Secure Enclaves for Linux with KVM

Strengthening Security Measures in Linux Kernel-based Virtual Machine (KVM) Environments through Penetration Testing and Hypervisor Configuration

Nathaniel Desany, Nicholas Phillips, Zachary VanDerVelden

-

Marist University

CMPT 479 - Cybersecurity Capping

Dr. Casimer DeCusatis

-

In Conjunction With:

IBM, Fernando Pizzano

2024

Table of Contents

Introduction

Abstract

Virtualization technology allows multiple virtual machines (VMs) to run on a single physical server, sharing resources such as processor, memory, and I/O devices while maintaining logical isolation between them. In this paper, we explore the process of hardening a kernel-based virtual machine (KVM) stack, focusing on the installation and configuration of Ubuntu Linux 24.04 within a VM under the KVM hypervisor. The VM is managed by QEMU 8.2.2, an open-source machine emulator and virtualizer, and Libvirt 10.0.0, an open-source tool that facilitates virtualization management. We examine how these tools can be used to create a custom security profile, which is attached to processes to limit their actions based on specific conditions.

Our project investigates vulnerabilities that could potentially allow for unauthorized access between VMs running on the same hardware, highlighting the security implications of KVM compared to other Linux-based hypervisors. The security profiles developed through this research have broader applications, including the possibility of enhancing enterprise-class Z System servers with zero-trust architectures, paving the way for more secure, isolated, and efficient virtualized environments.

Summary

This project focuses primarily on developing a virtual environment and testing security profiles for a Linux-based virtual environment running on x86 hardware, using a Kernel-based virtual machine (KVM) as the core hypervisor technology. Our goal for the project was to set up a KVM stack and test the hypervisor’s features, eventually coming to a point where we could modify system settings, enabling and disabling hypervisor features to enhance security. This project, which was done with client Fernando Pizzano's help inside the Marist College ECRL Lab, emphasizes strengthening VM isolation to resist known Common Vulnerabilities and Exposures (CVEs) for real-world application.

Tools such as QEMU, Libvirt, and Ubuntu running as the host OS were utilized in this project to ensure the environment setup went smoothly. This project explores VM management, real security threats, and the risk of lateral attacks within a zero-trust setup. Additionally, a Software Bill of Materials (SBOM) was developed to address risk management efforts surrounding the software components of our setup.

Thorough research on CVEs was conducted, to find relevant and applicable flaws in our setup. The CVVs were tested and documented based on how real threat actors would approach a partitioned environment like what we have set up.

Ultimately, this Security Profiles and Secure Enclaves for Linux with KVM capstone project aims to produce a publishable technical report and presentation, contributing valuable information to the field of virtualized security. The results presented have implications for secure, zero-trust architectures and can inform best practices in similar environments where exact isolation and security are paramount. This research and application aims to support industry and education in building more resilient systems capable of withstanding evolving security threats.

Contributions

Introduction

All group members contributed to the project setup, deployment, research, and application. Each group member also worked on substantial parts of the project paper and presentation, completing individual sections and proof-reading each other's work. We have listed each group member's individual contributions to the Security Profiles and Secure Enclaves for Linux with KVM project, project plan, and presentation.

Nathaniel Desany

For this project, I wrote the introduction, including the project abstract and summary. I created the project plan timeline and assisted in the creation of the weekly traffic light reports. For the environment setup, I downloaded the initial Ubuntu OS and set it up on the physical server we utilized for the project and set up permissions as well as allocated disk space to the separate VMs partitioned by the hypervisor. I conducted extensive research on CVEs, how they work, and how to exploit them. I researched VM penetration methods for CVE-2016-2184, deployed the malicious code, and created its final analysis. I wrote the introduction of the Industry Usage section, along with the CVE and Hyperscale Cloud Computing documentation. I researched and documented future competitions and opportunities for our work to be published in academic and technical journals.

Nicholas Phillips

For this project, I wrote and documented the setup of our Hypervisor and Virtualization environment, along with explaining the hypervisor we chose and why we chose it. I researched CVEs that affected QEMU 8.2.2, found CVE-2024-8354, and tested the vulnerability on our system. This included passing through a USB to the Virtual Machine from the server, creating code to test the vulnerability, and having AI examine the system logs after attempted exploits. I wrote the Overview and the Classification of the CVE section, as well as the section regarding CVE-2024-8354. I also wrote the Hypervisor section regarding real world applications in the Industry Usage section.

Zachary VanDerVelden

For this project, I wrote the Project Plan portion along with creating all Weekly Traffic Light Reports, ensuring we stayed on track for our completion date of December 4th, 2024. I then transferred all reports into our final documentation. In addition, I wrote a detailed summary of our initial operating system and the processes performed to get it up and running. I also performed some research on CVEs related to Libvirt while also aiding in the exploitation of CVE-2016-2184 and CVE-2024-8354. Shortly after, I created the Software Bill of Materials and analyzed the results returned by Dependency-Track, forming plans and new ideas for the future of this project. With the knowledge gained from the SBOM, I wrote the entirety of the Software Bill of Materials section in our documentation and gave readers everything necessary to create their own SBOM while also informing them on the importance of the environment analysis tool itself. Once this was completed, I wrote the introduction for our Future Plans section, along with our plans to exploit more CVEs in the future, utilize the SBOM, and train artificial intelligence to read over and analyze logs outputted by our system. Lastly, I consolidated all sources and created a references page that includes all references written in IEEE format.

Project Plan

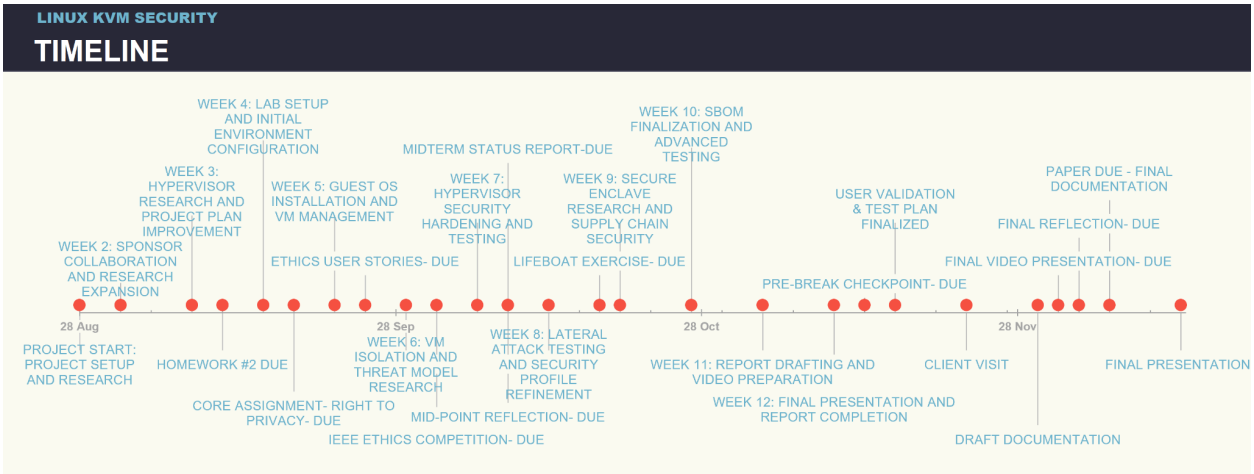
At the start of our project, we were tasked with creating a project plan that would guide us to the completion of our project by the end of the semester. Given the strict deadline of December 4th, our team was well aware of how important it was to develop a thorough plan that would allow us to work more efficiently. With the help of Professor Casimer DeCusatis and Fernando Pizzano of IBM, we discussed our project brief and jotted all of the requested deliverables down. At this point, our team decided it would be best to utilize ChatGPT to complete the plan for us. ChatGPT offers a large number of advantages when it comes to project management that can only benefit us and our project. If given the opportunity, it can create project plans, track progress, and document any research and progress. In our case, we made sure to limit its responsibilities to strictly creating a project plan.

In the image below is an edited version of the project plan generated by ChatGPT. After asking the AI tool to generate the plan for us, it returned a plan with Milestones, Tasks, and Notes. From there, we went through the output and made any changes that needed to be made and personalized it to make it a perfect plan. We divided roles between 3 team members and color-coordinated them to ensure the plan was legible and easy to understand.

Key				
Nate				
Zach				
Nick				
Date	**Milestone**	**Tasks**	**Responsible**	**Notes**
8/26/2024	Week 1: Project Setup and Research	Review the project and critical resources with Professor DeCusatis.	All	Goal: Establish foundational knowledge and environment setup.
8/26/2024	Week 1: Project Setup and Research	Meet with group members to lay out a general plan and discuss the project.	All	Goal: Establish foundational knowledge and environment setup.
8/26/2024	Week 1: Project Setup and Research	Review previous year's documentation and resources for context.	All	Goal: Establish foundational knowledge and environment setup.
9/2/2024	Week 2: Sponsor Collaboration and Research Expansion	Meet with sponsors of other groups to explore potential collaboration.	All	Goal: Explore potential collaboration and research methodologies.
9/2/2024	Week 2: Sponsor Collaboration and Research Expansion	Research methods from Quantum Computing and Leo Project presentations.	All	Goal: Explore potential collaboration and research methodologies.
9/2/2024	Week 2: Sponsor Collaboration and Research Expansion	Schedule lab time in the ECRL Lab with Professor Algozzine and his student workers.	Zach	Goal: Explore potential collaboration and research methodologies.
9/2/2024	Week 2: Sponsor Collaboration and Research Expansion	Continue deep-dive research into virtualization, hypervisors, and secure vault technologies.	Nate	Goal: Explore potential collaboration and research methodologies.
9/9/2024	Week 3: Hypervisor Research and Project Plan Finalization	Meet with client, Fernando Pizzano, to gain a better understanding of project requirements.	All	Goal: Meet with the client and improve the project plan
9/9/2024	Week 3: Hypervisor Research and Project Plan Finalization	Research Type 1 and Type 2 hypervisor implementations and security hardening methods.	Nate	Goal: Meet with the client and improve the project plan
9/9/2024	Week 3: Hypervisor Research and Project Plan Finalization	Improve project timeline and goals based on client feedback and team discussions.	Nate	Goal: Meet with the client and improve the project plan
9/16/2024	Week 4: Lab Setup and Initial Environment Configuration	Set up the server in the ECRL lab with lab technicians and student workers.	All	Goal: Set up project environment and initial configurations.
9/16/2024	Week 4: Lab Setup and Initial Environment Configuration	Begin installing and configuring the hypervisor (KVM) and VMs.	All	Goal: Set up project environment and initial configurations.
9/16/2024	Week 4: Lab Setup and Initial Environment Configuration	Begin researching QEMU and its potential implementation strategies.	Nate	Goal: Set up project environment and initial configurations.
9/16/2024	Week 4: Lab Setup and Initial Environment Configuration	Ensure the lab environment is ready for further testing.	All	Goal: Set up project environment and initial configurations.
9/23/2024	Week 5: Guest OS Installation and VM Management	Install various guest operating systems on the hypervisor.	All	Goal: Complete guest OS installations and verify VM functionality.
9/23/2024	Week 5: Guest OS Installation and VM Management	Manage VM configurations and document settings.	Nate	Goal: Complete guest OS installations and verify VM functionality.
9/23/2024	Week 5: Guest OS Installation and VM Management	Test basic communication between VMs and verify the hypervisor's are functioning properly.	All	Goal: Complete guest OS installations and verify VM functionality.
9/23/2024	Week 5: Guest OS Installation and VM Management	Document initial VM setup configurations.	Nick	Goal: Complete guest OS installations and verify VM functionality.
9/30/2024	Week 6: VM Isolation and Threat Model Research	Explore VM isolation techniques and begin testing.	All	Goal: Research and implement VM isolation techniques.
9/30/2024	Week 6: VM Isolation and Threat Model Research	Research trusted domains (Intel TDX) and secure vault technologies (Intel SGX).	All	Goal: Research and implement VM isolation techniques.
9/30/2024	Week 6: VM Isolation and Threat Model Research	Develop initial threat models based on potential VM vulnerabilities.	Zach	Goal: Research and implement VM isolation techniques.
9/30/2024	Week 6: VM Isolation and Threat Model Research	Evaluate VM isolation configurations for lateral movement prevention.	Zach	Goal: Research and implement VM isolation techniques.
10/7/2024	Week 7: Hypervisor Security Hardening and Testing	Apply security hardening techniques to the hypervisor	Nate	Goal: Strengthen hypervisor security and conduct initial tests.
10/7/2024	Week 7: Hypervisor Security Hardening and Testing	Test and verify hypervisor integrity.	Nate	Goal: Strengthen hypervisor security and conduct initial tests.

Linux KVM Security Project Plan developed with the help of ChatGPT

After the project plan was formed, the next step was transferring our milestones into a timeline. Having a timeline available has proved to be extremely useful, as it enables us to view our project plan in a more visually pleasing form. It has allowed us to motivate ourselves to stay ahead of our project and also holds us accountable when we do fall behind. In addition to the timeline, we also created user stories and user requirements that allowed us to fully grasp what our tasks were and how we should go about completing them. Both user stories and requirements have defined the end goals of our project and encouraged all team members to work toward the completion of those goals



Timeline created based on the Linux KVM Project Plan

Weekly Traffic Light Reports

Introduction

The last step in maintaining a perfect project plan was to hold ourselves accountable every week. By submitting Traffic Light Reports, we acknowledged all of our work completed along with the work in progress. Each report gives us and any viewers a very clear explanation of the events occurring in our project. Due to color coordination, each task, whether in progress or completed, can be easily identified and any further action can be performed based on the status of each item. Viewers can analyze each column to break down the report, starting with the RGY column. This column allows you to label each action under three different categories listed below.

Red→ Indicates a problem regarding a specific item
Green→ Indicates that no problems are present and item is complete
Yellow→ Item is in progress with no major issues

Below is each of our reports that helped guide us towards the completion of this project:

September 18, 2024

RGY	Item	Description	Mitigation/Help Needed
Green	ECRL Lab Server Setup	Received Lab access and configured server settings to begin implementing KVM environment	None Required.
Green	Ubuntu Installation	Ubuntu 24.04.01 installed and configured on server using USB drive	None Required.
Green	VMware Installation	VMware installed using terminal commands	None Required.
Yellow	Ubuntu .iso Download	Downloading Ubuntu .iso file to use on VMware Workstation	Waiting for download process to be completed.

September 25, 2024

RGY	Item	Description	Mitigation/Help Needed
Green	QEMU Installation	Used terminal to install and compile QEMU	None Required.
Green	Libvirt Installation & Configuration	Used terminal to install Libvirt and configure to run on startup	None Required.
Green	VirtManager Installation	Installed VirtManager for hypervisor and for a graphical interface of Libvirt	None Required.
Green	VM Installation & Testing	Installed Ubuntu 24.04.01 and Windows 10 as separate VMs. Also ran both VMs simultaneously to ensure both are functioning properly on the Hypervisor.	None Required.
Yellow	Weekly Check-ins	Setting up weekly meetings with Fernando Pizzano	None Required.
Yellow	QEMU Research	Consistently researching QEMU and some implementation strategies	None Required.

October 2, 2024

RGY	Item	Description	Mitigation/Help Needed
Green	Weekly Check-Ins	Weekly Check-Ins planned with Fernando Pizzano- Every Thursday 6:30 pm	None Required.
Green	Remote Desktop Connection	Set up Remote Desktop Connection to enable us to work outside of the ECRL	None Required.
Yellow	XML Security Profile	Reviewed and began implementing security profiles	None Required.
Yellow	QEMU Research	Consistently researching QEMU and some implementation strategies	None Required.
Yellow	Documentation	Continuously updating and editing project documentation	None Required.

October 9, 2024

RGY	Item	Description	Mitigation/Help Needed
Green	Weekly Check-In	Had meeting with Fernando Pizzano on Thursday October 3rd, and discussed project progress	None Required.
Green	Researched QEMU and Libvirt	Researched QEMU and Libvirt vulnerabilities along with other application information	None Required.
Green	Set Goals and Created New Ideas	Created future plans regarding AI, CVEs, and presentations with the help of Fernando	None Required.
Yellow	Documentation	Continuously updating and editing project documentation	None Required.
Yellow	Tech Meetup	Discussed possibility of attending MHV Tech Meetup at Bard College on November 8th	None Required.

October 16, 2024

RGY	Item	Description	Mitigation/Help Needed
Green	Weekly Check-In	Planned meeting with Fernando for October 17, at 6:30 pm in Hancock	None Required.
Green	Questions for Fernando	Created a list of questions to go over with Fernando during weekly check-in	None Required.
Green	Researched QEMU, Libvirt, and Ubuntu	Continued to research QEMU 8.2.2, Libvirt 10.0.0, and Ubuntu 24.04 vulnerabilities along with other application information	None Required.
Yellow	MHV Tech Meetup	Began brainstorming and formulating ideas for the 200 word abstract required to take part in the MHV Tech Meetup	None Required.
Yellow	Documentation	Continuously updating and editing project documentation	None Required.
Yellow	MHV Tech Meetup	Confirmed our plans of attending the MHV Tech Meetup on November 8th	Will continue to plan transportation and the 200 word abstract.

October 23, 2024

RGY	Item	Description	Mitigation/Help Needed
Green	Weekly Check-In	Met with Fernando Pizzano on Thursday October 17th to discuss progress and steps moving forward	None Required.
Green	MHV Tech Meetup Abstract	Submitted the MHV Tech Meetup Abstract with the help of Professor Casimer DeCusatis	None Required.
Yellow	Researched QEMU, Libvirt, and Ubuntu	Continued to research QEMU 8.2.2, Libvirt 10.0.0, and Ubuntu 24.04 vulnerabilities along with other application information	None Required.
Yellow	MHV Tech Meetup	Continued brainstorming and formulating ideas for the 200 word abstract required to take part in the MHV Tech Meetup	Purchase a poster board and begin putting information on it.
Yellow	Documentation	Continuously updating and editing project documentation	None Required.

October 30, 2024

RGY	Item	Description	Mitigation/Help Needed
Green	Planned Weekly Check-In	Planned weekly check-in with Fernando for Thursday November 7th, at 6:30 pm in Hancock	None Required.
Green	Updated Project Plan	Solidified our plans to be complete with any research, exploiting, and other tasks within the next couple weeks to allow full attention on documentation	None Required.
Green	MHV Tech Meetup	Purchased poster board and have begun to create material	None Required.
Yellow	Researched QEMU, Libvirt, and Ubuntu	Continued to research QEMU 8.2.2, Libvirt 10.0.0, and Ubuntu 24.04 vulnerabilities along with other application information	None Required.
Yellow	MHV Tech Meetup	Continued brainstorming and formulating ideas for the 200 word abstract required to take part in the MHV Tech Meetup	None Required.
Yellow	Documentation	Continuously updating and editing project documentation	None Required.
Red	CVE-2024-8354	Began exploiting CVE-2024-8354: Exploit should send malformed USB requests that lead to an assertion failure in QEMU which will then crash the process causing a DOS on the host.	Attempted on Linux Server and Windows VM. To be exploited, Python 2 needs to be installed in the terminal but, it is outdated and no longer supported

November 6, 2024

RGY	Item	Description	Mitigation/Help Needed
Green	Confirmed Weekly Check-In	Confirmed weekly check-in with Fernando for Thursday November 7th, at 6:30 pm in Hancock	None Required.
Green	Updated Project Plan	Solidified our plans to be complete with any research, exploiting, and other tasks within the next couple weeks to allow full attention on documentation	None Required.
Green	MHV Tech Meetup	Completed poster board and prepared for the MHV Tech Meetup	None Required.
Green	CVE-2024-8354	Completed exploitation attempts for CVE-2024-8354	None Required.
Yellow	Researched QEMU, Libvirt, and Ubuntu	Continued to research QEMU 8.2.2, Libvirt 10.0.0, and Ubuntu 24.04 vulnerabilities along with other application information	None Required.
Yellow	Software Bill of Materials	Started working towards the creation of the SBOM	None Required.
Yellow	Documentation	Continuously updating and editing project documentation	None Required.

November 13, 2024

RGY	Item	Description	Mitigation/Help Needed
Green	Planned Weekly Check-In	Planned weekly check-in with Fernando for Thursday November 21st	None Required.
Green	Pre-Break Checkpoint	Completed Pre-Break Checkpoint, allowing us to map out the rest of the semester	None Required.
Green	MHV Tech Meetup	Participated in the Mid Hudson Valley TechMeet at Bard College on Friday November, 8th	None Required.
Green	Researched QEMU, Libvirt, and Ubuntu	Completed research of QEMU 8.2.2, Libvirt 10.0.0, and Ubuntu 24.04 vulnerabilities along with other application information	None Required.
Green	Software Bill of Materials	Finished the creation of the Software Bill of Materials	None Required.
Yellow	Documentation	Continuously updating and editing project documentation	None Required.
Yellow	Final Report	Began creating and filling in outline for final report	None Required.

November 20, 2024

RGY	Item	Description	Mitigation/Help Needed
Green	Confirmed Weekly Check-In	Confirmed weekly meeting with Fernando for Thursday November, 21	None Required.
Green	Final Report	Created and filled in outline for final report	None Required.
Yellow	Documentation	Continuously updating and editing project documentation	None Required.
Yellow	Final Presentation	Began forming script and finalizing presentation	None Required.

November 27, 2024

RGY	Item	Description	Mitigation/Help Needed
Green	Documentation	Completed final documentation	None Required.
Yellow	Final Presentation	Began forming script and finalizing presentation	None Required.
Yellow	Confirm Check-In	Planning meeting with Fernando for next Tuesday, December 3rd	Send follow up email to Fernando.

December 4, 2024

RGY	Item	Description	Mitigation/Help Needed
Green	Confirmed Check-In	Planned meeting with Fernando for next Monday, December 2nd	None Required.
Green	Documentation	Completed final documentation	None Required.
Green	Final Presentation	Presented our Project on December 4th, at 8:15 am	None Required.
Green	Final Video Presentation	Recorded final video presentation	None Required.

December 11, 2024

RGY	Item	Description	Mitigation/Help Needed
Green	Final Reflection	Completed project Final Reflection	None Required.
Green	Peer Review	Completed project Peer Review	None Required.

Infrastructure Design Diagrams

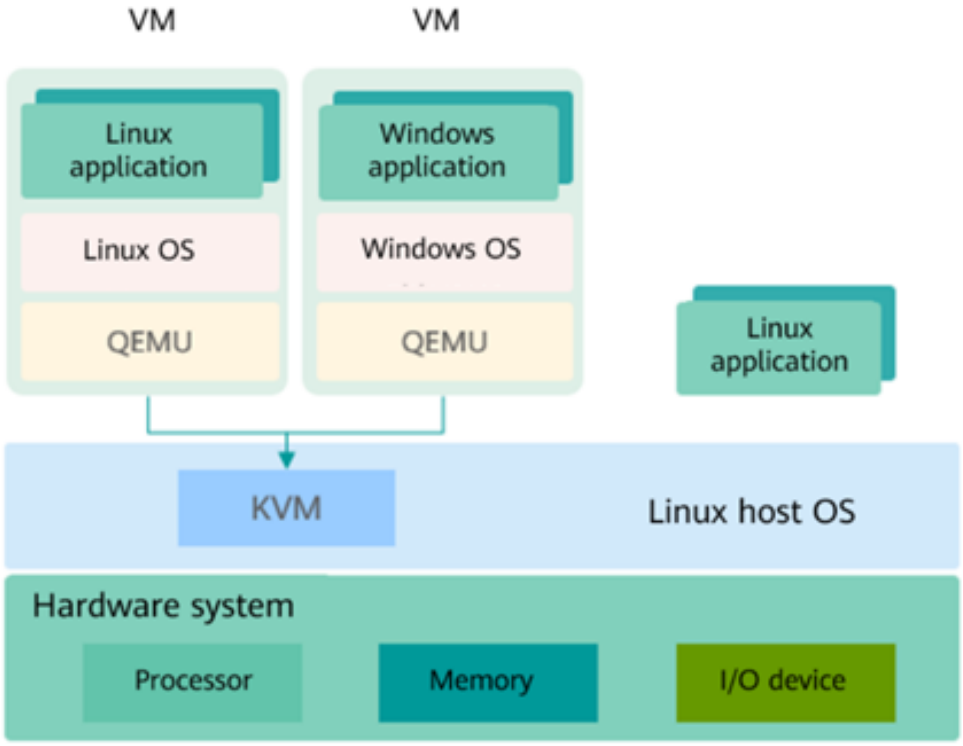


Diagram 1

Diagram 1 shows specifically how the KVM hypervisor manages virtual machines on a Linux-based host system. This diagram illustrates each VM running a different operating system. In our case, it is Ubuntu Linux 24.04 and Windows 10. Because of the virtualization setup in the environment, the operating systems running on each VM have full access to the hardware of the server they are set up on. The QEMU 8.2.2 hardware virtualization emulator is crucial to our setup. QEMU emulates the hardware layer it is running on for the operating systems to use. In our environment, we also have Libvirt 10.0.0 set up to run on startup to simplify the creation, configuration, and general monitoring of the VMs. It works with KVM to manage resources and QEMU to configure and launch the guest VMs.

The Linux host OS in this model uses the KVM module within the Linux kernel to allocate and manage resources for the VMs. This runs directly on the hardware and interacts with the processor, memory, and I/O devices. The resources available through the hardware system are made available and allocated out to the VMs through the Linux kernel and KVM.

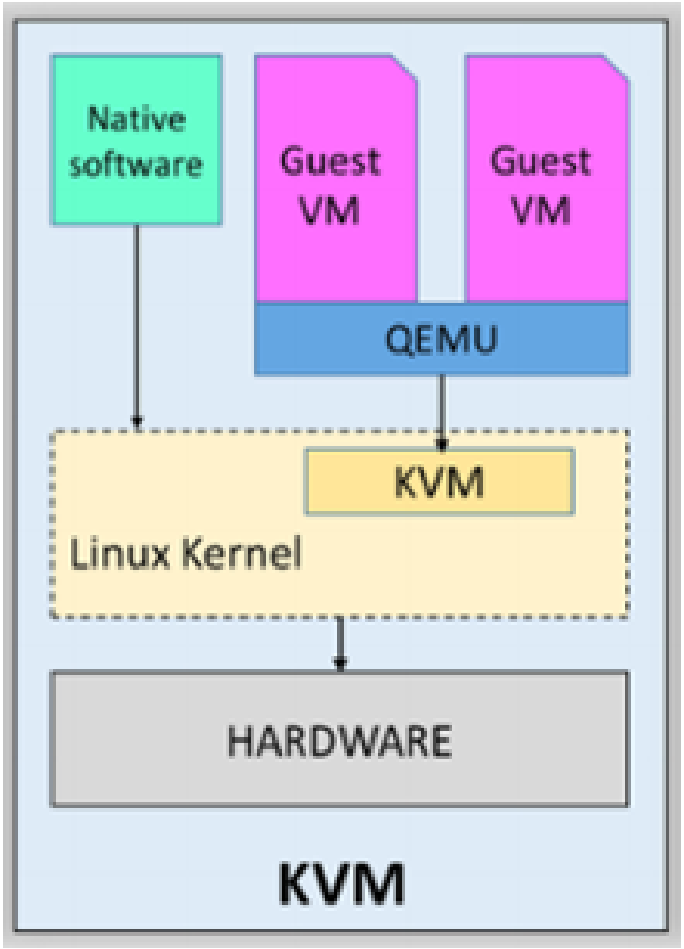


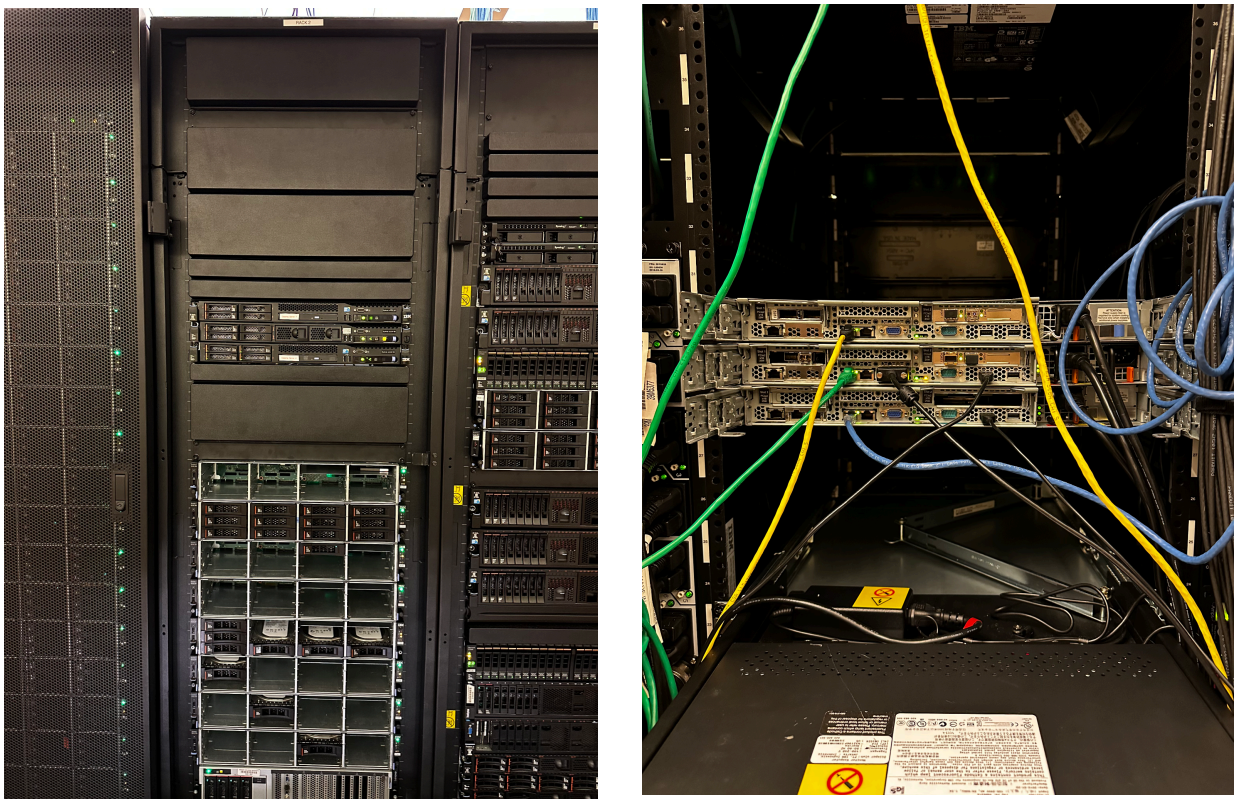
Diagram 2

Diagram 2 presents a more compact overview of the relationship built between the hardware, the Linux kernel, and VMs. The guest VMs run on top of the QEMU emulator, which are their own isolated systems with their own OSs, applications, and resources allocated through the hypervisor. Resources from the hardware layer are controlled by the kernel and are spread to the guests.

Deployment

Installation of Initial Operating System

At the start of the semester, we were immediately faced with our first task of the project which was installing our operating system. Although it seemed to be a quick and easy task, its importance was extremely significant. The operating system installed on the x86 IBM server, assigned by Professor Chris Algozzine, would go on to serve as the backbone and foundation of our project. Using deployment documentation created by the previous Linux KVM Security capstone team, we were easily able to setup the server making only a few adjustments. The first step in the installation process was to download Ubuntu 24.04 on a flash drive from the Ubuntu website. From there, we inserted the drive into the server and booted it up entering the BIOS. Next, we selected our drive which rebooted our server and began the installation of Ubuntu where we continued the deployment of our operating system. Throughout the installation, we were requested to select our language and keyboard layout along with network and storage configurations. The last step was to then create a user account and complete installation. Once installation was finally completed, we were then able to fully focus on future steps such as virtualization and hypervisor setup.



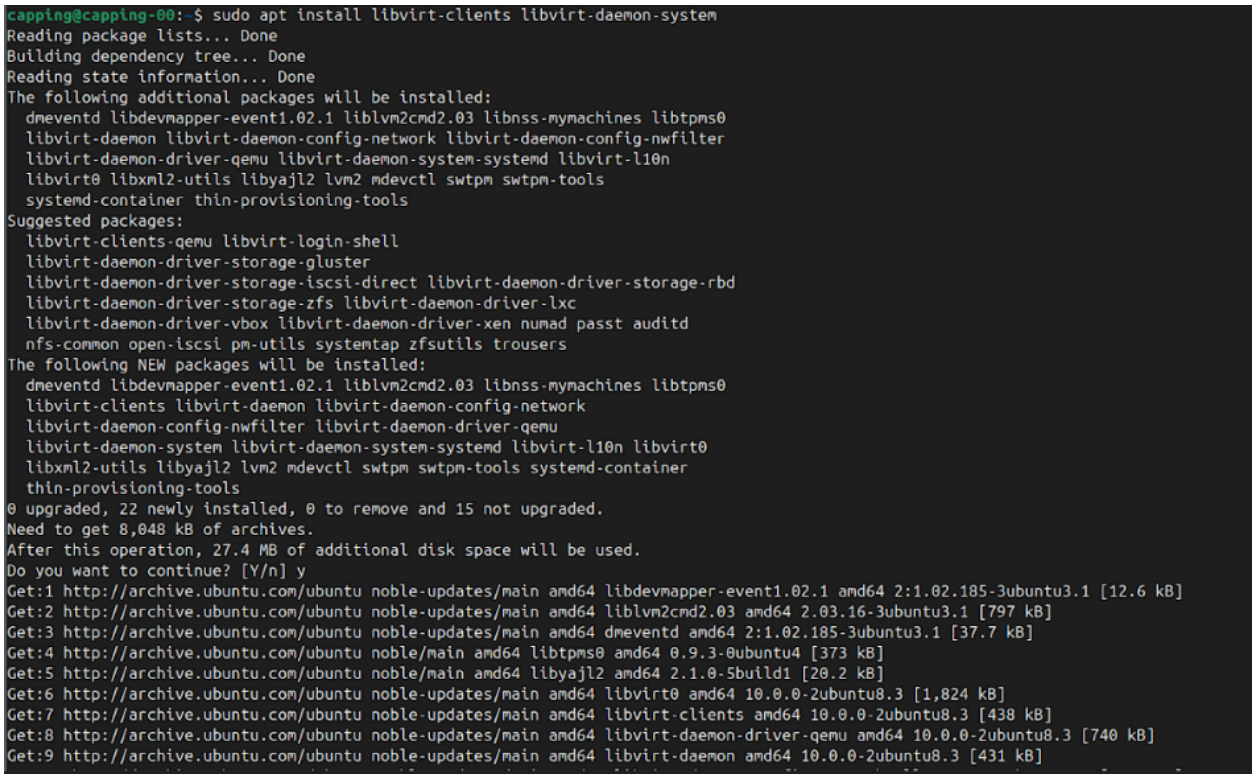
Images of our x86 IBM server in the Marist College Enterprise Computing Research Lab (ECRL)

Virtualization and Hypervisor Setup

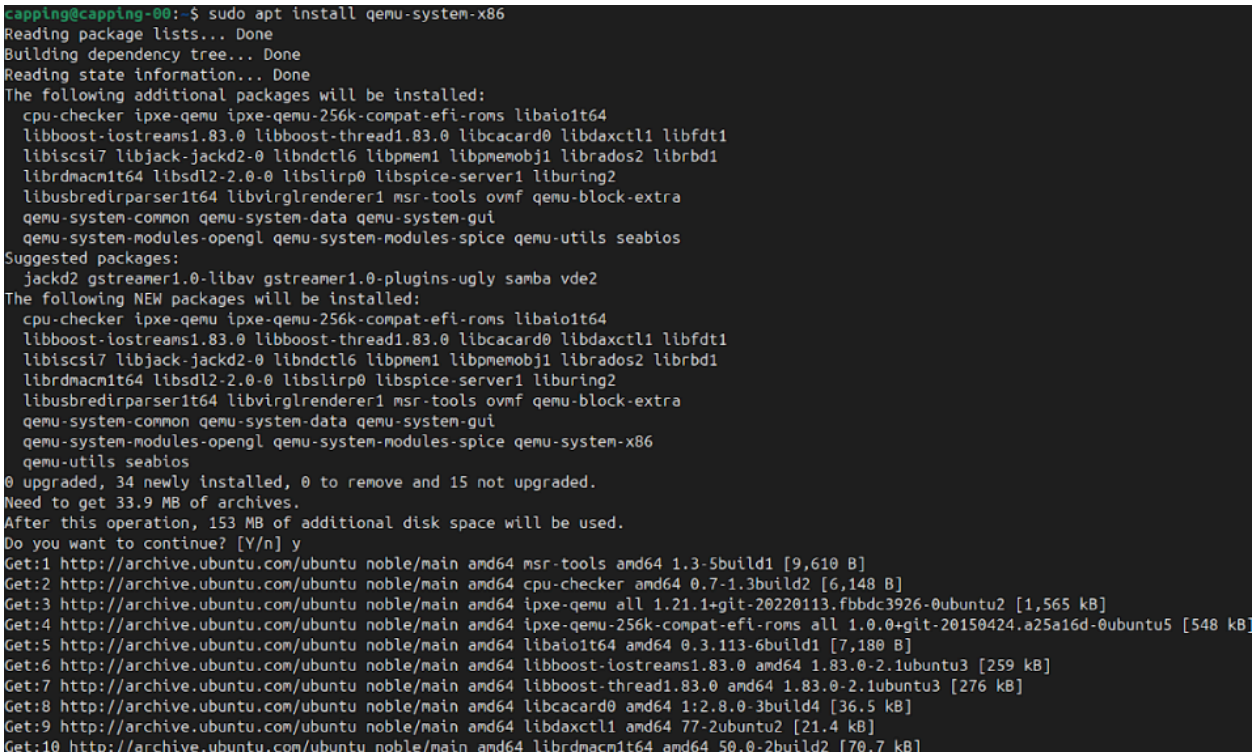
The next step of our project was to get our hypervisor set up so we could then install and set up our Virtual Machines. Rather than a software application like VMWare for our hypervisor, we went with a specific software stack of QEMU and Libvirt as our hypervisor. We installed QEMU and Libvirt through the Linux Bash terminal, the specific versions automatically installed to be compatible with our version of Ubuntu 24.04. QEMU was installed as version 8.2.2 and Libvirt was installed as version 10.0.

In our software stack, QEMU is the primary emulator. QEMU allows full hardware virtualization and lets us run the guest operating systems that we wish. QEMU has its own CPU emulation and in turn lets us run guest operating systems which may have different architecture from that of our host. Libvirt handles more of the managing side of our virtualization process. Libvirt acts as a library that provides a high level API for managing virtualization technologies, like QEMU. The two work together to provide us with a smooth virtualization experience to conduct our project.

The process of installing QEMU and Libvirt onto our Ubuntu Server was relatively straightforward. We began with the installation of Libvirt by running the command ‘sudo apt install libvirt-clients libvirt-daemon-system’ in the Linux bash terminal.

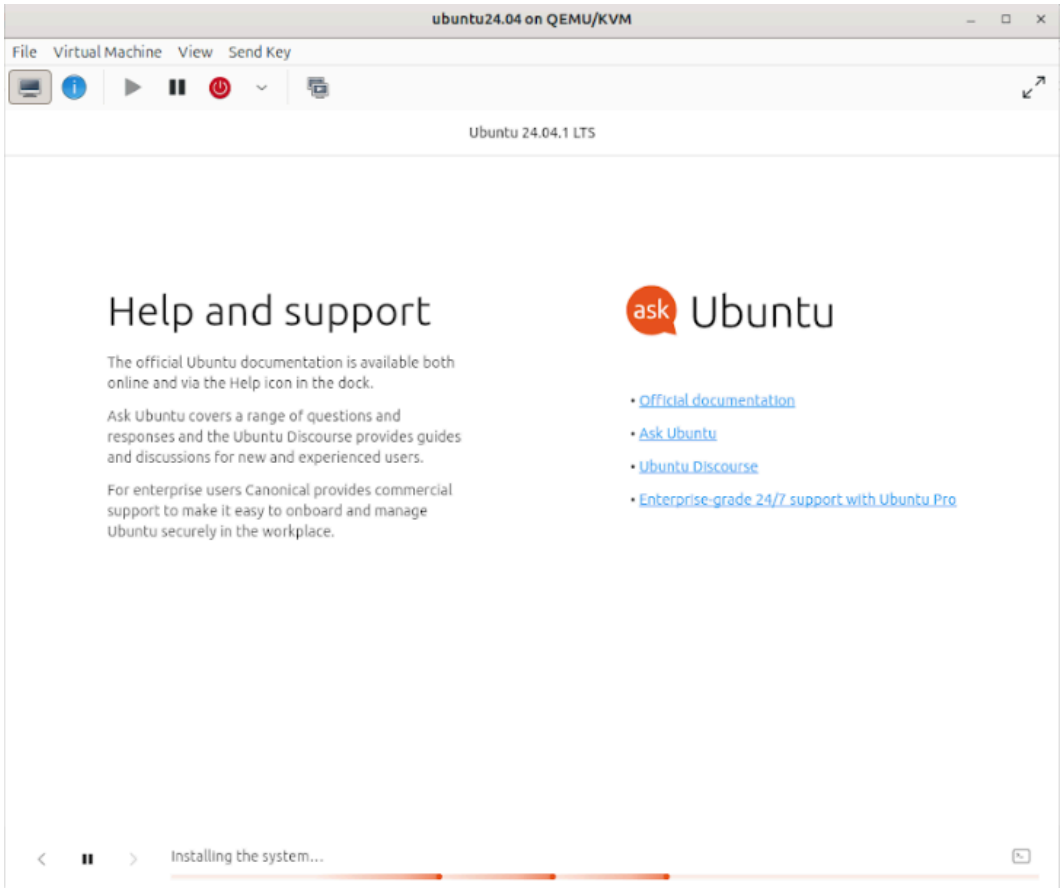


After Libvirt successfully installed, we moved onto the process of installing QEMU. To install QEMU we ran the command ‘sudo apt install qemu-system-x86’ in the Linux bash terminal.

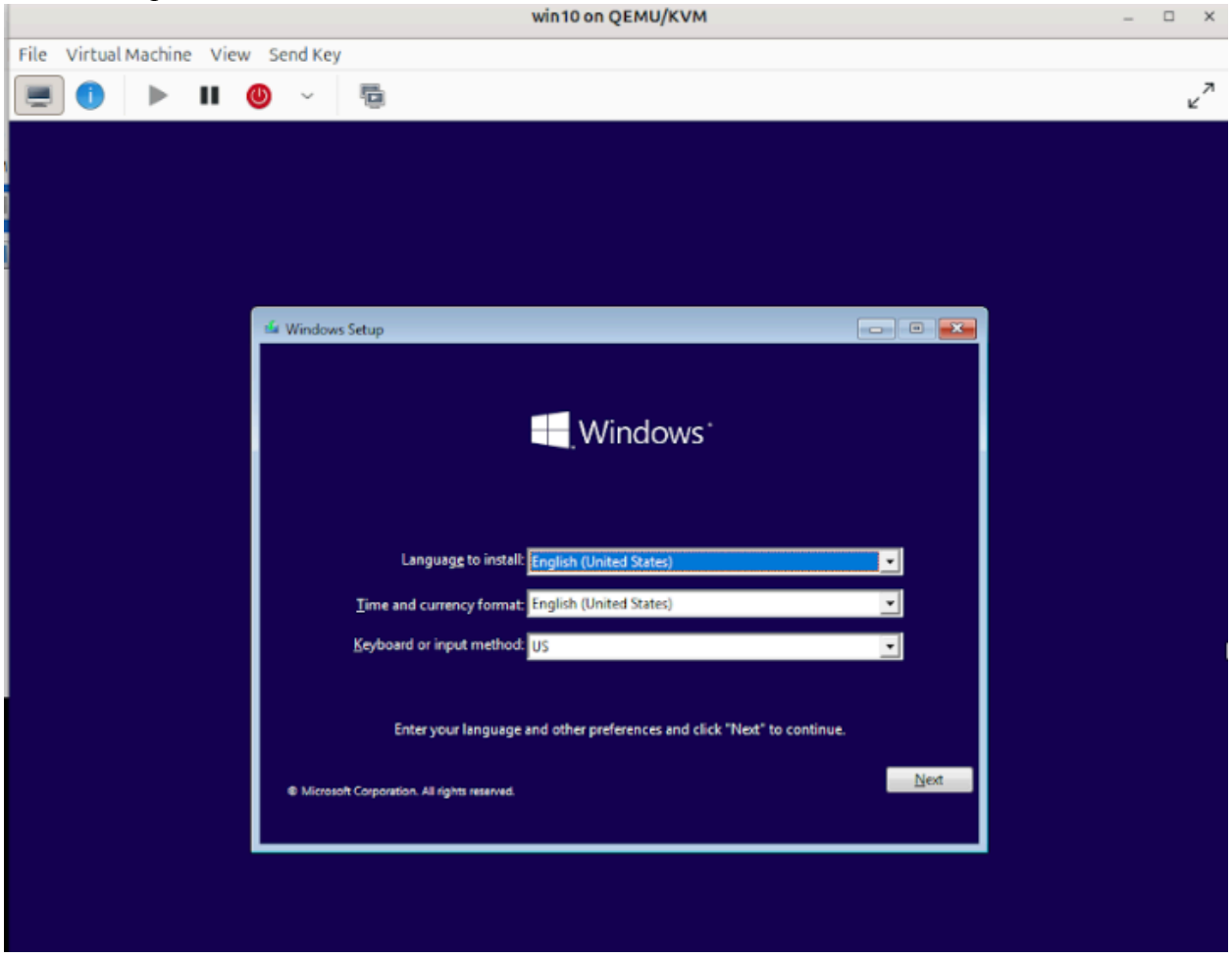


Now that we had successfully installed both QEMU and Libvirt, we ran the command ‘sudo systemctl enable libvirtd’ in order to ensure that Libvirt is always running when the system is booted up. We then installed our final application which is ‘Virt-Manager’, simply a user interface for Libvirt so we could work with our Virtual Machines aside from the command line. That was done by running the command ‘sudo apt-get install virt-manager’ in the Linux bash terminal.

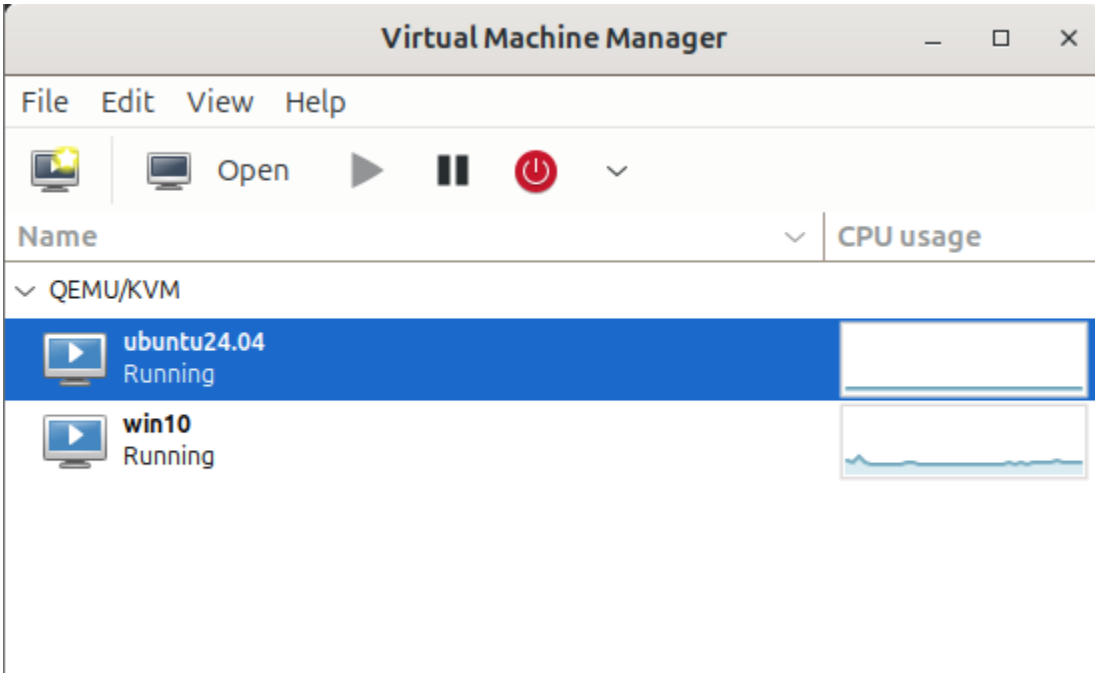
Our full software stack was now installed on our Ubuntu Server and the process of creating our Virtual Machines could begin. We started with the creation of the Ubuntu 24.04 VM. We installed the ISO file for Ubuntu and created the VM.


ubuntu24.04 on QEMU/KVM
File Virtual Machine View Send Key
Ubuntu 24.04.1 LTS
Help and support
The official Ubuntu documentation is available both online and via the Help icon in the dock.
Ask Ubuntu covers a range of questions and responses and the Ubuntu Discourse provides guides and discussions for new and experienced users.
For enterprise users Canonical provides commercial support to make it easy to onboard and manage Ubuntu securely in the workplace.
ask Ubuntu
• Official documentation
• Ask Ubuntu
• Ubuntu Discourse
• Enterprise-grade 24/7 support with Ubuntu Pro
Installing the system...

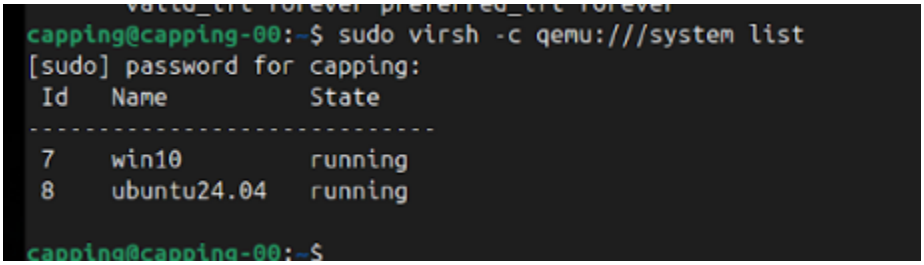
We then went on to create the Windows 10 VM. We installed the ISO file for Windows 10 and set up the VM as shown in the screenshot below.



After both VMs had been created and were up and running, we were able to see the status in the Virtual Machine Manager menu.



To verify further that both VMs were running properly we issued the command ‘sudo virsh -c qemu:///system list’ to display the status of our current VMs in the bash terminal.



Now we had our Server setup with Ubuntu 24.04 installed as the OS, QEMU and Libvirt installed, and both of our VMs installed and running. The setup of our virtualization environment was complete and we could move on to the next steps.

Common Vulnerabilities and Exposures

Overview

The term CVE (Common Vulnerabilities and Exposures), is a term used for security vulnerabilities that are found in either the hardware or software of a system. They can be caused by things like coding errors, outdated software, misconfigurations, and much more. To maintain an organized list of CVEs, they are each assigned an ID and have a description to go along with them. CVEs are assigned an ID which consists of the year it was discovered and a random ID number following it. For example, one of the CVEs we researched was assigned the ID of CVE-2024-8354, where ‘2024’ was the year it was discovered and ‘8354’ is the random ID number. The MITRE organization is in charge of maintaining the list of CVEs, they are a non-profit organization and receive government funding for research and development centers.

The work that MITRE does with maintaining and updating the list of CVEs is crucial to the field of Cybersecurity. Cyber Criminals and attackers use vulnerabilities for malicious purposes such as stealing or deleting sensitive data, installing malware, executing code, and many more. When it comes to exposures, they are slightly different. While vulnerabilities are a flaw in software code that allows attackers to gain access to a system and act as a legitimate user. Exposures are a mistake in software code or configuration which allows an attacker to gain access to a system or network. Exposures typically lead to things like data breaches, data leaks, and the sale of personally identifiable information (What is a CVE?, n.d.).

The list of CVEs by MITRE serves multiple purposes in the field of cybersecurity. For one thing, it allows for a standardized way to keep track of and reference vulnerabilities and exposures. This organization provided by the standard of the CVE list allows for smooth communication between groups when discussing different vulnerabilities. Another reason why the list of CVEs is so important is the fact that it's all centralized in one location. Rather than there being many different sources and webpages to look through to find a certain vulnerability, it is all kept in one centralized place. Along with those reasons, it is obvious that a thorough list of vulnerabilities can help to reduce attack surfaces of organizations.

Classification

Every CVE is assigned a score which is determined by the Common Vulnerability Scoring System, also known as the CVSS. The CVSS uses the numbers 0 to 10 to determine the severity of a CVE, with 0 meaning no risk and 10 being critical. The full breakdown of the CVSS scores is shown in the following table:

CVSS Score	Severity
0.0	None
0.1-3.9	Low
4.0-6.9	Medium
7.0-8.9	High
9.0-10.0	Critical

There are several different aspects which are examined in order to determine the CVSS assigned to any given CVE. There are three sets of metrics which determine the CVSS that is assigned to a CVE, these include: the Basic Metric Group, the Temporal Metric Group, and the Environmental Metric Group. These different Metric Groups can be broken down even further. The Basic Metric Group is broken down into two individual sections, one being Exploitability Metrics and the other being Impact Metrics. The Exploitability Metrics section contains several different factors including: Attack Vector, Attack Complexity, Privileges Required, User Interaction, and Scope. The Impact Metrics section also contains several factors which include: Compatibility Impact, Integrity Impact, Availability Impact, and Scope. Now the other two Groups only contain one section of factors each. The Temporal Metric Group contains the following: Exploit Code Maturity, Remediation Level, and Report Confidence. The Environmental Metric Group contains the following: Confidentiality Requirement, Integrity Requirement, Availability Requirement, and Modified Base Metrics. The Basic Metric Group essentially focuses on the overall characteristics of the vulnerability. The Temporal Metric Group essentially focuses on the current exploitability of a vulnerability and how available controls like patches are. Finally, the Environmental Metric Group is a section that allows an organization to adjust the base CVSS Score depending on their Security Requirements. (Goodman, 2024)

CVE-2024-8354

The vulnerability CVE-2024-8354 was found while we researched known CVEs which still affect the QEMU version 8.2.2. It was the only significant vulnerability found that was still unpatched, so we would be able to try and exploit it on our system.

CVE-2024-8354 is a vulnerability that is present when passing through a USB device to a VM and sending certain requests when accessing it. It is more specifically present in a certain function involved in the USB process within QEMU 8.2.2, this would be the ‘usb_ep_get()’ function. It can be exploited by writing code that takes advantage of the vulnerability and executing the code in the Linux Bash terminal on your VM. A successful exploit of this vulnerability on the VM would result in a denial of service (DoS), and would cause the QEMU process to crash.

After proper research was done on the vulnerability and we gained more knowledge, we would then begin the process of how exactly we would test out this vulnerability and exploit it on our system. First, we would have to have a USB device plugged into our system and pass it through to our VM. We used a Lenovo Mouse for this step, we plugged it into our Server and began finding out how we could exactly pass the USB through to our VM. There were several ways we found in which this could be accomplished, the way that worked the best for us was adding a section of code into our XML configuration file on our VM.

The following code was what we added into our XML configuration file, under the devices section:

```
<hostdev mode='subsystem' type='usb'>
  <source>
    <address bus='3' device='3'/>
  </source>
</hostdev>
```

In the following screenshot, the code added to our XML configuration file is displayed:

```
GNU nano 7.2                    /tmp/virshU1RYW2.xml *
  </controller>
  <controller type='virtio-serial' index='0'>
    <address type='pci' domain='0x0000' bus='0x03' slot='0x00' function='0x0'
  </controller>
  <interface type='network'>
    <mac address='52:54:00:0b:69:be'/>
    <source network='default'/>
    <model type='virtio'/>
    <address type='pci' domain='0x0000' bus='0x01' slot='0x00' function='0x0'
  </interface>
  <hostdev mode='subsystem' type='usb'>
  <source>
      <address bus='7' device='7'/>
  </source>
  </hostdev>
  <serial type='pty'>
    <target type='isa-serial' port='0'>
      <model name='isa-serial'/>
    </target>
  </serial>
```

Now that we added the ability to pass through a USB device from our physical server to our VM we were able to continue with the next steps of exploiting our system. To exploit this we needed to create code that would send requests to the USB device on our VM and hopefully exploit the vulnerability and crash the QEMU process. A template of the code was created, and from there we had to replace certain values with ones specific to our USB device. The Vendor ID and Product ID of the USB device was required in the code to properly identify it. To discover these values the command ‘lsusb’ was used. As displayed in the screenshot below:

```
ubuntuvm1@ubuntuvm1-Standard-PC-Q35-ICH9-2009:~$ lsusb
Bus 001 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
Bus 001 Device 002: ID 0627:0001 Adomax Technology Co., Ltd QEMU Tablet
Bus 001 Device 003: ID 17ef:6019 Lenovo M-U0025-O Mouse
Bus 002 Device 001: ID 1d6b:0003 Linux Foundation 3.0 root hub
ubuntuvm1@ubuntuvm1-Standard-PC-Q35-ICH9-2009:~$
```

The command was executed on the Ubuntu VM, which shows the successful passthrough of our USB device, the Lenovo Mouse, from the Server to the VM. We can also see the Vendor ID and the Product ID from each USB device, we then took those values and implemented them into our code for the exploit.

The following code was what we used to attempt and exploit the vulnerability:

```
‘#include <libusb-1.0/libusb.h>
#include <stdio.h>
#include <stdlib.h>

#define TIMEOUT 1000 // Timeout for control transfer in milliseconds

int main() {
    libusb_device_handle *handle;
    int result;

    // Initialize the libusb library
    result = libusb_init(NULL);
    if (result < 0) {
        fprintf(stderr, "Failed to initialize libusb: %s\n", libusb_error_name(result));
        return EXIT_FAILURE;
    }

    // Open the USB device with Vendor ID 0x17ef and Product ID 0x6019
    handle = libusb_open_device_with_vid_pid(NULL, 0x17ef, 0x6019);
    if (handle == NULL) {
        fprintf(stderr, "Failed to open USB device\n");
        libusb_exit(NULL);
        return EXIT_FAILURE;
    }

    uint8_t bmRequestType = LIBUSB_REQUEST_TYPE_STANDARD |
LIBUSB_ENDPOINT_OUT; //
uint8_t bRequest = 0x01;      // Malformed request type (replace with actual values based on
CVE)
uint16_t wValue = 0xFFFF;      // Malformed value (replace with actual values based on CVE)
uint16_t wIndex = 0x0000;      // Typically the first interface or endpoint

result = libusb_control_transfer(handle, bmRequestType, bRequest, wValue, wIndex, NULL, 0,
TIMEOUT);
    if (result < 0) {
        fprintf(stderr, "Control transfer failed: %s\n", libusb_error_name(result));
    } else {
        printf("Control transfer succeeded\n");
    }

    // Close the USB device and cleanup
    libusb_close(handle);
    libusb_exit(NULL);
    return EXIT_SUCCESS;
```

}’

In order to fully test the exploit we had to run the code multiple times but with different combinations of values for the following variables: bRequest, wValue, and wIndex. In the table below the values we tested are listed:

Variable	Value1	Value2	Value3
bRequest	0x01	0xFF	0x03
wValue	0xFFFF	0x0001	0x7FFF
wIndex	0x0000	0x0001	0x00FF

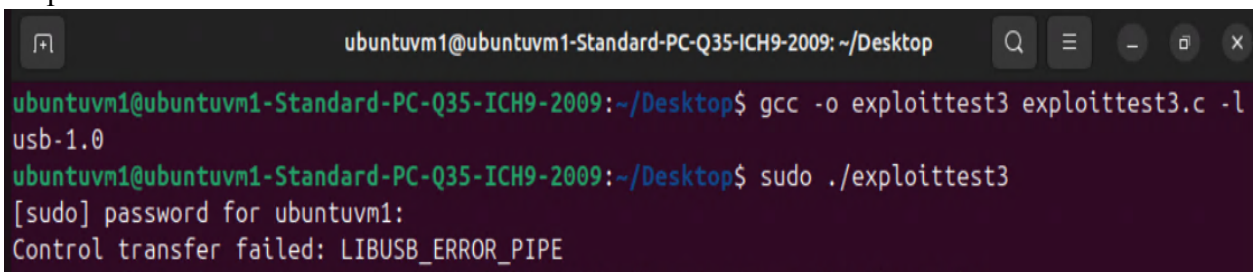
With combinations of these values we were able to try different types of interactions with the USB device to try and exploit the vulnerability. The next step was to create a ‘C’ file of our code on our VM, this is displayed in the following screenshot:

Now that we have everything set up, we could begin testing our code in an attempt to exploit the vulnerability.

In order to make the code executable we had to first compile it with the command:
‘gcc -o exploittest3 exploittest3.c -lusb-1.0’

To break down this command further:
‘gcc’ - This is used to compile C programs
‘-o exploittest3’ - This specifies the name of the output file
‘exploittest3.c’ - This specifies the source code file to compile
‘-lusb-1.0’ - This indicates that the library libusb-1.0 is to be linked

In the screenshot below the compilation command is run first, then the command ‘sudo ./exploittest3’ is run to execute the code.



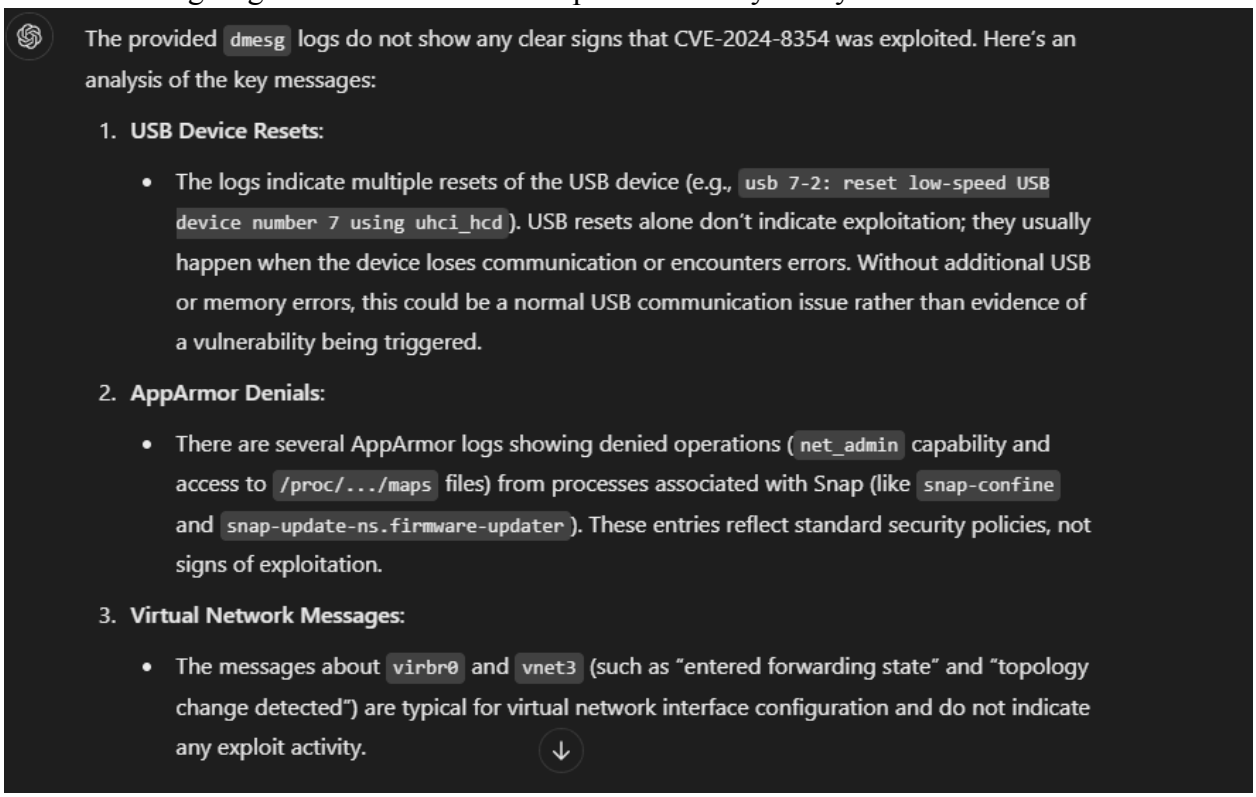
Once the exploit code was ran the following error was received as seen in the screenshot:
‘Control transfer failed: LIBUSB_ERROR_PIPE’
After attempting to run the code with several different combinations for the values mentioned before, the same error still appeared. At this point we investigated the system logs in order to see what was going on and if the vulnerability had successfully been exploited. The ‘dmesg’ log was the primary log analyzed.

The screenshot below displays a section of the ‘dmesg’ log after we attempted to exploit the vulnerability.

```
[1765325.715490] audit: type=1400 audit(1732165207.276:549): apparmor="DENIED" operation="capa
ble" class="cap" profile="/usr/sbin/cupsd" pid=81304 comm="cupsd" capability=12  capname="net_
admin"
[1765371.798095] audit: type=1400 audit(1732165253.359:550): apparmor="DENIED" operation="capa
ble" class="cap" profile="/usr/lib/snapd/snap-confine" pid=81352 comm="snap-confine" capabilit
y=12  capname="net_admin"
[1765371.812018] audit: type=1400 audit(1732165253.373:551): apparmor="DENIED" operation="open
" class="file" profile="snap-update-ns.firmware-updater" name="/proc/81364/maps" pid=81364 com
m="5" requested_mask="r" denied_mask="r" fsuid=1000 ouid=0
[1765371.971865] audit: type=1400 audit(1732165253.533:552): apparmor="DENIED" operation="open
" class="file" profile="snap.firmware-updater.firmware-notifier" name="/proc/sys/vm/max_map_co
unt" pid=81352 comm="firmware-notifi" requested_mask="r" denied_mask="r" fsuid=1000 ouid=0
[1776171.639039] audit: type=1400 audit(1732176053.395:553): apparmor="DENIED" operation="open
" class="file" profile="snap-update-ns.firmware-updater" name="/proc/82181/maps" pid=82181 com
m="5" requested_mask="r" denied_mask="r" fsuid=1000 ouid=0
[1776171.828553] audit: type=1400 audit(1732176053.586:554): apparmor="DENIED" operation="open
" class="file" profile="snap.firmware-updater.firmware-notifier" name="/proc/sys/vm/max_map_co
unt" pid=82169 comm="firmware-notifi" requested_mask="r" denied_mask="r" fsuid=1000 ouid=0
[1786971.396073] audit: type=1400 audit(1732186853.350:555): apparmor="DENIED" operation="open
" class="file" profile="snap-update-ns.firmware-updater" name="/proc/82493/maps" pid=82493 com
m="5" requested_mask="r" denied_mask="r" fsuid=1000 ouid=0
[1786971.527277] audit: type=1400 audit(1732186853.480:556): apparmor="DENIED" operation="open
" class="file" profile="snap.firmware-updater.firmware-notifier" name="/proc/sys/vm/max_map_co
unt" pid=82481 comm="firmware-notifi" requested_mask="r" denied_mask="r" fsuid=1000 ouid=0
[1797771.192531] audit: type=1400 audit(1732197653.343:557): apparmor="DENIED" operation="open
" class="file" profile="snap-update-ns.firmware-updater" name="/proc/82939/maps" pid=82939 com
m="5" requested_mask="r" denied_mask="r" fsuid=1000 ouid=0
```

We noticed several interesting things about the log here, such as the AppArmor ‘DENIED’ messages. However we were unable to tell by these logs if the vulnerability was successfully exploited. The next step was to utilize AI to analyze the system logs and determine if there was any indication that the vulnerability was exploited.

The full ‘dmesg’ log from each test of the exploit was analyzed by ChatGPT.



The screenshot above is the analysis of the logs from ChatGPT. Unfortunately there was not a successful exploit of the vulnerability, even though several different combinations for the input values in the code were used. Regardless of the unsuccessful exploit, it is still useful to see how ChatGPT was able to examine the system logs for us and feed us back this information. ChatGPT was able to tell us that there were multiple resets of the USB device found in the logs. Even though that does not directly mean a successful exploitation, we were able to learn that there was interaction to some extent with the USB device that was passed through to the VM.

Final Analysis

CVE-2024-8354 remains unpatched to this day on QEMU 8.2.2. Regardless of the unsuccessful exploitation of it on our system, it is still a significant vulnerability for host machines running QEMU 8.2.2. The vulnerability is specifically present in the function ‘usb_ep_get()’ which is used when interacting with USB devices that are passed through to the VM. If an attacker was to have access to the VM, they could have the ability to abuse this vulnerability and cause a Denial of Service on QEMU, effectively stopping the QEMU process in its tracks.

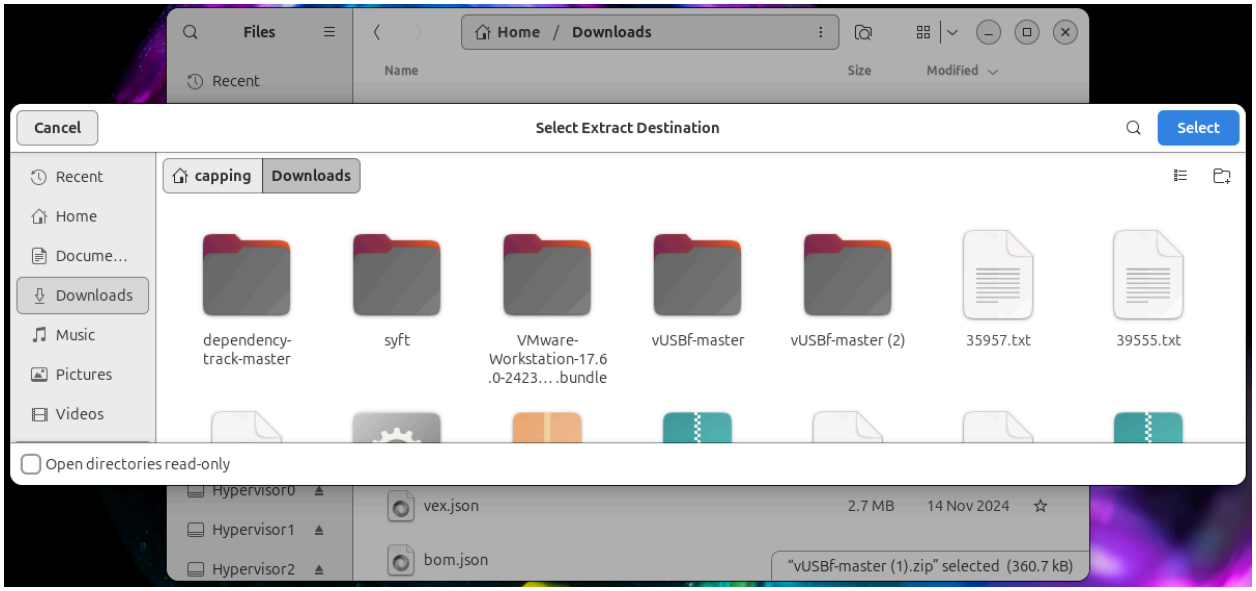
CVE-2016-2184

Inside of our VM environment, we tested CVE-2016-2184, which is a security vulnerability associated directly with the kernel surrounding IPsec (Internet Protocol Security). This vulnerability is classified as a denial-of-service (DoS) exploit due to improper handling of certain conditions in the kernel's networking components. The xfrm subsystem manages the IPSec state and policies, and is unfortunately where the issue lies. A flaw lies in the handling of the packets allowing the kernel to hang or crash, resulting in a DoS condition. Improper validation of user-supplied data or improper handling of certain edge cases in the xfrm_aevent_net() function results in NULL pointer dereferences when processing malformed packets.

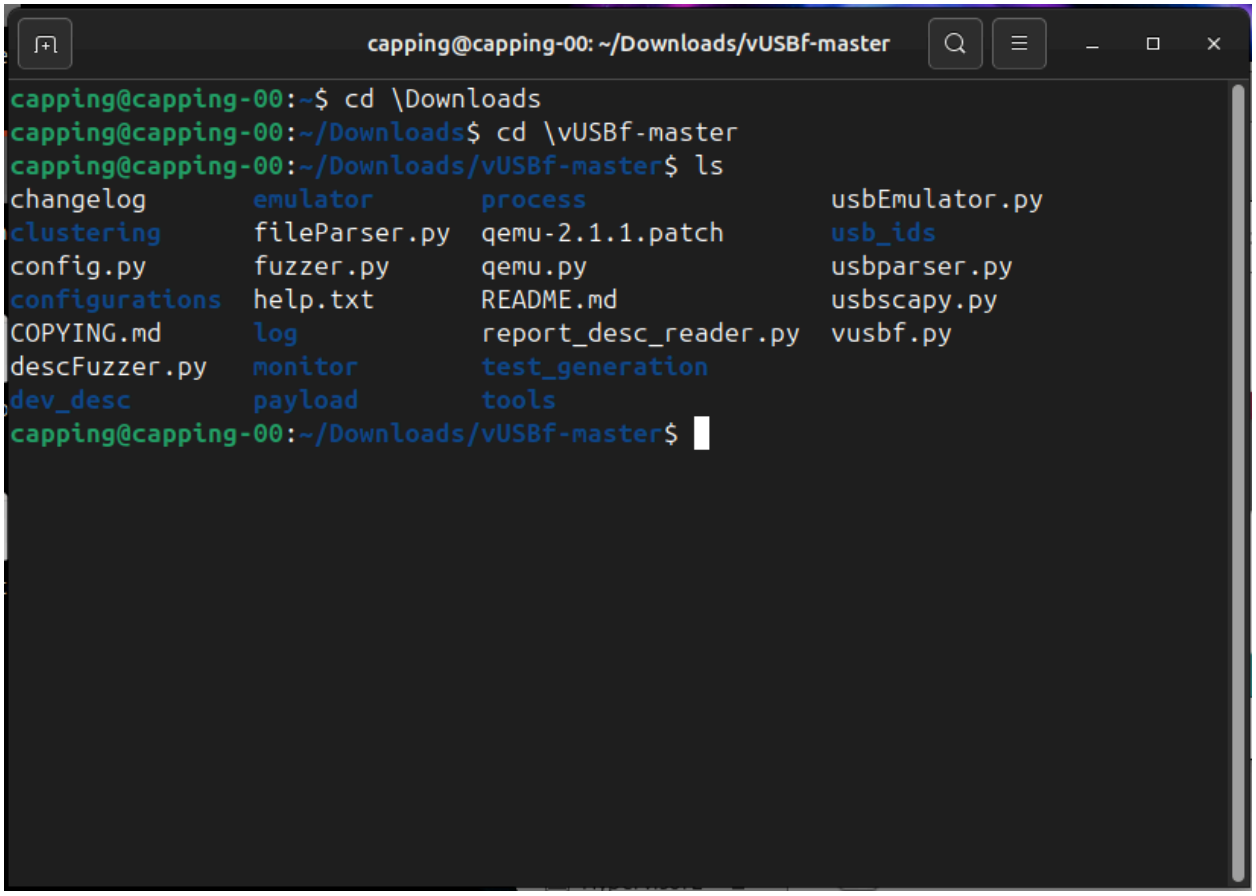
Exploiting this vulnerability can cause the system to become unresponsive, taking the system offline. If a user were to send malicious IPsec packets over the network, they could conduct this vulnerability remotely, but we decided to simulate it locally.

Utilizing a public vUSBf zip file provided by *schumilo* on GitHub, we were able to simulate the implementation of this CVE. The GitHub repository provided us with a USB-fuzzer, which tests USB device drivers and stacks for security flaws by sending malformed or unexpected data to simulate malicious devices. This approach uncovers vulnerabilities in how operating systems and software handle USB interactions.

After downloading the vUSBf-master.zip file from GitHub, the file was unzipped to the downloads folder on our Ubuntu 24.04 system.



We opened the terminal application and navigated to the downloads folder. Using the ls command, we showed the filenames and directories inside of the vUSBf-master directory.



To start the fuzzing process, we ran the **python vusbf.py -l** command because we have Python 3 installed on our system. This command would run the vusbf.py script and list all predefined or available payloads that can be sent to the target system. These payloads represent crafted USB data structures or configurations designed to test specific vulnerabilities. This step helps choose a payload for further testing.

```
capping@capping-00:~/Downloads/vUSBf-master$ python3 vusbf.py -l
/home/capping/Downloads/vUSBf-master/vusbf.py:29: SyntaxWarning: invalid escape
sequence '\ '
  splash += " \ \ / /| || '__|| __|| | | | / _` || | | | | |/ __|| '_ \ \n"
/home/capping/Downloads/vUSBf-master/vusbf.py:30: SyntaxWarning: invalid escape
sequence '\ '
  splash += "  \ V / | || |   | |_ | |_| || (_| || | | |_| |\__ \| |_) |\n"
/home/capping/Downloads/vUSBf-master/vusbf.py:31: SyntaxWarning: invalid escape
sequence '\_'
  splash += "   \_/  |_||_|    \__| \__,_| \__,_||_|  \__,_||___/|_.__/ \n"
/home/capping/Downloads/vUSBf-master/vusbf.py:34: SyntaxWarning: invalid escape
sequence '\|'
  splash += "  | |_ | | | ||_  /|_  // _ \| '__|\n"
/home/capping/Downloads/vUSBf-master/vusbf.py:36: SyntaxWarning: invalid escape
sequence '\_'
  splash += "  |_|   \__,_|/___|/___|\___||_|   \n"
/home/capping/Downloads/vUSBf-master/vusbf.py:40: SyntaxWarning: invalid escape
sequence '\_'
  splash += "| | / / / ___/ __/ / / / __ `/ /  / / / /\__ \/ __  |\n"
/home/capping/Downloads/vUSBf-master/vusbf.py:42: SyntaxWarning: invalid escape
sequence '\_'
  splash += "|_______/   \__/\__,_/\__,_/_/   \____//____/_____/  \n"
/home/capping/Downloads/vUSBf-master/vusbf.py:44: SyntaxWarning: invalid escape
sequence '\/'
  splash += "  / /_/ / / /_  /_  / / _ \/ ___/ \n"
/home/capping/Downloads/vUSBf-master/vusbf.py:46: SyntaxWarning: invalid escape
sequence '\_'
  splash += "/_/  \__,_/ /___/___/\___/_/ \n"
  File "/home/capping/Downloads/vUSBf-master/vusbf.py", line 107
    print "EXECUTE OBJECT MODE (NETWORK)"
    ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
SyntaxError: Missing parentheses in call to 'print'. Did you mean print(...)?
capping@capping-00:~/Downloads/vUSBf-master$
```

Since the script is in Python 2 and we are working with Python 3, the terminal window throws a list of syntax errors. Unfortunately, as of January 1, 2020, support for Python 2 officially ended and was fully replaced with Python 3, which was introduced in 2008. This means we cannot conventionally download Python 2 on our system, but it can be formed to use it for legacy purposes, such as an older system or software still running it.

We still simulated the execution of the fuzzer, and have listed the next steps and commands that would be used in carrying out a USB DoS attack.

1. **python vusbf.py -eon 127.0.0.1 1235 panic_1.obj**
    a. **-eon** specifies that a payload is being sent to an external QEMU instance over the network.
    b. The payload chosen in this example to be sent is **panic_1.obj**. There are many payloads available inside of the initial vUSBf-master folder.

2. **python vusbf.py -eo panic_1.obj -o ubuntu1404.config -v1**
    a. **-eo** specifies the payload file being executed (**panic_1.obj**)
    b. **-o** ubuntu1404.config provides a config file for the test environment. In this example, it is a VM running Ubuntu 14.04, but use whatever OS you have running on your target VM.
    c. **-v1** specifies verbosity level, which in this case, is basic output. This should display information about the payload being sent and the system's response.

    This command executes the specified payload in the configured virtual environment, simulating the USB device within the system.

3. **python vusbf.py -sp 127.0.0.1 1235 -e ex2**
    a. **-sp 127.0.0.1 1235** Sends the payload to an external VM located at 127.0.0.1:1235.
    b. **-e ex2** Executes the payload.

    This command runs single-core mode, restricting the USB fuzzing operation to a single CPU core, simplifying analysis and ensuring consistency.

4. **python vusbf.py -r -e ex1 -o ubuntu1404.config -rl**
    a. **-r** enables the single-core mode for testing.
    b. **-rl** specifies that the fuzzing script should be run in a repeat loop.

    This command runs the fuzzing operation in a single-core mode entirely within the local virtual environment.

5. **python vusbf.py -rm -p 20 -e ex1 -o ubuntu1404.config -rl**
    a. **-rm** enables multi-core mode for testing.
    b. In this example, **-p 20** runs the test across 20 parallel processes.

    This mode leverages multi-core processing to speed up the fuzzing process by testing multiple payloads simultaneously if you decide to investigate more than one.

Final Analysis

CVE-2016-2184 is a critical vulnerability in the Linux Kernel's handling of IPsec, the xfrm subsystem specifically. IPsec is a widely adopted protocol for securing internet traffic, so the possibility of this exploit being used remotely through malformed IPsec packets is concerning. This vulnerability leads to system downtime, loss of service, and even full system crashes if left unpatched, which could create a possible vector for further exploitation.

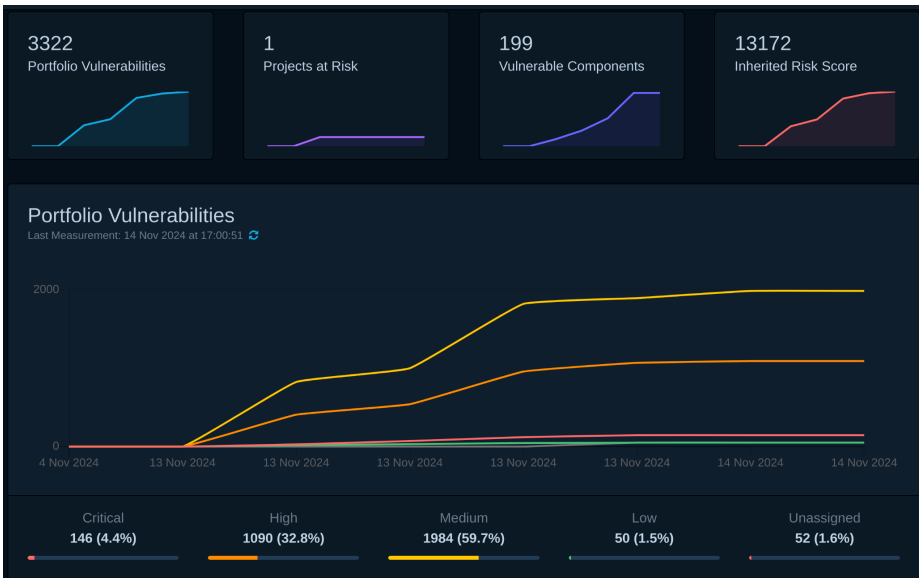
Testing this CVE in controlled environments such as VMs or legacy systems running outdated software is essential for understanding the full scope of the attack without compromising essential systems. Given that Python 2 is no longer supported but is still relevant in legacy configurations, this testing becomes critical for ensuring that older systems that still run vulnerable versions of the kernel we are working with, are protected. Simulating attacks using fuzzing tools like vUSBf allows researchers and security specialists to identify and mitigate CVEs before they become harmful to an environment.

Software Bill of Materials

Overview

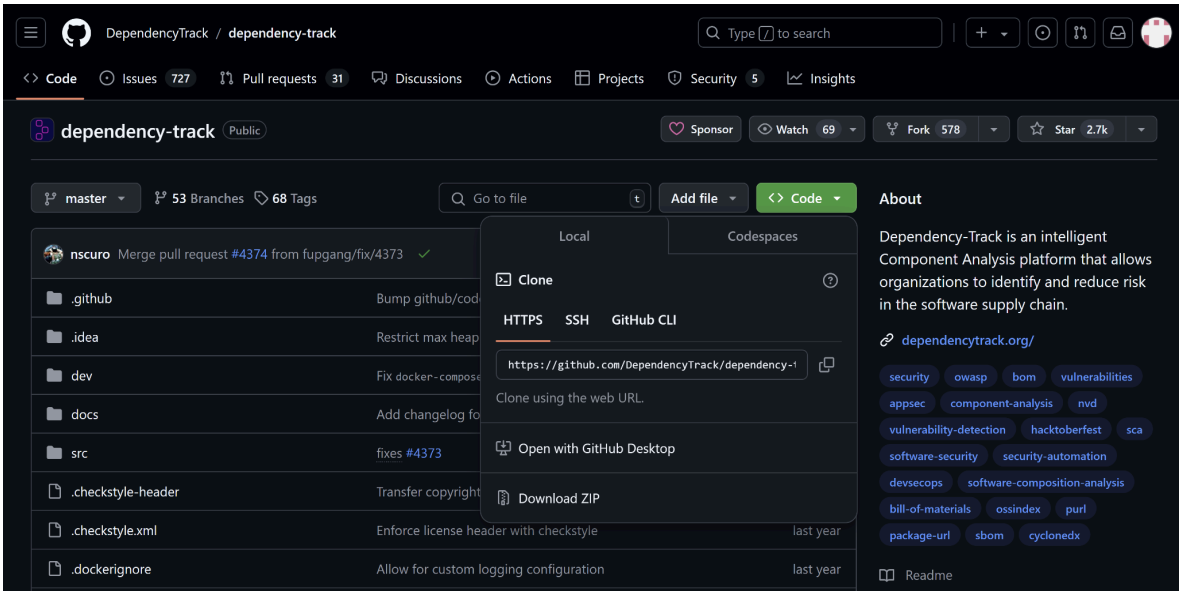
In today's rapidly changing digital world, we have constantly seen vulnerability after vulnerability emerge from what seems to be thin air. When studying and learning about Cybersecurity, there is always an emphasis on keeping up with new threats along with old ones. When implementing a Software Bill of Materials (SBOM), you are giving yourself an opportunity to learn more about the ins and outs of your software components, dependencies, and metadata. Keeping track of all of the previously listed elements, SBOMs enable teams to manage vulnerabilities, stay within compliance of any standards that may be required, and maintain transparency with stakeholders and other teams throughout an organization. Without an SBOM, it is almost impossible to keep track of all known vulnerabilities that may be prevalent in your environment.

In May of 2021, the United States Government released an Executive Order requesting the improvement of Cybersecurity in the U.S. The order reiterated the importance of creating a Software Bill of Materials and stated that "understanding the supply chain of software…and using it to analyze known vulnerabilities are crucial in managing risk". In the order, President Biden compared SBOMs to the ingredient label on food packaging. When thinking about the analogy, it seems to make perfect sense. Once you let your tool of choice analyze your environment, it tells you the amount of vulnerabilities you have in your portfolio, vulnerable components, inherited risk score, and a list of CVE's that are able to be exploited, along with other important information. Everything you need to know is handed to you in an easy-to-understand format. With this in mind, we thought implementing an SBOM in our project would only be beneficial. It has provided us with an opportunity to look into more CVEs in the future and go further with our exploitations beyond the completion of the semester.



Creating the SBOM

1) Navigate to the Dependency Track GitHub and download dependency-track-master
    a) https://github.com/DependencyTrack/dependency-track



2) Once downloaded, go into the terminal and go to the directory in which the dependency-track-master is downloaded it and run the command *sudo docker-compose up -d*.
    a) This command will build and start both the API server and frontend
3) To verify everything is up and running, run the *docker ps -a* command.
4) Go to the Downloads/syft directory and run *sudo apt update*
5) Once completed, enter *sudo apt install -y golang*

```
capping@capping-00:~/Downloads/syft$ sudo apt update
[sudo] password for capping:
Hit:1 https://dl.google.com/linux/chrome/deb stable InRelease
Hit:2 http://archive.ubuntu.com/ubuntu noble InRelease
Hit:3 http://security.ubuntu.com/ubuntu noble-security InRelease
Hit:4 http://archive.ubuntu.com/ubuntu noble-updates InRelease
Hit:5 http://archive.ubuntu.com/ubuntu noble-backports InRelease
Hit:6 https://ppa.launchpadcontent.net/deadsnakes/ppa/ubuntu noble InRelease
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
74 packages can be upgraded. Run 'apt list --upgradable' to see them.
capping@capping-00:~/Downloads/syft$ sudo apt install -y golang
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following additional packages will be installed:
  golang-1.22 golang-1.22-doc golang-1.22-go golang-1.22-src golang-doc
  golang-go golang-src libpkgconf3 pkg-config pkgconf pkgconf-bin
Suggested packages:
  bzr | brz mercurial subversion
The following NEW packages will be installed:
  golang golang-1.22 golang-1.22-doc golang-1.22-go golang-1.22-src golang-doc
  golang-go golang-src libpkgconf3 pkg-config pkgconf pkgconf-bin
0 upgraded, 12 newly installed, 0 to remove and 74 not upgraded.
Need to get 45.9 MB of archives.
After this operation, 229 MB of additional disk space will be used.
Get:1 http://archive.ubuntu.com/ubuntu noble-updates/main amd64 golang-1.22-doc all 1.22.2-2ubuntu0.3 [107 kB]
Get:2 http://archive.ubuntu.com/ubuntu noble-updates/main amd64 golang-1.22-src all 1.22.2-2ubuntu0.3 [19.7 MB]
Get:3 http://archive.ubuntu.com/ubuntu noble-updates/main amd64 golang-1.22-go amd64 1.22.2-2ubuntu0.3 [25.9 MB]
Get:4 http://archive.ubuntu.com/ubuntu noble-updates/main amd64 golang-1.22 all 1.22.2-2ubuntu0.3 [5,702 B]
Get:5 http://archive.ubuntu.com/ubuntu noble/main amd64 golang-src all 2:1.22~2build1 [5,078 B]
Get:6 http://archive.ubuntu.com/ubuntu noble/main amd64 golang-go amd64 2:1.22~2build1 [43.9 kB]
Get:7 http://archive.ubuntu.com/ubuntu noble/main amd64 golang-doc all 2:1.22~2build1 [2,788 B]
Get:8 http://archive.ubuntu.com/ubuntu noble/main amd64 golang amd64 2:1.22~2build1 [2,736 B]
Get:9 http://archive.ubuntu.com/ubuntu noble/main amd64 libpkgconf3 amd64 1.8.1-2build1 [30.7 kB]
Get:10 http://archive.ubuntu.com/ubuntu noble/main amd64 pkgconf-bin amd64 1.8.1-2build1 [20.7 kB]
Get:11 http://archive.ubuntu.com/ubuntu noble/main amd64 pkgconf amd64 1.8.1-2build1 [16.8 kB]
Get:12 http://archive.ubuntu.com/ubuntu noble/main amd64 pkg-config amd64 1.8.1-2build1 [7,264 B]
Fetched 45.9 MB in 4s (11.2 MB/s)
Selecting previously unselected package golang-1.22-doc.
(Reading database ... 245942 files and directories currently installed.)
Preparing to unpack .../00-golang-1.22-doc_1.22.2-2ubuntu0.3_all.deb ...
```

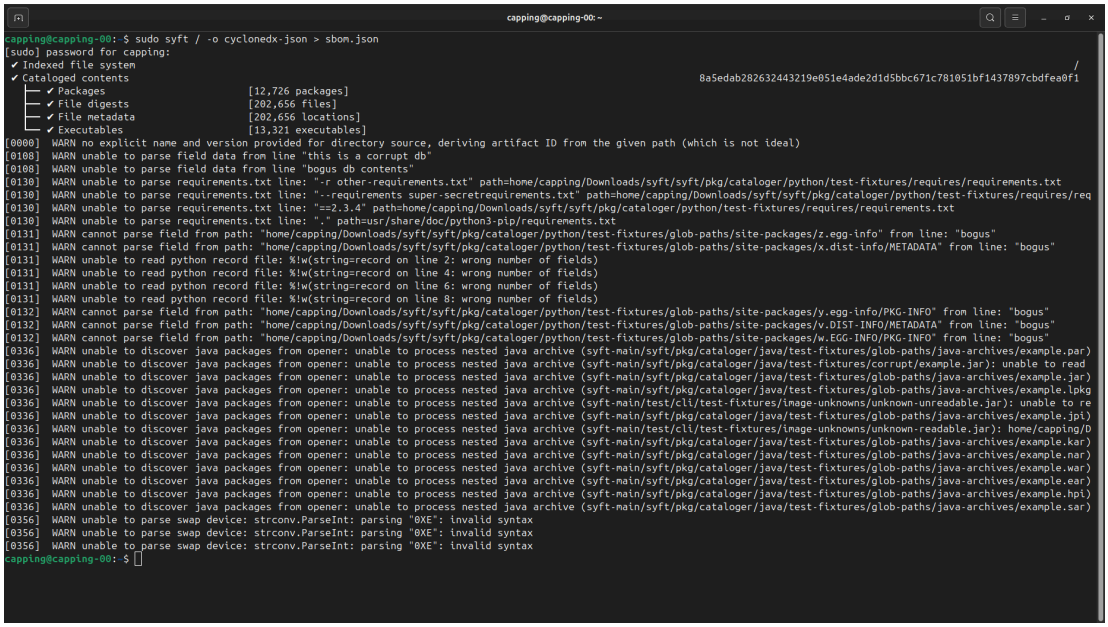
6) Run *go version* to ensure that it has been installed correctly

```
capping@capping-00:~/Downloads/syft$ go version
go: downloading go1.22.9 (linux/amd64)
go version go1.22.9 linux/amd64
capping@capping-00:~/Downloads/syft$ make build
[0000]  INFO binny version: 0.8.0
[0000]  INFO installing tool=benchstat version=latest
[0009]  INFO tools installed
task: [build] .tool/goreleaser build   --config .tmp/goreleaser.yaml --clean --snapshot --single-target
  • skipping validate...
  • cleaning distribution directory
  • loading environment variables
  • getting and validating git state
    • git state                                    commit=ac8be4ad4100df7385fada54e061771869070261 branch=main current_
    • pipe skipped                                 reason=disabled during snapshot mode
  • parsing tag
  • setting defaults
  • partial
  • snapshotting
    • building snapshot...                         version=1.16.0-SNAPSHOT-ac8be4ad
  • ensuring distribution directory
  • setting up metadata
  • writing release metadata
  • loading go mod information
  • build prerequisites
  • building binaries
    • partial build                                match=target=linux_amd64_v1
    • partial build                                match=target=linux_amd64_v1
    • partial build                                match=target=linux_amd64_v1
    • building                                     binary=snapshot/linux-build_linux_amd64_v1/syft
    • took: 1m16s
  • writing artifacts metadata
  • build succeeded after 1m15s
  • thanks for using goreleaser!
capping@capping-00:~/Downloads/syft$
```

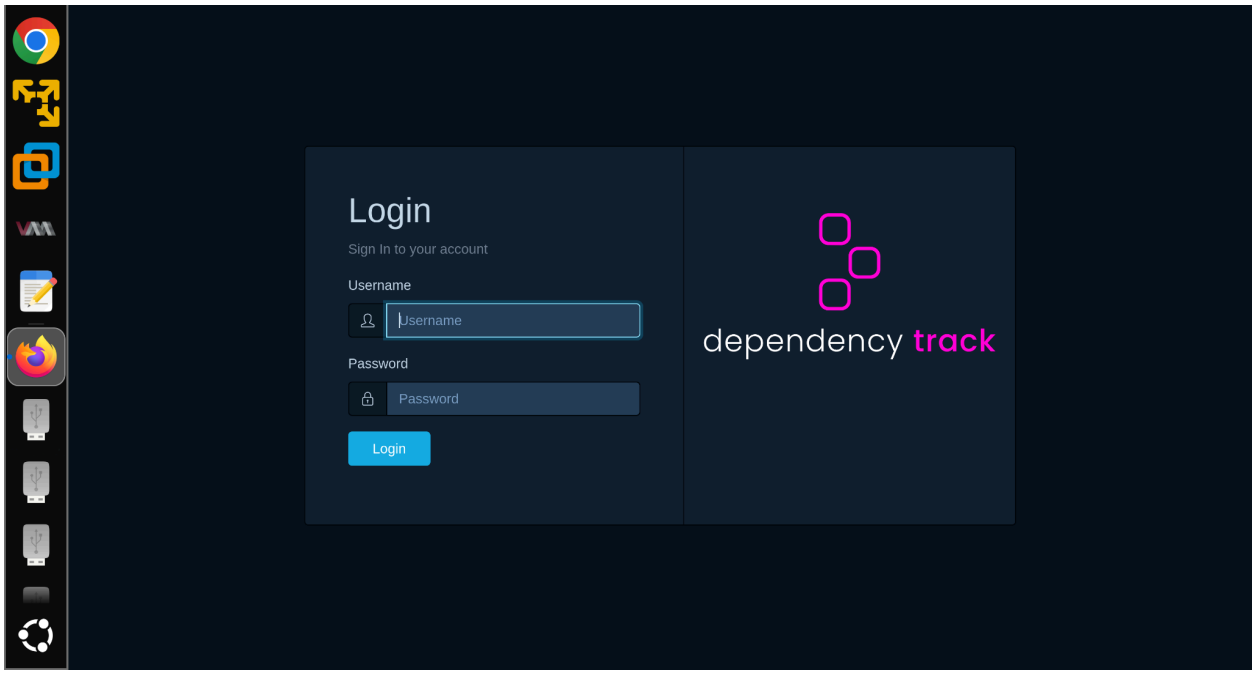
7) Run the command *make build*

8) To make the SBOM, you will then have to run the command *sudo syft / -o cyclonedx- json > sbom.json*

   a) This will create the SBOM along with a file that holds the SBOM which will be located in your downloads folder



9) From here you will then have to go to localhost:8080 on the browser of your choice and log in using the given credentials, admin:admin. Once prompted, you will enter the username and then change the password to whatever you would like.



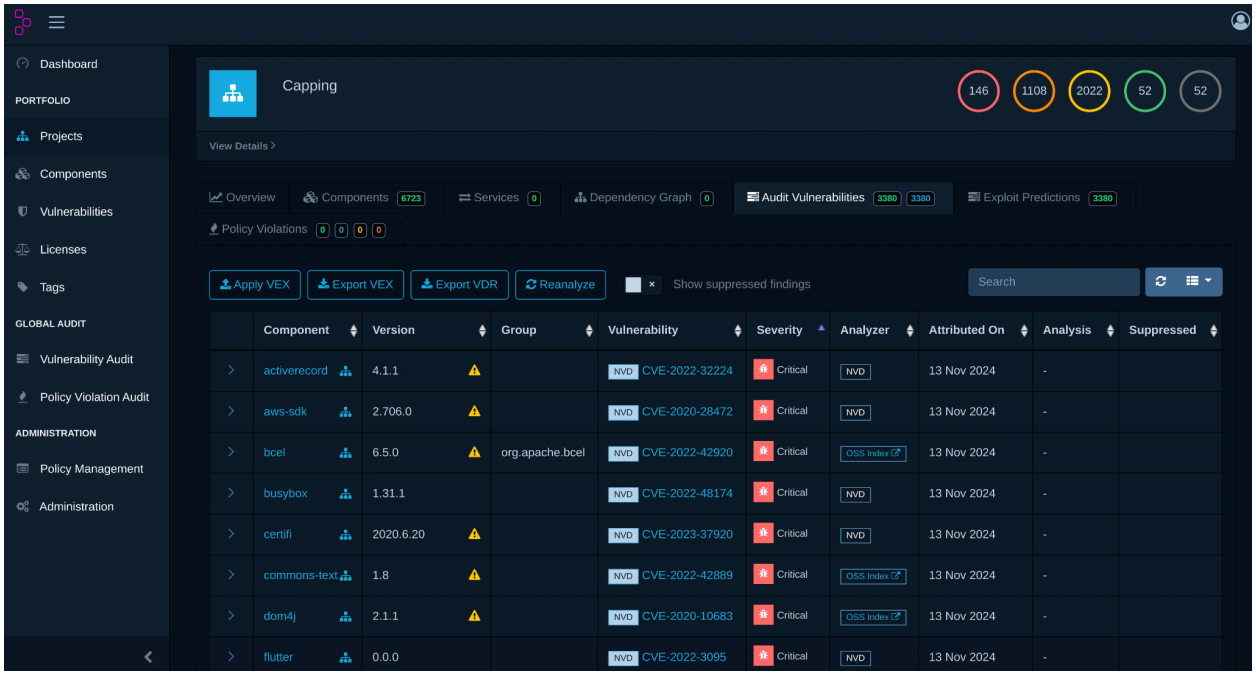
10) Once you gain access to dependency-track, you will add yourself as a user by going to the menu on the left side, selecting Administration, then going to Access Management, and then Managed Users. From there, you will be given the option to add yourself.
11) When that is completed, go to the projects menu and create a project. Input all of the information that you think may be useful. From there, you will go to the components tab and select and upload the BOM.
    a) Once uploaded, it will take a little while to process and analyze, but slowly results will start to show on the dashboard and other menus.

SBOM Benefits

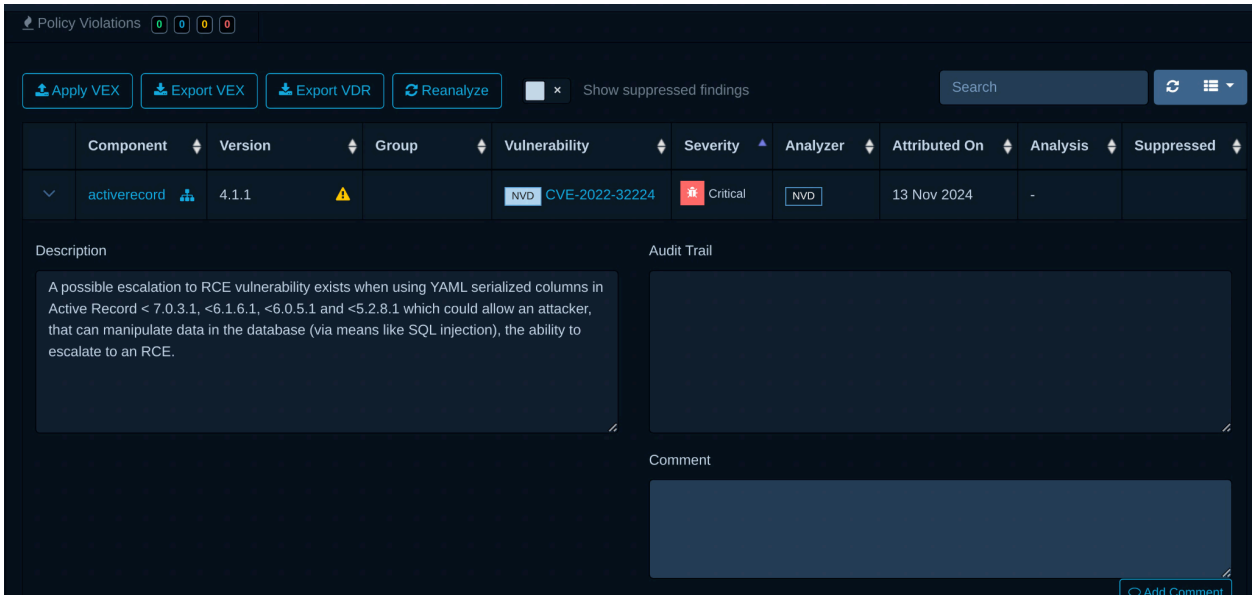
As discussed previously, the Software Bill of Materials provides a great amount of insight on the daily operations and security stance of software systems. SBOMs are a crucial tool that aids in identifying vulnerabilities, searching for any outdated components, and quickly dealing with any threats that may arise from any of the installed elements. Although SBOMs are fairly new to most industries, there are several different ways one can take advantage of what it has to offer. Below is a list of some of the many benefits that SBOMs provide and how you can use this information to perform further research and analysis.

Identifying and Avoiding Vulnerabilities

With the integration of a Software Bill of Material, one of the first items you are going to notice will be the amount of vulnerabilities present in your environment. When you first open the SBOM, you will be greeted with an organized dashboard that contains several different pieces of information. Immediately, your eyes are brought to a line chart that contains information on all of your vulnerabilities, separating them into categories including Critical, High, Medium, Low, and Unassigned. If you do more research and dig further beyond the numbers, you will be able to find a list of all vulnerabilities that contains information on what component the vulnerability is present in, the CVE, severity, and where the vulnerability information is coming from.



Once you have analyzed all of the information in the previous image, you can go further and click the down arrow at the start of each row, which will bring you to a new page that will explain the vulnerability even more. From there, you will be given the opportunity to input comments on the vulnerability to stay organized and project your thoughts on how to deal with it.



The amount of information that is provided by the SBOM regarding vulnerabilities is endless. Around each corner, you will always find new information that is crucial to keeping your environment safe. Using the provided information will guide you to better identify and avoid all vulnerabilities. Although it does not provide a step-by-step process to mitigate any risks, it gives you most of the necessary information for you to go and do it yourself.

Open Communication and Collaboration

Something that always has been and always will be important is communication. You see on every job posting that having good communication is nonnegotiable and expected when joining a company. When a team is good at communicating, they instantly make tasks easier to manage. When considering vulnerabilities and CVEs, communicating is necessary to lowering and even mitigating any risk that may be present. Having an SBOM that is visible between multiple teams is a perfect way to ensure that everyone is aware of all risks and potential vulnerabilities. Implementing an SBOM brings a large amount of transparency to a company and warns people what is happening behind the scenes. It decreases the amount of time and effort that goes into making sure each and every team is aware of what is occurring in any software components, dependencies, and metadata. In addition, it allows teams that are not wellversed in a certain area to get a rundown of another area using the SBOM and additional resources.

Licensing Compliance

Another feature of the Software Bill of Materials that is extremely beneficial would be the ability to check licensing compliance. At any company, large or small, licenses can be tough to keep track of. People are constantly asking for new licenses or to renew existing ones. SBOMs will inform authorized users who have access to the tool of any licenses associated with each piece of their environment. From there, users are able to monitor the licenses and look further into them to see if they are required or if there are any legal obligations that they need to focus on related to the licenses. As written earlier, the SBOM offers a large amount of transparency that also involves the need of licensing compliance

Industry Usage

Intro

The virtualization technology utilized in the project plays a critical role in modern industry usage, enabling organizations to optimize hardware utilization and enhance flexibility when it comes to operating virtual environments. The kernel-based virtual machine the project leverages allows Linux to act as a hypervisor, which is a widely adopted practice in enterprise environments, because of the performance and scalability benefits it poses. CVEs are critically studied in the industry, as the security they threaten can have severe implications for virtualized infrastructures. The standard x86 server architecture the project is built on also supports scalability and compatibility, making it a leading foundation for real-world deployment of secure and efficient virtualized environments. Large-scale infrastructure providers address data management through the use of hyperscale cloud systems. This integration of scalability security and data flexibility drives the relevance of virtualization technology to meet the demands of today's industry.

CVEs

The objective of CVEs is to build awareness and share information about security loopholes and the effects they might have on a system. For organizations actively managing security threats, the studying of CVEs is paramount for these companies to find security breaches and trends, and to update their systems to fight security flaws. Security teams use CVEs to assess the severity and potential impact of vulnerabilities in their systems. High severity CVEs are obviously prioritized over low severity, as the Common Vulnerability Scoring System (CVSS) helps differentiate to save time and resources. Organizations use CVEs to track which vulnerabilities are ignorable and which ones require patches. Analyzing the context and urgency of a CVE requires consideration of real-world factors, such as:

1. Whether the vulnerability is exploitable in a service's default configuration or only under very specific configurations.
2. Whether a reachable path to the vulnerable code exists.
3. Whether the software library vulnerability has a code precondition (meaning someone must use it in a vulnerable manner).
4. The network environment and the overall security mechanisms applied to the vulnerable software.

Organizations might not interact directly with the CVE database but will utilize tools that do to save time, money, and resources. Scanners like Nessus and Qualys, which specialize in cloud security, rely on CVE databases to detect unpatched and exploitable vulnerabilities in networks and systems. CVEs are integrated directly into threat intelligence feeds, assisting organizations in anticipating attacks before they happen and mitigating the chance of attacks

breaching through and targeting specific vulnerabilities. Red Hat is a subsidiary of IBM and provides open-source products to enterprises. They use CVE IDs to track security vulnerabilities and to maintain their security update database. Organizations now more than ever generate SBOMs to inventory components and map them to known CVEs for proactive risk management. Since SBOMs list software libraries, dependencies, and third-party components in use, these components can be checked across CVE databases to determine how many vulnerabilities there are and the specifics of each of them. Nowadays, many SBOM creation software automatically cross-check the results of SBOM generation with CVE databases and will provide the user with an in-depth summary of how at risk they are. This tool allows organizations to identify risks and take action proactively.

Another way CVEs are utilized in industry is through regulatory frameworks and industry standards. These standards often require organizations to address CVEs as part of compliance. Auditors use CVE tracking to verify that organizations are managing vulnerabilities in a safe, timely, and effective way.

Servers

Throughout the progression of this project, we have consistently worked hard in creating the best possible environment that would give us the greatest opportunities in developing our skills and knowledge. When considering our project, one of the most crucial components of it is the server. In a way, the server is the backbone and foundation of not only just our project, but also modern computing as a whole. We have worked hard learning the most we possibly can that will soon translate to the work we will be executing post-college as Cybersecurity professionals. All of our work is easily able to be connected to real-world scenarios and has been nothing but beneficial for us to learn.

At the very start of the project, we built our system and installed Ubuntu on our server along with configuring and starting up a couple of virtual machines using Virtual Machine Manager. In the real-world, virtual machines are used to simulate potential changes in an environment instead of going and testing on the host machine. They can also be used to study any security vulnerabilities that may be of concern. Being able to build our server up to suit our needs has been a great learning experience and will surely benefit us in years to come.

In addition to the creation of the server, we also spent a large amount of time installing tools, such as Dependency-Track and Syft, that could be used to help improve supply chain security and identify vulnerabilities. In our case, we were more focused on using those vulnerabilities for exploiting rather than vulnerability patching. The Software Bill of Materials played a key part in ensuring the safety of our server and environment. One of the main reasons we decided to implement it was because we knew it would be a a good tool to learn about. In

class, we had multiple guest speakers talk to us about the importance of SBOM's and how it is a new focus point for a lot of companies. They are now being seen as a tool that is essential to implement due to its ability to aid in security standards compliance.

Overall, the server knowledge we have accumulated throughout the completion of this project has given us the opportunity to gain first-hand experience looking into the security, scalability, and integrity of technical environments. Having this knowledge will lead us to places that will provide us more opportunities to learn and grow even more beyond this project. The technical skills we have gained in the past few months, along with soft skills such as adaptability and problem-solving, has taught us everything we need to know that will enable us to work efficiently in the a workplace environment post-graduation.

Hypervisors

A hypervisor is the software that allows the use of virtualisation, it acts as the intermediary between the hardware and the virtual machines. The way a hypervisor works is by allocating a specific amount of the resources from the host machine to use for virtual machines. These resources include parts like the CPU and memory to be used for the virtual machines.

There are several benefits that come with the use of hypervisors in the real world, such as hardware independence, overall efficiency, scalability, and portability. The hardware independence that comes with the use of a hypervisor and virtual machines can be very useful, for example you can run operating systems with different architecture from your host machine because of the independence. With a hypervisor, you can configure it to immediately create your virtual environment which will greatly save time which is often manually spent installing software or operating systems. Hypervisors can allocate a computer's resources specifically so you can run multiple virtual machines on one host machine. Then you can run multiple different workloads on one physical machine instead of using multiple physical machines to do so.

One use case that can be seen in the real world with hypervisors would be desktop virtualization. An employee can use desktop virtualization running on a server to access their workstation environment and files remotely. Another use case where a hypervisor could apply would be failure recovery. With a hypervisor you can create a snapshot of a virtual machine's state and return to that exact state whenever you want. If there is a system failure, but you have a snapshot of a previous version of the virtual machine, you can easily boot that up and restore the system to that version. Hypervisors can provide many different benefits to a company if they choose to implement them.

Hyperscale Cloud Computing

Hyperscale is a distributed computing environment and architecture that is designed to provide extreme scalability to accommodate extremely large workloads. Hyperscale data centers are significantly larger than on-premises data centers and can handle more data. Now more than ever, organization projects require more than they used to to be completed. Many organizations require hyperscale computing, which provides more benefits compared to enterprise data centers. With hyperscale computing, companies can gain access to, and even build almost infinitely scalable databases. Hypervisors inside data centers use abstraction layers to let apps in virtual machines be relocated from one physical location to another. All of this allows companies to maintain thousands of servers, with some of the largest companies in the world dedicating tens of thousands of square feet of space to house tens of thousands of servers. Within the last 20 years, hyperscale computing has expanded to adopt cloud solutions, which opens the door to more services and enterprise applications. These hyperscale cloud systems manage scale computing, storage, and network resources even more efficiently than traditional computing environments.

Hyperscale cloud systems are typically managed by large providers like Amazon Web Services (AWS), Microsoft Azure, and Google Cloud Platform (GCP). These companies not only offer computing and storage, which is widely used by many, but also offer machine learning solutions, virtual machine access, AI integration, and more. These systems are widely used in all aspects of industry today, reaching corners like E-commerce, finance, and even gaming media.

Our KVM project involving the use of a Linux virtualization security setup aligns with modern-day usage of hyperscale cloud systems in a variety of ways. Hyperscalers rely on mass virtualization to manage resource allocation across massive data centers. Our project mirrors this essential function by implementing and hardening virtualization technology. Our hypervisor divides the resources from our host machine to our VMs, ensuring each one operates in a logically isolated environment with carefully managed and secure resources. This approach reflects the same process of developing security profiles that drive hyperscale cloud systems in today’s industry.

Future Plans

Introduction

Throughout the semester, we have been fully focused on creating the greatest possible project that would enable us to learn and grow as we go. We have spent countless hours in the Enterprise Computing Research Lab working on setting up our environment, researching CVEs and other interesting topics, and communicating with Fernando Pizzano on our progress. Each member of our team has been fully focused on completing the deliverables given to us by Professor DeCusatis and Fernando which has led us to the completion of the project. As we look back on our progress, we are all exceedingly proud of the work we have completed and the research executed. With that being said, we are not ready to conclude our project and have created a list of items we would like to complete beyond the completion of this semester. The "Future Plans" section will detail our next goals for the project and what we would like to look into for the upcoming months.

CVE Research

While working on our project, a lot of time and effort went into researching CVEs and finding one we could exploit in our environment. We gathered a list of vulnerabilities and wrote down a potential plan that would allow us to exploit them. After three attempted exploitations, we decided it was time to move along with our project and focus on other deliverables. With the completion of the semester, we will receive a little more freedom and be able to focus on anything we wish. Finding more CVEs is something we are very interested in and would love to be able to fully exploit a few. Identifying CVEs to exploit remains an important aspect of our project, as it plays a large role in further research and analysis.

SBOM Research

As the end of the semester neared, we decided that implementing an SBOM would be a good addition to the work we had already completed. SBOMs are a fairly new component of cybersecurity and we loved the idea of experimenting with it and learning more about how they work. Because we did not know too much about SBOMs, we were not aware of how helpful it could have been for our project. Being able to get a list of CVEs that we can potentially exploit is very beneficial and makes our project a little bit easier. With this being said, we would love to utilize the SBOM more in the future now that we know what it is fully capable of. Being able to formulate new ideas and plans as we go with the help of the SBOM will help aid the development of our project immensely. In the coming months, we plan on fully utilizing what the Software Bill of Material has to offer to make a stronger project. We believe this will be of great benefit and has no real downsides.

AI Log Analysis

The last addition we would like to include when it comes to technical work would be implementing AI to analyze the logs outputted by our server. As we worked towards the exploitation of CVE-2016-2184 and CVE-2024-8354, we thought it would be of great assistance to use AI to break down the logs for us and give us a quick summary of each of them. To do this, we aim to introduce machine learning. Machine learning will enable us to find any complex information that would be tough to identify manually. In addition, it could help us group and organize the logs so we are easily able to sort through them as opposed to reading through each log one by one. With the help of machine learning, we will be able to go further in exploiting CVEs, as we will understand more of the errors we are receiving when exploiting.

Competitions

We are advancing our capstone project into spring 2025, with plans to enter into regional competitions. On November 8th, 2024, we showcased our work at the inaugural Mid-Hudson Valley TechMeet at Bard College. Looking ahead, we aim to present our project at events such as the 2025 Mid Hudson Region Tech Fest in Kingston, NY and the 2025 Mid-Hudson Regional Business Plan Competition at Marist College. These venues offer great opportunities to network with industry professionals and garner valuable feedback on our research, development, and implementation of this project.

Publication

We aim to publish our research in a reputable technical journal upon completing our work. As a group, we would make final revisions to ensure our work adheres to the strict guidelines of academic publications specializing in topics such as cybersecurity, virtualization, and related fields. One primary target for submission is the Institute of Electrical and Electronics Engineers (IEEE), an organization that aligns well with the scope and technical depth of our research. Once we have refined our work to meet the standards of the publication, we will formally submit it to IEEE and other publications for consideration with the goal of having our work officially published and shared with the broader cybersecurity community.

References

"Libvirt vs QEMU: What are the differences?," StackShare, https://stackshare.io/stackups/libvirt-vs-qemu#:~:text=In%20summary%2C%20QEMU%20is%20primarily,management%20interface%20across%20different%20hypervisors

C. Goodman, "What is a CVE? common vulnerabilities and exposures defined," Fortinet, https://www.fortinet.com/resources/cyberglossary/cve

"CVE-2024-8354," CVE, https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2024-8354

Chatgpt, https://chatgpt.com/

Schumilo, "Schumilo/vUSBf," GitHub, https://github.com/schumilo/vUSBf?tab=readme-ov-file

"CVE-2016-2184," Ubuntu, https://ubuntu.com/security/CVE-2016-2184#notes

"CVE-2016-2184 Detail," NVD, https://nvd.nist.gov/vuln/detail/CVE-2016-2184#match-11535809

"CVE-2016-2184," Ubuntu, https://ubuntu.com/security/CVE-2016-2184

"Executive order on improving the nation's cybersecurity," Executive Order on Improving the Nation's Cybersecurity, https://www.whitehouse.gov/briefing-room/presidential-actions/2021/05/12/executive-order-on-improving-the-nations-cybersecurity/

H. L, "SBOMS and the importance of inventory," NCSC, https://www.ncsc.gov.uk/blog-post/sboms-and-the-importance-of-inventory

S. Wickramasinghe and S. Watts, "The CVE & CVE Management, explained," Splunk, https://www.splunk.com/en_us/blog/learn/cve-common-vulnerabilities-exposures.html

S. Menashe, "Why do we need real-world context to prioritize cves?," Why Do We Need Real-World Context to Prioritize CVEs?, https://www.darkreading.com/cyber-risk/why-do-cve-scores-need-real-world-context-to-prioritize

"What is a CVE?," Red Hat - We make open source technologies for the enterprise, https://www.redhat.com/en/topics/security/what-is-cve

"What is a virtual machine? VM uses and benefits | google cloud," Google, https://cloud.google.com/learn/what-is-a-virtual-machine

What is a hypervisor? - hypervisor explained - AWS, https://aws.amazon.com/what-is/hypervisor/

P. Powell and I. Smalley, "What is hyperscale?," IBM, https://www.ibm.com/topics/hyperscale

E. Plesky, "What is a Hyperscaler cloud?," Plesk, https://www.plesk.com/blog/various/what-is-a-hyperscaler-cloud/

"What is a hyperscaler?," Red Hat - We make open source technologies for the enterprise, https://www.redhat.com/en/topics/cloud-computing/what-is-a-hyperscaler

"2025 mid hudson region tech fest - sponsor registration," Home, https://www.nyscate.org/NYSCATE/iCore/Events/Event_display.aspx?EventKey=MHRTF24

"Mid-Hudson Regional Business Plan Competition," Marist College, https://www.marist.edu/computer-science-math/mid-hudson-regional-business-plan-competition

IEEE - The World's largest technical professional organization dedicated to advancing technology for the benefit of humanity., https://www.ieee.org/

"The IEEE article submission process," IEEE Author Center Journals, https://journals.ieeeauthorcenter.ieee.org/submit-your-article-for-peer-review/the-ieee-article-submission-process/

"The IEEE article submission process," IEEE Author Center Journals, https://journals.ieeeauthorcenter.ieee.org/submit-your-article-for-peer-review/the-ieee-article-submission-process/

"Publish with IEEE Journals," IEEE Author Center Journals, https://journals.ieeeauthorcenter.ieee.org/

"How to publish your research," Author Services, https://authorservices.taylorandfrancis.com/publishing-your-research/

"Krebs on security," Krebs on Security, https://krebsonsecurity.com/