A Framework for Generation, Replay, and Analysis of Real-World Attack Variants

Phuong Cao, Eric C. Badger, Zbigniew T. Kalbarczyk, Ravishankar K. Iyer
Coordinated Science Laboratory
University of Illinois at Urbana-Champaign
1308 W Main St, Urbana, IL 61801
{pcao3,badger1,kalbarcz,iyer}@illinois.edu

ABSTRACT

This paper presents a framework for (1) generating variants of known attacks, (2) replaying attack variants in an isolated environment and, (3) validating detection capabilities of attack detection techniques against the variants. Our framework facilitates reproducible security experiments. We generated 648 variants of three real-world attacks (observed at the National Center for Supercomputing Applications at the University of Illinois). Our experiment showed the value of generating attack variants by quantifying the detection capabilities of three detection methods: a signature-based detection technique, an anomaly-based detection technique, and a probabilistic graphical model-based technique.

I. INTRODUCTION

In this paper, we address attacks that attempt to gain continuous control over enterprise and government networks, with a focus on reconnaissance and extraction of secret data [5]. These attacks broadly fall in the category of Advanced Persistent Threats (APT) [2]. Such persistent attacks can result in a system being compromised for a long time (e.g., 205 days, as reported in [16]) before the intruder is discovered. According to the FireEye Advanced Threat Report, 4,192 of such attacks targeted a variety of sectors such as government, financial services, energy services, and technologies in 2013 [16].

In the first stage, attackers gain initial access to machines in the target network, e.g., using stolen credentials or zero-day exploits [12]. Once a machine has been compromised, in the next stage, attackers may install remote administration toolkits (RAT) and establish covert communication channels. That facilitates persistent access to the compromised machines. During the intermediate stage, the attacker may gain different degrees of footholds in the system. In the final stage, attackers can continuously extract sensitive data, inject malicious commands, or disrupt critical production services.

To have clear visibility of an ongoing attack, host and network security monitors must be deployed at various levels of the system and network infrastructure, e.g., system logs daemon or the Bro network intrusion detection system (IDS) [1]. These monitors emit security event that indicate important activities in a target system. For example, a RAT installation by an attacker is often preceded by the transfer of a malicious file, e.g., a download of a file with a sensitive extension (.exe, .c, .sh) from a remote server using the HTTP protocol. The malicious file could be source code of a privilege-escalation exploit. An IDS such as Bro can generate an alert for the malicious file based on a subscription to a malware hash registry.

Detection of such persistent, multi-stage attacks is challenging. First, advanced attackers can create an *attack variant* that achieves the same objective of a known attack while bypassing the existing detection approaches, e.g., the attack variant can use a covert channel (e.g., Internet Relay Chat (IRC) or the Domain Name System (DNS) exfiltration technique [4]), rather than HTTP, to download code necessary for RAT installation. In that scenario, the detection mechanism based on the attack signature, which assumes the use of HTTP, would fail to detect the attack variant. Second, monitoring policies must be updated regularly, for example, to incorporate the signature of an obfuscated RAT binary file [11]. Even when the malicious file is detected, an attacker may already have misused the compromised system. Thus, one needs to investigate preceding events leading to the transfer of the malicious file.

This paper presents a framework for: (1) generating variants of known attacks, (2) replaying attack variants, and (3) validating detection capabilities of attack detection techniques against the generated variants. The contributions of this work are as follows:

- We develop a procedure for generating attack variants that aims to achieve the same objectives as the original attacks. An attack variant is represented by an event sequence (corresponding to attacker actions), in which some events in the event sequence of the original attack are substituted by equivalent events. A database of interchangeable events was manually constructed based on domain knowledge of the events present on a target system. Given a sequence of events in an attack, events in this sequence are repeatedly replaced with interchangeable events to generate new sequences, which represent attack variants.
- We develop a prototype of an attack replay framework to facilitate replay of attacks and their variants in a controlled environment, i.e., a testbed. Each attack is packaged into an attack container, which contains preinstalled vulnerable software, and host and network security monitors. An attack is replayed by executing a sequence of programs, such as exploit code or vulnerable software, in the attack containers, which results in security events or alerts being generated by network and host security monitors. When an event is observed by a security monitor, the event is routed to an attack-detection backend, where different attack detection techniques can be evaluated.

HotSoS '16, April 19-21, 2016, Pittsburgh, PA, USA
 ISBN 978-1-4503-4277-3/16/04…$15.00
DOI: http://dx.doi.org/10.1145/2898375.2898392

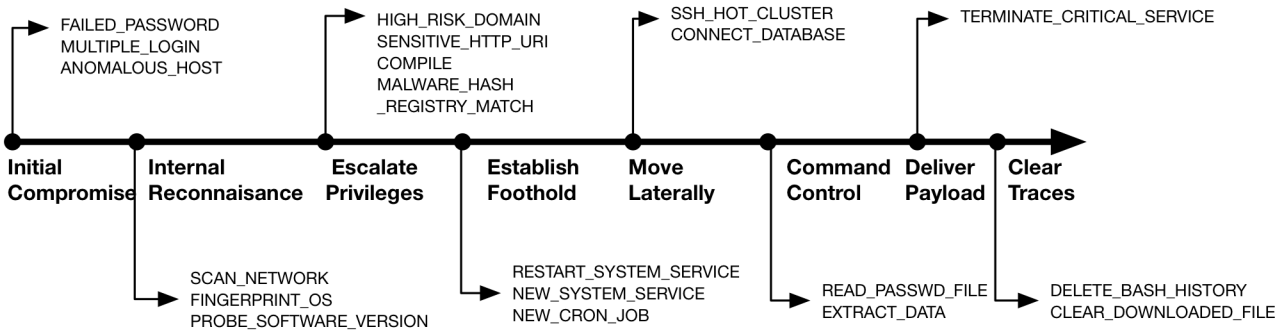

Figure 1. Timeline of a persistent attack. Major stages of the attack are: initial compromise, internal reconnaissance, escalation of privileges, establishing of a foothold, lateral movement, command and control, delivery of payload, and clearing of traces. At each stage, we list example alerts generated by the security monitoring system at NCSA during real attacks.

- We evaluate the framework on three real-world attacks for which we generated a total of 648 unique attack variants (either corresponding to other known attacks, or new (unknown) possible attacks that might happen in the future). We evaluated the detection efficiency of the following techniques (in order of increasing sophistication): (1) signature-based detection, using a file hash of known malicious files [1]; (2) anomaly-based detection, using high-frequency events observed in past attacks as an indicator of future attacks; and (3) detection based on probabilistic graphical models (e.g., factor graphs) that capture relationships between multiple events generated by security-monitoring tools to enhance detection efficiency [12]. The results show that factor graph analysis (using the AttackTagger [12]) could detect more than half of the attack variants (up to 75%), whereas the signature-based approach detected 25%, and the frequency-based approach detected up to 33%. That indicates that simple techniques such as signature-based techniques cannot detect the majority of the variants, whereas more sophisticated techniques, such as factor graph analysis, are less sensitive to attack variants. The proposed framework is being experimented at the National Center for Supercomputing Applications. The generated variants provide an additional dimension for assessing the efficacy of various detection techniques.

II. BACKGROUND

This section describes key concepts used in this paper.

An attack is a process of violating confidentiality, integrity, or availability of a targeted computer and network system. Attacks are classified into two types: transient and persistent attacks.

A transient attack is a brief attack that occurs at irregular and unpredictable times. The attack is executed by one or a series of network requests or shell commands that happen in a short period of time. For example, SQL injection attacks use a specially crafted SQL command to attach malicious SQL queries into a legitimate SQL command. Transient attacks are often carried out by script kiddies using off-the-shelf exploitation kits (e.g., Metasploit). They often cause immediate damage to a target system. More importantly, these attacks often come without any prior symptoms that indicate an incoming attack.

A persistent multi-stage attack is an attack that spans a relatively long period of time, on the order of days, weeks, or months. The main goal of attackers is to gain persistent access to the compromised system to continuously gather intelligence and/or exploit the system infrastructure, for example, to build a botnet. Because we mainly focused on persistent attacks, in this paper, an attack simply refers to a persistent multi-stage attack.

Such an attack may consist of multiple stages, including *initial compromise, internal reconnaissance, escalation of privileges, establishing of a foothold, lateral movement, command and control, delivery of payload, and clearing of traces* (Figure 1). From the defender's perspective, the attack is represented by a sequence of events. *An event* indicates an important activity in a target system, which is observed by security monitors.

A variant of a persistent or a multi-stage attack aims to achieve the same objective as the original attack. An attack variant is represented by an event sequence (corresponding to the attacker actions), in which equivalent events have been substituted for some events in the event sequence of the original attack. The attack variant can be a known or an unknown (but plausible) attack. In addition, some unimportant (for achieving the attack objectives) events (e.g., downloading of a document file) can be present in the variant to create noise that may confuse security-detection mechanisms.

Monitors are tools that collect operational data on a computer system or a network. A *security monitor* analyzes the collected data and produces *events*, which are abstractions of an important activities in the system being monitored (e.g., security alerts). Signatures of known attacks are often used for the detection of transient attacks. For example, the Bro IDS [1] looks for a malformed heartbeat request in a Secure Sockets Layer/Transport Layer Security (SSL/TLS) session to identify the Heartbleed attack. We distinguish two types of security monitors: host monitors and network monitors. Host monitors extract information from the activities observed on a given system (e.g., a workstation). For instance, the host monitors could look at what commands the user is executing, what ports are open, or what files have changed. Network monitors extract information that is being sent between two endpoints, such as the workstation and a Web server. The network traffic is analyzed, and important information is extracted to generate an alert to indicate the presence of suspicious content or network activity patterns. Examples of alerts are malformed packets, matching payloads for malware signatures, or anomalous hosts.

III. THREAT MODEL AND A MOTIVATING EXAMPLE

Threat Model. We assume that attackers use valid credentials to gain initial access to a target system. That assumption is reasonable, given that a relatively high number of leaked credentials (on the order of hundreds of millions) have recently been sold in underground markets [17]. Further, we assume that host and network security monitors in the target system are set up properly to monitor attackers' activities and generate raw logs, including system logs, network flows, and IDS alerts. That assumption is reasonable because: (1) host monitors run with highest privileges (e.g., in kernel mode), and thus can observe initial stages of an attack; and (2) network monitors are often distributed and not present on the compromised machine. Thus, it requires much effort from the attackers to tamper with the monitors. The challenge is to extract events from such heterogeneous logs and detect an ongoing attack in order to enable detection before the system is compromised. Under the threat model we are examining, we next describe an example of a real-world multi-stage attack.

A Motivating Example. In multiple security incidents reported at NCSA, attackers infiltrated a target system using stolen credentials, e.g., a private Secure Shell (SSH) key or a username-password pair; we categorized those incidents as credential-stealing attacks [14]. The target system allows multiple users to interact with the system using remote terminals via the SSH protocol. Persistent access to the target system was achieved by installing a backdoor.

In a variant of such an attack, attackers can achieve the same goal (i.e., gaining persistent access) by employing minor modifications of the attack payload. Table 1 shows events observed during a persistent attack (the second column) and events collected during a variant of this attack (the third column). Attackers can evade signature-based intrusion-detection systems, particularly, if the signature has been constructed based on the events reported by the security-monitoring system.

In our example (Table 1), the attacker logged into the system from a remote computer using the stolen credentials of a legitimate user. At the same time, the legitimate user was also accessing the system using his or her terminal, causing the "log in from multiple IP addresses" alert. The cause of the stolen credentials was revealed to be the use of a weak, guessable password by ("log in using weak password" alert). Immediately after logging in, the attacker cleared their traces by disabling the logging facility of the Bash shell. The corresponding command was "unset HISTFILE," which caused the alert "disable Bash command history logging." The attacker then attempted to install a privilege escalation exploit, obtained from an external HTTP server to get root permissions. To make the system access permanent, the attacker injected a malicious backdoor code into the SSH authentication daemon. That technique did not create any new processes in the target system, making detection and forensic analysis difficult. Furthermore, the technique ensured that the backdoor code always ran as a system daemon. In the authentication daemon, the backdoor code established a connection to an external IRC server, which served as a proxy receiving commands from the attacker, giving the attacker permanent access to the system.

In a variant of this attack, attackers can achieve the same attack goal (i.e., gaining persistent access) by employing minor modification of the attack payload to evade detection. Specifically, the attacker can substitute an equivalent action for an existing action, or execute dummy actions to introduce noise. In our example (Table 1), attackers can create an attack variant as follows. Instead of obtaining the exploit file from an HTTP server, the attacker can download the exploit file using the File Transfer Protocol (FTP), or a secret DNS tunnel [4]. From a monitoring point of view, actions generate different events. If an IDS rule is configured to detect an attack based on the event ALERT SENTITIVE HTTP URI, the rule does not work when the attacker uses the attack variant. To introduce noise, the attacker can execute legitimate operations during an attack session, such as download a document file or edit the Bash init file (indicated as *Noise* in the right column in Table 1).

It is challenging for a traditional IDSes to detect such persistent attacks. Since IDSes only look for indicators of compromise in specific events, they fail to take into account contextual information about other events. Sophisticated attackers can obfuscate observed events by using different techniques at each stage of the attack, making IDSes less effective.

Observation number	Alert sequence	Variant of the alert sequence
1	* N/A	Brute-force guess SSH password
2	Log in from multiple IP addresses	Log in from an anomalous host (e.g., a remote host)
3	Log in using weak password	* Noise
4	* N/A	Log in using an inactive account
5	Disable Bash command history logging	Set number of commands recorded by Bash history to 0
6	Download a sensitive file using HTTP	Download a sensitive file using telnet/scp/DNS
7	* N/A	* Noise
8	Compile and run the source exploit file	Compile and run the source exploit file
9	Inject backdoor to the SSH authentication service	Install backdoor as a system service
10	Establish connection with C&C server using IRC	Establish connection with C&C server using DNS

Table 1. A real-world persistent attack using stolen credentials and one of the attack variant.

IV. GENERATING ATTACK VARIANTS

This section describes a technique to generate attack variants from a sequence of events in a known attack. Each attack variant is checked for validity to ensure that the variant achieves the same attack objective (e.g., gaining persistent access to a system) as the original attack.

A. Generating attack variants

Given an attack represented by an event sequence, the goal is to generate a finite set of attack variants. The main ideas are as follows. First, use an interchangeable (or equivalent) event to replace an existing event in the sequence, while still achieving the same objective. For example, instead of obtaining the exploit file from an HTTP server, the attacker could download the exploit file using a secret DNS tunnel [4]. Such a variant make it difficult to detect the attacks, if the IDS is configured to detect only a specific event or a specific pattern of events. Furthermore, one could insert random noise between events (e.g., add events corresponding to legitimate actions during an attack session, such as downloading of a document file or editing the Bash init file). Those actions do not directly contribute to the success of an attack. In this paper, we consider only interchangeable events, and we do not consider noise in our algorithm.

In order to ensure that the variant achieves the same attack goal as the original attack, we maintain a map of interchangeable (or equivalent) events (recall that each event corresponds to an attacker action) as an input to our algorithm for generating attack variants. Each event in the original attack *maps* into one or more equivalent events.

The map of interchangeable events was created by analyzing events observed in past attacks at NCSA. First, each event was automatically classified into an attack stage by the monitoring tool that generated the event. Second, events in the same attack stage were manually grouped into an attack action, using domain knowledge of security experts. An attack action contains a set of *interchangeable* events. For example, an event "download a sensitive file from an HTTP server" can have several equivalent events (e.g., download from an IRC server or an FTP server). Actions corresponding to these events achieve the same objective of downloading a file with a sensitive extension (e.g., .c, .sh, .exe), but use different file transfer protocols.

We implemented an iterative algorithm for generating attack variants (Listing 1). Given an attack, the algorithm computes the Cartesian product of the sets of substitutable events. The number of attack variants is in the polynomial order to the length of the common subsequence. The algorithm works by repeatedly building more complex attack variants from simpler ones, starting from an attack variant of a common subsequence of length 1. By doing so, the algorithm makes sure that all possible variants are generated.

The algorithmic complexity of the attack variant-generation procedure is O(M^N), where N is the length of the attack and M is the number of equivalent events. An exponential number of attack variants can be generated, which is helpful for evaluating different attack scenarios. The procedure repeats M times for each of the N events in the attack. In our attacks, the number of equivalent events, M, is usually from 5 to 10, and the number of events, N, is usually from 10 to 20. Thus, the computation is fast and can be completed in less than a second using commodity hardware.

Algorithm: Generate attack variants from a sequence of events.

Input:

An event sequence E, for example, [3,1,5], where each number is an identifier of an event.

A list L contains the sets of equivalent events; for example,

[

3: [2,4],

1: [],

5: [6],

]

In this list, event 3 is equivalent to events 2 and 4; event 1 does not have any equivalent event; and event 5 is equivalent to event 6.

Output: Variants of the event sequence E; for example,

[

[3,1,6]

[2,1,5]

[4,1,5]

[2,1,6]

[4,1,6]

].

In the above list of variants, one of the variants is [3,1,6], where event 5 in the event sequence E is replaced by event 6.

Algorithm pseudo-code:

Attack variants are created by printing Cartesian products of the sets in list L. The procedure works by iterating over indexes of the events in list L. A pseudo-code of the procedure is shown in Listing 1.

```
generate_variant(L):
    # an array of [0..0] of size len(L)
    indexes = [0]*len(L)
    while indexes:
        print(indexes)
        indexes = next_indexes(indexes, L)

next_indexes(indexes, L):
    n = length(indexes)
    i = n - 1
    while True:
        indexes[i] = indexes[i] + 1
        if indexes[i] < length(L[i]):
            break
        indexes[i] = 0
        i = i - 1
        if i < 0:
            return None
    return indexes
```

Listing 1. Algorithm for generating attack variants.

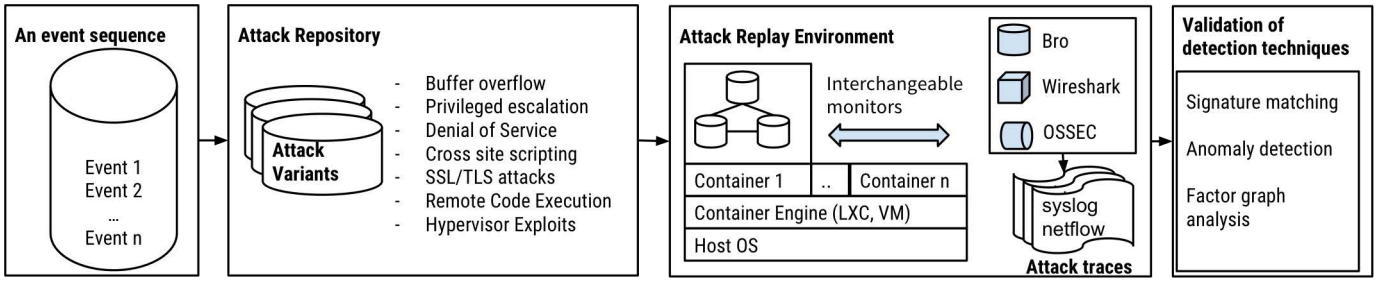

Figure 2. An architecture of the attack replay framework

B. Validity of an attack variant

An attack variant must be valid (i.e., it must achieve the same objectives as the original attack). In other words, when an attack is being launched against a target system, the system state of an attack variant must be the same as the system state of the original attack, i.e., the target system is compromised. Since an attack variant is generated by replacing an event with an interchangeable event, both events are the result of the same attack action that makes the same change to the system state.

As an example, consider the following events of an attack. An attacker logs in from an anomalous host, downloads source code (.c, .sh) or an executable exploit file (.exe), and then compiles and executes the downloaded file to gain privileged access to the target system.

[ALARM_ANOMALOUS_HOST,

ALERT_SENSITIVE_HTTP_URI,

COMPILE,

ALERT_NEW_SYSTEM_SERVICE]

One possible variant of the original attack is:

[ALARM_MULTIPLE_LOGIN,

ALERT_SENSITIVE_FTP_URI,

EDIT SOURCE FILE,

COMPILE,

ALERT_NEW_SHELL_INIT_ENTRY]

In that variant, two alternative techniques are used: (1) to obtain the source code of the exploit file, and (2) to gain persistent access to the compromised system. The exploit file is obtained using an FTP server instead of an HTTP server. The substituted event is ALERT_SENSITIVE_HTTP_URI (using an HTTP server to download the exploit file), which has the same effect as the event ALERT_SENSITIVE_FTP_URI. In the original attack, the attacker installed a backdoor as a new system service, i.e., a system-wide daemon in **/etc/init.d/**. In the attack variant, the attacker installed the backdoor as a per-user startup script in the bash init file **~/.bashrc**. That attack variant is valid because it achieves the same objectives as the original attack.

V. AN ATTACK REPLAY FRAMEWORK

This section describes a framework to replay generated attack variants in a controlled environment (Figure 3). The purposes of the framework are: (1) to verify the validity of the variants, (2) to collect attack traces, and (3) to test the detection capability of monitoring techniques.

Given an attack, the framework generates attack variants and stores them in an *attack repository*. Then, an attack container is constructed for each attack variant to facilitate the attack replay in the attack replay environment. The attack containers are managed and isolated by a container engine that makes use of virtualization techniques. During replay, host and network monitors collect *attack traces* such as system logs and network flows. Our framework can be used to validate the efficiency of security-monitoring tools and detection mechanisms/algorithms implemented using signature generation and matching, anomaly-detection algorithms, or probabilistic graphical models such as Bayesian networks or factor graphs [1][12].

Input. The input to our attack replay framework is an attack represented as a sequence of events describing the key stages of the attack. We use a comma-separated values (CSV) file to store the attack. Each line in the file indicates an event.

Generating attack variants. Using our attack variant-generation technique (Section IV.A), the framework generates a set of variants for the original attack and stores the variants in an attack repository. Currently, our repository includes attack artifacts of representative attacks from multiple categories, such as buffer overflow, privilege escalation,[1] and denial of service. The attack artifacts include exploit code, vulnerable software, monitors, and scripts to launch the attacks. A container of an attack variant is built using the artifacts and transferred to the attack replay environment for validation and testing.

Attack replay environment. In the attack replay environment, each attack variant is executed in a separate *attack container* that contains exploit code, vulnerable programs, and monitors. An attack container is a virtualized environment for running an isolated system. Virtualization is a suitable technique to quarantine execution of an exploitation code or vulnerable programs to avoid impacting a production environment.

Our attack *container engine* uses both Linux Containers (LXC) and virtual machines (VM); each targets a different type of attack. LXCs create an isolated system using a process model on a shared kernel. VMs create an isolated system by spawning a full-featured virtual machine. Compared to VMs, LXCs are lightweight and can only be used to run application-level attacks (e.g., a SQL injection attack in a Web application). On the other hand, VMs can run kernel-level attacks (e.g., an integer overflow in a device driver).

LXC is an operating-system-level virtualization environment that enables running of multiple isolated Linux systems [19]. Attack

[1] In this paper, we focus on privilege escalation attacks, because they are good examples of persistent attacks and multiple examples are present in our dataset obtained from NCSA.

Event	Action/Command	Description
EVENT READ HOST CONFIGURATION	`uname –rvim; cat /proc/cpuinfo`	Command *uname* reads the Linux kernel version of the operating system. Command *cat* reads CPU information of the target machine.
ALERT ROOT LOGIN	`ssh root@target-machine.local`	Command *ssh* connects as a *root* user to the target machine using the SSH protocol.
ALERT SENSITIVE HTTP URI	`wget http://attacker-server.local/exploit.py`	Command *wget* downloads a Python exploit script from an HTTP server to the target machine.
ALERT SENSITIVE FTP URI	`ftp –u ftp://anonymous@attacker-server.local/exploit.py`	Command *ftp* downloads a Python exploit script from an FTP server to the target machine.
ALERT CLEAR BASH HISTORY	`rm $HOME/.bash_history`	Command *rm* deletes the Bash history file, which logs commands typed in a Bash session.

Table 2. An example list of mappings from events to actions/commands; execution of the actions (second column) would cause generation (by the security-monitoring system) of the stated events (first column).

containers are run on a host system, and isolation between the host and the containers is guaranteed by Linux cgroup. LXC runs major Linux distributions such as Debian and Red Hat, thus allowing us to reproduce a wide spectrum of vulnerabilities, such as local privilege escalation or remote exploitation.

Kernel-based Virtual Machine (KVM) is a hardware-assisted virtualization (HAV) technology. Both CPU vendors (e.g., VT-x from Intel or SVM from AMD) and the Linux kernel support KVM by adding extensions to the instruction set [20], which allow a simpler and potentially more secure hypervisor.

We are using Linux to host the container engine because the Linux kernel supports LXC and KVM natively. A host Linux can be run locally on a user's laptop and contained in a VM or run remotely on a public cloud computing infrastructure (e.g., Amazon Web Services or Microsoft Azure). That provides a layer of separation between the attack container and the user's environment.

Replaying an attack. An attack is represented as a time-ordered sequence of events corresponding to attack steps (i.e., actions performed by the attacker). When an attack is being executed, each event is mapped into an execution through launching of an executable program or a script. The mapping is based on a dictionary that was created manually based on our knowledge of the system and discussion with the NCSA security team.

For example, (see Table 3) the event ALERT SENSITIVE HTTP URI, which represents a download of a sensitive file from an HTTP server, is executed by *wget*, a file-retrieving program, using the HTTP protocol, with additional parameters specifying a Uniform Resource Identifier (URI) to an exploit file.

Collecting logs. During the execution, traces of the attacks are collected by both host and network monitors for further analysis. A host monitor targets security and performance-sensitive activities on a single computer (e.g., what commands a user is executing, what ports are open, or what files have been changed). A network monitor targets network packets that are being exchanged among network endpoints (e.g., network traffic between a workstation and a Web server). Network packet analysis is performed to alert on performance and security-sensitive activities (e.g., alerts on malformed packets, matching of a signature for malicious binaries, or communication with anomalous hosts).

Our implementation uses open-source monitors and Linux built-in logging mechanisms such as OSSEC (file integrity monitoring, log monitoring, root check, and process monitoring), Snoopy (system call logging), and Bro IDS (network security monitor). For host monitors, we implemented dozens of custom rules to capture a wide range of sensitive activities on a host computer. For network monitors, we use scripts provided with the default installation of Bro, which captures the majority of sensitive network activities, such as a download of files with sensitive extensions or a connection to anomalous hosts.

Our framework supports pluggable monitors. By default, we use OSSEC for the host monitor and Bro IDS for the network monitor. A user can change the network monitor (e.g., using the Snort IDS instead of the Bro IDS) by specifying a monitoring template when initiating an attack container. A template specifies monitoring packages and configurations to be initiated. Our template uses Dockerfile, a text document that contains commands to assemble a container image [18]. Thus, advanced users can modify Dockerfile to specify their own monitoring systems.

Host and network monitors output raw logs such as system logs, OSSEC event logs, and network flows. The collected logs are transformed into a *sequence of events* for further analysis. A log entry of an event consists of following the main components: a *timestamp*, an *accountable entity, an event*, and additional *metadata*. An example log entry is shown below.

```
1453418734, ALERT SENSITIVE HTTP URI,
130.126.xxx.yyy, {rule_id=23}
```

In the above log entry, an epoch timestamp (e.g., 1453418734) is used to correlate events (ALERT SENSITIVE HTTP URI), and happen in different parts of the system and network infrastructure. An *accountable entity* specifies a user, a process, or a machine. A machine can be represented by a machine name or an IP address (130.126.xxx.yyy) that is responsible for the event. Finally, the metadata (e.g., information that the event has been triggered by a rule with rule_id 23) provide contextual information on the recorded event, such as the rule that triggered the event.

Validation of attack analysis and detection techniques. At the back end of the framework, various attack analysis, and detection techniques can be validated in terms of their ability to detect the presence of an attack replayed in our framework. Usually, a single

Attack stage	Description	Event (real NCSA alerts)	Interchangeable events
Initial compromise	An abnormal login activity	ALERT ANOMALOUS HOST	ALERT WEAK PASSWORD LOGIN ALERT ROOT LOGIN ALERT WATCHED COUNTRY LOGIN ALERT COMPROMISED PROFILE LOGIN ALERT SENSITIVE CREDENTIAL LOGIN
Escalate privilege	A download of a source code file	ALERT SENSITIVE HTTP URI	ALERT SENSITIVE FTP URI ALERT SENSITIVE SCP FILE ALERT NEW IRC DOWNLOAD
Establish foothold	An attempt to gain persistent access	ALERT NEW SYSTEM SERVICE	ALERT NEW SHELL INIT ENTRY
Establish foothold	An attempt to gain persistent access	ALERT CHANGE CREDENTIAL	ALERT NEW USER ALERT NEW SSH AUTHORIZED KEY
Internal reconnaissance	An attempt to connect to command and control server	ALERT COLLECT SYSTEM INFO	ALERT COLLECT SHELL HISTORY ALERT READ USER LIST
Deliver payload	Extraction of secret data	ALERT VIEW PASWORDS FILE	ALERT VIEW PRIVATE SSH KEY
Deliver payload	Misuse of the target system	ALERT HIGH NETWORK FLOW	ALERT HOSTING HIDDEN SPAM

Table 3. A mapping of interchangeable events (real alerts) for the attacks used in the case studies.

alert is not sufficient to declare the presence of an attack (or conclude that a system is compromised), particularly when the attack consists of multiple stages. In our framework, attack-analysis techniques/detection techniques correlate events recorded by the host and network monitors to determine the presence of an attacker.

Currently, we have implemented the following attack analysis techniques. Signature matching looks for a specific signature in terms of a file hash or a network packet checksum to identify an attack [1]. Anomaly-detection techniques look for high-frequency events observed in past attacks as an indicator of future attacks. Factor graph analysis makes decisions using contextual information about an event, in relation to events observed in the past [12].

VI. EVALUATION

This section presents case studies of attack variants generated from three real-world attacks. We illustrate the generation of attack variants and analyze the efficiency of several detection techniques against the variants.

A. Experimental setup

Our experiments were set up using two servers that were run on-site at NCSA. Server 1 contained a 24-core Intel Xeon X5650 processor running at 2.67 GHz and 32 GB of RAM running at 1,333 MHz. Server 2 contained a 12-core Intel Xeon X5650 processor running at 2.67 GHz and 24 GB of RAM running at 1,333 MHz. Both servers run the Ubuntu 14.04 LTS operating system.

B. Dataset

We used data on 116 real-world security incidents observed at NCSA during a six-year period (2008–2013) as the basis for generating attack variants. Most incidents considered in our dataset are related to multi-staged attacks, in which an attack spanned a duration of 24 to 72 hours. Incident data include written incident reports and raw logs. For each incident in our dataset, we obtained the incident report manually created by NCSA security analysts in free-format text. Each incident report contains a detailed postmortem analysis of the incident, including alerts generated by NCSA security-monitoring tools. Incident reports often include snippets of raw logs associated with malicious activities. They may also contain extra information about the incident, such as records of emails exchanged among security analysts during the incident. For a subset of security incidents, we also gathered raw logs for a period of 24 to 72 hours before and after the NCSA security team detected the incident. That duration of time is sufficiently long to cover most of the traces of attacks in our dataset. Since the data-retention policy changed during the time when incident data were being collected, the raw logs were available only for 23 of 116 incidents. We constructed event sequences for all 116 incidents using both raw logs and incident reports. Information on each attack is stored in a CSV file. Each row in the file specifies the observed time of the event, the event identifier, and the identifier for the user who was accountable for the event. We used events observed in past incidents at NCSA to provide a basis for identifying the mapping of interchangeable events.

C. Case studies: attack variants analysis

To analyze the detection efficiency of different techniques, we generated and tested attack variants based on the 116 real-world attacks reported at NCSA. We selected a relatively sophisticated persistent attack to demonstrate our attack-generation technique.

To provide input for the attack variant generation, we collaborated with the NCSA security team to build a map of interchangeable events. Table 4 shows the interchangeable events for the attacks in our case studies. For example, in the escalate privileges stage, the objective of the attacker is to place an exploit file in the target system. That objective can be achieved by downloading the file from an HTTP server or an FTP server, or using a more covert channel such as IRC, or directly copying the file using a secure copy program (*scp*). We manually constructed the map based on domain knowledge

of the NCSA system and the network infrastructure. Except for some NCSA-specific events, most of the events are common to other enterprise systems. Thus, the map can be generalized for other systems with slight modifications.

Using the attack variant-generation procedure (Section IV), we generated and validated attack variants for three attacks. Each variant generated an event sequence with the same length as the original attack. The difference between an event sequence corresponding to an attack variant and the one corresponding to the original attack can be one or multiple events, which were selected from the map of interchangeable events. Note that, if an event in an attack is not listed in Table 4 , the event is not interchangeable.

Case study 1. Credential-stealing attack

The objective of this attack was to gain persistent access to a shared supercomputing system, where multiple users interact with the system using remote terminals via the SSH protocol.

The observed event sequence in the attack is as follows.

[LOGIN,

ALERT_ANOMALOUS_HOST,

ALERT_FAILED_PASSWORD,

ALERT_CLEAR_HISTORY,

ALERT_COLLECT_SYSTEM_INFO,

ALERT_SUDO_BRUTEFORCE,

ALERT_SENSITIVE_HTTP_URI,

ALERT_NEW_SYSTEM_SERVICE,

ALERT_RESTART_SYSTEM_SERVICE]

In this attack, the attacker obtained credentials to the target system (by brute-force guessing of a password or by phishing for the password of a legitimate user). Instead of logging in from a recognized computer (e.g., the legitimate user workstation), the attacker logged in from an anomalous host (ALERT ANOMALOUS HOST**)**, which is a computer that has never been used to log into the system before. The attacker tried to get root privilege without success (ALERT SUDO BRUTEFORCE, ALERT FAILED PASSWORD), and then the attacker collected system information such as the kernel version, in order to find a suitable exploit. The attacker then downloaded an exploit file with sensitive extensions via an HTTP GET request (ALERT SENSITIVE HTTP URI) to obtain root permissions. After compromising the machine, the attacker installed a compromised version of the SSH service to the /usr/sbin directory. This service aimed to collect credentials of future user logins and to provide the attacker with persistent access to the compromised system for further misuse.

When a legitimate user interacts with the target system, an individual event in the analyzed event sequence may occasionally be observed. Observation of an individual alert is not a sufficient basis for declaring the presence of an attack. For example, ALERT SENSITIVE HTTP URI may raise a lot of false positives, e.g., when a user downloads a legitimate executable file from the Internet. However, observation of the entire event sequence of this incident suggests an ongoing attack with the objective of gaining persistent access to the target system.

An example variant of that attack is shown below (the interchangeable events are in bold).

[LOGIN,

ALERT_MULTIPLE_LOGIN,

ALERT_FAILED_PASSWORD,

ALERT_CLEAR_HISTORY,

ALERT_COLLECT_SYSTEM_INFO,

ALERT_SUDO_BRUTEFORCE,

ALERT_SENSITIVE_FTP_URI,

ALERT_NEW_SHELL_INIT_ENTRY,

ALERT_RESTART_SYSTEM_SERVICE]

By definition of the interchangeable events, each variant is a valid attack, because the attacker action corresponding to an interchangeable event achieves the same objective as the action corresponding to the event in the original attack. In this variant, the attacker logs in at the same time as a legitimate user. Since the legitimate user is a frequent user of the target system, an SSH authentication monitor raised an alert on the concurrent uses of the user account (ALERT MULTIPLE LOGIN). From a monitor point of view, the event ALERT ANOMALOUS HOST is interchangeable with the event ALERT MULTIPLE LOGIN because the monitor views the two events an abnormal login activities. Similarly, to achieve the goal of delivering a privilege exploit file to the target system, this attack variant obtains the file from an FTP server **(**ALERT SENSITIVE FTP URI**)** rather than from an HTTP server (ALERT SENSITIVE HTTP URI). In this variant, the attacker installed a backdoor as a startup entry in the Bash init file (.bash_profile) for persistent access (ALERT NEW SHELL INIT ENTRY**)**.

Case study 2. Outbound brute-force SSH attack

The objective of this attack was to misuse NCSA infrastructure to run brute-force SSH attacks against an external target system. By using this technique, the attacker can hide his or her true identity.

The observed event sequence in the attack is as follows.

[LOGIN

ALERT_FAILED_PASSWORD

ALERT_COLLECT_SYSTEM_INFO

ALERT_COLLECT_SHELL_HISTORY

ALERT_CHANGE_CREDENTIAL

ALERT_VIEW_PASSWORD_FILE

ALERT_SENSITIVE_HTTP_URI]

In this attack, the attacker obtained credentials to the target system by brute-force guessing of the password (ALERT FAILED PASSWORD), and then the attacker collected command history of the system and system information, such as kernel version, in order to find a suitable exploit (ALERT_COLLECT_SYSTEM_INFO and ALERT_COLLECT_SHELL_HISTORY). In addition, the attacker immediately changed the user's password (using the *passwd* command) to block access by the legitimate user. The attacker then attempted to view the password file at */etc/passwd* (ALERT_VIEW_PASSWORD_FILE) and downloaded a potential exploit file with sensitive extensions via an HTTP GET request (ALERT SENSITIVE HTTP URI) to obtain root permissions. In this incident, the details of outbound SSH attacks were discovered by the the NCSA security team and were not captured by the monitoring system. Thus, we did not have events for the misuse stage of this attack. Regardless, we include this attack in order to demonstrate different variants in the initial stage of the attack.

An example variant of this attack is shown below (the interchangeable events are in bold).

[LOGIN

ALERT_FAILED_PASSWORD

ALERT_READ_USER_LIST

ALERT_COLLECT_SHELL_HISTORY

ALERT_NEW_USER

ALERT_VIEW_PRIVATE_SSH_KEY

ALERT_SENSITIVE_HTTP_URI]

In this variant, the attacker obtained a list of active users in the system (using command *w* to identify logged in users and their current activities); if no administrator was active, the attacker proceeded with the attack (ALERT_READ_USER_LIST). Instead of changing the password of the compromised user account, the attacker can create a new user to gain persistent access (ALERT_NEW_USER); in future attempts, the attacker can log in using the new user account. Similarly, to achieve the goal of extracting credentials from the compromised system, the attacker can read a private SSH key, e.g., the private RSA key stored in $HOME/.ssh/id_rsa (ALERT VIEW PRIVATE SSH KEY).

Case study 3. Misuse the system for Denial of Service attack

The objective of this attack was to misuse NCSA infrastructure to build a botnet and to run Denial of Service attacks against an external server.

The observed event sequence in the attack is as follows.

[LOGIN,

ALERT_SENSITIVE_CREDENTIAL_LOGIN,

ALERT_ANOMALOUS_HOST,

ALERT_SENSITIVE_HTTP_URI,

ALERT_INVALID_MIME_EXT,

COMPILE,

ALERT_HIGH_NETWORKFLOWS]

In this attack, the attacker obtained credentials for the target system by brute-force guessing the password of a system account named *mailman*, which is used to manage emails. This account had a weak password that consisted of seven lowercase characters, allowing the attacker to gain access in just a few guesses ('ALERT_SENSITIVE_CREDENTIAL_LOGIN'). Similar to previous attacks, this login also results in the anomalous host alert. The attacker then downloaded a file with sensitive extensions via an HTTP GET request (ALERT SENSITIVE HTTP URI). In this incident, the attacker disguised the file (.c) as an image file (.jpg) which resulted in an alert ('ALERT_INVALID_MIME_EXT'). The file was compiled and used to run a botnet. The botnet sent a large number of UDP packets to launch Denial of Service attacks against an external server, which resulted in the alert 'ALERT_HIGH_NETWORKFLOWS'. An example variant of this attack is shown below (the interchangeable events are in bold).

[LOGIN,

ALERT_WEAK_PASSWORD_LOGIN,

ALERT_WATCHED_COUNTRY_LOGIN,

ALERT_SENSITIVE_HTTP_URI,

ALERT_INVALID_MIME_EXT,

COMPILE,

ALERT_HOSTING_HIDDEN_SPAM]

In this variant, the attacker logged in from a foreign country that has a high level of attack activities **('ALERT WATCHED COUNTRY LOGIN')**. The attacker used a weak password, that can be found in common password dictionaries 'ALERT_WEAK_PASSWORD_LOGIN'. In the misuse phase, the attacker used NCSA infrastructure to send spam emails instead of DoS attacks.

D. Results

We generated 144, 216, and 288 attack variants for attacks in case studies 1, 2, and 3, respectively. Each variant was replayed and used to validate the following detection techniques. Although some of the techniques are simple, the purpose of our experiment was to demonstrate the ability of our framework to assess the efficacy of different techniques.

Signature-based detection techniques only look for a specific signature in terms of a file hash or a network packet checksum in order to identify a malicious user. In our implementation, the signature approach employs a database of file hashes, which can be provided by an open-source antivirus program, such as ClamAV. We assume a signature-based IDS that only inspects downloads that use HTTP protocol. When an attacker downloads a file with a sensitive extension using the HTTP protocol (ALERT SENSITIVE HTTP URI), the file is checked for its signature. The attacker can evade that detector by using other protocols, such as SCP, FTP, or IRC.

Frequency-based detection techniques look for the most frequent event observed in the past attacks as an indicator of future attacks. In our NCSA dataset, one of the most frequent events was ALERT ANOMALOUS HOST. In the initial compromise stage of the observed attacks, attackers often logged in from a remote location or used a device different from that of the legitimate user. That action resulted in the event ALERT ANOMALOUS HOST. When such an event is found in a variant, the frequency-based technique detects an attack.

Factor graph analysis is a sophisticated technique. It not only looks for an individual event, but also analyzes the entire event sequence collectively. A factor graph-based detection tool, such as AttackTagger [12], constructs graph events linked by factor functions; each function identifies a signal of how the events relate to the suspiciousness of the user, e.g., the user account has been compromised in the past. An entire sequence of hidden-user states is inferred, based on observed user behaviors (represented as an event sequence generated by the monitoring system) and defined factor functions. This technique works with multistage attacks because it considers contextual information about an event in relation to events observed in the past.

Results (Table 5) show that factor graph analysis can detect more than half of the attack variants, whereas both the signature-based approach and frequency-based approach detect a smaller number of attack variants. We discuss the attack detection rate of the evaluated

	Attack 1	**Attack 2**	**Attack 3**
Signature-based	36/144 (25%)	54/216 (25%)	72/288 (25%)
Frequency-based	24/144 (16.7%)	0/216 (0%)	96/288 (33.3%)
Factor-graph-based	108/144 (75%)	108/216 (50%)	186/288 (64.6%)
Total number of variants	144	216	288

Table 4. Attack variant detection results. There is no technique that can detect all the variants.

techniques in the following paragraphs.

Because the signature-based technique relies on a specific signature (i.e., the event ALERT SENSITIVE HTTP URI), it does not work when an attacker delivers an exploit using a different protocol, e.g., using SCP to evade deep packet analysis of file content. The attacker can obfuscate file content to evade signature analysis. Obfuscation services, e.g., ExeCrypt, can obfuscate an executable file and test to confirm that the obfuscated file is not detected by multiple antivirus engines [22]. The signature-based technique achieved only a 25% detection rate.

The frequency-based technique achieved a low detection rate (up to 33.3%) because it relies on only a single event, ALERT ANOMALOUS HOST, for detection. In case study 2, the attacker did not raise any anomalous host alert, possibly because the attacker resided in the same network as the legitimate user. Advanced attackers can hijack an existing SSH session of a legitimate user, thus avoiding alerts on abnormal logins [21].

The factor graph analysis obtained a higher detection rate (up to 75%) compared with the signature-based and frequency-based techniques, because the factor functions were designed to be insensitive to variants. For example, a factor function treats two slightly different events (e.g., ALERT SENSITIVE FTP URI and ALERT SENSITIVE HTTP URI) in the same way.

VII. RELATED WORK

Initial work on intrusion detection focused on building monitoring infrastructure for computer and network systems [23]. At the host level, syslog (Linux) and Security Log (Windows) are used to record activities of applications and system services. For example, Snoopy is a library that logs all executed commands and arguments on a host system to syslog for auditing purposes [24]. Since more machines are being provisioned in the cloud computing environment, recent work has focused on VM monitoring (e.g., for rootkit detection, using introspection techniques [25]). Our approach complements those studies by using alerts generated by such techniques to generate attack variants and applying them to test the efficiency of monitoring and detection techniques.

As attacks have become more sophisticated, much work has been dedicated to the forensic analysis of attacks, which is valuable for understanding attack development. With the rise of advanced persistent threat (APT) attacks, top security companies regularly release reports on high-profile attacks (e.g., Stuxnet [26]). In the first stage, such attacks often make use of stolen credentials to gain initial access to a target system. Detailed analysis of this stage has been done by Sharma in [14], where such attacks are categorized as credential-stealing attacks. In addition, recent work has focused on preempting such attacks in the initial stages and analyzing obfuscated network attack vectors [12][27].

Recent research has contributed to reproducible attacks. Exploit codes of known attacks have been organized by the Metasploit project and open-source communities on GitHub [28]. At NCSA, the OpenNSM project [29] is creating Linux container images of different monitoring software. These studies, however, do not produce a unified end-to-end platform.

VIII. CONCLUSION

In this paper, we presented a framework for generating attack variants from known attacks, replaying of such attacks in a controlled testbed environment, and the validation of example detection techniques against 648 generated attack variants. Our attack replay framework is an important step toward providing a platform for attack replay in a controlled environment to facilitate validation of monitoring tools, security policies, and detection techniques in an open environment and in the presence of realistic attack scenarios. The generated variants provide an additional dimension for assessing the efficacy of various detection techniques.

ACKNOWLEDGMENTS

We acknowledge the NCSA security team for providing incident data and ground truth; undergraduate researchers Surya Bakshi and Simon Kim for contributing to the attack repository of the testbed. This work was supported by the National Security Agency under Award No. H98230-14-C-0141, in part by the National Science Foundation under Grant No. CNS 10-185303 CISE, and in part by the Air Force Research Laboratory and the Air Force Office of Scientific Research under agreement No. FA8750-11-20084. The opinions, findings, and conclusions stated herein are those of the authors and do not necessarily reflect those of the sponsors.

REFERENCES

[1] Bro IDS, https://www.bro.org/, 2016

[2] Sood AK, Enbody RJ. "Targeted cyberattacks: A superset of advanced persistent threats." *IEEE Security & Privacy* 2013 Jan; 1(1):54–61.

[3] Black PE. "Ratcliff/Obershelp pattern recognition." In *Dictionary of Algorithms and Data Structures* [online], Vreda Pieterse and Paul E. Black, eds. 2004.

[4] Hunt JW, Szymanski TG. "A fast algorithm for computing longest common subsequences." *Commun. ACM* 1977 May; 20(5), 350–353.

[5] Langner R. "Stuxnet: Dissecting a cyberwarfare weapon." *IEEE Security & Privacy* 2011 May; 9(3):49–51.

[6] Wu Z, Xu Z, Wang H. "Whispers in the hyper-space: High-speed covert channel attacks in the cloud." In *USENIX Security Symposium* 2012 Aug 8 (pp. 159–173).

[7] Owens Jr JP. *A study of passwords and methods used in brute-force SSH attacks* (Doctoral dissertation, Clarkson University), 2008.

[8] Javed M, Paxson V. "Detecting stealthy, distributed SSH brute-forcing." In *Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security* 2013 Nov 4 (pp. 85–96).

[9] Merkel D. "Docker: lightweight Linux containers for consistent development and deployment." *Linux Journal* 2014 Mar 1; 2014(239):2.

[10] Barham P, Dragovic B, Fraser K, Hand S, Harris T, Ho A, Neugebauer R, Pratt I, Warfield A. "Xen and the art of virtualization." *ACM SIGOPS Operating Systems Review* 2003 Dec 1; 37(5):164–177.

[11] Lee B, Kim Y, Kim J. "binOb+: A framework for potent and stealthy binary obfuscation." In *Proceedings of the 5th ACM Symposium on Information, Computer and Communications Security*. ACM, 2010.

[12] Cao P, Badger E, Kalbarczyk Z, Iyer R, Slagell A. "Preemptive intrusion detection: Theoretical framework and real-world measurements." In *Proceedings of the 2015 Symposium and Bootcamp on the Science of Security* 2015 Apr 21 (p. 5).

[13] Dittrich D, Dietrich S. "Discovery techniques for P2P botnets." *Stevens Institute of Technology CS Technical Report 2008* 4 (2008): 1–14.

[14] Sharma A, Kalbarczyk Z, Barlow J, Iyer R. "Analysis of security data from a large computing organization." In *Dependable Systems & Networks (DSN), 2011 IEEE/IFIP 41st International Conference on*, pp. 506–517, 2011.

[15] Hirschberg DS. "Algorithms for the longest common subsequence problem." *Journal of the ACM (JACM)* 24, no. 4 (1977): 664–675.

[16] FireEye report, http://www2.fireeye.com/rs/fireye/images/fireeye-advanced-threat-report-2013.pdf.

[17] Thomas K, McCoy D, Grier C, Kolcz A, Paxson V. "Trafficking fraudulent accounts: The role of the underground market in Twitter spam and abuse." In *USENIX Security*, pp. 195–210. 2013.

[18] Dockerfile reference, https://docs.docker.com/engine/reference/builder/

[19] LXC - Linux Containers. *linuxcontainers.org*, 2016.

[20] Kivity A, Kamay Y, Laor D, Lublin U, Liguori A. "kvm: the Linux Virtual Machine Monitor." In *Linux Symposium*, pp. 225–230, 2007.

[21] PuttyHijack, https://www.insomniasec.com.

[22] ExeCrypt obfuscation service, http://execrypt.com/en/.

[23] Anderson JP. Computer security threat monitoring and surveillance (TR), 1980.

[24] Snoopy logging library, https://github.com/a2o/snoopy.

[25] Pham C, Estrada ZJ, Cao P, Kalbarczyk Z, Iyer RK. "Building reliable and secure virtual machines using architectural invariants." *IEEE S&P* 2014; 12(5):82–85.

[26] Langner R. "Stuxnet: Dissecting a cyberwarfare weapon." *IEEE Security & Privacy* 2011; 9(3):49–51.

[27] Du H, Yang SJ. "Probabilistic inference for obfuscated network attack sequences." *2014 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pp. 57–67, 2014.

[28] Maynor, David. *Metasploit Toolkit for Penetration Testing, Exploit Development, and Vulnerability Research*. El, Syngress Publishing Inc., 2007.

[29] OpenNSM project, www.open-nsm.net