CALIFORNIA STATE UNIVERSITY SAN MARCOS

THESIS SIGNATURE PAGE

THESIS SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE

MASTERS OF SCIENCE

IN

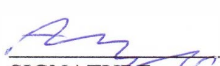
CYBERSECURITY

THESIS TITLE: DESIGN AND IMPLEMENTATION OF A CYBER SECURITY TEST BED

AUTHOR: Henry Brooks

DATE OF SUCCESSFUL DEFENSE: May 7 2019

THE THESIS HAS BEEN ACCEPTED BY THE THESIS COMMITTEE IN
PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF MASTERS OF
SCIENCE IN CYBERSECURITY.

Teresa Macklin		5-7-19
THESIS COMMITTEE CHAIR	SIGNATURE	DATE
Dr. Ali Ahmadinia		05.07.19
THESIS COMMITTEE MEMBER	SIGNATURE	DATE

Design and Implementation of a Cyber Security Test Bed

Henry Brooks

Table of Contents

List of Figures and Tables

Executive Summary

Design and Implementation of a Cyber Security Test Bed

California State University San Macros

Henry Brooks

Masters of Science in Cybersecurity, California State University San Marcos

This paper presents a working implementation of a cyber security test bed with a focus on automating the process of generating reproducible and consistent lab environments for cyber security education. The goal of this project was to provide a framework that faculty and students could use to implement hands-on cybersecurity labs and experiments. A major barrier to providing students with a rich hands-on learning environment is the effort and expertise needed to setup and configure the underlying virtual network that they lab will be built on. Having access to a framework that can consistently create a standardized lab setup would allow instructors to focus their time on designing interesting labs and dedicate less time to worrying about how they will implement the lab environment.

In designing this project, I focused on utilizing open source tools and standards to ensure that the finished project would be easy to implement by instructors and students alike. This framework can be used as an entry point for users who are looking for an easy method to start designing and implementing their own networking labs and cyber ranges. I spent hundreds of hours on this project working through technical problems and documentation issues with the hope of saving this time for future users. If this project can assist an instructor or student in saving a couple of hours of time when they decide to implement a security lab I will feel rewarded for my effort. Lowering the barriers currently present in setting up a virtual network can only lead to more frequent and varied labs for future students.

Acknowledgements

To my wife Ingrid for putting up with these last few years of classes. I didn't think it would take this long either.

Introduction: Design and Implementation of a Cyber Security Test Bed

This paper covers the process and design decisions involved in implementing a virtual network for use as a cyber security test bed. Over the course of the project I implemented small virtual networks of connected virtual machines in most of the major virtualization platforms available today. I have seen first hand how long it can take to research, and trouble shoot issues related to building a fully functional test environment.

I was previously involved in a project that needed a virtual network for testing IoT devices. After researching multiple methods for implementing a virtual network on their local network, the team concluded that it would be more cost efficient and practical to purchase a license from a vendor that designs and runs cyber security test beds. While there were free and open-source solutions available to implement a virtual network, the project couldn't justify the time and costs associated with training team members to productively use these tools. Building and maintaining a virtual network requires a mixture of network engineering and software development skills.

From experience working with virtual networks, I have found that a major barrier to implementing a testing environment comes from difficulties in achieving consistent results. Most of my own learning about network virtualization came from reading blog posts and forum entries and then trying to implement out dated and inconsistent instructions. I would be able to properly provision a virtual server one time, and then forget a step and spend hours trying to reimplement the server days later. The process usually relies on the heavy use of bash scripts and ad hoc configuration changes that must be performed for every virtual machine that you are setting up.

This reliance on one off configuration changes leads to a fundamental problem with setting up a virtual network for use in an educational setting. If the instructor cannot quickly and consistently implement a solution for a problem, they will be disinclined to repeat the process. Related to this I can imagine that there are a staggering number of

worthwhile and innovative lessons and labs that instructors have scrapped because they were too difficult to implement, and the desired results did not justify the time commitment needed to build out a proper test environment. I previously worked in education and know that a major requirement teachers have when they consider implementing a lesson is a cost to benefit analysis of the time required to get the lesson prepped, verse the time and value the students will receive from the lesson. Considering the amount of material that needs to be covered for any subject, an instructor needs to make decisions based on preparation time and learning objectives. Taking hours to set up a lab that will only be used once to demonstrate a security concept is hard to justify, especially if the process cannot be generalized for use in another project.

With these thought in mind I decided that implementing a framework for setting up and running virtual networks for educational use would be a worthwhile project. The goals and requirements for this project were built up over time by talking with my professors and associates about the minimum requirements they would need of a test bed for cyber security labs. From these discussions I identified the minimum viable specs needed by the framework.

- Minimal setup required on the host virtualization system
- Minimal configuration required for the provisioning system
- Preference for open-source solutions
- Final product should offer turn-key method for starting and stopping labs
- Final product should create the same lab environment every time it is run
- The lab setup scripts should utilize a human readable markup language instead of xml or bash

With these goals in mind I decided that the project should take an Infrastructure as Code (IaC) approach to the design and implementation of virtual networks and labs. Since a primary goal was to decrease the number of times that an individual must manually edit and configure systems while setting up a virtual lab, utilizing IaC practices meant that the final lab environment would have most of its configuration requirements located in a single organized location. With the majority of the system configurations

automated through the use of provisioning tools, we could to ensure that the resulting labs would be reproduced consistently every time the system was used to setup a lab.

My initial design for this project called for the use of XCP-ng as the hypervisor technology with KVM backed virtual machines provisioned by Ansible. XCP-ng and Ansible are proven tools in the Infrastructure as a Service (IaaS) space. I initially believed that choosing these tools for my framework stack would lead to the quickest working product and that I would have ample time during this project to implement example labs and demonstrations. Instead I found through extensive testing that there were several issues that kept these tools from working together to create the streamlined framework I had initially planned. With consideration to the design goals I had set up for this project I eventually ended up evaluating many virtualization systems and provisioning tools.

The final product that I am delivering for this project makes use of the Proxmox virtualization system with most of the virtual machines running on the Linux Containers (LXC) architecture and with Ansible being used as the provisioning agent. This software stack accomplishing the overall goals of this project better than any of the other alternatives I tested. While there were tradeoffs made in selecting these tools verse others, the finished product closely matches my initial expectations and provides some functionality that was unexpected when the project was initially proposed. In its current implementation the framework provided by this project allows a user to quickly setup a stock virtual network with minimal user input. Only two areas of the setup fall outside of the IaC guidelines, the initial setup of the Proxmox server, and the initial configuration of the Ansible server inside of Proxmox. All other setup and provisioning is accomplished through the use of Ansible playbooks. A typical user can expect to have a basic virtual network running in under 15 minutes with example labs and networks ready to examine and modify immediately.

The remainder of this paper will cover the process and design decisions that were made during the process of completing this project. In building this framework I researched

and implemented the basic concepts of this project under many of the leading virtualization platforms available today. I attempt to explain why certain technical choices were made and how different software tools were compared and evaluated.

Materials and Methods

This project was initially designed and implemented on an AMD 2700x server with 32 GB of Ram and a 500 GB SSD for storage. This hardware was chosen since I initially expected the project to rely heavily on KVM architecture and needed sufficient cpu cores and ram to run multiple simultaneous virtual machines.

After settling on the Proxmox virtualization system and the use of LXC virtual machines I have been able to implement the framework with virtual machines on a ThinkPad t450 with an i5-4300U processor and 12 GB of Ram with a 500GB HDD for storage. I found that during my testing that Proxmox can be running from an external USB hard drive as long as the laptop supports booting from an external USB drive and the processor supports virtualization.

During this project I tested the following virtualization systems. The testing implementing the basic concepts of the project within the given virtualization system and evaluating how closely the process followed the guild lines of the project. Some of the systems tested required too much upfront configuration, or lacked sufficient tools to automate the process of setting up lab environments. While most systems were capable of implementing all possible virtual networks required by the project, there was not always a readily available method of automating the process.

Resources

	Host OS	Guest OS	Software GUI	Web GUI	API/CLI	License
XCP-NG	Linux	Multiple	Yes	Yes	Yes	GNU GPLv2
Proxmox	Linux	Multiple	No	Yes	Yes	AGPLv3
Hyper-V	Microsoft	Multiple	Yes	Yes	Yes	Proprietary
ESXi	Linux	Multiple	Yes	Yes	Yes	Proprietary
FreeNAS	FreeBSD	Multiple	Yes	Yes	Yes	BSD
oVirt	Linux	Multiple	No	Yes	Yes	Apache 2.0
KVM	Linux	Multiple	Yes	Yes	Yes	GPLv2
LXD/LXC	Linux	Linux	No	No	Yes	GPLv2
GNOME Boxes	Linux	Multiple	Yes	No	Yes	LGPLv2

Figure 1- Virtualization Systems

The following provisioning tools were tested during this project. These tools were tested for their ability to consistently and quickly setup the expected lab environment based on given instructions. While all of these tools have been proven to be effective, the setup for Puppet and Chef was found to be excessive. The requirement for this project didn't need the extra functionality provided by specialized provisioning tools like Puppet and Chef. The configuration and deployment of the virtual machines used in the project was able to be accomplished with simple SSH commands. Ansible also provided a cleaner configuration language choice in YAML.

	Configuration Language	License
Ansible	YAML	GPLv3
Puppet	Ruby DSL	Apache 2.0
Chef	Ruby DSL	Apache 2.0

Figure 2- Provisioning tools

Techniques and Methods

The current implementation of the cyber security test bed makes use of a Proxmox VE hypervisor to run a collection of KVM and LXC virtual machines. Proxmox VE was selected as the virtualization platform for this project after an extensive review of competing alternatives. Determining which platform best met the needs of this project required implementing small virtual networks in each offering. My eventual decision was based primarily on Proxmox VE being an open source project with a bundled web interface and easily accessible command line API. These factors meant that it would be relatively straight forward for users to setup and start using the framework for designing and running labs. During testing it was found that many of the virtualization platforms did not have updated or currently maintained Ansible modules to assist with the automation of the virtual machine setup. Having the Proxmox API readily available from the command line meant that Ansible could still be used for simplified tasks related to starting and stopping virtual machines on the Proxmox host.

After a lot of research into different virtualization methods I eventually decide that this project would rely heavily on Linux Containers (LXC) virtual machines for simulating most of the virtual devices. LXC virtual machines operate similarly to Docker containers while providing a better implementation of state management within the individual virtual machine. Docker is primarily used for running stateless services because it was designed around having a minimal footprint on the system and allowing for multiple instances to be run concurrently without conflicting. LXC is designed for running both stateless and stateful services and lends itself to tasks that do not require constant and heavy resource usage. Instead of locking up resources with a full virtual machine implementation this project relies on smaller LXC instances to model multiple machines within a network acting as database and web servers, as well as, adversaries running scripted attacks and active network inspections. The choice of LXC virtual machines allows this framework to model more simultaneous virtual machines while also requiring less expensive hardware for the Proxmox host machine.

Concerning full virtual machine implementations, this project utilizes a virtualized pfSense firewall for network monitoring and security inspections. While it is possible to implement a secure firewall within a normal Linux distribution or container, pfSense is a proven and open source implementation that provides numerous plugins and extensions that easily expand the number of possible lessons that can be presented to students. With the ease of adding Snort or Suricata to the firewall a cyber security lesson can go beyond simple firewall rules and examine methods for analyzing and detecting malicious traffic and events.

This project also uses Kali Linux as a default KVM choice for interacting with the lab environment. Kali Linux is utilized since it provides a relatively slim Linux distribution that also provides most of the inspection tools and environments that a user would expect to leverage in a cyber security focused lab.

Discussion

Objectives of this project

This project was designed with the goal of making it easier for instructors and students to implement cyber security labs and experiments in an educational setting. Access to an open source framework that can automate the construction and configuration of a test environment would allow users to quickly start learning about security concept without being slowed down with technical issues related to the running a virtual network. Having a reproducible and consistent system to design labs and lessons around would allow instructors to design and implement more engaging and frequent learning experiences. It would also allow for students to easily build on and test concepts and best practices demonstrated in class. Consistency and reproducibility would fundamentally improve the quality of cyber security lessons by removing a major impediment faced by new users and instructors attempting to work with virtual networks.

Currently, if a student is looking to learn about cyber security concepts they must rely on blog posts and forum guides for instructions on setting up virtual networks and systems to test against. Before the student can start experimenting and learning about security vulnerabilities and best practices they must work out how to build a test environment to work with. Many students end up building ad hoc virtual environments that rarely represent networks that they would see in the real world. A student built virtual lab will usually consist of a known and outdated virtual machine image running through virtualization software on their personal computer. They will rarely implement enterprise solutions even if they are freely available as open source projects. Usually the student will default to the simplest solution when implementing their home lab and progressively fix and adjust their test systems over time. The students learning objectives and goals are being slowed down and side tracked with the minutiae of maintaining their ad hoc system, and the method they have used to setup and maintain their system means that they will have difficulty if asked to rebuild the system.

Cyber security instructors also face difficulties with building and updating labs as they currently lack a consistent method of building and testing a lab environment. Before they can start implementing a new lesson they have to build a virtual environment that can be run the lesson. The instructor will have a decide if the lessons value is equal to the time commitment required to implement the lesson. Without access to a reliable method for setting up a lab, an instructor is effectively disincentivized from developing small labs for experiments or in class demonstrations. The instructor will instead rely on older labs that they have experience setting up and running and will only make small incremental adjustments over time. Further compounding this issue is that the labs that they built are modified and adjusted as needed by instructors which means that sharing these lessons and experiments between instructors is difficult because of compatibility issues. Implementing a new lab based on new research or findings becomes a possible time sink for instructors trying to cover more material with each year.

The fundamental goal of this project is to utilize open source build and automation tools to streamline the process of building out a cyber security focused test bed. Utilizing

open-source virtualization and provisioning tools, a user can quickly build out a generic test environment. The test bed generated by these tools can simulate a standardized small network with the ability to model network traffic and automate virtual systems.

Instructors utilizing the test bed framework can focus on developing lessons around cyber security networking concerns with the expectation that the build scripts will consistently create the same base network every time they run it. The framework utilizes “Infrastructure as Code” principles to ensure reliability and reproducibility in the lab environments that it creates. Modification of the test environment is also simplified since the changes can be tracked and tested using open source version control tools and systems.

Students also benefit from this framework implementation as they can quickly jump into lessons concerning network security issues without worrying about configuring systems and network settings. A student can run the same build script as their instructor and expect to generate the same test environment. This also assists students with being able to share their own findings and issues since the methods used to create their test environment are clearly defined in the code base. They can expect to have the same results for a given input as the instructor. The students also gain a working knowledge of systems and tools currently used by professionals to manage and maintain virtual networking environments. The tools used in this framework are regularly used within industry for the provisioning and orchestration of virtual machines and networks.

Design of this project

This project was initially planned around the use of the XCP-ng open source hypervisor with Ansible being utilized for the provisioning of virtual machines running on that system. XCP-ng is an open-source implementation of the Citrix XenServer system and is currently used in industry by companies providing cloud based infrastructure implementations. It has feature parity with several commercial hypervisors solutions

while still being an open source project. AWS specifically uses a modified version of the Xen system for the automation of some of their cloud based virtual networking systems.

XCP-ng can be natively managed by a Windows only management tool XenCenter or through the Xen Orchestra web interface run on a VM within the system. While XCP-ng is based on a proven hypervisor solution, it is a newer fork with current and regular development being made to the system. The documentation and examples found online for this system are only a few years old and are based on current development goals and technologies. For these reasons I decided that XCP-ng would be the best place to start this project and would provide the fewest hurdles to implementing this framework.

When deciding on a provisioning tool for use in this project, I decided to use Ansible. Ansible is an industry standard for distributed configuration management. Ansible relies on a push based configuration system where a host machine running Ansible creates an SSH connection to the managed machines and pushes configuration changes and settings through that connection.

In researching provisioning methods I found that the other two leading tools, Puppet and Chef, both relied on a master/slave setup with the target devices having to running software that regularly polled configuration management changes from a centralized store. While this system does provide some additional guarantees concerning the current working state of the individual machines inside of the network, it also presented extra configuration steps that would need to be implemented with the installation and configuration of the management system on the slave devices. With the goal of the project being the quick and automated setup of a test lab, I felt that these steps would go against the general goals of the project.

The third major component of the framework was the inclusion of the pfSense firewall within the virtual network. PfSense is an enterprise grade open source firewall build on the FreeBSD kernel. PfSense also includes a built in package manager utility that offers a number of security related extensions and plugins. Having previously seen

demonstrations of the individuals running pfSense within virtual environments, and I knew that having access to a fully functional firewall system was critical to any cyber security lab environment. There are many labs and experiments that require a student to understand how to setup and maintain a network being partitioned and managed by a firewall. PfSense's plugin library also includes popular tools like Snort and an open source Intrusion Prevention System in Suricata. I feel that most of the value of this project couldn't have been realized without the ability to implement a virtualized firewall within the system. Many of the introductory lessons and policies within cybersecurity instruction revolve around methods of setting up and partitioning networks to secure different systems and data.

The final major tool that I planned to utilize for this project was Tcpreplay. This is a free open source tool that can record and playback network traffic. My initial idea was that this tool could be used to record specific traffic data related to normal internet traffic and targeted cyber-attacks. These recordings could then be played back as part of a lab to demonstrate methods for identifying and mitigating network traffic concerns. Since one of the goals for this project was to create a test bed for running reproducible labs I knew that I needed a method for running the same experiment repeatedly for a given set of inputs. Tcpreplay allows instructors to simulate network traffic between virtual machines and the internet without exposing the virtual environment to the actual internet. Instructors can choose to run specific traffic, or general traffic depending on what the lab specifically needs.

Implementation of this Project

The first issue I had with building out this project was related to my initial choice of using XCP-ng as the hypervisor the system would be built on. Underlying XCP-ng and XenServer is the implementation of the Xen virtual machine and its related design choices. XCP-ng can be considered a true type-1 hypervisor in that its management kernal is designed to be loaded into a smaller partition on the main system drive. The virtual machines that will be loaded onto the system need to be loaded through an ISO repository setup separately from the default installation of XCP-ng. Further, the official

management tool for XCP-ng is the XenCenter software tool that is a Windows only program. The free open source web interface for XCP-ng, Xen Orchestra, needs to be run from a separate virtual machine and to achieve the best results requires the user to build the project following directions from its github page.

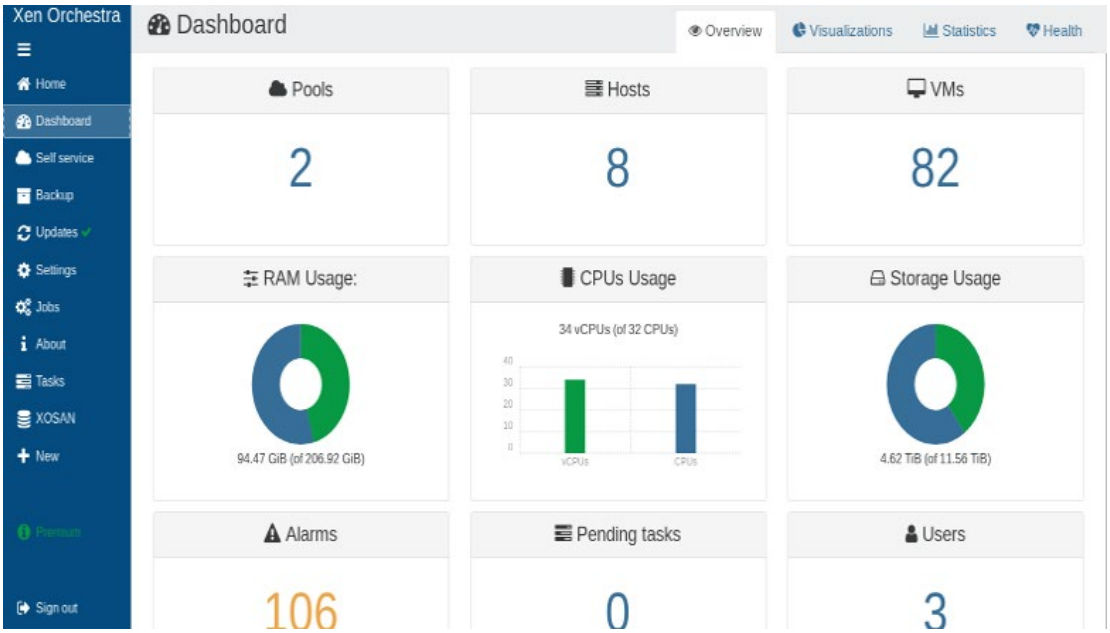


Figure 3- XCP-ng Web Interface

While I was able to get XCP-ng to eventually run well following all these steps, I was never able to sufficiently automate the tasks needed for setting up the system. A major driver for this project was the goal of reducing the number of steps required for an end user to achieve a set virtual network state consistently. I feel that the steps required to properly set everything up presented too much of a hurdle for many users and would not have significantly simplified the process of setting up a virtual network in the eyes of most users.

Having identified issues with implementing XCP-ng as the base hypervisor I started a new investigation of hypervisor and virtualization technologies that could be used as the basis for my project. During this time I installed and implemented simple virtual networks with VMWare ESXi and Microsoft Hyper-V. While both products represent closed source virtualization implementations I felt that their ease of use might justify these tradeoffs.

VMWare ESXi represents years of development and it has a free edition that mostly removes features that are not needed for this project. The major features removed from the free edition are the ability to create a cluster of hypervisors and a restriction to 8 virtual machines. Since these were not features that I expected the lab to need the tradeoff seemed justifiable. ESXi also offered some extra benefits for potential students in that they would be able to gain experience working with an industry standard for virtualization technology. Many companies currently utilize the enterprise version of this software within their development and production stack. The major issue with implementing this project with ESXi ended up being the issue of how it is licensed and its closed source nature. I wanted this project to be relatively straight forward for the user to implement and requiring that they sign up with VMWare for the free edition of the software made this a less attractive choice.

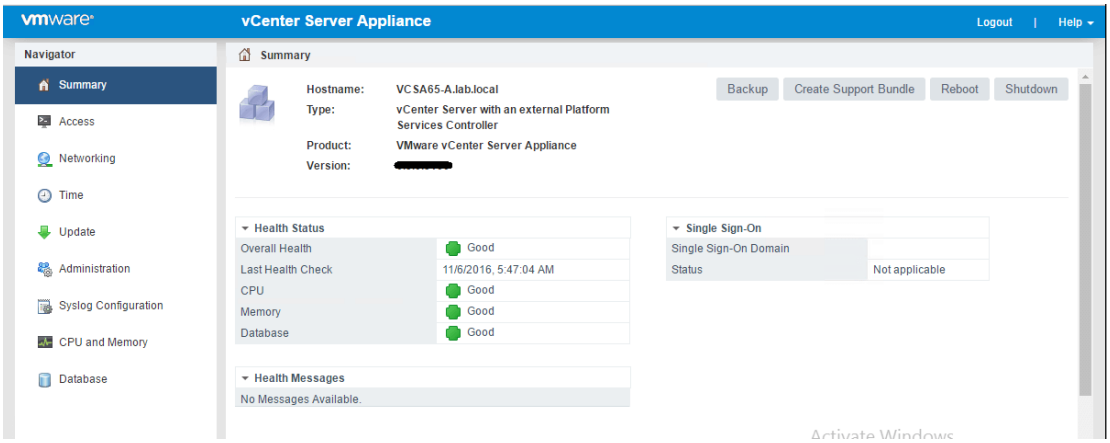


Figure 4- ESXi Web Interface

Microsoft Hyper-V is another closed source virtualization implementation that I investigated. Hyper-V comes bundled with Windows 10 Pro and Education editions as well as the free Hyper-V Server operating system. While it would have been helpful to have the software installed natively on all versions of Windows, there was already an expectation within the project that you would have to install a dedicated operating system to run the virtual machines so this wasn't a deal breaking issue. I also found that the virtual switch implementation in Hyper-V to be superior to that presented by ESXi as it allowed for more fine grained control over the rules on the switch and presented a

better analog for physical devices. Considering that some cyber security labs would need a method for implementing switching rules I considered the extra functionality to be a welcome addition to the project. However, like the case for ESXi, the closed source nature of the product made me pass on implementing the project using Hyper-V. The process of acquiring the licenses and setting up the systems meant that these alternatives couldn't compete with the simplicity of purely opensource options.

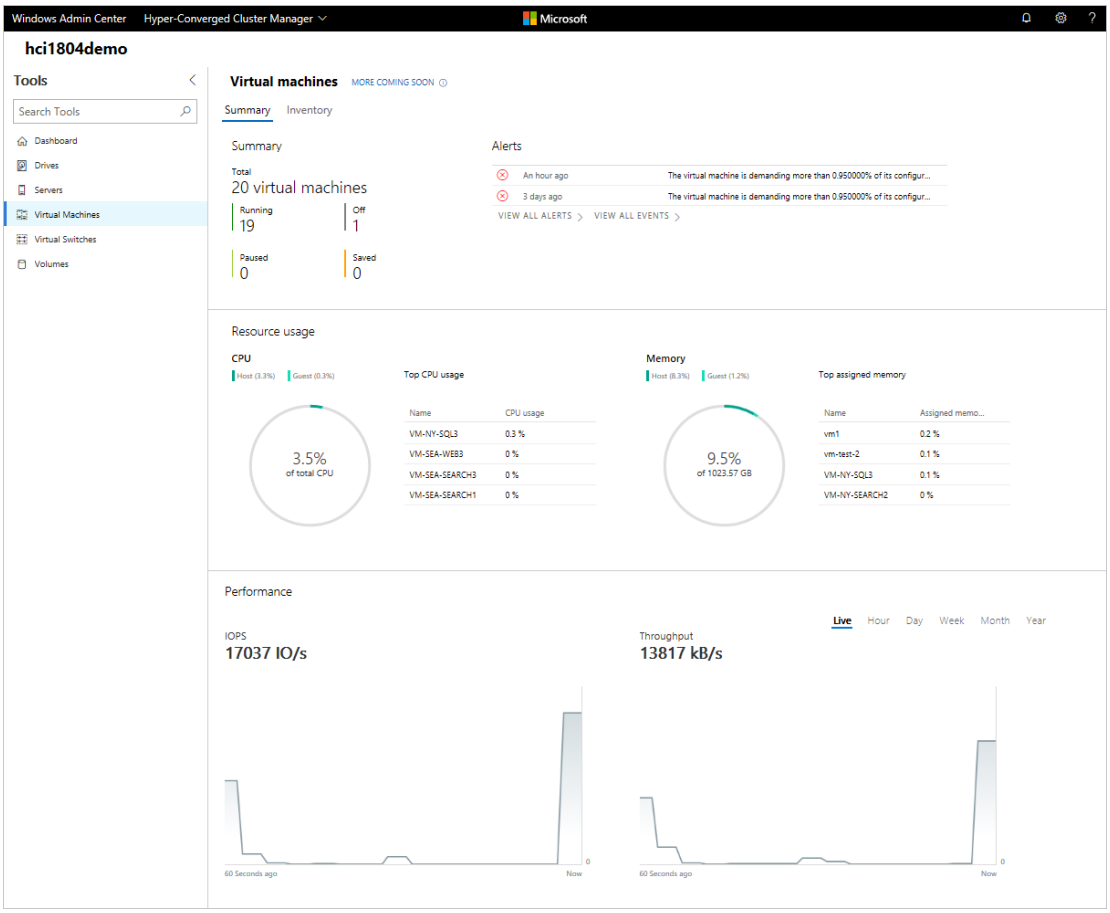


Figure 5- Hyper-V Web Interface

Expanding my search for a suitable virtualization technology I next investigated Proxmox, Virt, KVM, LXC, and GNOME Boxes. These virtualization implementations sit closer to the type-2 hypervisor line, however the Linux kernel exposes a good portion of its underlying hardware to these systems meaning that they require less overhead than a typical type-2 hypervisor like VirtualBox. These projects are all open source and while they lose some functionality against ESXi and Hyper-V when it comes to virtualization of switches they still presented solid alternatives.

I initially started looking at Proxmox and GNOME Boxes before switching to look at oVirt and the ecosystems of projects that support the KVM QEMU virtual machine architecture. I have worked on enterprise products that make use of KVM virtual machines running on hardened SELinux distributions from Red Hat. I found that implementing a KVM virtual network under CentOS was a more complicated experience than creating a similar network with the XCP-ng, ESXi, or Hyper-V. Implementing the virtual network under SELinux did present some benefits to the users as it allowed them to start on the right foot when learning about security concerns and techniques. Going forward the users could make informed choices based on experience gained working with a security conscious software implementation. In looking at the software for managing the system the Virt-manager software package wasn't as polished as the management software used by the other projects. The oVirt web interface provided similar functionality to Xen Orchestra with good support for ansible automation and virtual machine monitoring. I would have probably stuck with this software for the final implementation of the project however, I ended up going with an implementation with better LXC virtualization integration.

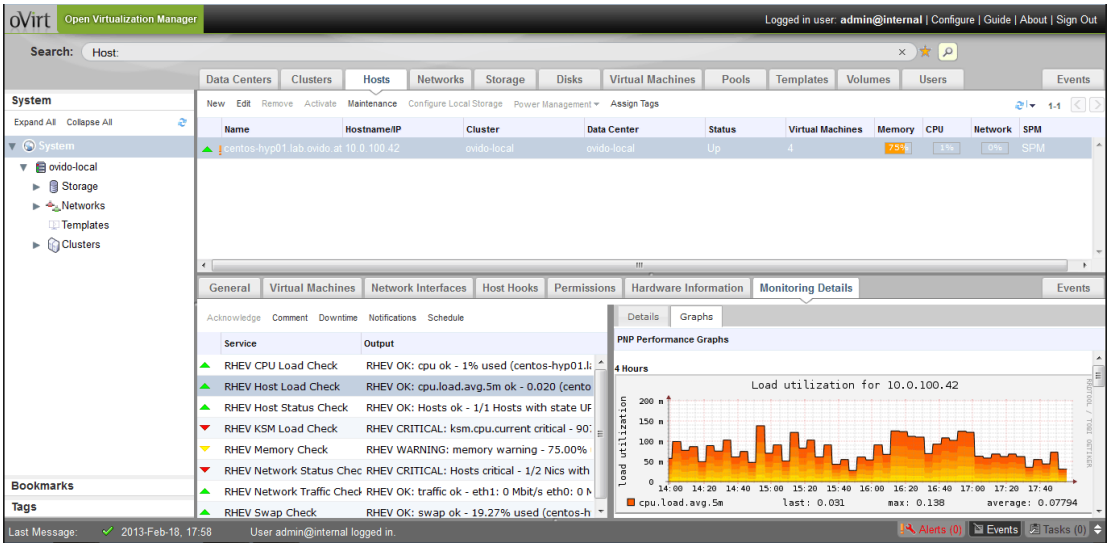


Figure 6- oVirt Web Interface

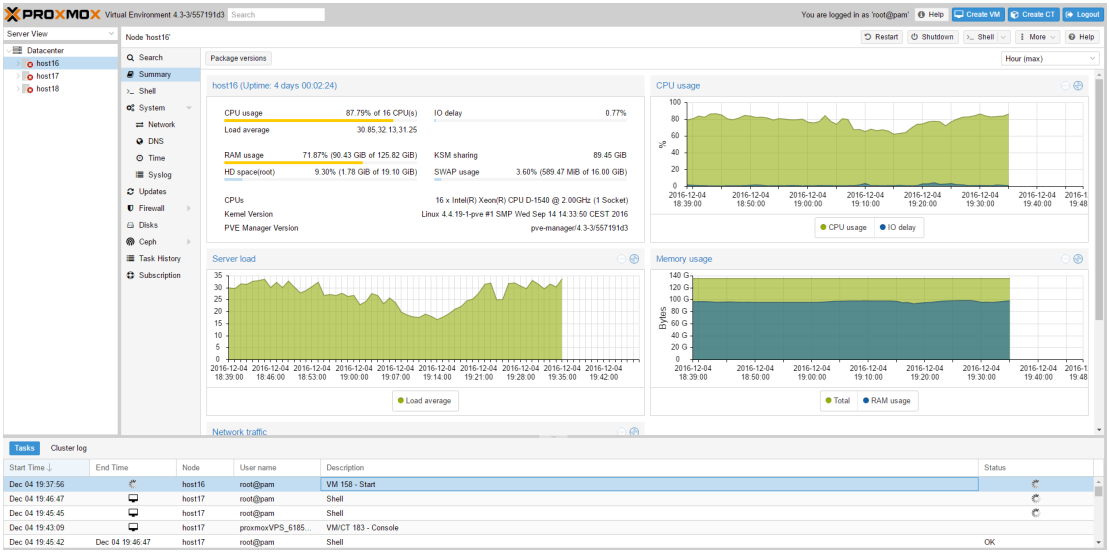


Figure 7- Proxmox Web Interface

During my investigation of virtualization technologies on Linux I learned about the LXC virtual machine implementation and realized that it would be possible to simulate many smaller virtual machines utilizing this technology. In examining the goals of the project, I realized that most of the machines that would be virtualized in the lab would not require a fully virtualized system. Specifically, I only needed a couple of devices to act as full virtual machines with graphical interfaces and persistence. Utilizing LXC virtual machines I could reasonably have dozens of systems sending and receiving network traffic simultaneously without requiring significant computing power.

Considering these findings, I worked at implementing the basic virtual network with the native Ubuntu LXD implementation as well as LXDUI and Proxmox. I ended up sticking with Proxmox because it seems to have the best implementation of both KVM and LXC management. The relative process of installing Proxmox VE on a server and getting virtual machines running was smoother than any of the other alternatives that I tried at this point.

During this process of deciding on the base operating system, I was also struggling to implement the provisioning system for the virtual machines in each of these virtualization systems. While the general process for setting up a virtual machine on most of these systems is well documented, the experience can be time consuming and a

goal for this project was to automate these types of tasks to smooth over the process of getting a work virtual network started for instructors and students.

Though I initially planned on using Ansible for this project I quickly found that the modules used by Ansible for managing and provisioning virtual machines under these virtualization systems were not being consistently maintained. I learned that some of the libraries used for connecting to hypervisors are years out of date and make heavy use of deprecated API systems which required significant modification to work correctly. The difficulty with configuring virtual machines with Ansible made me look into running dedicate provision software on the virtual machines through either Puppet or Chef.

While I didn't want to add unneeded complexity to the virtual machines that would be run in the network, I did need to identify a method of quickly setting up multiple machines following a set of rules. I wanted this project to properly demonstrate "Infrastructure as Code" and I needed the results to be replicable between runs. Having the instructor and users set up machines one by one on the Proxmox server defeats most of the purpose for the project. I found that both Chef and Puppet have modules that work well with Proxmox after some adjustment. Both projects have methods to connect directly to the Proxmox host and create LXC and KVM hosts based on direction laid out in their provisioning code. The major issue with deciding to implement these systems was that both of them are no longer actively maintained. These provisioners offered better functionality than the Proxmox module for Ansible however, they were just as out of date and there is no guarantee that they will be updated or maintained if there is a breaking change.

Eventually I went back to using Ansible as the provisioning tool for this project because I eventually found a reddit post showing an example of how a user was able to provision virtual systems directly from the Proxmox shell utilizing the Proxmox API. While it doesn't present the nicer aesthetics of some more fully fleshed out provisioning implementations, it was still enough to automate the task of setting up and staging

multiple virtual machines with minimal user supervision. I have been able to write Ansible playbooks that will cleanly switch between different labs by starting and stopping different virtual machines that were preconfigured and staged by Ansible. Once the virtual machine has been created through the Proxmox shell Ansible is able to finish the configuration by creating an SSH connection directly to the virtual machine. While the inability to fully create and provision a virtual machine in one go added some complexity to the build scripts, it was a much smaller addition compared to the inclusion of either Puppet or Chef.

The last hurdle I faced was creating playbooks for the build out and implementation of the different labs. I am still working to identify the best set of base machines for most situations. My current working goal is to have a master provisioning playbook that will set up a dozen virtual machines with networking rules so that an instructor can easily design multiple small lessons concerning network traffic and cyber security concerns. To assist instructors and students in implementing their own labs with this tool, I need to include some example labs that they can use as models and tutorials for their own work. Starting with the base lab environment shown in lab number 1, an instructor could ask the students to implement firewall rules that will allow the outside internet to access the webserver while not exposing the other machines located on the network.

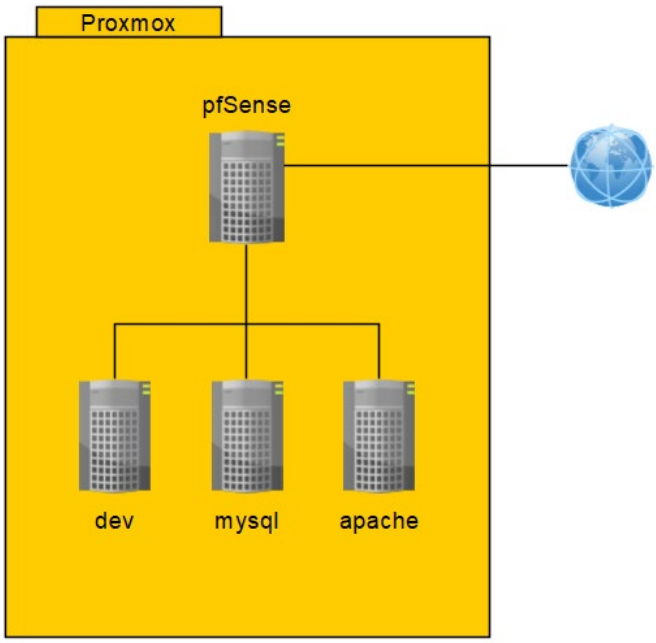


Figure 8- Lab #1 Network Diagram

An instructor could also develop lab around lab number 2 and ask student to identify what traffic can flow through the DMZ and back to the git server.

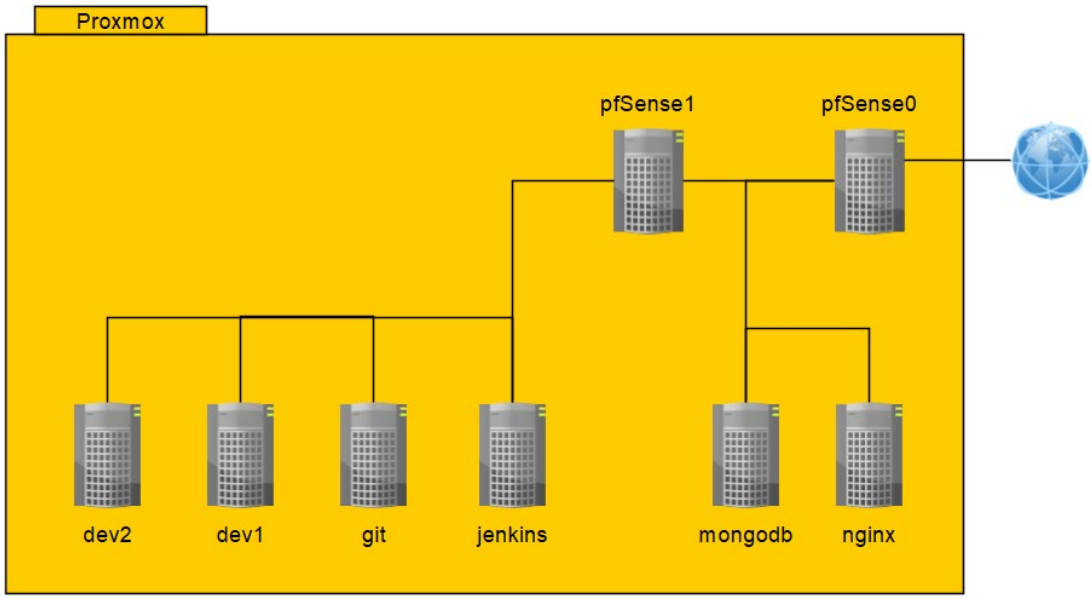


Figure 9- Lab #2 Network Diagram

There is still a need to further refine and develop these basic labs so that users can clearly see the diversity of options that are possible in designing new hands on cybersecurity labs.

An added bonus that I identify near the end of this project was that Proxmox VE could be run off of a USB hard drive as long as the main system supported it. With this knowledge I have been able to successfully implement this framework on an older ThinkPad t450 laptop without modifying the operating system kept on main storage. This means that students can implement this framework on their personal computers without dedicating that device to running these labs. Instead the lab becomes a system that they setup and run off of a USB stick when they have a need to. I believe that this finding will dramatically increase the likelihood that students will attempt to utilize the framework as it significantly lowers the barriers presented in setting up the host system.

Conclusion

I feel that this project has been able to reach most of its initial design goals and has exceeded some of my initial expectations. With the current implementation I can have a virtual network setup and running in less than 30 minutes with less than 5 minutes given to managing the process. For most of the setup the user just runs the needed script and waits for the system to finish downloading and installing the software packages. I feel that this framework will significantly reduce the time professors currently take when implementing virtual networks. I believe that this framework provides a significant improvement in streamlining the process of creating and managing virtual networks for use in labs. The quality of virtual labs should also improve because the implementation steps are clearly defined in code and are not a process of ad hoc configuration changes. This framework should truly allow for reproducible and consistent networking labs.

This project has also been incredibly informative and instructive for my own learning. A side goal of this project was to expand my knowledge of networking protocols and practices. In researching the different virtual switch implementations and setting up the

pfSense firewall for different labs, I learned a great deal about network design. This project also introduced me to several virtualization technologies and implementations that I wasn't aware of. While I had used and heard of some of these tools before, I never had a reason to use them for a practical purpose before. Having hands on experience implementing this framework in these different systems has done more for my understanding of the underlying technologies than I would have expected. Hopefully instructors will be able to utilize this framework to design new hands on labs that will allow future students to gain greater knowledge through hands on work with designing and implementing virtual networks.

References

Elisabeth Dubois at the University of Albany designed a test platform for running and collecting data on simulated cyber-attacks. This test platform was used to collect data and aggregate findings from attacks launched against virtual machines.

EXSi Web Interface. Digital image. vSphere Client. 25 April 2019, http://www.enterprisedaddy.com/2016/11/vsphere-6-5-management-interfaces/

Facolta di Ingegneria at the Universita Degli Studi Di Genova designed a framework for the cyber ranges with an emphasis on creating scenarios for testing red and blue team skills. This framework focuses on streamlining the process of setting up and running consistent and reliable cyber ranges.

Hyper-V Web Interface. Digital image. docs.microsoft.com, 25 April 2019, https://docs.microsoft.com/en-us/windows-server/manage/windows-admin-center/use/manage-virtual-machines

LaCroix, J. LearnLinux.tv. 2019, January 12. Setting up your own virtualization server with Proxmox Virtual Environment [Video file]. Retrieved from https://www.youtube.com/watch?v=MO4CaHn1EjM

Lawrence, T. Lawrence Systems / PC Pickup. 2018, May 16. 2018 Getting started with pfsense 2.4 from install to secure! including multiple separate networks [Video file]. Retrieved from https://www.youtube.com/watch?v=9kSZ1oM-4ZM

Lawrence, T. Lawrence Systems / PC Pickup. 2018, January 14. My Home Virtualization Server Running pfsense Inside of Citrix Server & Autostarting VM's in XEN [Video file]. Retrieved from https://www.youtube.com/watch?v=HUzWnkfxcok

Proxmox Web Interface. Digital image. forum.proxmox.com. 25 April 2019, https://forum.proxmox.com/threads/webinterface-is-showing-all-as-offline.30793/

oVirt Web Interface. Digital image. ovirt.org. 25 April 2019, https://ovirt.org/develop/release-management/features/ux/uiplugins43.html

Rakshith Thayi at California State Polytechnic University examining security concerns related to network connected SCADA devices and developed a test bed for simulating these systems.

Sujatha Krishnaswamy at Iowa State University examines the benefits of using cyber test beds for testing and validating engineering software and systems.

"Virtualizing pfSense with Proxmox." docs.netgate.com, https://docs.netgate.com/pfsense/en/latest/virtualization/virtualizing-pfsense-with-proxmox.html. Accessed 2019, March 31

Xen Orchestra Web Interface. Digital image. Xen Orchestra. 25 April 2019, https://citrixready.citrix.com/vates/xen-orchestra.html