University of New Orleans

ScholarWorks@UNO

University of New Orleans Theses and Dissertations

Dissertations and Theses

Summer 8-9-2017

Lightweight Environment for Cyber Security Education

Vivek Oliparambil Shanmughan

Vivek Oliparambil Shanmughan, volipar1@uno.edu

Follow this and additional works at: https://scholarworks.uno.edu/td

Part of the Information Security Commons

Lightweight Environment for Cyber Security Education

A Thesis

Submitted to Graduate Faculty of the
University of New Orleans
in partial fulfillment of the
requirement of the degree of

Master of Science
in
Computer Science
Concentration in Information Assurance

by

Vivek Oliparambil Shanmughan

B.Tech., Mahatma Ghandi University, 2012

M.S., Teesside University, 2012

August 2017

Dedication

This thesis work is dedicated to my friends, family & teachers

Acknowledgment

I would first like to thank my thesis advisor Dr.Vassil Roussev of the Dept. of Computer Science at University of New Orleans. The door to Prof. Roussev's office was always open whenever I ran into a trouble spot or had a question about my research or writing. He consistently allowed this paper to be my own work, but steered me in the right the direction whenever I needed it. I am extremely thankful and indebted to him for sharing expertise, and sincere and valuable guidance and encouragement extended to me.

Special mention to Mr. Saranyan Senthivel, Miss. Sneha Sudhakaran, Mr. McLellan and my peers at the University of New Orleans for providing me their valuable feedback, suggestions and other recommendations that helped me work towards the completion of this research.

Finally, I express my very profound gratitude to my siblings Mr.Vishak and Mrs.Chitra, my girlfriend Miss Anna Weber and my parents for providing me with unfailing support and continuous encouragement throughout my years of study and through the process of researching and writing this thesis. This accomplishment would not have been possible without them. Thank you.

Vivek Oliparambil Shanmughan

Table of Contents

Table of Figures

Abstract

The use of physical systems and Virtual Machines has become inefficient and expensive for creating tailored, hands-on exercises for providing cyber security training. The main purpose of this project is to directly address these issues faced in cyber security education with the help of Docker containers. Using Docker, a lightweight and automated platform was developed for creating, sharing, and managing hands-on exercises. With the help of orchestration tools, this platform provides a centralized point to monitor and control the systems and exercises with a high degree of automation. In a classroom/lab environment, this infrastructure enables instructors and students not only to share exercises but also helps create and deploy exercises more easily. By streamlining the end to end delivery and deployment of the exercises, instructors can now efficiently make use of the class/lab hours in educating the students rather than performing system administration tasks.

Keywords: Cybersecurity, Docker, Ansible, containers, automated learning infrastructure, cost-effective platform, hands-on training, computer security, support course project, virtual network.

Chapter 1 Introduction

1.1 Cybersecurity Education and current learning infrastructure

The number of cybercrimes has been increasing recently, with the years 2015 and 2016 reporting some of the biggest data breaches ever [1][2]. In those two years alone, at least half a billion personal records have been stolen and cost the organizations several millions dollars, directly or indirectly, from the breach. Existing technical defenses, like antivirus and network security devices, are designed to shield the users from most cybersecurity breaches and threats. However, they can be ineffective against highly sophisticated and targeted attacks based on social engineering. The ultimate responsibility of safeguarding the computers and organizational network from the cyber threats therefore rest upon the users – the weakest link in cybersecurity space. Cyber security education is therefore essential in educating the users for securing any critical piece of information, system or network from any potential threats and vulnerabilities.

According to the analysis made by security organizations like Bunker and SANS Institute [3][4], it was revealed that about 93% of all breaches that happened in 2015 and 2016 was avoidable. Servers were found to be running vulnerable version of software, unpatched or outdated version of operating systems etc that the attackers used to compromise the security. Such lapse in security often occurs because of lack of proper knowledge, experience and tools required for protecting resources from cyber threats. Hands-on cyber security education is therefore vital in developing the knowledge and necessary skills required to tackle cybersecurity threats of today.

In order to facilitate education and hands-on training in cybersecurity, instructors often resort to the use of simulated or hosted cybersecurity scenarios. Such scenarios provide the base required to help the trainers to educate on various cyber-attack/defense scenarios and different cyber forensic tools. It also helps to study and evaluate the strengths and weakness of different cybersecurity tools in such scenarios. The knowledge and experience acquired from these hands-on exercises aids in better decision-making against the different cybersecurity scenarios we encounter in the real world.

Typically, for providing hands-on training the instructors and/or admins requires a learning environment where different security scenarios can be hosted or simulated. In the past, several projects were funded to create environments that help simulate real world cybersecurity scenarios for the purposes of learning or research. But, there has not been much effort to make learning

environments accessible and supported by hardware resources available to the students in a typical classroom or lab facility. The exercises developed based on the data gathered from scenarios are useful to demonstrate or teach different security attack or defense methodologies. However, such exercises are unfit as an evaluation tool in classroom environment due to factors such as their large size, limited flexibility to extend or modify data used in the exercises, availability of solved exercises, etc. Common challenges with such scenarios are the significant amount of human resources and complicated or expensive hardware required for hosting those scenarios. It is neither practical nor financially justifiable to host such scenarios in a classroom or lab environment where resources can be limited.

Learning scenario

There are two essential scenarios that need to be supported in any hands-on cybersecurity learning environment—management of live lecture/lab classes, and support for practice outside the classroom.

In a classroom environment, hosting cybersecurity exercises requires lot of time and effort from admins and instructors, especially for developing and delivering realistic cybersecurity exercises. Despite efforts to make the management of these exercise easy and accessible, in practice, this has been an elusive goal. Many still require the execution of system administration tasks, such as the installation of software packages, and the adjustment of configuration settings on the student systems. Automation scripts that attempt to hide these complexities are often not robust enough to handle all possible situations, thereby necessitating manual debugging and setup of the environments. This can be particularly disruptive during lectures and lab exercises, as the instructor ends up wasting valuable class time.

The classroom infrastructure currently deployed for cybersecurity exercises consists of general purpose virtualization tools, and does not *specifically* support the classroom scenario. In particular, there is little built-in automation and monitoring capabilities to manage the exercises. For example, if students are having difficulty with some of the tasks, instructors do not have a ready mechanism to be notified, or to intervene from their workstation.

In the out-of-class scenario, it is crucial for students to be able to work on the exercises in a minimal environment, such as an affordable laptop. Prior approaches, based on full-stack VMs, often have implied system requirements that are beyond the capabilities of the consumer devices owned by the

students. For example, exercises may consist of several virtual machines, which often exhausts the RAM capacity, leading to poor performance and a bad user experience. Therefore, it is essential that the learning infrastructure be lightweight enough to comfortably run on relatively constrained hardware.

1.1.1 Current approaches

Providing hands-on training in a live networked environment is not just unsafe but impractical for labs or classroom learning. To facilitate learning that resembled real word scenarios, cybersecurity trainers often consider simulated environments as a reliable alternative. Simulated environments are created by scripting various cybersecurity scenarios as carefully planned steps/procedures executed in a network of systems. Many solutions have been developed for creating and deploying such cybersecurity scenarios however, most of them lacks the flexibility enough to cover for the needs of cybersecurity education, today. Some of the commonly used solutions for providing hands-on cybersecurity training are discussed below in detail.

a) Physical System Infrastructure

In this infrastructure, all the computational or networking requirements of the scripted scenarios are met with physical hardware like wired LAN, switches and computer/server. In addition to the hardware, successful execution of the scenario critically relies on dedicated professionals to perform specific tasks at/over a specific interval of time. This strong dependence on hardware, software, and human resources makes physical system infrastructure less flexible, inefficient and costly to support a cybersecurity classroom.

The reproducibility of the scenario depends on several factors like availability of specific version of software, operating system, kernel version, etc. Even inside an ideal setting of this environment, successful recreation of the scenario demands strict adherence to the script, requiring dedicated personals to perform specific tasks at/over a specific interval of time. This strictness makes the infrastructure less flexible, as any changes made to the scenario or failure to comply with the scenario may require re-execution of the entire script. This not only increases the amount of time required to create and update exercises but also increases the cost as a result of the high demand for human and system resources. All the system and network requirements of a scenario in this infrastructure are done manually, often with very little automation. Considering the fact that the requirements for scenario are usually different from case to case, setting up the environment can be a time consuming

and technically challenging. In a classroom environment, this creates a significant gap in the amount of time required to switch between two exercises. This time gap makes the infrastructure unfit for rapid deployment of the scenarios.

The hardware and software requirements vary between scenarios, with some scenarios requiring more resources than the others. This varied requirement can potentially create a situation where large sums of money have to be invested in specialized hardware, even when the currently available resources are not used to their full potential. In a well-funded organization, this additional investment maybe insignificant however this is not the case with most small classroom or other academic facilities where money and resources are limited. Further, when scenarios are modified over time to meet new requirements, there is a distinct possibility that some of that expensive hardware might not be of any use at all.

Lot of time is required for creating hands-on cybersecurity exercises in a physical system infrastructure. Even more time is required for deploying them as the exercises and their environments cannot be shared, monitored or controlled by the administrators, remotely. Students and instructors are also expected to possess some level of knowledge in the installation and troubleshooting of systems, software and networking elements to handle issues in this learning environment. This additional learning requirement makes the infrastructure less user friendly and difficult to manage, especially in the hands of inexperienced users.

b) Virtual Machine Infrastructure

Lot of the limitations in supporting hands-on cyber security in a classroom environment with physical system infrastructure was overcome with the help of virtualization. With Virtual Machines or VMs, it was possible to create and deliver hands-on cybersecurity exercise that was more flexible and manageable. It was quickly adopted as the most common means to create and deliver hands-on cyber security exercises, especially because of the low cost of VM infrastructure. However, even the VM based infrastructure was not without flaws when it came to m

Solutions based on VMs brought, to those involved in cybersecurity education, an effective means for creating, sharing and delivering reproducible exercises. VMs enabled instructors to create custom scenarios and exercises as portable VM images and distribute them to students other instructors using sharable media. These preconfigured images are platform and OS independent, and can be brought up and running in a few minutes. This highly reduced the complexity and effort required to

set up exercises as well as switch between different exercises without any hardware or configuration changes. These advantages, along with were their relatively low cost of setup and efficient resource utilization led to the widespread acceptance of VMs in supporting cybersecurity classrooms.

However, the VMs were not without flaws and had several limitations like difficulty in management, storage and resource requirements, etc. which was often overlooked in the past to favor their low cost. Over time, as the cybersecurity scenarios got complicated, these limitations became more prominent and ultimately, prevented VMs from efficiently satisfying the needs of cybersecurity education. For example, while it was relatively easy to run cybersecurity exercises inside a VM, creating and updating them was a lot harder. Once a scripted scenario was available, admins or instructors have to manually install, create and configure the requirements of the exercises into the VM. Automated scripts can be unreliable, especially when it comes to meeting complicated requirements and troubleshooting configuration/installation issues. Updating or modifying the exercises is an additional challenge and might require the entire image to be recreated. Dealing with these challenges requires skilled professionals with good knowledge and experience in dealing with system administration tasks.

One of the inherent limitations with VM based cybersecurity exercises is that they are quite 'heavyweight'. These exercises required a lot of storage space and computing resources to function. This large size is due to the fact that the exercises catered as VM images are built on top of   the exercise VM image also carried the size of base OS. Lot of the space can be saved by creating exercise on top of a minimal OS image but even then, it doesn't fully address all the issues. For instance, the size of a simple LAMP server configured as a VM image is around 300-400 MB. This is not much when compared to the size of an Ubuntu OS image, which is around 700-800MB. So, comparatively it is more portable but as complexity of the exercise increases, the portability decreases rapidly. This reduced portability can be attributed to the fact that multiple VMs maybe required to simulate a single scenario and therefore, drastically increase the overall size of exercises.

The increase in the size of the exercises meant that large amount of storage space have to be allocated for these exercises. Additionally, Sharing of large images over the network is time consuming and create network congestion, especially with concurrent request from multiple machines. Sharing of exercise by copying using removable media is effective for small size network, however time consuming and physically demanding when there are multiple exercises to be copied. Physically copying is a daunting tasks in a large sized network even for small sized exercises.

Moreover, VM based infrastructure lacks a centralized monitor and control facility over exercises and systems in the infrastructure. In the absence of such a facility, students and instructors are forced to learn basic system administration tasks to tackle the challenges involved with starting and using these exercises. Virtualization software like VMware Server or VirtualBox can provide facility for centralized control, however provisioning of such facility require purchase of expensive licenses. Even then, most virtualization software only allow GUI based control over other virtualization clients and therefore cannot be scripted or automated. These software solutions lack the fine grained control required over the exercise and systems in infrastructure, making purchase of their licenses unreasonable or questionable at best.

In short, the complexity of creating, sharing and delivering cybersecurity exercises along with lack of centralized monitor and control facility makes VM based infrastructure, from a management perspective, difficult and expensive. From a user perspective, the increase in storage, memory and CPU requirements pushed the limits of an average consumer laptop, especially when handling exercises that requires multiple VMs.

c) Using data from real world network and simulations

Instead of setting up complicated test environment, data acquired from real world network environment/infrastructure or other simulated experiments can be used. Network data, system logs, memory/disk images make up for valuable forensic information regarding a cybersecurity scenario. These data does not allow instructors to create a live attack scenario, but they can be used to build exercises that focus on post attack investigation. The presence of sensitive or confidential information in the real world data carries a potential risk but a desensitized version of the same data can be of great use in cyber security education. However, censoring gigabytes of data is a lot of hard work and inefficient use of time, especially when there is no guarantee on the usefulness of desensitized data.

Alternatively, data can also be acquired from a test network or environment by simulating an attack or defense scenario. For example, the M57-Patents project [5] funded by the National Science Foundation, present such a case. The project followed a scripted scenario, where three hostile insiders perform different exploits in the small sized network of a fictional company. After execution of the script, spanning over three weeks, all the hard drive and main memory images, network packets and contents from other hot swappable devices were captured on a daily basis. The data,

summing to a total of 1.5 TB of forensic information, obtained from the M57 scenario presents a valuable dataset and still remain as a testbed for examination of different cyber forensic techniques.

However, from a realistic point of view these data set comes with a few limitations that makes it less practical for use in a classroom/lab learning environment. Firstly, the data sets gathered from the above experiments require large amount of storage space, often several terabytes. Secondly, the amount of manual work and infrastructure required for the setup of these scenarios is not justifiable, from a financial point of view, for a small lab environment. Thirdly, all the infrastructure and custom scripted actions makes the data set from these infrastructure represent only a specific cyber security scenario. Making changes to the scenario or data set is often not possible unless the entire script of the scenario was executed again. Due to this limited flexibility, data sets and scenario are hardly changed, thereby making the exercises based on this infrastructure less challenging over time. Even more so, as the exercises and their answers becomes available online. As a result, gathered dataset can get outdated and unfit for evaluation purposes very quickly, making the entire effort and investment made to generate those data set, futile.

Summary

From observations made on the current learning infrastructure, it is reasonable to assume that the task to create, deliver and share exercises for hands-on cybersecurity training is still an unresolved problem. Incorporating different applications, environments and other supporting services into the exercises requires the infrastructures to meet a very complex set of requirements. Requirements and challenges becomes more complex with the continual need to raise the standard and quality of the hands-on exercises with updated content or cybersecurity scenarios. As a result, it is getting difficult for classical solutions using physical or virtual workstations to keep up with these complicated requirements.

Most of the exercises currently in use today for cybersecurity education has inherently a high cost for development and deployment. The main cause of this high cost can be attributed to complexity of the current infrastructure and lack of automation to handle them. Additionally, there is neither an efficient means to store all the exercises nor there is an easy means to share them. The lack of a central point of control means that the instructors will have no means to track or assist the progress of the students in the lab/classroom. Instructors will not be able to diagnose or rectify the issues remotely and will have to resort to fixing them individually. In this process, instructors lose precious class/lab hours in troubleshooting system/configuration issues rather than training or teaching the

core contents of that particular class/lab. To make effective use of the time and resources spend on cybersecurity education, a simple, automated and streamlined platform needs to be introduced.

1.1.2 Goals

Primary goal of this work is to build a platform that addresses the needs of cyber security education today by providing the following:

a) a system for building and maintaining exercises as lightweight images, with high degree of automation;

b) a classroom management environment that provides centralized control over the infrastructure with ability to monitor and modify the status of supporting services and exercises in the student system;

c) a repository to share and store the images, with a simple privilege based access control;

d) a streamlined GUI to abstract away the implementation of the underlying infrastructure, and obviate the need for user to explicitly manage it.

Chapter 2 Related Work

Many works been done in an effort to make cyber security education easy to develop, access and deliver. For example, the Emulab facility/software system is such an effort, where a network testbed provided researchers with a wide range of networking environments for their testing purposes [6]. These testbeds allowed controllable, predictable, and repeatable environment with an operating system of choice to be run with root level access permission. Another example is USC/ISI's DETERLab project, a security and education enhanced version of the Emulab funded by National Science Foundation and Department of Homeland Security [7]. Most of the simulated environment like these relied on physical systems for their implementation and therefore required a lot of resources and often not portable.

There have been many works to address this portability issue, even before virtual machines, with works like Tele-Lab IT Security, a training system developed in 2006 to make interactive and portable security exercises by incorporating lab environment onto a bootable CD [8]. This solution, although secure and repeatable, obviously did not address every issue like exercise creation and updates, hardware requirements, exercise flexibility, efficient delivery mechanism etc. As the computer technologies started moving away from optical discs storage devices to solid state devices, these solutions became obsolete.

The idea of creating and delivering environments as bootable images saw better use with the advent of Virtual Machine. As the VMs became popular, there was an increasing trend among several universities to favor virtual machine labs over physical labs to provide education and training environments. For example, North Carolina State University shifted to a Virtual Computing Lab (VCL} infrastructure, in 2010, making redundant almost half of its computer labs [9]. Their virtual labs were made accessible online and enabled the students to reserve a computer with desired set of applications/environments to be booted from a VM image.

Academic establishments like SANS Institute added on top of such virtualized environments to provide small scale training and assessment for their in-house forensic team with security exercises. The exercises were based on tailored scenarios setup by an instructor and packaged as VM images for easy sharing and distribution [10]. The Advanced System Security Education Research and Training (ASSERT) Lab at UAF also worked on a similar infrastructure using VMware to host lab environment for large number of students. The VM images are stored on a fiber channel SAN to

facilitate multiple access, locally as well as remotely [11]. Meanwhile, cyber security education and training initiatives like National Initiative for Cybersecurity Education (NICE) have started encouraging the use of virtualized materials to support their curricula and competitions. The movement was in an effort to increase the system scalability, provide content availability of the learning/training materials [12].

However, the setup, maintenance and upkeep of such classical infrastructures using virtual machines demands costly systems, technical expertise, and with image size going up to 10-15 GBs requires large amount of storage space. There were many efforts for learning platforms that enabled instructors to create and deliver web-based or online hands-on training. SWEET (Secure WEb dEvelopment Teaching) platform developed by Georgetown University's Center for Strategic and International Studies is such an example [13]. University of New Mexico's virtual laboratory (VLAB) extends on this concept with VMware Lab Manager and providing a web based portal to access and run the materials hosted in their laboratory [14]. With the help of some terminal access applications like Remote Desktop students can access the allocated virtual machine. However, such platform does no address our goals completely, mainly because of their resource usage, high cost, lack of sharing and a centralized control over the classroom/exercises.

V-NetLab, developed in the Stony Brook University was an NSF funded project that aimed at a cost effective platform for supporting security exercises/labs [15]. Such projects are more in line with our goals work, but their reliance on VMs means they still carry some of the inherent issues with VM based platforms seen before. Moreover, not many seem to have addressed the issues with lack of automation, centralized control and sharing of exercises as a complete packaged solution for cybersecurity education.

Summary

In short, most current learning platforms in use today only address some of the issues and challenges faced by cyber security education. If not all but most approaches can be summarized as virtual labs created using some means of virtualization products either on the client side or server side. While these type of infrastructure help meet some of the challenges, their many limitations makes them less desirable as an effective, lightweight and flexible solution. Having a powerful cluster/group of computers to host the VMs and providing access remotely tackles some of those issue but at the expense of high infrastructure cost. There seems to have a very few attempts to address the storage issue created by the large size of the VMs. Sharing of the exercises is yet another unresolved issue.

While the client can control certain aspects of the exercises, there is limited centralized control over the hosted VMs or their underlying host system. This makes coordinating classroom/exercises remotely a challenging issue. Virtualization products like Citrix XenServer can provide a centralized control and resource management over the client system, however their performance cannot be guaranteed in a large networked environment. Hosting in cloud is an attractive option that helps to scale and boost performance on demand and handle multiple hosts in a virtualized environment with ease. However, the performance of such an architecture relies heavily on the bandwidth available at the client side. It can be inferred from analyzing the current approaches that, there has been very little effort for automating and streamlining the development, deployment and sharing of learning materials in a cyber security classroom/lab environment.

Chapter 3 Background

With the current infrastructures, studies and simulations of real world scenarios for cybersecurity education have become a logistical challenge. These challenges are mostly because of the lack of skilled professionals and overall difficulty in managing large number of physical or virtual systems involved with current infrastructures. Significant portion of that difficulty can be attributed to the manual work or lack of proper automation in managing exercises and systems in these infrastructures. Additionally, as the class size increases, the lack a centralized monitoring and control facility make the task of managing these infrastructure even more complicated. These limitations of technologies used in current infrastructures can be considered as an opportunity for exploring alternative solutions.

One emerging technology capable enough to address most deficiencies faced by cyber security infrastructure is container virtualization. With the help of container management tools like Docker, and automation tools like Ansible, an infrastructure with fine grain control can be developed. A cybersecurity learning platform supported by containers can ensure that exercises are easy to create and deploy, with reliability and consistency, irrespective of their deployment environment. Even though there has not been much application of containers outside the IT industry, there is a lot of potential for them in the education field. Containers in the recent years have gained a lot of popularity as a lightweight virtualization mechanism among developers and several mainstream companies, especially in the field of cloud computing. However, less than half a decade ago containers were still in their infancy and was considered a nascent technology.

3.1 A brief history of Containers

The origin of container or the idea of an isolated environment for process execution can be traced back to jails, one of the tools used to enhance the security in FreeBSD 4.0 system [16]. However, the most traditional method for restricting a process was done using chroot system call, introduced in Unix system in 1979. Several years later, chroot was introduced in Free BSD v2.0 and changed the way a process viewed the file system [17]. It was made possible for processes to view any directory as a root directory. In effect, a process created inside the chroot environment could only access files and folders within that directory and not outside them. Jails, extended on the concepts of chroot and allowed to create an isolated environment for the purpose of virtualization.

In 2001, the VServer project introduced virtualization into the Linux systems, and utilized kernel level isolation to create multiple virtual Linux servers in a single host [18]. These servers had high degree of independence and offered enhanced security along with efficient resource management. These factors influenced few ISPs to consider containers for providing online virtual private servers, however the need to patch and rebuild the kernel, made their widespread adoption less likely.

In 2006, Paul Menage's effort to utilize host operating system's kernel for virtualization helped set the momentum for incorporating containerization in Linux systems [19]. His concept of generic containers was expanded further by Google with the creation of control groups or cgroups for managing resource dynamically, among groups of user defined processes, with fine-grained control. Cgroups was adapted into the Linux kernel (version 2.6.24) in 2008 and with the introduction of user namespaces into the Linux system (version 3.8), it was possible to wrap a set of processes, assign users to those processes, and grant root privileges to those users inside their isolated environment.

Utilizing the concepts of chroot, cgroups and namespaces, the first out-of-the box containerization was introduced in Linux systems in 2013 as Linux Containers or LXC [20]. The user-space tooling provided by LXC allowed for easy deployment and management of Containers. As a support for LXC, the Docker project was started in 2013 to provide assistance on building and storing single application containers as images. In order to standardize the containers and their image format, the Open Container initiative was created under Linux foundation. In the effort to prevent fragmentation and promote open standards, Docker made codes to the project open source in June 2015. Docker, has evolved a lot since then, and now it has become the most popular container management tool for automating the creation and deployment of container.

3.2 Containers

Containers are a strong candidate as a platform for creating exercises to support cybersecurity education, chiefly because of its light weight nature. They are lightweight because, unlike traditional VMs, they do not carry the size or "weight" of an entire operating system. Only the necessary components required to execute the application or service are maintained and rest of the operating system is decoupled from the container. Instead of using a hypervisor to provide for the operating system needs of a containers, a daemon was used to share the host operating system kernel. This kernel sharing vastly reduced the container size and increased their portability between different systems and platforms.

To help understand how containers function, architecture of containers are detailed in Figure 1 [21]. The key component that enables the container to share the kernel with the host OS is the container engine. The container engine sits above the host operating and physical hardware of a host. The container engine acts like a virtual layer between the kernel and the container hosted in the system and creates an isolated execution environment for execution. This isolation is achieved with the help of chroot, cgroups and namespaces. Container engine not only help share the operating kernel and resources between the host and a container, but also between two containers inside the same host. The binaries and other library files required by a containers are created on top of the container engine and shared between two containers of the same type.
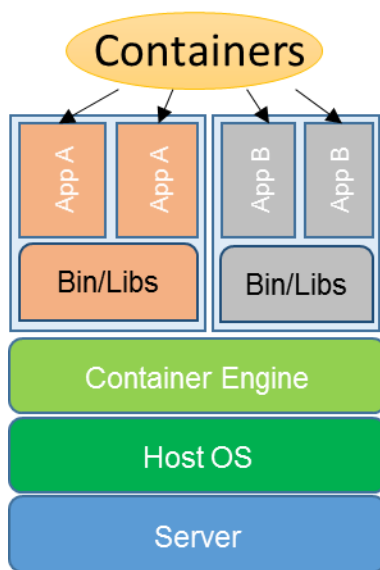
Figure 1 Container Architecture

One caveat of abstracting the OS away from the application is that containers cannot function on their own and require a Linux host to run. At the same time, this abstraction makes the containers lightweight and portable. If the host provided a container runtime environment, it was possible for the containers to be easily moved and executed in any Linux OS [22]. Containers could bring several other advantages to support a cybersecurity learning infrastructure, for example, minimal resource consumption, in terms of CPU and memory usage, capability to run multiple lightweight container instances inside a single hosts etc. This makes it possible to utilize the lab/classroom resources efficiently and therefore eliminate the need for powerful systems or other expensive hardware to support cybersecurity learning.

All containers run in isolated environments unless specified, and therefore prevent the containers from compromising the security of other containers. This layer of security is ideal in cybersecurity education because of vulnerabilities present in some of the exercises. Unlike most VMs, containers do not emulate a hardware layer and can execute at near bare-metal speeds. This execution speed makes it possible for containers hosting the cybersecurity exercises to get started in less than a second. Such speeds would be near impossible with most solutions based on traditional VMs or physical workstations. Another advantage in the use of containers is their low management overhead, especially when updating or patching the OS, fixing bugs in the kernel, etc. There is no need to update the containers separately, as updates in the host OS are automatically reflected in its hosted containers. This helps admins keep the infrastructure secure and updated without disturbing the execution environment of the exercises.

3.1.2 Limitations of Containers

There are minor limitations with containers that conflict with the goals proposed in our work for cybersecurity learning infrastructure. Linux containers, though actively developed, are not well documented beyond the Ubuntu platform, and they lack proper cross distribution documentation. This deficiency restricts the development of containers and their migration to other Linux distributions. As a result, portability of the exercise created using Linux containers is restricted. The lack of proper documentation and presence of outdated or obscure information online will make it harder for instructors and admins involved in creating cyber security exercises to use containers effectively. Most of these limitations can be overcome with the help of a container management technology like Docker.

3.3 Docker

Docker is a container management tool that help automate the deployment of applications in software containers and makes it easier to manage containers in a host system. Docker helped to build, deploy and manage containers by creating a new and standardized container format that was more suitable for automated build commands. As a result, creating containers were much easier, reproducible, and capable of being automated with the help of scripts. With Docker, it was possible for the key components of a distributed application to run on separate containers. In cybersecurity education, this allows the possibility of specific component in an exercise to be updated without disturbing other components.

Like LXC, containers in Docker is also very portable because all the low level system resource dependencies of the application is eliminated from the container and is shared with the host. It ensures that the containers are small is size, consume less resources and environment required for exercises can be provided, regardless of the host OS on which the exercises are run on. These features makes Docker an ideal platform for making exercises to support the dynamic requirements in cybersecurity education.

3.2.1 Docker Architecture
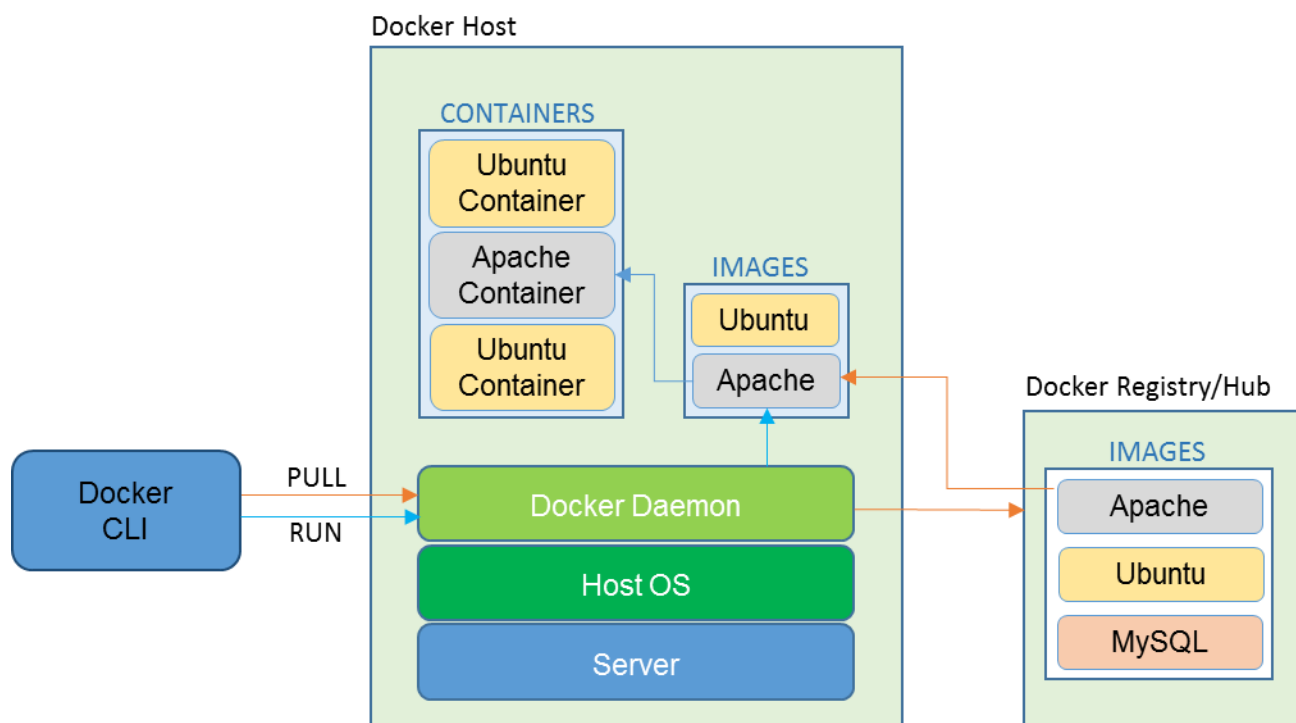
Figure 2 Docker Architecture

As shown in the Fig.2, there are three main components to Docker's architecture – Docker Daemon, Docker CLI and a REST API [23]. Docker daemon handles all the tasks for building, running and sharing Docker container, while the Docker CLI provides means for interacting with the daemon. The daemon is responsible for creating and managing the Docker objects like containers, images, networking, etc. The CLI is what the users or any other program/scripts can utilize to interact with the Docker containers and images inside the host system. The client CLI uses a REST API to accept commands, arguments and configuration files to communicate with the Docker daemon. The CLI and Daemon can be run on the same system, however it is possible for the clients to connect to the

Daemon running on a remote machine. It is also possible for a single Docker client to interact with multiple Docker daemons running on different systems.

Docker containers are executable instances of a Docker image that are preconfigured to provide a certain service, application or an environment. The CLI allows the containers in Docker to take additional parameter as arguments like networking information or other environment variables. This allowed Docker containers to be run in their isolated environment but at the same time open ports for exposing them to the outside world.

Docker images are template files used to create Docker container exactly how they were configured, on to any host machine. Images in Docker are made up of several read-only layers, so any changes to the running container does not affect the base container image. Changes made to containers forms a new layer on top of the base image layers are therefore lost when they are exited and removed. This allows exercises created as Docker images to ensure their original environment upon restarting. To save changes made to a running container, they have to be saved as a new Docker image. All saved or Docker image resides locally and is accessible only from that particular host system.

An additional feature that makes Docker attractive for cybersecurity education is repository, which is essentially a library to save Docker images outside the local system. There are two main types of repositories in Docker – a public repository hosted by Docker called the Docker Hub, and private registries that can be hosted by third parties. With the help of repositories, it was possible to save or download Docker images to or from a centralized location. This ensures that, cybersecurity exercises can be developed and shared easily among any Linux system configured with Docker.

3.2.2 Creating Docker Images

In most traditional approaches, all the installation and configuration required for creating exercises are done manually. Docker helps to automate this process using the Dockerfile. It is a text file containing a list of ordered Docker commands which, upon execution, assemble the desired container as a Docker image. Dockerfiles contain information regarding the base image to be used, files to be copied, configuration changes, ports to be opened and linked to the host machine, startup commands, etc. Using a combination of Docker keywords and system commands, a Dockerfile can be used to configure and containerize any environment or application.

Figure 3 shows how a sample Dockerfile looks like. Each line in a Dockerfile starts with a keyword that instructs the build command what to do with the instructions following that keyword. In this example, Dockerfile instructs Docker to create a container using yaronr/debian-wheezy as a base image, with the keyword FROM. Keyword RUN is used to execute Linux commands - update OS, install applications, make directories inside the container being built, etc. Containers by definition are run in isolated environments and ports need to be exposed to grant access from outside the container; this is done using the EXPOSE command. The COPY command allows a folder or file from the current directory to be copied/created into a destination inside the image being created. The ENTRYPOINT command allows Dockerfile to specify the default executable for the image. In this example, a script is given as the entry point, so when the container is started the script will be executed, allowing the container to be run as an executable.

.

```
#Step1 - preparing the machine
FROM yaronr/debian-wheezy
RUN apt-get update && apt-get install -y apache2 netcat
EXPOSE 80
EXPOSE 3333
RUN echo "@STEP@ 1 - Vulnerable apache server"

#Step2 - transfer some script
COPY script.sh /tmp/script.sh
RUN echo "@STEP@ 2 - Transfer scripts to compromise the server"

#Step3 - run the script
ENTRYPOINT ["bash", "/tmp/script.sh"]
RUN echo "@STEP@ 3 - Execute the script on the server"
```

Figure 3 Sample Dockerfile

After the Dockerfile is created, it can be executed using the Docker build command. Docker first checks if the base image is available and automatically downloads it, if not available, from a private or public repository. Execution of rest of the commands in Dockerfile begins on top of this base image and a new image layer is formed with each line of execution from the Dockerfile. This new image layer is given a unique ID and the process continues, adding new image layers till the execution completes.

The ID of the image created after execution of the last line in Dockerfile is given a name and tag provided by the user along with the build command. All the intermediate image layers created, starting from the base image, are saved with the final image. Figure, 4 shows the intermediate images created by executing the Dockerfile in Figure 3. These intermediate containers can be reused by Docker when a new image is being created that has the exactly the same or similar Dockerfile. In such cases, Docker keeps all the intermediate images of the existing image and then used that as a base to execute the rest of the command in the Dockerfile to form the final image.

IMAGE	CREATED	CREATED	BY SIZE	COMMENT
c76c195856a0	6 days ago	/bin/sh -c echo "@STEP@ 3 - Execute the scrip	0 B	
7f00f61301be	6 days ago	/bin/sh -c #(nop) ENTRYPOINT &{["/bin/sh" "-c	0 B	
b6db75e128d3	6 days ago	/bin/sh -c echo "@STEP@ 2 - Transfer scripts	0 B	
24fc9989e7cc	6 days ago	/bin/sh -c #(nop) COPY file:2ab4aaa5952a6f15d	38 B	
934ac8341bb3	6 days ago	/bin/sh -c echo "@STEP@ 1 - Vulnerable apache	0 B	
07e545169e4b	9 days ago	/bin/sh -c #(nop) EXPOSE 3333/tcp	0 B	
412aadc1cc28	9 days ago	/bin/sh -c #(nop) EXPOSE 80/tcp	0 B	
4073be09be58	9 days ago	/bin/sh -c apt-get update && apt-get install	53.62 MB	
bd81ef1880d1	5 months ago	/bin/sh -c DEBIAN_FRONTEND=nonintera	46.12 MB	
bfb09bfa338a	5 months ago	/bin/sh -c (echo "deb http://cdn.debian.net/d	276 B	
9282f3258556	5 months ago	/bin/sh -c #(nop) MAINTAINER yaronr	0 B	
8c9ab014af26	13 months ago	/bin/sh -c #(nop) CMD ["/bin/bash"]	0 B	
917ec8b91822	13 months ago	/bin/sh -c #(nop) ADD file:41fbf9f100beb8	84.89 MB	

Figure 4 Intermediate images created during execution of Dockerfile

The intermediate images created are also interactable. For example, in the figure 4 the image ID *4073be09be58* if ran, will give a container that has the apt-get updates and softwares installed on top of the debian-wheezy image. Similarly, the image *412aadc1cc28* will give a container that has its port 80 exposed in addition to the updates and softwares installed on top of the base image. This is particularly useful in creating exercises for cybersecurity because it allows exercises to be brought to a desired stage by executing a particular intermediate image.

3.2.3 Docker Registry

Images created using Dockerfile reside in local systems for the lifetime of the system or until it is deleted. Persistent storage and management of container images is done using Docker's own stateless and highly scalable server side application called Docker Registry. There are several other cloud-based registry alternatives like Amazon EC2 Container Registry, Google Container Registry,

Docker Hub, etc that can offer the same features as a private Docker registry does but not without issues. All of the above cloud-based solutions can create performance bottlenecks because of their dependencies on hardware resources, local network bandwidth, Internet access speeds, cloud service providers, and so on. Comparatively, managing a private registry is lot easier and provides the most efficient solution for saving cybersecurity exercises build in Docker.

A private Docker registry has no external dependencies, functions on LAN and WAN and provides full control over the stored images. In a classroom or lab environment where simultaneous access from multiple systems can be expected, a private repository in the local network provides superior performance. The private registry's performance will surpass any cloud-based registry services, even if they were accessed using an optical fiber cable. Additionally, Docker registry is available as a runnable container and can be easily downloaded, configured and deployed for use in few minutes.

3.2.4 Limitations of Docker

Docker has several limitations when handling containers in a classroom or other large size networked environment. For example, sharing of container is another difficult task, since the only two ways are either to import configured container images from the repository, or acquire the Dockerfile and run it locally to create the container image. In a classroom, neither option is ideal as they involve coordinating students to execute commands to compile the Dockerfile or login and download exercises from some private registry.

Besides, Docker can only manage containers in the system in which it is hosted and has no easy means to provide a centralized point to monitor and control the containers in a network. This is problematic as the instructors will not be able to control or manage exercise on student systems. Users are expected to have some level of mastery over the Docker commands to use or manage the exercises. Therefore, like traditional systems, installing, configuring and managing Docker requires lot of manual effort and time. Tools like Ansible can be used to help to overcome these limitations by providing a platform to effectively manage all the hosts and their respective containers in the infrastructure.

3.4 Ansible

Ansible is a configuration management and provisioning tool for orchestrating tasks across multiple systems [24]. Like most configuration management software, Ansible has two groups of machines in its architecture – a control machine and nodes. All orchestration commands are provided from the

control machine to the nodes. Installation is only required in the control machine and no custom agents are required to managed the node machines. Ansible leverages the use of SSH daemon in the node machines. This makes it fairly easy to setup the infrastructure and add nodes to it without much administrative tasks.

Ansible can automate the tasks efficiently by getting context before running commands or scripts in the nodes. Ansible uses these contexts or facts to check the current state of the containers and only execute those tasks necessary to bring the containers to their desired state. This benefits the infrastructure by eliminating the need for most edge cases which otherwise would have required long, complicated scripts. One of the other benefits of using Ansible is its ease of use. Ansible has very simple sets of commands and formats that are not only easy to use but also simple to learn. Infrastructure can be easily configured and managed in a JSON format, using these commands. Since these commands are executed and results are obtained in command line, they can easily be automated using some scripting language. In other words, Ansible can help admins and instructors to easily handle student systems as well as create and manage exercises developed using Docker, effortlessly.

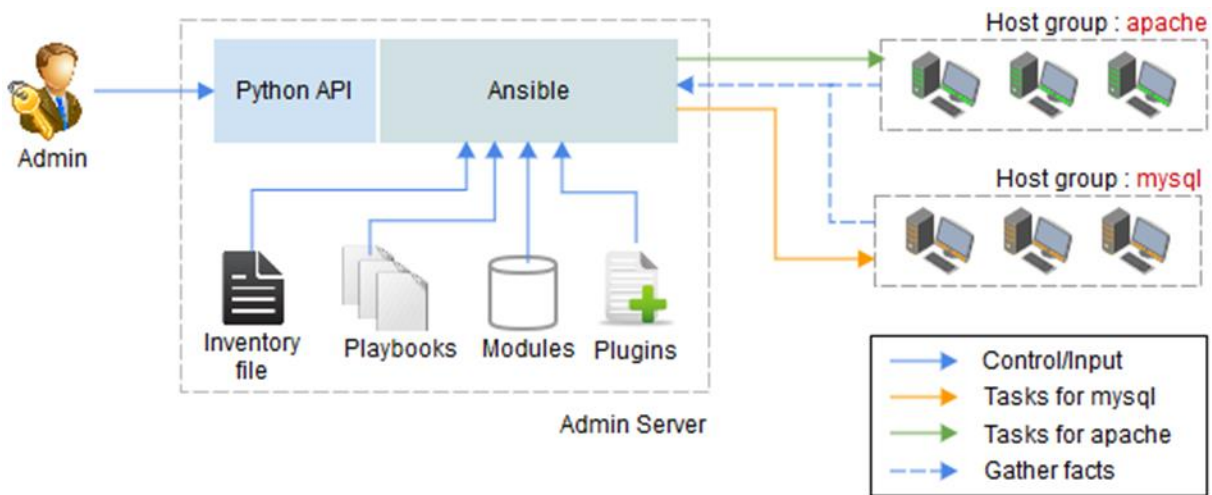## 3.3.1 Ansible architecture

Figure 5 Ansible architecture

Overview of how Ansible orchestrate tasks to different groups is shown in Fig.5. Admins and instructors can interact with Ansible using its python API to issue commands and controls. Playbooks begins executing by selecting the target hosts listed inside the inventory file. Facts or important

information regarding various attributes of the target hosts are gathered prior to the execution of tasks inside the playbook. Ansible then loads all the required files, modules and plugins for supporting the execution from the admin server. Once the host information and modules are loaded, Ansible executes all the tasks in parallel, one after the other, in the target hosts. While executing, the success or failure of the task for each host is displayed along with a short summary of the execution at the end.

3.3.2 Managing remote hosts

All nodes that needs to be managed, must be configured with password-less ssh login or access from the Ansible server. Ansible maintains all the nodes that it has access to inside an inventory file called hosts. Ansible allows fine grain control over the infrastructure by grouping nodes into groups and other information specific to a particular host. Besides the IP address, information may include passwords, usernames, host groups, etc., required by Ansible to manage these nodes. These information allows admin and instructors to execute commands or change state of client nodes selectively as well as collectively. The Fig.6 shows a sample inventory host file with two host groups, apache and mysql. The host groups are mentioned inside square braces and all the hosts belonging to a particular group are listed directly under them line by line. The nodes can be mentioned either by its IP address or by a hostname that can be resolved by a DNS server.
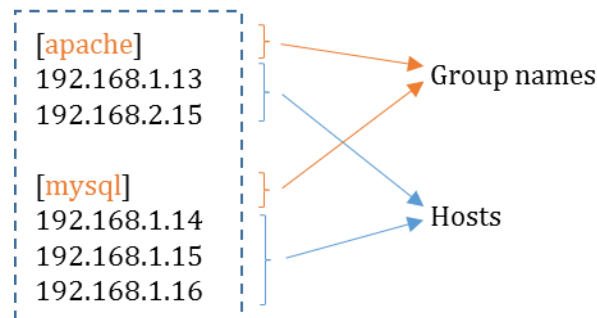

Figure 6 Sample inventory host file

3.3.2 Ansible modules, plays and Playbooks

Ansible can issue various ad-hoc commands to the remote nodes for quick information retrieval and task execution. For managing more complicated task, like installing software, configuring and setting up environments, Ansible make use of files called Playbooks. Playbooks is a simple human readable

text file that uses YAML format to structure and define the different operations needed to be performed in the nodes.

Every playbooks is composed of one or more Plays in a list. The main purpose of the plays are to map a set of well-defined roles to one or a group of hosts. At a very basic level, these roles are represented as tasks and involves calls to one or more Ansible modules. A library of Ansible modules help control and manage various system resources, services, files, etc. of a remote host. These modules can be either directly executed on to the remote hosts as ad-hoc commands or used inside a playbook. Ansible modules can be used in combination inside a play to achieve a specific task inside the playbook. With a combination of plays, a playbook can be used to effectively orchestrate a multi-machine deployment and task execution over a group of systems.
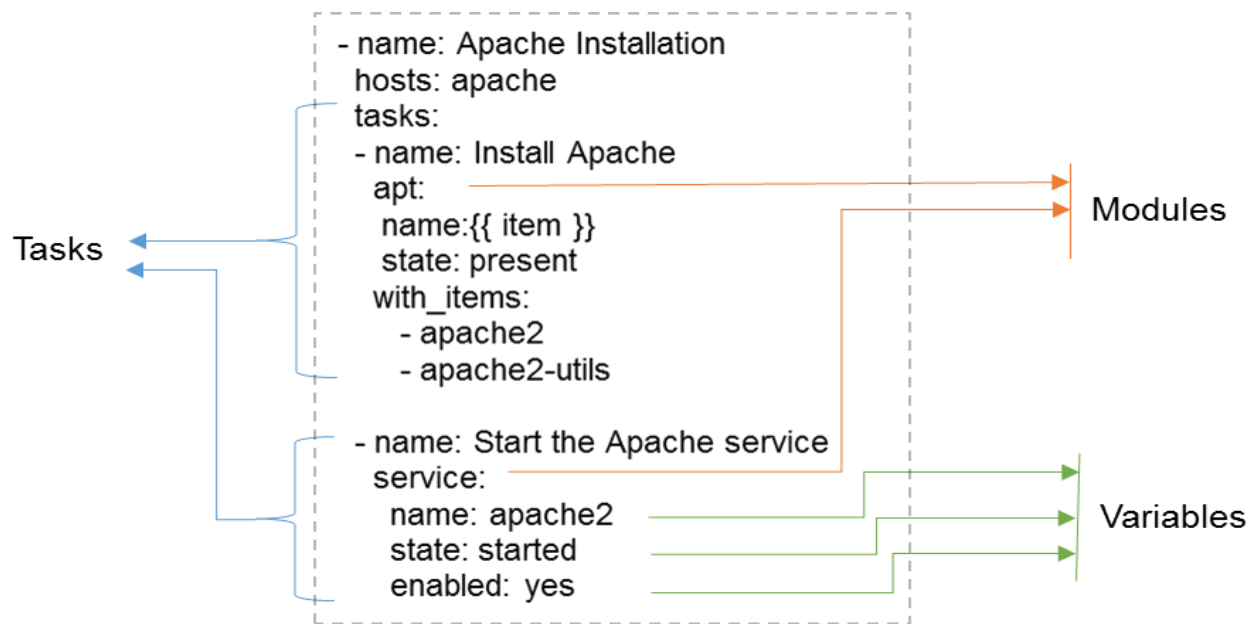

Figure 7 Sample Playbook for installing apache

Plays inside the Playbooks are idempotent and Ansible checks the state of the client prior to execution of the tasks. If a task is designed to bring a change to the system like installing a package or creating a file, then the state represents the status of that package or file inside that system. Tasks are executed only if a host is found to be not in its desired state. For example, if a task is set to install a package, then Ansible checks if that package has already been installed. If it is found to be installed in the system then no installation is made. In case if it is not installed or the installed version of that package is different, then that package is installed or updated accordingly.

30

A sample playbook for installing and starting an apache service inside a group of remote hosts is shown in Figure 7. There tasks defined inside the playbook will be executed on all the hosts listed under the host group 'apache' in the inventory file. There are two tasks defined under the tasks section, one to install software packages and other to start apache service. The first task utilizes the 'apt' module to install a list of given packages after checking if that package is present inside the host. After the packages are installed, the status of apache service is checked and started if has not yet been started. Additional parameters can be set when the services are started, like enabled in this case ensures that the service remain started even after a reboot.

Playbooks are highly portable and can be executed from any system configured to use Ansible. Unlike Dockerfile, an exercise created using Playbook would not limited to just Docker container, but instead, can be recreated in other platforms like Vagrant, bare metal VMs and cloud instances. Ansible can be effectively used to automate container management as well as the systems and environment hosting them. When playbooks get bigger, distinct individual tasks can be segmented as roles and saved as separate files; roles are like building blocks and can be reused in other Playbooks.

Ansible can eliminate much of the complexity involved with container management and system administration tasks, away from its users. However, there are still a few limitation that hinders this tool from being really user friendly and easy to use. Both Ansible and Docker, helps to manage its various components and infrastructure using their respective set of proprietary commands and configuration files. From an administrator or user point of view, they have to learn these configurations, commands and understand how to interpret their outputs in-order to be able to effectively make use of Ansible. These limitations add a significant learning curve and management overhead to the deployment and execution of exercises in infrastructure. This management overhead can be relaxed with the help of backend scripts to automate the coordination and management procedures. A webapp can be employed to abstract away most of the background mechanism and provide a GUI to interact with the exercises and orchestrate the tasks of Docker and Ansible.

3.5 Python

3.5.1 WebApp

Python is one of the most widely used languages for developing web applications and Flask is the preferred choice for any python based webapp that requires only the core components of a web development framework. Flask is a micro-framework with plenty of options for extensions that support web application development. Since Python comes preinstalled on most Linux distributions,

and is available as an installable package on all other OS, webapp developed in Flask requires minimal installation to get started. Moreover, Flask is lightweight, very well documented and supported by an active developer community. These advantages make Python Flask an ideal choice for developing the interface required for supporting the cybersecurity infrastructure.

3.5.1 Backend Scripts

One of the biggest setback with Ansible is that the hosts file need to be manually updated. It is a problem, especially in a classroom environment because the instructors may not have prior information about the IP addresses of all the systems in the network. Instructors will have to manually coordinate all the machine's IP addresses into the hosts file prior to execution. This tasks becomes even more difficult if the machine's IP is dynamically allocated over the network by a DHCP server. As the lease time expires for the granted IP address, the DHCP server will grant a new IP for the machine. Since there is no means for instructors to keep track of which machine's IP has been changed, he/she will have to manually inspect each machine in the lab to coordinate the IPs in the hosts file. Configuring the machines with a DNS address might seem like an immediate workaround, however it adds dependency to the infrastructure by having to run and maintain a DNS server. Also it becomes difficult for configuring new systems and adding them on the fly, thereby reducing the scalability of the infrastructure.

Backend scripts developed in python can take care of automating the task of adding hosts automatically by broadcasting the service and having the clients added when they respond to the broadcast. Other supporting scripts can help with the task of getting vital information regarding the hosted containers, process the information and have them stored in a database. The stored information from the database can then be accessed by the webapp to present to the admin/instructor in an easy to understand format. The controls requested by the client using the webapp can be processed, equivalent commands generated and executed using the backend scripts to get the desired outcome.

Chapter 4 Architecture

In summary, to address the deficiencies in the current learning infrastructure and achieve goals set for the required cybersecurity education platform requires a combination of three main components – Docker, Ansible and Python. Docker facilitates a simple platform to create, store, share and run exercises in the form of lightweight Docker containers. Ansible automates and coordinates task involved with monitoring and controlling remote systems and exercises. Python Flask provides the GUI for admins and users to interact with the infrastructure and establish control without having to go through the complexity of the background mechanisms. Based on the functionalities for admins and users supported by these components, the infrastructure can be divided conceptually into two – server side and client side.

The server side contains components that supports functionality required by admins and instructors to manage the infrastructure. The server side has four main function in this infrastructure

a) broadcast it's presence over the network offering a centralized control service;
b) authenticate and accept incoming connection from clients requesting the service and populate/update the database with information regarding its logged in user, IP address, hosted exercises etc;
c) create and issue control commands, supporting script and configuration files to all the connected clients and other connected systems;
d) create or push exercises into the repository, as well as issue commands to download and/or execute exercises or specific components within an exercise inside the clients

Client side contains components that support the clients to manage their respective system and exercises. The client side has the following functions

a) provide a GUI based interaction to execute/download the exercises and manage its supporting environment
b) detect the presence of an admin server in the network, attach itself to the centralized control service, and provide periodic information regarding the system's status to the server
c) receive control instructions from the server and execute them to bring the desired changes to effect

4.1 Design

As shown in the Figure 8, a server side would typically be comprised of two server – Docker registry server and an admin server. Apart from creating exercises, the admin server helps monitor and control exercises in the client side. Meanwhile, the registry server provides a centralized storage facility for the exercises created or uploaded by the admin server. The access to the registry is handled by an Nginx server configured as proxy between the users and the Registry server. Users get to access and download exercises from the registry only after a valid authentication to the Nginx proxy server.
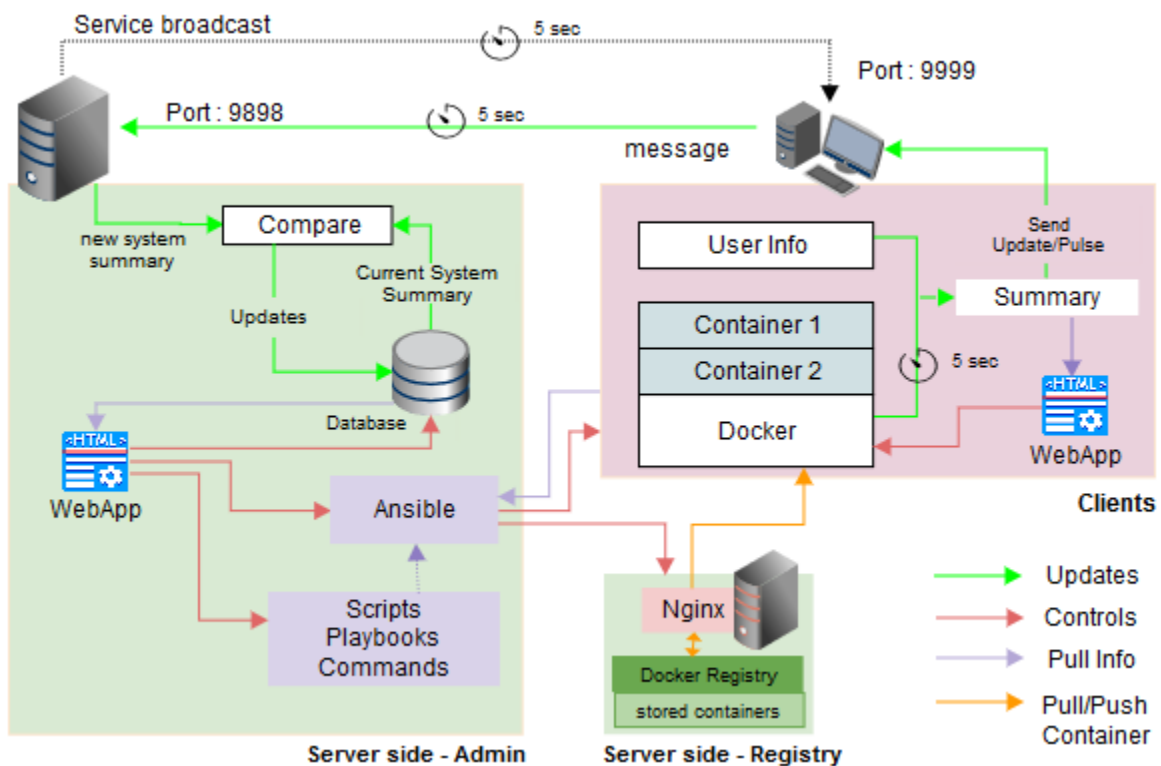

Figure 8 Design supporting Cybersecurity Education using Ansible, Docker and Python

Once the exercises and their corresponding index and description files are downloaded into the client machine, they will be made available to execute in the client side GUI. As the client side machines download and execute the exercises, a background script running in the client side scans the system every 5 seconds to generate a summary. This summary is then compared to a previously stored summary. If a change is found to have occurred between the two, the client sends the new summary to port 9898 of the admin server and updates the stored summary. If no changes was found, the client

34

will send out a pulse message to the admin server on the same port, notifying of its presence in the network.

In the case if a client loses network connectivity to the admin server, the newly generated summary is send upon reconnecting. A client can lose network connectivity to the admin server because of networking issues either at the client side or server side. In either case, the client shifts from sending the status messages to a listening mode, where it waits indefinitely for the server to become active in the network. When a server connects to the network, it sends UDP broadcast messages advertising its presence and details about the service it is hosting. The client listening in the network picks up information from the broadcast, establishes a new connection to the server and continues to transmit the updates.

The server side maintains a database regarding the status of exercises and other user or system related information for all connected clients. When a summary is received from the client, it is compared against the stored information from the database for that particular client. After comparing the two, necessary updates are made to rows concerning that particular client in the database. While these updates happen in the background, the server side webapp periodically updates the information provided in the GUI with information from the database, every 5 seconds. Admins or instructors can issue controls to the clients, based on the displayed information, to bring the exercises or components of the exercises to a desired state. These controls from the webapp are processed by the backend python program to generate equivalent commands with necessary files, scripts and playbooks required to support the execution. These commands are executed in the required clients, using Ansible, and changes made will be reflected in the serverside with the subsequent client side summary update.

4.1.1 Server Side

The monitor and control facility provided by the admin server in the network is achieved with the help of a combination of tools and scripts. Python scripts help with much of the background automation tasks and utilizes Ansible to orchestrate the tasks across different systems. The sections below discusses the key server side functionalities in more detail.

Creating and storing exercises

Consider a sample SQL Injection scenario which involves compromising a website by injecting SQL commands into the webpage URL. This exercises requires MySQL as backend database, Apache as

webserver, and PHP to link vulnerable webpages to the database. An admin may choose to create all the above inside a single container or as three separate but linked containers hosting PHP, Apache and SQL. Ideally, creating separate containers brings modularity to the exercise and provides greater control over the state of each component. The containers in the exercises can be either created using Dockerfile or downloaded from an online hub or registry, if available. In either case, to prevent ambiguity in the names of the stored or running containers, the admins or instructors have to ensure that a naming sequence is followed.

Naming sequence for Exercises and other supporting files

In order to maintain uniformity in the names, all exercises in this infrastructure, are referred internally as 'lessons'. Each exercise must have a unique name, which is a combination of the word 'lesson' and a lesson number. The lesson number is a unique number that is maintained by the admin or instructor and incremented every time with a new exercise. For example, in an empty registry, the first lesson would be stored with the name lesson1 and second lesson as lesson2 and so on.

An exercise can be composed of one or more Docker containers. The containers that constitutes an exercise are referred to as components. All components in an exercise must carry the lesson name of that exercise and their component number; the two must be separated by a forward slash character.

Syntax: *lesson_name/component_number*

Example: lesson1/comp1

The component number itself is combination of the word 'comp' and a number incremented with each exercise component. Considering the SQL injection exercise, the three components would have to be named as lesson1/comp1, lesson1/comp2 and lesson1/comp3 respectively. Figure 9 shows a pictorial representation of the naming the components of the exercise.
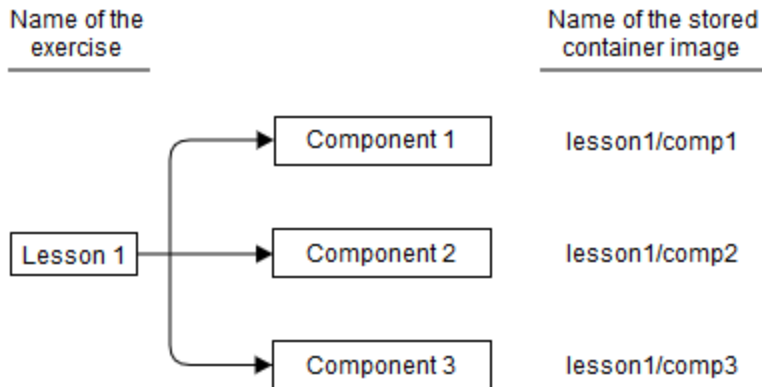
Figure 9 Naming of a sample exercise and its components

While it is easy for the admins and instructors to know what the components are or how they contribute to an exercises, it wouldn't be clear to anyone other than them. Docker doesn't have a neat way of providing these details and this lack of information is addressed with the help of two set of files.

The first is an inventory file named 'lessons.txt' that lists all exercises in the registry and their respective components. This file is common to all the exercises stored in the registry and must be updated every time an exercise or component is added or deleted from the registry. Each new exercise entered must start with the lesson name followed by component names of all the containers, which belongs to that particular exercise. Each exercise entry must be made in a single new line inside 'lessons' file, with the syntax below.

Syntax: *lesson_name component_name_1 component_name_2 ...*

Consider the same SQL injection exercise, with lesson name as lesson1 and its component names as lesson1/comp1, lesson1/comp2 and lesson1/comp3 respectively. In this case, the lessons file for lesson1 should be the following line*.*

Example: lesson1 lesson1/comp1 lesson1/comp2 lesson1/comp3

The second is a description file that exists for each exercise and its individual components. They hold information in plaintext regarding the title and description of that exercise or component. The description file for an exercise or component must be saved after their corresponding lesson or component name. For example, description about an exercise with lesson name as 'lesson1' must be stored inside a description file named 'lesson1', and that of a component named 'lesson1/comp1'

inside the description file 'lesson1-comp1' and so on. Note that component name has their forward slash replaced by a hyphen because, Linux's naming convention for files does not support forward slashes. Inside a description file, the following format must be followed

Syntax:

title_of_the_exercise

description_of_the_exercise

An admin needs to ensure that these files are updated whenever a new exercise is uploaded into the registry, or when an existing components or their information is modified.
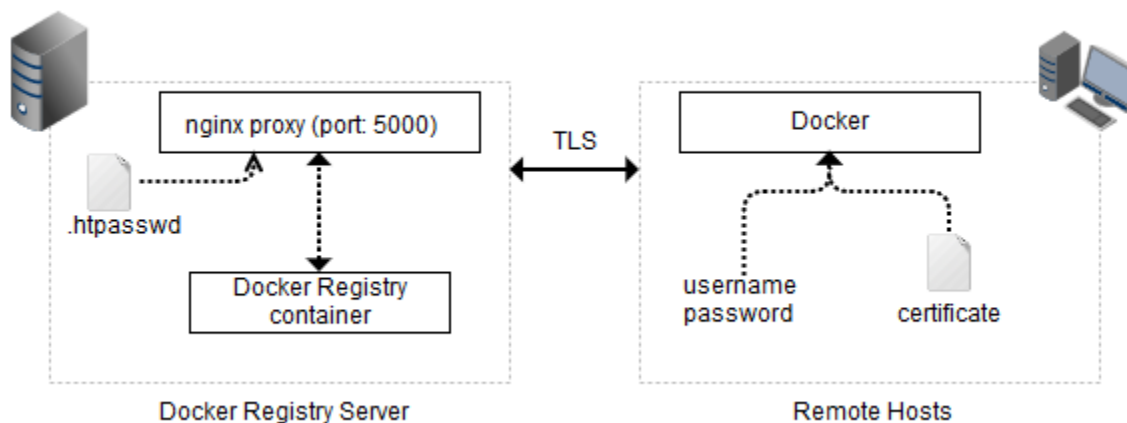
Access Control over the Registry

Figure 10 Authenticating a user to the Docker registry

Docker registry by default does not have any means to control over the access of its contents. So, an Nginx proxy server was configured to authenticate users and moderate all communication between the registry and the users. Nginx proxy protects end-to-end communication with the help of Transport Layer Security (TLS). Public key certificates of the server is used by the client to trust the authenticity of the server and the server in turn uses a simple username-password authentication mechanism to prove the authenticity of the users trying to access its contents. Every client requires a public key certificates of the Docker registry server and valid credentials to successfully communicate with the Docker registry. The access controls rules configured with Nginx ensures that non-members of the infrastructure cannot access or download exercises from the registry.

To access a container in the registry server, the client has to authenticate itself by passing a username-password combination, and public key certificates of the registry server. The Nginx proxy listens to port 5000, which is the default port used by Docker for communicating to the registry. For all incoming requests, the proxy server validates the access by authenticating the users with a simple htpasswd file. Once authenticated, all legit users can download exercises from the registry.

Monitoring and controlling the infrastructure

Webapp in the server side is tailored for fine grain monitor and control of remote systems and exercises inside them. Such control requests from the server side webapp is processed by the backend and equivalent ad-hoc commands are dynamically generated. Tasks that cannot be achieved through ad-hoc command can be achieved through script execution. Some commonly executed tasks have already been scripted and made available for use as templates within the webapp. As per the control requests, appropriate modifications are made to these templates and executed. When the complexity of the tasks increases, use of multiple commands or scripts becomes too complex and inefficient. In such cases, playbooks are the best alternative as it incorporate different Ansible modules and other scripts in a structured manner to achieve task in an efficient manner. Once the playbooks and commands have been generated, they are executed using Ansible on the desired hosts or host groups.

WebApp Directory structure

The server side web app has the directory structure as shown in the figure 11. The main python Flask executable file is immediately available inside the main app folder and it is the only executable file required to be run to bring the webapp online. This python file is responsible for all the backend task that orchestrate Docker and Ansible, links the webpages, execute script, summarize the result, etc. All the html webpages required by the webapp must be maintained inside the 'templates' folder. All static contents required by the webapp, like images, fonts, custom scripts, playbooks, css files and etc., must be stored inside the 'static' folder. All the css files and jquery scripts for the webpages are made available immediately inside the 'static' folder.
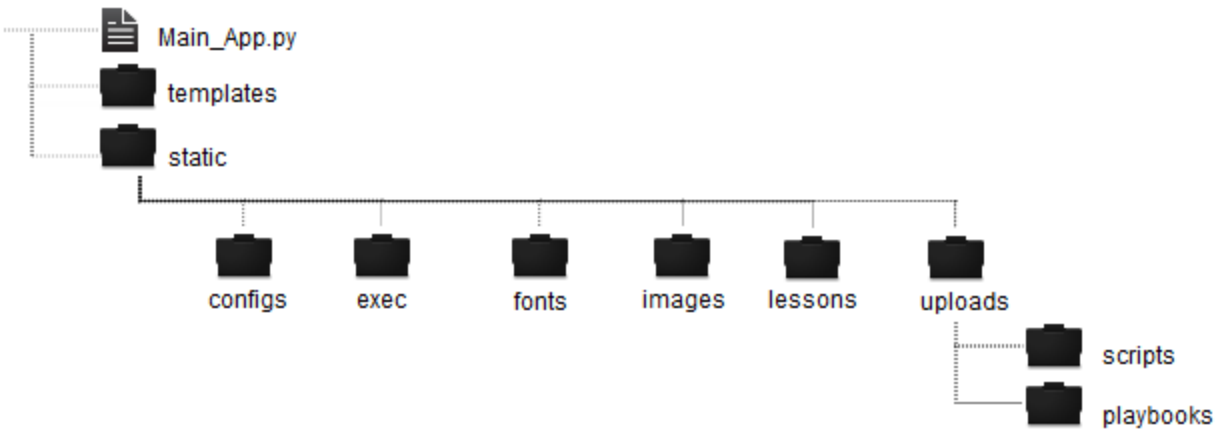
Figure 11 Webapp directory structure

There are six folders maintained inside the 'static' folder – configs, exec, fonts, images, lessons and uploads. All the configuration files, Ansible hosts files, certificates, lessons file and etc. must be store inside the configs folder. When a task is to be executed as a playbook, the webapp temporarily generates them into the 'exec' folder. All the custom scripts and playbook created or uploaded by the admins and instructors must be stored inside the 'uploads' folder. All scripts inside the 'uploads' folder are placed inside the 'scripts' folder and playbooks inside the 'playbooks' folder respectively. All the images and fonts used by the webapp are stored inside the folders, 'images' and 'fonts' respectively. The 'lessons' folder contains a list of files that carries information regarding the title and description of an exercises and its respective components.

Using the webapp

Consider a cybersecurity scenario which demonstrates the tools and techniques an attacker could use to compromise and gain access into a server running a vulnerable software or service. For the purpose of this scenario, the goals for the exercises was set based on a server with vulnerable version of apache as the target. First goal was to set to scan and identify vulnerable services in this server using some reconnaissance tool. Once a potentially vulnerable service was identified, the next goal was to compromise the server using a suitable exploit available for that particular vulnerability in the software. After a privileged access is obtained, final goal was to run some user transferred scripts to open a netcat connection on port 3333 and transfer the user-login information from the Linux passwd file.

Creating this scenario as an exercise requires careful planning from the part of admins or instructors. In this particular example, only a single component was used for hosting the vulnerable apache

service. The component in the exercise was divided into stages, with each stage representing a predefined state in the exercise. First stage simulated the vulnerable server prior to the attack, second stage simulated the server after it has been compromised and some script have been transferred to the server and finally, the third stage simulated a compromised server after the script has been executed to transfer the passwd file information through port 333. These stages help students and instructors to jump to a particular stage in the exercise, in case if the student is falling behind rest of the class. After the exercise and its stages have been structured, a Dockerfile was created to build the Docker container image representing the exercise. Figure 12 shows the Dockerfile for creating the image with different stages.

```
#Step1 - preparing the machine
FROM yaronr/debian-wheezy
RUN apt-get update && apt-get install -y apache2 netcat
EXPOSE 80
EXPOSE 3333
RUN echo "@STEP@ 1 - Vulnerable apache server"

#Step2 - transfer some script
COPY script.sh /tmp/script.sh
RUN echo "@STEP@ 2 - Transfer scripts to compromise the server"

#Step3 - run the script
ENTRYPOINT ["bash", "/tmp/script.sh"]
RUN echo "@STEP@ 3 - Execute the script on the server"
```

Figure 12 Exercise structured as steps within a Dockerfile

Once the Dockerfile is executed a Docker image will be created and for the sake of explanation, let this newly created image be saved as lesson4/comp1, following the required naming convention. As this image does not carry any information regarding what it is or what the purpose is, details about the title and description for the images needs to be added before they are regarded as components. The server side webapp has visibility over all the images, components and exercises available in the registry as well as those available locally. Details to an image are added using the webapp as shown in Figure 13. When the form is submitted, a local copy of the component description is made or updated in the server side before being uploaded to the registry server.

Figure 13 Adding details to a Docker image to create a component

After updating the descriptions for that component inside the registry, an exercise can be formed using the webapp by describing the names of the component that constitute the exercise. Additionally, a suitable title and a description for the exercise also need to be provided using the webapp. In this scenario, there was only one component, so a lesson or exercise can be created using the webapp as shown in figure 14. Descriptions for the exercise and components as well as the component images will be synced with the registry server upon form submission.

Figure 14 Creating and adding description to a lesson

Meanwhile, all the clients in the network that have access to the registry can download them or a serverside can issue controls to download them. Either way, all clients that responded to the service broadcast from the server will automatically send periodic updates to server regarding all their status information on exercises, every 5 seconds. Figure 15 shows the server side view of the processed information gathered from these client updates. This provides an admin or instructor a good overview of the client machines and the status of exercises inside them at near real time. An admin or instructor can then select an exercise of choice to view the status of all clients with respect to that exercise. Information includes if the exercise or components of that exercises have been downloaded, started, stopped etc. He/she can then issue controls to a client or group of clients to bring that particular exercise to the desired state.

Figure 15 Server side view of the exercises and status of clients

With the help of webapp, all the back end scripting for coordinating, monitoring and controlling the connected machines are encapsulated from the instructors, making the infrastructure simple and easy to use. All the metrics available at the server side dashboard allows an admin/instructor to have fine grain control over the connected clients and lessons inside them, with click of a button.

4.1.2 Client Side

Client side is designed to be easy to setup and simple to use. Besides Docker and Python Flask, no other installation is required in the client side and therefore reduces the time and effort required to setup client machines for using this infrastructure. Full control over the exercise is provided to the users without exposing them to the complicated background mechanism using a simple webapp. However, when inside a classroom environment supervised by an admin or instructor, the server side takes preferential control over the exercises in the client machines.

Similar to the server side webapp, a basic username-password authentication is enforced before granting access over the infrastructure to the users. The login credentials are stored in a MySQL database inside the Docker registry server and is checked for validity during each login attempt. All login attempts from the server side webapp is checked for account privileges to protect regular users from having access to admin controls. In the client side webapp, a successful login will additionally create a stub file that stores useful login information required by the admin system.

The backend of the webapp gathers information about all the available images and deployed containers inside the client machine. The description file available at the client side is then used by

44

the webapp to display these containers as exercises and components with relevant titles, description and status information. The client side webapp gives a very basic set of control over the exercises and its components using a simple GUI. When clients interact with the exercises or components using this GUI, equivalent commands are generated in the background for the desired outcome. Once these commands are executed, results are summarized and reflected in the webapp.

When an exercise is started, all the supporting components will also be started, in the required order, and bind to the host OS or other containers on ports configured by the admin when creating those exercises. Users can connect to any web services/app hosted inside these components directly from the host's web browser using the host IP. For any other services hosted in the containers that requires a command line interaction, a shell/terminal to those containers can be initiated using the webapp with the click of a button. The webapp thereby, effectively shield the users from most of the background mechanism of this learning infrastructure.

Establishing connection and sending updates

In addition to performing the task of organizing and controlling exercises at client side, the python scripts employed in the backend also takes care of maintaining connection with the server side and sending regular updates.
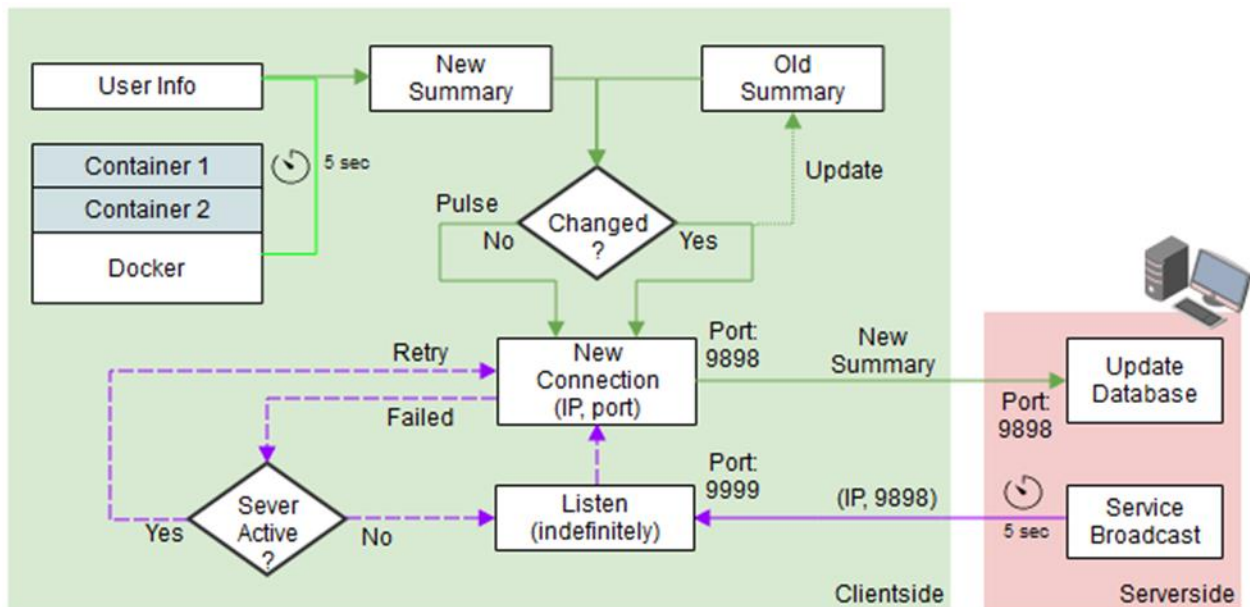

Figure 16 Updating client side details about exercises to server side

45

Figure 16 shows the overall control and message flow from client side to the server side. The background script checks for downloaded exercises and status of those exercises to create a summary every 5 seconds. This summary is checked with the summary that was last updated to the server and checks if any change has occurred. If a change is not found, then a pulse is sent to the server, to maintain its presence in the network. If a change has occurred, the new summary is send to the server after creating a connection to the server on port 9898. If it fails to establish a connection to the server, the backend script waits indefinitely listening on port 9999 for a service broadcast from the server. When a broadcast message from the server is obtained, notifying server side IP address and port to be connected, a new connection is established using those details. Updates are send to then send to the server side and continues this process every 5 seconds.

Using the webapp

Once the exercises are made available in the registry server, with appropriate authentication the client side webapp will be able to view them with their respective title and description. When an exercise is requested to be downloaded by the client, all the components of the exercise and its supporting description file will be automatically downloaded.
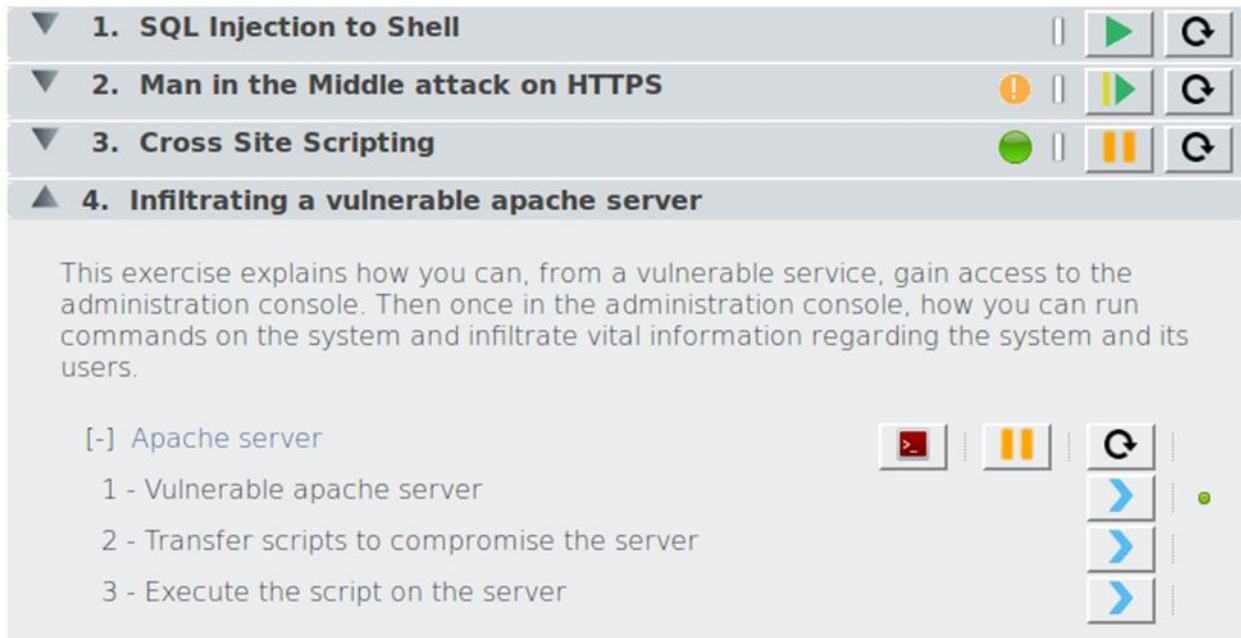

Figure 17 Client side view of the downloaded exercises and various controls

When downloading an exercise, Docker only download the non-existent layers of the component images, and any layer that already exists in the client side are skipped. Image layers skipped during the download is shared with components that have those layers. This makes downloads faster at the client side when downloading similar exercises or exercises with same image layers. Once the exercise is downloaded, it will be made available to the user by the client side webapp. An exercise in the client machine can then be started, stopped or paused either from the client side or the server side. Figure 17 shows the client side view of the exercises with their status information and various controls available to them. Depending on the nature of the exercise components, clients can then directly create a terminal access into the component using the webapp or access service hosted on that components directly from their system. This effectively hides all the underlying infrastructure details and makes it easy for students to download and manage exercise on their own.

Chapter 5 Evaluation

For the purpose of evaluation, a classroom environment was created with one admin server for admins or instructors, one registry server and 16 client machines. All the machines ran Lubuntu 16.04 (lightweight Ubuntu) operating system inside a Virtual Machine running on top of a Windows 8 host connected to a GigaBit network. The admin server ran three separate python scripts – one for broadcasting the service, second for accepting requests and summary updates from clients and third for the server side webapp. Since lot of database and file operations were involved with the operation of the admin server, it was configured with a dual-core processor and was given 3 GB of RAM. The registry server was configured with a single core processor and 1.5 GB of RAM since its only function was to run two Docker containers – one hosting the docker registry and another hosting an nginx proxy server. All the client machines machines were configured with a single processor and a gigabyte of RAM required to run the Docker containers and scripts to provide client side GUI and information updates to the server.

The performance of the infrastructure was evaluated using the time required to performs various functions. The server side functionality span over creating, sharing and controlling exercises, whereas, the client side functionality was limited to downloading and executing the exercises. Since, the server side and client side had different set of functions, performance measurements were done separately based on the execution time of their respective functionalities.

5.1 Evaluation of server side

5.1.1 Creating components for scenario

For measuring the performance, two scenarios were designed and exercises for these scenarios were developed in the server side. These exercises were then uploaded into the registry server, and controls were issued to the clients to download and execute these exercises. The timings for execution of these common server side tasks were measured and then evaluated.

Scenario 1 represented a cybersecurity exercise that demonstrated how an attacker could use cross-site scripting on a vulnerable php webpage to gain admin access and execute commands in the server. A Dockerfile was created to build the container for supporting this scenario and for the sake of explanation let it be named component1. Scenario 2 represented a more complex case with 2 vulenrable machine – one with a vulnerable php webpage and other with a vulnerable apache service. The machine simulating the vulnerable php webpage reuses the Docker image created for scenario

1, modifying only the port it is hosted on. Whereas, a new Dockerfile was created to build the second machine hosting the vulnerable apache service. Let these two images be named component1 and component2, respectively. It is to be noted that, all the Docker images were build from a base image downloaded from the Docker's public registry and not from the local machine or private repo. So, this particular operation is dependent on the bandwidth of the network connection available to the admin server, 90 Mbps in this case.

The figure 18 shows the average execution times recorded for building different Docker images. It can be seen that the component 1 from scenario 1 took less than 60 seconds to build whereas, the component 2 from scenario 2 took about 32 seconds to finish executing the Dockerfile. Since, the component 1 from scenario 2 was build from component 1 of scenario 1 with minor updates, the execution of Dockerfile was completed in less than a second. This is an indicative of how fast it is to make modifications to an exisitng exercise.
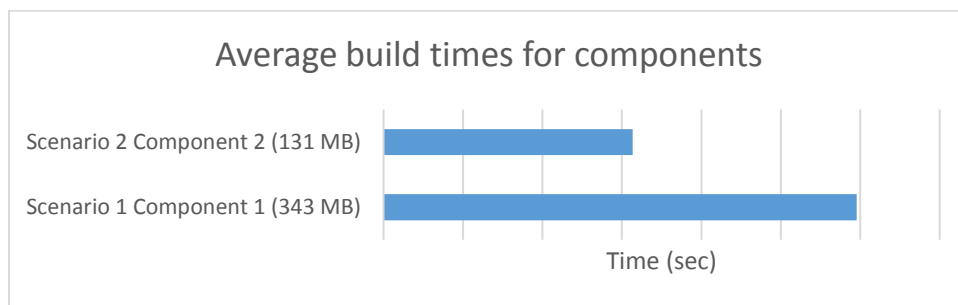

Figure 18 Average build times for components in the server side

It can be derived from these values that, given the necessary scripts and files, the execises can be build in relatively less time when compared to the traditional approaches existing today. Component 1 required 343 MB of storage space where as component 2 required 131 MB. It is to be noted that, a VM image for Debian wheezy is atleast 500 MB in size before allocating the harddisk space or installing the exercise requirements and other updates. So, compartively, exercises build using Docker container consumed less amount of resources.

5.1.2 Sharing Exercises – Uploading and Updating

An exercise has two key elements – a textfile file containing title and description for the exercise, and an index file containing list of all component images required to support the exercise. Additionally, each component themselves have a text file describing their title and description. When a task is issued to upload or update the exercise to the registry, a playbook is dynamically created by the

python webapp to perform the necessary tasks. The tasks includes creating or updating the description files locally and tagging the component image with registry name to make it ready for uploading. Once the images and files are ready, additional task include logging in to the registry server, pushing the component images of the exercise, and finally copying all the created or updated description and index files to the registry server. Therefore, the upload or update time for an exercise is the elapsed time between issuing and completing of all the above tasks.
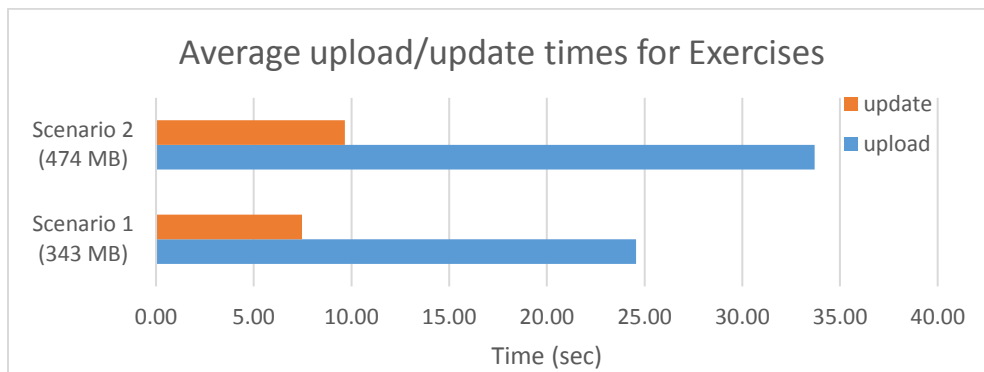

Figure 19 Average upload and update times for exercises

The figure 19 shows the average upload and update times required for scenario 1 and scenario 2. It can be seen that, scenario 1 with a single component of size 343 MB was uploaded in less than 25 seconds while the scenario 2 with two components, 343 MB and 131 MB respectively, was uploaded in less than 35 seconds. It is to be noted that, the registry was cleared and restarted to remove any cache prior to uploading the exercises, for each tested trials. When minor modifications were made to the component image, we noticed that the update time only require less than 10 seconds. These values indicate how quick it is to upload and update cybersecurity exercises in this infrastructure and make it available for sharing.

5.1.3 Controlling exercises in remote machine

One of the main function of the admin server is to provide a centralized monitor and control facility over the clients in the infrastructure. The effectiveness of this feature was measured by considering the time required to perform three main controls or actions – download, start and stop exercises in the client side.

When these controls are issued from the webapp to a set of selected clients, two files are dynamically created – an inventory file containing details of all the selected clients and an ansible playbook describing the controls to be performed on these clients. Each of these controls requires different set

of tasks to be performed inside a playbook. For example, a task to download an exercise from the registry server requires atleast 3 ansible tasks to be performed. First, to authenticate the client to the registry server, second to download and rename all the component images, and third to download all the descrption and index files for exercise and its components. At the same time, only one ansible task is required to start or stop and exercise or its components. Additionally, in the case of downloading an exercises, a database entry is made into the mysql table indicating the status of component images in the selected clients as 'downloading'. As the components in the exercises are downloaded, started, or stopped the client side scripts picks up these changes and updates them periodically to the server. The time is measured for each download, start and stop task from the moment the control is issued from the webapp to the moment when they are executed and all the client side updates have been synced and reflected in the webapp. The time required to complete these tasks was measured for scenario 1 and 2 over a group of clients and averaged.
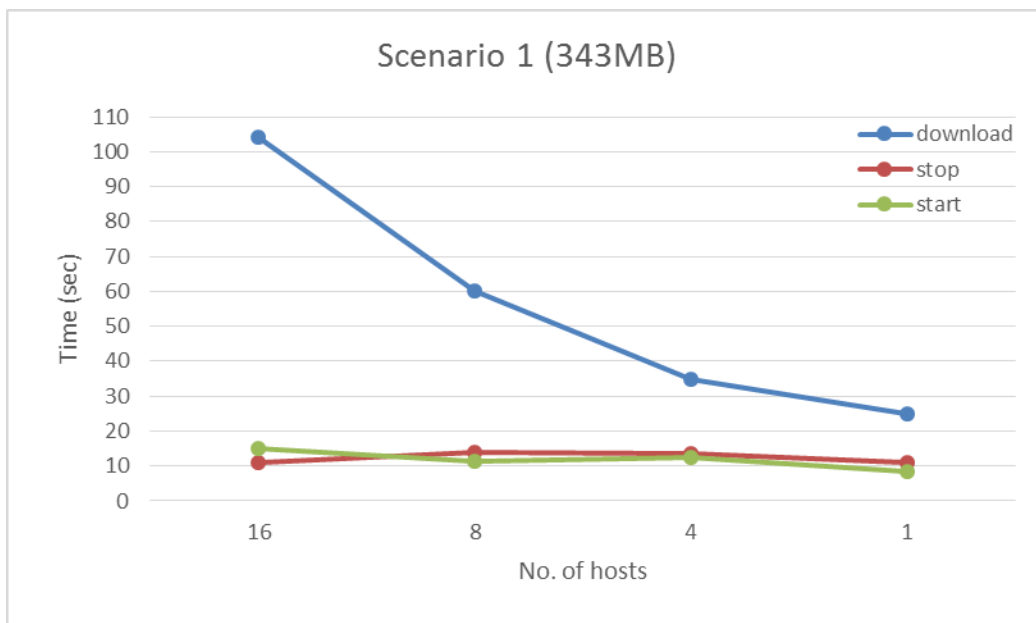

Figure 20 Performance of various controls issued on multiple host for scenario 1

Figure 20 shows the average time recorded for downloading, starting and stopping exercise components of scenario 1 in a group of 1 to 16 clients. To deploy a 343 MB exercise into the client machines it takes only over a minute and half for 16 clients and is under 30 seconds for a single client. Figure 21 shows that for scenario 2, a two component exercise with a combined size of 474 MB, can be deployed to 16 clients under one and half minutes. This indicate that the download times doesnot increase drastically and is fairly consistent even with 100 MB increase in the size of the exercises.

While starting the components is also fairly consistent, with timing under 20 seconds for both the scenarios throughout the different groups of clients, it varies significantly for stopping the exercises. Figure 1 shows, the time to stop scenario 1 component is under 20 seconds for all client sets but takes a maximum of 52 seconds and minimum of 20 seconds to stop both the components of scenario 2 for 16 and 1 client, respectively. It appears that the amount of time required to stop a containers seems to grow with the number of components and clients. In fact, it was noted that most of the clients stopped the exercise in reasonable amount of time and the delay was mostly created by one or two clients. It is to be noted that, stopping the exercise is a two stage process and is done by sending a SIGTERM signal to its running container. The Docker container waits till all the pending or received requests are processed before gracefully exiting. Only after it has exited, Docker removes the container instance and all its supporting files. The stopping time is therefore different in each client and this dependency is increased when stopping 2 components. Even then, the increase in time is not substantial and is well within a minute.
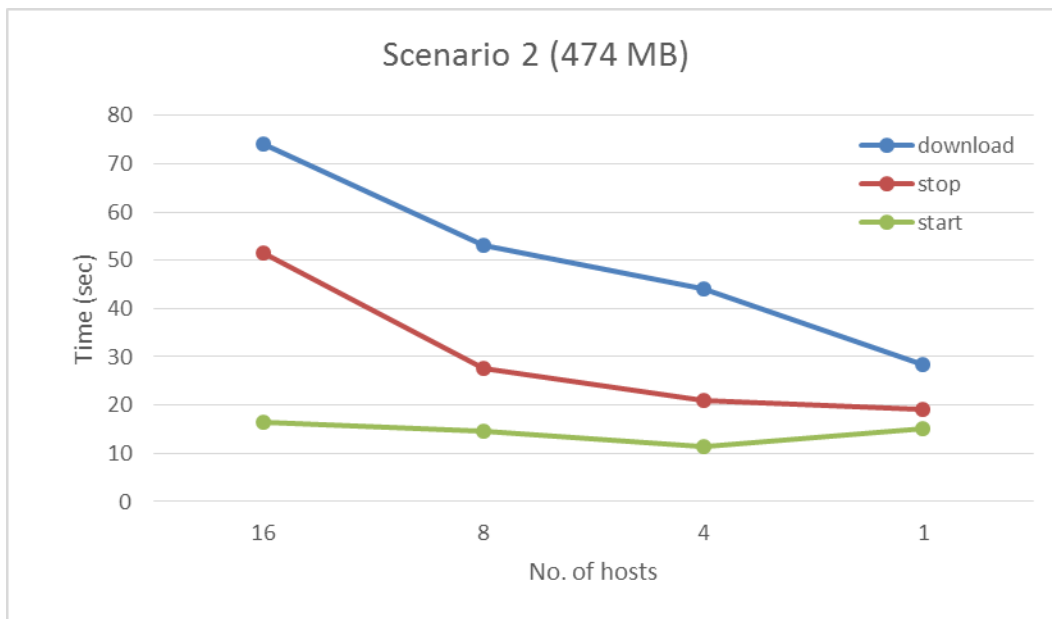

Figure 21 Performance of various controls issued on multiple host for scenario 2

It can be estimated from these figures that, with this infrastructure, the admins or instructors can drastically reduce the time required to get a mid sized classroom ready for hands-on training in a very short amount of time. Since client side updates regarding the exercises are reported every 5 seconds, appropriate action can be without a significant delay.

5.2 Evaluation of client side

Client side measurements are made to evaluate the performace from the standpoint of a single client using this infrastructure. To measure the performance, time required to complete various client side functionalities like downloading, starting and stopping exercises were measured.

For downloading an exercise, similar to the server side, the client also has to authenticate to the registry, download all the component images and their respective description and index files. However, unlike the server side, no database entry, playbooks or inventory file creation is required - equivalent docker and download commands are dynamically created and executed at the press of a button. The execution time recorded for the client side was therefore much faster.
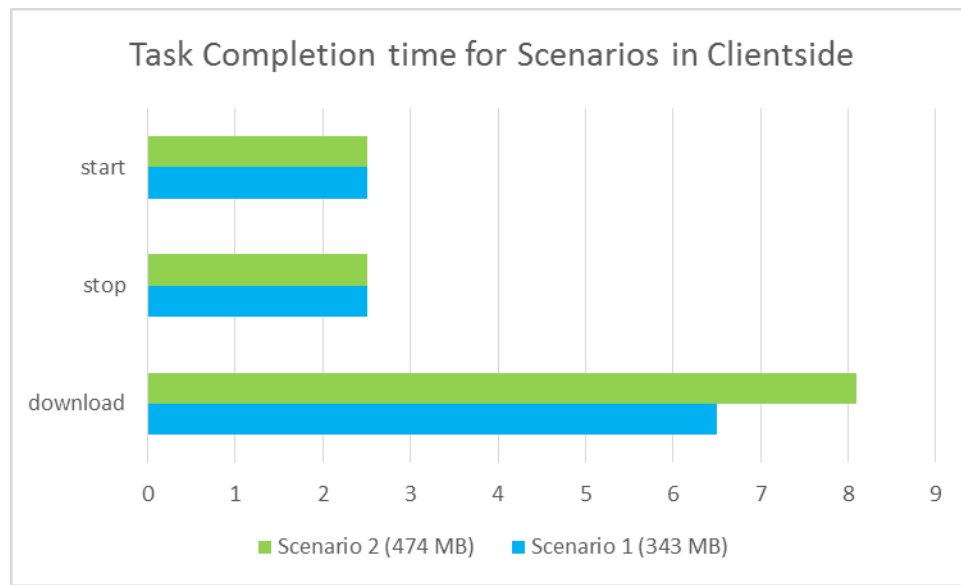

Figure 22 Execution times for scenario 1 and scenario 2 in client side

The figure 22 indicate the average times for download, start and stop tasks performed in a client side machine recorded from 3 iterations. The client side download was complete for scenaro 1 and 2 exercises in less than 9 seconds. Whereas, the starting and stopping for the exercises were complete within 3 seconds. These timing reflect how quick students can manage these hands-on cybersecurity exercises with minimal knowledge or interaction with the underlying infrastructure.

Chapter 6 Conclusion

The main goal of this work was to develop a platform to streamline the process of developing, managing and delivering hands-on training in cybersecurity education. With the help of Docker containers, Ansible orchestration tool and python, an automated platform was developed that is simple, lightweight and easy to use. Evaluation of this platform shows that it can address the issues of current approaches by automating most of the manual tasks and streamlining the process of creating, updating and sharing exercises. Significant reduction in the amount of time and space required for managing these hands-on exercises, from server side as well as client side, makes it more desirable. With a simple and easy to use GUI, the students using this infrastructure can quickly get familiarized with deploying the hands-on exercises with zero exposure to any of the underlying mechanism. On the server side, the GUI gives admins an excellent overview of the clients in the infrastructure and control over its exercises without much delay. This eliminates the unnecessary amount of time spend during class hours on system administration tasks. Bottom line, with this infrastructure, imparting training on simulated cybersecurity scenarios can now be more accessible and manageable even in the hands of the less experienced.

Bibliography

[1] Delaney, J. (2017). Into the Breach. Cacm.acm.org. Retrieved 24 May 2017, from https://cacm.acm.org/news/215838-into-the-breach/fulltext

[2] Data Breach Impact Estimation (2017). Sans.org. Retrieved 24 May 2017, from https://www.sans.org/reading-room/whitepapers/dlp/data-breach-impact-estimation-37502

[3] Verizon Data Brach Investigation Report 2015. (2017) Retrieved from https://iapp.org/media/pdf/resource_center/Verizon_data-breach-investigation-report-2015.pdf

[4] The biggest data breaches of 2016 could and should have been prevented, says The Bunker | The Bunker. (2017). The Bunker. Retrieved 24 March 2017, from http://www.thebunker.net/the-biggest-data-breaches-of-2016-could-and-should-have-been-prevented-says-the-bunker/

[5] M57 Patents Scenario (2017). Retrieved 15 February 2017, from http://digitalcorpora.org/corpora/ scenarios/m57-patents-scenario

[6] Emulab.Net - Emulab - Network Emulation Testbed Home. (2017). Emulab.net. Retrieved 28 March 2017, from https://www.emulab.net/

[7] DETERLab Docs. (2017). Docs.deterlab.net. Retrieved 28 March 2017, from http://docs.deterlab.net/

[8] Hu, J., & Meinel, C. (2004). Tele-Lab "IT-Security" on CD: portable, reliable and safe IT security training. Computers & Security, 23(4), 282-289. doi:10.1016/j.cose.2004.02.005

[9] Rindos, A., Vouk, M., & Jararweh, Y. (2014). The Virtual Computing Lab (VCL):. International Journal of Service Science, Management, Engineering, And Technology, 5(2), 51-63. http://dx.doi.org/10.4018/ijssmet.2014040104

[10] Building a Security Lab with Virtual Machines. (2002). GSEC, 1.4(1). Retrieved 27 February 2017, from https://www.giac.org/paper/gsec/2186/building-security-lab-virtual-machines/103719

[11] Virtual Laboratory Environments: Methodologies for Educating Cybersecurity Researchers. (2009). Methodological Innovations Online. http://dx.doi.org/10.4256/mio.2010.0002

[12] Using Virtualization Technology to Support Cybersecurity Curricula and Competitions. (2013). National Institute of Standards and Technology, Gaithersburg, MD.

[13] Secure Web Development Teaching Module. (2017). Csis.pace.edu. Retrieved 29 February 2017, from http://csis.pace.edu/~lchen/sweet/

[14] Burd, S., Gaillard, G., Rooney, E., & Seazzu, A. (2011). Virtual Computing Laboratories Using VMware Lab Manager. 2011 44Th Hawaii International Conference On System Sciences. http://dx.doi.org/10.1109/hicss.2011.482

[15] K. Krishna, W. Sun, P. Rana, T. Li, R. Sekar, "V–NetLab: a cost-effective platform to support course projects in computer security", presented at the Proceedings of 9th Colloquium for Information Systems Security Education, 2005.

[16] Chapter 14. Jails. (2017). Freebsd.org. Retrieved 22 December 2016, from https://www.freebsd.org/doc/handbook/jails.html

[17] Moments in Container History (2017). Pivotal.io. Retrieved 10 April 2017, from https://pivotal.io/platform/infographic/moments-in-container-history

[18] des Ligneris, B. Virtualization of Linux Based Computers: The Linux-VServer Project. 19Th International Symposium On High Performance Computing Systems And Applications (HPCS'05). http://dx.doi.org/10.1109/hpcs.2005.59

[19] The History of Containers. (2017). Red Hat Enterprise Linux Blog. Retrieved 18 April 2017, from http://rhelblog.redhat.com/2015/08/28/the-history-of-containers/

[20] Wan, Z., Lo, D., Xia, X., Cai, L., & Li, S. (2017). Mining Sandboxes for Linux Containers. 2017 IEEE International Conference On Software Testing, Verification And Validation (ICST). http://dx.doi.org/10.1109/icst.2017.16

[21] Mattetti, M., Shulman-Peleg, A., Allouche, Y., Corradi, A., Dolev, S., & Foschini, L. (2015). Securing the infrastructure and the workloads of linux containers. 2015 IEEE Conference On Communications And Network Security (CNS). http://dx.doi.org/10.1109/cns.2015.7346869

[22] Containers, C. (2017). Chapter 1. Introduction to Linux Containers - Red Hat Customer Portal. Access.redhat.com. Retrieved 19 April 2017, from https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux_atomic_host/7/html/overview_of_containers_in_red_hat_systems/introduction_to_linux_containers

[23] Docker overview. (2017). Docker Documentation. Retrieved 30 April 2017, from https://docs.docker.com/engine/docker-overview/#docker-architecture

[24] Hat, A. (2017). IT Automation with Ansible. Ansible.com. Retrieved 12 February 2017, from https://www.ansible.com/it-automation

Vita

The author Vivek Oliparambil Shanmughan is a computer science graduate at the University of New Orleans. He completed his Bachelors of Technology in Computer Science in 2010 at the Mahatma Gandhi University, India. He acquired his Master in Network Systems from Teesside University, England in 2012. He has two years of system administration experience, handling the network and services for India's biggest research based university, Indian Instiute of Science. For the last two years, he has been working under Dr. Roussev of the Department of Computer Science on various NSF funded projects.