

Software-based Reference Protection for Component Isolation

By

Yuval Yarom

September 12, 2014

A Thesis Submitted for the Degree of
Doctor of Philosophy
In the School of Computer Science
University of Adelaide

Contents

Abstract	viii
Declaration	ix
Acknowledgements	x
1 Introduction	1
1.1 Classification of Reference Protection Systems	5
1.2 Exported Types	7
1.3 Implementation of Exported Types	8
1.4 Summary of Contributions	8
1.5 Thesis Structure	9
2 A Model and Classification of Reference Protection	10
2.1 Zones	10
2.2 Complete Isolation	15
2.2.1 MVM	16
2.2.2 JX	16
2.2.3 The .Net Framework	17
2.2.4 JNode	18
2.2.5 OVM	18
2.2.6 Summary	19
2.3 Object Sharing	20
2.3.1 KaffeOS	20
2.3.2 Singularity	21
2.3.3 Rust	22
2.3.4 XMem	23
2.3.5 CoLoRS	24
2.3.6 Real-Time Java	24
2.3.7 Summary	25
2.4 Partial Isolation	25
2.4.1 J-Kernel	26

2.4.2	Confined Types	27
2.4.3	Ownership Types	27
2.4.4	Summary	28
2.5	Initial Isolation	28
2.5.1	I-JVM	29
2.5.2	Alta	29
2.5.3	Summary	29
2.6	Isolation Techniques	30
2.6.1	Reachability Roots	30
2.6.2	Controlling Reference Propagation	33
2.7	Evaluation of the Classification	34
3	Exported Types	38
3.1	Type Systems	39
3.2	Partial Isolation and Types	41
3.3	Component Name Spaces	44
3.4	Creating the Interface Zones	45
3.5	The Remote Interface	47
3.6	Compatibility with Language Features	49
3.6.1	Structure-based Type Equivalence	49
3.6.2	Implicit Operations on Types	50
3.6.3	Universal Methods	51
3.6.4	Special Semantics of Types	52
3.6.5	Reflection	53
3.7	Summary	53
4	An Implementation of Exported Types	55
4.1	The Java Virtual Machine	58
4.1.1	Class Loaders	59
4.1.2	Security Manager	60
4.1.3	Byte Code Verifier	61
4.1.4	Summary	62
4.2	JikesRVM	62
4.2.1	Compilation Framework	63
4.2.2	Memory Management	64
4.2.3	Compiler Magic	65
4.2.4	Library Interface	66
4.2.5	Virtual Machine Build	66
4.2.6	Security	67
4.3	An Overview of S-RVM	69

4.4	Import and Export	72
4.5	The VM Interface Layer	73
4.6	Creating a Trust Boundary	76
4.7	Privileged Access for the virtual machine Task	77
4.8	Exceptions	79
4.9	Implementation Verification	80
4.10	Summary	81
5	Performance Evaluation	82
5.1	Memory Usage	83
5.2	Application Task Startup	85
5.3	Steady-State Execution Speed	86
5.4	Summary	92
6	Summary and Conclusions	93
6.1	Classification Framework	93
6.2	The Exported Types Design	94
6.3	S-RVM	95
6.4	Revisiting the Classification Framework	96
6.5	A Multi-tasking Virtual Machine	99
6.6	Structure-based Type Equivalence	100
6.7	Summary	101
A	Exported Types Specifications	102
A.1	Type Systems	102
A.2	Exported Types	103
A.3	Mapping to zones	105
B	S-RVM Implementation Details	107
B.1	The Upcall Interface	107
B.2	Initialising RVMTask	110
B.3	String Backing Store	111
B.4	Exception Conversion	114
C	Performance Data	117
	Bibliography	138

List of Tables

2.1	Methods for controlling reference propagation	34
2.2	Zones in systems providing reference protection	35
2.3	Classification of reference protection	36
2.4	Classification of reference protection	37
5.1	Minimum heap size	84
5.2	Normalised execution times of the DaCapo benchmarks on S-RVM relative to JikesRVM with 90% confidence intervals (Same Heap Size scenario)	88
5.3	Normalised execution times of the DaCapo benchmarks on S-RVM relative to JikesRVM with 90% confidence intervals (Same Heap Pres- sure scenario)	89
6.1	Zones in systems providing reference protection	97
6.2	Classification of reference protection	98
6.3	Classification of reference protection	99
C.1	Mean execution times of the DaCapo benchmarks at the 1 st iteration (ms)	118
C.2	Mean execution times of the DaCapo benchmarks at the 2 nd iteration (ms)	119
C.3	Mean execution times of the DaCapo benchmarks at the 3 rd iteration (ms)	120
C.4	Mean execution times of the DaCapo benchmarks at the 4 th iteration (ms)	121
C.5	Mean execution times of the DaCapo benchmarks at the 5 th iteration (ms)	122
C.6	Mean execution times of the DaCapo benchmarks at the 6 th iteration (ms)	123
C.7	Mean execution times of the DaCapo benchmarks at the 7 th iteration (ms)	124

C.8	Mean execution times of the DaCapo benchmarks at the 8 th iteration (ms)	125
C.9	Mean execution times of the DaCapo benchmarks at the 9 th iteration (ms)	126
C.10	Mean execution times of the DaCapo benchmarks at the 10 th iteration (ms)	127
C.11	Mean execution times of the DaCapo benchmarks at the 11 th iteration (ms)	128
C.12	Mean execution times of the DaCapo benchmarks at the 12 th iteration (ms)	129
C.13	Mean execution times of the DaCapo benchmarks at the 13 th iteration (ms)	130
C.14	Mean execution times of the DaCapo benchmarks at the 14 th iteration (ms)	131
C.15	Mean execution times of the DaCapo benchmarks at the 15 th iteration (ms)	132
C.16	Mean execution times of the DaCapo benchmarks at the 16 th iteration (ms)	133
C.17	Mean execution times of the DaCapo benchmarks at the 17 th iteration (ms)	134
C.18	Mean execution times of the DaCapo benchmarks at the 18 th iteration (ms)	135
C.19	Mean execution times of the DaCapo benchmarks at the 19 th iteration (ms)	136
C.20	Mean execution times of the DaCapo benchmarks at the 20 th iteration (ms)	137

List of Figures

1.1	The spectrum of isolation using reference protection	6
2.1	A component system with isolated zones	12
2.2	A component system with isolated zones and a shared zone	13
2.3	A component system using sealed zones	13
2.4	Structure of the JX system	17
2.5	Heap structure in KaffeOS	21
2.6	The exchange heap in singularity	22
2.7	Taxonomy of methods for handling reachability roots	31
3.1	A component system with partial isolation	42
3.2	Overlaying the sealed zones model over the type hierarchy	43
3.3	Sealed zones	44
3.4	Type hierarchies with exported types	45
3.5	Type hierarchies after import	46
3.6	Extending an imported type	47
4.1	The JikesRVM runtime environment	63
4.2	Type hierarchy in JikesRVM	68
4.3	The S-RVM runtime environment	69
4.4	RVMTask class diagram	71
4.5	Bi-directional communication between tasks	74
4.6	ClassLoader class hierarchy	74
5.1	S-RVM Boot image overhead	84
5.2	Class loading during task startup	85
5.3	Mean execution time on S-RVM relative to JikesRVM.	90
5.4	Steady state normalised execution times of the DaCapo benchmarks on S-RVM at several heap sizes	91
5.5	Separate vs. combined profiling	92
6.1	A multi-tasking virtual machine	99

Abstract

Reference protection mechanisms are commonly used to isolate and to provide protection for components that execute within a shared run-time environment. These mechanisms often incur an overhead due to maintaining the isolation or introduce inefficiencies in the communication between the components. Past research operated under the assumption that some performance loss is an acceptable price for the added security that comes with better isolation. This thesis sets out to demonstrate that good isolation does not imply performance loss.

While numerous models for implementing reference protection have been suggested, there is a lack of a unified terminology that allows the comparison of systems from across the domain. This thesis presents a classification framework that captures the trade-offs present in the design of reference protection. It identifies four main models of reference protection: *complete isolation*, where components do not share references to objects; *object sharing*, where components can share data while still maintaining private, unshared data; *partial isolation*, where components have private, unshared data and an exposed interface that allows other component's indirect access to the private data; and *initial isolation*, where components are isolated when created, but the model allows the programmer to share references without restriction.

Applying the classification to systems providing reference protection identifies a gap in the prior research. Partial isolation promises the level of security expected from component isolation combined with efficient communication. Yet, the only implementation of partial isolation of components uses expensive run-time checks to enforce the protection.

To bridge this gap, this thesis presents the Exported Types design. Exported Types is a type system design that enforces partial isolation at compile time. Using compile-time checks removes the run-time overhead of enforcing the protection model. The design is applied to a meta-circular Java virtual machine to isolate the virtual machine code from the application. Applying reference protection in this scenario reduces the number of classes the virtual machine exposes to the application by two orders of magnitude. Performance tests demonstrate that reference protection, and the higher security it provides, are achieved at no performance cost.

Declaration

I certify that this work contains no material which has been accepted for the award of any other degree or diploma in any university or other tertiary institution and, to the best of my knowledge and belief, contains no material previously published or written by another person, except where due reference has been made in the text. In addition, I certify that no part of this work will, in the future, be used in a submission for any other degree or diploma in any university or other tertiary institution without the prior approval of the University of Adelaide and where applicable, any partner institution responsible for the joint-award of this degree.

I give consent to this copy of my thesis, when deposited in the University Library, being made available for loan and photocopying, subject to the provisions of the Copyright Act 1968.

I also give permission for the digital version of my thesis to be made available on the web, via the University's digital research repository, the Library catalogue and also through web search engines, unless permission has been granted by the University to restrict access for a period of time.

Yuval Yarom
September 12, 2014

Acknowledgements

Ithaka gave you the marvelous journey.
Without her you would not have set out.
She has nothing left to give you now.
—Constantine Cavafy, “Ithaka”

The journey described in the following pages would not have been possible without the dedicated guidance of my supervisors Dave Munro and Katrina Falkner. It is through the wisdom they shared, the encouragement they provided and the patience they exercised that this work has come to fruition. It was an honour working with them and the lessons they have taught will stay with me for the rest of my life.

I would also like to thank Henry Detmold who for a long time acted as an unofficial supervisor. Many thanks also to Steve Blackburn, Ron Morrison and John Zigman for the help they provided and their useful suggestions in various stages of this work. Thanks are also due to Richard Jones and to the anonymous examiner for the careful reading of this thesis and for the wise and helpful comments provided.

I am indebted to Kathy Cooper, Cathie Liersch, Sharyn Liersch, Julie Mayo, Jo Rogers and Tracey Young who helped navigating the treacherous waters of the university administration and for providing some geek-free refuge. Thanks also to William Brodie-Tyrrell, Diana Hill, Peter Kelly, Ekim Kocadag, Joseph Kuehn, Matt Lowry, Peter Nguyen, Travis Olds and Stani Ovtcharova who, over the years, shared the working space with me and helped make the time spent on this PhD an enjoyable experience.

The sages teach that without food there is no learning.¹ It would be futile to try listing all fine establishments providing sustenance around the university. However, special mention must be made of the spicy chicken at Raah Cafe, the mayonnaise chicken at Zen Garden and everything at Vego and Love’n It.

Most of all, I would like to thank my wife Karen Gekker and my children Yarden and Itay for their help, support and encouragement throughout the years. I acknowledge and deeply appreciate the sacrifices they made to allow me to continue my studies.

¹Ethics of the Fathers 3:17 **אם אין קמח, אין תורה**. Lit.:*If there is no flour, there is no (study of the) Torah.*