Lab: Malicious Software

A lab for reflection on trust and reproducible binaries

Daniel Bosk

January 22, 2019

1 Introduction

This laboratory work will cover the topic of malicious software, or malware. Malware comes in many different forms: viruses which infect other programs, worms which actively spread themselves through networks, logic bombs which does nothing until a given criteria is fulfilled. The reason for creating malware differs. In the early days, the main reason was out of curiosity, as was the case of the Morris Worm. Nowadays, the reason ranges from economic gain through ransomware to state sponsored attacks on foreign infrastructure.

To counter these we can use several approaches. One is antivirus software, which tries to detect all sorts of malware—not just viruses—despite the name. Another approach is to configure access control and other mechanisms to prevent running these programs, e.g. by never executing programs on USB-sticks, programs received by email, or programs downloaded from the Web, and so on. However, as the adversarial power grows, the sophistication of the malware growns with it. As such, the protection mechanisms might not always be obvious.

1.1 Aim

The aim of this assignment is for you to reflect on trust in source code and binaries. More specifically, the intended learning outcome is that you will:

- Be able to reason about the detection of malware.
- Have an understanding for how malicious software work and the limitations in detection.

The next section covers what you must read before you understand this assignment and how to do the work. Section 3 covers the work to be done, i.e. how you should learn this. Section 4 covers how it will be examined, i.e. how you show that you have fulfilled the intended learning outcomes given above.

2 Theory

To be able to do this assignment you should first read Chap. 5, 7, 10 in *Computer Security* [Gol11]. Then you should read Sect. 21.3 in *Security Engineering* [And08].

You should then read the classic paper "Reflections on trusting trust" by Thompson [Tho84]. The assignment will focus on the ideas in this paper.

Although you can probably make it without knowing any assembly language in this assignment, it might come in handy. So, read up on some x86-64 assembly and some tools. For this you should read *x86-64 Machine-Level Programming* by Bryant [Bry05]. You also need to be acquainted with some tools, for that reason, study the manual pages for objdump(1), as(1), and gdb(1).

3 Assignment

The first part of this assignment consists of implementing Thompson's idea for including malicious code in the C compiler and then removing it from the source code. The C compiler should be modified so that it injects a printf-statement printing the line "May the Source be with you ...\n" in the beginning of all programs it compiles. This means that after the compiler itself is recompiled it will also be printing this line when compiling programs.

The second part of the assignment is to add a compiler specific injection which injects the printf-injecting code above. Once we have recompiled the compiler with this code, then we can restore the original source code and the compiler will continue to do this despite recompiling it with its original source code.

This first and second part of the assignment will be solved together during a full-class hackathon in the computer lab. There will be a projector with the code for all to see, then we will rotate who will be by the keyboard writing what the rest of the class is saying. This way we will discuss together and write the code together, everyone will thus participate in the process.

The last part of the assignment is to dicsuss the consequences of this, among other things we will discuss the following two questions:

- How can we possibly detect if this attack has occured somewhere?
- What if a powerful adversary would mount this attack against e.g. the GNU C Compiler's main repository?
- How can we prevent this from happening?

4 Examination

To pass this assignment you must first actively participate in the hackathon lab session. You must also actively contribute to the post-coding discussions.

If you cannot participate in the lab session you have to solve the lab yourself, then orally present your solution during one of the lab sessions after the course-end.

Acknowledgements

References

[And08] Ross J. Anderson. *Security Engineering. A guide to building dependable distributed systems*. 2nd ed. Indianapolis, IN: Wiley, 2008. ISBN: 978-0-470-06852-6 (hbk.) URL: http://www.cl.cam.ac.uk/~rja14/book.html.

[Bry05] David R. Bryant Randal E. and O'Hallaron. *x86-64 Machine-Level Programming*. Sept. 2005. URL: https://www.cs.cmu.edu/~fp/courses/15213-s07/misc/asm64-handout.pdf.

[Gol11] Dieter Gollmann. *Computer Security*. 3rd ed. Chichester, West Sussex, U.K.: Wiley, 2011. ISBN: 9780470741153 (pbk.)

[Tho84] Ken Thompson. "Reflections on trusting trust". In: *Communications of the ACM* 27.8 (1984), pp. 761–763. URL: http://dl.acm.org/citation.cfm?id=358210.