École Polytechnique Fédérale de Lausanne

Recovering type information from compiled binaries to aid in instrumentation

by Louis Merlin

Master Thesis

Approved by the Examining Committee:

Prof. Dr. sc. ETH Mathias Payer
Thesis Advisor

Damian Pfammatter
External Expert

Antony Vennard
Thesis Supervisor

EPFL IC IINFCOM HEXHIVE
BC 160 (Bâtiment BC)
Station 14
CH-1015 Lausanne

July 5, 2021

Nobody tosses a dwarf !
— Gimli, son of Glóin

TODO Dedication

Acknowledgments

TODO Acknowledments

Lausanne, July 5, 2021 Louis Merlin

Abstract

The abstract serves as an executive summary of your project. Your abstract should cover at least the following topics, 1-2 sentences for each: what area you are in, the problem you focus on, why existing work is insufficient, what the high-level intuition of your work is, maybe a neat design or implementation decision, and key results of your evaluation.

Contents

Chapter 1

Introduction

The introduction is a longer writeup that gently eases the reader into your thesis [1]. Use the first paragraph to discuss the setting. In the second paragraph you can introduce the main challenge that you see. The third paragraph lists why related work is insufficient. The fourth and fifth paragraphs discuss your approach and why it is needed. The sixth paragraph will introduce your thesis statement. Think how you can distill the essence of your thesis into a single sentence. The seventh paragraph will highlight some of your results The eights paragraph discusses your core contribution.

This section is usually 3-5 pages.

Setting: compiled binaries; RetroWrite; push for DWARF; closed-source C++ binaries in the wild (android?).

Work on C++ began in 1979, as a "C with classes" [3]. Since then, the language has grown in popularity, and even surpassed C itself [2]. Nevertheless, reverse-engineering efforts have been focused towards C binaries, because analysis methods found this way will often work on C++ binaries too.

The recent RetroWrite [1] project by the HexHive lab is a static rewriting tool for x86_64 position-independent binaries. It enables the instrumentation of projects when we do not have access to the source code. This can include a legacy project, a closed-source product or even malware.

[IF WE SUCCEEDED] In this thesis we would like to show how we brought C++ support to RetroWrite, and what research opportunities this will create.

[IF WE FAILED :(] In this thesis we will detail how we tried to bring C++ support to RetroWrite, and what remains to be done for the implementation to work.

Main challenge: C++-specific ELF sections are not as well documented as they could be (?).

Related work: RetroWrite only does C; debugging tools don't have nice support for C++ features; DWARF will be explored more and more in the coming years.

Our approach: Tool for static analysis of C++ binaries, to recover classes and display them in a helpful way. Fixing RetroWrite to support C++ binaries.

Why it is needed: Nobody (?) has been able to rewrite C++ binaries before, this could lead to very interesting discoveries.

Highlight results: 53% of packages on debian that use C++ have extracteable classes, [whatever we are able to do with C++ and RetroWrite]

Core contribution: dis-cover as an open-source tool; [whatever we are able to add to RetroWrite]

Chapter 2

Background

The background section introduces the necessary background to understand your work. This is not necessarily related work but technologies and dependencies that must be resolved to understand your design and implementation.

This section is usually 3-5 pages.

Chapter 3

Design

Introduce and discuss the design decisions that you made during this project. Highlight why individual decisions are important and/or necessary. Discuss how the design fits together.

This section is usually 5-10 pages.

Chapter 4

Implementation

The implementation covers some of the implementation details of your project. This is not intended to be a low level description of every line of code that you wrote but covers the implementation aspects of the projects.

This section is usually 3-5 pages.

Chapter 5

Evaluation

In the evaluation you convince the reader that your design works as intended. Describe the evaluation setup, the designed experiments, and how the experiments showcase the individual points you want to prove.

This section is usually 5-10 pages.

Chapter 6

Related Work

The related work section covers closely related work. Here you can highlight the related work, how it solved the problem, and why it solved a different problem. Do not play down the importance of related work, all of these systems have been published and evaluated! Say what is different and how you overcome some of the weaknesses of related work by discussing the trade-offs. Stay positive!

This section is usually 3-5 pages.

Chapter 7

Conclusion

In the conclusion you repeat the main result and finalize the discussion of your project. Mention the core results and why as well as how your system advances the status quo.

Bibliography

[1] Sushant Dinesh, Nathan Burow, Dongyan Xu, and Mathias Payer. “RetroWrite: Statically Instrumenting COTS Binaries for Fuzzing and Sanitization”. In: *IEEE International Symposium on Security and Privacy*. 2020.

[2] stackoverflow. *Most Popular Technologies*. `https://insights.stackoverflow.com/survey/2020#most-popular-technologies`. Accessed: 2021-06-05.

[3] Bjarne Stroustrup. *When was C++ invented?* `https://www.stroustrup.com/bs_faq.html#invention`. Accessed: 2021-06-05.