Security Testbed: Scalable Infrastructure for Interactive Attack Replay and Testing of Security Monitoring Tools

Seoung K. Kim, Surya Bakshi, Phuong Cao, Eric C. Badger,
Zbigniew T. Kalbarczyk, Ravishankar K. Iyer
University of Illinois at Urbana-Champaign
{skim104,sbakshi3,pcao3,badger1,kalbarcz,rkiyer}@illinois.edu

1. INTRODUCTION

Security researchers reproduce and analyze attacks and vulnerabilities to understand behaviors. The process involves repetitive and time-consuming tasks like finding the vulnerable version of the software, replicating the environment or circumstance, and setting up monitoring tools for analysis. With growing number of security threats, researchers need an infrastructure that simplifies such process.

To address this need, we develop a security testbed, a controlled system and network environment where researchers safely reproduce attacks and vulnerabilities or test security monitoring tools against them [1]. It generates logs of the replayed attacks so that the researchers can study various attack stages and also serves as a database for sharing information on security threats [1].

Main requirements of the proposed security testbed are: i) *isolation* to prevent replayed attacks from affecting production infrastructure, ii) *instrumentation* to collect system and network events in various attack stages, and iii) *repeatability* to provide a convenient platform for users to reproduce attacks.

2. APPROACH

We use Linux containers (LXC) to host: i) vulnerable software or service, ii) malicious code to exploit the vulnerabilities, and iii) monitoring tools. LXC is an operating-system-level virtualization environment that enables running multiple isolated Linux systems on a single host [4]. To create and manage the containers, we used Docker container technology, which provides a layer of abstraction and automation of LXC [7].

To monitor network traffics between the containers, we use Bro IDS (Intrusion Detection System) – a network traffic analyzer that can initiate actions, such as raising alerts, on suspicious activities based on user-defined policy scripts (written in Bro scripting language) [3]. We create an attack repository, a private Docker registry where images of the Docker containers can be published, so that the attacks can be replayed again.

2.1 Architecture of Security Testbed

Our security testbed architecture consists of: attack packages, attack repository, and attack replay environment.

For each attack to be replayed, multiple containers are prepared as a package. Figure 1 (a) shows the attack package for Heartbleed as an example. Each attack package is composed of: an *Executable Exploit* container(s) that runs exploits (e.g. Heartbleed[1] exploit code of Metasploit Framework[2]), a *Vulnerable Server* container(s) that hosts vulnerable software (e.g. a webserver using a version of OpenSSL library that is affected by Heartbleed), and a monitoring container(s) that runs Bro. The *Executable Exploit* and *Vulnerable Server* containers communicate through *docker0*, a virtual bridge interface created by Docker, and the containerized Bro monitors the traffics on this interface. Attack repository, as shown in Figure 1 (b), stores the packages for different attacks. The attack replay environment depicted in Figure 1 (c) allows the attacks to be replayed with the packages pulled from the repository. The resulting logs are managed by Kafka, a distributed messaging system, and delivered to Attack Tagger that detects various attack stages and provides feedback [1].

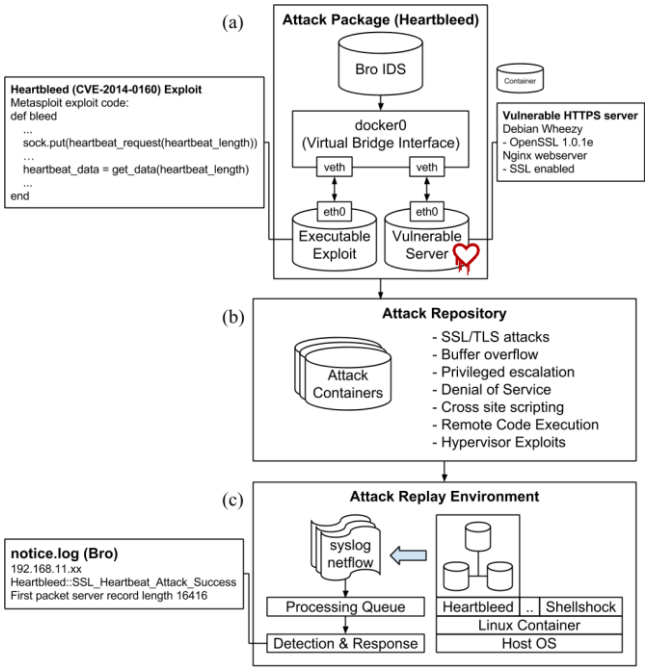

Figure 1. Architecture of the Security Testbed

2.2 Tested Attacks

In the current state, we implemented 5 attack packages corresponding to common attacks to demonstrate the versatility of our architecture:

- SSL/TLS – Heartbleed
- Arbitrary Command Execution – Shellshock
- Web Applications – cross-site scripting
- Injection – SQL injection
- Denial-of-Service – SlowDOS

[1] Heartbleed is a vulnerability in OpenSSL library that allows the attacker to read more data from the affected systems than allowed as a result of improper input validation [2]

[2] Metasploit Framework is a penetration testing tool for developing and executing exploit code [5]

3. PERFORMANCE EVALUATION

One of the main functions of the testbed is generating accurate and comprehensive logs of the attack replays. Since the performance of monitoring tools determines the quality of the logs, we evaluated the performance of Bro in two scenarios that could result in performance loss.

1) Running Bro inside containers, instead of running it directly on the bare metal host, to test performance loss due to an extra overhead of using containers.
2) Running multiple Bro instances (in a form of a cluster), instead of a single Bro instance, to test performance gain when processing a large volume of traffic, for instance, from a Denial-of-Service attack.

In both scenarios, we used a large sample packet capture (PCAP) file obtained from the Tcpreplay website [6]. It is "a [5 minute] capture of real network traffic on a busy private network's access point to the Internet" [6]. The file is 368 MB in size and contains 791,615 packets and 40,686 flows [6]. To keep the experiment environment consistent, we cleared the page, inode, and dentry caches from the system after each run.

3.1 Bare Metal vs. Container

To compare the performance of a Bro instance running on bare metal and a container Bro, we used the GNU time command to measure the time Bro takes to analyze the fore-mentioned sample capture.

In the average of 10 experiments, the Bro running on bare metal took 23 seconds whereas the containerized Bro took 27 seconds, which implies 19% performance loss. This result alone indicates noticeable performance overhead of containerized loss. However, unlike the sample capture, most of the traffics generated from replaying the attacks in the testbed last less than a minute, consist of much smaller volume of packets and flows (less than 1,000). Considering such difference in conditions, we expect no substantial performance loss from running containerized Bro in the testbed.

3.2 Single vs. Multiple Bro Instances

To simulate a large volume of real-time traffic, we ran Tcpreplay, a tool for replaying previously captured network traffic, inside a container to replay the sample capture onto *docker0* (we chose the virtual network interface because the performance of a physical network interface was inconsistent) [6]. We replayed the traffic at 15 different data transfer speeds, ranging from 100 to 1,500 Megabits per second (Mbps), for 15 times each to observe how the speed affects Bro's performance. We used the number of received and dropped packets, which were extracted from the logs that Bro generated, to evaluate the performance. We repeated the process for running 1, 2, and 3 instances of Bro. We ran all Bro instances on bare metal host with an 8 core processor, and each instance was configured to run on a separate core.

Figure 2 is a plot of the percentage of packets dropped at each speed for different number of Bro instances. It shows that as the speed increases, the percentage of packets dropped increases. For example, a single Bro instance monitoring at 100 Mbps traffic does not drop packets, while 1,000 Mbps traffic results in more than 50% packets dropped. Furthermore, at around 400 Mbps, a single Bro instance dropped nearly 10% of packets while 2 and 3 Bro instances do not drop packets. Based on these results, we conclude: i) when the data transfer at faster speed, Bro becomes more likely to fail to capture new packets as it may not have finished processing the previous ones in time and (ii) with higher number of instances, Bro becomes less likely to drop packets because the volume of traffic that each Bro instance needs to process decreases. In other words, Bro is likely to perform better and produce more accurate logs when running in multiple instances. We consider implementing such Bro configuration when the volume of traffic is too large to be handled by a single Bro instance.

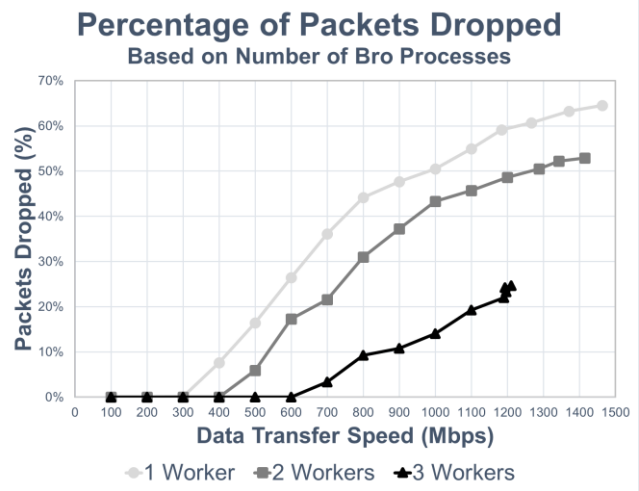

Figure 2. Performance Evaluation: Single vs. Multiple Bro Instances

4. CONCLUSION AND FUTURE WORK

We presented the initial architecture and implementation of a scalable infrastructure for interactive attack replay and testing of security monitoring tools.

To improve the overall performance of the testbed, we plan to conduct more thorough evaluations and revise the architecture if necessary, for instance, by implementing multiple Bro instances. We also plan to create attack packages for other types of attacks, such as hypervisor and kernel-level exploits, and create an interactive interface for the testbed.

5. ACKNOWLEDGMENTS

This material is based upon work supported by the Maryland Procurement Office under Contract No. H98230-14-C-0141.

6. REFERENCES

[1] Cao, P., Badger, E., Kalbarczyk, Z., Iyer, R., Withers, A., Slagell, A. Towards an unified security testbed and security analytics framework. In *Proceedings of the 2015 Symposium and Bootcamp on the Science of Security* (HotSoS '15). ACM, New York, NY, USA, Article 24 , 2 pages. DOI= http://doi.acm.org/10.1145/2746194.2746218

[2] Heartbleed Bug. http://heartbleed.com. Accessed: 2015- 07- 20.

[3] Introduction – Bro 2.4 documentation. https://www.bro.org/sphinx/intro/index.html. Accessed: 2015- 07- 20.

[4] Merkel, D. Docker: lightweight linux containers for consistent development and deployment. Linux Journal 2014, 239 (2014), 2.

[5] Penetration Testing For Risk Validation with Metasploit. http://www.rapid7.com/products/metasploit/. Accessed: 2015- 07- 20.

[6] Tcpreplay - Pcap editing and replaying utilities. http://tcpreplay.appneta.com/. Accessed: 2015- 07- 13.

[7] What is Docker?. https://www.docker.com/whatisdocker. Accessed: 2015- 07- 20.