LINGI2144: Secured System Engineering
Tutorial 6: Format String

Academic year : 2020 - 2021

Teacher : Legay Axel
Course : LINGI2144
Collaborators :
Crochet Christophe
Duchene Fabien
Given-Wilson Thomas
Strebelle Sebastien

1 Prerequisite

Foreword This is going to be the first tutorial done online, and also comes after the last two tutorials which not all students were able to finish. This tutorial is short, but may still take some time.

1.1 Environment Configuration

On most Linux systems and with most compilers there are protections built in to prevent various exploits. For today's tutorial we may have to turn some of these off.

One is the randomisation of memory segments by the Linux kernel. We can see the current value with

```
sudo cat /proc/sys/kernel/randomize_va_space
```

This is "2" by default on Kali Linux (and most Linux systems). To turn this off for the rest of the sessions by setting the value to "0" we can run

```
echo 0 | sudo tee /proc/sys/kernel/randomize_va_space
sudo cat /proc/sys/kernel/randomize_va_space
```

The compiler can also insert various protections into code that is compiled. For the stack, `gcc` includes some stack protection by default on many versions. We can force this to be turned off with the argument

```
-fno-stack-protector
```

At times we may also wish to force gcc to compile code with executable instructions on the stack, we can enable this with

```
-z execstack
```

2 Exercise

2.1 Evading Stack Protection - Pointer Rewritting Attack

In "`vuln.c`" is a program that has a vulnerability even when compiled with stack protection. This file can be built with:

```
gcc -z execstack -o vuln vuln.c
```

NOTE: we do compile with stack protection! The goal of this part of the tutorial is for you to exploit code with stack protection enabled!

We can run this program with:

```
./vuln AAAA BBBB
```

and see that we are shown some memory addresses (these are included to help you).

By examining the code we can see that there is an instance of `strcpy` that we can probably exploit.

The goal is to provide command line arguments that allow you to execute a shellcode (of your choice, but a simple `/bin/sh` one is fine including taking one from an earlier tutorial).

HINT: Use the first `strcpy` to overflow and change the value of where "p" points to.

2.2 Breaking ASLR

Make to sure to enable ASLR. In "`ret2text.c`" you will find a program that that should be exploitable even with stack protection. You can build the program with (include debugging if you want):

```
gcc -o ret2text ret2text.c
```

You should now be able to cause an overflow that allows you to jump into the secret function.

- HINT: This was used as an example in the lecture with brief discussion of how to perform the exploit.

In "`vulnerable.c`" you will find a program that we injected shell code into when ASLR was disabled. Can you inject a shellcode into it with ASLR enable?

- HINT: Some solutions were presented in the lectures, or feel free to instrument the code and learn about the behaviour on your system before exploiting it.