Environment Provisioning and Management for Cybersecurity Education

John Ford, David Arnold, and Jafar Saniie

Embedded Computing and Signal Processing (ECASP) Research Laboratory (http://ecasp.ece.iit.edu)
Department of Electrical and Computer Engineering
Illinois Institute of Technology, Chicago IL, U.S.A.

***Abstract* – Hands-on learning environments and cyber ranges are popular tools in cybersecurity education. These resources provide students with practical assessments to strengthen their abilities and can assist in transferring material from the classroom to real-world scenarios. Additionally, virtualization environments, such as Proxmox, provide scalability and network flexibility that can be adapted to newly discovered threats. However, due to the increasing demand for cybersecurity skills and experience, learning environments must support an even greater number of students each term. Manual provisioning and management of environments for large student populations can consume valuable time for the instructor. To address this challenge, we developed an Environment Provisioning and Management Tool for cybersecurity education. Our solution interacts with the exposed Proxmox API to automate the process of user creation, server provisioning, and server destruction for a large set of users. Remote access will be managed by a pfSense firewall. Based on our testing, a six-machine user environment could be provisioned in 14.96 seconds and destroyed in 15.06 seconds.**

Keywords – Cybersecurity, Education, Automation, Testbed, Cyber Range, Virtualization, REST

I. Introduction

Cyber threats and threat actors continue to evolve and adapt to the changing technology landscape, taxing defense infrastructure and increasing demand for personnel. According to the International Information System Security Certification Consortium (ISC)^2, there is a global gap of 3.4 million cybersecurity jobs and a gap of over 400,000 in the United States alone [1]. This shortage shows no sign of shrinking as the U.S. Bureau of Labor Statistics predicts that cybersecurity roles, such as cybersecurity analyst, are growing 35% over the next decade [2]. Similarly, the high demand for cybersecurity experience is driving enrollment in cybersecurity degree programs, requiring an increase in resources for educational programs. Among these resources are cyber ranges, testbeds, and practice environments, which provide essential hands-on experience. Higher enrollment increases the amount of time required to provision and manage network resources. In order to address these challenges, we explored automation solutions for provisioning and managing environment resources.

Since the late 90s and early 2000s, research has been conducted for cybersecurity and Information Technology (IT) testbeds in the commercial, academic, and military domains [3]. Generally, these domains required testbeds for collecting data on new technologies prior to wider implementation and for training and educational applications. For instance, the Lincoln Adaptable Real-time Information Assurance Testbed (LARIAT) was presented in 2002 and was based on DARPA testbeds in order to evaluate intrusion detection (ID) models [4]. Similarly, the Real-Time Immersive Network Simulation Environment for Network Security Exercises (RINSE) was presented in 2006 to support large-scale security exercises for denial-of-service (DoS) attacks [5]. More recently, testbeds have expanded into domains such as the Internet of Things (IoT), Cyber-Physical Systems, and Industrial Control Systems (ICS) [6-9]. IoT is a relatively new field, with explosive growth within the last decade. On the other hand, digitization efforts within ICS require retraining of engineers and IT staff.

While cybersecurity testbeds can be applied to research and education, our primary motivation is to ease the administrative burden for teaching applications. In order to ensure the testbed remains up to date with current trends, cybersecurity testbeds for education fall into a seven-phase cycle which is presented in Table I [10]. The iterative design cycle focuses on constant adjustments to the underlying hardware and virtualization software along with refreshing cybersecurity challenges in order to keep them up to date with emerging threats. During a typical semester, manual deployment and maintenance of challenges can become a burden on educators and administrators, consuming valuable time and resources at a critical time.

TABLE I
DESIGN CYCLE FOR CYBERSECURITY TESTBEDS

1	Define Environment
2	Deploy Environment
3	Define Challenges
4	Deploy Challenges
5	Conduct Challenges
6	Maintain Environment
7	Maintain Challenges

Our contributions to the field are centered around the development of an automation tool for interacting with Proxmox VE and a pfSense firewall for ease of management and maintenance. The primary objective was to create a system for dynamically provisioning cybersecurity environments in a manner that is scalable, reliable, and freely accessible. Instructors and administrators will be able to generate a template, clone a virtual machine, and purge virtual machines. Management will be completed via the command line with

configurable parameters related to the subnet, static IP, clone name, and username. Cloning operations should occur in a minimal amount of time, ideally within one minute for a six-machine environment. Students may access their machines via a web user interface or an OpenVPN server. We were successfully able to meet these requirements, making use of Proxmox VE as our hypervisor solution, the Python programming language, and Python's proxmoxer package for interacting with the Proxmox instance.

Through the remainder of this paper, we will explore related work in cyber range and testbed research. In Section III, we will present the underlying environment architecture we will be using to evaluate our automation tool. Next, we will discuss the proxmoxer libraries used by our tool. After discussing our software, we will present our tool in action. Finally, we will wrap up our analysis and discuss future work.

II. Related Works

Extensive research continues to be conducted regarding the design and evaluation of cyber range, testbed, and environment architectures. This research falls within six general categories: Scenario, Monitoring, Learning, Environment, Teaming, and Management [11]. The first category, Scenario, investigates mechanisms for generating and developing new challenges and environments for student training. For instance, the Alpaca project implemented an AI engine for generating vulnerability lattices, which were composed of sequences of vulnerabilities and exploits for the user to tackle [12]. Next, Monitoring explores data collection process for evaluating and policing user activity within the environment. Learning refers to tools that assess student learning as they navigate through the environment challenges. Assessment can be completed by score bots that monitor the status of environment services or post-exploit questionnaires. Our next category is Environment, which deals with the hardware and software that hosts the network. For our application, we opted to use the Proxmox VE operating systems and virtualization platform to host our virtual machines and the pfSense open-source firewall and virtual router for our environment. Teaming refers to research based on the parties participating in cyber range and security testbeds. Finally, Management research involves exploration of the roles, interfaces, command and control, and resource management for the testbed. Of the mentioned categories, Management is the closest fit for our research and we will primarily compare our work to others within this category.

Automation of user generation and challenge deployment have historically been completed within YAML or novel test case description languages. For example, the Cyber Range Instantiation System (CyRIS) utilizes YAML files for describing the composition and content within the desired cyber range [13]. Similarly, the work completed by Frank et al. also used YAML files in order to describe the testbed configuration [10]. Alternatively, the Virtual Cyber Security Testing Capability (VCSTC) utilized a novel description language to specify the desired environment and challenge machines [14]. While these configuration files are preferrable to manual user generation and deployment, the administrator must generate fresh entries for each user. Our command line tool simplifies the process and provides flexibility to the administrator when generating new accounts.

III. Environment Architecture

As part of our investigation, we developed an educational environment that served as the target application for our automation tool. The core of the environment was the Proxmox Virtual Environment, which hosted the virtual machines used in cybersecurity challenges and would be the primary point of interaction with our tool. We selected this virtualization software over competing platforms, such as ESXI and KYPO, as our members were familiar with the administrative and management features of the platform. Proxmox VE also exposes REST API that can be used for most management functions, which will be used as the foundation of our automation tool. In addition to Proxmox VE, a pfSense VM was utilized for internal DHCP management and remote access. Similarly, this was selected as the team was familiar with the platform. Additionally, six template VMs were generated for cybersecurity challenges. The environment architecture used throughout this project is presented in Figure 1.

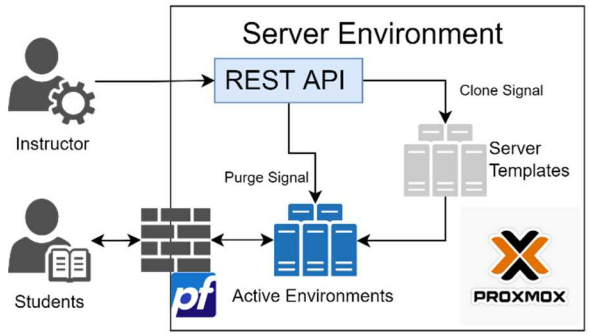


Fig. 1. Environment architecture. Upon request from the Instructor, our automation tool interacts with the Proxmox REST API to provision student machines from templates. Additionally, environments may be purged at the request of the Instructor as well.

Proxmox VE is an open-source virtualization platform that allows for creating and managing virtual machines (VMs) and containers, providing a web-based graphical user interface (GUI) [15]. Proxmox VE is developed by Proxmox Server Solutions GmbH, a company based in Austria that specializes in open-source virtualization solutions. The company was founded in 2005 and has since then been actively developing and improving the Proxmox VE platform. Proxmox VE is distributed under the open-source license GPLv2, which means that it is free to use, modify, and distribute. It is designed to be a complete virtualization solution for both small businesses and large enterprises. Proxmox offers the full functionality of other bare-metal hypervisors such as VMware's ESXi. It allows for managing virtual machines along with emulating various types of disks and IO devices and connecting them through internal or to external networks. It supports running both Linux and Windows VMs on the same infrastructure and provides solutions for maintaining these VMs with clustering, live migration, and

high availability. Proxmox also offers some advantages to its competitors, providing full command line access to its underlying Debian-based Operating system (OS) and an Application Programming Interface (API) that can be leveraged by developers to create tools for managing Proxmox remotely.

PfSense is an open-source firewall and routing platform based on FreeBSD [16]. It was developed by Netgate, a company that specializes in network security solutions. pfSense was first released in 2006 as a fork of the m0n0wall project and has since then been actively developed by Netgate and its community of contributors. pfSense is distributed under the open-source license BSD, which means that it is free to use, modify and distribute. It is designed to provide comprehensive network security and routing functionality for small to medium-sized businesses and home networks. pfSense provides a web-based graphical user interface (GUI) that makes it easy to manage firewall rules, routing, and network services. It supports a wide range of security features, including stateful packet inspection, VPN connectivity, intrusion detection and prevention, and content filtering. One of the key features of pfSense is its flexibility and extensibility. It supports third-party packages and plugins that can be used to add additional functionality, such as DNS and DHCP servers, web proxies, and load balancers. pfSense is also highly customizable and can be configured to meet a wide range of network security and routing needs. It supports advanced routing protocols such as OSPF and BGP and can be used to build complex network topologies.

Along with creating fully isolated networked environments, it's important that these environments have access to the internet but in a contained manner that would not pose a security risk. This requires routing network traffic through a firewall inaccessible to the student that will allow all outgoing connections but block any incoming connections except those that are explicitly allowed for the environment at hand. In light of our decision to keep this project free and open source, we elected to use the pfSense firewall for this purpose. We were successfully able to configure pfSense by remotely connecting to it through SSH, modifying the XML configuration file, and reloading changes through the php command.

When interacting with our environment, the Instructor is provided with three options: *template*, *clone*, and *purge*. Our *template* script allows the Instructor to generate a Proxmox VE virtual machine template based on an existing VM. Within the environment, templates are saved states of virtual machines that can be used to create copied virtual machines. Instructors may use this after creating student challenges that they intend to clone later. *Clone* completes the cloning process and provisions a set of virtual machines based on the Instructor's specifications. Arguments to this function include the username, the machines to clone, the IP addresses of the new machines, and DNS servers for the subnet. Finally, *purge* allows the Instructor to delete virtual machines from the environment, ideal for cleaning up at the end of the semester.

Development of our tool were completed in Python. Provisioning and account management were accomplished via the Proxmox API, which is based on a RESTful architectural style. Discussion regarding these technologies is discussed in the next section.

IV. Development Tools

The Proxmox API is an interface that allows users to programmatically interact with the Proxmox VE virtualization platform for automating tasks, integrating with other systems, or building custom applications [17]. The API facilitates all provisioning of VMs and provides access to performance and monitoring data, allowing for retrieval of information about resource usage, health, and status of the virtualization infrastructure. The web-based GUI itself is built on this API and makes API calls to the back end to perform all provisioning operations. The Proxmox API is based on the RESTful architectural style and uses standard HTTP and HTTPS requests and responses. It supports both JSON and XML data formats for data exchange and includes comprehensive documentation to help developers get started, including an online comprehensive API viewer that details results each request will produce and which parameters are accepted.

The robustness of the API has fostered the growth of many frameworks in various languages for managing Proxmox both locally and remotely. Several client libraries are available for different programming languages, such as Python, Ruby, and Perl, that serve as wrappers for interactions with this API. In Python, this library is called proxmoxer, which supports performing API calls using the JSON data format [18]. It is built using Python's standard requests library for making HTTP requests but allows for writing requests in Python-style dotted notation rather than in forward-slash HTTP notation. It is also available as a package, installable through Python's pip package manager, for easy importing. Many systems administration and automation tools additionally use these client libraries to integrate with Proxmox. This includes Ansible for remote host management, which uses the Python client library for managing Proxmox, and Terraform for virtual machine provisioning, which uses the Go client library for managing Proxmox.

Ansible is an open-source IT automation and configuration management tool. It is designed to simplify the management of complex IT environments by automating routine tasks, such as configuration management, application deployment, and system orchestration. Ansible uses a declarative language, called YAML, to define system configurations and tasks. YAML is a human-readable language similar to markup that makes it easy for system administrators and developers to define tasks and parameters for those tasks without having to specify implementation details. Ansible operates over SSH for Linux and WinRM for Windows, making it easy to manage systems regardless of their operating system or location. It also has a large and active community of contributors, which provides a wide range of pre-built modules and plugins for common tasks.

Despite the pfSense web user interface being fully documented, its back end lacks any documentation, so development related to managing pfSense through its back end comes down to reading and understanding the publicly available source code. pfSense handles configurations for all its components by first writing to an XML configuration file

/cf/conf/config.xml and then calling helper functions in PHP for performing the proper modifications on the underlying Operating system. These helper functions are stored in inc files in /etc/inc and are imported in other scripts with require_once. By reading the the web front-end source code in /usr/local/www, it's possible (though with some difficulty) to ascertain which helper functions should be called for various operations. These functions can be run in pfSense standalone in one of two ways: by running the /usr/local/php command that comes with the base PHP installation or by running the /usr/local/sbin/pfSsh.php developer shell created for pfSense. In our implementation, we settled on running PHP functions through /usr/local/php since this allows executing code in line with the -r command line flag rather than writing to the command's standard input like the developer shell. To make this process easier, we consulted the source code for the pfSensible Ansible community module, which contains many of the helper functions needed for performing an operation, listed without any additional user interface-related code [19].

V. Environment Provisioning and Management

After implementing our test environment and deploying our automation tool, we conducted tests on our *template*, *clone*, and *purge* commands. Six virtual machines were created and converted into templates for this task. These VMs are described in Table II. We varied the VM storage, memory, and Operating Systems in order to test the tool's ability to operate with different scenarios. Aside from installing the QEMU guest agent, no further modifications were made to the base installation.

TABLE II
Sample Virtual Machine Configurations

# of VMs	BIOS	Storage	Memory	OS
2	SeaBIOS	2 GB	512 MB	Debian 11
2	SeaBIOS	16 GB	4 GB	Ubuntu 22.10 Desktop
2	OVMF (UEFI)	64 GB	8 GB	Windows 11 Desktop

Timing was measured by prepending each of the *template*, *clone*, and *purge* commands with time on Linux. The Linux time command measures the time taken to fully execute a command. Template, clone, and purge are implemented to detect when all configuration changes to Proxmox and pfSense are completed, so the time they take to execute is an accurate measure of the time taken to fully perform these operations. Each command was executed ten times, and these times are averaged in Table III.

TABLE III
Timing Evaluation for the Automation Tool

Script	Average Execution Time (seconds)
Template	4.02
Clone	14.96
Purge	15.06

Based on these results, we have successfully achieved the goals of this project. Exact timing for templating, provisioning (cloning), and purging of environments is dependent on the environment, its size, and its state. Additionally, in order for virtual machines to be converted to templates, they were powered off prior to being converted to templates. If they were powered on, time taken to convert them to templates would include the time taken to receive the shutdown signal and perform a graceful shutdown, which is platform dependent.

Speed for HTTPS API requests to the Proxmox VE server and SSH connections made to the pfSense firewall will depend on distance between the remote manager running the scripts and these devices. It will also depend on the specifications for all devices involved, including allocated RAM and CPU speed. Therefore, the values in this table should not be taken as absolute, and we recommend performing testing on your environment prior to releasing it for production. However, with similar hardware, timing comparable to this is expected.

Sample operation of our *clone* command can be seen in Figure 2. In this figure, we can see the Administrator creating a new user "David" within the environment, selecting the six virtual machines for the environment, and setting up the DHCP and DNS configurations for the user. Changes made within Proxmox are shown in Figure 3, showing successful server provisioning.

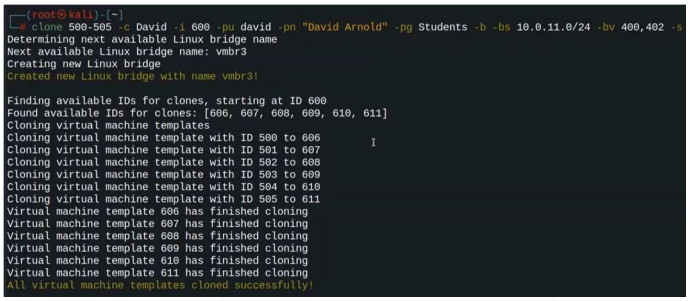
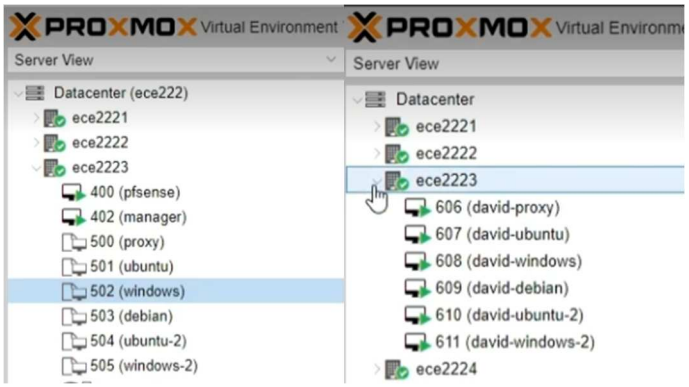


Fig. 2. Sample command line output from our *clone* command. The user "David" is created within the environment and six VMs are provisioned for their use.



a) b)

Fig. 3. Successful provisioning of servers for the user "David". In a), the administration view of the Proxmox web interface is shown with the six templates that we wish to clone. After cloning, we see the results in b). Since the user does not have administrative permissions within the environment they may only view their own machines.

Additionally, sample operation of our *purge* command can be seen in Figure 4. This command is simpler and should only be used when instruction has been completed as deleted server cannot be recovered after deletion. During this sample, the user "John" is purged from the environment and their virtual

machines are removed from both the Proxmox interface and the pfSense firewall configuration.

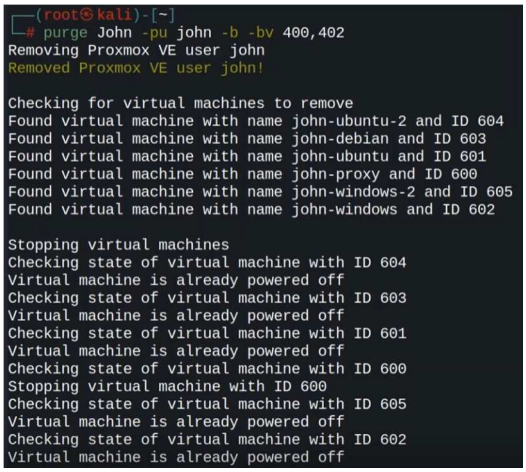


Fig. 4. Sample command line output from our *purge* command. The user "John" is purged from the environment and the VMs associated with the user are removed.

VI. Conclusion

Overall, we were able to successfully develop an automation tool for managing users and provisioning within a cybersecurity practice environment. Our tool was developed in the Python programming language and used the proxmox and ansible-pfsense packages in order to configure our Proxmox VE hypervisor and pfSense firewall. The tool exceeded our expectations in dynamically provisioning cybersecurity environments for students, providing a simple command line tool for creating templates, cloning VMs, and purging users. Further evaluation of the tool showed that it could create VM templates in 4.02 seconds, provision environments for students in 14.96 seconds, and purge environments in 15.06 seconds. This performance exceeded our initial goal of task completion within one minute.

In the future, we plan on incorporating this tool into our cybersecurity education programs, providing automated provisioning for course projects and exercises. We will also explore additional virtual machine and network configurations, conducting timing analysis and comparing it to the results received during this report. Additional network configurations would expand the number of scenarios available to evaluate students.

References

[1] U.S. Bureau of Labor Statistics, "Information Security Analysts," 8 September 2022. [Online]. Available: https://www.bls.gov/ooh/computer-and-information-technology/information-security-analysts.htm.

[2] (ISC)2, "(ISC)2 Cybersecurity Workforce Study," (ISC)2, 2022.

[3] J. David and S. Magrath, "A Survey of Cyber Ranges and Testbeds," Cyber Electronic Warfare Division, Australian Government Department of Defense , Edinburgh, 2013.

[4] L. M. Rossey, R. K. Cunningham, D. J. Fried, J. C. Kabek, R. P. Lippmann, J. W. Haines and M. A. Zissman, "LARIAT: Lincoln adaptable real-time information assurance testbed," in *Proceedings, IEEE Aerospace Conference*, 2002.

[5] M. Liljenstam, J. Liu, D. M. Nicol, Y. Yuan, G. Yan and C. Grier, "RINSE:The Real-Time Immersive Network Simulation Environment for Network Security Exercises (Extended Version)," *Simulation,* vol. 82, no. 1, pp. 43-59, 2006.

[6] O. Nock, J. Starkey and C. M. Angelopoulos, "Addressing the Security Gap in IoT: Towards an IoT Cyber Range," *Sensors,* vol. 20, no. 18, pp. 5439-5457, 2020.

[7] K. E. Balto, M. M. Yamin, A. Shalaginov and B. Katt, "Hybrid IoT Cyber Range," *Sensors,* vol. 23, no. 6, pp. 3071-3106, 2023.

[8] C. Cruz and P. Simões, "Down the Rabbit Hole: Fostering Active Learning through Guided Exploration of a SCADA Cyber Range," *Applied Sciences,* vol. 11, no. 20, pp. 9509-9531, 2021.

[9] V. Giuliano and V. Formicola, "ICSrange: A Simulation-based Cyber Range Platform for Industrial Control Systems," *https://doi.org/10.48550/arXiv.1909.01910.*

[10] M. Frank, M. Leitner and T. Pahi, "Design considerations for cyber security testbeds: A case study on a cyber security testbed for education," in *2017 IEEE 15th Intl Conf on Dependable, Autonomic and Secure Computing, 15th Intl Conf on Pervasive Intelligence and Computing, 3rd Intl Conf on Big Data Intelligence and Computing and Cyber Science and Technology Congress (DASC/PiCom/DataCom/CyberSciTech*, 2017.

[11] M. M. Yamin, B. Katt and V. Gkioulos, "Cyber Ranges and Security Testbeds: Scenarios, Functions, Tools and Architecture," *Computers & Security, doi: https://doi.org/10.1016/j.cose.2019.101636.*

[12] J. Eckroth, K. Chen, H. Gatewood and B. Belna, "Alpaca: Building dynamic cyber ranges with procedurally-generated vulnerability lattices," in Proceedings of the 2019 ACM Southeast Conference, 2019.

[13] C. Pham, D. Tang, K.-i. Chinen and R. Beuran, "Cyris: A cyber range instantiation system for facilitating security training," in *Proceedings of the 7th Symposium on Information and Communication Technology*, 2016.

[14] G. Shu, D. Chen, Z. Liu, L. Na, L. Sang and D. Lee, "VCSTC: Virtual Cyber Security Testing Capability–An Application Oriented Paradigm for Network Infrastructure Protection," in *Testing of Software and Communicating Systems: 20th IFIP TC 6/WG 6.1 International Conference, TestCom 2008 8th International Workshop, FATES 2008*, Tokyo, 2008.

[15] Proxmox Server Solutions GmbH, "Proxmox," 2023. [Online]. Available: https://www.proxmox.com/en/.

[16] Electric Sheep Fencing, LLC, "pfsense," [Online]. Available: https://www.pfsense.org/.

[17] "Proxmox VE API," 16 February 2023. [Online]. Available: https://pve.proxmox.com/wiki/Proxmox_VE_API

[18] "Welcome to Proxmoxer," 22 March 2022. [Online]. Available: https://proxmoxer.github.io/docs/2.0/.

[19] "ansible-pfsense / pfsensible.core," 15 March 2023. [Online]. Available: https://github.com/pfsensible/core