: for example, the preliminary evaluation of a new technique could be done using LESS, followed by a testbed evaluation of the system in a specific scenario.

LESS applies two general classes of simulation techniques to large scale security system experimentation. First, *agent based simulation* [19], in which complex systems are modeled as a set of agents that interact with each other based on individual decision processes. Second, *stochastic simulation* [20], in which systems are analyzed with inputs drawn from statistical distributions of real observations. More specifically, LESS models usage in a large scale network from the perspective of a collection of *host agents* that interact with each other in a client-server model to generate both *background* and *malicious* traffic. The

actions and behaviors of these host agents are defined by processes that are *configured stochastically*, with distributions derived from real sample traces.

3 Large Scale and Collaborative Security Systems

In this section, we summarize four systems that demonstrate both the potential of large scale security and the data related challenges of experimentation in this domain.

3.1 Entropy Based Anomaly Detection

Wagner et al. [1] observed that large scale security threats, such as worm outbreaks and high volume DDoS attacks, skew the distribution of flow level traffic features such as source and destination IP addresses. Based on these observations, they proposed a technique to detect such events by measuring the entropy of traffic features.

Wagner et al. evaluated their technique using large scale traces from the outbreaks of two real worms. Such traces are difficult to acquire, and do not allow researchers to analyze how worm behavior properties (*e.g.* inter-scan time) affect system performance. LESS allows researchers to work around these issues by generating background traffic based on parameters measured from publicly accessible samples, and malicious traffic based on parameters selected by tunable threat models.

3.2 Highly Predictive Blacklisting

The Highly Predictive Blacklisting System [3] generates blacklists customized for individual networks by analyzing the attack patterns of malicious hosts across multiple collaborating networks. The proposed approach is based on the observation that many attackers target small, stable sets of networks [4].

The authors evaluated this system on a set of over 700 million Intrusion Detection System log entries submitted to DShield [21] by over 1500 networks over a two month period of time. This data set cannot be used to measure false alert rates, as IDS log entries alone do not provide sufficient context to determine the true maliciousness of the hosts causing alerts. LESS can provide perspective on how false alerts affect a system, by using well defined malicious traffic models and record sets where malicious traffic is clearly marked.

3.3 Peer-to-peer Botnet Detection

Coskun et al.. [5] proposed a technique to detect peer-to-peer bots by correlating host communication patterns, based on the observation that the bot-hosts in a network are more likely to communicate with common external hosts.

The authors evaluate their approach by mixing a 24 hour trace collected at the boarder of a university network containing approximately 2000 hosts, and

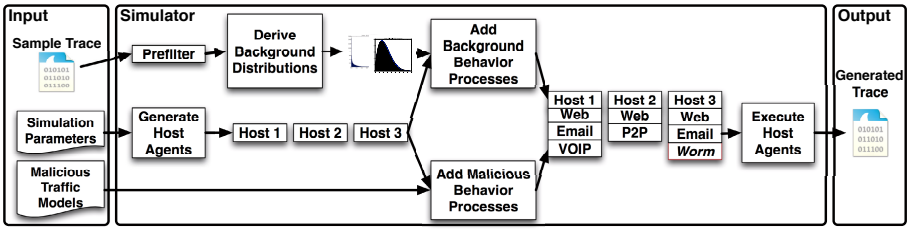

Fig. 1. An overview of LESS

communication records from approximately 900 bots from the Nugache botnet. A data set from one botnet does not allow researchers to measure how the topology of the peer-to-peer botnet affects their systems. With LESS, however, users can estimate the effects of this factor with a tunable botnet traffic model. Further, this data set is from a single network, which makes it difficult to analyze how a security system will perform at other networks that carry different types of traffic. LESS can generate data sets based on samples of traffic from other networks, which in turn can be used to estimate how robust a security system is to diverse network conditions.

3.4 Collaborative Anomaly Detection

Boggs et al. [2] proposed a technique to significantly reduce the false positive rate of anomaly detectors, by correlating the alerts generated by detectors deployed at multiple networks. Their system is based on the observation that if multiple unrelated hosts or networks observe similar anomalies, they are more likely to be due to a common attack.

Boggs et al. evaluated the proposed technique on eight weeks of traffic collected at two university web servers. Their evaluation demonstrates how challenging it is to collect sufficiently detailed data from multiple administrative domains. LESS is beneficial in this scenario because it allows users to simulate many networks based on parameters drawn from a single, more accessible traces.

4 Simulation for Large Scale Network Security

LESS, illustrated in Figure 1, generates traces for evaluating large scale security systems by generating host agents, assigning them stochastic processes that define their behavior, and then recording their activities in a simulation where the host agents execute their behavior processes. A behavior process models a single host's communication pattern using a single class of network applications (*e.g.* web, email, VOIP, worm). LESS accepts the following inputs:

– a **sample trace** containing representative background traffic, which LESS measures statistics from to configure non-malicious host behavior;

- two **simulation parameters** N and T, that specify the number of host agents to generate and the duration of the simulation;
- a **malicious traffic model** M, which LESS uses to configure malicious host behavior processes.

When a LESS simulation runs, the host agents autonomously determine *when* to communicate and *which other host agents* to communicate with using their behavior processes. LESS monitors their communications and generates records consisting of timestamps, source and destination host IDs, and source and destination ports.

4.1 Preprocessing

LESS configures the background behavior processes of its host agents with measurements derived from a user provided sample trace. The sample trace can be anonymized and without payloads. In our LESS based security system re-evaluations, discussed in Section 5, we used short, publicly available traces collected by CAIDA from single Internet backbone links [22], and found that they were more than sufficient.

The preprocessing stage of LESS has three steps. First, it converts the trace into flow level records using Argus [23]. Next, it classifies the flow records based on their application type. In our current iteration, LESS classifies based on destination port; however, there are many more advanced techniques for traffic classification, such as [24]. Finally, LESS determines the client / server roles of each pair of communicating hosts. In our current version, we identify servers by port (*i.e.* the server is the host using a well known port below 1024). There are also more advanced techniques for identifying clients and servers that can be integrated into LESS [25]. LESS currently discards all flows in which neither port is well known.

4.2 Assigning Applications to Host Agents

After preprocessing the input trace, LESS generates N host agents and determines which applications types each host agent will model, based on three statistic samples measured from the input trace:

- **client application count sample**, or the number of application types each host from the input trace uses as a client;
- **server application count sample**, or the number of application types each host from the input trace uses as a server;
- **application host percent sample**, or the percentage of hosts from the input trace that use each application.

LESS then assigns a set of client and server applications to each host agent h with the following procedure:

1. draw a client count c from the client application count sample and a server count s from the server count sample;
2. select c applications observed in the input trace randomly without repetition, weighted by the application host percent sample, and assign them to h's client application set;
3. select s applications observed in the input trace randomly without repetition, weighted by the application host percent sample, and assign them to h's server application set;

4.3 Configuring Background Traffic Generation Processes

Next, LESS configures a behavior process for each client application type assigned to each host agent. A background behavior process accepts the following parameters:

- **connection inter-generation distribution:**, a distribution that defines how long the host agent waits before generating a new connection using the modeled application;
- **server count**: a parameter that defines the maximum number of servers the host agent should connect to with the modeled application;
- **server weight distribution**: a distribution that defines the preference of the host agent for connecting to each host agent that serves the modeled application;
- **community ID**: a parameter that specifies which community the host agent belongs to for the modeled application;
- **inter-community ratio**: a parameter that specifies the host agent's preference for connecting to servers belonging to its own community.

A background behavior process consists of the following two steps, that repeat until the end of the simulation:

1. **Connection Generation:** determine when to generate the next connection by drawing a sample from the *connection inter-generation distribution*, and adding it to the current timestamp.
2. **Destination Selection:** when the timestamp computed in 1 arrives, determine what host agent to initiate a connection with using a decision process that narrows the range of possible destinations by first determining whether to connect with a previously contacted host or a new host (based on whether this process has reached its *server count*), and then determining whether to connect with a host agent inside or outside of its community (based on the process's *inter-community ratio*) After removing the host agents that do not satisfy the criteria selected above, LESS selects one of the remaining host agents randomly, weighted by the *server weight distribution*.

LESS selects inputs for a background behavior process p that models the use of application a by a single host agent with measurements from the input trace, as follows:

1. LESS measures the connection inter-generation distributions and server counts of all hosts that use application a in the input trace, and randomly selects one inter-generation distribution and one server count for p;
2. LESS measures the server weight distribution for application a, and assigns it to all processes that model a, including p.
3. LESS groups all the hosts from the input trace that use application a into communities using the algorithm described in [26], and then selects a community ID for each process that models application a by selecting a community at random, weighted by the size of the community in the input trace;
4. Finally, LESS selects an inter-community ratio for process p at random from the inter-community ratios of all hosts that belong to the corresponding community from the input trace.

4.4 Configuring Malicious Traffic Generation Processes

LESS also adds malicious behavior processes to host agents, based on user provided models of malicious network behavior. Below, we define the three models that we apply in Section 5 to re-evaluate large scale security systems. These models are straightforward, and based on previously published observations about malicious host behavior.

The models are also tunable, with parameters specified by the user. All of the models accept the following three parameters:

- N**:** the number of host agents that model the malicious behavior;
- I**:** an inter-event timing distribution for the malicious behavior processes;
- T**:** the start time of the malicious behavior processes.

Random Worm Outbreak Model The random worm outbreak model, based on observations from [14], simulates the propagation of a randomly spreading worm. This model accepts inputs N, I, and T, as defined above.

To augment the hosts with this malicious behavior, LESS first selects N host agents and marks them as vulnerable to the worm. At time T, a randomly selected vulnerable host initiates the following *propagation algorithm*, which repeats indefinitely:

1. Select a target host, uniformly at random.
2. Send a probe to the host. If the host is marked as vulnerable, mark the host as infected and start the propagation algorithm on that host.
3. Select an inter-probe time from I. At time $currenttime + selectedtime$, repeat from step 1.

Targeted Attacker Model The targeted attacker model simulates attackers that persistently target a small number of networks. This phenomenon has previously been observed in large scale alert repositories [4], and is similar in nature to Advanced Persistent Threats [27].

In addition to N, I, and T, this model also accepts an input C: a distribution measuring the number of networks targeted by each attacker.

To augment hosts with this malicious behavior, LESS first groups all hosts into networks of size between 2 and 128, chosen uniformly at random. It then selects N host agents uniformly at random, and marks them as targeted attackers. For each attacker, LESS draws a value c from C, and then randomly selects c networks for the attacker to target. Finally, at time T, the host agents marked as targeted attackers begin running the follow traffic generation algorithm:

1. Select a target network from the list of targeted networks assigned to this host agent.
2. Select a host in that network uniformly at random.
3. Initiate a connection to the selected host agent.
4. Select an inter-attack time from I. At time *currenttime* + *selectedtime*, repeat from step 1.

P2P Botnet Communication Model The last model simulates a network of infected hosts communicating in a peer-to-peer overlay network, which increases botnet resilience [15].

In addition to N, I, and T, this model also accepts an input G: a graph generation algorithm.

To augment hosts with this malicious behavior, LESS selects N host agents uniformly at random to mark as bots. Next, the simulator uses G to generate an overlay graph O, connecting all of the bots. Finally, at time T, the host agents marked as bots begin running the following traffic generation algorithm:

1. Select a bot to communicate with by choosing one uniformly at random from the host agents that the given bot shares an edge with in O.
2. Initiate a connection with the selected host.
3. Select an inter-communication time from I. At time *currenttime* + *selectedtime*, repeat from 1.

4.5 Executing the Simulation

After generating the host agents and augmenting them with background and malicious behavior processes, LESS signals the host agents to begin their behavior processes. Host agents log their activities: the timestamp, destination, and application used in each connection; and these logs are collected after the simulation finishes. LESS is currently a single threaded application written in Python, and maintains event ordering using a queue. We have found this sufficiently fast (*i.e.* each of our trials finishes in under an hour on an Intel i7 laptop); however, since host agents make decisions autonomously, LESS is well suited to scaling, as we discuss in Section 6.

5 Evaluation

In this section, we analyze the four previously discussed large scale network security systems using LESS. We present two types of results, *reproduced* results that

Table 1. LESS setting for our re-evaluations of entropy-based anomaly detection [1], Highly Predictive Blacklisting [3], peer-to-peer botnet detection [5], and cross domain anomaly detection [2]

Param.	Description	[1]	[3]	[5]	[2]
H	*Number of Hosts*	$100,000$	$100,000$	$100,000$	$100,000$
S	*Simulation Duration*	100	100	100	100
M	*Malicious Model*	Worm Outbreak	Targeted Attacker	P2P Botnet	Worm Outbreak
N	*Malicious Host Count*	$1250 - 5000$	5000	904	5000
T	*Attack Start Time*	0	0	0	0
I	*Inter-attack Distribution*	$\mathcal{N}$ $(.1, .001)$,	From [4]	$\mathcal{N}$ $(.1, .001)$	$\mathcal{N}$ $(.1, .001)$
C	*Target Count Distribution*	-	From [4]	-	-
D	*Number of Peers*	-	-	$5 - 30$	-

validate parts of the original evaluations of these systems, and *extended* results that apply simulation to go beyond the original evaluations. These results serve multiple purposes. First, they demonstrate that the generalized models used by LESS are capable of evoking realistic performance from large scale security systems. Second, they validate and extend the evaluations of the studied systems. Finally, they demonstrate use cases for LESS, and how it complements existing analysis methods.

5.1 Experimental Setup

Table 1 lists the parameters we tuned LESS with, unless otherwise noted. LESS derives configurations for background traffic generation from 60 seconds of CAIDA's anonymized 2012 Internet survey [22], a trace containing approximately 800000 hosts with payloads stripped and prefix preserving host and destination IP anonymization. We also apply the three malicious traffic models described in Section 4 with varying parameters that we describe in a per experiment basis. For the purpose of these evaluations, we also re-implemented each of these systems in Python based on the original published algorithms.

To compare results from LESS based simulations with original results, we converted graphs from the original evaluations into numeric data with the Plot Digitizer tool [28]. All plots in this section labeled *Original* are based off of this digitized data.

5.2 Entropy Based Detector

Wagner et al. tested their entropy based method for detecting large scale security threats on a trace collected during the Code Red [29] worm outbreak. We were able to produce similar results using LESS and the random worm outbreak model with the following parameters: $N = 5000$, $I = \mathcal{N}$ $(.1, .001)$, $T = 0$. Figure 2 illustrates both the original results, and our results from our simulation based re-evaluation.

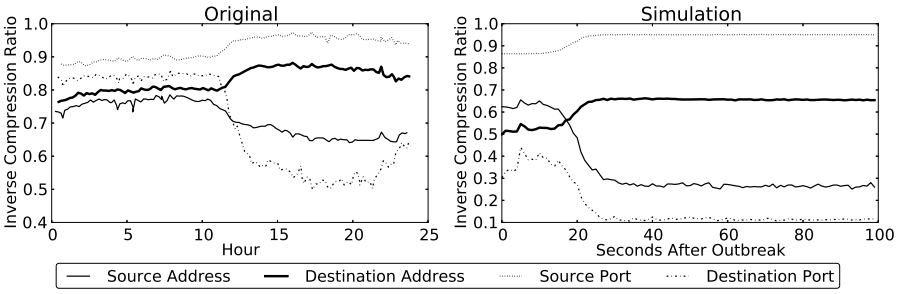

Fig. 2. Original entropy detection results from [1], and entropy detection results in a LESS simulation

There are several quantitative differences between the results. First, the simulated outbreak reached the maximum entropy more quickly, due to the shorter time scale of the simulation. Second, the baseline entropy values differ significantly.We validated that the baseline entropy values in the simulated experiment corresponded to the average entropy values from our source CAIDA trace [1]. Despite the differences between the baseline entropies, these results are *qualitatively* similar: the entropy of each feature either increased in both experiments or decreased in both experiments.

The quantitative differences illustrate a benefit of LESS: by deriving the traffic model from a recent CAIDA trace, we are able to investigate how a system proposed when background network conditions were significantly different would perform in a present-day scenario. Another important benefit of simulation in this domain is the ability to evaluate large scale security techniques under different threat conditions. Figure 3 shows the results of a set of experiments where we used LESS to compare the destination IP address entropy rate during the outbreaks of worms with different average scan per second rates (I) and vulnerable population sizes (N). Scan rate and vulnerable population size both increase the entropy rate and make the change more sudden. However, according to our simulation there is an upper bound on both the maximum entropy rate and how rapidly the change can occur.

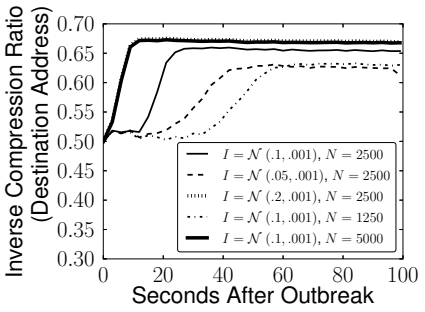

Fig. 3. Extending the results of [1] with LESS, by measuring the technique's sensitivity to different worm outbreak behaviors

[1] average source IP entropy: 0.6133, average destination IP entropy: 0.5618, average source port entropy: 0.9289, average destination port entropy: 0.3518

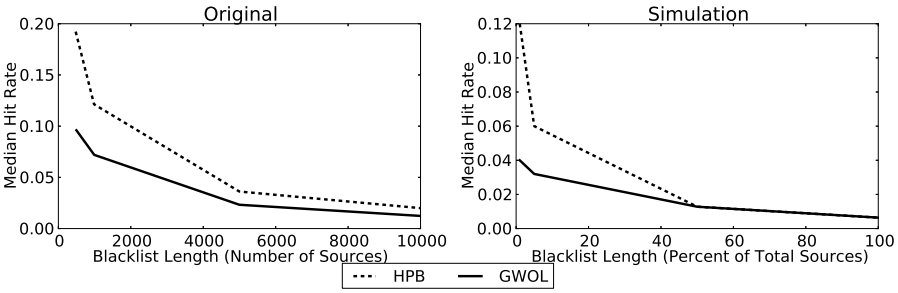

Fig. 4. Predictive (HPB) and Global(GWOL) blacklist hit rates as blacklist length increases, from the original HPB evaluation [3], and simulation experiments with LESS.

5.3 Highly Predictive Blacklisting

Zhang et al. compared the performance of blacklists generated using their system to blacklists generate using a baseline system by measuring blacklist *hit rates*, or the number of IP addresses blocked / the length of the blacklist. They used data from the Dshield repositories for their experiments, so to re-evaluate their system we configured LESS using statistics about Dshield repository logs measured in [4]. We selected values for C, the target count distribution, and I, the inter attack timing distribution, based on their measurements. We used the first half of generated records as input to the blacklist generators, and tested the resulting blacklists with the second half of the generated records.

Figure 4 shows the median hit rates for blacklists generated by the HPB system, and the baseline GWOL alternative, as blacklist length varies, for both the original evaluation and our simulation based re-evaluation. The relative difference between HPB blacklists and GWOL blacklists were similar in both the real and simulated data sets. However, there were quantitative differences: the scale of the hit rates for both types of blacklists were different, and the blacklist hit rates converged sooner in simulation. We believe that there are two primary causes for these differences. First, our simulation likely differed with respect to the number of attackers, as we were unable to determine how many attackers the original data set contained. Blacklist hit rate is very sensitive to this parameter, as it directly affects the number of attacks a network is likely to observe. Additionally, our parameters were derived from measurements of the DShield repository taken approximately 3 years before the HPB system was proposed; the threat landscape, and likely DShield itself, has changed significantly in that time.

Despite these differences, our simulation based analysis reaches the same conclusion as that done on a real, large scale data set: HPB generated blacklists achieve higher hit rates than GWOL generated blacklists, especially when they are short. This demonstrates that LESS allows researchers to reach the same qualitative conclusions, but using significantly higher level and easier to obtain data. Statistical summaries, like those we used as input for this experiment, re-

veal much less sensitive information and are significantly smaller than large scale traces or IDS log sets.

We also extended the evaluation by studying the *detection rate* (*i.e.* percentage of attackers detected by a blacklist) and *false alert rate*(*i.e.* percentage of non-attacker hosts that the blacklist generates alerts for) of generated blacklists. Figure 5 illustrates an experiment where we varied the number of non-malicious records provided to the HPB and GWOL generators, and measured the true and false positive rates of the resulting blacklists. As more false alerts were submitted to the blacklist generation systems, the detection rates of the generated blacklists decreased while their false alert rates increased. In all cases, the HPB blacklists performed better than the GWOL blacklists. However, the benefit of HPB blacklists increased with the number of false alerts, revealing that not only does HPB achieve higher hit rates, but that it is also *more robust to inaccurate input.*

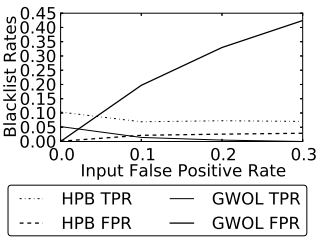

Fig. 5. Extending [3] with LESS: the true and false positive rates (TPR & FPR) of blacklists as the false positive rates of input IDSes varies

Although this experiment was straightforward to perform with LESS, it was impossible using the original data set, which is composed of IDS alert logs that lack sufficient information to determine the true maliciousness of the hosts that caused alerts.

5.4 Peer-to-Peer Bot Detector

Coskun et al. evaluated their peer-to-peer botnet detector with traffic from approximately 900 Nugache bots grafted into a 24 hour trace collected at the border of a university network containing approximately 2000 active hosts.

To evaluate this system with LESS, we used the peer-to-peer malicious traffic model to augment 904 host agents with malicious behavior. We set the malicious timing distribution $I = \mathcal{N}\ (.1, .001)$, and generated an overlay network for the bot hosts using the NetworkX [30] library to generate a random regular graph [31]. The regular graph generation algorithm requires one parameter: D, the desired degree of each node in the generated graph.

We then replicated the original experimental procedures with the generated data set, selecting M of the 904 agents augmented with the bot behavior at random, to act as the bot nodes in the simulated evaluation network, and $2000 - M$ non-malicious host agents at random, to act as the innocuous hosts in the simulated evaluation network.

Figure 6 shows measurements taken in both the original re-evaluation, and our re-evaluation with LESS (using $D = 21$), of the number of bot and non-bot hosts detected by the system as M, the number of bots grafted into the trace, varied. Despite our background traffic being modeled after data collected from a different network, our results were very similar to the original results, suggesting that their approach would generalize well to other networks.

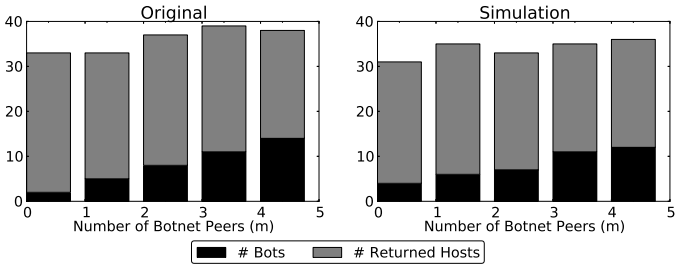

Fig. 6. Number of bots and non-bots detected by the system proposed in [5], in the original evaluation and our LESS based re-evaluation

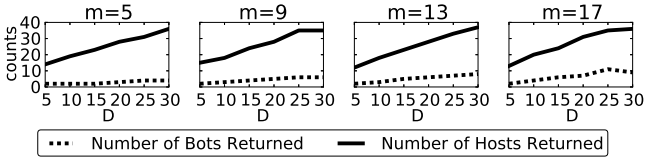

Fig. 7. Extending [5]: evaluating the performance of the botnet detector as bot structure, parameterized by (D), varied in LESS simulations

We also extended the results from the original evaluation, by analyzing the technique's performance as D changed. Figure 7 shows how the number of bots detected and number of hosts returned change, for different M values, as D varied between 1 and 21. This result demonstrates that the density of the botnet communication overlay graph has a large effect on the effectiveness of the system: as the density decreased, so did the accuracy of the detection technique, particularly for larger M values.

Also, this result demonstrates that LESS allows researchers to work backwards from a published result to determine properties that the underlying data set is likely to have. As an example, in our preliminary research with the bot detection technique, we first generated botnet communication graphs using preferential attachment [32] and Erdos-Renyi [33] models, based on previous results that suggested these models fit peer-to-peer overlay networks. However, we found that these models led to less accurate reproduction of the original results, leading us to conclude that the random regular graph generation algorithm is a better fit for the communication network of the Nugache bots used in the original evaluation.

5.5 Collaborative Anomaly Detection

Boggs et al. tested their proposed cross-domain anomaly detection technique on a small deployment involving 3 HTTP servers located at different administrative domains. Using LESS, we examined the potential for larger scale deployment of a similar correlation technique. We implemented a simple threshold based

anomaly detector, that monitors all the hosts in a network and generates an alert whenever an external host initiates more than T connections with hosts inside the network in a 1 second period of time. Each network determines T independently, by measuring the distribution of external to internal connections per second in a training data set, and then setting T to the 75th percentile value.

For this experiment, we ran LESS with the worm outbreak model and parameters set to $N = 5000$, $I = \mathcal{N}(.1, .001)$, and $T = 0$, and divided the $100,000$ host agents into networks of uniformly at random selected sizes between 2 and 128. We submitted the first half of generated traffic to the training process for each network, and then ran the anomaly detector on the traffic received by each network during the second half of the simulation. We then compared the false positive rates of two different strategies for raising alerts, depicted in Figure 8: first, an *autonomous* strategy, in which each network raises an alert for each anomalous IP address detected; second, a *collaborative* strategy, in which each network raises an alert for an anomalous IP address only if the IP address has generated an alert at another collaborating network.

The benefits of collaboration increase very rapidly, with orders of magnitude fewer false alerts when under 20 networks participate in the collaborative system, suggesting that even a small deployment would have large benefits. There are many other questions about collaborative defense which we do not address here due to space constraints (*e.g.* who should collaborate? what kind of information should collaborators share? how does scale affect detection rate). Such questions are relevant to all the systems we have studied. LESS allows researchers to explore these issues, without facing the often impossible challenge of acquiring a large scale data set, or data sets from many different administrative domains.

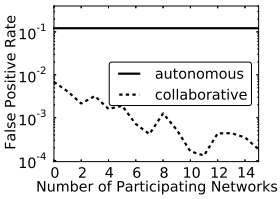

Fig. 8. Extending [2]: evaluating the benefit of cross-domain collaborative anomaly detection when used at scale, in a LESS simulation

6 Discussion

Section 5 demonstrates that LESS simulations can evoke realistic performance from large scale and collaborative network security systems, and has many use cases. In this section, we discuss current limitations and potential expansions to our simulator, as well as integration with other systems and techniques.

Long Term Temporal Dynamics and System Evolution: LESS, the security systems we evaluated, and the threats they were designed to defend against, all only take short term temporal properties, such as inter-arrival time, into account. However, the Internet is a dynamic, evolving network: new hosts connect to it and network applications are launched; hosts change IP addresses and networks; and user behaviors follow diurnal patterns [8]. These, and other long term dynamics, could potentially affect the performance of security systems. By integrating evolutionary network models [34] and parameters that describe longer

term dynamics into host agent models, our simulator could be used to evaluate their effects.

Flow Payloads and Packets Due to the volume of traffic that large scale network security systems analyze, most are prohibited from analyzing traffic at the payload or packet level. LESS generates traffic records that do not contain these details. However, LESS hosts' traffic generation processes could be augmented with payload and packet generation techniques such as [11]. This would allow our simulator to evaluate a broader class of security systems, and provide a platform to investigate the benefits of large scale collaboration and data sharing for security systems that monitor lower level data.

Scalability and Testbed Integration Our current implementation of LESS is single threaded. However, each host agent behaves autonomously, deciding, on its own, when to generate traffic and which other host agents to communicate with. Due to the decentralized nature of this process, LESS's architecture is well suited for deployment on a physical or virtual testbed [10, 9].The benefits of distributing the agents across such a testbed are twofold: first, it would distribute the workload, providing faster simulations; second, and more importantly, it would allow us to also model and evaluate the effects of network topology on large scale security systems.

7 Conclusion

Large scale and collaborative security systems have demonstrated great potential. However, scale also presents data related research challenges. Although simulation is an effective tool for overcoming these data challenges in non-security domains, existing simulation tools are not well suited to evaluating large scale security systems. We propose a simulator designed specifically for the large-scale security system domain, LESS, which generates host agents, configures them with stochastic behavior processes, and monitors their activities throughout simulations to collect experimentation data sets. LESS configures non-malicious host behaviors with measurements from real traces, and malicious host behaviors with user defined threat models. We used LESS to validate and extend the evaluations of four recently proposed large scale security systems, demonstrating not only that LESS generates realistic and usable data sets, but also that LESS can compliment existing analysis techniques and real data by allowing researchers to evaluate systems in different dimensions. LESS, and future specially designed simulation tools, can help researchers analyze more complex issues and advance the promising field of large scale and collaborative security.

Acknowledgements. We wish to thank the anonymous reviewers for their feedback. This research was partially supported by ONR grants N001614WX30023 and N00014-12-1-0757, and DoD grant contract number H98230-14-C-0137.

References

[1] Wagner, A., Plattner, B.: Entropy based worm and anomaly detection in fast ip networks. In: WETICE (2005)

[2] Boggs, N., Hiremagalore, S., Stavrou, A., Stolfo, S.J.: Cross-domain collaborative anomaly detection: so far yet so close. In: Sommer, R., Balzarotti, D., Maier, G. (eds.) RAID 2011. LNCS, vol. 6961, pp. 142–160. Springer, Heidelberg (2011)

[3] Zhang, J., Porras, P., Ullrich, J.: Highly predictive blacklisting. In: USENIX Security, vol. 8, pp. 107–122 (2008)

[4] Katti, S., Krishnamurthy, B., Katabi, D.: Collaborating against common enemies. In: ACM IMC (2005)

[5] Coskun, B., Dietrich, S., Memon, N.: Friends of an enemy: identifying local members of peer-to-peer botnets using mutual contacts. In: Proceedings of the 26th Annual Computer Security Applications Conference (2010)

[6] Sonchack, J., Aviv, A., Smith, J.M.: Bridging the data gap: Data related challenges in evaluating large scale collaborative security systems. In: 6th Workshop on Cyber Security Experitmentation and Testing (2013)

[7] Aviv, A.J., Haeberlen, A.: Challenges in experimenting with botnet detection systems. In: USENIX 4th CSET Workshop (2011)

[8] Floyd, S., Paxson, V.: Difficulties in simulating the internet. IEEE/ACM Transactions on Networking (TON) 9(4), 392–403 (2001)

[9] Riley, G.F.: The georgia tech network simulator. In: Proceedings of the ACM SIGCOMM MoMeTools Workshop, pp. 5–12. ACM (2003)

[10] Lantz, B., Heller, B., McKeown, N.: A network in a laptop: rapid prototyping for software-defined networks. In: Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks, p. 19. ACM (2010)

[11] Weigle, M.C., Adurthi, P., Hernández-Campos, F., Jeffay, K., Smith, F.D.: Tmix: a tool for generating realistic tcp application workloads in ns-2. ACM SIGCOMM Computer Communication Review 36(3), 65–76 (2006)

[12] Konda, V., Kaur, J.: Rapid: Shrinking the congestion-control timescale. In: IEEE INFOCOM 2009, pp. 1–9. IEEE (2009)

[13] Cao, J., Cleveland, W.S., Gao, Y., Jeffay, K., Smith, F.D., Weigle, M.: Stochastic models for generating synthetic http source traffic. In: INFOCOM 2004, vol. 3, pp. 1546–1557. IEEE (2004)

[14] Moore, D., Shannon, C., Voelker, G.M., Savage, S.: Internet quarantine: Requirements for containing self-propagating code. In: IEEE INFOCOM 2003, pp. 1901–1910. IEEE (2003)

[15] Grizzard, J.B., Sharma, V., Nunnery, C., Kang, B.B., Dagon, D.: Peer-to-peer botnets: Overview and case study. In: HOTBOTS, pp. 1–8 (2007)

[16] Sommers, J., Yegneswaran, V., Barford, P.: Recent advances in network intrusion detection system tuning. In: IEEE 40th Annual CISS, pp. 1490–1495 (2006)

[17] Chen, G., Gray, R.S.: Simulating non-scanning worms on peer-to-peer networks. In: ACM INFOSCALE, p. 29 (2006)

[18] Rossey, L.M., Cunningham, R.K., Fried, D.J., Rabek, J.C., Lippmann, R.P., Haines, J.W., Zissman, M.A.: Lariat: Lincoln adaptable real-time information assurance testbed. In: IEEE Aerospace Conference Proceedings 2002, vol. 6, pp. 6–2671. IEEE (2002)

[19] Bonabeau, E.: Agent-based modeling: Methods and techniques for simulating human systems. PNAS 99(suppl. 3), 7280–7287 (2002)

[20] Ripley, B.D.: Stochastic simulation, vol. 316. Wiley. com (1987)

[21] Dshield.org, http://www.dshield.org/
[22] Caida data overview, http://www.caida.org/data/overview/
[23] Argus: Audit records generation and utilization system, http://qosient.com/argus/
[24] Xie, G., Iliofotou, M., Keralapura, R., Faloutsos, M., Nucci, A.: Subflow: Towards practical flow-level traffic classification. In: IEEE INFOCOM, 2012 Proceedings, pp. 2541–2545. IEEE (2012)
[25] Tan, G., Poletto, M., Guttag, J.V., Kaashoek, M.F.: Role classification of hosts within enterprise networks based on connection patterns. In: USENIX Annual Technical Conference, General Track, pp. 15–28 (2003)
[26] Blondel, V.D., Guillaume, J.L., Lambiotte, R., Lefebvre, E.: Fast unfolding of communities in large networks. Journal of Statistical Mechanics: Theory and Experiment 2008(10), P10008 (2008)
[27] Hutchins, E.M., Cloppert, M.J., Amin, R.M.: Intelligence-driven computer network defense informed by analysis of adversary campaigns and intrusion kill chains. Leading Issues in Information Warfare & Security Research 1, 80 (2011)
[28] Plot digitizer, http://plotdigitizer.sourceforge.net/
[29] Moore, D., Shannon, C., et al.: Code-red: a case study on the spread and victims of an internet worm. In: Proceedings of the 2nd ACM SIGCOMM Workshop on Internet Measurment, pp. 273–284. ACM (2002)
[30] Hagberg, A., Swart, P., Schult, D.: Exploring network structure, dynamics, and function using networkx. Technical report, Los Alamos National Laboratory, LANL (2008)
[31] Steger, A., Wormald, N.C.: Generating random regular graphs quickly. Combinatorics Probability and Computing 8(4), 377–396 (1999)
[32] Barabási, A.L., Albert, R.: Emergence of scaling in random networks. Science 286(5439), 509–512 (1999)
[33] Erdos, P., Renyi, A.: On random graphs i. Publ. Math. Debrecen 6, 290–297 (1959)
[34] Dorogovtsev, S.N., Mendes, J.F.: Evolution of networks. Advances in Physics 51(4), 1079–1187 (2002)