CVE-2019-18683: Linux Local Privilege Escalation Vulnerability

IT19142906

H.H.S.Y.Wijeveera

Offensive Security Conference
February 14-15 2020 // Berlin
CVE-2019-18683: Local Privilege Escalation Demo

Introduction

What is a vulnerability?

In cyber security, a vulnerability is a weak point that can be exploited using a cyber-attack to take advantage of unauthorized access or performing unauthorized actions on a computer system. Vulnerabilities could allow attackers to run code, gain access to system memory, install malware and leverage, destroy or modify confidential facts. To exploit the vulnerability, an attacker must be able to connect to a computer system. Vulnerabilities can be exploited through a series of strategies, including SQL injection, buffer overflows, Web page move (XSS) scripts, and open source exploit kits that look for recognized vulnerabilities and weaknesses in Internet packages.

Linux vulnerabilities

If you have been working with a computer for the remaining decade, you understand the importance of keeping your program up to date. Those who do not do this build up vulnerabilities, waiting for others to use them. Although Linux and the most open source software can be considered, software shortcomings in the programs remain. Despite the fact that it is not easy to eliminate every vulnerability for your system, we can at least create a reliable process around it.

Common Linux Vulnerabilities

Linux has drawbacks similar to other operating systems. These flaws are inherent in the operation of computers. Most of them are called up throughout the software development cycle. Weakness is usually somewhere in the logic involved. One missing "if" statement may be enough to make a piece of software vulnerable to a routine attack right away. The big difference is that each working device has different approaches to deal with them. This starts with the assembly flags used during compilation of the delivery code, until the software runs.

Programming flaws

Most security updates delivered through Linux distributions fix one or additional software defects. From buffer overflows to misuse of sources, they all have a special chance for security. In all cases, they contain vulnerability. Million-dollar questions - who can abuse them and their consequences. Some vulnerabilities can only be caused by local clients, in which another is probably part of a commonly used web server.

CVE-2019-18683: Linux Local Privilege Escalation Vulnerability

- This describes the exploitation of CVE-2019-18683, which refers to a pair of five-year racing situations in the Linux kernel V4L2 subsystem. It was discovered and fixed at the end of 2019. Here I am going to describe the use of PoC for x86_64, which benefits from privilege escalation from the context of the kernel thread (where user space is not displayed), bypassing KASLR, SMEP, and SMAP on Ubuntu Server 18.04.
- An issue was discovered in drivers/media/platform/vivid in the Linux kernel through 5.3.8. It is exploitable for privilege escalation on some Linux distributions where local users have /dev/video0 access, but only if the driver happens to be loaded. There are multiple race conditions during streaming stopping in this driver (part of the V4L2 subsystem).
- These issues are caused by wrong mutex locking in vivid_stop_generating_vid_cap(),vivid_stop_generating_vid_out(), sdr_cap_stop_streaming(), and the corresponding kthreads. At least one of these race conditions leads to a use-after-free.

- Alexander Popov is the one who found this vulnerability.
- Linux kernel developer Security researcher at Positive Technologies.

About V4L2

V4L2 stands for Video for Linux version 2. It is a collection of drivers and an API for supporting video capture. The vulnerable driver at drivers/media/platform/vivid . The vulnerable driver emulates hardware of various types for V4L2: ex: - video capture and output, radio receivers and transmitters, software-defined radio receivers, etc. The vulnerable driver is used as a test input for application development without requiring special hardware. On Ubuntu the vivid devices are available to the normal user. Ubuntu applies RW ACL when the user is logged in.

Time periods

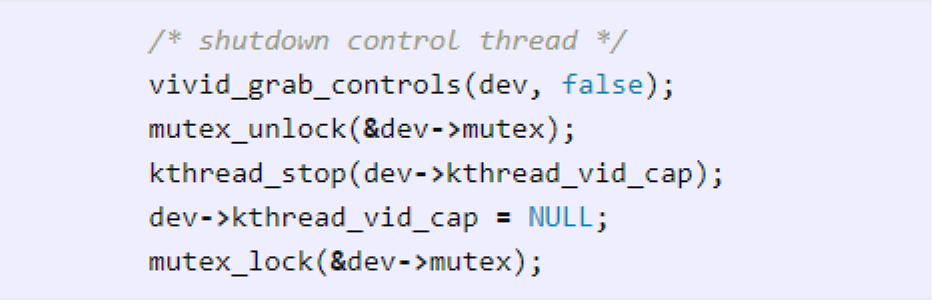
In 25th of August 2014, Bugs have been introduced. In 5th of September 2019 , his custom syzkaller gets a crash. In September 13, 2019, He starts the investigation. In November 1, 2019 his PoC exploit and fixing patch are ready. He sends the crasher and patch to security@kernel.org and review starts. In November 2, 2019 he prepares v2 and v3 of the patch. Linus Torvalds allows to do full disclosure. In November 4, 2019 Linus finds a mistake in v3 of the patch. Alex sends v4 to the LKML. CVE-2019-18683 is allocated. In November 8, 2019 the fixing patch is merged to the mainline. November 27, 2019 the fixing patch is taken to the stable trees.

Bugs

He has used the syzkaller fuzzer with custom adjustments to the kernel supply code and was given a suspicious kernel crash. KASAN detected use-after-unfastened at some point of related list manipulations in vid_cap_buf_queue(). Investigation of the reasons led him pretty a long way from the reminiscence corruption. Ultimately, he observed that the same wrong technique to locking is used in vivid_stop_generating_vid_cap(), vivid_stop_generating_vid_out(), and sdr_cap_stop_streaming(). This ended in 3 similar vulnerabilities.

These capabilities are called with vivid_dev.Mutex locked whilst streaming is being stopped. The features all make the equal mistake whilst preventing their kthreads that want to lock this mutex as nicely.

Here is the example from vivid_stop_generating_vid_cap():



```
/* shutdown control thread */
vivid_grab_controls(dev, false);
mutex_unlock(&dev->mutex);
kthread_stop(dev->kthread_vid_cap);
dev->kthread_vid_cap = NULL;
mutex_lock(&dev->mutex);
```

To fix these issues, he has done the following.

1. Avoided unlocking the mutex on streaming stop. For example, see the diff for `vivid_stop_generating_vid_cap()`:

```
    /* shutdown control thread */
    vivid_grab_controls(dev, false);
-   mutex_unlock(&dev->mutex);
    kthread_stop(dev->kthread_vid_cap);
    dev->kthread_vid_cap = NULL;
-   mutex_lock(&dev->mutex);
```

2. Used `mutex_trylock()` with `schedule_timeout_uninterruptible()` in the loops of the vivid kthread handlers.

   The `vivid_thread_vid_cap()` handler was changed as follows:

```
    for (;;) {
            try_to_freeze();
            if (kthread_should_stop())
                    break;
-           mutex_lock(&dev->mutex);
+           if (!mutex_trylock(&dev->mutex)) {
+                   schedule_timeout_uninterruptible(1);
+                   continue;
+           }
            ...
    }
```

EXPLOITATION, STEP BY STEP

Step 1: - Runs this in several pthreads:

```
#define err_exit(msg) do { perror(msg); exit(EXIT_FAILURE); } while (0)

 for (loop = 0; loop < LOOP_N; loop++) {

 int fd = 0;

 fd = open("/dev/video0", O_RDWR);

if (fd < 0)

err_exit("[-] open /dev/video0");

read(fd, buf, 0xfffded);

 close(fd);

}
```

Deceiving V4L2 subsystem.

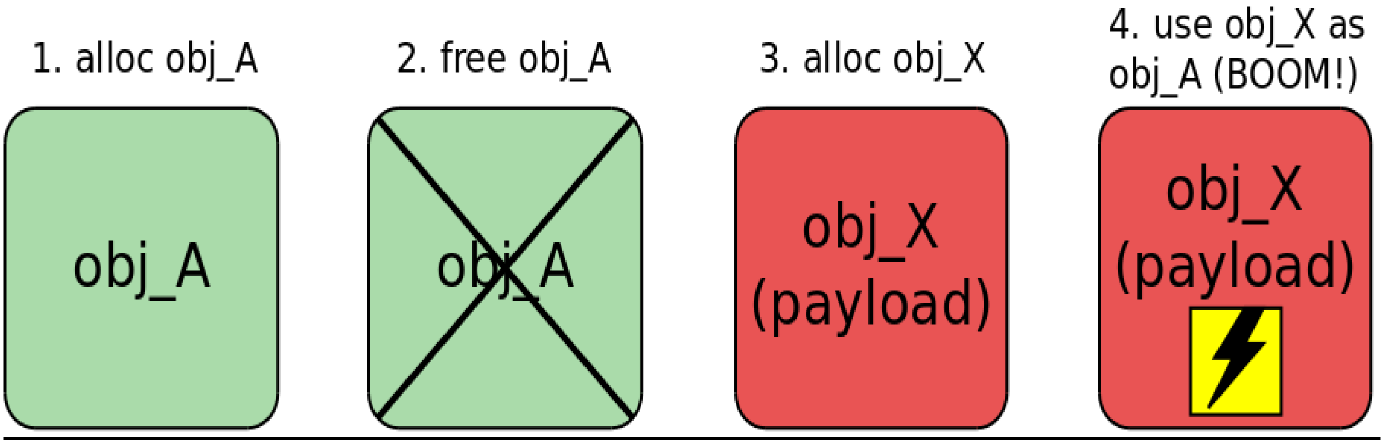
Meanwhile, streaming has completely stopped. The closing connection with /dev/video0 is released and the V4L2 subsystem calls vb2_core_queue_release(), that is answerable for freeing up sources. It in turn calls __vb2_queue_free(), which frees our vb2_buffer that changed into introduced to the queue when the exploit gained the race.

But the driver isn't privy to this and still holds the connection with the freed object. When streaming is began again on the next make the most loop, brilliant motive force touches the freed object this is caught with the aid of KASAN:

```
BUG: KASAN: use-after-free in vid_cap_buf_queue+0x188/0x1c0 Write of size 8 at
addr ffff8880798223a0 by task v4l2-crasher/300 ... The buggy address belongs to
the object at ffff888079822000 which belongs to the cache kmalloc-1k of size
1024
```

Step 2 : - Overwriting vb2_buffer

First idea: apply setxattr()+userfaultfd() technique (Vitaly Nikolenko) to exploit use-after-free



Vulnerable vb2_buffer is not the last one freed by __vb2_queue_free(). Next kmalloc() doesn't return the needed pointer. So having only one allocation is not enough for overwriting . Spraying with Vitaly's technique is not easy:

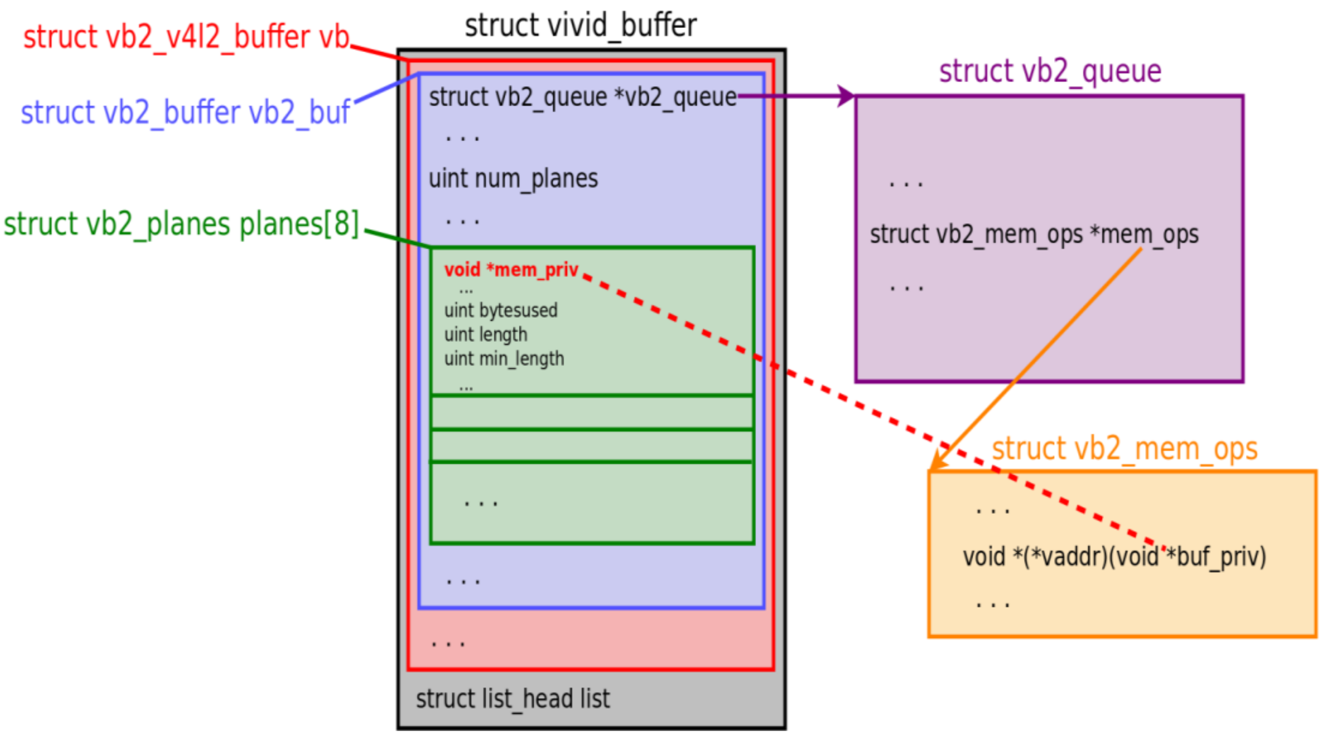
Process calling setxattr() hangs until the userfaultfd() page fault handler calls UFFDIO_COPY ioctl

Overwriting vb2_buffer: Brute-Force Solution

Alex popv had created a spraying pthreads pool(dozens of them). Each pthread calls setxattr () based on userfaultfd () and hangs. Pthreads is allocated between processors using sched_setaffinity (). Thus, the spray covers all the slab caches (they may be in accordance with the CPU). After his heap spray is successful, vb2_buffer is overwritten. This vb2_buffer is processed by V4L2 after the start of the next streaming.

Step 3 :- Control Flow Hijack for V4L2 Subsystem

V4L2 is a very complicated Linux kernel subsystem. The following diagram (not to scale) describes the relationships among the objects which are part of the subsystem:



Unexpected Troubles: Kthread

After discovering vb2_mem_ops.Vaddr, began to analyze the minimum payload that I wanted to get in order to get the V4L2 code to achieve this characteristic pointer. First of all, I turned off SMAP (Supervisor Mode Access Prevention), SMEP (Supervisor Mode Execution Prevention), and KPTI (kernel page table isolation). Then I pointed vb2_buffer.Vb2_queue to the mmap memory area in user space. The dereferencing of this pointer turned into a gross error: "it is impossible to handle the error of the web page." It turned out that the pointer is dereferenced in the context of the kernel thread, where my user space is not displayed in any way.

We can get the kernel stack area using V4L2 parsing. And then count on the fate of the transaction using the payload! This was the highest quality second of research. The type of moment that makes everything else well worth it :).

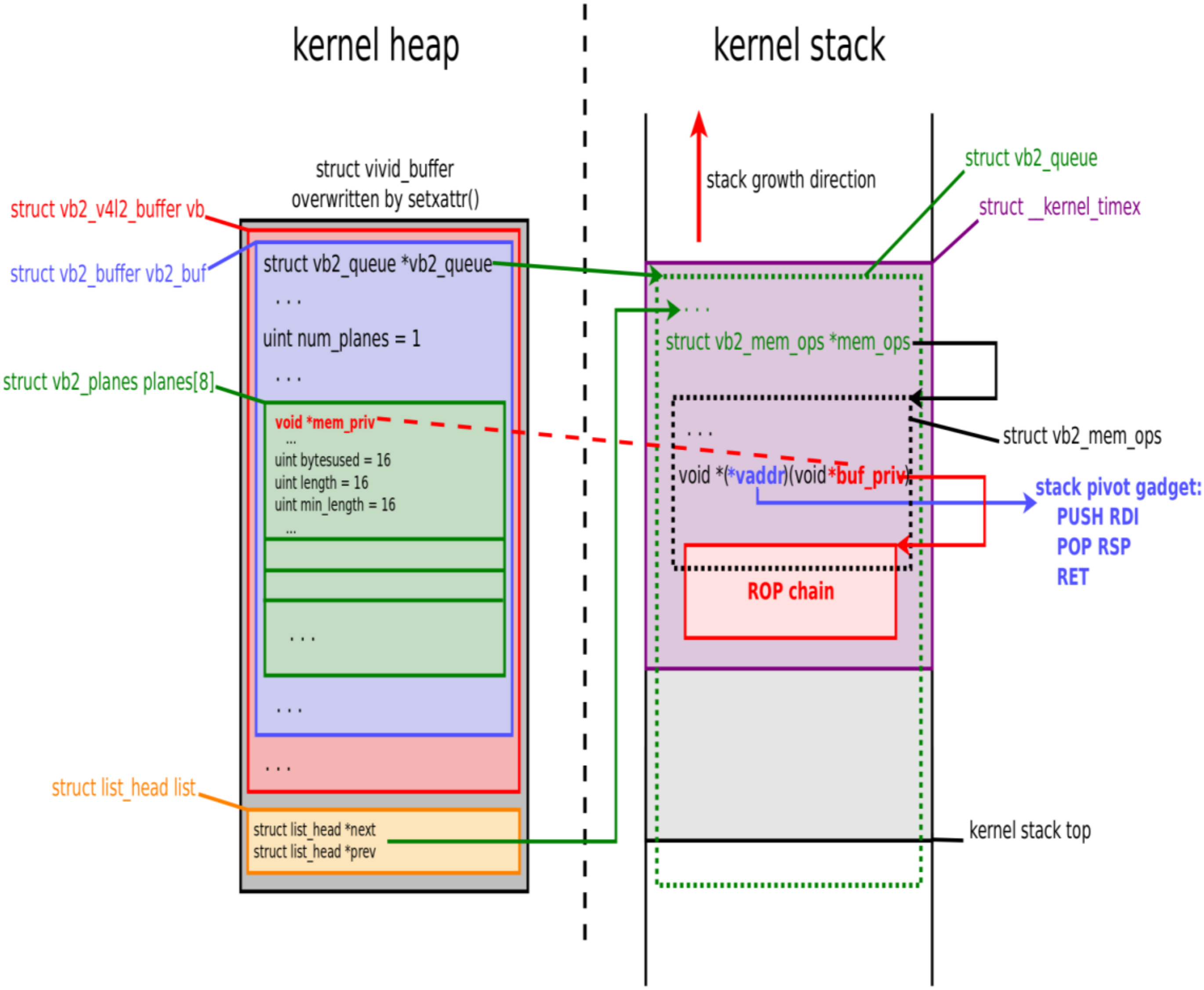
So creating an exploit orchestra, we can capture the float.

Anatomy of the Exploit Payload

The exploit payload is created in two locations and they are in kernel heap by sprayer pthreads using setxattr() syscall, in kernel stack by racer pthreads using adjtimex() syscall , both powered by userfaultfd()

The exploit payload consists of three parts and they are vb2_buffer in kernel heap , vb2_queue in kernel stack , vb2_mem_ops in kernel stack.

Anatomy of the Exploit Payload: A Diagram



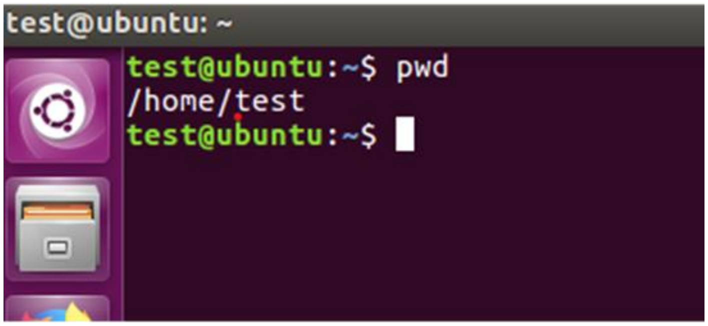
Privilege Escalation

run_cmd() executes “/bin/sh /home/a13x/pwn” with root privileges. That script rewrites /etc/passwd to log in as root without password:

```
#!/bin/sh
# drop root password
sed -i ’1s/.*/root::0:0:root:\/root:\/bin\/bash/’ /etc/passwd
```

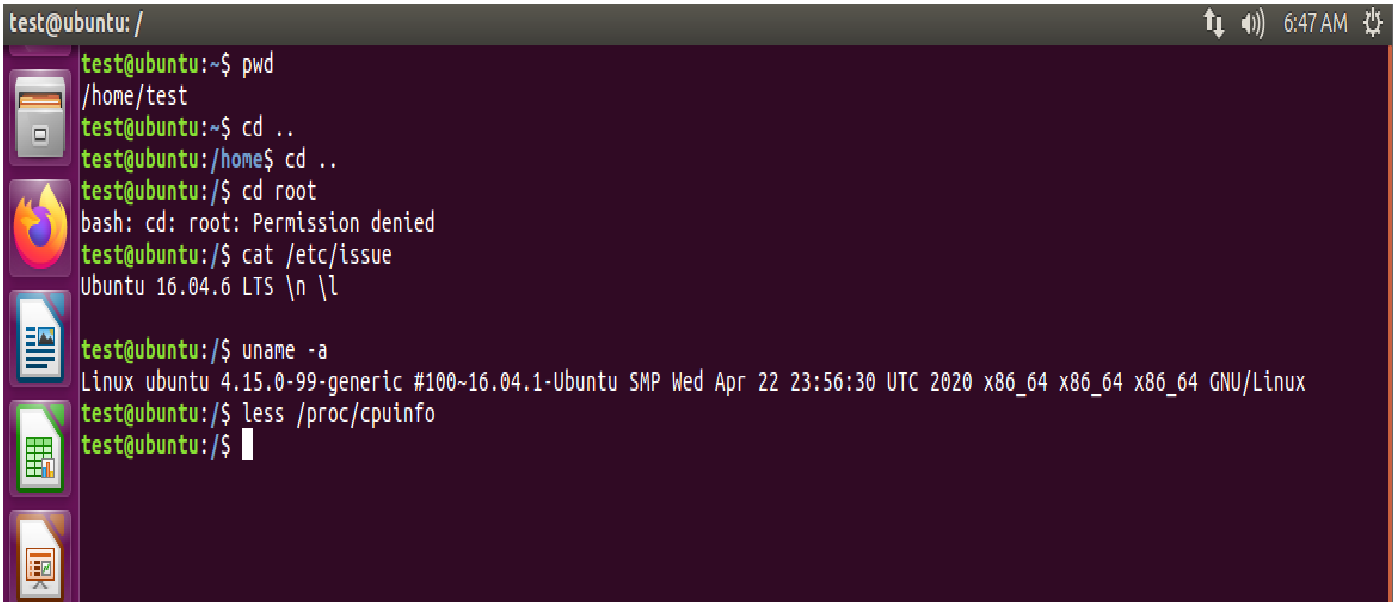
Possible Exploit Mitigation

- Against userfaultfd() abuse – set /proc/sys/vm/unprivileged_userfaultfd to 0
- Against infoleak via kernel log – set kernel.dmesg_restrict sysctl to 1 N.B. Ubuntu users from adm group can read /var/log/syslog anyway
- Against anticipating stack payload location – PAX_RANDKSTACK from grsecurity/PaX patch
- Against my ROP/JOP chain – PAX_RAP from grsecurity/PaX patch
- Against use-after-free (hopefully in future) – ARM Memory Tagging Extension (MTE) support for kernel



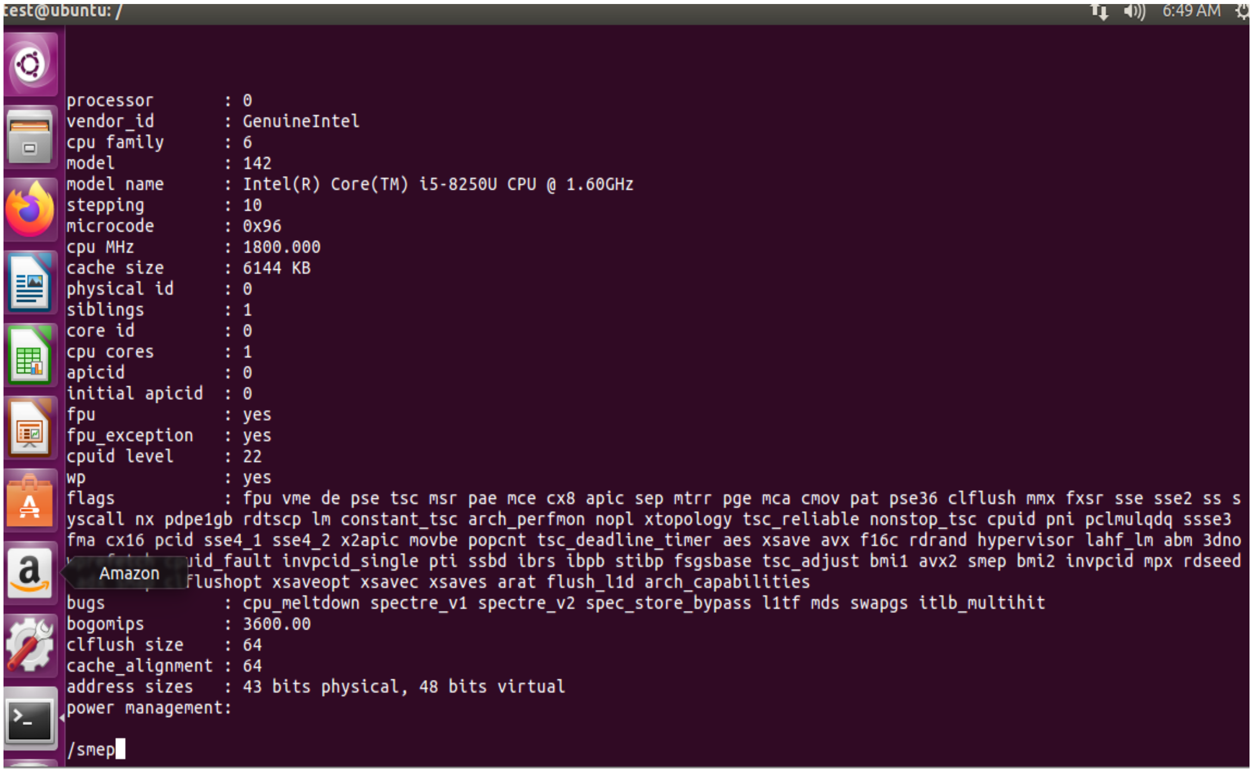
```
test@ubuntu:~$ pwd
/home/test
test@ubuntu:~$ █
```

Here are the screen shots of the exploitation code in Ubuntu Linux



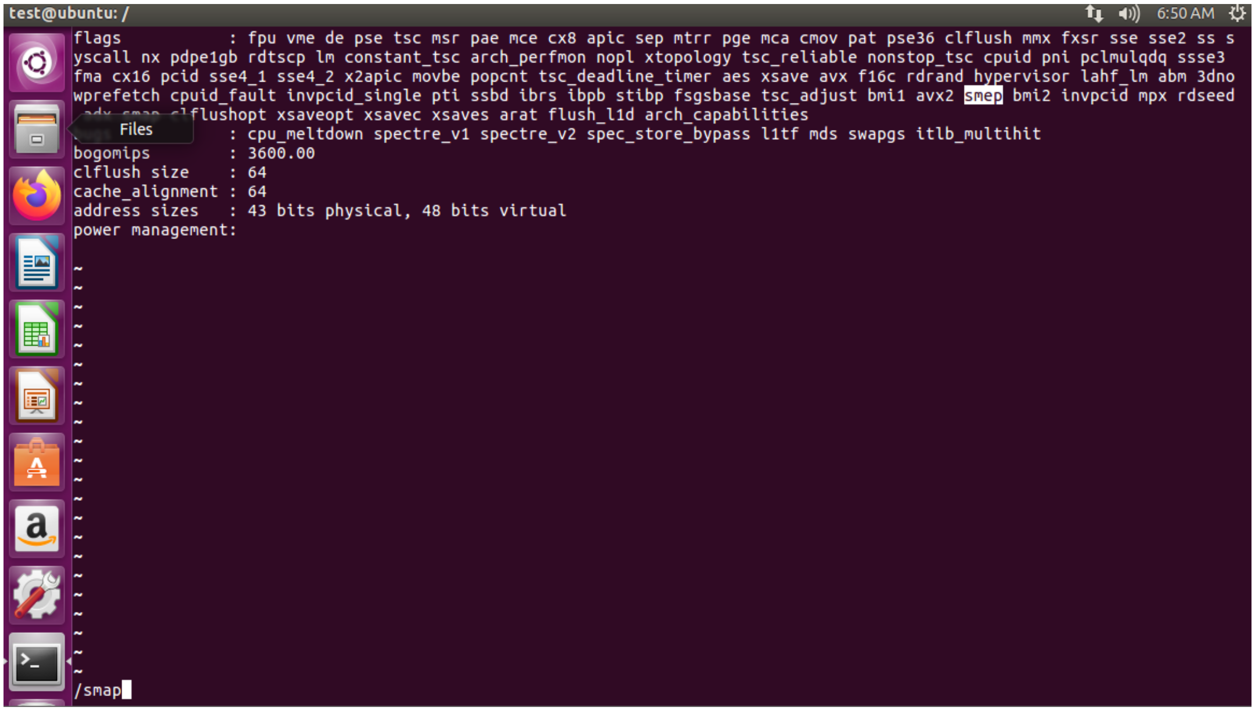
```
test@ubuntu:~$ pwd
/home/test
test@ubuntu:~$ cd ..
test@ubuntu:/home$ cd ..
test@ubuntu:/$ cd root
bash: cd: root: Permission denied
test@ubuntu:/$ cat /etc/issue
Ubuntu 16.04.6 LTS \n \l

test@ubuntu:/$ uname -a
Linux ubuntu 4.15.0-99-generic #100~16.04.1-Ubuntu SMP Wed Apr 22 23:56:30 UTC 2020 x86_64 x86_64 x86_64 GNU/Linux
test@ubuntu:/$ less /proc/cpuinfo
test@ubuntu:/$ 
```



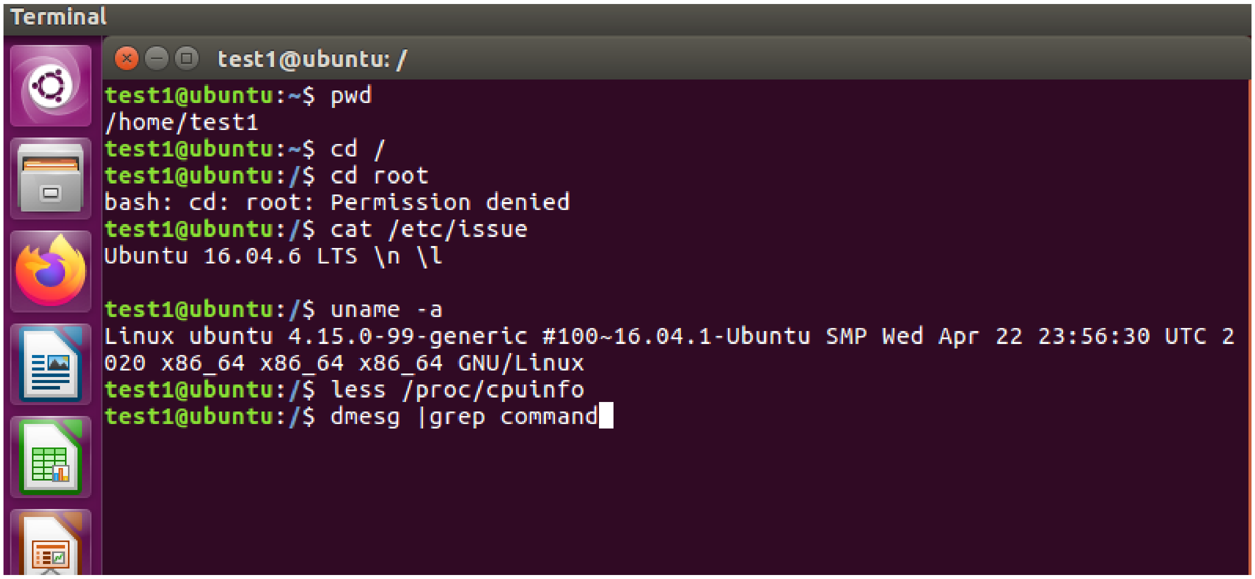
```
processor       : 0
vendor_id       : GenuineIntel
cpu family      : 6
model           : 142
model name      : Intel(R) Core(TM) i5-8250U CPU @ 1.60GHz
stepping        : 10
microcode       : 0x96
cpu MHz         : 1800.000
cache size      : 6144 KB
physical id     : 0
siblings        : 1
core id         : 0
cpu cores       : 1
apicid          : 0
initial apicid  : 0
fpu             : yes
fpu_exception   : yes
cpuid level     : 22
wp              : yes
flags           : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36 clflush mmx fxsr sse sse2 ss s
yscall nx pdpe1gb rdtscp lm constant_tsc arch_perfmon nopl xtopology tsc_reliable nonstop_tsc cpuid pni pclmulqdq ssse3
fma cx16 pcid sse4_1 sse4_2 x2apic movbe popcnt tsc_deadline_timer aes xsave avx f16c rdrand hypervisor lahf_lm abm 3dno
            uid_fault invpcid_single pti ssbd ibrs ibpb stibp fsgsbase tsc_adjust bmi1 avx2 smep bmi2 invpcid mpx rdseed
            flushopt xsaveopt xsavec xsaves arat flush_l1d arch_capabilities
bugs            : cpu_meltdown spectre_v1 spectre_v2 spec_store_bypass l1tf mds swapgs itlb_multihit
bogomips        : 3600.00
clflush size    : 64
cache_alignment : 64
address sizes   : 43 bits physical, 48 bits virtual
power management:

/smep
```



```
flags           : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36 clflush mmx fxsr sse sse2 ss s
yscall nx pdpe1gb rdtscp lm constant_tsc arch_perfmon nopl xtopology tsc_reliable nonstop_tsc cpuid pni pclmulqdq ssse3
fma cx16 pcid sse4_1 sse4_2 x2apic movbe popcnt tsc_deadline_timer aes xsave avx f16c rdrand hypervisor lahf_lm abm 3dno
wprefetch cpuid_fault invpcid_single pti ssbd ibrs ibpb stibp fsgsbase tsc_adjust bmi1 avx2 smep bmi2 invpcid mpx rdseed
 adx smap clflushopt xsaveopt xsavec xsaves arat flush_l1d arch_capabilities
bugs            : cpu_meltdown spectre_v1 spectre_v2 spec_store_bypass l1tf mds swapgs itlb_multihit
bogomips        : 3600.00
clflush size    : 64
cache_alignment : 64
address sizes   : 43 bits physical, 48 bits virtual
power management:

~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
/smap
```



```
test1@ubuntu:~$ pwd
/home/test1
test1@ubuntu:~$ cd /
test1@ubuntu:/$ cd root
bash: cd: root: Permission denied
test1@ubuntu:/$ cat /etc/issue
Ubuntu 16.04.6 LTS \n \l

test1@ubuntu:/$ uname -a
Linux ubuntu 4.15.0-99-generic #100~16.04.1-Ubuntu SMP Wed Apr 22 23:56:30 UTC 2
020 x86_64 x86_64 x86_64 GNU/Linux
test1@ubuntu:/$ less /proc/cpuinfo
test1@ubuntu:/$ dmesg |grep command
```

References

- Popov, A., 2020. *CVE-2019-18683: Exploiting A Linux Kernel Vulnerability In The V4L2 Subsystem*. [online] Alexander Popov. Available at: <https://a13xp0p0v.github.io/2020/02/15/CVE-2019-18683.html> [Accessed 9 May 2020].
- Upguard.com. 2020. *What Is A Vulnerability?*. [online] Available at: <https://www.upguard.com/blog/vulnerability> [Accessed 9 May 2020].
- Boelen, M., 2020. *Linux Vulnerabilities: From Detection To Treatment*. [online] Linux Audit. Available at: <https://linux-audit.com/linux-vulnerabilities-explained-from-detection-to-treatment/> [Accessed 9 May 2020].
- 2020. [online] Available at: <https://www.linkedin.com/in/alexander-popov-a1398b69> [Accessed 9 May 2020].
- A13xp0p0v.github.io. 2020. [online] Available at: <https://a13xp0p0v.github.io/img/CVE-2019-18683.pdf> [Accessed 11 May 2020].