JUNO: Automated Creation of Vulnerable Practice Networks

Casey Colley
School of EECS
Oregon State University
Corvallis, OR, USA
colleyc@oregonstate.edu

Arian Ghorbani
School of EECS
Oregon State University
Corvallis, OR, USA
ghorbana@oregonstate.edu

Carter MacNab
School of EECS
Oregon State University
Corvallis, OR, USA
macnabc@oregonstate.edu

Alex Marx
School of EECS
Oregon State University
Corvallis, OR, USA
marxal@oregonstate.edu

Yeongjin Jang
School of EECS
Oregon State University
Corvallis, OR, USA
jangye@oregonstate.edu

***Abstract*—Cybersecurity games have proven to be successful as a practical learning tool for students and professionals, as shown in many Capture-The-Flag (CTF) and Cyber-Defense competitions (CDC). However, the infrastructure required for Cyber-Defense competitions is more intricate than other cybersecurity games, such as Jeopardy-style Capture-The-Flag competitions, as it necessitates a network of computers with interconnected services and various vulnerabilities. To address this challenge, we develop JUNO, a work-in-progress framework for building infrastructure for Cyber-Defense competitions. JUNO is designed to automate the creation of virtual networks and rapidly deploy them for defensive cybersecurity games. The system comprises a core program that can generate vulnerable machine images and additional microservices to host these in virtual machines automatically.**

***Index Terms*—cyberdefense competitions, practice networks, blue team, red team**

I. INTRODUCTION

Jeopardy-style Capture-The-Flag competitions (CTFs) [1] are the most prevalent cybersecurity games due to their effectiveness in teaching specific exploitation techniques, and their compatibility with large-scale deployment through technologies such as Kubernetes [2]. The challenges in CTFs are simple, ephemeral, and independent of one another.

In contrast, Cyber-Defense competitions (CDCs) [3] are less frequently hosted and more intricate. Rather than singular challenges, CDCs are comprised of a network of several computers that host common business services such as websites, mail servers, and Domain Controllers, which are often interdependent. CDCs teach students system administration, log analysis, memory forensics, and incident response rather than exploitation. Consequently, CDCs have the potential to encompass a significantly wider range of topics within cybersecurity education.

Teams that wish to practice attacking or defending such a network must first design it themselves and verify the intended vulnerabilities are exploitable, which is time and effort-intensive. As a result, the team becomes intimately familiar with the network, and the process of exploring an unknown network for indicators of compromise (IOC) to remediate and vulnerabilities to patch becomes less effective at teaching these skills. Defensive teams require a separate, dedicated *misconfiguration* team to design these practice networks, but that is not always possible for smaller collegiate cybersecurity clubs or classes.

JUNO was designed to be the misconfiguration team for these defensive teams. At the team's request, JUNO will generate a network of machine configurations with a semi-random selection of security liabilities that is automatically deployed on a hypervisor. Teams can then access this virtualized network to begin practicing. By automating the set-up requirement, the goal of JUNO is that teams can reduce the amount of time that it takes to create a practice network, preserve unfamiliarity with the network, and focus their efforts instead on strengthening their cyberdefense skills.

Here, we design JUNO to function as the automatic *misconfiguration team* for defensive teams. Upon the team's request, JUNO will produce a network of machine configurations with semi-randomly selected security vulnerabilities and misconfigurations that will be automatically deployed on a hypervisor. Teams can then access this virtualized network to commence practice sessions. JUNO's purpose is to automate the set-up requirement, thereby reducing the time required to create a practice network. The system is designed to help teams maintain unfamiliarity with the network, allowing them to concentrate on strengthening their cyber defense skills.

II. ARCHITECTURE

JUNO comprises three main design components, which are as follows: 1) the Environment Architect, 2) the Hypervisor, and 3) the Identity Management Server.

Figure 1 illustrates how components are connected each other in a nutshell.

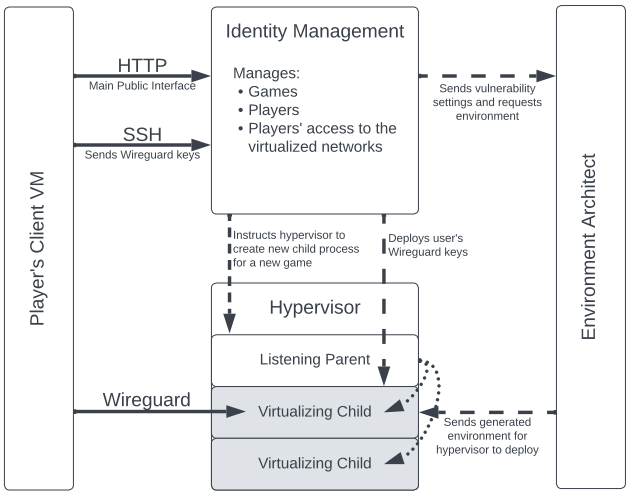

Fig. 1: An architecture diagram of JUNO.

A. Environment Architect

The Environment Architect is responsible for generating network configurations. Its primary function is to receive game requests, along with specific vulnerability settings, from the Identity Management server. It selects machine images and generates misconfiguration scripts for a network consisting of 6-7 computers per game and sends them to the Hypervisor for deployment (number of computers can be adjusted in scale based on the preferred game size, say, up to 100+ computers).

To generate these misconfiguration scripts, the Environment Architect utilizes a collection of modular `cloud-init` settings and maintains a database that tracks the dependencies of each script. Additionally, the Environment Architect can be configured to include a hypervisor and create the final virtual hard disk, instead of offloading the task to the Hypervisor.

B. Hypervisor

The Hypervisor is responsible for securely deploying the computer images that the Environment Architect provides in an isolated environment. Upon receiving a network to deploy, a parent process listens for the request and creates a child process. The child process provisions virtual machines (VMs) as needed, installs the machine image, and runs the misconfiguration script via `cloud-init`. An additional VM is established with a VPN server to manage access to the network. The Hypervisor is KVM-based [4] and currently fulfilled by QEMU [5]. Since JUNO was designed with collegiate cybersecurity clubs in mind, it is expected to be deployed on university servers. Penetration testers may also wish to utilize a worm, active malware, or another similarly destructive technique. As a result, it is essential that malicious activity taking place inside the virtualized network are unable to escape the hypervisor or the network, inspired by academic work GQ [6]. As the Hypervisor's emulation requirements are minimal, the attack surface for hypervisor escape can be reduced by avoiding the use of QEMU, which is a sizable project that has a huge trusted computing base (TCB). In the future, QEMU will be replaced by a different hypervisor.

C. Identity Management

The Identity Management Server, which serves as the primary interface of the system, consists of a website and a database. The server allows users to request games and manage the keys utilized to access game networks. When a user requests a game, the Identity Management Server sends a request to the Environment Architect, which includes the specified settings.

The website component of the Identity Management Server enables users to access the system's primary features, including game requests and key management. The database component of the Identity Management server stores user data and configuration settings, allowing for quick and easy retrieval when needed.

Upon receiving a game request, the Identity Management server sends a request to the Environment Architect, which contains the necessary specifications for game setup. By facilitating communication between the user and the Environment Architect, the Identity Management server streamlines the game request process, enabling users to rapidly obtain customized game environments that suit their needs.

III. LIMITATIONS AND IMPROVEMENTS

By automatically generating misconfigurations, the resulting practice environment lacks the coherence and personalization of a human-designed network, which teams would typically encounter during competition. Nonetheless, the authors consider this an acceptable limitation (except for one scenario) because the lack of personalization does not diminish teams' ability to utilize the networks for defensive practice.

The one exception that will be enhanced in future iterations is the random generation of websites as services in practice networks. Web security is essential in Cyber-Defense practice networks, and websites created for the purposes of CDC networks require their own bespoke design. During competition, student teams are expected to examine their websites and patch vulnerabilities in the source code while safeguarding the rest of the network. The authors anticipate that these changes would necessitate additional redesign of how the Environment Architect generates networks, but it is still a necessary step to undertake.

These bespoke services that teams are expected to inspect and patch mid-competition also resembles the type of services that are featured in Attack & Defense CTFs, such as `DEF CON CTF Finals` [7]. When the Environment Architect is able to generate bespoke websites, the authors expect it would be able to generate various other types of services and can be configured to help teams practice for Attack & Defense CTFs.

Finally, when the project began, the authors were unfamiliar with Rust [8] and opted to develop the system's components in C. Going forward, the components should be rewritten in Rust to reduce the likelihood of memory corruption vulnerabilities.

REFERENCES

[1] C. Eagle and J. L. Clark, "Capture-the-flag: Learning computer security under fire," Naval Postgraduate School Monterey CA, Tech. Rep., 2004.

[2] T. Kubernetes, "Kubernetes," *Kubernetes. Retrieved May*, vol. 24, p. 2019, 2019.

[3] A. Conklin, "Cyber defense competitions and information security education: An active learning solution for a capstone course," in *Proceedings of the 39th Annual Hawaii International Conference on System Sciences (HICSS'06)*, vol. 9. IEEE, 2006, pp. 220b–220b.

[4] A. Kivity, Y. Kamay, D. Laor, U. Lublin, and A. Liguori, "kvm: the linux virtual machine monitor," in *Proceedings of the Linux symposium*, vol. 1, no. 8. Ottawa, Ontorio, Canada, 2007, pp. 225–230.

[5] F. Bellard, "Qemu, a fast and portable dynamic translator." in *USENIX annual technical conference, FREENIX Track*, vol. 41. California, USA, 2005, p. 46.

[6] C. Kreibich, N. Weaver, C. Kanich, W. Cui, and V. Paxson, "Gq: Practical containment for measuring modern malware systems," in *Proceedings of the 2011 ACM SIGCOMM conference on Internet measurement conference*, 2011, pp. 397–412.

[7] C. Cowan, S. Arnold, S. Beattie, C. Wright, and J. Viega, "Defcon capture the flag: Defending vulnerable code from intense attack," in *Proceedings DARPA Information Survivability Conference and Exposition*, vol. 1. IEEE, 2003, pp. 120–129.

[8] N. D. Matsakis and F. S. Klock, "The rust language," *ACM SIGAda Ada Letters*, vol. 34, no. 3, pp. 103–104, 2014.