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 10, 2021

Nobody tosses a dwarf !
— Gimli, son of Glóin

TODO Dedication

Acknowledgments

TODO Acknowledments

Lausanne, July 10, 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

Work on C++ began in 1979, as a "C with classes" [8]. Since then, the language has grown in popularity, and even surpassed C itself [7]. Examples of well-known C++ projects include the zoom [2] conferencing software, the gold linker [9] or [TODO: find another "well-known" example]. Nevertheless, reverse-engineering efforts have been focused towards C binaries, because analysis methods found this way will often work on C++ binaries too.

This has meant reverse engineering tools like ghidra [5], IDA Pro [6] or Binary Ninja [1] have treated non-C binaries as second-class citizen. These tools will often show C++ specific features as passing comments, failing to show the real implications of a try/catch block or a polymorphic class.

The recent RetroWrite [3] 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.

In this thesis we would like to present the **dis-cover** [4] static analysis tool, as well as improvements made to RetroWrite to support C++ binaries.

The dis-cover tool is able to extract information from a C++ binary, and re-inject it as debug information using the DWARF format into the binary. This enables other debugging tools to see and display this information.

[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.

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.

The C++ programming language implements polymorphism. This feature enables complex code logic that can comply with external business logic for example. Polymorphic classes are defined by having at least one virtual method, which is inheritable and overridable. With this polymorphism comes type conversion, and more interestingly for us the **dynamic_cast** expression. Dynamic casting is a safe kind of type conversion. It only will successfuly cast if the value is of the referenced type or a base type of that type. In order to achieve that, the system must have some kind of information about the object's data type at runtime. This information is called Run-Time Type Information (**RTTI**), and is stored in the binary if it is needed. We will go into more details about the implementation of RTTIs in C++ in later chapters.

DWARF is the debugging standard used widly in conjunction with executable ELF files. It is often included in these ELF files in the **.debug_info** section (and other related sections). This debug information is made up of Debugging Information Entries (**DIEs**). These entries can contain information about variable names, method definitions, the compilation process, or more importantly for us, class names and class inheritance.

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] Vector 35. *Binary Ninja*. https://binary.ninja/. Accessed: 2021-06-06.

[2] Zoom Video Communications. *Zoom Cloud Meetings*. https://zoom.us/. Accessed: 2021-06-09.

[3] 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.

[4] Louis Merlin. *dis-cover*. https://github.com/HexHive/dis-cover. Accessed: 2021-06-06.

[5] NSA. *ghidra*. https://ghidra-sre.org/. Accessed: 2021-06-06.

[6] Hex Rays. *IDA Pro*. https://hex-rays.com/IDA-pro/. Accessed: 2021-06-06.

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

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

[9] Ian Lance Taylor. *Gold Linker*. https://en.wikipedia.org/wiki/Gold_(linker). Accessed: 2021-06-09.