



<http://researchspace.auckland.ac.nz>

## ***ResearchSpace@Auckland***

### **Copyright Statement**

The digital copy of this thesis is protected by the Copyright Act 1994 (New Zealand).

This thesis may be consulted by you, provided you comply with the provisions of the Act and the following conditions of use:

- Any use you make of these documents or images must be for research or private study purposes only, and you may not make them available to any other person.
- Authors control the copyright of their thesis. You will recognise the author's right to be identified as the author of this thesis, and due acknowledgement will be made to the author where appropriate.
- You will obtain the author's permission before publishing any material from their thesis.

To request permissions please use the Feedback form on our webpage.

<http://researchspace.auckland.ac.nz/feedback>

### **General copyright and disclaimer**

In addition to the above conditions, authors give their consent for the digital copy of their work to be used subject to the conditions specified on the Library Thesis Consent Form.

---

*Department of Computer Science  
The University of Auckland  
New Zealand*

---

# Design and Evaluation of Software Obfuscations

---

*Anirban Majumdar*

*2008*

*Supervisors:*

*Prof. Clark D. Thomborson*

*Dr. Stephen J. Drape*



A THESIS SUBMITTED IN PARTIAL FULFILMENT OF  
THE REQUIREMENTS OF DOCTOR OF PHILOSOPHY  
IN COMPUTER SCIENCE



# The University of Auckland

## Thesis Consent Form

This thesis may be consulted for the purpose of research or private study provided that due acknowledgement is made where appropriate and that the author's permission is obtained before any material from the thesis is published.

I agree that the University of Auckland Library may make a copy of this thesis for supply to the collection of another prescribed library on request from that Library; and

1. I agree that this thesis may be photocopied for supply to any person in accordance with the provisions of Section 56 of the Copyright Act 1994.

Or

- ~~2. This thesis may not be photocopied other than to supply a copy for the collection of another prescribed library.~~

*(Strike out 1 or 2)*

Signed: .....

Date: .....

Created: 5 July 2001

Last updated: 30 September 2007



# Abstract

Software obfuscation is a protection technique for making code unintelligible to automated program comprehension and analysis tools. It works by performing semantic preserving transformations such that the difficulty of automatically extracting the computational logic out of code is increased. Obfuscating transforms in existing literature have been designed with the ambitious goal of being resilient against all possible reverse engineering attacks. Even though some of the constructions are based on intractable computational problems, we do not know, in practice, how to generate hard instances of obfuscated problems such that all forms of program analyses would fail.

In this thesis, we address the problem of software protection by developing a weaker notion of obfuscation under which it is not required to guarantee an absolute blackbox security. Using this notion, we develop provably-correct obfuscating transforms using dependencies existing within program structures and indeterminacies in communication characteristics between programs in a distributed computing environment. We show how several well known static analysis tools can be used for reverse engineering obfuscating transforms that derive resilience from computationally hard problems. In particular, we restrict ourselves to one common and potent static analysis tool, the static slicer, and use it as our attack tool. We show the use of derived software engineering metrics to indicate the degree of success or failure of a slicer attack on a piece of obfuscated code. We address the issue of proving correctness of obfuscating transforms by adapting existing proof techniques for functional program refinement and communicating sequential processes.

The results of this thesis could be used for future work in two ways: first, future researchers may extend our proposed techniques to design obfuscations using a wider range of dependencies that exist between dynamic program structures. Our restricted attack model using one static analysis tool can also be relaxed and obfuscations capable of withstanding a broader class of static and dynamic analysis attacks could be developed based on the same principles. Secondly, our obfuscatory strength evaluation techniques could guide anti-malware researchers in the development of tools to detect obfuscated strains of polymorphic viruses.



# Acknowledgements

This thesis has benefitted immensely from the wisdom and advice of many individuals. This is a good opportunity to thank them all.

First and foremost, I am deeply indebted to my supervisors Professor Clark Thomborson and Dr. Stephen Drape for mentoring me towards a completion in the minimum allowable time. Over the last three years, Professor Thomborson has given me both the freedom to pursue my own research interests and the insightful guidance that I have needed. Through him I learnt the art of lateral thinking and the use of rigour in scientific research. I am grateful to him for funding my research through the New Zealand government's New Economy Research Foundation fund on "Securing Software for New Zealand". I am extremely fortunate to have had Dr. Drape as my co-supervisor over the period of last eighteen months. Dr. Drape has not only been an excellent thesis advisor but also a valued friend. During the troubled times of funding uncertainty, he has taught me to persevere and remain focussed on my research. Our mutual admiration for progressive rock and Ricky Gervais resulted in some memorable time spent at rock concerts and in front screens appreciating countless episodes of *The Office*. Dr. Drape has been instrumental in introducing me to programming language theory and I am particularly indebted for his help in adapting a framework for proving correctness of imperative transforms. I am thankful to Professor Thomborson for his help in shaping up the final thesis draft and to Dr. Drape for taking time off of his busy schedule at Oxford University to proofread my thesis drafts.

I have benefitted immensely from interactions with the former members of the Secure Systems Group. I grateful to Dr. Antoine Monsifrot, of Thomson R & D France, for his collaborative efforts on static analysis experiments with BDDBDDDB and Indus. Dr. Jasvir Nagra, of University of Trento, has been my peer mentor all these years and have enlightened me with his mastery of two dozen programming languages and analytical skills. Michael Stay, of Google Mountain View Labs, fascinated me with his alternative view of distributed opaque predicates using the cryptographic theory of multi-party computations. I have thoroughly enjoyed the intriguing discussions on computer forensics and technorealism with Dr. Simson Garfinkel of Harvard University. I am thankful to my former mentors Professor Hon Fung Li of Concordia University, Canada, Professor Subhansu Bandopadhyay and Dr. Nabendu Chaki of Calcutta University, India for their unvarying impetus that catalysed my yearning to take up research in computer science as my professional career. I am thankful to my school and university friends in India and abroad for providing me with encouragement and entertainment right from the beginning. I am thankful to Dr. David Pearce, my oral examiner, for his help in improving the overall quality of the thesis with his detailed suggestions.

My greatest gratitude goes to my parents for their unconditional support during the course of my PhD. My wonderful Ma has nurtured me with love and care all my life.



She has been the source of limitless encouragement during these difficult years. My dad motivated my inquisitiveness from very early on and supported my career decisions no matter where they took me. Being a scientist himself, my dad provided immensely valuable suggestions at critical junctures of my research and thesis writing. I am thankful to my sister for helping me choose a career in computer science over medicine right after my high school days. I thank Sikharini Majumdar for her encouraging phone conversations and for generously sharing the plight of fellow PhD students across the Pacific. She made me feel that I am not alone, after all. Finally, I am thankful to my wife Raka; her unwavering patience and encouragement during these trying times have made it all worth it. I could not have envisioned this personal journey without her. It is to Raka that I dedicate this thesis.

# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	Alternative protection techniques . . . . .	3
1.2	Motivation . . . . .	4
1.3	Outline of the thesis . . . . .	6
<b>2</b>	<b>Obfuscation and Related Work</b>	<b>7</b>
2.1	Defining Obfuscation . . . . .	8
2.2	Data obfuscations . . . . .	10
2.2.1	Variable encoding . . . . .	11
2.2.2	Variable splitting and merging . . . . .	11
2.2.3	Array transformations . . . . .	12
2.3	Control-flow obfuscations . . . . .	13
2.3.1	The dynamic dispatcher model of control-flow obfuscation . . . . .	15
2.3.2	Opaque predicates . . . . .	19
2.4	Conclusion . . . . .	24
<b>3</b>	<b>Evaluation of Obfuscatory Strength</b>	<b>27</b>
3.1	Background . . . . .	28
3.1.1	Reverse Engineering and deobfuscation . . . . .	28
3.1.2	Obfuscatory strength evaluation . . . . .	30

---

3.2	The attack process . . . . .	31
3.3	Developing an attack model . . . . .	34
3.3.1	Concept lattices . . . . .	34
3.3.2	Points-to analysis . . . . .	41
3.3.3	Program slicing . . . . .	45
3.3.4	Discussion . . . . .	47
3.4	Conclusion . . . . .	51
<b>4</b>	<b>Design and Evaluation of Slicing Obfuscations</b>	<b>53</b>
4.1	Background . . . . .	54
4.1.1	Program Slicing . . . . .	55
4.1.2	Slicing for program comprehension . . . . .	57
4.1.3	Slicing Notation . . . . .	59
4.2	Using slicing to define obfuscation . . . . .	61
4.3	Metrics . . . . .	63
4.3.1	Choosing a slicer . . . . .	65
4.3.2	Slicing residues . . . . .	66
4.4	Illustration with an example . . . . .	68
4.4.1	The Word Count Example . . . . .	68
4.4.2	Adding a bogus predicate . . . . .	69
4.4.3	Variable Encoding . . . . .	71
4.4.4	Adding to the guard of a <b>while</b> loop . . . . .	73
4.4.5	Using a Variable Split . . . . .	74
4.4.6	Arrays . . . . .	75
4.5	Further Examples . . . . .	76
4.5.1	Product and Sum . . . . .	76
4.5.2	Search Sort . . . . .	78
4.5.3	Rover . . . . .	79
4.5.4	Scattering . . . . .	80

---

4.5.5	Results and discussion . . . . .	81
4.6	Conclusion . . . . .	84
<b>5</b>	<b>Proving Slicing Obfuscations Correct</b>	<b>87</b>
5.1	Proof Framework . . . . .	88
5.1.1	Modelling statements as functions . . . . .	89
5.1.2	Using the refinement framework . . . . .	91
5.1.3	Obfuscating Statements . . . . .	93
5.1.4	Simultaneous Equations . . . . .	95
5.1.5	Steps in a proof . . . . .	96
5.2	Proving obfuscations correct . . . . .	98
5.2.1	Correctness of a bogus predicate . . . . .	98
5.2.2	Correctness of a variable encoding . . . . .	98
5.2.3	Correctness of a <b>while</b> loop . . . . .	100
5.2.4	Correctness of a variable split . . . . .	103
5.2.5	Correctness of an array transformation . . . . .	104
5.3	Applying the transformations . . . . .	106
5.3.1	Placing the transforms . . . . .	106
5.3.2	Localising the transformations . . . . .	107
5.3.3	Combining transformations . . . . .	108
5.4	Conclusion . . . . .	110
<b>6</b>	<b>Designing Distributed Opaque Predicates</b>	<b>111</b>
6.1	The malicious host problem of mobile agents . . . . .	112
6.2	Distributed computing and global properties . . . . .	118
6.3	Designing distributed opaque predicates . . . . .	120
6.4	Engineering Distributed Opaque Predicates . . . . .	121
6.4.1	Selecting <i>guard</i> processes . . . . .	122
6.4.2	Incorporating a Knapsack problem instance . . . . .	123
6.4.3	Defining the local state update rules . . . . .	125

---

6.4.4	Selection of communication pattern and message types . . . . .	126
6.4.5	Embedding distributed opaque predicates . . . . .	132
6.5	Conclusion . . . . .	134
<b>7</b>	<b>Evaluation of Distributed Opaque Predicates</b>	<b>137</b>
7.1	Correctness of message-passing programs . . . . .	138
7.1.1	Proof framework . . . . .	138
7.1.2	Proof rules . . . . .	141
7.1.3	Proof sketches . . . . .	144
7.2	Attacking distributed opaque predicates . . . . .	150
7.3	The attack model . . . . .	156
7.3.1	Slicing attack . . . . .	158
7.3.2	Evaluation attack . . . . .	160
7.3.3	Substitution attack . . . . .	169
7.4	Conclusion . . . . .	169
<b>8</b>	<b>Concluding Remarks and Future Work</b>	<b>173</b>
8.1	Thesis contributions . . . . .	174
8.2	Linking up two approaches . . . . .	176
8.3	Open issues and future work . . . . .	176
	<b>Bibliography</b>	<b>188</b>

# List of Figures

2.1	Array obfuscations . . . . .	13
2.2	An example program and its CFG . . . . .	16
2.3	CFG after basic flattening . . . . .	16
2.4	CFG after adding data flow . . . . .	17
2.5	CFG after adding dummy blocks and aliasing . . . . .	18
2.6	Random nondeterministic opaque predicate . . . . .	21
2.7	Opaque predicates from aliases . . . . .	22
3.1	Relationship between forward and reverse engineering . . . . .	29
3.2	An overview of the attack process . . . . .	32
3.3	A simple gcd calculating test code. . . . .	35
3.4	Obfuscated code using identifier renaming. . . . .	36
3.5	An example program for concept lattices . . . . .	38
3.6	A concept lattice . . . . .	39
3.7	A concept lattice from KABA . . . . .	40
3.8	Concept lattice of gcd code . . . . .	41
3.9	Concept lattice of obfuscated code . . . . .	41
3.10	An example of allocation site sensitivity . . . . .	45
3.11	A slice using Indus . . . . .	47
3.12	Example of a bigger slice using Indus . . . . .	48

---

4.1	A sum/product example . . . . .	60
4.2	Simple obfuscation on sum/product example . . . . .	61
4.3	A Word Count example . . . . .	68
4.4	Bar charts with residue metrics . . . . .	70
4.5	Adding bogus predicates . . . . .	71
4.6	A simple variable encoding . . . . .	72
4.7	A three variable encoding example . . . . .	72
4.8	Adding a new loop variable . . . . .	73
4.9	A backward slice of variable split . . . . .	74
4.10	Array transformations . . . . .	75
4.11	Bar chart with residue metric values . . . . .	83
5.1	A commuting diagram for data obfuscation . . . . .	92
5.2	Rewrite rules for a variable encoding . . . . .	99
5.3	Correctness proof for variable encoding . . . . .	100
5.4	Proof of correctness for a <b>while</b> loop . . . . .	103
5.5	Correctness proof for variable split . . . . .	105
5.6	An example of composition of obfuscations . . . . .	109
6.1	The attack tree . . . . .	117
6.2	Example of protected processes with guards . . . . .	123
6.3	Doubly circular linked-list configuration . . . . .	124
6.4	Local state update rules . . . . .	126
6.5	Vector Clock update rules . . . . .	128
6.6	A nondeterministic event diagram . . . . .	128
6.7	No ordered delivery . . . . .	130
6.8	No delivery guarantee . . . . .	131
6.9	A deterministic event diagram . . . . .	132
6.10	A pseudo code with distributed opaque predicates . . . . .	135

---

7.1	A two process illustration with no restricted postcondition . . . . .	144
7.2	A two process illustration with restricted postcondition . . . . .	145
7.3	A two process illustration with an acknowledgement message . . . . .	146
7.4	Two guard processes $G1$ and $G2$ sending messages to $P$ . . . . .	148
7.5	Process $P$ receiving messages from $G1$ and $G2$ . . . . .	149
7.6	Process $P$ with strengthened guards . . . . .	150
7.7	$P_1$ : Sum/Product example with an Opaquely True predicate . . . . .	152
7.8	Agent thread path of execution . . . . .	154
7.9	UML class diagram . . . . .	155
7.10	$P_1$ under slicing attack using $\Phi$ as slicing criterion . . . . .	159
7.11	$P_2$ under slicing attack using $\Phi_D$ as slicing criterion . . . . .	161
7.12	An evaluation attack scenario . . . . .	163
7.13	A simple event diagram . . . . .	166
7.14	Lattice for three process message-passing . . . . .	167
7.15	$P_1$ after $\Phi$ is substituted . . . . .	169





# List of Tables

2.1	Correlated dynamic opaque predicates . . . . .	23
4.1	Table of results for the residues of our Word Count example . . . . .	69
4.2	Slicing metrics for all programs . . . . .	77
4.3	Residue metrics for all programs . . . . .	82
7.1	Evaluation trace for $\Phi$ for $P_1$ . . . . .	162

