An Advanced Persistent Threat Simulation Range for Research of Self-Adaptive Systems

Ryan Wagner
Carnegie Mellon University
Pittsburgh, PA
rrwagner@cs.cmu.edu

David Garlan
Carnegie Mellon University
Pittsburgh, PA
garlan@cs.cmu.edu

Matt Fredrikson
Carnegie Mellon University
Pittsburgh, PA
mfredrik@cs.cmu.edu

ABSTRACT

While security is important to self-adaptive systems research, it is particularly difficult to simulate the characteristics of the most insidious adversary, the advanced persistent threat (APT). The research community requires an APT simulation environment that provides the realism and depth that is necessary for a sufficient simulation, but this should not be too complex, difficult, or expensive to use. Additionally, a successful simulation environment must ensure that malware does not escape into the wild. We present a cyber range for self-adaptive systems researchers to simulate APTs, complete with an example system under test and attack scenario.

KEYWORDS

APT, simulation, range, testbed

ACM Reference format:

R. Wagner, D. Garlan, M. Fredrikson. 2018. SEAMS Artifact Paper in word Format. In *Proceedings of SEAMS conference, Gothenburg, Sweden, May 2018 (SEAMS'18), 6 pages.*
DOI: 10.1145/123 4

1 INTRODUCTION

Security is important to self-adaptive systems research. It is becoming increasing important as systems must function in a global, interconnected world in which adversaries abound. A failure to account for security attacks can result in serious or devastating impacts to a business (e.g., Sony [1], Aramco, [2], Target [3]).

Security is an especially difficult problem for adaptive systems for several reasons. First, it requires consideration of adversarial behavior rather than expected behavior. Instead of thinking about concepts like mean time to failure, researchers must assume an adversary will seek to deliberately expose and exploit weak points in a system. Security also interacts in complex ways with other qualities of interest like performance, usability, deployment cost, and more. To make matters more difficult, the security terrain is constantly changing as new vulnerabilities are discovered. All of this combines to create high degrees of uncertainty regarding the security of systems. It can be difficult to know whether security events mean an attack is occurring; if so, so is carrying out the attack; what the objectives of the attack are; which system resources are affected; etc.

Self-adaptive systems researchers have looked at a variety of aspects of security in the past. For example, some researchers have explored how to respond to denial of service (DoS) attacks [4], while others have examined malware on Android devices [5].

However, an important form of the security problem exists that yet to be researched in depth—advanced persistent threats (APTs). APTs require a more holistic, high-level understanding and response than approaches that focus on mitigating a single vulnerability or class of vulnerabilities.

The US National Institute of Standards and Technology says an APT "possesses sophisticated levels of expertise and significant resources which allow it to create opportunities to achieve its objectives by using multiple attack vectors (e.g., cyber, physical, and deception)." And "The advanced persistent threat: (i) pursues its objectives repeatedly over an extended period of time; (ii) adapts to defenders' efforts to resist it; and (iii) is determined to maintain the level of interaction needed to execute its objectives." [6]

Many of the recent, high-profile and high-impact attacks like those on Sony have been the result of APTs. APTs combine into one attack scenario the aspects of timing (e.g., the sequencing of attack steps), observability (i.e., of the attack to the defender and of the defenses to the attacker), uncertainty of multiple plausible attack paths, and other factors that complicate analysis. Because of this, APTs take the security quality attribute to a greater level of complexity.

For a researcher wishing to simulate the characteristics of an APT, they need to develop a plausible scenario with multiple vulnerabilities present that can be leveraged by an attacker. The attacker must have opportunities to respond to the defender. The attacker's response may occur over an extended period of time, and the attacker's behavior may be difficult to distinguish from ordinary behavior. Due to these complex characteristics, one of the impediments to APT research in self-adaptive systems is the lack of testbeds within which new approaches can be protoypted and evaluated. Ideally such a testbed would provide (a) **realism**:

sufficient reality to represent realistic APT scenarios, (b) **flexibility**: the ability to tailor the testbed to new vulnerabilities, (c) **deployability**: the ability to deploy the testbed without worrying about compromising the infrastructure on which it is hosted or escaping into the wild.

There is a delicate balance between being faithful to a large scale system while being lightweight enough for multiple runs of a simulation in a research environment. Too much fidelity is expensive and time-consuming both to build and run repeatedly as a simulation. On the other hand, insufficient fidelity will lead to unreliable results when evaluating approaches to APTs. For example, a consideration of only one type of class of vulnerabilities or one set of system functionality may not be rich enough to understand the impact of an APT or system adaptations in a particular type of environment.

Even more important than the utility of the results, a testbed must be operated in a secure manner. Malware should not be able to escape into the wild, or it can cause harm to others and expose the researcher to liability. This means that a testbed must be constructed to either strongly contain live malware, or it should focus on simulating the effects of malware while avoiding the use of malicious software that could escape the test environment.

In this paper, we present a testbed for evaluating self-adaptive systems against. APTs. We provide a testbed environment with key functionality for research use. We also provide a small, representative enterprise network architecture for use in such an environment. The architecture has built-in vulnerabilities that we are able to exploit via injects that simulate the effects of APTs. The scenario is based on a real-world example APT, and our system can be extended or reconfigured to run other APT attacks. Our system can be downloaded and run from our Github site [7].

In Section 2, we present the primary objectives, requirements, and constraints of this system. In Section 3, we describe our key design choices. Section 4 provides an explanation of the architecture for the simulation environment and the system under test, which is evaluated in Section 5. Section 6 describes potential use cases for our exemplar. In Section 7, we discuss our future plans for the exemplar, and we conclude in Section 8.

2 GOALS

2.1 Primary Objectives

The primary objective of this example is to provide a reusable, extensible, and realistic security testbed environment in which self-adaptive systems researchers can evaluate their approaches to mitigating the threat of APTs. This environment should be able to provide an environment in which researchers can quickly stand up a demonstration system under test (SUT) and run a series of experiments on the SUT. Each test run might have a different initial state (e.g., network architecture) or different injects (e.g., different exploits used by an attacker or defenses applied by the defender). At the conclusion of each test run, logging and other state data should be captured, and the system should be automatically rebuilt to prepare it for the next test run.

2.2 Benefit to Researchers

There are several benefits in having an example environment with pre-built attack scenario. For researchers who are not security experts, a peer-reviewed example of a complex attack ensures that research into APT defenses is validated against a realistic attack scenario.

Even for researchers who are familiar with security, benefits exist. By using a consistent test environment, researchers can ensure accurate comparisons of approaches. Additionally, a ready-made environment and corpus of accepted attack scenarios saves precious research time, enabling researchers to focus on how to solve a security problem rather than creating an attack example.

Lastly, an example attack scenario aids dialog between researchers and often their industry or government funding sources, which are bound by confidentiality agreements. An agreed-upon neutral scenario provides researchers the ability to reason about and evaluate their approaches without being bound to non-disclosure agreements that could limit the impact and reuse of research.

2.2 Primary Quality Attributes

A number of criteria guided the creation of this testbed. In particular, our non-functional requirements were that the testbed must enable APT scenarios that are realistic, of sufficient depth to be useful, and abstracted enough to be lightweight for multiple test runs. All of this contributes to a quality of being useful to researchers. These are described in more depth below.

• **Realism**. To the extent possible, the example should draw on actual attacks that have occurred "in the wild." A key assumption here is that attackers and defenders have finite resources, including time. The attack complexity should be roughly commensurate with actual APT attacks, both in number of exploit types and difficulty of attack. The defender's environment should be a representative scale model of a typical network environment, with the types of components seen in real environments. The attack should have a goal (e.g., exfiltrating a specific data set) just like traditional APT attacks.

• **Depth**. The example should be complex enough to demonstrate a multi-stage attack scenario. At multiple points in the scenario, the attacker and defender should be confronted with trade-offs (e.g., security versus functionality) that complicate their individual goals.

• **Abstraction**. To guide researchers, this scenario should be simple enough to be modeled without the burden of numerous extraneous steps and details that might obscure the salient information relevant to understanding the attacker's and defender's current state and actions.

• **Utility for research**. The ultimate goal of this scenario is to provide researchers something that clearly presents interesting research challenges and can be used to validate research in defending against sophisticated APT-style attacks.

2.2 Constraints

Additionally, there are a series of constraints that any successful testbed must acknowledge. The two primary ones are cost and security. Researchers must contend with limited budgets. This means that any testbed must be cost effective to obtain, straightforward to maintain, and efficient to operate. The components should be inexpensive, and the testbed should be able to support realism and depth without breaking the budget. Security is even more tricky. It is non-negotiable to allow malware to escape into the wild. Any system must be able to guarantee that no live malware will escape from the test environment. In some cases, a public cloud environment may specifically prohibit the use of live malware in the cloud.

3 KEY DESIGN CHOICES

To describe our design, we must first divide it into two distinct views: there is an outer architecture that is the simulation environment (sometimes called a range), and there is an inner architecture that is the system under test (SUT). This separates concerns between the orchestration of the testing and the subject of the test. It promotes our functional requirement to be able to easily change the initial state or injects without needing to modify the simulation environment's own architecture. The division of these two views was our first, and most far-reaching, design decision.

Second, we decided to focus on the use of commonly-used open source software for both the range and SUT. First and foremost, this kept our costs manageable. Proprietary software can be expensive, and if one needed to buy licenses to operate tens of instances of proprietary software in a SUT, costs would grow extraordinarily quickly.

We believe our choice of open source software also promotes realism because it does allow for scaling quickly to larger, richer environments than we would otherwise be able to afford. Similarly, open source minimizes licensing concerns, enabling us to share the product of our work with other researchers. We recognize a tradeoff in some fidelity with the environments we are trying to simulate. We also may have suboptimal performance in cases in which the open source software we chose is not as fast as proprietary software. However, we determined these were reasonable tradeoffs for the cost reduction, scalability, and ability to share within the research community.

Our third design choice was to run in a public cloud environment. This further enhances scalability by allowing arbitrarily large SUTs. On the other hand, this inhibits realism by forcing us to run our range and SUT software in virtualized environments. It also does increase cost for those who already have sufficient hardware already available to them. However, we suspect many researchers will prefer a cloud environment due to the ability to rapidly scale, and also for the ability to only pay for hardware as it is used.

Our fourth design decision was intended to deal squarely with our security concerns. We were concerned about the ability for live malware to escape our environment, and we were also concerned about whether or not we could trust malware to only have the claimed functionality. For example, we did not want to install what we thought was fully-functional keylogger only to find that there was a non-documented "feature" that sent keylog data back to an untrusted third party. For this reason, we chose to simulate malware by focusing on producing similar observables (e.g., having a particular file name, creating a file with specific contents, etc.) rather than running live malware.

4 ARCHITECTURE DESCRIPTION

This exemplar consists of a system within a system. The outermost system is the simulation range. This range is a test harness responsible for providing an operating environment for the system under test, supplying injects, and logging events for the researchers. The system under test (SUT) is the inner system and simulates a network environment in which researchers can test approaches to self-adaptation.

4.1 Outer Architecture: The Range

The simulation range was constructed on Amazon Web Services (AWS) [8] and leverages well-known open source tools to accomplish its functions. These tools were chosen because they could automate much of the range at minimal cost while relying on tools and techniques that are common in today's industrial environments. This promotes the realism and allows researchers to automate environments with a greater level of depth.

These tools include:

- Packer for creating and configuring virtual machine images [9]
- Terraform for managing and orchestrating the deployment of virtual machines on AWS [10]
- Ansible for applying the injects to the SUT [11]
- The ELK stack (Elasticsearch, Logstash, and Kibana) for searching, storing, and visualizing SUT log data [12]

Together, these tools form the harness for automating the build, deployment, manipulation, and logging of the system under test. They are remotely accessed by a researcher as depicted in Figure 1.

A limitation of the current logging structure is that we do not currently have a simple way to detect log manipulation by an adversary. Ideally, we could have two logs—one would be the omniscient log with the ground truth, and the other would be an alterable log as the defender might see it. The architecture can be extended to support such logging, and this is under consideration for future work.

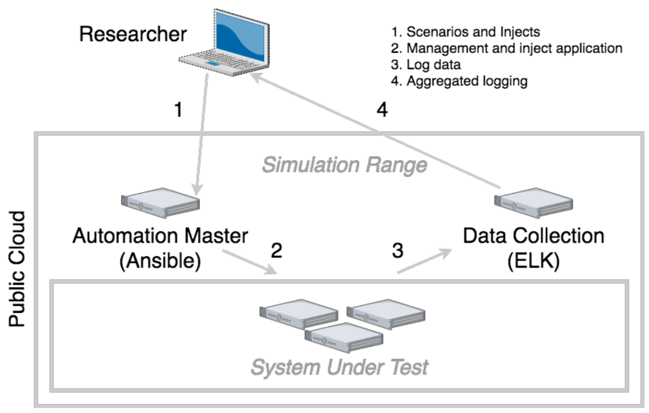

Figure 1: High level allocation view, showing relationship of researcher, public cloud, range, and SUT

4.2 Inner Architecture: The Exemplar System Under Test

The system under test is intended to have many of the same properties of an actual advanced persistent threat attack. In 2013, the retailer Target was the subject of a sophisticated attack that resulted in the loss of information, including credit card information for up to 110 million individuals [13]. The attack was caused, in part, by the theft of a third party contractor's login credentials for a Target-operated system used by Target's suppliers [3]. The attacker used these credentials to gain access to Target's internal network. From there, the attackers were able to leverage vulnerabilities and misconfigurations within Target's corporate network to ultimately compromise Point of Sale (PoS) terminals. The compromised PoS terminals collected sensitive payment information and exfiltrated it to servers operated by the attackers [14]. While the exact details of the attack are not public, this exemplar follows the outline of that attack instantiated over a small enterprise network that is sufficiently complex to be representative of a similar style of attack. The exemplar also draws from other published examples of cyber attacks [15] [16].

To model an attack, we construct a notional architecture with properties analogous to prior known attacks. With the exception of scale, this architecture should have properties in common with typical enterprise architectures.

A variety of hosts can exist within the network; some of them may contribute to only one specific function, but many hosts are necessary for the delivery of a variety of functions.. These hosts are likely to have a variety of vulnerabilities; some might be known, but many are not. These vulnerabilities fall into a variety of categories like software bugs (e.g., buffer overflows), misconfigurations, etc.

The exemplar SUT architecture includes a client machine to simulate the contractor, a web service to simulate the vendor web portal, a payment logging service, and a point of sale simulator that generates sample payment transactions. In addition to simulating the enterprise, the attacker has a host from which to launch attacks, and she has an FTP server for receiving exfiltrated data.

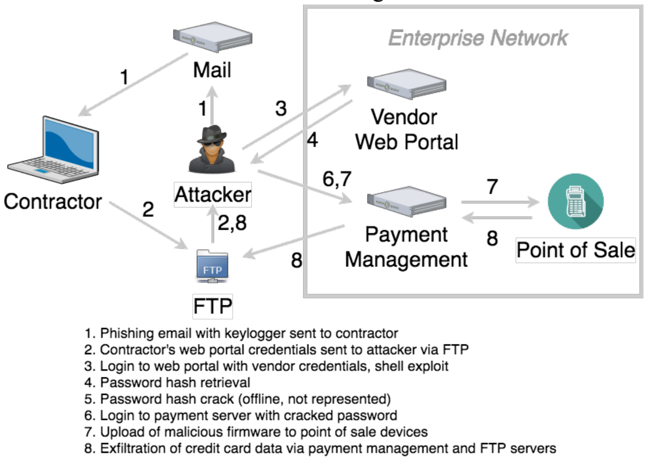
Enterprise Network
Mail
1
1
3
4
Vendor Web Portal
Contractor
Attacker
2
2,8
6,7
7
Payment Management
8
8
Point of Sale
FTP
1. Phishing email with keylogger sent to contractor
2. Contractor's web portal credentials sent to attacker via FTP
3. Login to web portal with vendor credentials, shell exploit
4. Password hash retrieval
5. Password hash crack (offline, not represented)
6. Login to payment server with cracked password
7. Upload of malicious firmware to point of sale devices
8. Exfiltration of credit card data via payment management and FTP servers

Figure 2: Overview of attack steps and major SUT components

4.3 Interaction: Injects and Observables

Of course, the architecture alone is just part of an attack; the attack must be set in motion through a sequence of events. These are injects into the SUT that modify the state of the SUT. The modifications lead to observables that can include, among other observables, the motion of data through the network or the existence of data or a process on a host.

Like the architecture described above, the attack should also have many properties in common with attacks seen in the wild. For example, the attacker should possess the attributes of an advanced persistent threat described by NIST, including "using multiple attack vectors," a willingness to operate over a long period of time, adaptability based on defenses, and a strong incentive to carry out a particular mission. Attacks consist of various stages that include reconnaissance, exploitation of a vulnerability to gain or escalate privileges, installation of software to maintain persistence, lateral movement, and continued command and control through the attack [17]. It is common for attacks to begin with phishing [18]. To make the attack coordination easier, some APTs leverage a malware toolkit like Poison Ivy to provide a turnkey malware solution [19].

The architecture at the outset of the attack is depicted in Figure 2. The attacker has an established presence on a host system outside the enterprise network. The attacker coordinates the attacks from this host and also operates a File Transfer Protocol (FTP) server elsewhere on the internet. Also, a third party contractor to the enterprise has a host that is located outside the enterprise network; the contractor logs in from this host to an enterprise web portal for the enterprise's vendors.

The web portal leverages a SQL Server for database services. The SQL contains a vulnerability that is leveraged during the attack. The web portal and database server are both within the enterprise network boundary.

Payments are processed by PoS terminals located within each store. A redacted log of payment information is sent from the PoS terminal on the store network to a payment server on the enterprise network. The payment server in this case also stores copies of firmware for download by the PoS terminals.

4.4 Observables and Abstraction

Indicators of compromise can include things like file hashes and URLs. Observations can include data like time(s) of observation(s) and numbers of observations within a timeframe. Our approach is to model malware through a focus on observables only. Take as an example a keystroke logger. Observables can include network logs like the URL, hash of the downloaded file (if not encrypted), IP address of the remote host, and filetype of the download (if not encrypted). The host observables can include process information and extracted file information (e.g., file names and types, contents, hashes).

We do not use live malware to model a keylogger. Instead, our approach uses a benign Python script in lieu of malware. This file is downloaded from a remote host in the simulation environment. When executed, the script creates a file on the host that can contain predefined strings like the word "password," but—importantly—it does not capture actual keystrokes. The script can then send the file to a remote host in the simulation environment. For our purposes, the primary difference between the functionality of the simulated keylogger and its real world equivalent is the fact that the simulated keylogger is not capturing actual keystrokes.

We require that a relevant observable in the simulation environment must be present if the corresponding observable would be seen in the live environment. Not all observables must be represented in the simulation environment.

We determine if an observable in the live environment should have a corresponding observable in the simulation environment by determining if the observable contributes to a behavior or effect of interest. A registry edit to ensure malware persistence is a relevant observable if our simulation must span reboots (e.g., if rebooting a machine is a tactic the defender may choose to evict the malware). However, if the simulation does not need to span reboots, it may not be necessary for the registry edit to recreate the behavior of starting an executable on boot.

5 EXEMPLAR EVALUATION

5.1 Realism

This scenario is composed of realistic components–both from an attack and a defense perspective. The scenario is originally constructed loosely from the details of the breach of Target in 2013 [3] [14][17]. In this breach, the exploitation began with a phishing email sent to a third-party contractor for Target. The contractor had access to service running on a host within Target's internal network. Apparently, this host utilized a privileged service with a default username; this service ran with the same credentials across the network. Due to lack of sufficient isolation of internal subnetworks, the attacker was able to move laterally through the internal network to a point from which they could install malware on the Point of Sale systems. The malware exfiltrated credit card data to the attackers.

The exemplar in this paper begins with that scenario, but it simplifies the network, which could have a large number of hosts and network devices. Further, the exemplar incorporates assumptions about how an attacker might execute such an attack using a combination of common attack methods. These methods include buffer overflow, SQL injection, and abuse of a misconfiguration. By incorporating a variety of vulnerabilities in the scenario, researchers can examine the impacts of exploits that occur with varying levels of visibility to the defender and are mitigated using different tactics.

For example, the exemplar assumes SQL injection is possible on the vendor web portal, the web application where the contractor can submit invoices to the enterprise. Additionally, the exemplar assumes a misconfiguration of the SQL server; in this example, the SQL server is a Microsoft SQL server running with administrator privileges and has PowerShell enabled, allowing for command line interaction with the server. The SQL injection will have a different level of observability and different set of defense tactics than network reconnaissance. This enables researchers to explore strategies such as focusing defensive tactics on increasing attack observability.

5.2 Depth

In our analyses to date, this exemplar has proven to have sufficient complexity. It has hosts that are both within and also not within the defender's control. There are multiple potential paths to exploitation. Some hosts provide just one overall function (e.g., vendor servicing or payment processing), while other hosts contribute to multiple functions. The number of classes of exploits is sufficient to warrant a variety of defense tactics.

On the other hand, the scenario is not too complex to cause modeling problems like state explosion. The number of hosts and vulnerabilities is relatively small when compared to a full scale enterprise network environment.

5.3 Abstraction

For our current analyses, we found the level of abstraction in this particular scenario to be appropriate. Without understanding the categories of weakness being exploited (e.g., buffer overflow, SQL injection, misconfiguration), we cannot make assumptions about exploit observability to the defender, effective defense tactics, mitigation observability to the attacker, etc.

5.4 Utility for Research

This attack scenario is useful to researchers with a variety of security objectives. As examples, this scenario can be used to model timing, observability, and graceful degradation. These are described in more detail in the next section.

6 POTENTIAL USE CASES

We envision a number of potential use cases in which the testbed can be useful. These use cases demonstrate the system's ability to replicate the steps of an APT, the observables produced in an attack, and the impact to a realistic system. These use cases include:

Timing of Attacker Eviction: For researchers interested in the impact of timing on defenses, the scenario includes exploits with a range of timing requirements from phishing (involving luck, but potentially fast) to password cracking, which could take days or longer. Similarly, defense tactics can range in timing. A password change can occur nearly instantaneously, but the instantiation of a large honeynet or rebuilding a compromised server from scratch might take longer. This range of timing for both attack and defense tactics enables researchers to explore the impact of timing on their models. As an example, we are exploring the tradeoff between attempting to evict an attacker based on the current level of knowledge held by the defender, or continuing to observe to gather more knowledge while increasing the likelihood of a successful attacker eviction.

Observability: Observability is an attribute that appears across the scenario. Some steps in the exploit might not raise the attacker's observability, while others do. For example, when the attacker uses stolen credentials to log in to the vendor web portal, this might look indistinguishable from a login by the legitimate user. Observability of defense tactics is also captured in the scenario. Changing a password would be detected by an attacker. The decision to deploy a honeynet in response to an attack is observable; however, if the honeynet were deployed prior to an attack, similar information would be gathered without tipping off the attacker that her presence was detected. In the first case, the attacker might note the addition of numerous decoy hosts on the network; in the second case, there is no detectable change in the enterprise infrastructure in response to an attack.

Graceful Degradation: Each of the enterprise's components has an associated function or group of functions, and these functions can be associated with a utility. For example, the vendor web portal's web server only supports the function of vendor invoicing. The Point of Sale devices contribute to the function of payment processing. However, a directory server could provide an identity function that is integral to both the vendor invoicing and payment processing functions. If a defender were exploring options for graceful degradation, there are a number of potential outcomes for loss of functionality. We are exploring how knowledge of an attacker's presence, combined with knowledge of the network topology, can be used to determine strategies for self-adaptive graceful degradation while under attack. The multiple overlapping functions provide a testbed for researchers who wish to explore how systems can maximize utility while managing risk in the face of an ongoing or predicted attack.

7 FUTURE PLANS

As we plan to use this testbed in our own research, we plan to maintain and expand upon the current system. This includes continued maintenance and the consideration of extensions as they prove useful to the community and our needs. A number of extensions are under consideration.

Our current logging system does not have the ability to display definitively when an attacker has modified logs. We are considering an approach that would have two logging systems. One would have the ground truth, showing all log data, with annotations to show where logs have been altered or deleted by the attacker. The second system would display the logs as a defender would see them—in their altered state. We are also evaluating recent APT attacks to understand if we should add other types of injects to our system. As attackers come up with new ways of attacking or modify old ones, our system should keep pace and reflect these changes. To make the system easier to integrate with self-adaptive software systems, we are considering the creation of an API to allow for interaction with the simulation environment and SUT. This would allow researchers' software to integrate directly, applying injects and monitoring log data. Last, we are considering the ability to simulate arbitrary times. For example, if a researcher wishes to apply an inject and then wait five hours in the SUT before applying the next inject, that should not translate into five hours of waiting in the real world. We would like to be able to essentially fast-forward the clock, marking when the injects are applied, and simulating the passage of time without researchers having to experience undue amounts of it in the real world.

8 CONCLUSIONS

In summary, our simulation range provides researchers with a framework for testing their approaches to self-adaptive defense against APTs. This framework provides realism, depth, and the appropriate level of abstraction to enable a number of research uses. Within the range, we provide an example proof of concept APT example modeled on a real attack. This demonstrates the capability of the range while providing a realistic example for consideration.

ACKNOWLEDGMENTS

This material is based upon work funded and supported by the Department of Homeland Security under Contract No. FA8702-15-D-0002 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center sponsored by the United States Department of Defense, and the National Security Agency.

REFERENCES

[1] Andrea Peterson. 2014. The Sony Pictures hack, explained. *The Washington Post*. Retrieved from https://www.washingtonpost.com/news/the-switch/wp/2014/12/18/the-sony-pictures-hack-explained/.

[2] Dehlawi, Zakariya, and Norah Abokhodair. 2013. Saudi Arabia's response to cyber conflict: A case study of the Shamoon malware incident. In *Intelligence and Security Informatics (ISI)*. IEEE, New York, NY, 73-75.

[3] Brian Krebs. 2014. Target Hackers Broke in via HVAC Company. Retrieved from https://krebsonsecurity.com/2014/02/target-hackers-broke-in-via-hvac-company/.

[4] Bradley Schmerl, Javier Cámara, Jeffrey Gennari, David Garlan, Paulo Casanova, Gabriel A. Moreno, Thomas J. Glazier, and Jeffrey M. Barnes. 2014. Architecture-based self-protection: composing and reasoning about denial-of-service mitigations. In *Proceedings of the 2014 Symposium and Bootcamp on the Science of Security*. ACM Press, New York, NY, 2.

[5] Riyadh Mahmood, Naeem Esfahani, Thabet Kacem, Nariman Mirzaei, Sam Malek, and Angelos Stavrou. 2012. A whitebox approach for automated security testing of Android applications on the cloud. In *Automation of Software Test (AST), 2012 7th International Workshop on*. IEEE, New York, NY, 22-28.

[6] Richard Kissel (Ed.). 2013. Glossary of Key Information Security Terms. NIST. DOI: https://dx.doi.org/10.60208/NIST.IR.7298r2.pdf.

[7] APT Cyber Range. 2017. Retrieved from https://github.com/cmu-apt/aptcyberrange.

[8] Amazon.com. Amazon Web Services. Retrieved from https://aws.amazon.com/.

[9] HashiCorp. Packer. Retrieved from https://www.packer.io/.

[10] HashiCorp. Terraform. Retrieved from https://www.terraform.io/.

[11] Red Hat, Inc. Ansible. Retrieved from https://www.ansible.com/.

[12] Elastic. ELK Stack: Elasticsearch, Logstash, Kibana. Retrieved from https://www.elastic.co/elk-stack.

[13] Elizabeth A. Harris, Nicole Perlroth. 2014. For Target, the breach numbers grow. *The New York Times*. Retrieved from https://www.nytimes.com/2014/01/11/business/target-breach-affected-70-million-customers.html.

[14] Brian Krebs. 2014. A first look at the Target intrusion, malware. Retrieved from https://krebsonsecurity.com/2014/01/a-first-look-at-the-target-intrusion-malware/.

[15] Munk Centre for International Studies. 2009. Tracking Ghostnet: investigating a cyber espionage network. *Information Warfare Monitor*. Retrieved from https://issuu.com/citizenlab/docs/iwm-ghostnet.

[16] Mandiant. 2013. APT1: exposing one of China's cyber espionage units. Retrieved from https://www.fireeye.com/content/dam/fireeye-www/services/pdfs/mandiant-apt1-report.pdf.

[17] Eric M. Hutchins, Michael J. Cloppert, Rohan M. Amin. 2011. Intelligence-driven computer network defense informed by analysis of adversary campaigns and intrusion kill chains. *Leading Issues in Information Warfare & Security Research* 1, 1 (2011), 80.

[18] Verizon Business. 2016. Verizon Business 2016 data breach investigations report. Retrieved from http://www.verizonenterprise.com/resources/reports/rp_DBIR_2016_Report_en_xg.pdf.

[19] Jamie Butler, Kris Kendall. 2008. Blackout: what really happened. *Black Hat USA 2007*. Retrieved from https://www.blackhat.com/presentations/bh-usa-07/Butler_and_Kendall/Presentation/bh-usa-07-butler_and_kendall.pdf.