LINGI2144: Secured System Engineering
Tutorial 7: Evading Stack Protection & Format String

Academic year : 2020 - 2021

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

1 Prerequisite

Working directory: `~/SecurityClass/Tutorial-07`

Connection:

username	**password**
admin	nimda

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 Format String

Here you will find a program "`example.c`" that has a format string vulnerability in the `printf` statement that you can use.

This is a program designed to help you explore how to abuse format string and pass in parameters. You can build the program with:

```
gcc -o example example.c
```

and then run it with format string arguments, for example:

```
./example "%x %x %x %x"
```

Observe that there is a value called "`test_value`" that you **may** be able to modify.

NOTE: Due to memory layout and your architecture it MAY NOT be easy or possible to inject a format string (e.g. your address may require a `\x00` which would break the formatting). For the rest of this section, you can also use `format1.c` or `format2.c` if these have better memory layout for you (e.g. you want to change values on the stack).

The goal now is to use a format string to change the value of either "`test_value`" in `example.c`, or "`target`" in `format1.c` or `format2.c`.

HINT: The format string component `%n` writes back from format string into an address. You can use this to change the value of one of the "arguments" to the function using your format string.