Low-Level Software Security for Compiler Developers

Contents

Chapter 1

Introduction

Compilers, assemblers and similar tools generate all the binary code that processors execute. It is no surprise then that for security analysis and hardening relevant for binary code, these tools have a major role to play. Often the only practical way to protect all binaries with a particular security hardening method is to let the compiler adapt its automatic code generation.

With software security becoming even more important in recent years, it is no surprise to see an ever increasing variety of security hardening features and mitigations against vulnerabilities implemented in compilers.

Indeed, compared to a few decades ago, today's compiler developer is much more likely to work on security features, at least some of their time.

Furthermore, with the ever-expanding range of techniques implemented, it has become very hard to gain a basic understanding of all security features implemented in typical compilers.

This poses a practical problem: compiler developers must be able to work on security hardening features, yet it is hard to gain a good basic understanding of such compiler features.

This book aims to help developers of code generation tools such as JITs, compilers, linkers and assemblers to overcome this.

There is a lot of material that can be found explaining individual vulnerabilities or attack vectors. There are also lots of presentations explaining specific exploits. But there seems to be a limited set of material that gives a structured overview of all vulnerabilities and exploits for which a code generator could play a role in protecting against them.

This book aims to provide such a structured, broad overview. It does not necessarily go into full details. Instead it aims to give a thorough description of all relevant high-level aspects of attacks, vulnerabilities, mitigations and hardening techniques. For further details, this book provides pointers to material with more details on specific techniques.

The purpose of this book is to serve as a guide to every compiler developer that needs to learn about software security relevant to compilers. Even though the focus is on compiler developers, we expect that this book will also be useful to other people working on low-level software.

1.1 How this book is created

The idea for this book emerged out of a frustration of not finding a good overview on this topic. Kristof Beyls and Georgia Kouveli, both compiler engineers working on security features, wished a book like this would exist. After not finding such a book, they decided to try and write one themselves. They immediately realized that they do not have all necessary expertise themselves to complete such a daunting task. So they decided to try and create this book in an open source style, seeking contributions from many experts.

As you read this, the book remains unfinished. This book may well never be finished, as new vulnerabilities continue to be discovered regularly. Our hope is that developing the book as an open source project will allow for it to continue to evolve and improve. The open source development process of this book increases the likelihood that it remains relevant as new vulnerabilities and mitigations emerge.

Kristof and Georgia, the initial authors, are far from experts on all possible vulnerabilities. So what is the plan to get high quality content to cover all relevant topics? It is two-fold.

First, by studying specific topics, they hope to gain enough knowledge to write up a good summary for this book.

Second, they very much invite and welcome contributions. If you're interested in potentially contributing content, please go to the home location for the open source project at https://github.com/llsoftsec/llsoftsecbook.

As a reader, you can also contribute to making this book better. We highly encourage feedback, both positive and constructive criticisms. We prefer feedback to be received through https://github.com/llsoftsec/llsoftsecbook.

Add section describing the structure of the rest of the book.

Chapter 2

Memory vulnerability based attacks and mitigations

Write chapter on memory vulnerabilities and mitigation.

Chapter 3

Physical access side-channel attacks

Write chapter on physical access side-channel attacks.

Chapter 4

Remote access side-channel attacks

This chapter covers side-channel attacks for which the attacker does not need physical access to the hardware.

Define side-channel better.

4.1 Timing attacks

An implementation of a cryptographic algorithm can leak information about the data it processes if its run time is influenced by the value of the processed data. Attacks making use of this are called timing attacks.

The main mitigation against such attacks consists of carefully implementing the algorithm such that the execution time remains independent of the processed data. This can be done by making sure that both:

a) The control flow, i.e. the trace of instructions executed, does not change depending on the processed data. This guarantees that every time the algorithm runs, exactly the same sequence of instructions is executed, independent of the processed data.

b) The instructions used to implement the algorithm are from the subset of instructions for which the execution time is known to not depend on the data values it processes.

   For example, in the Arm architecture, the Armv8.4-A DIT extension guarantees that execution time is data-independent for a subset of the AArch64 instructions.

   By ensuring that the extension is enabled and only instructions in the subset are used, data-independent execution time is guaranteed.

At the moment, we do not know of a compiler implementation that actively helps to guarantee both (a) and (b). A great reference giving practical advice on how to achieve (a), (b) and more security hardening properties specific for cryptographic kernels is found in (Pornin 2018).

As discussed in (Pornin 2018), when implementing cryptographic algorithms, you also need to keep cache side-channel attacks in mind, which are discussed in the section on cache side-channel attacks.

4.2 Cache side-channel attacks

Write section on cache side-channel attacks. See the first comment on PR24 for suggestions of what this should contain.

Chapter 5

Other security topics relevant for compiler developers

Write chapter with other security topics.

Write section on securely clearing memory in C/C++ and undefined behaviour.

Appendix: contribution guidelines

Write chapter on contribution guidelines. These should include at least: project locaton on github; how to create pull requests/issues. Where do we discuss - mailing list? Grammar and writing style guidelines. How to use todos and index.

Index

Todo list

References

Pornin, Thomas. 2018. "Why Constant-Time Crypto?" 2018. https://www.bearssl.org/constanttime.html.