

Concordia Institute for Information Systems Engineering (CIISE)

Concordia University

**INSE 6130 Operating System Security**

**Project Report**

Submitted to:

**Professor Dr. Suryadipta Majumdar**

Submitted By:

| **Student Name** | **Student ID** | **Role** | **Implementation** |
| --- | --- | --- | --- |
| Avin Vincent | 40265132 | Attack | Using ‘docker socket’ |
| Likitha T Reddy | 40265131 | Attack | Using ‘sys\_ptrace’ |
| Challa Likitha | 40293973 | Attack | By injecting “Web Shell” |
| Chinnabathini Hima Bindu | 40292127 | Attack | Using ‘dac\_read\_search’ |
| Moksh Sood | 40294266 | Attack | Using unprotected TCP socket |
| Divya Varshini Murathoti | 40293222 | Security | Using Cgroups and Capabilities |
| Kunati Bala Krishna Yadav | 40292128 | Attack & Security | Using Disk Mount |
| Yash Khosla | 40232363 | Security | Using “Seccomp security” |

Table of Contents

**PART A – Implementation of Attacks on Docker** 3

1. **Attacking insecure volume mounts using ‘docker socket’** 3
2. **Attacking container with** ‘**sys\_ptrace**’ **capability** 4
3. **Attacking host system security with dac\_read\_search capability** 5
4. **Attacking docker Source Image by injecting a Web Shell** 7

**PART B – Implementation of Security Applications** 16

1. **Cgroups** 16
2. **Capabilities** 16
3. **Apparmor - Attack and Defence against Privilege Escalation using Disk Mount** 17
4. **Seccomp Profile** 19

**Challenges** 21

**Part A – Implementation of Attacks on Docker**

1. **Attacking insecure volume mounts using ‘docker socket’**

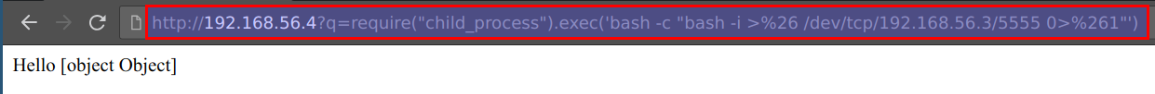
Docker socket is a tool used to communicate with the Docker daemon from within a container. In enterprise environments, Docker socket is typically enabled to facilitate container orchestration, automate deployment processes, and enable monitoring and logging solutions to interact with the Docker daemon for performance analysis and security auditing. Enabling the docker socket provides the docker with the volume mounting feature directly to the host. In this attack, we're Using volume mounted docker.sock to gain privileges in the host system.

**Execution:**

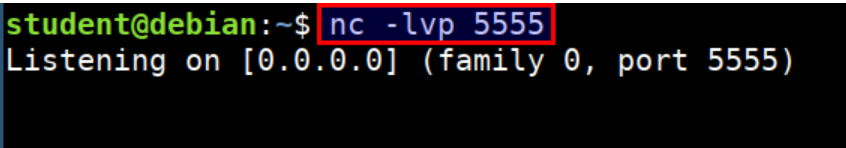
1. Initially, we verify whether a Node.js program—known to be susceptible to Remote Code Execution (RCE)—is operating on the target system's container. By providing a query parameter to the application, we confirm the exploitability of the RCE vulnerability.



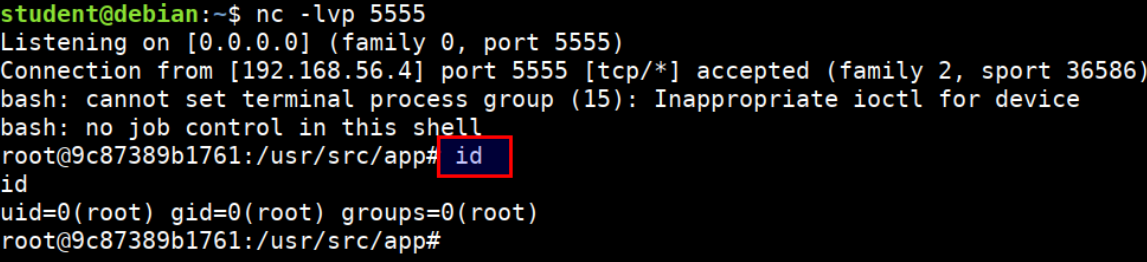
1. Once the RCE has been verified, we take advantage of it by inserting a payload that is a reverse shell, which gives us access to its surroundings remotely.



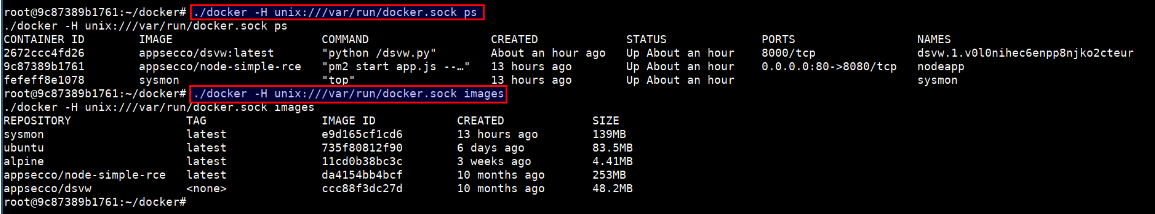
1. To intercept the reverse shell connection, we set up a listener on our student virtual machine using Netcat.



1. Once the reverse shell connection is established, we leverage the docker.sock socket to access host resources directly from the compromised container.



1. With access to the host system, we gain control over other containers present on the host, enabling us to execute, delete, or modify them as desired.



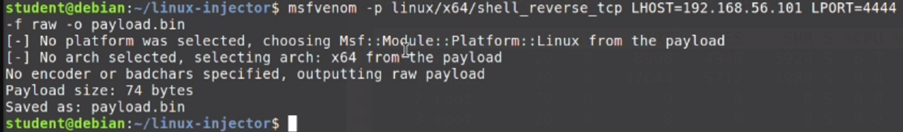
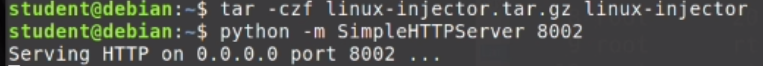
1. **Attacking container with ‘sys\_ptrace’ capability**

In Unix-like systems, sys\_ptrace helps one process to monitor and manage another. A container that has the SYS\_PTRACE capability enabled in Docker can make use of sys\_ptrace system calls inside its isolated namespace. In business or enterprise environments, enabling Sys\_ptrace in Docker containers is permissible for monitoring, debugging, and gathering system-level performance metrics using programs like perf. However, it does pose security problems because it allows container activities to track down and control other processes on the host, which could result in privilege escalation or unauthorized access.

We are breaking out the container in this attack by taking advantage of the sys\_ptrace capabilities.

**Execution:**

1. We are using a **simple Python server** for transferring the Metasploit **msfvenom** reverse shell payload and injector application to the container.

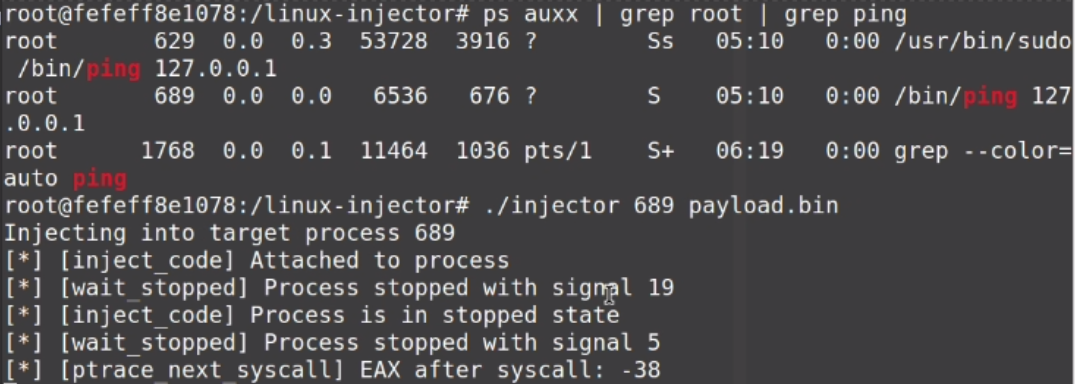
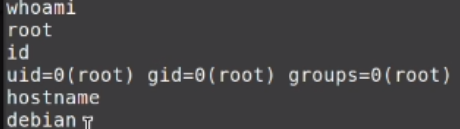
  
  


1. Using the **Curl** command to download the payload from the virtual machine container.



1. Using **netcat** listener to connect back shell in the student virtual machine.



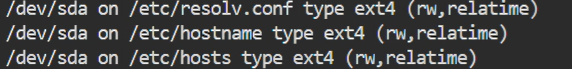
1. Once inside, we use the **ps auxx** and **grep** commands to identify the process that is executing as root on the host system and obtain root access to connect back. 
2. After the payload is successfully injected, our listener receives a reverse connection that allows access to the host system outside of the container.
3. **Attacking host system security with ‘dac\_read\_search’ capability**

The cap\_dac\_read\_search capability in Docker grants a container unusually broad access to the filesystem, allowing it to read files and traverse directories that might otherwise be restricted based on file and directory permissions. This capability is particularly significant in Docker because of the potential it has to bypass the usual isolation and security boundaries that containers are supposed to enforce, posing a significant security risk. If misused or exploited, this capability can lead to unauthorized access to sensitive files, facilitating data exfiltration and privilege escalation.

**Execution:**

1. First we are identifying the mounted files within the container, especially those mounted from the host and accessing SSH Configuration and Keys.

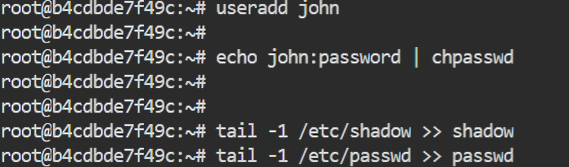




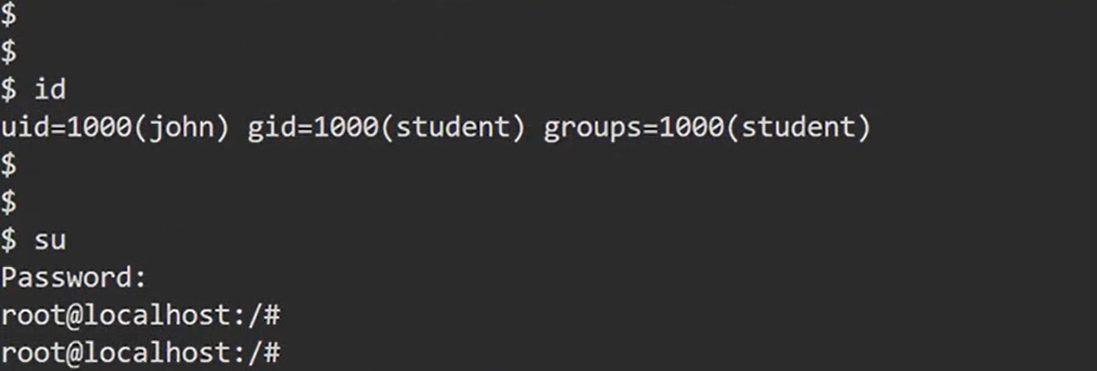
2. We look for exploits that specifically target CAP\_DAC\_READ\_SEARCH. Then We navigate to directories such as /etc/ssh/ for SSH configuration files and /root/.ssh/ for SSH keys, which are critical for remote access configuration.

3. Create a Local User and Modify Password Files

We open /etc/passwd and add a new user line, which creates a new root-level user. We open /etc/shadow and add a password hash for the new user. This step secures the new user with a known password.



4. Using the credentials of the newly created user, or by utilizing modified SSH keys/configurations, we then SSH into the host system. This step is crucial as it represents the breaking out of the container’s isolated environment and gaining unauthorized access to the broader host system.

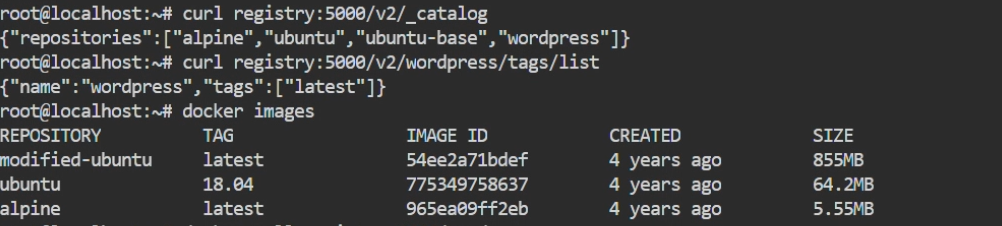


**4.Corrupting Docker Source Image**

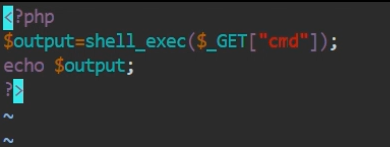
The purpose here is to modify an existing Docker image. Backdooring on WordPress Image allows an attacker to get remote control of the server. A web shell is a malicious software that an attacker can use to gain control of a web server. The WordPress image is modified by producing a Dockerfile for the Backdoored Image, which is a text file containing commands for modification.Once the Dockerfile is complete and the image has been produced, it must be submitted to the private Docker registry.Curl allows an attacker to interact with the web shell. curl is a command-line program for sending data over multiple protocols, including HTTP.The attacker can remotely execute activities on the hacked server by delivering particular commands to the web shell via curl.

**Execution:**

1. Use curl to interact with the private registry present on the network. Retrieve a list of tags available for the Docker image named wordpress from the Docker Registry server located at registry:5000.

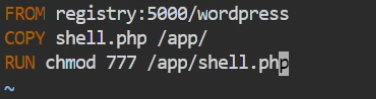


2.Backdoor the wordpress image by adding a webshell to it. Create a basic webshell in PHP.

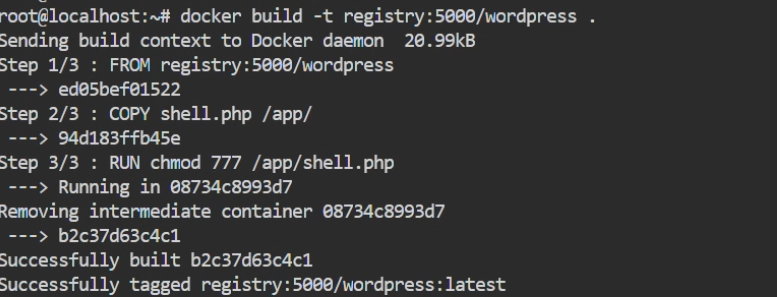


3.Creating a Dockerfile for the Backdoored Image,the Dockerfile will include commands to modify the WordPress image by adding the web shell.





4.Execute the docker build command to create the image containing the backdoor.



5.Push the newly created docker image to private registry.



6.After the corrupted image is deployed and the web server is running, an attacker can interact with the web shell using curl.

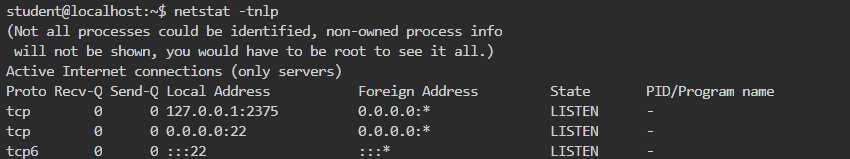


**5.Misconfigured Docker Socket**

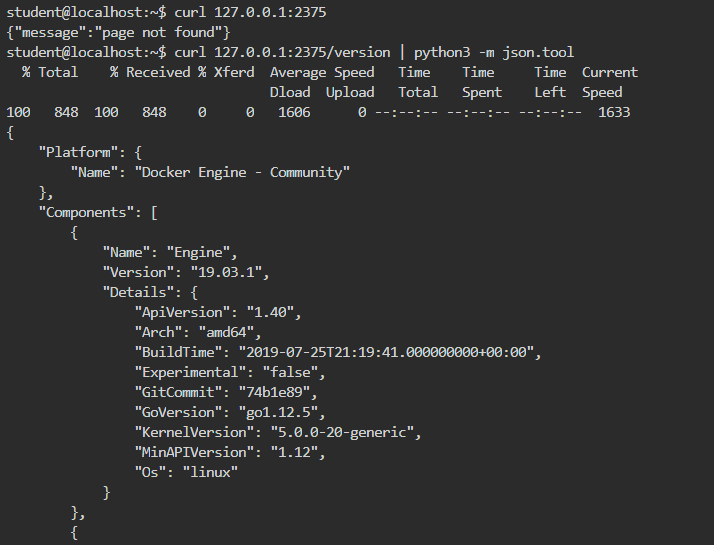
In this attack scenario, we are leveraging the Docker daemon's misconfiguration along with the curl command-line tool to interact with the Docker API. In this case, the curl tool is used to send HTTP requests to port 2375 on the Docker daemon in order to communicate with the Docker API. This enables us to carry out a number of tasks, including managing Docker resources, executing containers, and listing images. Because the Docker daemon is incorrectly configured to listen on an open TCP socket (port 2375), unauthorized users can access the Docker API and potentially compromise sensitive Docker functions.

**Execution:**

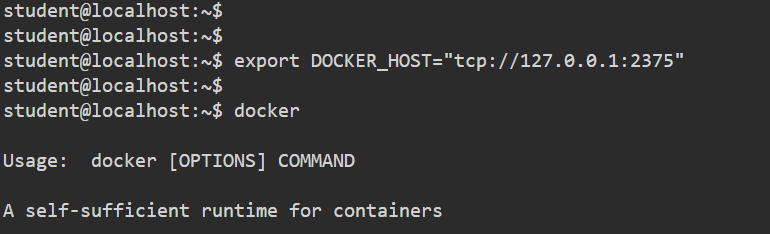
1.Check for Docker daemon on port 2375.



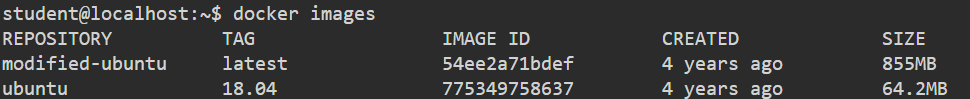
2.Confirm Docker daemon connectivity with curl.



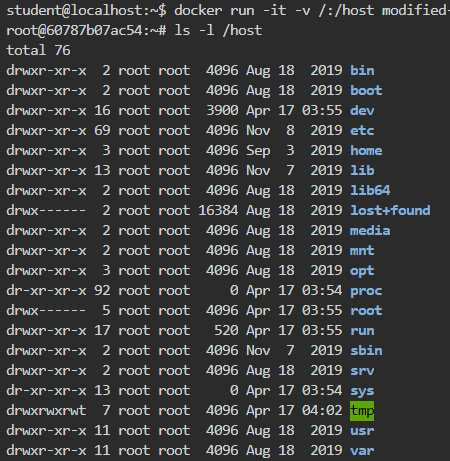
3.Set Docker client to use TCP socket.



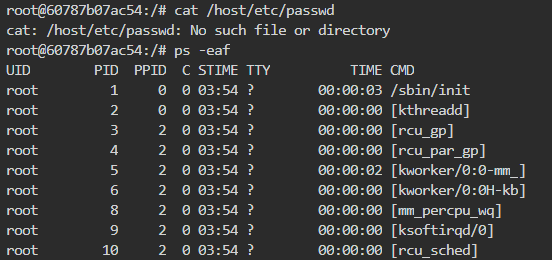
4.List available Docker images.



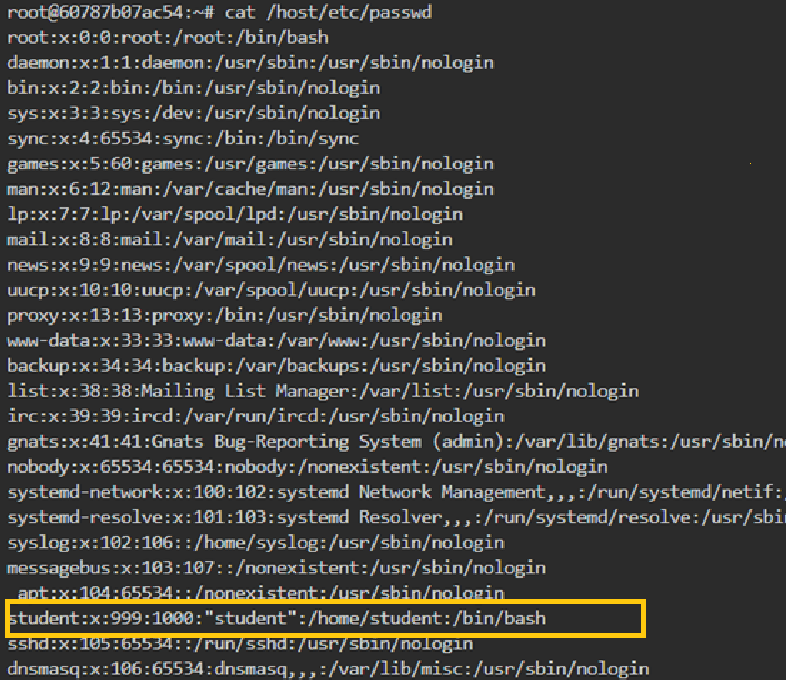
5.Launch Ubuntu container with host filesystem mount.



6.Gain root access using chroot.



7.Access host filesystem within the container.

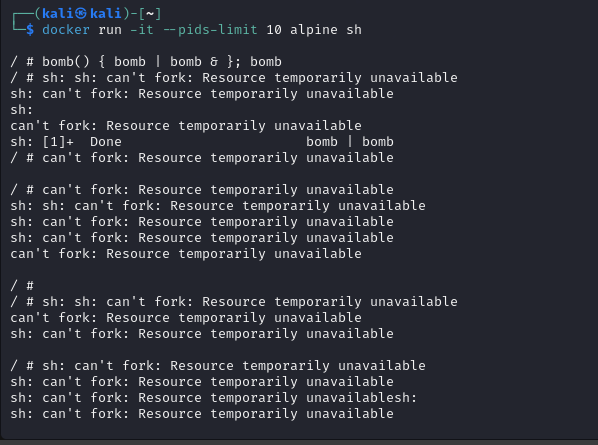


**Part B – Implementation of Security Application**

1. **Cgroups**

Control groups (Cgroups) play an important role in optimizing resource utilization, thereby ensuring scalability, performance, and reliability in containerized environments. They primarily regulate resource consumption, such as memory, CPU, RAM, and network bandwidth, preventing any single container from consuming all the available resources. Additionally, Cgroups provide the capability to prioritize critical tasks within containers, ensuring that essential processes receive the necessary resources over less critical ones. This prioritization mechanism enhances the responsiveness and efficiency of key functionalities, even under resource constraints or heavy workloads, contributing to the overall stability and effectiveness of containerized applications.

1. Setting limit on number of processes that can be created inside the container

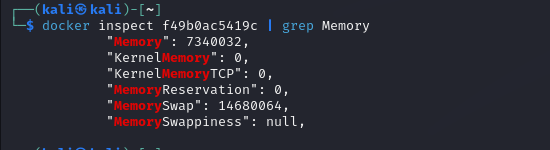


Fork bomb is a one of the denial of service attacks, when fork bomb command is executed it creates unlimited processes and utilizes all the resources making a system unusable by any other users. By setting limits to the processes that can be created inside the container, forkbomb can be mitigated.

Setting limit on CPU and memory usage

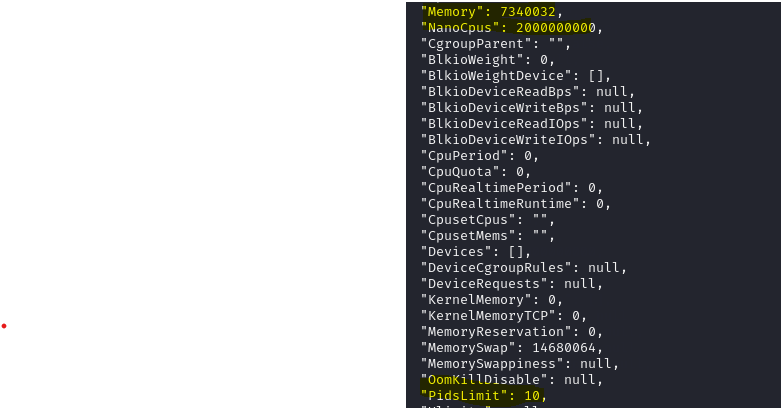


Based on the given condition on memory, memory limit is set on container

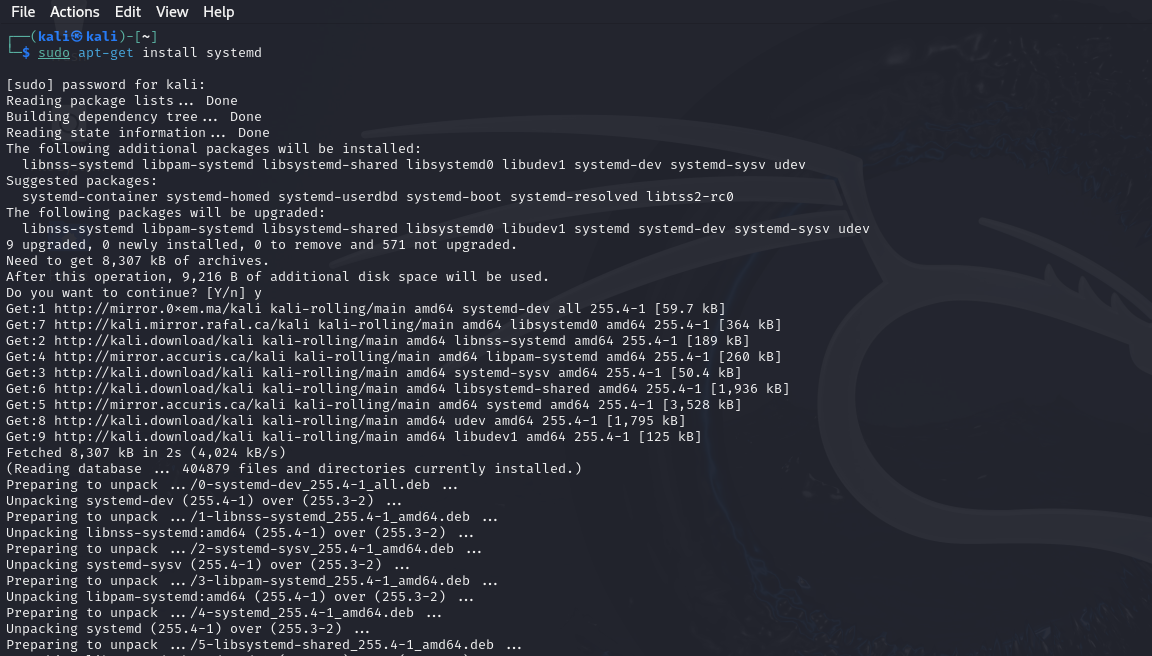


And docker inspect <container id> command gives information on various resource usage





Moreover,Cgroups also gives the logs of resource usage within the container.

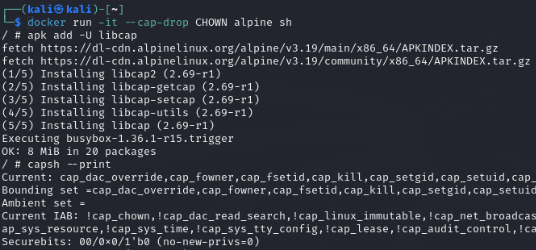




**2. Capabilities**

Capabilities are one of the fundamental aspects of Linux security. These are the privileges granted to the processes in the containers. Using capabilities fine grained control over processes can be achieved. Instead of allowing the process to run with root privileges, as capabilities align with the least privilege principle, which is providing only the necessary privileges to perform a task can help to minimize the security breaches.

1. Dropping CHOWN privilege



1. Operation is not permitted as CHOWN privilege is dropped



Overall, Capabilities allow administrator to provide only necessary privileges.

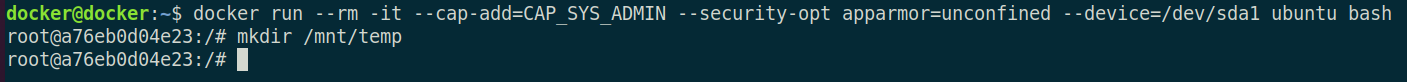
**2. Apparmor - Attack and Defence against Privilege Escalation using Disk Mount**

These are another security component within the Linux kernel, granting administrators the ability to control user access to various programs and files within Docker Containers.​

Through Apparmor profiles, permissions for reading, writing or executing files on specific paths can be specified.

**Attack Execution :**

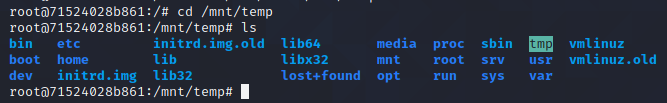
1. Mounting Host’s Disk inside the Docker containers, gives the containers access to the Host’s File System, thereby causing a Privilege Escalation Attack. But, by default, mounting operation is restricted for docker containers due to apparmor profile.  
   Hence, to give mounting permissions, we will start the docker container by disabling the apparmor profile.



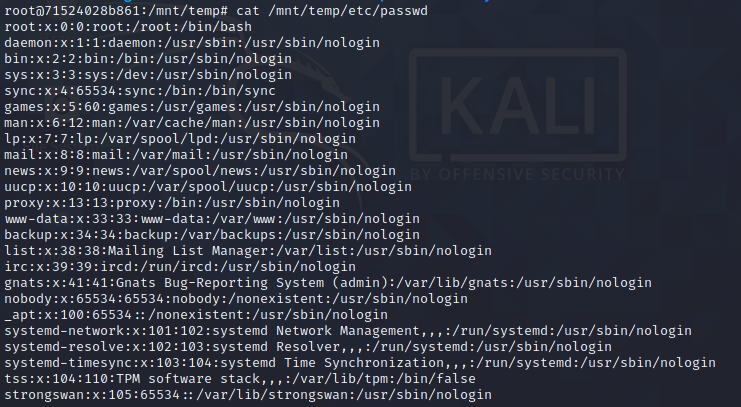
1. Now we can mount the host disk within the container.  
   

we can see that the disk mount is successful.

1. Thus the attack is executed and now we can view all the contents of the host inside the container from /mnt/temp directory.

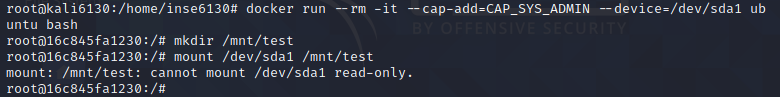


1. Hence, we can see all root files and passwords like /etc/passwd file, which shouldnt be accessible from within the container :



**Mitigation :**

We can mitigate this attack , by starting the container, without excluding the apparmor profile, or by properly configuring the apparmor profile and giving rights only that are necessary.



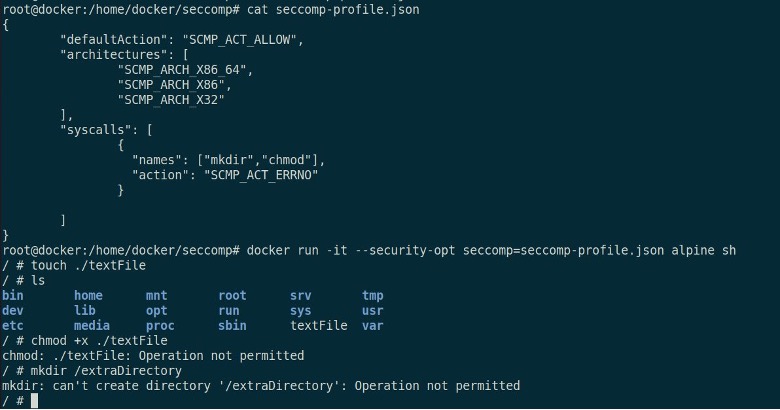
Thus, the attack is mitigated by using the default apparmor profile, which restricted the mounting operation from the container.

**3. Seccomp Profile**

Seccomp profiles in Docker are used to mitigate various types of attacks that exploit system calls to perform malicious activities. These can include privilege escalation, data breaches, container escapes, and denial-of-service attacks. By restricting the set of system calls that a container can execute, seccomp reduces the surface area an attacker can use to compromise the Docker host or other containers.

**1. Type of Attack Mitigation in Docker:**

Seccomp profiles in Docker are designed to mitigate attacks that exploit system calls to compromise containers and the host system. By filtering system calls, seccomp can prevent:

* Privilege Escalation Attacks: Where an attacker uses a system call to gain higher privileges than those granted.
* Container Breakouts: Where an attacker escapes the container to gain access to the Docker host or other containers.
* Kernel Exploits: Where less-secure or rarely used system calls might have vulnerabilities that could be used to compromise the kernel.
* Resource Exhaustion: Where an attacker uses system calls in a way that can deplete system resources, leading to a denial of service.

**2. Purpose of Seccomp in Docker:**

The purpose of seccomp in Docker is to limit the attack surface by allowing only necessary system calls required by a container to function and blocking all others. This limits the capabilities of any process within the container, making it harder for attackers to exploit the system. It's a form of application sandboxing that increases container isolation and limits the potential damage from a compromised container.

**3. Working of Seccomp and Preventive Features:**

Seccomp filters system calls at the kernel level. Here is how it works when you start a Docker container with a seccomp profile:

1. **Profile Configuration:** Docker configures the container’s processes to use the specified profile.
2. **System Call Execution:** When a process within the container attempts to execute a system call, the seccomp filter checks against the profile.
3. **Decision Making:** If the profile allows the system call, it proceeds; if not, the action specified in the profile (usually to deny the call) is taken.
4. **Security Enhancement:** The profile you provided denies mkdir and chmod (as seen in the first screenshot) and socket and connect (as seen in the second screenshot), effectively preventing file manipulation and network connections that are not explicitly permitted. This helps in preventing file system tampering and unauthorized network access.

* **Execution in the Container:**

The steps to execute seccomp within a Docker container as illustrated by the provided screenshots are:

1. **Profile Definition:** Define a seccomp profile (seccomp-profile.json) with specific rules to allow or deny system calls.
2. **Container Initialization:** Run a Docker container with the security option to enforce the seccomp profile using the command: **docker run --security-opt seccomp=seccomp-profile.json**
3. **Execution Monitoring:** Attempts to execute blocked system calls within the container result in errors, as observed when trying **mkdir** or **chmod**.

* **Practical implementation**To execute seccomp within a Docker container in practice:

1. Create a seccomp profile in JSON format with the desired rules.
2. Start the Docker container with the **--security-opt seccomp=/path/to/seccomp/profile.json** option.
3. Inside the container, try to execute the allowed and blocked system calls to verify the profile is working correctly.

* **Next Steps**

1. **Review Seccomp Profiles:** Ensure the seccomp profiles cover all the system calls that need to be restricted for your application.
2. **Run Containers with Custom Profiles:** Run your Docker containers with the --security-opt flag pointing to your custom seccomp profiles.
3. **Monitor and Adjust:** Monitor your applications for any legitimate system calls that may be incorrectly blocked and adjust your profiles accordingly.

**Challenges**

As we concluded our work on the Docker security project, we encountered and successfully addressed several minor challenges. Our focus was on Docker attacks and security applications, where we implemented effective countermeasures. One significant hurdle we faced involved replicating specific attacks and vulnerabilities, requiring extensive troubleshooting efforts. Additionally, obtaining the appropriate vulnerable versions of Docker images and VMs for attack replication proved to be a challenge due to a lack of comprehensive documentation. Despite these obstacles, we remained dedicated to delivering the project appropriately, maintaining our commitment to robust security practices and thorough project execution.