Improving the use of Genetic Algorithms for Signature-Based Web Threat Detection

by

Tyler Wilding

A thesis submitted in conformity with the requirements
for the degree of Bachelor of Computer Science
Department of Mathematics and Computer Science
Algoma University

Abstract

Improving the use of Genetic Algorithms for Signature-Based Web Threat Detection

Tyler Wilding
Bachelor of Computer Science
Department of Mathematics and Computer Science
Algoma University
2017

Lorem Ipsum

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Micro-optimization is a technique to reduce the overall operation count of floating point operations. In a standard floating point unit, floating point operations are fairly high level, such as “multiply” and “add”; in a micro floating point unit (μFPU), these have been broken down into their constituent low-level floating point operations on the mantissas and exponents of the floating point numbers.

Chapter two describes the architecture of the μFPU unit, and the motivations for the design decisions made.

Chapter three describes the design of the compiler, as well as how the optimizations discussed in section 1.2 were implemented.

Chapter four describes the purpose of test code that was compiled, and which statistics were gathered by running it through the simulator. The purpose is to measure what effect the micro-optimizations had, compared to unoptimized code. Possible future expansions to the project are also discussed.

1.1 Problem Definition

The idea of micro-optimization is motivated by the recent trends in computer architecture towards low-level parallelism and small, pipelineable instruction sets [1]. By getting rid of more complex instructions

Figure 1.1: Armadillo slaying lawyer.

Table 1.1: Armadillo slaying lawyer.

and concentrating on optimizing frequently used instructions, substantial increases in performance were realized.

Another important motivation was the trend towards placing more of the burden of performance on the compiler. Many of the new architectures depend on an intelligent, optimizing compiler in order to realize anywhere near their peak performance. In these cases, the compiler not only is responsible for faithfully generating native code to match the source language, but also must be aware of instruction latencies, delayed branches, pipeline stages, and a multitude of other factors in order to generate fast code.

Taking these ideas one step further, it seems that the floating point operations that are normally single, large instructions can be further broken down into smaller, simpler, faster instructions, with more control in the compiler and less in the hardware. This is the idea behind a micro-optimizing FPU; break the floating point instructions down into their basic components and use a small, fast implementation, with a large part of the burden of hardware allocation and optimization shifted towards compile-time.

Along with the hardware speedups possible by using a μFPU, there are also optimizations that the compiler can perform on the code that is generated. In a normal sequence of floating point operations, there are many hidden redundancies that can be eliminated by allowing the compiler to control the floating point operations down to their lowest level. These optimizations are described in detail in section 1.2.

1.2 Objective

In order to perform a sequence of floating point operations, a normal FPU performs many redundant internal shifts and normalizations in the process of performing a sequence of operations. However, if a compiler can decompose the floating point operations it needs down to the lowest level, it then can optimize away many of these redundant operations.

If there is some additional hardware support specifically for micro-optimization, there are additional optimizations that can be performed. This hardware support entails extra "guard bits" on the standard floating point formats, to allow several unnormalized operations to be performed in a row without the loss information[1]

1.2.1 Hypothesis

1.3 Thesis Overview

1.3.1 Scope

1.3.2 Limitations

[1]A description of the floating point format used is shown in figures and. A discussion of the mathematics behind unnormalized arithmetic is in appendix.

Chapter 2

Description of Web-Threats

2.1 What is a Web Threat

2.2 SQL Injection

2.3 Cross-site Scripting

2.4 Remote File Inclusion

Chapter 3

Current Detection and Prevention Methods

3.1 Development Related Prevention(Better Coding Practices)

3.2 Older Methods of Detection (Before Signature Based)

3.3 Signature Based Detection

3.3.1 How is Traditional Based Signature Detection now Falling Short

Chapter 4

Recent Developments and Potential Solutions

4.1 Improving Signature Based Detection with Genetic Algorithms

4.1.1 Existing Results with this approach

4.2 Alternative Solutions

4.2.1 Classification Tools (SVM)

4.3 Conclusions and Analysis on Potential Solutions

Chapter 5

Methods & Procedures

5.1 Testing and Training Data

5.2 Genetic Algorithm Based Signature Detection

5.2.1 Advantages and Disadvantages

5.2.2 Measured Metrics

5.2.3 Algorithm and Testing Procedure

5.3 Support Vector Machine Detection

5.3.1 Advantages and Disadvantages

5.3.2 Metrics

5.3.3 Algorithm and Testing Procedure

Chapter 6

Results

6.1 Genetic Algorithm

6.1.1 Training

6.1.2 Testing

6.1.3 Genetic Algorithm Bitstring Length Comparison

6.1.4 Genetic Algorithm vs Random Permutations

6.1.5 Genetic Algorithm vs Random Permutations with Fitness

6.2 Support Vector Machine

6.2.1 Training

6.2.2 Testing

Chapter 7

Discussion

7.1 Genetic Algorithm

7.2 Support Vector Machine

Chapter 8

Conclusions

8.1 Future Work

8.2 Conclusions

Chapter 9

Glossary

Chapter 10

Appendices

Bibliography

[1] I. Freely, “A small paper,” *The journal of small papers*, vol. -1, 1997. to appear.