2018 IEEE Symposium on Security and Privacy Workshops

Simulated User Bots: Real Time Testing of Insider Threat Detection Systems

Preetam K. Dutta, Gabriel Ryan, Aleksander Zieba and Salvatore J. Stolfo
Department of Computer Science
Columbia University
New York, NY 10027, USA
{pkd2108, gr2547, arz2116, sjs11}@columbia.edu

***Abstract*—The insider threat is one of the most serious security problems faced by modern organizations. High profile cases demonstrate the serious consequences of successful attacks. The problem has been studied for many years, leading to a number of technologies and products that have been deployed widely in government and commercial enterprises. A fundamental question is: how well do these systems work? How may they be tested? How expensive are widely deployed monitoring infrastructures in terms of computational cost?**

Measurement of real systems, which are dynamic in nature, encounter unknown configuration bugs and have sensitivities to the vagaries of human nature and adversarial behavior, requires a formal means to continuously test and evaluate deployed detection systems. We present a framework to deploy *in situ* simulated user bots (SUBs) that can emulate the actions of real users. By creating a user account and by running a host in the enterprise network, a SUB can be introduced into an enterprise system that runs at a realistic pace and does not interfere with normal operations. Infusing malicious behavior into the SUB should be detected by the insider threat monitoring infrastructure. The SUB framework can be controlled to explore the limits of deployed systems and to test the effectiveness of insider evasion tactics, especially low and slow behaviors.

We demonstrate our framework in a synthetic ecosystem as well as in a live enterprise deployment. We created a synthetic environment of users based on data collected in a West Point cadet study. Various machine learning based intrusion detection algorithms are used to validate the ability of the SUB framework to generate both normal and malicious users. In a live University network, we launched a number of attacks on its intrusion detection system and showcased the ability to devise malicious users. In addition, we further deployed low and slow attacks that perform malicious actions over an extended period of time and demonstrate how even a large enterprise is ill equipped to combat such attacks.

I. Introduction

The insider threat is among the most severe security threats in modern organizations. A number of technologies and products are being deployed broadly [1]–[3]. The User Behavior Analytics (UBA) market is predicted to approach $1 billion by 2021 [4]. The current generation of available commercial products is based upon network monitoring sensors and large scale data analytics that compute user models to detect abnormal behaviors indicative of malicious acts (eg., [5], [6]). A fundamental question is how well do they actually work and at what computational cost?

In this paper, we introduce simulated user bots (SUBs), a novel system designed to create realistic user behaviors in an enterprise computing environment that can be programmatically designed to simulate normal or malicious users. Similar to BotSwindler [7], we employ an endpoint agent to run host applications to mimic a real user without interfering with the normal operations on the enterprise system. The SUBs generate network and host data in the same manner as ordinary users generate those data, but can be controlled to inject any behavior we wish. There are no traces that would clearly indicate that the SUBs are not real users in the system logs. To generate simulations, SUBs are dependent on a database, a corresponding translator and advanced user generation modeling, which culminates in actions files. The SUBs are identical to users in almost every facet from having to login to the speed at which a user modifies a document.

For the purposes of training our models, we used a detailed dataset consisting of 63 West Point cadets, containing up to two months of real usage data. Despite being from a homogeneous organization, users in the dataset were unique in their behaviors. Statistics were gathered on the users in terms of the average frequency of visits to particular types of websites, the types of activities on files, etc.

The SUB behaviors introduced in this paper are designed to generate trace data typically used as indicators and detectors in identifying malicious users by insider threat systems. We enumerate a sample of typical indicators used in deployed systems in Section 2c. These indicators are temporal statistics derived from an analysis of monitored network logs. These statistics derive group norms from which abnormal users are identified. A clever insider adversary can avoid detection and it is thus of paramount importance to consider more robust indicators and detectors.

Controlling the pace and the frequency of these trace indicators generated by a SUB provides the means of testing the detection system at its margins, and hence can provide a detailed analysis of the ease or the difficulty of evading detection. Errors in the deployed monitoring infrastructure, either due to bugs in configurations or noise introduced by faulty sensors, may also be revealed if a SUB is undetected although it is directed to purposely exhibit the indicator. Of particular interest is whether SUBs may be used to measure the computational costs of maintaining long term temporal statistics. Low and slow behaviors would cause an insider threat monitoring system to keep long term state information

DOI 10.1109/SPW.2018.00038

for many users, which would cause an increasing cost in terms of storage and computation. Hence, the SUB testing framework may also provide a means of evaluating not only the accuracy of a system, but its computational costs.

We have implemented our SUB framework using QEMU [8], running a Linux host. The SUB user actions are learned from a database of real user behavior previously acquired in prior work, which are modeled extensively and translated to Python action files, which make use of our augmented version of VNCDoTool [9]. The SUB framework can be applied to any host operating system, but our work presently focuses on Linux. We present tests on various SUB configurations on many commonly used algorithms for insider threat detection such as Gaussian Mixture Models, Support Vector Machines and Bayesian Networks. The results from our experiments validate our ability to modify the behavior of our SUBs and to create innocuous ones as well. The ability to programmatically create dynamic users demonstrates the ease of deployment of SUBs in real systems at little to no cost other than creating SUB user accounts within an enterprise system. We conduct live enterprise based experiments on a university wide network with its own intrusion detection system. Again, we are able to create users that go unnoticed and others that trigger the alerts of the system.

The remainder of the paper will be organized as follows. In Section II, we describe the design of the simulated user bot development framework. We present both our experiments and results in Section III. In Section IV, we detail related work. The final two parts, Sections V and VI, are dedicated to future work and the conclusions of our work.

II. Design and Architecture of SUB Framework

SUBs are created to replicate a real user with no hint of artificiality from the view of the system and its anomaly detection system. To create users in this manner builds on both tools previously used for other purposes, but modified and enhanced, and those specially built for SUBs. The data generated from the SUB process is approximately identical to a real user in a system and can be fine-tuned to have either normal or abnormal tendencies. The user behaviors are derived from an existing database of users that has been extensively modeled and allows for greater control over the types of behaviors for a given synthetic user. The framework is easily modifiable and the results are not random as each user is reproducible given a specified set of actions and controls. The framework of our SUB setup consists of three main steps, which can further be generalized into the following layers: data, simulation and deployment. An overview of the process is shown in Figure 1 and explained in the following sections.

A. Data Layer

The data layer is composed of the first step in the diagram. Strong data is pivotal in our experiments as we impose precise actions for our simulated users to perform once in a simulated environment. To that end, we employed a SUB database translator, written in Python. The translator takes a database

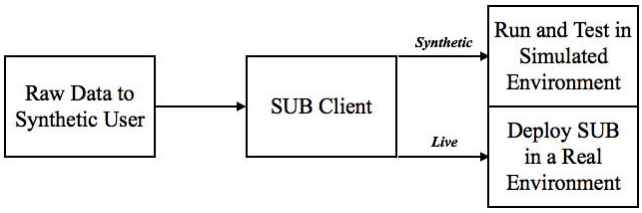


Fig. 1. Simulated User Bot System Overview

stored in SQL with at least the following fields: user ID, event and timestamp. It then converts those fields into Python action files for users. Python action files can be thought of as providing directions for the SUB to follow and they have the logical sequences of actions for the SUB considered. We have defined various rules for our translator such that the translated actions can be performed on a SUB that does not have necessarily have access to existing files on a real user's local machine. We have created rules for all the various types of interactions a normal user may have on his or her local machine. The following is an example of our rules for a user's web browsing:

1) Derive the browser used by a particular user by searching through the information provided in a given browser tab if the information is not provided otherwise.
2) Perform the search with the same service as a given user by searching through the information contained in the browser tab unless the information is provided in another field.
3) Search through the browser tab for information regarding the email service used unless provided explicitly by the dataset.
4) Search social media site specified in the browser tab unless provided in the dataset.
5) Browse to the website referred to if the browser tab contains a full HTTP URL.

After the database has been translated, we have action files for each of the users. However, we continue to refine our data and model it using long short term memory models [10], a variant of a recurrent neural network architecture. The high level schematic for our approach is shown in Figure 2. We verify that our users generated using this technique are effectively indistinguishable from the original population by dividing our original group of users into three groups based on their term frequency inverse document frequency (TF-IDF). The clusters were created using a Gaussian Mixture Model that minimized the Bayesian Information Criterion. After the clusters were formed, the models were trained on each individual cluster to generate synthetic data. Training data were assembled by sampling three users from each group and then training models for 30 epochs with a learning rate of 1.5 and decay rate of 0.1. We reasoned that if a synthetic user from a given group was reclassified into its original group that our model represents the original population reasonably well.

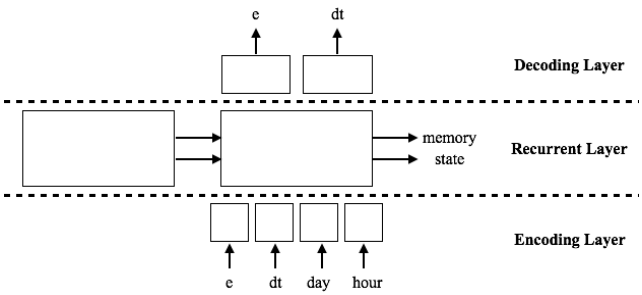


Fig. 2. High Level Overview of LSTM Framework

Once the users have been modeled, we can generate a host of diverse users with the parameters we wish to alter such as the frequency of time spent on social media websites. User modeling was used primarily to create new users who could not be found directly in the existing dataset yet shared characteristics of the original data. We also note the timestamps to be aware of the gap between activities. It is unrealistic to assume that a user proceeds directly from one task to the next without a pause. The delta in time between activities in the database allows us to train our users to have pauses between actions and helps improve our overall level of realism. For all our work, we translate all of our raw data into the translated action files even if the user is not altered. This step is taken to guarantee that users are not different from one another solely on the basis of whether or not our translator was employed.

B. Simulation Layer

The simulation layer is handled by our SUB client. Once the data has been translated into action files using the aforementioned method, we feed the actions for our predefined users into the automation framework built upon Quick Emulation (QEMU) and connected through virtual network computing (VNC), using VNCDoTool. VNCDoTool is a well written tool, but still suffers from dropped actions that can result in a serious error for a SUB. For example, if the user is instructed to open a web browser, but fails to do so then the subsequent actions will be compromised. Therefore, each feature must have a state checking mechanism that allows it to continue executing even if one of its calls to VNCDoTool does not work properly. While this may sound like a daunting task, one solution is to take a screenshot of an indicative portion of the screen both before and after the action is supposed to be performed, and if that portion changes, it is indicative that the action succeeded. Certain actions such as typing specific strings require more sophisticated state checking mechanisms, but this is underlying logic used to create our Python extension of VNCDoTool.

Since we record the logs at the end of a run by our SUBs, it is paramount that there is no extraneous activity conducted locally, which is guaranteed by using a virtual connection. It is further important that there is no external activity noted by an intrusion detection system that could detect a client or any other local software being run. To boost the performance of our SUBs, we make use of Kernel Virtual Machine (KVM), which allows for hardware virtualization and increases the performance of the SUB client by over 40 percent. Our automation framework allows for a simulated user to perform any task a regular user would be able to do: login to an account, send an email, open a website, create and modify documents, etc. An example script and action is displayed in Figure 3.

In some cases, we used an open source equivalent of a commercial software to avoid any potential limitations with licensing. For example, we used Apache OpenOffice instead of the corresponding Microsoft Office suite. The open source software works almost identically and handles all of the features that we observed in the database. Thus, a negligible amount of realism was compromised by using the open source alternative. We have the ability to capture both the system and web history logs after a completed run. The logs allow us to understand the subtle underpinnings of the system when performing the various actions that a normal user would in his or her day to day activities. We originally experimented with several nonnative solutions, including the implementation of a feature to capture all actions on a machine and the corresponding logs. However, external solutions leave traces of sensors in the logs themselves and we therefore opted to use the logs produced by the host operating system and capture them after the completion of a run.

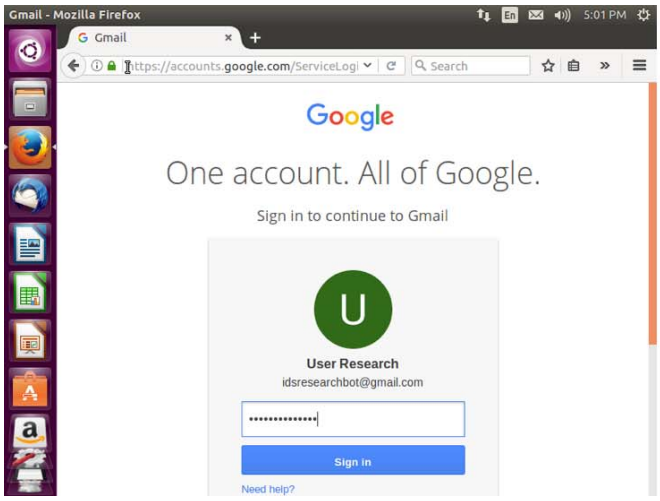
C. Deployment Layer

The deployment layer varies based on whether we are testing in a live ecosystem or in our simulated environment, where we can control the intrusion detection algorithms.

Live testing is conducted on the Columbia University Network, which is a large research university network with over 55,000 MAC addresses active on average, approximately 80,000 active email addresses and over 100,000 network nodes. There is no sniffing traffic and scanning machines are not allowed. There are additionally no firewalls. We created users on the network and then allowed our SUBs to autonomously run their actions.

From work declassified and made available during the IARPA's Scientific advances to Continuous Insider Threat Evaluation (SCITE) program, we learned about the types of relationships between the threat type, behaviors, indicators and detectors from their redacted challenge problems as can be seen in Table 1. The insight into how real anomaly detection systems are constructed helped us define our own system for experiments using our SUBs. For the purposes of illustrating the effectiveness of our SUB framework, we then consider three commonly used algorithms to detect anomalous behavior: Gaussian Mixture Models, Support Vector Machines and Bayesian Networks.

1) Gaussian Mixture Model: A Gaussian Mixture Model is a probabilistic model that has gained popularity in intrusion detection systems. Promising results have been found using Gaussian Mixture Models in well known datasets such as the




```
move 0 0
rexpect vdo/firefox.png 0 100 0
move 25 175
click 1
rexpect vdo/firefox_search_bar.png 100 25 0
move 150 75
click 1
type "gmail.com”
key enter
rexpect vdo/gmail_loaded.png 100 0 0
type [redacted password]
key enter
```

Fig. 3. Example of Simulated Email Login

TABLE I
EXAMPLE OF THREAT TYPES, BEHAVIORS, INDICATORS AND DETECTORS

Threat Type	Behavior	Indicator	Detector
Individuals with abnormal work habits	Uses a work-owned machine outside of normal work hours (i.e., 12AM-7AM EST) at work, at home, or at remote sites	Has anti-virus updates between 12AM and 7AM EST	At least 1 anti-virus log entry for definition updates between 12AM and 7AM EST
		In the top 5% of the frequency distribution of VPN activity between 12AM and 7AM EST	At least 28 VPN log records showing a connection that started between 12AM and 7AM EST
		In the top 5% of the frequency distribution of workstation logins between 12AM and 7AM EST	At least 128 ADDC log entries with timestamps between 12AM and 7AM EST
		In the top 5% of the frequency distribution of email activity between 12AM and 7AM EST	At least 24 email log records for outbound emails sent between 12AM and 7AM EST
		In the top 5% of the frequency distribution of website activity between 12AM and 7AM EST	At least 24,500 proxy log entries with a timestamp between 12AM and 7AM EST
Individuals with abnormally large data transfers that are not easily explained	Uses a work-owned machine for abnormally large inbound or outbound data transfers	In the top 5% of the frequency distribution of attempted access to prohibited file sharing websites	At least 2000 proxy log entries of attempted access to prohibited file sharing websites
		In the top 5% of the frequency distribution of large emails sent	At least 3 emails with attachments larger than 5 MB
		In the top 5% of the frequency distribution of VPN sessions that transfer large amounts of data	At least 7 VPN sessions where at least 210 MB are transferred
		In the top 5% of the frequency distribution of firewall log entries that transfer large amounts of data	At least 3 firewall log entries where at least 200 KB is transferred
		In the top 5% of the frequency distribution of web browsing sessions that transfer large amounts of data	At least 1 proxy log entry where at least 51 MB is transferred
		In the top 5% of the frequency distribution of USB attachments	At least 11 application white listing log entries of devices attached or mounted
Individuals that have unusual contact with foreign entities	Digitally communicates with non-US entities through a work-owned machine	In the top 5% of the frequency distribution of outbound emails to non-US email addresses	At least 1 outbound email to a non-US email address
		Have VPN sessions from non-US locations	At least 1 VPN session initiated from a non-US location
		In the top 5% of the frequency distribution of firewall log entries to non-US locations	At least 37 firewall log entries to non-US locations
		In the top 5% of the frequency distribution of web browsing sessions to non-US websites	At least 7100 proxy log entries to non-US owned websites

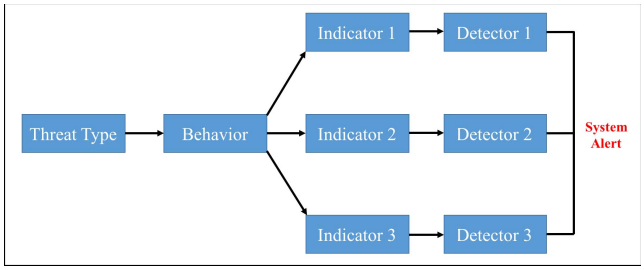


Fig. 4. Bayesian Network Representing IARPA SCITE Challenge Problem

KDD99 data set from Lincoln Labs of MIT [11]. The model itself can be represented as shown in Equation 1.

$$p(\vec{x}|\lambda) = \sum_{i=1}^{M} w_i g(\vec{x}|\vec{\mu_i}, \vec{\Sigma_i}), \tag{1}$$

where $\vec{x}$ is a D-dimensional continuous value data vector, w_i are mixture weights and $g(\vec{x}|\vec{\mu_i}, \vec{\Sigma_i})$ are component Gaussian densities. A key differentiating factor between Gaussian Mixture Models and k-means clustering is the inclusion of the covariance structure of the data. The EM algorithm is run to estimate the parameters of the model. The process is done in a way such that there is a guaranteed monotonic increase in the model's likelihood value. Further details on Gaussian Mixture Models and the EM algorithm can be found in the reference by Douglas Reynolds [12].

2) Support Vector Machine: Support vector machines are normally supervised classifiers that attempt to map input vectors into a high dimensional feature space using optimal hyperplanes, those with the maximal margin between the vectors of the two classes, for separable classes. The details of the algorithm are described in detail by Cortes and Vapnik [13]. However, researchers a few years later pushed the concept further to tackle unlabeled data, better known as unsupervised learning. The method attempts to find a function that is positive on a subset of the input space and negative on the complement by mapping the input data into a higher dimensional space and using the origin as a negative training point. The objective function is given by

$$\begin{aligned} \min_{w \in F, \xi \in R^l, \rho \in R} & \frac{1}{2}||w||^2 + \frac{1}{vl}\sum_i \xi_i - \rho \\ \text{s.t. } & (w \cdot \Phi(x_i)) \geq \rho - \xi_i, \xi_i \geq 0 \end{aligned} \tag{2}$$

where ν is a parameter between 0 and 1 that controls how tightly the support vector machine fits the data. Complete details on the algorithm can be found in their seminal work [14].

3) Bayesian Network: Bayesian Networks make use of probabilistic relationships among variables of interest in an acyclic graphical model. Figure 4 shows an example of how such a network can be used to model a threat type, behavior, indicators and detectors. The flow of the diagram was adapted from challenge problems released by an external third party operating and analyzing a deployed insider threat solution and is designed to be representative of a real system. Formulaically, a Bayesian network can be represented as follows:

$$P(X_1, X_2, \ldots, X_n) = \prod_{i=1}^{n} P(X_i|\pi_i) \tag{3}$$

where $X_1, X_2, \ldots, X_n$ represents random variables and π_i is the set of parents of X_i. In many regards, it is more difficult to learn the structure of a Bayesian network than it is to learn the parameters. The EM algorithm can again be used to learn the parameters when the structure of the network is known. It often takes expert knowledge to suggest a structure for a given problem. Further details on the algorithm are contained in [15].

III. Experiments and Results

A. Data and Preparation

1) Data: We used a dataset collected from a group of 63 West Point cadets [1]. The number of cadets was a result of the number of cadets that volunteered for the study and had data was successfully extracted from their machines. Two additional cadets were in the study, but the quality of data extraction was too low to be considered. Each cadet had software installed on his or her machine to track usage. The earliest installations were January 15, 2015, and the latest installation was on February 13, 2015. Each user had a participant/device Windows System ID and unique ID number. The cadets had up to three extraction dates for the data from their machines, ranging from February 10, 2015, for the first pull to March 12, 2015, for the last data collection. Despite crashes from the installed sensors, several gigabytes of data were collected for each of the users on average.

The participants were assigned different labs to simulate masquerader data within the dataset. Participants were given different labs to prevent adjacent sitting cadets from influencing each other's work. Time stamps were taken from the desktop of the device as soon as the masquerader turned the platform on or off. There were two masquerader labs in total. For the purposes of our work, we opt to not use the imposter data explicitly and remove the data points from the dataset since our primary objective is to replicate how a normal user acts on a machine without being directed to perform a certain set of actions.

Information contained within the dataset includes, but is not limited to, a unique ID for each user, a time stamp for a given action, an action column to describe the event that takes place and a details field that provides more information on a given action. The details field contains specific information such as the exact terms searched for in a search, the title of a page visited, etc. The West Point dataset is composed of cadets, which is a nearly homogeneous population in terms of activity. However, as can be seen in Figure 5, the number of visits to social media websites vary widely from one cadet to

[1]Data from the West Point cadets was gathered under an IRB approved protocol

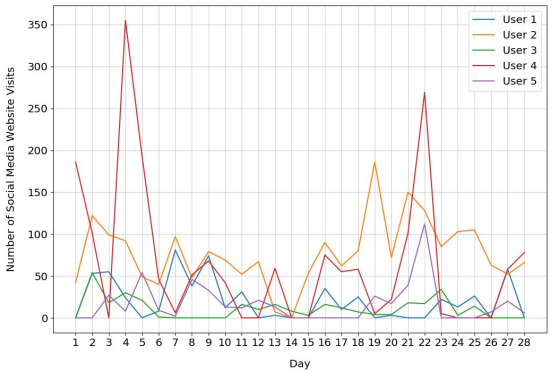


Fig. 5. Social Media Website Visits from West Point Cadets

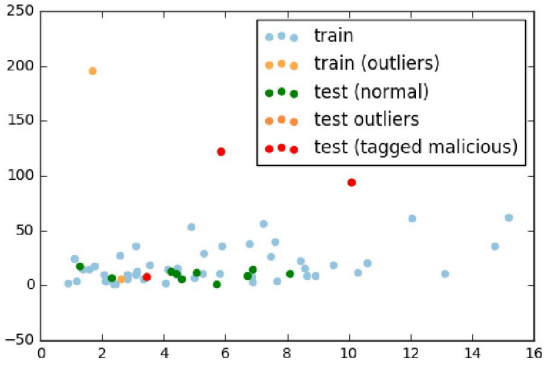


Fig. 6. GMM, SVM and BN Run on Injected, Anomalous SUBs

the next. The same observation can be made for any variable behavior and such diversity of behavior is useful for machine learning algorithms.

2) Hardware: We used an Ubuntu based server with 128 gigabytes of RAM (DDR3, 1066MHz) and six physical cores (Intel(R) Xeon(R) CPU E5-2620 v2 @ 2.10GHz). The server is also equipped with two Tesla K40c video cards, which are helpful for more rapid training of the LSTMs. However, we ran all of our local environment experiments on a MacPro with sixteen gigabytes of RAM (DDR3, 1066MHz) and four physical cores (Intel(R) Xeon (R) CPU E5520, 2.27 GHz). The operating system on the host machine has been changed from the default OS X to Ubuntu 16.04 Desktop. QEMU has been developed for both OS X and Linux distros, but performance is much better on Linux. Our simulations are run on an image running Ubuntu 16.04 Desktop as well. However, our framework can be adapted to work on Windows-based SUBs as well.

3) Threat, Behavior, Indicators and Detectors: We will use the challenge problems from IARPA SCITE as a guide to set our threat, behavior, indicators and detectors. We define our threat to be individuals who use their machines with abnormal work habits. The behavior associated with this threat is a user who uses their machine outside normal work hours (between 5:00:01 PM and 6:59:59 AM EST).

We will consider three indicators of this behavior:

1) In the top 5 percent of the daily frequency average distribution of Google or Bing searches between 5:00:01 PM and 6:59:59 AM EST
2) In the top 5 percent of the daily frequency average distribution of social media website visits between 5:00:01 PM and 6:59:59 AM EST
3) In the top 5 percent of the daily frequency average distribution of actions on files and documents between 5:00:01 PM and 6:59:59 AM EST

This corresponds to the following three detectors:

1) At least 13 log entries for a Google or Bing search between 5:00:01 PM and 6:59:59 AM EST.
2) At least 61 log entries for a social media website visit between 5:00:01 PM and 6:59:59 AM EST.
3) At least 90 log entries for actions on files and documents between 5:00:01 PM and 6:59:59 AM EST.

B. Experimental Setup, Results and Discussion

1) Synthetic Environment: For our experiments, we used a shuffling algorithm to split the 63 West Point cadets into the training group of 50 users and the remainder were placed in the testing group. We selected Gaussian Mixture Models as the first method to run on the data. After running the algorithm, we determined anomalous users in the test data composed of 13 users. We then randomly selected three users in the test data who were not originally marked by the algorithm and added behaviors to their actions to make them appear anomalous to test our ability to create anomalous SUBs. We then ran the previously described Gaussian Mixture Models, Support Vector Machines and Bayesian Network algorithms to detect malicious users.

2) Gaussian Mixture Models: In our experiments, we used the detectors listed above as the three features of our users and set the number of components in our Gaussian Mixture Models correspondingly. We also set the false positive rate to be 5 percent since we assume that all of our users are normal. Gaussian Mixture Models were able to detect all three of the injected malicious users.

3) Support Vector Machines: Similarly, using Support Vector Machines, we were able to detect all three of our injected malicious users. We used a grid search for hyperparameter tuning for the algorithm since poor parameters resulted in poor performance. After searching, we found the optimal values for the parameters with $\nu = 0.264$ and $\gamma = 1.0$. We additionally used the Gaussian kernel, which performed better than sigmoid kernel.

4) Bayesian Networks: Given the importance of expert knowledge, we were not surprised to find that the thresholds had to be adjusted to detect any of the users as malicious. This is given the simple network we considered. We adjusted

the thresholds for the detectors as follows: Detector 1 to 8, Detector 2 to 40, and Detector 3 to 50. We then assumed that the probability of an intruder given any single detector is $\frac{1}{3}$ and that two detectors being set off meant that the probability of an intruder increased to $\frac{2}{3}$. Using the probabilities listed, we were able to detect all three of our injected malicious users. The difficulty to define conditional probabilities a priori is a major challenge with Bayesian networks and explains the use of either expert knowledge or significant data analysis from an enterprise environment. Without access to either, one can only make basic assumptions that are unlikely to be reflective of reality. It is unrealistic to assume that all detectors are equally as predictive of an insider threat. For problems posed by an external third party operating and analyzing a deployed insider threat solution this is a major challenge that evaluators face as well and is often approximated from existing datasets albeit with different users since direct access to enterprise data is not always available. Thus, it validates the use of SUBs in such a network to create different users of differing levels of maliciousness to generate enough data to determine those probabilities for the development of meaningful intrusion detection systems.

5) University, Live Environment: For the live experiments, we started our exploration with a basic SUB that logged into a machine and performed various sequences of actions as defined by users from the West Point dataset. This did not cause any of the alerts to be triggered and the SUB performed all of its actions without trouble. This was a base case to make sure that the university network did not detect any artifacts from our simulated user and trigger an alert.

We subsequently increased the complexity of the actions of the simulated user bot and deliberately attempted to cause the IDS to cause an alert with a SSH attack, where we tried to login to a remote host a large number of times. Other similar attacks on particular ports were implemented as well and the University network was able to detect these malicious SUBs. The university network is best designed to handle network based, signature attacks as opposed to anomalous behavior on a particular machine since students and faculty normally use their own machines instead of the university's computing resources.

However, we were able to deceive the University system with low and slow attacks where a SUB performed the same number of malicious actions, but over a longer period of time. For example, in the case of the SSH attack, we spaced the same number of login attempts over a period of three days. The university network has a threshold it considers to be possibility malicious and does not retain suspected malicious users over longer periods of time. This is a vulnerability that allowed us to rerun all of our previous attacks over time without triggering any alerts from the system.

IV. Related Work

Testing intrusion detection systems has been a long standing problem and one that has been identified almost since the inception of the systems themselves. Research at Columbia in 1998 garnered attention for its use of data mining and machine learning based approach to intrusion detection [16]. Lincoln Laboratory of MIT performed comparative evaluation of various systems developed under DARPA funding in both 1998 and 1999 [17], [18]. However, the methods used by Lincoln Laboratory came under scrutiny for various reasons such as a failure to explain the validation of their test data [19]. Despite the identification of this issue regarding a lack of testing tools as early as the late 1990s and continued attention to the challenges of cybersecurity threats in 2008 [20], the 2015 report on Cybersecurity Experimentation identifies the lack of common tools for testing systems as one of the major challenges in current cybersecurity research [21]. Related work in this area can be categorized into synthetic data generation approaches, network based approaches, and integrated test bed approaches.

A. Synthetic Data Generation

At a high level, many have embarked on the task of generating realistic data sets, which is a direct by product of our work with SUBs. Data generation has been done at all levels for various purposes such as the work done by Boggs et. al. for the purposes of defense in depth measurement of web applications [22]. Generation of synthetic data was used successfully during the DARPA ADAMS project to test Insider Threat Detection models offline. The synthetic data generation system developed for ADAMS used a set of interconnected models individually trained on real world data sets to model a large organization and directly generate the data that would be collected by an Insider Threat Detection System [23]. However, while synthetic data generation systems are evidently effective at conducting offline tests, they cannot be used to test live systems *in situ* unlike our SUBs. Given the recent advent of deep neural network based techniques, researchers have started using deep learning based architectures for generating synthetic data sets. Alzantot et. al. use a dataset of accelerometer traces to illustrate that their synthetic generator, a combination of Long Short Term Memory networks and Mixture Density Network, can create believable data [24]. However, even their work acknowledges that discriminator models can distinguish between real and synthetic data with an accuracy of roughly 50 percent.

B. Network Based Approaches

Development of Network Traffic Modeling and Generation systems is a mature field with many open source tools available such as Ostinato, Iperf3, and Netperf [25]–[27]. These tools allow the user to configure a large number of parameters that may be measured by an Intrusion Detection System, such as flow volume distribution, burstiness, and packet rates. In addition, extensive work has been done developing systems that both model and generate network traffic, such as Swing [28]. More recently, Spatio-Temporal Cluster Models have been used to model and generate traffic across a cluster [29]. All of these systems have the potential to be used in testing network based IDS in action. However, none of them can also

drive simultaneous tests for host based IDS or Insider Threat Detection Systems.

C. Integrated Test Beds

Cyber test bed systems are the most similar to SUBs in that they frequently involve tests that drive both the host and network level events. The Lincoln Labs of MIT testbed LARIAT has a complete framework for specifying network configuration, driving user inputs, and modeling user behavior [30]. LARIAT is one of the core pieces of their greater work in cyber ranges, which are realistic, surrogate networks [31]. The Skaion Traffic Generation Tool is another tool used in the National Cyber Range and has the ability to simulate user actions within the test bed environment [32], [33].

Given its robust toolkit and commercial adoption, LARIAT is a powerful environment to generate and access the performance of various intrusion detection systems. However, for its realistic user traffic generation, LARIAT depends on programmatic hooks in common applications such as word processing. However, this approach reduces the realism that results from an image recognition based approach, which LARIAT uses for unfamiliar software. Braje concedes that using an image recognition based approach is more realistic. A programatic hook will tend to perform actions more quickly than a normal user would be able to do so. SUBs use image recognition to perform all user actions as a real user must first process the images on the screen before proceeding. Thus, SUBs are more realistic in their behavior compared to a user generated by LARIAT.

Another distinguishing characteristic of SUBs is the ability to measure the non-obvious insider attacks, non-obvious to IDS positioned to detect remote to local attacks. For example, SUBs can be commanded to uploaded a sensitive corporate document to a remote benign website, a clear security violation. This http post, for example, would not likely be tagged as malicious, but an insider monitoring detection system looking specifically for exfiltration of ”vital” documents might. Hence, systems like Lariat are not prepared to test and measure this class of attack. SUBs are designed to execute these kinds of insider attacks to measure whether any security architecture is prepared to detect and defend against these. SUBs also perform in situ testing of the IDS as opposed to recreating a network to test offline. The key advantage to this approach is that the users are in a real environment, which is where an attack would take place. The SUB performs its actions at a realistic pace and looks indistinguishable from a real user in nearly every facet.

V. Future Work

In future work, the immediate application of our work would be to find another suitable enterprise to test our framework to determine its true efficacy. We tested our SUB in a simulated environment with different learning algorithms as well as a University environment. However, to test its robustness, we would like to deploy SUBs in a corporate environment with a more sophisticated anomaly detection system.

In the absence of the ability to apply our work to a real enterprise, we will test our framework with other datasets such as the Are You You? (RUU) dataset [34]. This will allow us to generalize our SUB database translator to handle a wider variety of datasets. Through the use of Hidden Markov Models with adequate memory, we could derive users and their paths, building on the work applied to Clickstream [35]. Also, the ability for one SUB to interact with another live user or SUB would add to the realism. For example, an engine to perform real time answering of emails based on content would greatly enhance the realism of a SUB.

In this paper, we enumerated a sample of indicators used in intrusion detection systems from a third party operating and analyzing a deployed insider threat solution. We are actively investigating other indicators to enhance SUB behaviors to ensure those indicators are fully tested and vetted. The SUB framework also provides a testing mechanism for newly created indicators that may be proposed by security researchers.

Another area of development for the SUB framework is to improve the performance albeit it is nearly native speed. We can continue work on passing the onboard video card to the instance instead of relying on the CPU, which is a performance constraint when using QEMU. Another area of improvement lies in our ability to increase the speed of running a simulation by speeding up the system clock for non-work hours. This would reduce the realism of the system, but vastly reduce simulation time. The feature would be optional since users in a real environment would need to emulate their real user counterparts and be idle during non-work hours.

The SUB framework has potential for intrusion detection algorithm development. One algorithm or method of significant interest is to detect slow insider threats that often go unnoticed by traditional indicators and detectors such as the ones specified in our experimental section. Slow insider attacks are difficult to determine given the long time horizon over which they may take place. As such, there is significant memory overhead to keep track of the activities for each individual user. With the SUB framework, we can experiment with different algorithms or memory management techniques to see if a slow insider threat can be successfully detected consistently.

VI. Conclusion

We presented the framework to design and to create SUBs. The simulated users have the ability to nearly identically mimic the behaviors and actions of real users. All of this can be done within an enterprise network with no need to interfere with the normal operations of the system. We were able to design an experiment to show that we could inject malicious activities such that a normal user appears to act as a malicious user. In the case of the malicious users added, we were able to successfully detect them in our simulated environment as well as be detected in an enterprise environment. The ability to show that low and slow attacks are a true treat show that the health of an intrusion detection system need to be carefully monitored.

ACKNOWLEDGMENT

We would like to thank Professor Ross for supporting us with computational resources to develop and test our neural networks. Also, we would like to thank Professor Geambasu and her PhD student Mathias Lecuyer for their insights into the LSTM architecture. This work was supported as part of the IARPA SCITE Program, Scientific advances to Continuous Insider Threat Evaluation. Professor Stolfo is the founder of Allure Security Technology, Inc.

REFERENCES

[1] [Online]. Available: https://www.crunchbase.com/organization/observeit
[2] [Online]. Available: https://www.crunchbase.com/organization/niara-inc
[3] [Online]. Available: https://www.crunchbase.com/organization/alphasoc
[4] "Market guide for user and entity behavior analytics," Sep 2015. [Online]. Available: https://www.gartner.com/doc/3134524/market-guide-user-entity-behavior
[5] W. Eberle, J. Graves, and L. Holder, "Insider threat detection using a graph-based approach," *Journal of Applied Security Research*, vol. 6, no. 1, pp. 32–81, 2010. [Online]. Available: http://dx.doi.org/10.1080/19361610.2011.529413
[6] M. Kandias, A. Mylonas, N. Virvilis, M. Theoharidou, and D. Gritzalis, *An Insider Threat Prediction Model*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 26–37. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-15152-13
[7] B. M. Bowen, P. Prabhu, V. P. Kemerlis, S. Sidiroglou, A. D. Keromytis, and S. J. Stolfo, "Botswindler: Tamper resistant injection of believable decoys in vm-based hosts for crimeware detection," *Lecture Notes in Computer Science Recent Advances in Intrusion Detection*, p. 118137, 2010.
[8] F. Bellard, "Qemu, a fast and portable dynamic translator," pp. 41–41, 2005. [Online]. Available: http://dl.acm.org/citation.cfm?id=1247360.1247401
[9] [Online]. Available: https://github.com/sibson/vncdotool
[10] F. Gers, "Learning to forget: continual prediction with lstm," *IET Conference Proceedings*, pp. 850–855(5), January 1999. [Online]. Available: http://digital-library.theiet.org/content/conferences/10.1049/cp_19991218
[11] M. Bahrololum and M. Khaleghi, "Anomaly intrusion detection system using gaussian mixture model," vol. 1, pp. 1162–1167, Nov 2008.
[12] D. Reynolds, *Gaussian Mixture Models*. Boston, MA: Springer US, 2009, pp. 659–663. [Online]. Available: http://dx.doi.org/10.1007/978-0-387-73003-5_196
[13] C. Cortes and V. Vapnik, "Support-vector networks," *Machine Learning*, vol. 20, no. 3, pp. 273–297, 1995. [Online]. Available: http://dx.doi.org/10.1023/A:1022627411411
[14] B. Schölkopf, J. C. Platt, J. C. Shawe-Taylor, A. J. Smola, and R. C. Williamson, "Estimating the support of a high-dimensional distribution," *Neural Comput.*, vol. 13, no. 7, pp. 1443–1471, Jul. 2001. [Online]. Available: https://doi.org/10.1162/089976601750264965
[15] I. Ben-Gal, *Bayesian Networks*. John Wiley & Sons, Ltd, 2008. [Online]. Available: http://dx.doi.org/10.1002/9780470061572.eqr089
[16] W. Lee, S. J. Stolfo *et al.*, "Data mining approaches for intrusion detection." in *USENIX Security Symposium*. San Antonio, TX, 1998, pp. 79–93.
[17] R. Lippmann, D. Fried, I. Graf, J. Haines, K. Kendall, D. Mcclung, D. Weber, S. Webster, D. Wyschogrod, R. Cunningham, and et al., "Evaluating intrusion detection systems: the 1998 darpa off-line intrusion detection evaluation," *Proceedings DARPA Information Survivability Conference and Exposition. DISCEX'00*.
[18] R. Lippmann, J. W. Haines, D. J. Fried, J. Korba, and K. Das, "Analysis and results of the 1999 darpa off-line intrusion detection evaluation," *Lecture Notes in Computer Science Recent Advances in Intrusion Detection*, pp. 162–182, 2000.
[19] J. McHugh, "Testing intrusion detection systems: A critique of the 1998 and 1999 darpa intrusion detection system evaluations as performed by lincoln laboratory," *ACM Trans. Inf. Syst. Secur.*, vol. 3, no. 4, pp. 262–294, Nov. 2000. [Online]. Available: http://doi.acm.org/10.1145/382912.382923
[20] F. L. Greitzer, A. P. Moore, D. M. Cappelli, D. H. Andrews, L. A. Carroll, and T. D. Hull, "Combating the insider cyber threat," *IEEE Security & Privacy*, vol. 6, no. 1, 2008.
[21] D. Balenson, L. Tinnel, and T. Benzel, "Cybersecurity Experimentation of the Future," University of Southern California, Tech. Rep., 07 2015.
[22] N. Boggs, H. Zhao, S. Du, and S. J. Stolfo, "Synthetic data generation and defense in depth measurement of web applications," *Research in Attacks, Intrusions and Defenses Lecture Notes in Computer Science*, pp. 234–254, 2014.
[23] J. Glasser and B. Lindauer, "Bridging the gap: A pragmatic approach to generating insider threat data," *2013 IEEE Security and Privacy Workshops*, 2013.
[24] M. Alzantot, S. Chakraborty, and M. B. Srivastava, "Sensegen: A deep learning architecture for synthetic sensor data generation," *CoRR*, vol. abs/1701.08886, 2017. [Online]. Available: http://arxiv.org/abs/1701.08886
[25] "Ostinato network traffic generator and analyzer," http://ostinato.org/, accessed: 2017-03-17.
[26] "Iperf3," http://software.es.net/iperf/, accessed: 2017-03-17.
[27] "Netperf," http://www.netperf.org/netperf/, accessed: 2017-03-17.
[28] K. V. Vishwanath and A. Vahdat, "Swing: Realistic and responsive network traffic generation," *IEEE/ACM Transactions on Networking*, vol. 17, no. 3, pp. 712–725, June 2009.
[29] T. Li and J. Liu, "Cluster-based spatiotemporal background traffic generation for network simulation," *ACM Trans. Model. Comput. Simul.*, vol. 25, no. 1, pp. 4:1–4:25, Nov 2014. [Online]. Available: http://doi.acm.org/10.1145/2667222
[30] 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 *Proceedings, IEEE Aerospace Conference*, vol. 6, 2002, pp. 6–2671–2676, 6–2678–6–2682 vol.6.
[31] T. Braje, "Cyber ranges," *Lincoln Laboratory Journal*, vol. 22, no. 1, pp. 24–32, November 2016.
[32] "Skaion Corporation," http://www.skaion.com/, accessed: 2017-03-17.
[33] B. Ferguson, A. Tall, and D. Olsen, "National cyber range overview," in *2014 IEEE Military Communications Conference*, Oct 2014, pp. 123–128.
[34] M. B. Salem and S. J. Stolfo, *Modeling User Search Behavior for Masquerade Detection*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 181–200. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-23644-0_10
[35] A. L. Montgomery, S. Li, K. Srinivasan, and J. C. Liechty, "Modeling online browsing and path analysis using clickstream data," *Marketing Science*, vol. 23, pp. 579–595, 2004.