

# Operating Systems Security – Assignment 3

Version 1.0 – 2016/2017

Due Date: 02 Dec 2016 (23:59 CET)

Institute for Computing and Information Sciences,  
Radboud University, The Netherlands.

## 1 Bypassing ASLR

In this exercise, you will learn how to exploit buffer overflow to bypass ASLR on 64-bit Linux binaries. You are strongly encourage to read this article<sup>1</sup> before proceeding with the assignment.

Login to your (Kali) Linux system and download the source code from [https://www.cs.ru.nl/~vmoonsamy/teaching/ossec2016/a3\\_sourcecode.zip](https://www.cs.ru.nl/~vmoonsamy/teaching/ossec2016/a3_sourcecode.zip)

Compile `vuln.c` using the following command:

```
# gcc -fno-stack-protector vuln.c -o vuln
```

Enable ASLR:

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

### Objectives

- a) Find the addresses 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 a 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 `exploit.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.

---

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

## 2 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.