Leveraging Browser-Based Virtual Machines to Teach Operating System Fundamentals

Matt Ruff
College of Information Sciences and Technology
The Pennsylvania State University
University Park, PA, USA
mvr5567@psu.edu

Nicklaus A. Giacobe
College of Information Sciences and Technology
The Pennsylvania State University
University Park, PA, USA
nxg13@psu.edu

Abstract—In this paper, we identify challenges in delivering cybersecurity labs, including the overhead costs of delivering virtual machines to students. We propose instead to use JavaScript driven Browser-Based Virtual Machines (BBVMs) to overcome the challenges of Type I and II hypervisors, as well as vendor- specific cybersecurity lab ranges. BBVMs deliver configured VMs at lower cost to the student's web browser and are much easier for students to use. BBVMs require no hardware or infrastructure for students besides an Internet-connected device. As such, labs delivery via BBVMs can be run on mobile phones, tablets, or computers with limited resources. With this in mind, the authors detail BBVM implementation for cybersecurity labs. With very little physical infrastructure, programming, and systems administration, an educational institution at any level may implement a cybersecurity lab in such an environment. Our examples focus on addressing learning the Linux command line, introducing different Linux commands, and deepen student understanding of the Linux operating system itself. We combine BBVMs with previous work to address configuration, repeatability, assessment, academic integrity/cheating, and other similar constraints using our polymorphic configuration methodology called PolyLab. Lastly, we include a step-by-step procedure to implement BBVMs and show use-cases for cybersecurity education.

Keywords—operating system, Linux, command line, education, virtual machines, cybersecurity

I. Introduction

Previously, information technology and cybersecurity students relied on being physically present in a laboratory environment to gain hands-on skills in an air gapped sandbox to reduce concerns of unintentionally breaking the law. As hypervisors became more common, this made it easier for students to learn as one could simply create new virtual machines instead of physical ones. Unfortunately, this approach has several issues including cost of the hypervisor software and hardware limitations. With the advent of cloud computing, this further lowered the barrier to entry for learning hands-on skills. The main downside is that this typically requires a payment to a third-party company for infrastructure access, something that is not feasible at large scale. In academia, if lab content (regardless of the physical infrastructure) is static, educators can never ensure that students do not cheat or create a walkthrough and post it online.

Our solution ties together two concepts – browser-based virtual machines (BBVMs) and polymorphic system configuration to overcome these challenges. Students will be able to learn critical systems administration skills in a dynamic, hands-on environment and the only physical requirements are a web browser which supports JavaScript and an Internet connection.

II. Challenges and Limitations of Virtualized Cybersecurity Lab Systems

Most universities provide student access to laboratory environments one of three ways: in-person at on campus locations, on their own computers, or remotely via the web [5]. Lately, a fourth method of lab access has made its debut: Cybersecurity Labs as a Service. Regardless of how the lab environments are accessed, the lab environment should a) provide Internet access to acquire tools b) be isolated from campus networks c) provide realism d) have management oversight, and e) be provided to both resident and online/remote students [9].

A. In-Person Labs on University-Provided Hardware

The first approach of forcing students to come to campus to do lab-based assignments requires the professor(s) and teaching assistant(s) to build physical or virtual machines, create master copies, commonly referred to as golden images. There are a multitude of issues with this approach. First, this is exclusive as it does not allow distance learning as these labs are not easy to access remotely and are immobile [10]. Some instructors attempt to rectify this by doing a demonstration over broadcasting software, though this does not provide students with a hands-on experience [10]. Second, if running on Type II hypervisors on desktop hardware in the classroom, the hardware limitations of the systems are a bottleneck [5]. These virtual machines may run slowly, as they are typically provisioned in a limited resource capacity. Even if educators deployed the lab environment in a server environment on a Type I hypervisor, they still may have to throttle the amount of memory and processor allocation to ensure that everyone has an environment to work in. Third, these labs only have a handful of machines available to provision at one point in time [5].

B. Labs on Students' Hardware

The approach of using a student-owner machine and a Type II/desktop hypervisor lends itself to several of the same

issues as using university-provided hardware. For instance, students' computers are also limited by storage space, processing power, and memory available. In addition, this forces students to learn how to provision virtual machines using a hypervisor. This can be beneficial, as students may familiarize themselves with industry-leading software such as VMware but may also cause headaches as it is typical to change BIOS settings to use virtualized hardware. This also forces instructors into troubleshooting various issues that may arise due to the host operating system and hypervisor.

Du, et al. use this methodology with the SEED Labs [3]. In addition to the hypervisor installation, they experience issues with each lab's guest operating system changing from one version of Ubuntu X.04 LTS to the next. This causes dependency issues with the lab configuration. At last check, Du was working on a Dockerized version of the experience, so that each configuration can remain static, even if the guest OS changes. While this will address the stability of the lab from one long-term-stable (LTS) version of Ubuntu to the next, it does not address the hypervisor and network configuration concerns.

C. *Remotely Accessed via Web through Jump Boxes*

Some academic institutions moved to providing access via Secure Shell (SSH) or Microsoft's Remote Desktop Protocol (RDP), as the institution itself hosted externally facing software [5]. If the students are already on a *NIX system, then SSH is available via the command-line interface (CLI). Otherwise, if on Windows, students must download and use a tool such as PuTTY. Likewise, if using the Remote Desktop Protocol, students on *NIX systems must download the appropriate RDP tool for their systems. Other institutions provide access by running a VMware ESX instance or VMware vCenter server which students may connect to. Like the other two solutions, this is limited by the hardware resources as these virtual machines are typically provisioned with few resources. One study reports that the most commonly reported down-sides to hosting remotely accessed labs are funding and hardware [7]. Regardless of how the lab is remotely accessed, students are typically given a Virtual Private Network (VPN) and connect to the lab via that [9]. These environments could also be hosted in the cloud, such as on Amazon Web Services (AWS) as it may provide a way for educators with less funding to provide the same capabilities [10].

D. *Cybersecurity Labs as a Service*

The newest method of cybersecurity lab that exists is that of a service. One in which educators can pay a service provider, such as U.S. Cyber Range, Practice Labs, Tele-Lab [10]. These are commonly referred to as Cyber Ranges and they simulate cyber operations against fictitious organizations [8]. These options exist to eliminate the need to provision virtual machines for students. The instructors must create a golden image, as if the lab is being hosted on premises, but instead they send it to the cybersecurity lab service. This vendor hosts the lab, and the students log in. This suffers from many of the same faults as previous instances, such as the lack of resources. In the authors' experiences, these labs typically provide increased latency which serve as a source of frustration for students. These labs may also require specific configurations to work on the vendor's platform. This non-trivial task of adaptation of the instructor's lab to meet the vendor's requirements can be time consuming.

III. Related Works

A number of other innovative methods of providing access to cyber-security labs in the cloud have come before this iteration of PolyLab, and as such, our paper builds upon a handful of other similar ideas in other problem spaces: namely browser-based virtual machines, automatic problem generation for Capture the Flags, and hands-on Linux labs. We merge these three ideas together and apply the solution first to hands-on instruction of Linux system administration skills and commands and hopefully towards other skills critical for a successful career in information technology and cybersecurity.

A. *Cloud-Based Virtual Machines for Students*

Hosting virtual machines in the cloud for students to access is increasingly common and will provide in-person and remote students the same lab setups and experiences [9]. Providing these labs to all students, regardless of physical proximity to the university, is paramount for a well-trained cybersecurity workforce. One of the most critical factors of the move to the cloud is the ease of use for students, and the fact that the golden images provide all required software tools to students [9]. This mitigates any issues with installation by students and enables instructors to identify pitfalls with licensing, installation, virtual machine's memory capacity, and dependent packages. Removing these potential issues from the students' hands enables the students to focus solely on meeting the objectives of the lab itself.

Tele-Lab is an example of a cloud-based virtual machine. Students can connect to the site via web browser, but they must establish a remote desktop connection to a virtual machine [10]. The labs in Tele-Lab are referred to as learning units, and they follow identical formats: academic background information, the presentation of tools that could be used to carry out attacks, and then asking the student to conduct the attack.

Nevertheless, it is quite common to have to build and provision virtual machines by hand, at least once. RunLabs, from the University of Indiana – Purdue University Indianapolis, attempted to rectify this by using JavaScript Object Notation (JSON) files for the creation of virtual machines with specified network settings [5]. This automation saves time and is ideal for rapidly provisioning an operating system with a specified lab inside of it.

B. *Browser-Based JavaScript Virtual Machines*

Bellard seems to have invented the first iteration of a JavaScript-based Linux virtual machine in a web browser in 2011 [1]. This was built upon the QEMU emulator, which Bellard originally created and presented at the USENIX conference in 2005 [1]. QEMU is a machine emulator which can run a myriad of operating systems [1]. To use JSLinux, a

disk image would have to be built using QEMU and then JSLinux would interface with it. When porting PolyLab to the web began, we initially attempted to modify and use JSLinux. JSLinux is not truly open source and is copyrighted. At that time, user-friendly documentation for JSLinux was not available, and deobfuscated source code was only available via third-party GitHub repositories. As such, the authors sought out other solutions that worked similarly to JSLinux and QEMU.

After attempting to use JSLinux, we found the v86 project on GitHub by the user "copy" [2]. Licensed under a BSD 2-Clause license, we were able to use the v86 project as long as the original copyright is retained. With this in mind, we used this as the underlying platform for PolyLab. Several challenges were encountered and subsequently overcome along the way. First, the documentation for building a new image was three years old and was catered to Arch Linux [2]. As it was three years old, it was written with a specific variant of Arch Linux in mind, as the v86 project requires an x86 architecture. At the time of writing, Arch Linux 2020-11-01 works with the v86 project. Once a successful Arch Linux image was built, we were able to embed our polymorphic configuration code and the required packages. We carefully worked to minimize the size of the disk image itself. From there, v86 uses the 9p filesystem. This required manual steps to be executed. We will submit a pull request to the v86 project to update the documentation in the near future. With a 9p filesystem and disk image, artifacts must be created to serve the v86 filesystem over the web [2]. As a side note, we did not utilize any networking functionality, even though v86 can support networking via emulation. This may be of use to future PolyLab projects.

C. Current Hands-On Cybersecurity Lab Content

Several different lab environments exist for students to learn cybersecurity skills. Some of these are remotely accessed via the web browser or VPN credentials, some of them are hosted locally. Of the currently available cybersecurity laboratory environments, the Security Education Labs, or SEED Labs, from Syracuse University is one of the most well-known. It is provided via a ready-to-go virtual machine image that works with Oracle's VirtualBox [4]. These labs provide many different exercises for students to gain an understanding of various cybersecurity attacks and defenses. Examples of these exercises include Buffer Overflows, Distributed Denial-of-Service (DDoS) and Cross-Site Scripting (XSS)[4]. In December 2020, a SEED Labs 2.0 Beta was released [3]. This upgrades the virtual machine to a 64-bit version of Ubuntu 20.04 Linux. SEED Labs also provides a cloud-ready variant of the virtual machine for ease of use if not deployed locally, but remotely in the cloud.

In 2017, a pair of researchers modified SEED Labs, as written, to make them more useful for job seekers as the labs do not venture into domains such as malware analysis and reverse engineering, and vulnerability scanning and assessing [6]. As such, they have created an extension of the SEED Labs which they hope will be integrated with SEED Labs. These exercises provide yet another way for students to gain hands-on experience with industry-standard tools, such as writing YARA rules [6].

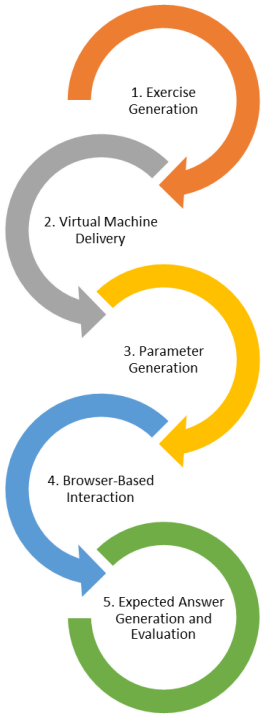
Another common cybersecurity exercise is the Bandit exercise hosted at https://bandit.overthewire.org. Like other websites, the purpose of Bandit is for Linux learning, but, similar to SEED Labs, it suffers from the problem of having become too public, and as such, walkthroughs have become widespread and are easily available with a simple Google query.

IV. System Design

A. Logical Components of a Lab

We now present the overall system design of several components to deliver a polymorphic experience to learn the Linux command line, with the goal of launching such an environment via a BBVM. The intent here is to be able to execute the lab experience with little-to-no infrastructure or resources. We wish to avoid hypervisors on local computers or a complex lab to deliver access to sandboxed guest operating systems. Instead, we rely on the JavaScript-based operating systems. We deliver the configured virtual machine to the student via a standard web server/browser arrangement. The virtual machine is delivered with scripts installed to configure the system to run the exercise. As the student clicks on the link, their browser downloads the virtual machine and runs it in their own browser. From this point forward, all resources run on the student's computer.

In Fig. 1, the exercise is coded in Step 1 – Exercise Generation. In this stage, developers program the different parts of the exercise. For the exercise presented here, each "level" of the game is generated. The password for the next level is encoded from the value of the user's hash. A script is created to fabricate the exercise on the virtual machine once the user enters their credentials. The script is pre-populated on the virtual machine image so that it can be delivered when the user asks for it in Step 2, Virtual Machine Delivery.



Step 1: Exercise Generation - Programming the Exercise Generating Script

Step 2: Virtual Machine Delivery – Pre-populated virtual machine image is Delivered to the end-user.

Step 3: Parameter Generation – Exercise Generating Script takes input from the user, generates the parameters and fabricates the exercise Environment.

Step 4: Browser-Based Interaction – End user plays the game on the Virtual machine in their own browser and figures out the answers.

Step 5: Expected Answer Generation and Evaluation – End user enters their answers on a web form. Web server generates the expected answers and compares to the submitted answers and reports the score/results.

Fig. 1. System Design

Once the end-user has asked for and received the virtual machine image, the virtual machine is sent, and all future actions are completed inside the JavaScript sandbox in the browser on the end-user's computer. It runs and launches the guest operating system and initiates the Exercise Generation script. The user is asked for their ID. Step 3, Parameter Generation begins, and each parameter is inserted into the exercise. Target files are placed on the guest operating system and the game is started at Step 4, Browser-Based Interaction. At this point, the user plays the game, figures out the answers to the various challenges and records the answers to each level (see Fig. 2).

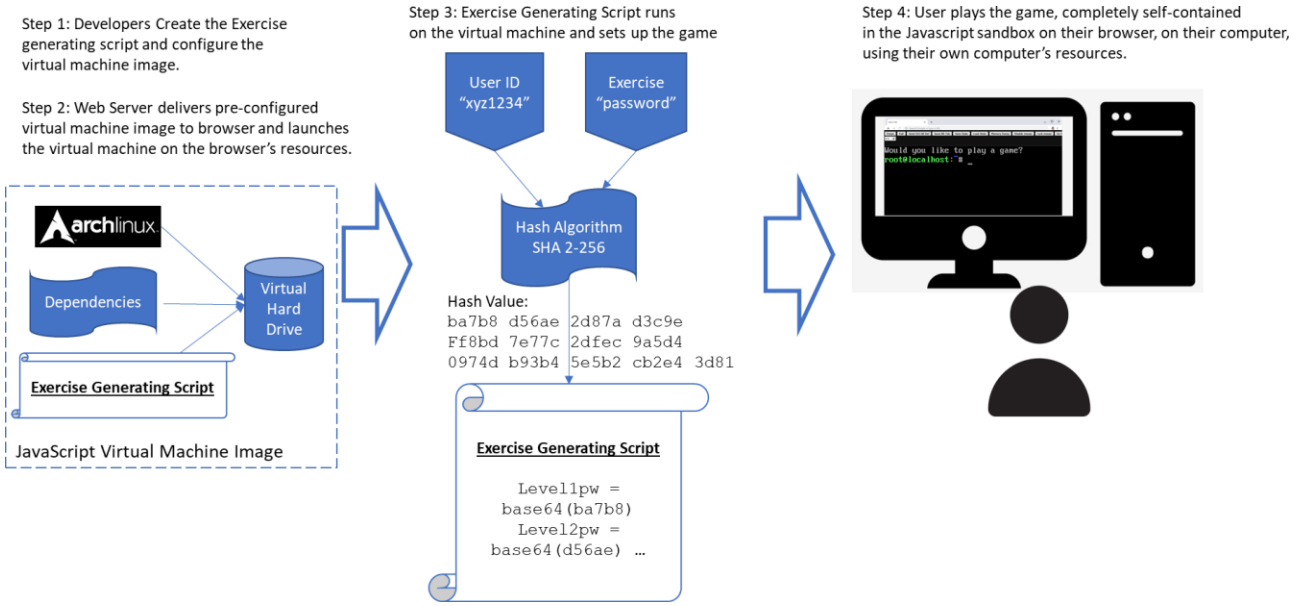


Fig. 2. Exercise Generation Using Javascript Virtual Machines

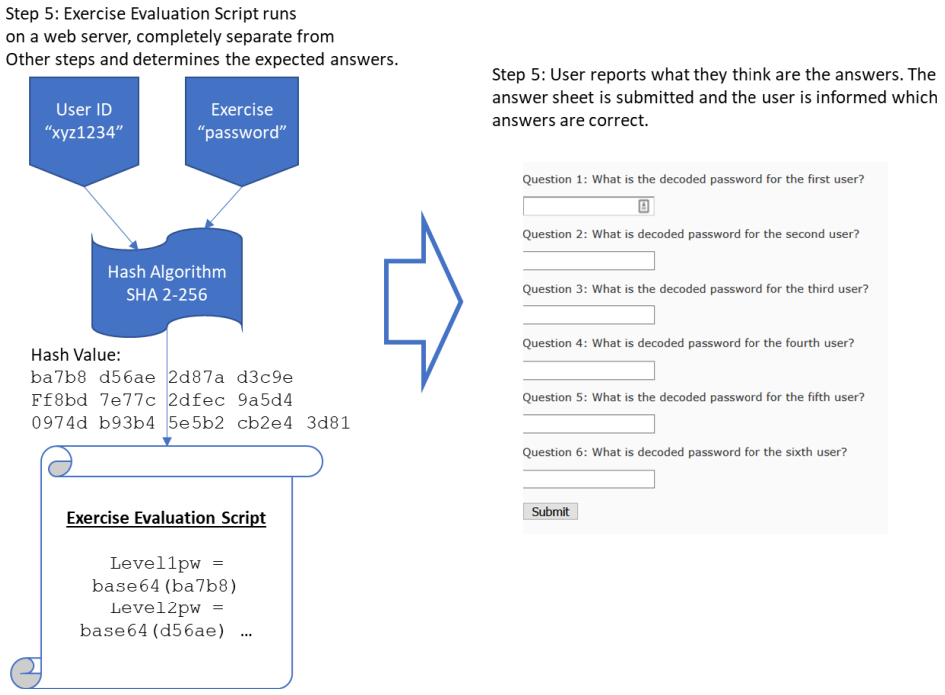


Fig. 3. Answer Evaluation

The user completes the exercise and then goes to a web form to enter their answers and verify that they completed the exercise. On that web server, Step 5 – Expected Answer Generation and Evaluation runs. The Exercise Evaluation Script asks the end user for their ID and generates the expected answers. It compares those answers to the ones the user provides and generates a score (see Fig. 3).

B. Disconnected Parameterization

We leverage the same technologies that Giacobe and Kohler used in PolyLab. The generation of parameters lend themselves to the disconnected nature of the JavaScript-based virtual machines on the end-user's browser. We cannot expect reliable network communication between the Linux virtual machine on the student's browser and any other server or system on the Internet. In fact, for many cybersecurity exercises, we may wish to completely block network access to keep the student "safe" from accidental use of cybersecurity/hacker tools. However, these parameters are generated from the hash of the user's ID and the exercise password. We can verify that the student has figured out the expected/right answers and verify their answers on a simple web form.

At some point in the future, integration of answer verifiers in commonly-used course management systems (Canvas, Blackboard, Google Classroom, etc.) may be possible if those systems can be configured with the algorithmic computation of the expected answers.

C. Repeatability

One challenge with some exercises is that they are not repeatable. Once a student knows the expected answer, giving them the question set again defeats the goal of having them do the exercise a second time. We solve this problem as a new set of questions can be generated by manipulating the input values of the hash. One such way is to simply add an "attempt number" to the input values of userid and password. Since a hash is used to create the pseudorandom parameter values, a good hashing algorithm (SHA-2, SHA-3, etc.) will be able to generate 224-512 bits of unique values. Each variation of the combination of userid, exercise password and other input values like dates, attempt numbers, would provide a different set of lab parameters.

This system is essentially capable of creating billions upon billions of different iterations of the labs, assuming that every bit of a long hash digest is used to parameterize an exercise.

V. Creating a Browser-Based Virtual Machine

PolyLab's Browser-Based Virtual Machines consist of four critical components: 1) the PolyLab code stored in a GitHub repository, 2) David Humphrey's browser-vm GitHub repository, which uses 3) the Build-root tool, and 4) the v86 project created by Fabian (who goes by his handle 'copy'). Ultimately, the PolyLab code gets embedded in a Buildroot- generated ISO operating system and launched by v86.

To begin building a Browser-Based Virtual Machine, one must first have an idea of which target operating system they

would like to use. Currently, v86 supports about 20 operating systems [2]. Originally, both Arch Linux and Debian were considered as Copy should support both. At the time of testing, we were unable to successfully build either. This was because the documentation in the v86 project was several years out of date for Arch Linux, and the Debian Docker container had several errors which prevented it from building correctly [2]. While we worked to fix these errors and update the documentation accordingly, we also pursued Buildroot as a viable alternative in case Arch's and Debian's issues could not be rectified in reasonable time.

At this point, Buildroot was the most direct path forward. Conveniently, it kept the images small, as this would be transmitted over the user's network connection, and the file system itself would also be stored in the browser cache. This is because the specified Buildroot configuration uses the 9P file system.

To install all the dependencies required for v86 and browser-vm onto a fully updated Ubuntu 20.04 instance, the following commands are needed:

```
sudo apt install openjdk-11-jdk git make
gcc gdb nasm qemu-system clang node npm
docker

sudo apt-get install libc6:i386
```

Users will also have to install Rust Nightly build with a wasm32-unknown-unknown target. Following this, users will run the following git clone commands:

```
git clone https://github.com/copy/v86

git clone
https://github.com/humphd/browser- vm git
clone
https://github.com/giacobe/PolyBandit
```

After these commands are executed, everything will be on the system. Users can also test the built v86 system in accordance with the v86 README.md file using wget and make to pull down test images and build them. This does add extra files to the disk and may not be necessary or desired on production systems.

Next, the user should move onto building the Buildroot Docker container from the buildroot-vm directory, as shown below. This creates the Docker image which can then be ran in a few different ways – one for rapid building, and one used for debugging or testing. To build the Docker container and run it for production, using the buildroot-vm defaults:

```
sudo docker -t buildroot .

sudo docker run –rm –name build- v86 -v
$PWD/dist:/build -v

$PWD/buildroot-v86/:/buildroot-v86
buildroot
```

The Buildroot Docker container generates a .iso file ready for use within the v86 system, but it may not be suitable for our purposes. It is good for testing, and seeing how the overall system works. To modify the Buildroot filesystem, ensure that the rootfs_overlay/ directory has a directory inside of it called "home" where the user will be dropped into. Inside of the home directory, place any subdirectories or files for the user to find upon first boot. To modify things like the hostname, this can be done using an interactive Docker session that drops the user into a Bash shell. Within Bash, the user may modify via make menuconfig and make linux-menuconfig to modify information such as the hostname and various packages that should be pre- built into the .iso. This results in overall modifications to the makefiles.

VI. Merging Browser Based Virtual Machines and Cybersecurity Labs

This work merges several previously presented ideas together. As presented here, the end goal is to require only a web browser and an Internet connection on the behalf of the student, and only a simple web server to deliver the BBVM. For the student, anyone in the world should be able access via a cellular phone, a tablet, or a computer. Access to a web browser with JavaScript enabled is the only requirement. For the instructor/institution, only a basic web server is required to deliver the configured VM.

When a student lands on the webpage, they will be dropped into a custom JavaScript-based Arch Linux virtual machine which is self-contained in the web browser. When the user exits the webpage, currently their session is gone. This system will provide several features to the user that should be familiar to anyone with experience using virtual machines. For instance, this will enable the user to save the state of their virtual machine to their local machine, at which point it could be restored back into the system at a later point. Similarly, one can upload files and folders directly to the virtual machine for testing. The combination of the saving and restoring in tandem with the file and folder upload means that it is fairly easy to share the state and content of a system. With this in mind, it can be used for students to aid their peers if they are stuck, or for the teaching team to troubleshoot issues encountered by students.

VII. Conclusion

We presented the incorporation of JavaScript-based virtual machines into a system to teach the basics of the Linux command line to new students in cybersecurity academic programs. This enhancement will allow us to engage students at our institution as well as students from across the world. Because a virtual machine runs in the end-user's web browser, no centralized hypervisor or resources are needed, which results in significantly lower costs to deliver this style of exercise to students.

We also presented concepts of polymorphism that will have the potential to positively impact student engagement and retention, while reducing copy/paste cheating. Future work in this area will focus on the specific learning objective development for the command line exercises. We have three command line Linux exercises planned. Additionally, we plan to develop exercises in basic and intermediate databases (SQL), as well as several exercises in network security.

REFERENCES

[1] Fabrice Bellard. [n.d.]. QEMU, a Fast and Portable Dynamic Translator. Technical Report.

[2] Copy. [n.d.]. v86: x86 virtualization in your browser, recompiling x86 to wasm on the fly. https://github.com/copy/v86

[3] Wenliang Du. [n.d.]. SEED Project. https://seedsecuritylabs.org/

[4] Wenliang Du, Zhouxuan Teng, and Ronghua Wang. 2007. SEED: A Suite of Instructional Laboratories for Computer SEcurity EDucation *. Technical Report. http://www.cis.syr.edu/

[5] Connie Justice and Vyas Rushabh. 2017. Cybersecurity Education: RunLabs Rapidly Create Virtualized Labs Based on a Simple Configuration File. Technical Report. https://scholarworks.iupui.edu/handle/1805/15799

[6] Min Jin Kwon, Gowoon Kwak, Siyoung Jun, Hyung Jong Kim, and Hae Young Lee. 2018. Enriching Security Education Hands-on Labs with Practical Exercises. In Proceedings - 2017 International Conference on Software Security and Assurance, ICSSA 2017. Institute of Electrical and Electronics Engineers Inc., 100–103. https://doi.org/10.1109/ICSSA.2017.8

[7] Nancy Martin and Belle Woodward. 2013. Building a Cybersecurity Workforce with Remote Labs. , 57–62 pages. https://isedj.org/2013-11/N2/ISEDJv11n2p57.html

[8] Marina Ribaudo and Andrea Valenza. 2019. Semi-automatic generation of cybersecurity exercises: A preliminary proposal. In EnSEmble 2019 - Proceedings of the 2nd ACM SIGSOFT International Workshop on Ensemble-Based Software Engineering for Modern Computing Platforms, co-located with ESEC/FSE 2019. Association for Computing Machinery, Inc, New York, New York, USA, 16–21. https://doi.org/10.1145/3340436.3342728

[9] Khaled Salah, Mohammad Hammoud, and Sherali Zeadally. 2015. Teaching Cybersecurity Using the Cloud. IEEE Transactions on Learning Technologies 8, 4 (Oct 2015), 383-392. https://doi.org/10.1109/TLT.2015.2424692

[10] Christian Willems, Thomas Klingbeil, Lukas Radvilavicius, Antanas Cenys, and Christoph Meinel. 2011. A distributed virtual laboratory architecture for cyber-security training - IEEE Conference Publication. https://ieeexplore.ieee.org/document/6148474