A Flexible Framework for Leveraging Verification Tools to Enhance the Verification Technologies Available for Policy Enforcement

James Larkin

A Thesis submitted for the degree of Master of Science (Computer Science)

School of Information Technology

Bond University

March 31, 2009

Statement of Originality

The material in this thesis has not been previously submitted for a degree or diploma in any university. To the best of my knowledge this thesis contains no material previously published or written by another person except where due acknowledgment is made in the thesis itself.

James Larkin

Acknowledgments

Firstly and foremostly I would like to thank my supervisors Dr. Padmanabhan Krishnan and Dr. Phil Stocks for their time, feedback and their continual support over the years of my research. I would especially like to thank them for their efforts and for proof-reading drafts of this thesis.

I would also like to thank the Australian Government for providing me with an Australian Postgraduate Research Award.

I would also like to thank my family for their on going support. In particular the vital role of my parents for their continual love and encouragement and my brother Henry for his feedback on drafts of this thesis and continual encouragement.

In addition, I would like to thank my office mates Shane Bracher, Percy Pari Salas, James Montgomery and Keith Hales who over the years made my time very enjoyable and were always there for support, encouragement and feedback.

Finally, I would like to thank all my friends, in particular Adrian Gepp, for their support and for making the duration of my thesis very enjoyable.

Abstract

Program verification is vital as more and more users are creating, downloading and executing foreign computer programs. Software verification tools provide a means for determining if a program adheres to a user's security requirements, or security policy. There are many verification tools that exist for checking different types of policies on different types of programs. Currently however, there is no verification tool capable of determining if all types of programs satisfy all types of policies.

This thesis describes a framework for supporting multiple verification tools to determine program satisfaction. A user's security requirements are represented at multiple levels of abstraction as Intermediate Execution Environments. Using a sequence of configurations, a user's security requirements are transformed from the abstract level to the tool level, possibly for multiple verification tools. Using a number of case studies, the validity of the framework is shown.

Contents

1 Introduction **1**
- 1.1 Program Verification . . . 2
- 1.2 Limitations of Program Verifiers . . . 3
- 1.3 Goals of this Dissertation . . . 4
- 1.4 Structure of this Dissertation . . . 5

2 Literature Review **6**
- 2.1 Policy Specification . . . 6
  - 2.1.1 Models of Security . . . 7
  - 2.1.2 Security Policy Representations . . . 9
- 2.2 Policy Enforcement . . . 15
  - 2.2.1 Characterising Security Policy Verification Technologies . . . 15
  - 2.2.2 Restriction Mechanisms . . . 17
  - 2.2.3 Verification Checking Mechanisms . . . 18
  - 2.2.4 Verification Technology Summary . . . 25
- 2.3 Analysis of Techniques for Leveraging Verification Technologies . . . 26
  - 2.3.1 The Heterogeneous Tool Set . . . 27
  - 2.3.2 Conclusion . . . 28

3 Leveraging Verification Technologies **31**
- 3.1 Thesis Framework . . . 32
- 3.2 Notation . . . 33
- 3.3 Execution Environments . . . 34
  - 3.3.1 Transforming and Linking Execution Environments . . . 35
- 3.4 Intermediate Execution Environments . . . 38
  - 3.4.1 Transforming Intermediate Execution Environments . . . 40
- 3.5 Implementation . . . 47
  - 3.5.1 IEEs . . . 47
  - 3.5.2 Configurations . . . 48
  - 3.5.3 The Transformer . . . 48

4 Case Studies and Examples **50**
4.1 Memory Case Study 51
4.1.1 Configurations 52
4.1.2 Memory Verification 56
4.2 Bank Case Study 59
4.2.1 Configurations 60
4.2.2 Verification of Thread Atomicity 61
4.3 Millionaire Case Study 62
4.3.1 *Millionaire* Background 63
4.3.2 Adaptation of *Millionaire* 63
4.3.3 Security Concerns 65
4.3.4 *Millionaire* Security Policy 66
4.3.5 Implementation of the *Millionaire* Application 74
4.3.6 Configurations 75
4.3.7 Program Verification 81
4.3.8 Discussion 82

5 Discussion **86**
5.1 Conclusion 87
5.2 Future Work 89

List of Figures

3.1 Framework 32
3.2 An example graph showing the possible configurations to apply to an IEE. 36
3.3 Framework using a sequence of configurations. 39
3.4 Example of a graph that uses configurations to create an intermediate level. 40

4.1 Graph of possible configurations for safe memory. The 57
4.2 Pseudo code for crediting and debiting money to and from an account. . . 59
4.3 Scenario that involves thread interference. 59
4.4 Graph showing each security concern represented as a separate IEE 66
4.5 Path of Transformations taken for the Millionaire Policy 82