Operating Systems Security – Assignment 3

2017/2018
Due Date: 7 Dec 2017 (23:59 CET)

1 Bypassing ASLR

In this exercise, you will learn how to exploit a buffer overflow, bypassing ASLR on 64-bit Linux binaries. You are strongly encouraged to read this article[1] before proceeding with the assignment.

Login your (Kali) Linux system and download the source code from https://www.cs.ru.nl/~vmoonsamy/teaching/ossec2016/a3_sourcecode.zip.

Compile `vuln.c` using the following command (**Hint:** add `-no-pie` on Ubuntu 16.10 and above to avoid the executable becoming ASLR'ed):

```
# gcc -fno-stack-protector -no-pie vuln.c -o vuln
```

Enable ASLR:

```
# echo 2 > /proc/sys/kernel/randomize_va_space
```

Objectives

a) Find the address of **`read@plt`** and **`system@plt`** by disassembling `vuln` using **`objdump`**.
b) In `vuln.c`, a ROP gadget is provided in the function helper.
   i) What does the gadget do and what can it be used for?
   ii) Find the offset of the gadget in the executable.
c) Use **`readelf`** to find writable memory that can be used to store 8 bytes of payload. Give the memory address and explain why you picked this address.
d) Locate `payload.py` in the source code folder you downloaded. The exploit is split up into two stages - Stage 1 and Stage 2. Explain for each stage what is being done.

   Run the vulnerable program as a server:
   `# socat TCP-LISTEN:3333,reuseaddr,fork EXEC:./vuln`
   This way you can send input using a socket on port 3333.

e) Fill in the memory addresses you found before in `exploit.py` and run it in a separate terminal. Confirm that you can successfully get a shell by providing a screenshot (this should be the case if you entered the correct addresses) and explain in detail why.

2 Self-replicating code

Write a (small) C program that prints its own source-code.

[1] https://blog.techorganic.com/2016/03/18/64-bit-linux-stack-smashing-tutorial-part-3/

3 Covert channels

An operating system tries to avoid information leakage between processes which are executed by different users. However, it is not always capable of identifying suspicious behaviour, especially if the processes use generic information leakage channels such as:

- Existence of a file
- File attributes
- CPU usage
- Temperature sensor
- "Disk full" errors

Objectives

a) Write two (simple) programs that communicate messages to each other using a covert information leakage channel that is not (inherently) identified by the operating system as communication channel between processes. Hand in the source-code and explain how it should be installed/used.

b) Execute the programs under two different (**non**-root) users and let it communicate to one another. Then, open the log files of your (Kali) Linux system and see if the communication leaves any visible trace (such as unusual and suspicious errors). Explain why your method is undetectable or how it could be optimized to avoid detection by operating systems that perform more advanced monitoring.