WORCESTER POLYTECHNIC INSTITUTE

DOCTORAL THESIS

Understanding Code

Author:
Zachary PORTER

Supervisor:
Professor Robert WALLS
Professor Gary POLLICE

A thesis submitted in fulfillment of the requirements
for the degree of Bachelors of Computer Science

in the

Cake lab
Computer Science

November 17, 2022

WORCESTER POLYTECHNIC INSTITUTE

Abstract

Faculty Name
Computer Science

Bachelors of Computer Science

Understanding Code

by Zachary PORTER

The Thesis Abstract is written here (and usually kept to just this page). The page is kept centered vertically so can expand into the blank space above the title too...

Acknowledgements

The acknowledgments and the people to thank go here, don't forget to include your project advisor…

Contents

List of Figures

List of Tables

List of Abbreviations

LAH	**L**ist **A**bbreviations **H**ere
WSF	**W**hat (it) **S**tands **F**or

Physical Constants

Speed of Light $c_0 = 2.997\,924\,58 \times 10^8\,\mathrm{m\,s^{-1}}$ (exact)

List of Symbols

a	distance	m
P	power	W (J s^{-1})
ω	angular frequency	rad

For/Dedicated to/To my…

.1 Executive Summary

.2 Introduction

Introduce glibc memory allocator

.3 Background

most improtant is problem definition define key tersma and ideas

.3.1 Current tools for source code understanding

.3.2 Need for progress in this domain

.3.3 Idea progression

.4 Methodology

.4.1 Librr-rs

.4.2 Query Framework

.4.3 Glibc memory allocator case study

.5 Druid

.6 Case study: glibc memory allocator

.7 Results

.7.1

.8 Further work

.9 Conclusion

.10 Efficient Address Recording

.10.1 The problem

When I began my analysis of librr, I quickly noticed that singlestepping and continuing (interrupting the process) incurred a heavy performance penalty (100-1000x pentalty depending on the workload). This is unacceptable for anything but the simplest of programs. As such, I decided to use code stomping to jump into an area that I control which can record addresses without having to interrupt the underlying process.

.10.2 My implementation

Added stomped code:

```
{instructions that were stomped}
XCHG rsp (beginning of my stack)
PUSH (address of instruction that was skipped)
XCHG rsp (beginning of my stack)
```

Setting up the stack: I created two additional memory mapped regions. One is r-w which I call my stack, and the other is rxw which I call my trampoline segment.