TEAM PROJECT

OPS300 Group 7

December 12, 2023

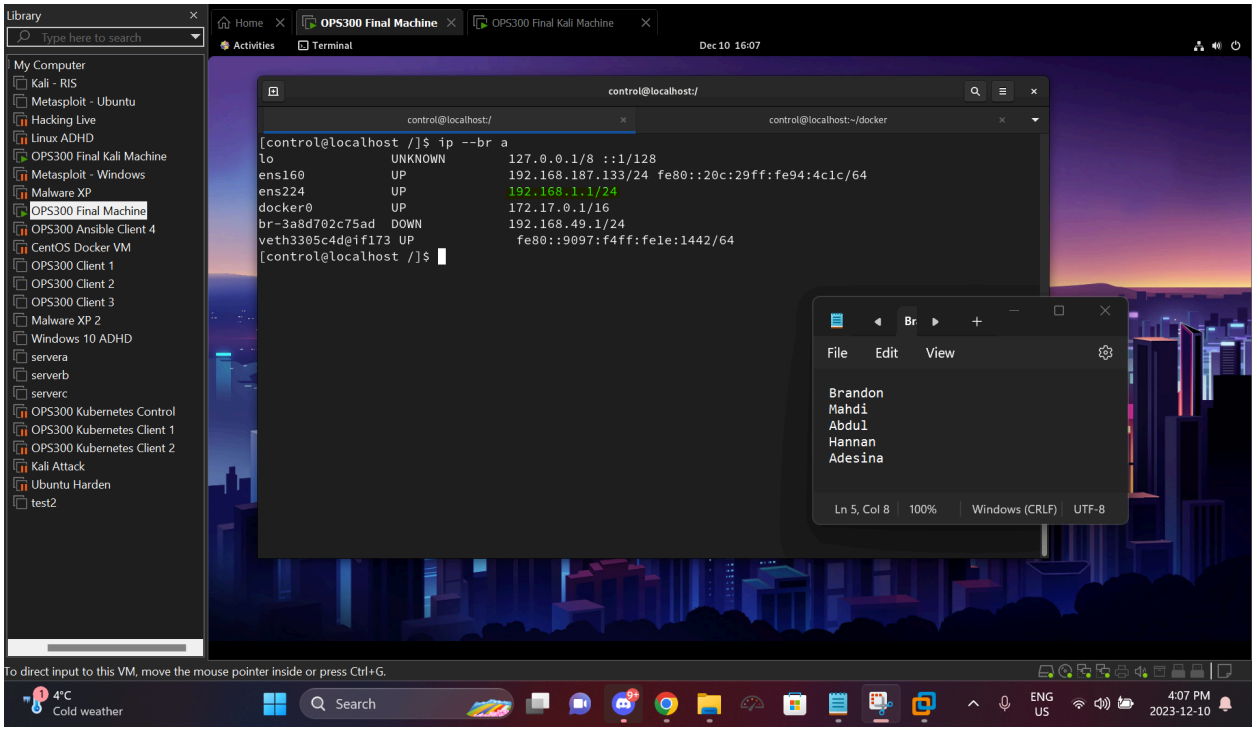
SENECA POLYTECHNIC
Abdulfatah Abdillahi, Mahdi Afshar, Brandon Da Costa,
Hannan Karedia, Adesina Laja

Table of Contents:

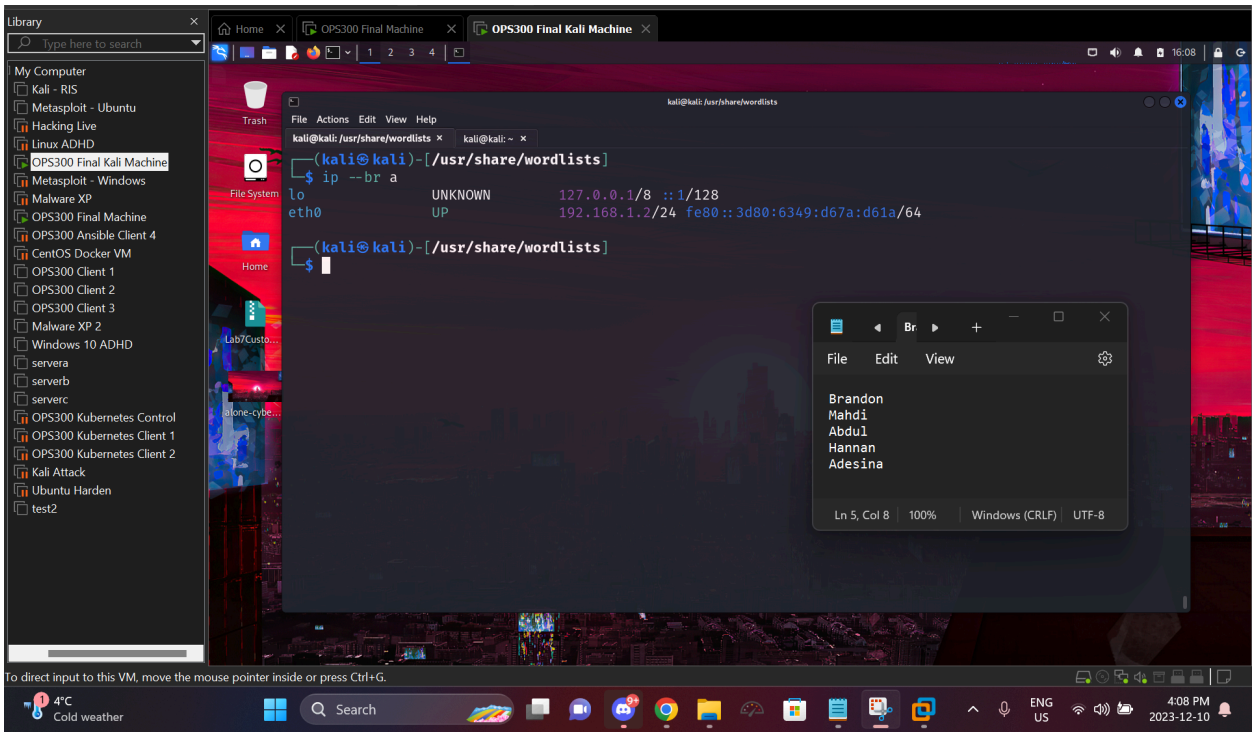
Privilege Escalation with Broken Sudo Permissions Ansible Playbooks:

Environment Setup:

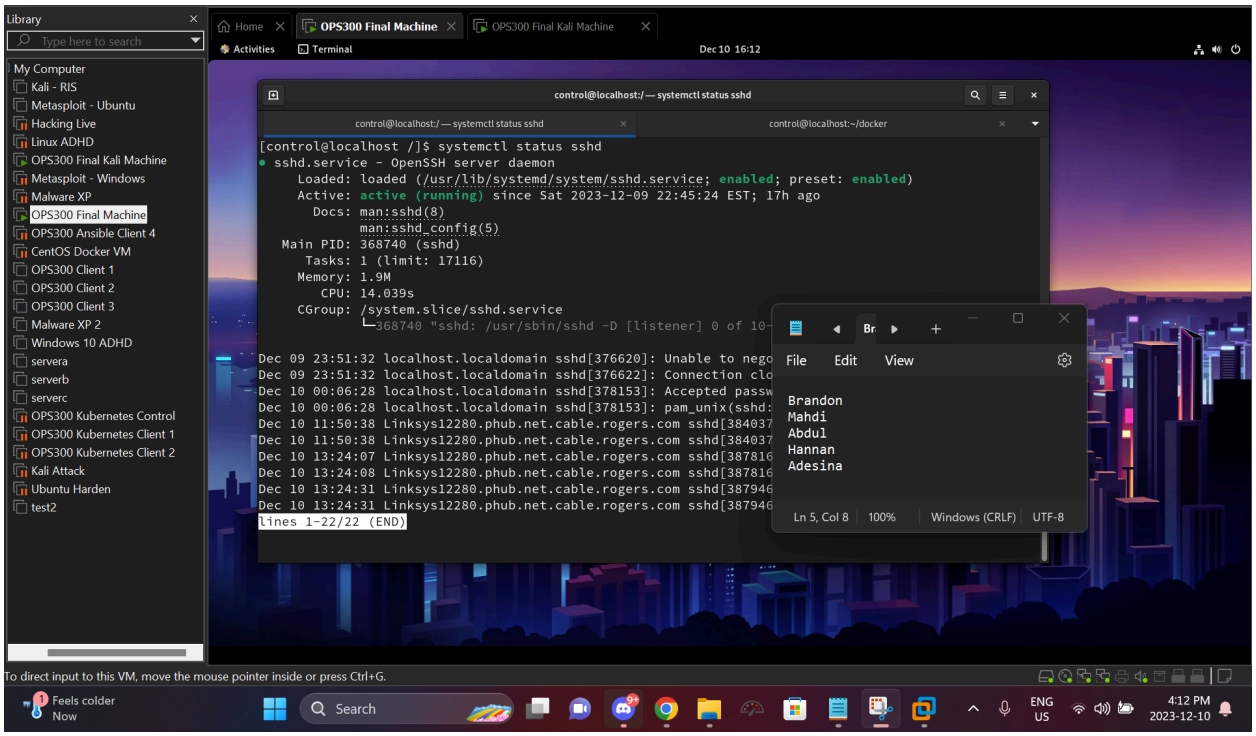
In preparation for this demonstration, we have configured a two-machine environment. The primary system is a CentOS Stream 9 Virtual Machine (VM), designated as the target machine, while the secondary machine is a Kali VM. These machines are seamlessly interconnected within LAN segment 1, ensuring a working network environment. The CentOS machine has ansible installed with a working ansible playbook and misconfigured sudo permissions which will be abused for privilege escalation in later steps.



Screenshot 1: This screenshot shows the target CentOS Stream 9 VM with an IP address 192.168.1.1/24.

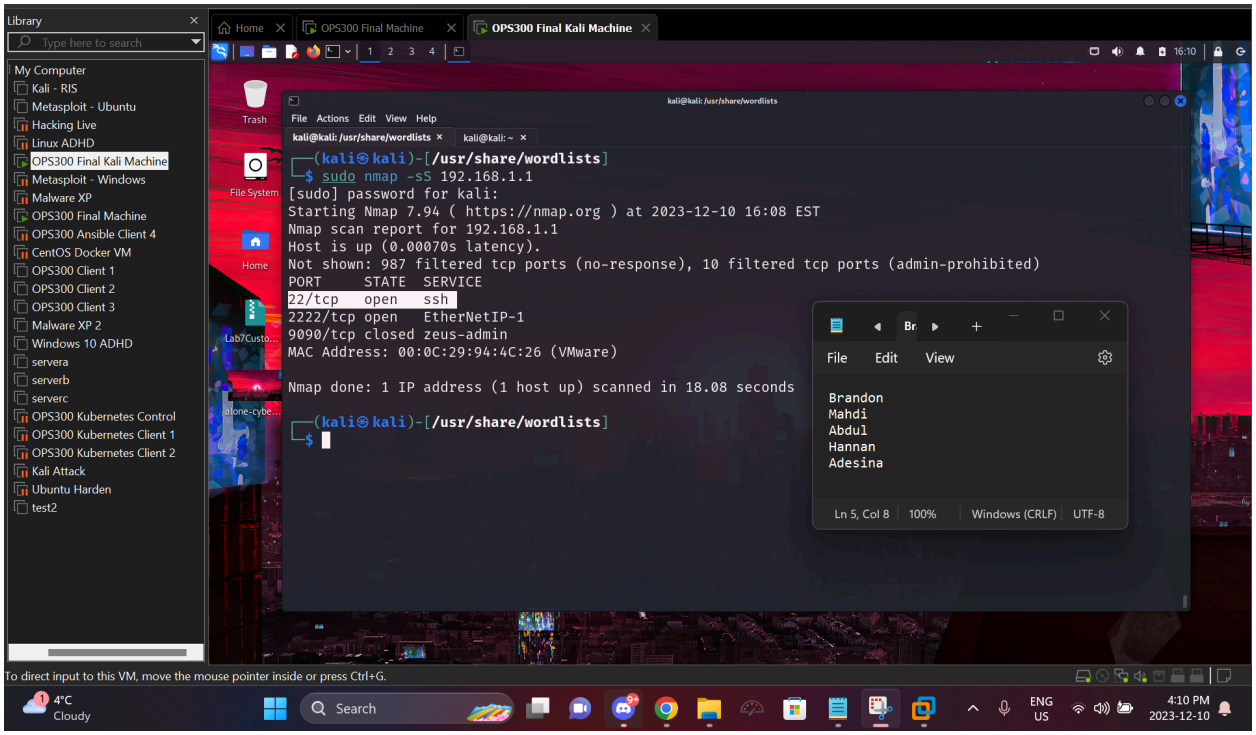


Screenshot 2: This screenshot shows the attackers Kali VM with an IP address 192.168.1.2/24.



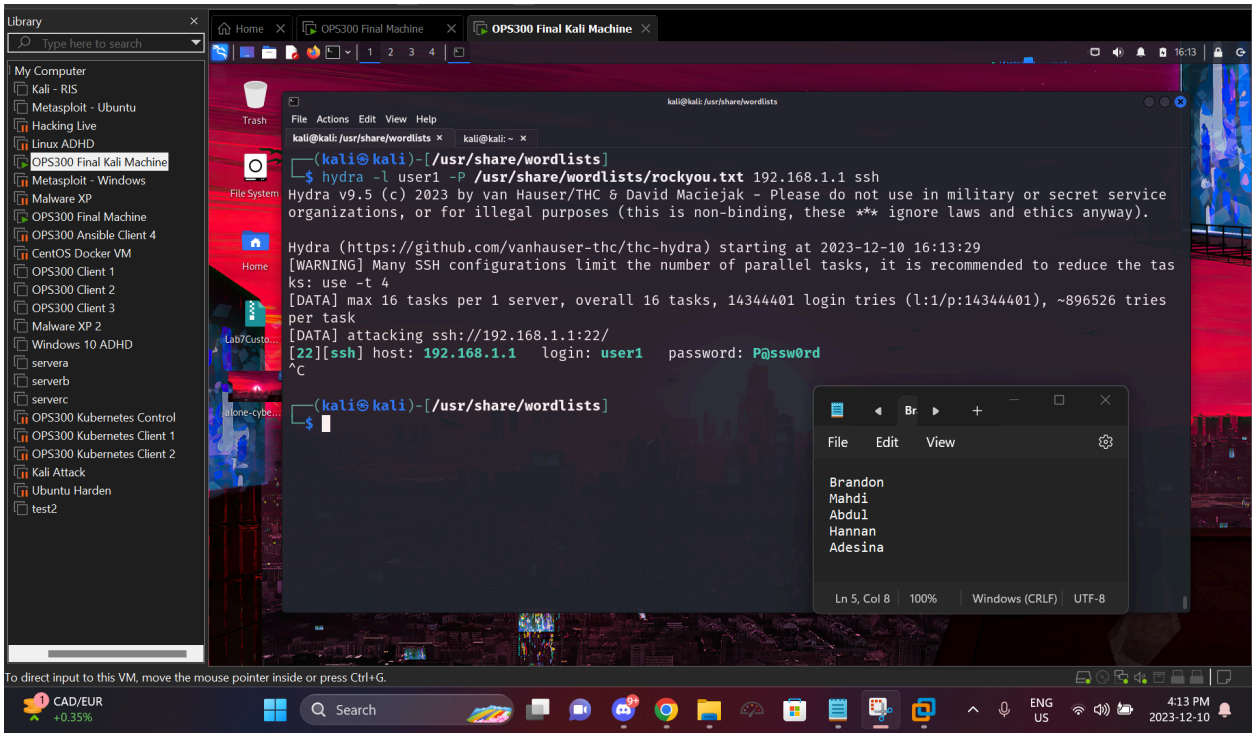
Screenshot 3: This screenshot shows the active status of ssh service on the target CentOS Stream 9 VM.

Enumeration of Target:

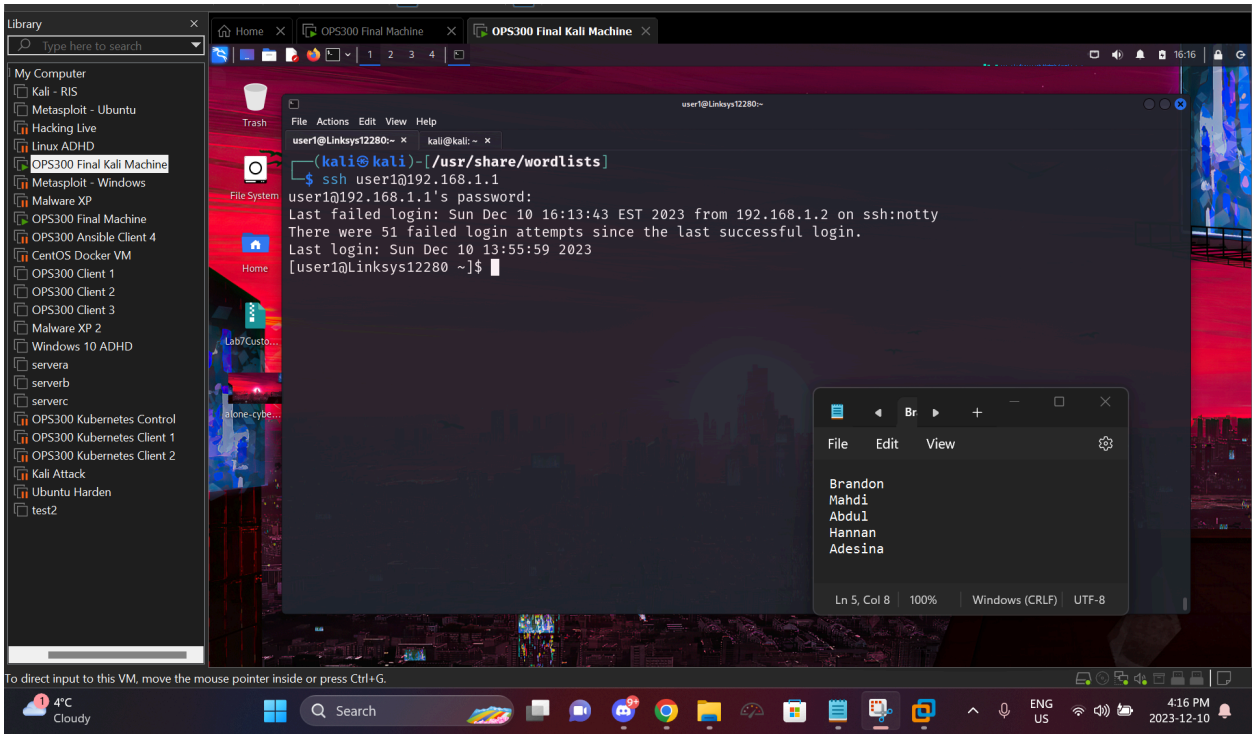


Screenshot 4: In the following screenshot, we demonstrate the utilization of nmap to conduct a scan on the target machine, represented by the CentOS 9 VM with the IP address "192.168.1.1". Employing the "-sS" option, which performs a SYN scan, we observe from the output that the "ssh" service is active. For this section our focus will be on targeting the SSH port (22) of the CentOS VM for further exploitation.

Initial Access:



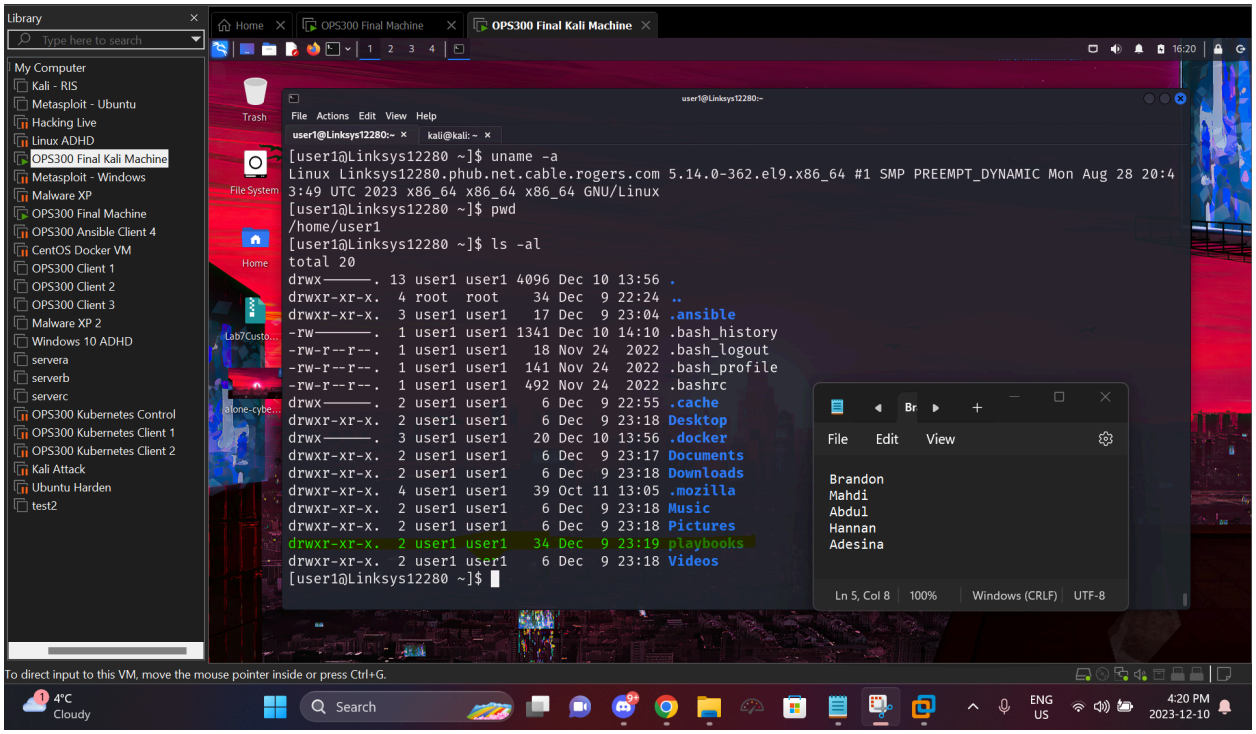
Screenshot 5: This screenshot displays us utilizing the Hydra tool to attempt a basic brute force attack for the target's credentials for SSH. The assumption here is that the account “user1” was obtained through means such as phishing/social engineering and that the target used a weak password. Please note that initial access to a target system can be obtained via exploiting other misconfigurations/vulnerabilities and this is not the only way.



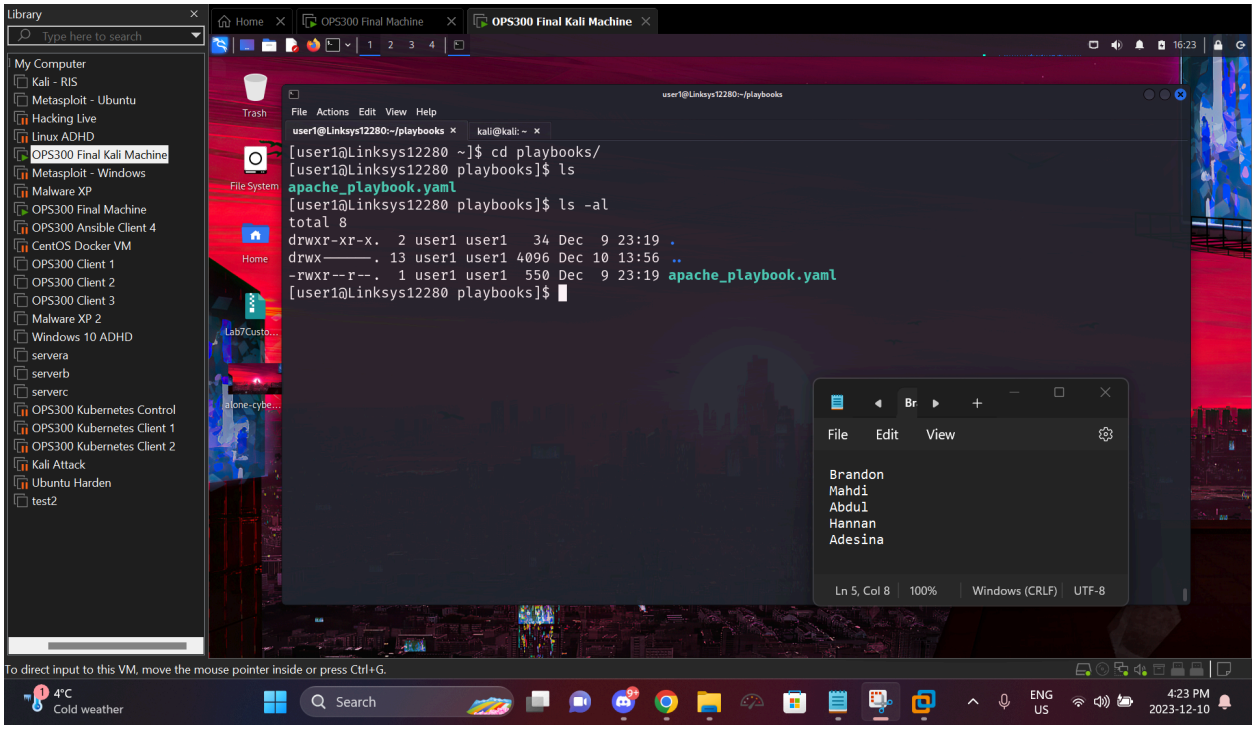
Screenshot 6: The screenshot below shows us using SSH to log into the "user1" account on the target machine "192.168.1.1". We have successfully logged into the target machine using SSH, as indicated in the output.

Exploiting the Ansible Playbook:

In cybersecurity, one common objective for attackers is to escalate their privileges and attain root access once the initial foothold is obtained. This strategic move grants them elevated control over the system, allowing for more extensive manipulation and attacks. Ansible playbooks, which are frequently used in system administration, might become a means of privilege escalation/attack when abused by threat actors. For example, if they have write access to a Playbook that can be run, they can modify it to perform harmful actions. Understanding these malicious techniques is crucial in strengthening system protections against unwanted privilege escalation/attack efforts.

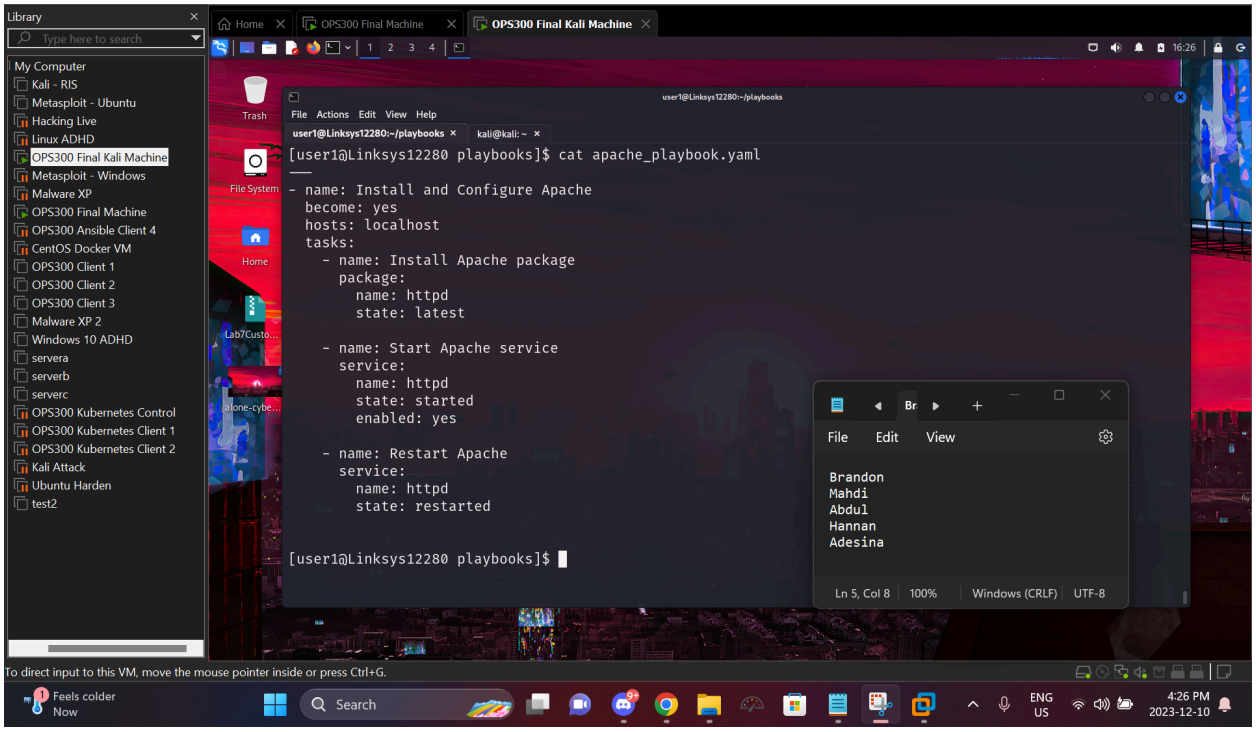


Screenshot 7: This screenshot mimics an attacker's behavior to enumerate the target machine upon initial access. In our case, running ***uname -a*** displayed some information about the OS, running ***pwd*** displayed our current directory/location, and running ***ls -al*** revealed a list of all the files in the directory. This list contained an interesting folder called playbooks.

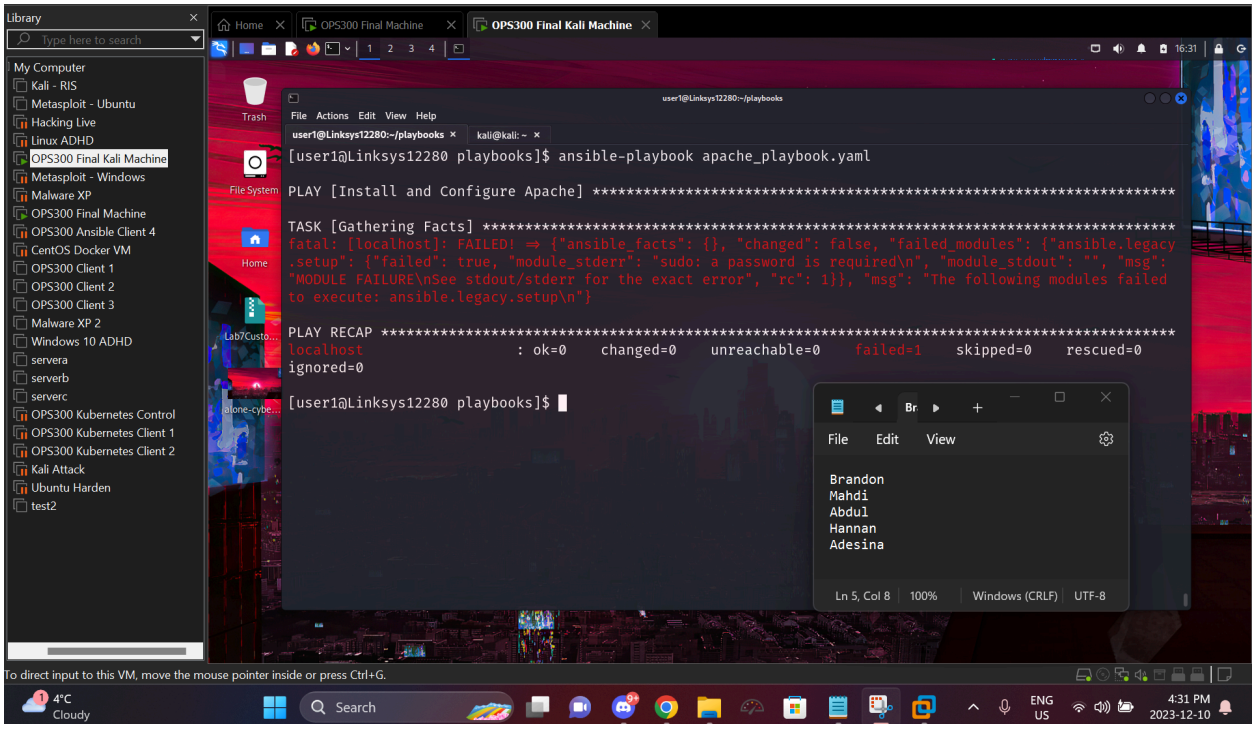


Screenshot 8: This screenshot shows the simulated attacker changing directories to the "playbooks" folder. This is followed by running the command ***ls -al*** to reveal a playbook and

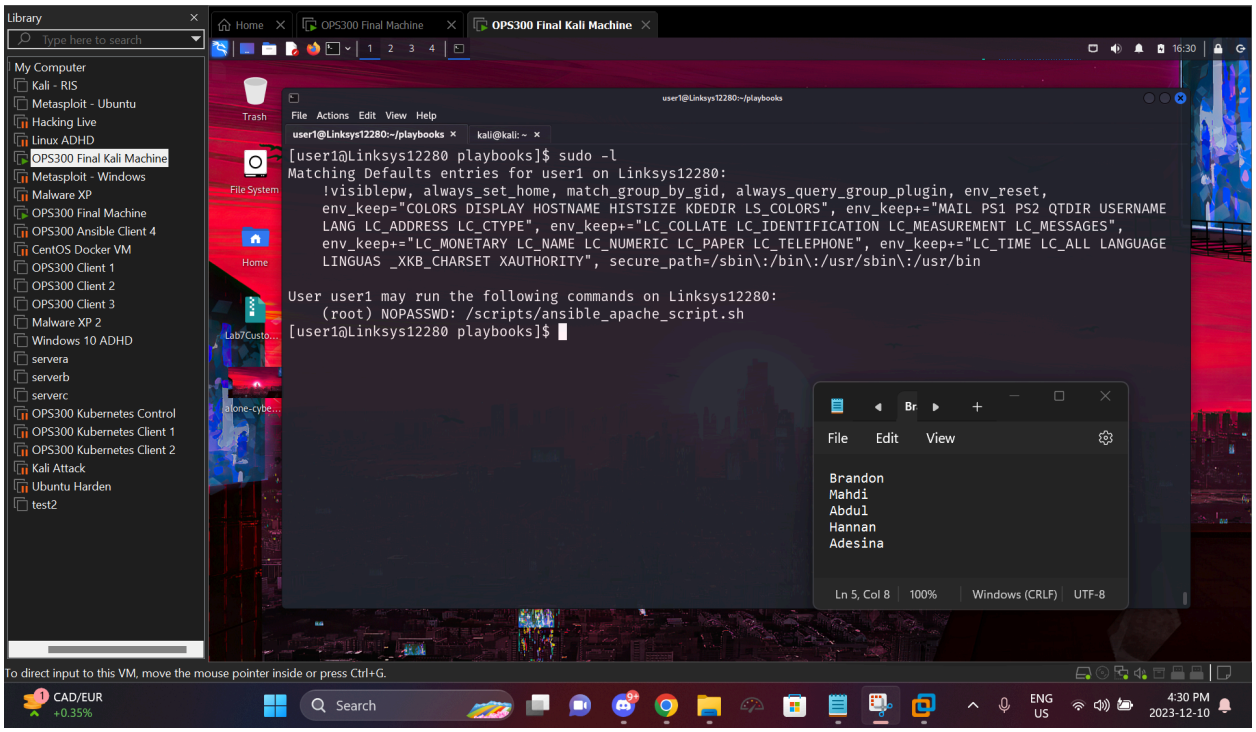
that user1 has full permissions (read, write, and execute) for the playbook (apache_playbook.yaml) file.



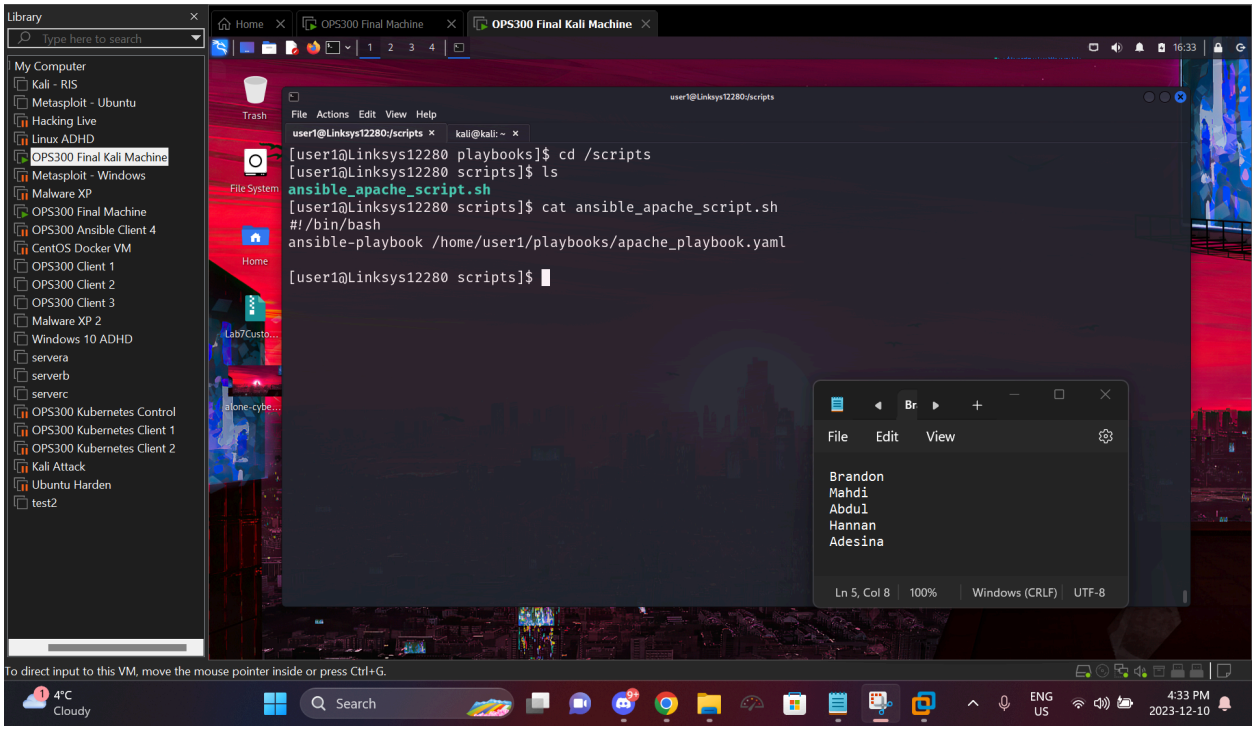
Screenshot 9: This screenshot shows the contents of the found ansible playbook file. The current task installs an Apache web server package, starts the service, and then restarts the Apache service.



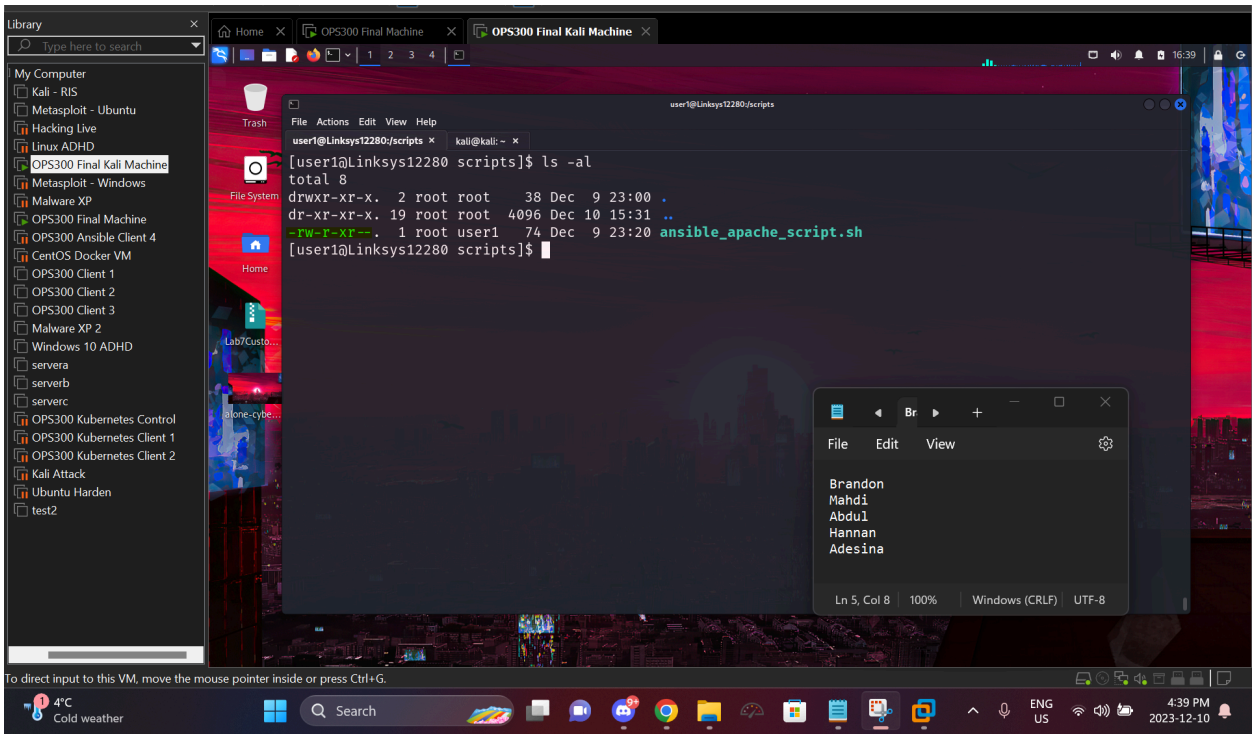
Screenshot 10: This screenshot shows a failed attempt to run the ansible playbook as user1. This is due to user1 not having the appropriate permissions to execute the ansible-playbook command on the centOS system.



Screenshot 11: This screenshot shows further enumeration conducted by the attacker by running ***sudo -l*** to see the permissions user1 has on the CentOS system. The output suggest that user1 has permission to run the /scripts/ansible_apache_script.sh script as root.

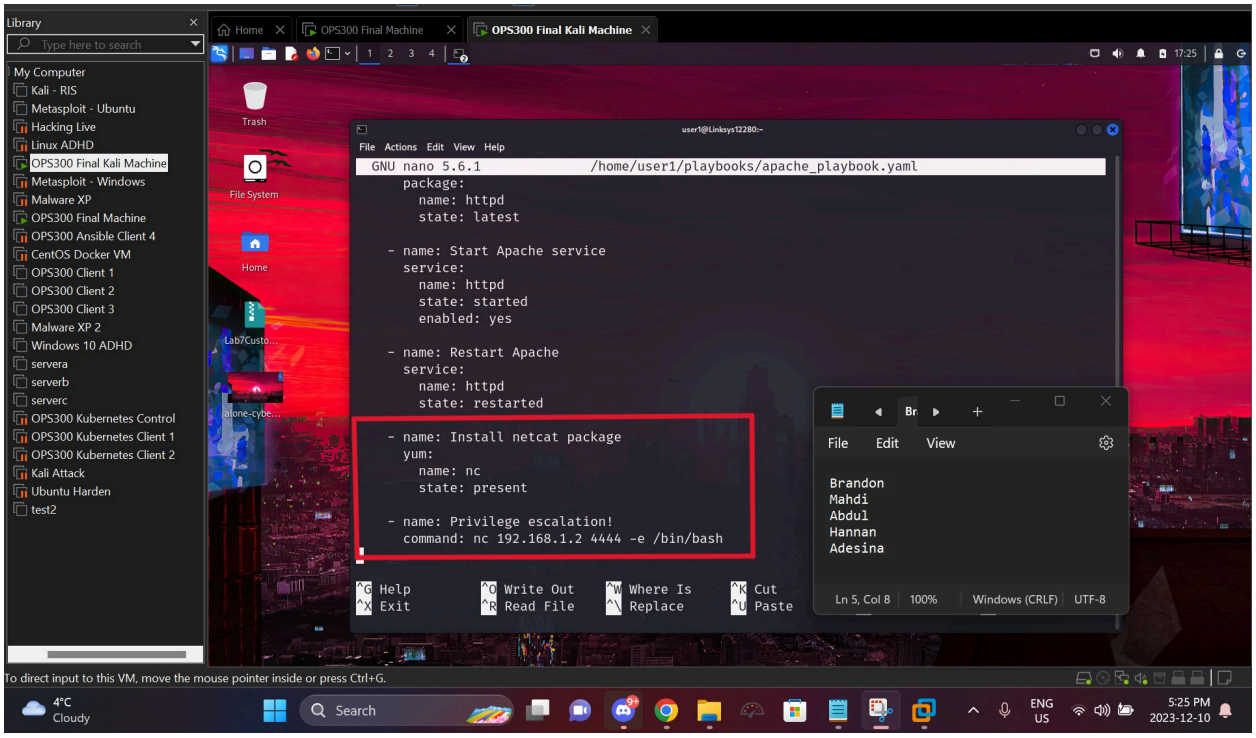


Screenshot 12: This screenshot shows the attacker attempting to see the contents of the script (encountered in the previous screenshot). The script seems to be running the playbook file (located in /home/user1/playbooks/apache_playbook.yaml) which user1 has failed to previously do.

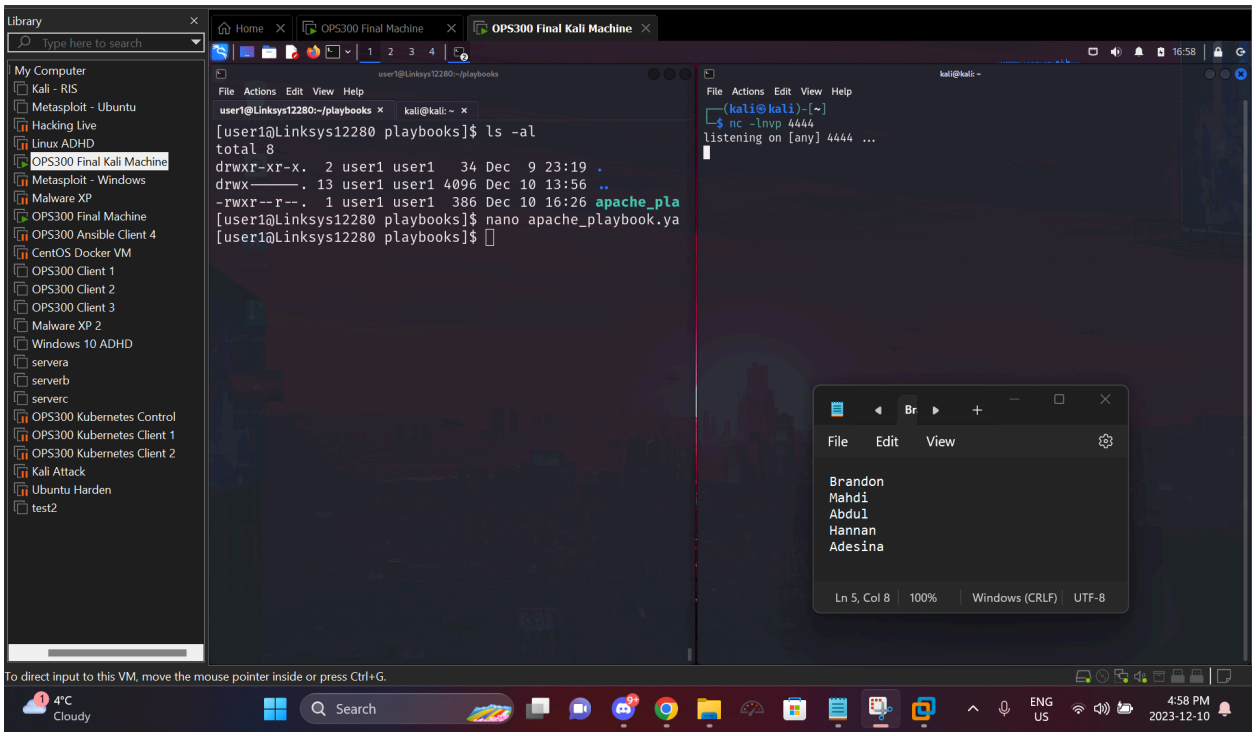


Screenshot 13: The following screenshot displays the attacker executing the command "ls -al," providing a detailed listing of all file names/associated attributes in the current working directory. Notably, the screenshot reveals the "ansible_apache_script.sh" script currently lacks editing permissions for user1, but user1 can execute it.

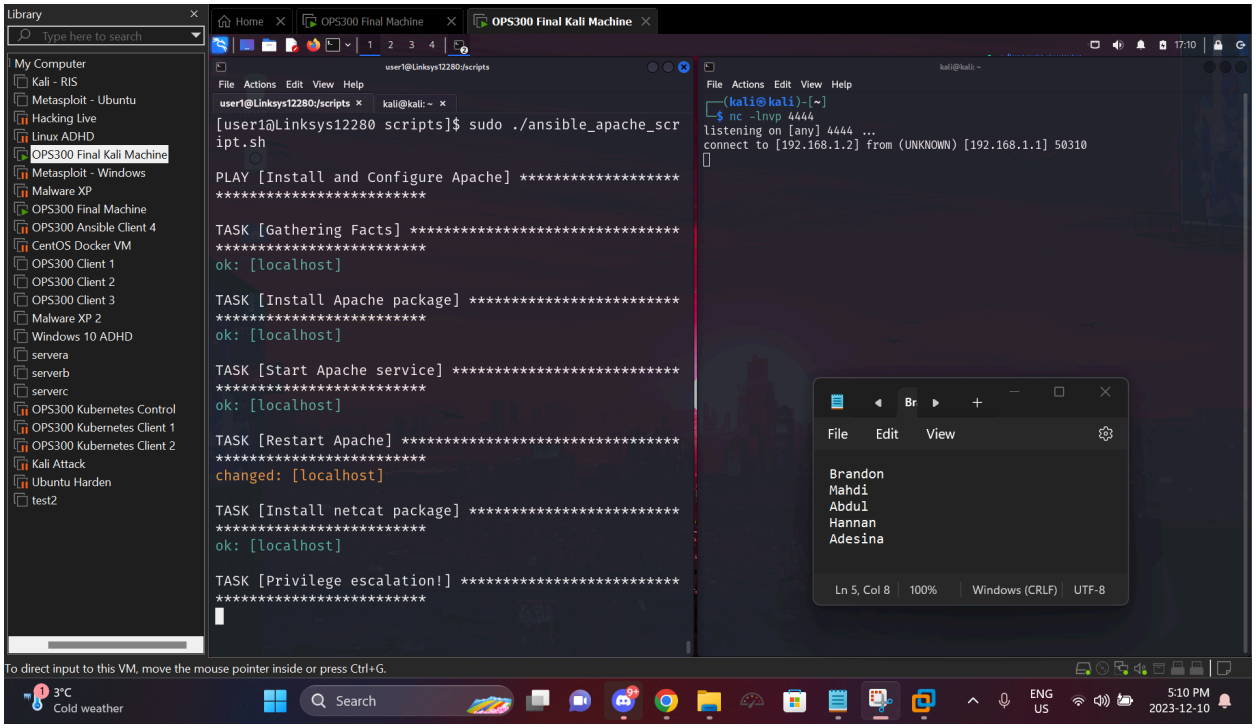
Since the /scripts/ansible_apache_script.sh script can be executed with sudo privileges by user1, and it runs the /home/user1/playbooks/apache_playbook.yaml playbook, an attacker can exploit this functionality. Leveraging the sudo permissions granted through the script, an attacker can gain the ability to execute additional tasks within the ansible playbook as the root user. This presents a way for performing actions with elevated privileges, enabling the attacker to further manipulate the system.



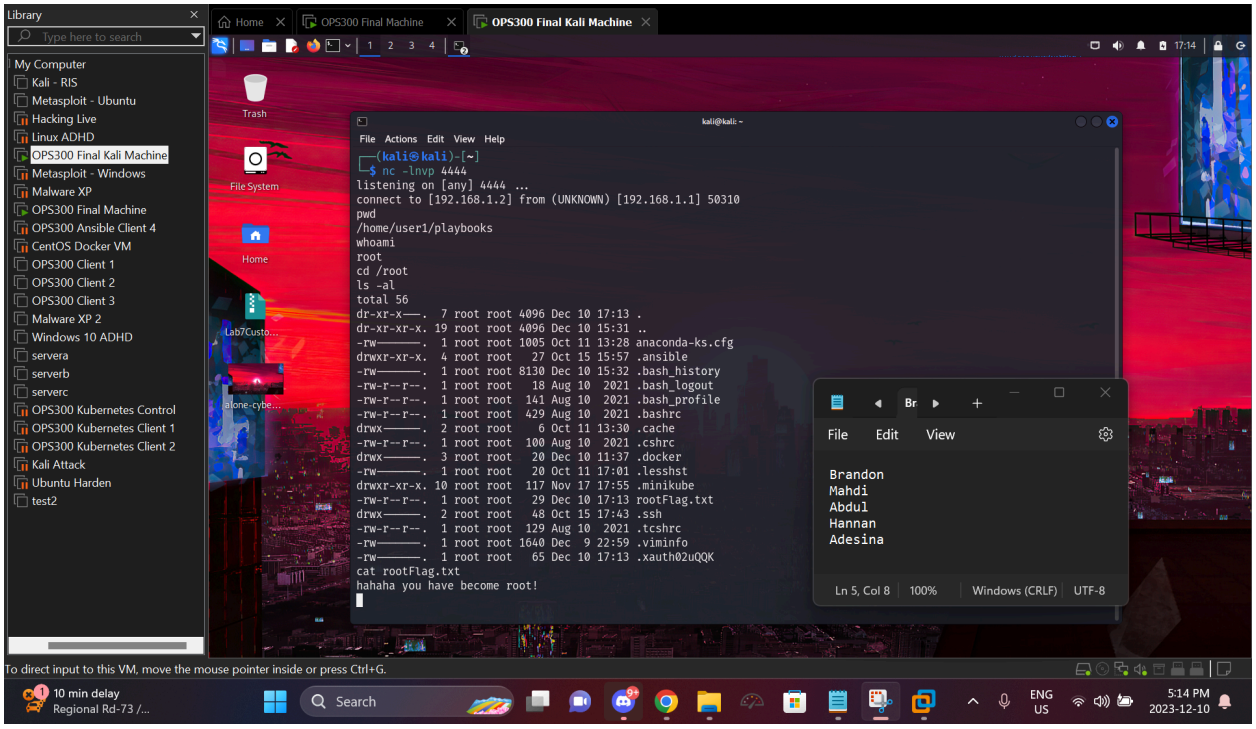
Screenshot 14: This screenshot shows the attacker modifying the original playbook to include two additional plays. The first play installs netcat on the target CentOS machine, and the second play uses the freshly installed netecat to send a reverse shell connection to the attackers machine (192.168.1.2) and execute /bin/bash to give the kali attacking machine an elevated working shell.



Screenshot 15: This screenshot shows the setup of a netcat listener listening on port 4444 on the attacker's machine (right side terminal). The intended goal is to be on stand-by and wait for the privilege escalation play in the apache_playbook.yaml to run, connect back to the listener, and establish an elevated shell.



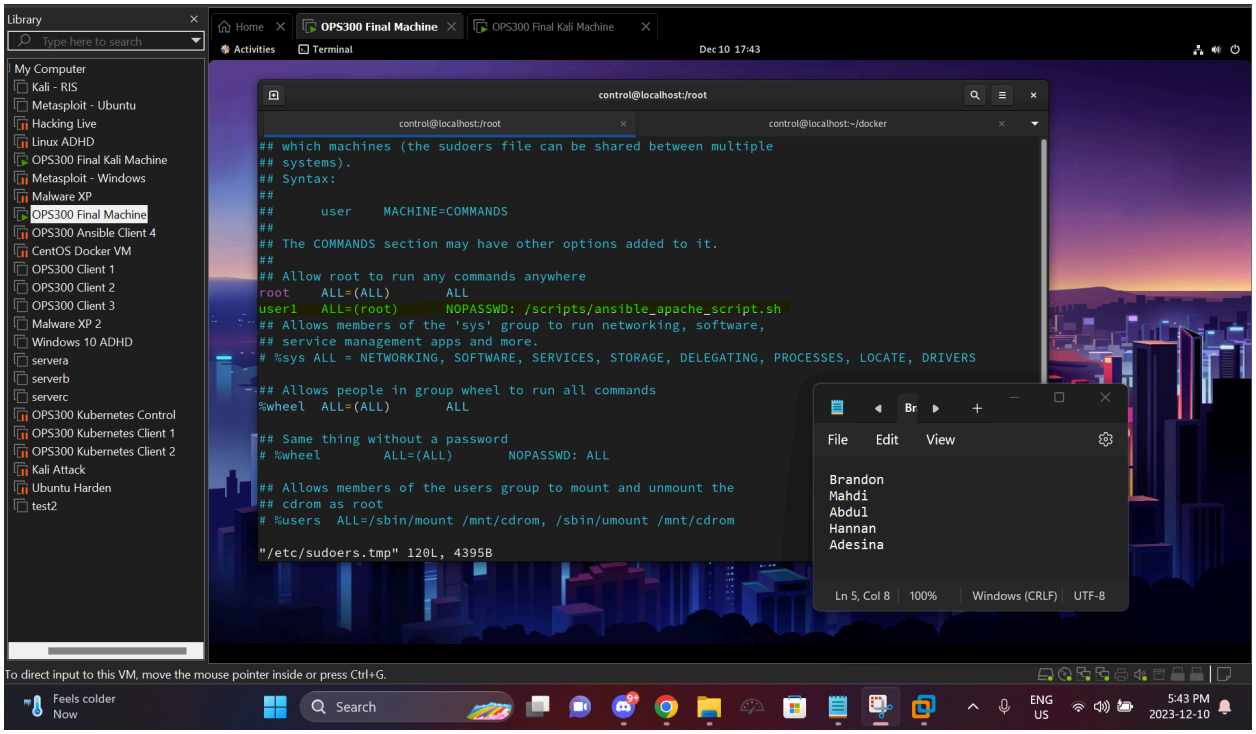
Screenshot 16: This screenshot shows the attacker running the previously discovered script (***ansible_apache_script.sh script***) since user1 has the appropriate permission to do so. Once the privilege escalation task executes, it sends a connection to the attacker's machine (right-side terminal), providing a bash shell.



Screenshot 17: This screenshot shows that the attacker went from a normal user account to a root account, by exploiting an ansible playbook that was run as the root user (through sudo). We see that the attacker utilizes the recently acquired escalated shell. The attacker ensures that they are in fact the root user by running ***whoami***, they then go into the root directory and cat out the contents of a text file owned by root.

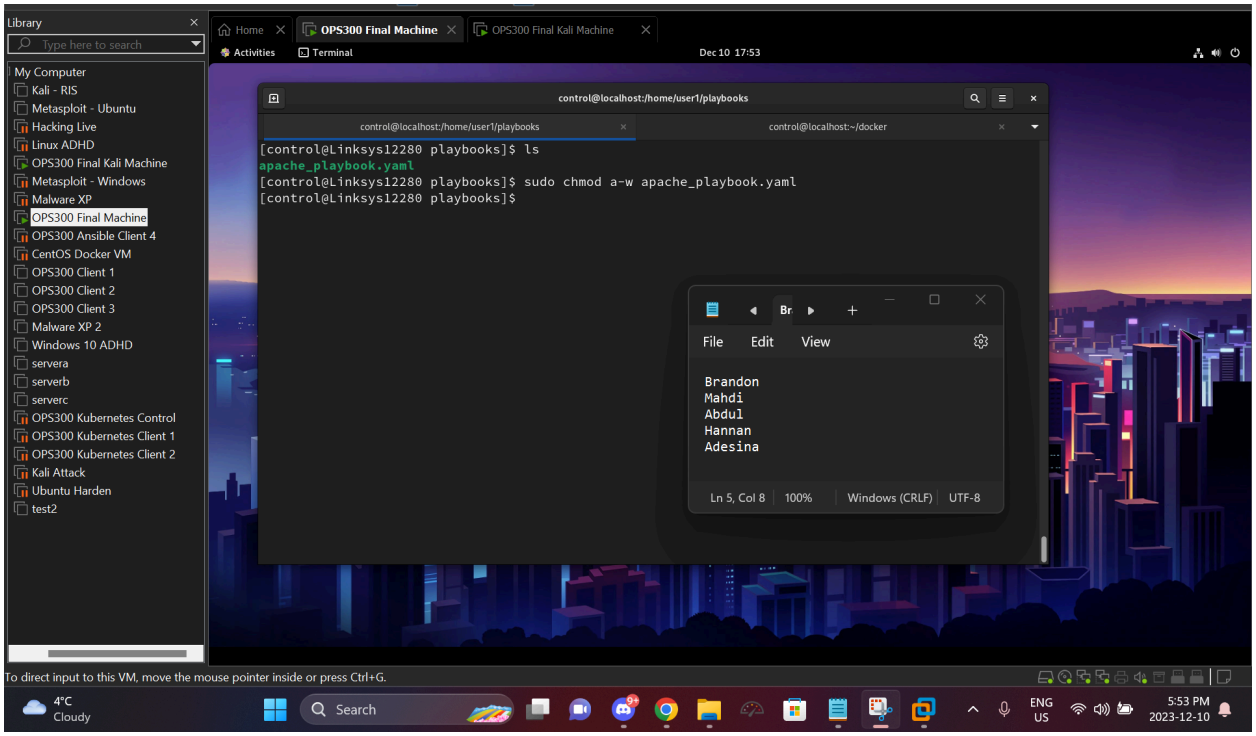
Prevention:

It is critical to provide strict restrictions to prevent any attempts by a regular user account to run an Ansible playbook with elevated root privileges, whether direct or indirect (e.g., via a script). Furthermore, strict measures should be in place to prevent unauthorized changes to a playbook that is being executed by a root user. This helps maintain the integrity and security of system configurations, lowering the risk of unauthorized access and potential exploitation of privileged activities.



Screenshot 18: This screenshot shows the "sudoers" file where as of right now there is a permission where user1 is currently able to execute the "/scripts/ansible_apache_script.sh" with root privileges. However an administrator will have to delete the line that is highlighted in the above sudoers file. This will remove the user1 permission to run the "/scripts/ansible_apache_script.sh" as a root user, which ensures the security of our machine, and prevents a standard user from indirectly executing the playbook as root.

In some environments, where users would not need to edit playbooks, the removal of write permissions from playbook files can help prevent them from being used for malicious purposes. This prevents standard users from making unauthorized edits to Ansible playbooks, ensuring the integrity and reliability of the scripts. By restricting write access, organizations can mitigate the risk of unintended or malicious changes to playbooks, resulting in a more secure environment.

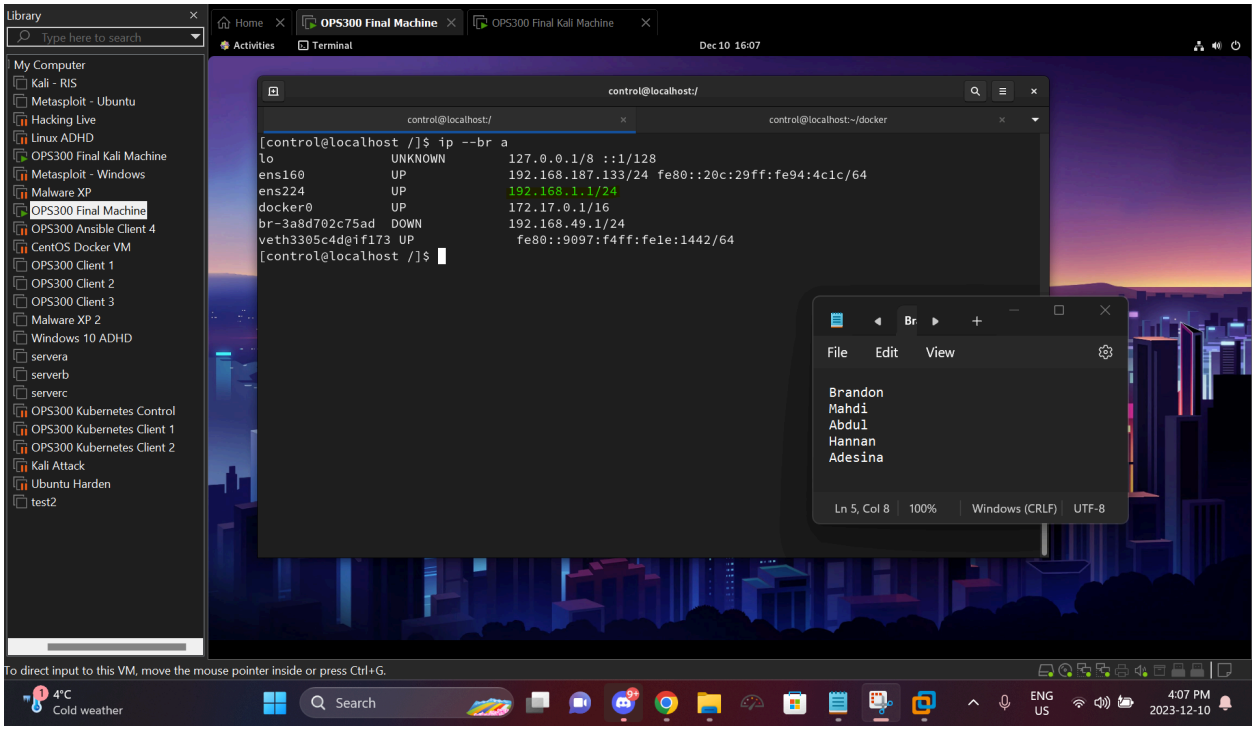


Screenshot 19: This screenshot shows the removal of write permissions on the ansible playbook for all users. This means that the user1 account can no longer edit the ansible playbook, and add malicious plays to create a reverse shell that connects back to the attackers machine.

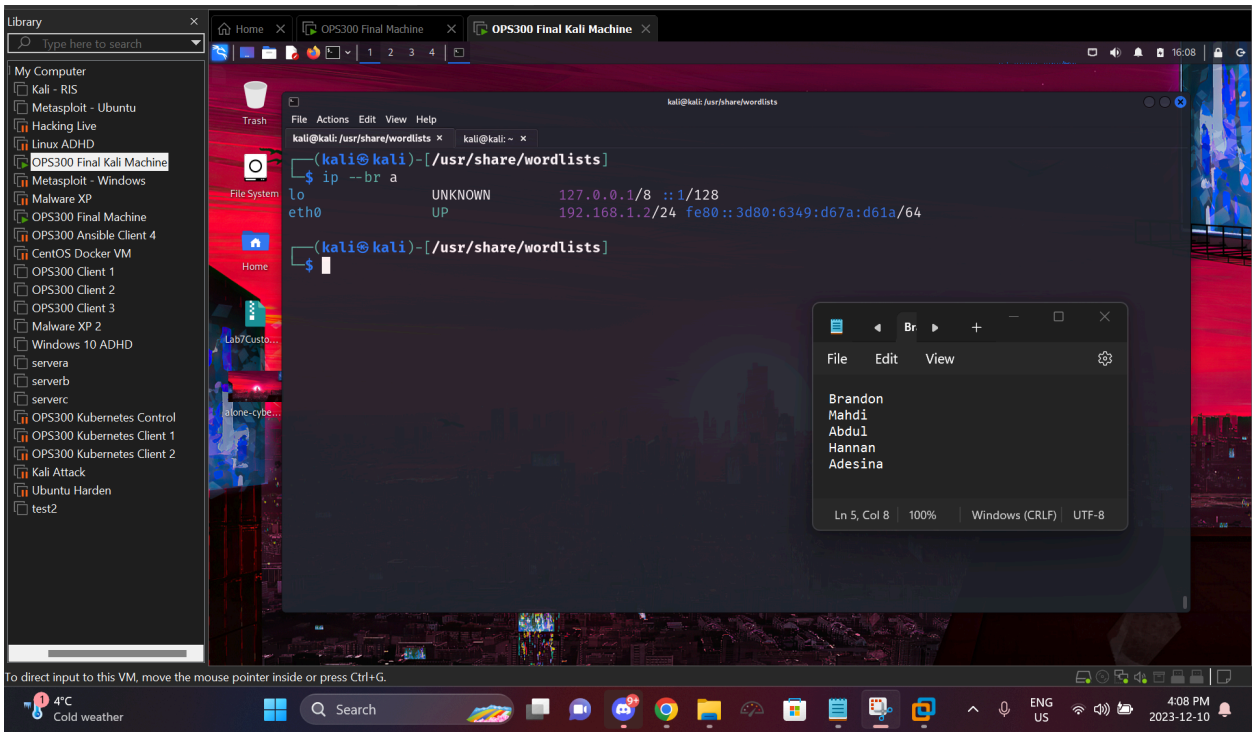
Escaping Docker Containers:

Environment Setup:

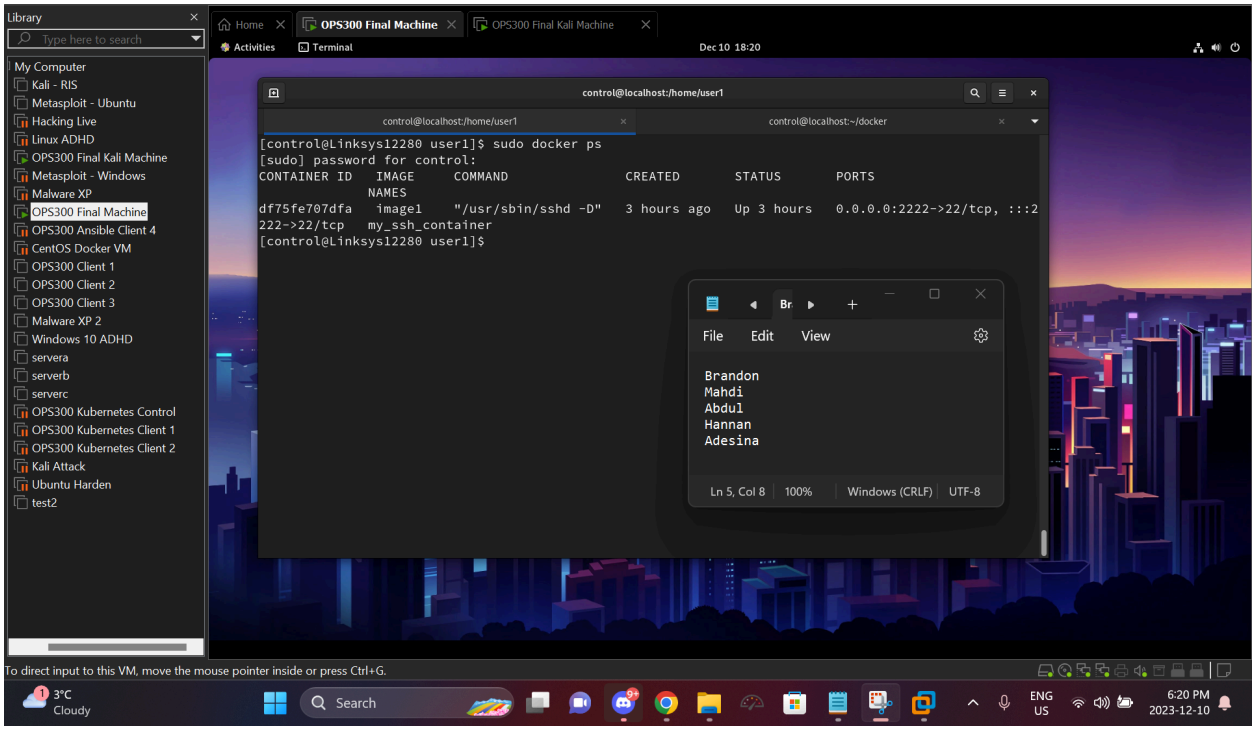
This environment is identical to the one set up in the previous section. The primary system is a CentOS Stream 9 Virtual Machine (VM), designated as the target machine, while the secondary machine is a Kali VM. These machines are seamlessly interconnected within LAN segment 1, ensuring a working network environment. The target CentOS machine is running an SSH server on it, and an Ubuntu based docker container that can be connected to using an SSH client through port 2222 on the host machine. This docker container has the hosts docker.sock file mounted to it, which can be used to communicate with the hosts docker daemon, which will be abused for container escaping in later steps.



Screenshot 1: This screenshot shows the target CentOS Stream 9 VM with an IP address 192.168.1.1/24.

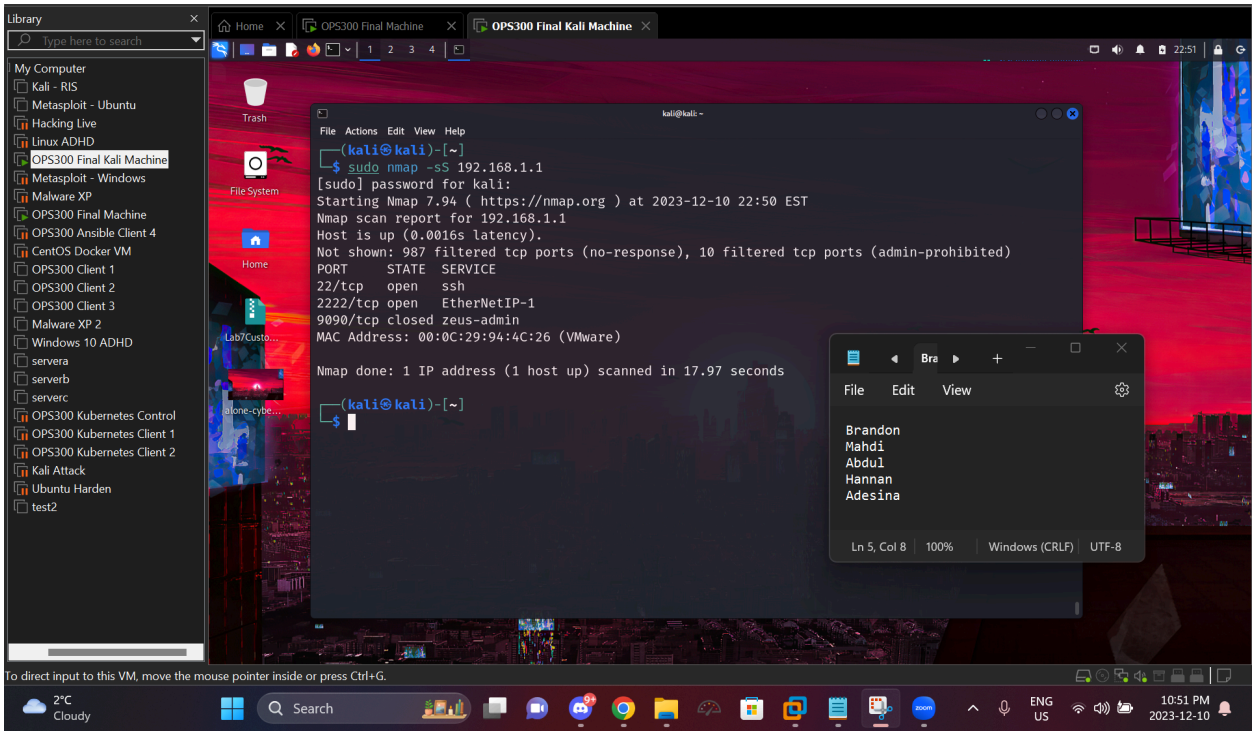


Screenshot 2: This screenshot shows the attackers Kali VM with an IP address 192.168.1.2/24.



Screenshot 3: This screenshot displays the active docker container on the target's CentOS machine which can be accessed via SSH through port 2222 on the host.

Enumeration Of Target:

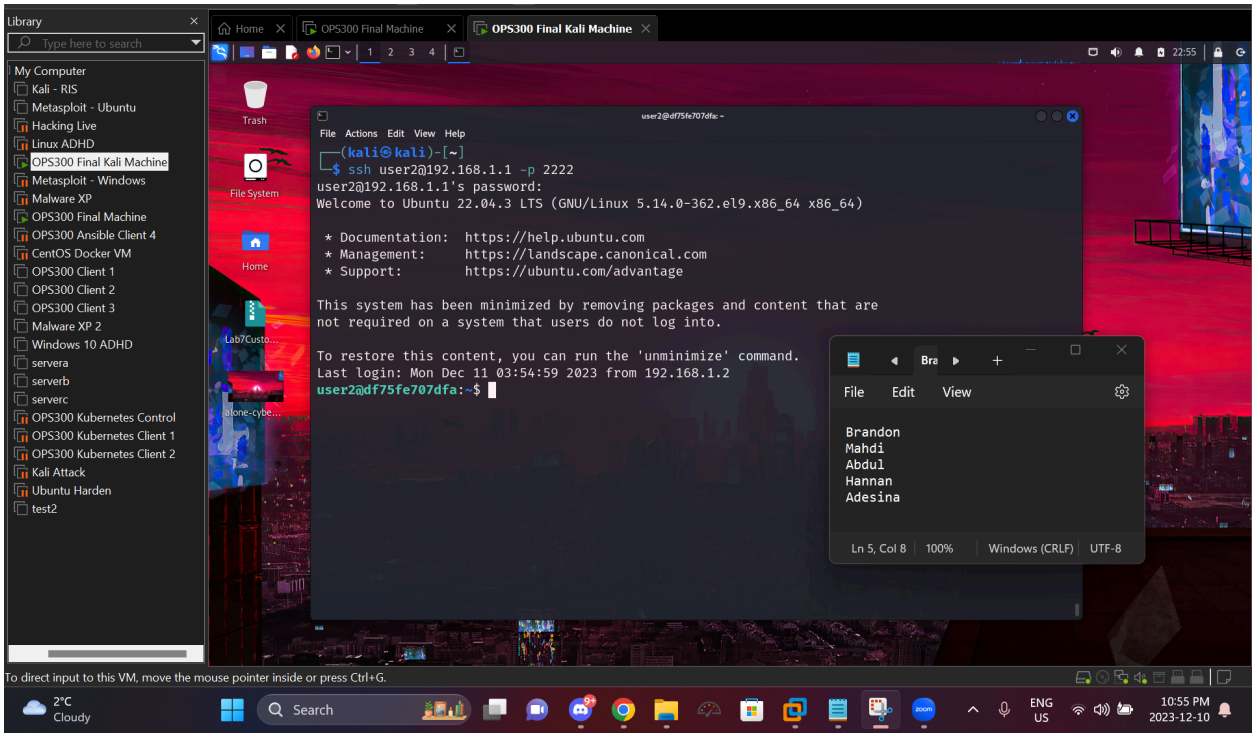


Screenshot 4: This screenshot shows a nmap scan ran against the CentOS machine. We can see

port 2222 open on the CentOS machine which can allow us to connect to the docker container through SSH.

Accessing the Docker Container:

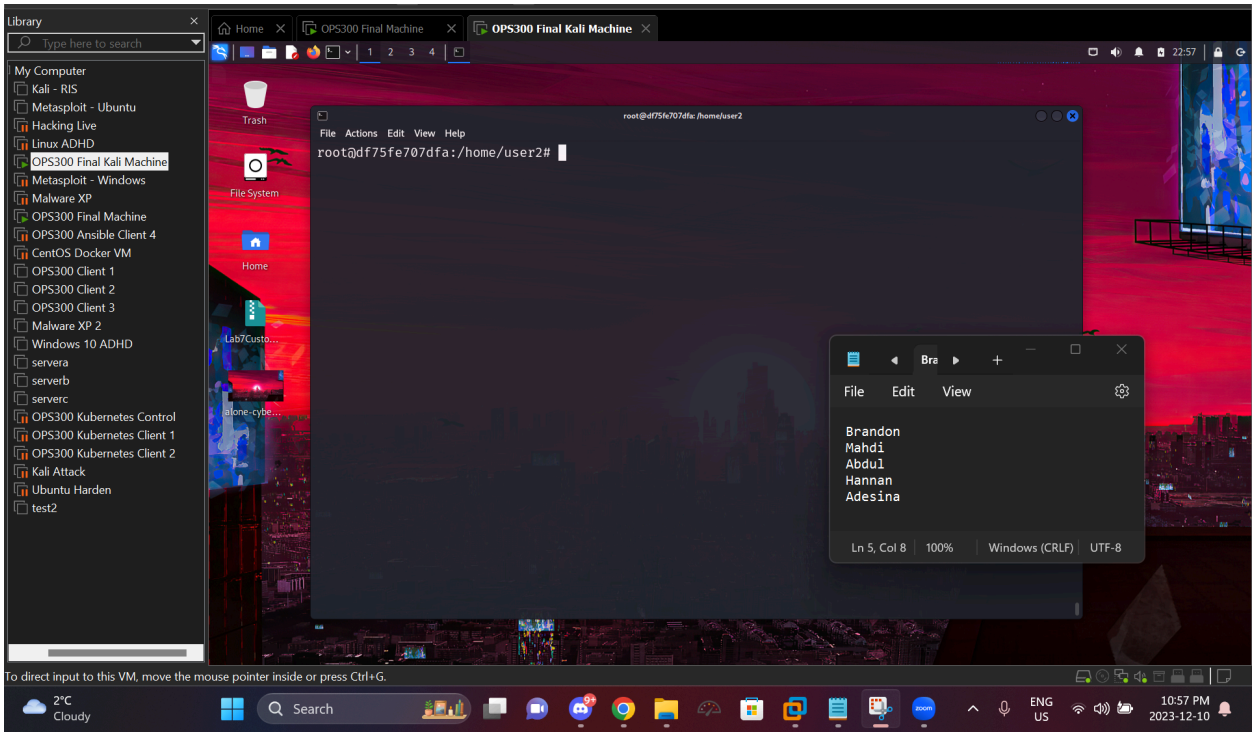
Note: We assume that the root credentials to SSH onto the docker machine were already obtained through methods such as social engineering.



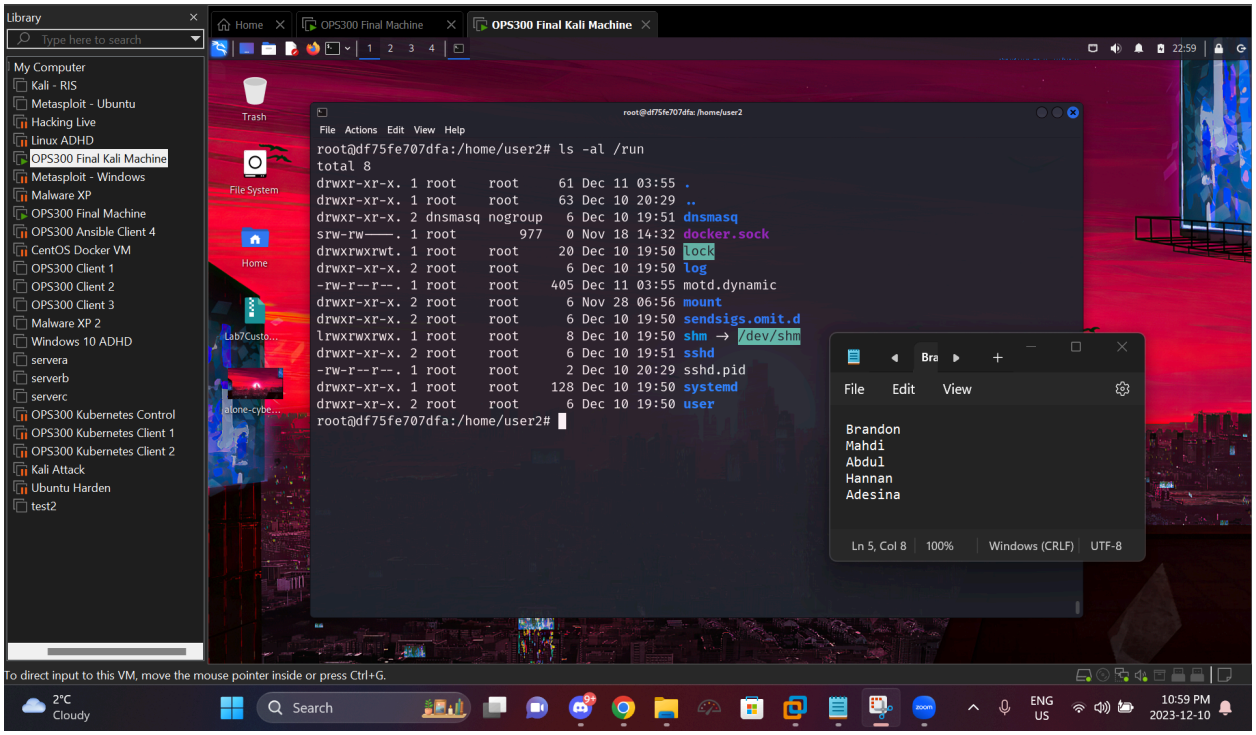
Screenshot 5: This screenshot shows the kali machine connecting to the docker container using SSH through port 2222 on the centos machine. We can see that the login to the docker container was successful.

Docker Container Escape #1 - Mounted Docker Socket:

Note: This method assumes that the attacker was able to escalate privileges to a root user within the docker container.



Screenshot 6: This screenshot simply demonstrates that the attacker has been able to get root access on the docker container.

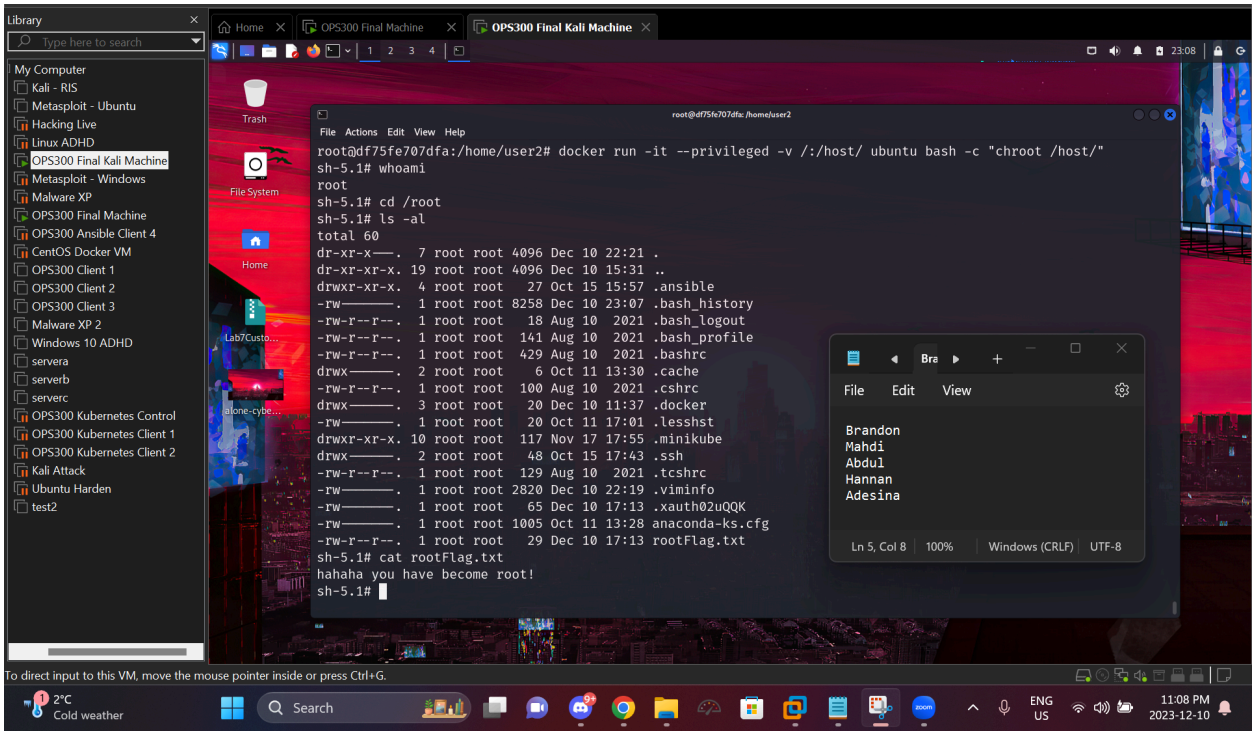


Screenshot 7: This screenshot shows that the host docker socket is mounted into the docker container that we are in (docker.sock). This was identified through the use of the “ls- al /run” command.

Docker.sock is the Unix socket that the docker daemon listens on by default. If mounted to a docker container it can be used to communicate with the host docker daemon from within the container. The docker daemon listens for Docker API requests and manages Docker objects such as containers, images, networks and volumes and can also be used to communicate with other daemons to manage docker services.

Docker.sock in a docker container allows for direct access to the docker daemon which allows control over Docker on the host system. You can run commands with elevated permissions as docker by default runs containers as root. Many people mount docker.sock to be able to run a container within a container, to manage other containers from a container, and also for logging purposes and auto service discovery. It is important to understand the security concerns that come with doing this.

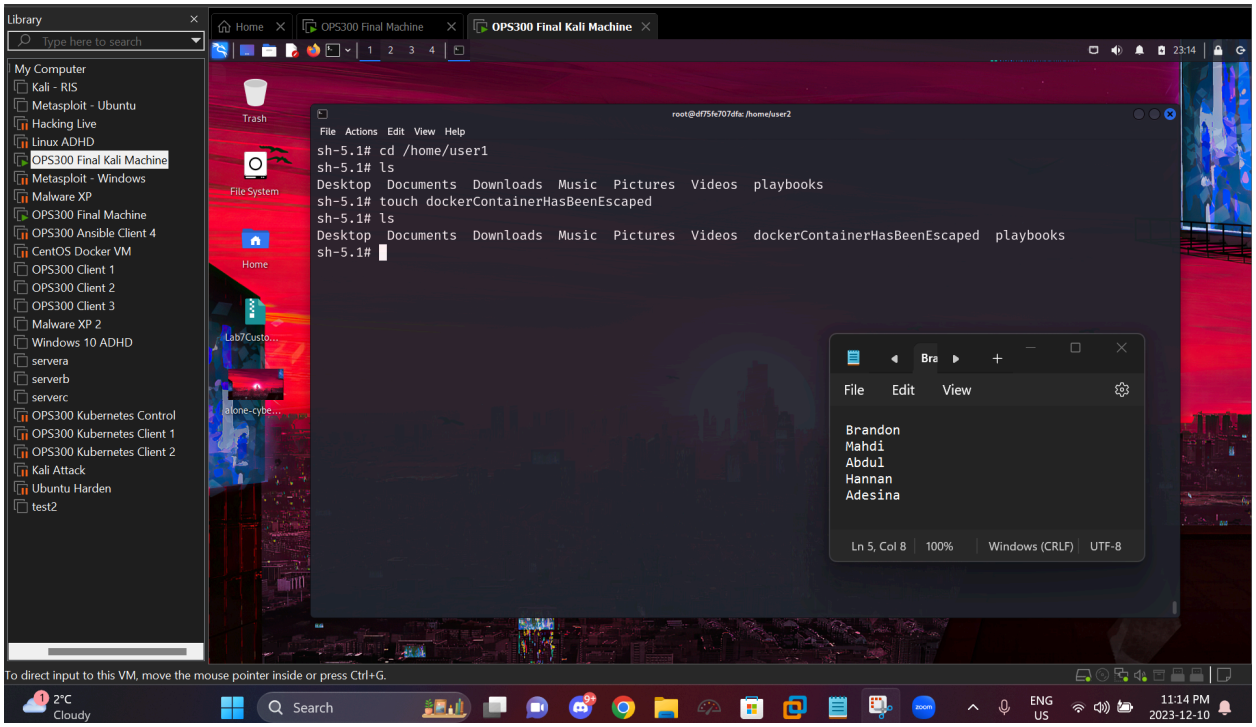
An attacker can abuse docker.sock by leveraging it to access the Docker daemon on the host machine to execute privileged commands within a container. For instance, they can exploit Docker.sock to communicate with the host's Docker daemon, creating a new docker container directly on the host machine. By mounting the entire host file system to the newly created container, the attacker will gain unrestricted access to the host machine. Access to the docker socket can also allow the attacker to listen in to and connect to the host's docker API.



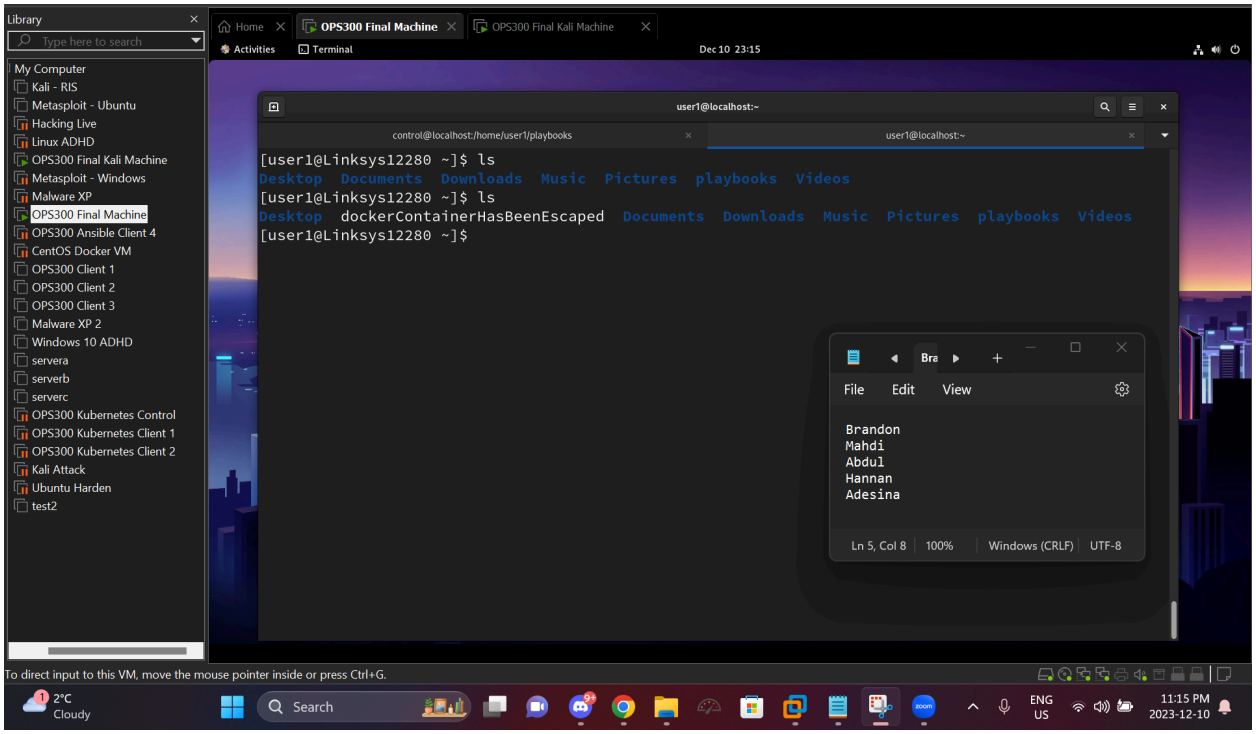
Screenshot 8: This screenshot shows the attacker running the command ***docker run -it --privileged -v /:/host/ ubuntu bash -c "chroot /host/"*** to escape the container. Here's a breakdown of this command, ***docker run -it*** creates a container and starts an interactive session with the container. ***--privileged*** is used to give the container full access to the host system and ***-v /:/host/*** is used to mount the root directory of the host system as /host/ inside the container.

ubuntu is the name of the Docker image that will be used to create the container and finally ***bash -c "chroot /host/"*** will be executed inside the container to start a new shell (bash) and then use the chroot command to change the root directory of the container to the root directory of the host system (/host/). Once this command was executed, we were able to fully control the host machine as the root user, proving that the docker container was fully escaped (we went into the root directory, and viewed the contents of a file owned by root).

To further prove that we are in fact controlling the live host machine, we will add a file to the user1 directory from the escaped docker container, and check to see if this file exists on the actual CentOS machine.



Screenshot 9: The following screenshot displays us moving into the "/home/user1" directory through the escaped docker container. As shown in the above screenshot we have added a file named "dockerContainerHasBeenEscaped" to it.



Screenshot 10: This screenshot shows us running ***ls*** in the target VM before and after the file was added by the attacker. Since the file was present after the attacker added it, it shows that we are able to fully control the centOS target host machine through the escaped docker container shell.

Prevention: Docker Container Escape #1 - Mounted Docker Socket:

Mounting docker.sock into the docker container is a bad practice because it gives the container root privileges which attackers can use to execute any commands that the docker service can use on the host machine. While it's recommended to avoid mounting docker.sock, if you have no choice but to use it, carefully consider the information a container contains, grant containers only access to the information it needs while using docker. sock, and set the information as read-only which will prevent the user from changing the content within. To restrict operations by the user accounts, you may enable the Transport Layer Security or authorization proxies. Docker can also block access to a socket through HTTP and provide additional authentication and authorization features via role-based access control configurations. Additionally, you may use Docker Bench for security, an automated auditing script from Docker Inc that checks the deployment against the Center for Internet Security benchmarks and best practices around container deployment

Some alternatives we can use are:

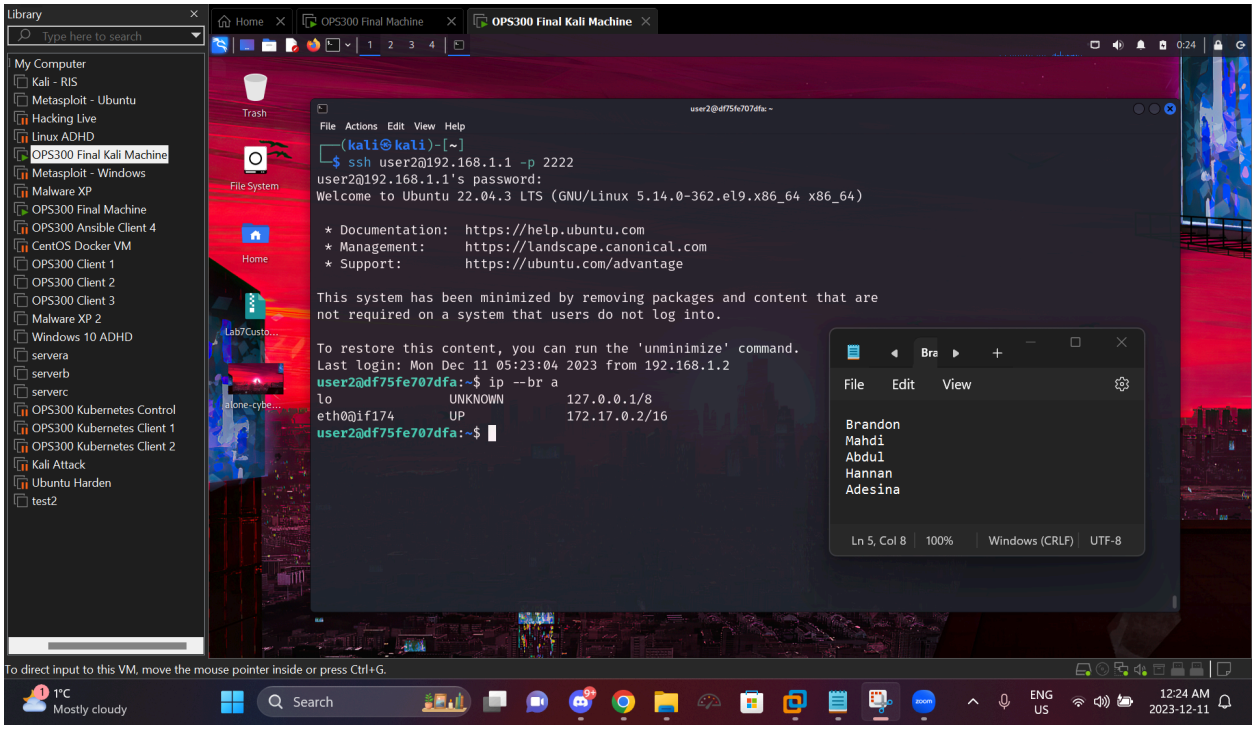
- HAProxy is an open-source load balancer that can be configured to support specified endpoints or remove docker.sock access.
- Docker-proxy-acl is a Unix socket proxy that can restrict or enable access to endpoints like containers, images, volumes and networks. It can also prevent users from building up

additional docker files which can be used to pass malicious instructions to a docker container image.

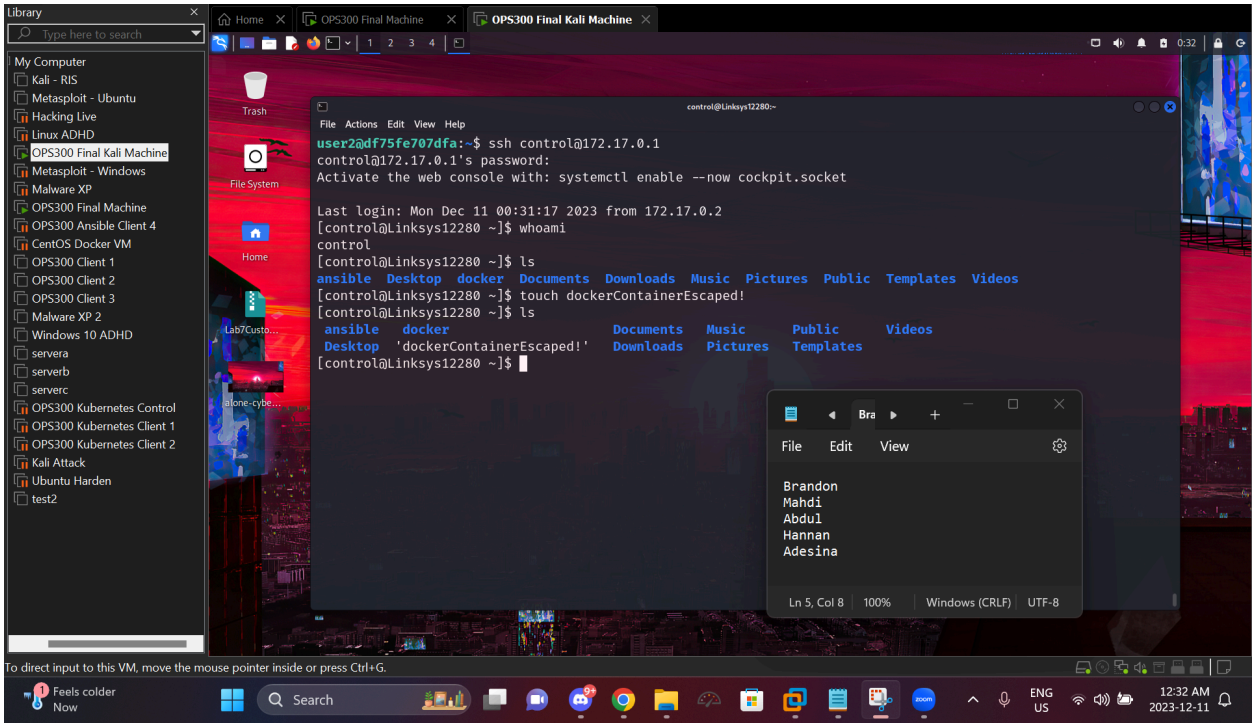
- Docker CLI can be installed in the container and configured to connect to a remote Docker Daemon. This way the docker socket isn't exposed and we can interact with docker through the command line.
- Docker-in-Docker (DinD): while not recommended for production, DinD involves running a docker daemon inside a docker container allowing it to manage its own sets of containers and images. This comes with its own sets of complexity and security concerns.
- Docker compose is a great alternative if the goal is to manage multi-container applications. It allows us to define and run multi-container Docker applications using a YAML file.

Docker Container Escape #2 - SSH onto the Host machine:

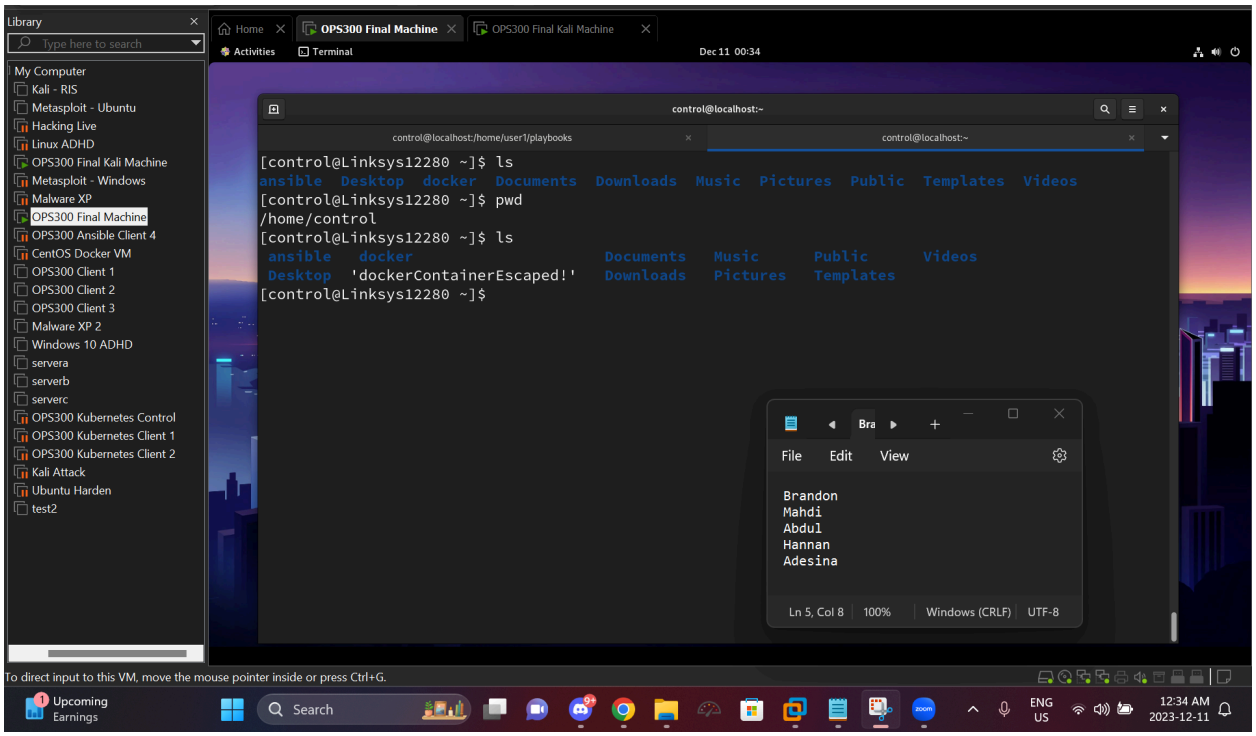
The default Docker network configuration assigns the IP address 172.12.0.1 to the host machine. Since we know the host docker IP address, it means that if SSH is running on this host, it should be possible to SSH into the host machine from within the Docker container, provided that the following conditions are met: (1) The user has valid SSH credentials for the host machine and (2) An SSH client application can be accessed/ran from within the docker container.



Screenshot 11: This screenshot shows that we have successfully SSHd onto the docker container as the user1 account. And that the available network interface has an address of 172.17.0.2. Before moving forward it is important to note that we are assuming that valid SSH credentials for the CentOS target machine have already been obtained.



Screenshot 12: This screenshot shows us SSHing into the host machine as the control account (once again assuming that we have already obtained valid SSH credentials to this machine). Once inside we have created a file called “dockerContainerEscaped!” and left it in the control users home directory.

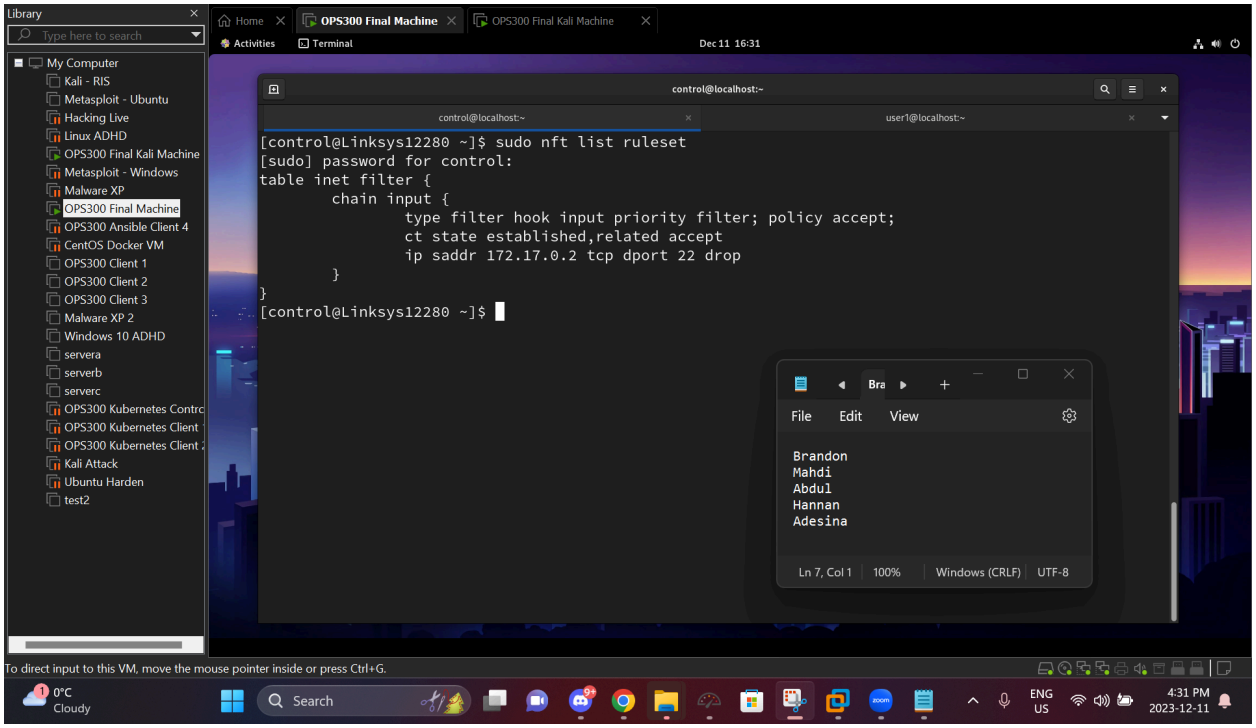


Screenshot 13: This screenshot shows us running ***ls*** in the CentOS host before and after the "dockerContainerEscaped!" file was added by the attacker to the control users home directory. This proves that we were able to fully escape the docker container.

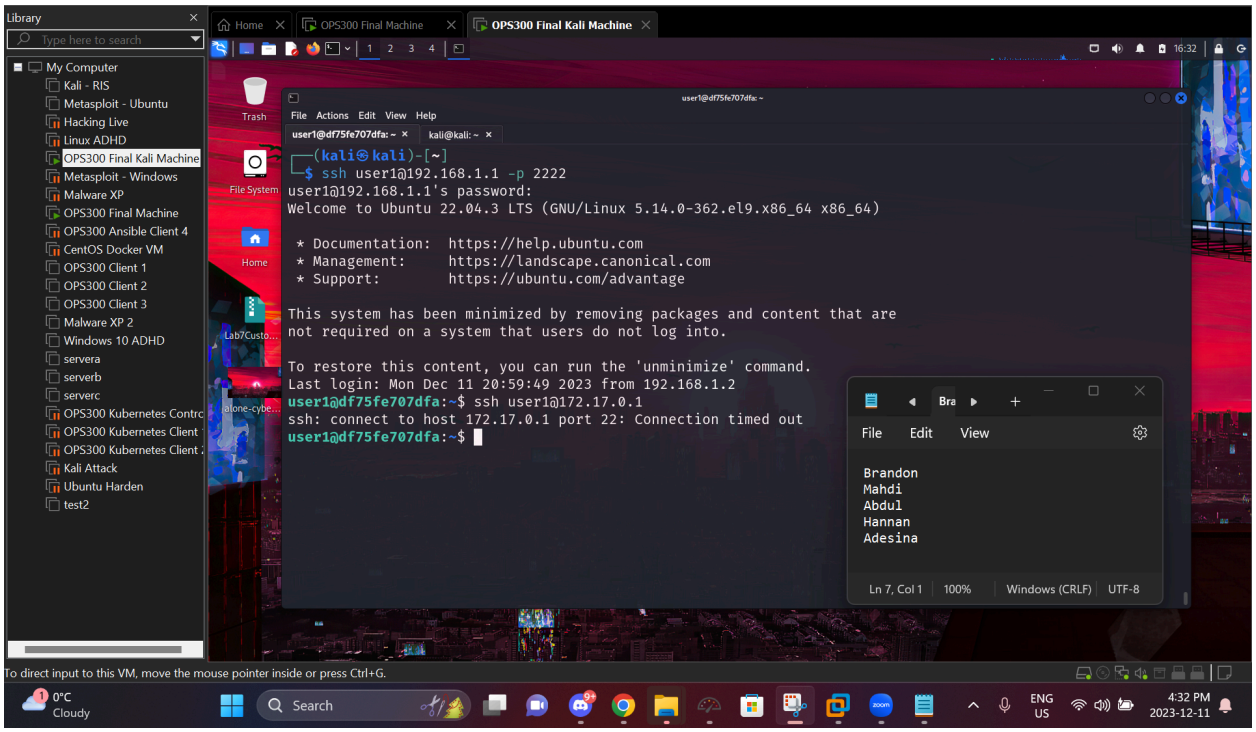
Prevention: Docker Container Escape #2 - SSH onto the Host machine:

If the host machine doesn't need to run an SSH server (a service that allows secure communication between machines), the best practice is not to have it running in the first place. This reduces the risk of an attacker using an SSH client to break out of the Docker container and access the host machine.

However, if SSH is required on the host machine for specific reasons, safety measures can be taken. For example, firewalls can be implemented to create a protective barrier, ensuring that Docker containers cannot access the SSH service on the hosting machine. Additionally, to enhance security further we can remove the SSH client from within the Docker container. This is equivalent to deleting any potential tools inside the container that may be used to connect to the host machine's SSH service. This adds an extra layer of protection, ensuring that the container has no means to establish a connection with the SSH service running on the host machine. This way, even if an attacker does gain access to the container, they won't have the necessary tools to exploit an SSH connection to the host.



Screenshot 14: This screenshot shows the nftables rules put in place to prevent the docker container from being able to SSH onto the host machine.



Screenshot 15: This screenshot demonstrates a failed attempt by the attacker to SSH on to the host machine from within the docker container after the firewalls have been established.

References

Abargil, O. (2023, August 28). *7 ways to escape a container*. Panoptica. https://www.panoptica.app/research/7-ways-to-escape-a-container

Baeldung. (2023, June 15). *How to give sudo privileges to a user in linux*. Baeldung on Linux. https://www.baeldung.com/linux/sudo-privileges-user

Docker. (n.d.). *Docker Overview*. Docker Documentation. https://docs.docker.com/get-started/overview/

Educative. (n.d.). *var/run/docker.sock*. https://www.educative.io/answers/var-run-dockersock

Gillis, A. S. (2019, January 25). *Get informed of the risks associated with docker.sock*. IT Operations. https://www.techtarget.com/searchitoperations/tip/Get-informed-of-the-risks-associated-with-dockersock

Hacktricks. (n.d.). Docker breakout / privilege escalation . https://book.hacktricks.xyz/linux-hardening/privilege-escalation/docker-security/docker-breakout-privilege-escalation

Hammond, J. (2022, February 3). *SQLi, SSTI & Docker Escapes / Mounted Folders - HackTheBox University CTF "GoodGame."* [Video]. YouTube. https://www.youtube.com/watch?v=0oTuH_xY3mw

Jain, N. (2022, March 27). *TryHackMe: Aratus*. Medium. https://infosecwriteups.com/tryhackme-aratus-62751adaa9f6

Jun, C. M. (2023, December 4). *The netcat command in linux*. Baeldung. https://www.baeldung.com/linux/netcat-command

Ramos, M. (2023, September 13). *How to SSH into a Docker Container*. Kinsta. https://kinsta.com/blog/ssh-into-docker-container/

Stack Overflow. (2016, April 1). *How to Mount Docker socket as volume in Docker container with correct group*. https://stackoverflow.com/questions/36185035/how-to-mount-docker-socket-as-volume-in-docker-container-with-correct-group

Wilson, B. (2023, November 6). *How to run Docker in Docker container [3 easy methods]*. DevopsCube. https://devopscube.com/run-docker-in-docker/