



AIN SHAMS UNIVERSITY
FACULTY OF COMPUTER
& INFORMATION SCIENCES
Abbassia, Cairo, Egypt.

Developing a Software Development and Analysis Framework using Runtime Monitoring

A thesis submitted to the Department of Information Systems Faculty of Computer and Information Sciences Ain Shams University, Egypt

In partial fulfillment of the requirements for the degree of Doctor of Philosophy, PhD, of Computer and Information Sciences

BY

Soha M Nabil AbdelHamid Hussein

Assistant lecturer in the Department of Information Systems
Faculty of Computer & Information Sciences Ain Shams
University, Egypt

Under the supervision of

Prof. Dr. Mohamed Essam Khalifa

Ex-Dean of the Faculty of Computer & Information Sciences, Ain Shams
University, Egypt

Prof. Dr. Mohamed Hashem Abd El Aziz

Vise Dean for Student Affairs Faculty of Computer & Information Sciences,
Ain Shams University, Egypt

Prof. Dr. Grigore ROȘU

Associate Professor in the Department of Computer Science, University of
Illinois at Urbana-Champaign, USA

Acknowledgments

This work would not have been completed without the support of many people to whom I wish to express my deep gratitude.

My deepest gratitude goes to my supervisors, Prof. Mohamed Essam Khalifa and Prof. Mohamed Hashim, who were abundantly supportive and helpful, and who were always there for me with their guidance treating me like a daughter.

I will also always be in debit to my supervisor Prof. Grigore Roşu for having me for 2 years in UIUC where he has continuously guided me with his knowledge and with his valuable support, not to mention the generous hospitality of his and his family throughout my visit. Also, I would like to thank my colleagues in the MOP team, especially Patrick Meredith and Dongyun Jin for their cooperation and for their notes on some parts of this work.

I would also like to convey thanks to the Faculty of Computer and Information Systems and especially the department of Information Systems, for supporting my joint research grant during my 2 years visit to the United States.

I also want to thank Prof. Dr. Mohamed Kouta who passed away a couple of months before I returned to Egypt. I consider him my God Father, I wish he lived to see that moment. Goes to him my deep gratitude and may his soul rest in peace.

Also, i would like to express my love and gratitude to my beloved parents, Prof. Mohamed Nabil and Prof. Asmaa AbdelMonium, for being there for me all the way through, and for giving me a model to look at on both the academic and personal levels.

Also I would like to thank my brothers, Dr. Haitham Hussein and Eng. Shady Hussein, for being the best brothers one could ever dream of having.

A special thanks goes to my daughter, Fairouz Otefy, for being there with me during my research and for being the joy of my life. I believe that she was the motivation for me to make it through. I also would like to thank my husband, Eng. Essam Otefy, for doing the best he could.

Finally, I would like to thank God for all his blessings, generosity, forgiveness, for giving me chances, and for being there for me when it was absolutely hard to go on.

Abstract

For a long time, *runtime verification RV* has been used to formalize system's properties for testing and verification, but (as far as our knowledge is) it hasn't been used to enforce security properties about systems. On the other hand there exists tailored runtime security enforcement systems (usually called *execution monitor*-based systems abbreviated as EM-based systems) that are specialized to specify and enforce only security properties during runtime.

In this thesis we do the following:

First, since runtime verification and execution monitors are completely two separate research disciplines, we carefully try to understand features where both disciplines overlap and where they do not. We go one step further by actually using an RV system to do the job of an EM system. This required the addition of a new feature (around join point) to our RV system as well as providing ways for policy composition and ensuring correctness on the generated monitored code. Our point here is that with slight additions on RV-based systems, RV-based systems can be generic enough to include features of any EM-based system, while still enjoying other features that comes with RV-based systems.

Second, we present a formal model that models specification monitors (which are used to enforce system's properties) as *finite state transducers*. Our formal model intends to solve problem of conflicts among different monitors that are co-existing at the same time. We show how the formal model can be used to detect potential conflicts among different monitors and to resolve them.

Third, we extend JavaMOP language (the main generic runtime verification and monitoring system used throughout the thesis) to support extending of specifications from one another; where we supported the addition of new syntax to the JavaMOP language, created a parser with its extended abstract syntax tree. Such an extension allows JavaMOP users to overload and/or override elements of a super specification, i.e., local variables, events properties and handlers, written in JavaMOP. Also we introduced 2 new pointcuts to JavaMOP, *event pointcut* and the *handler pointcut*, to allow specifications to pointcut other specification's events as well as their handlers.

Finally, we measure the performance overhead of the instrumented programs using execution-monitors based systems and those instrumented using a runtime verification and monitoring system via specialized programs and also using the Da-Capo benchmark.

Generally JavaMOP showed relatively low performance overhead of the instrumented program as opposed to other EM-based systems. That is in addition to

its expressive power in expressing different complex security specifications and the ability to resolve potential conflicts among them after extending the JavaMOP language.

Keywords: Runtime Verification and Monitoring, MOP, Execution Monitors, Inlined Reference Monitors, Security Access Control Policies, Policies Conflicts Detection and Resolution, Finite State Transducers, Extending Specifications.

Contents

1	Introduction and Motivation	1
1.1	Problem Definition	2
1.2	Thesis Outline	3
2	Survey over Runtime Security-Enforcement Models	5
2.1	Basic Terms	6
2.1.1	Execution Trace	6
2.1.2	Security Policy	6
2.1.3	Safety Property	6
2.2	Execution Monitor Model	7
2.2.1	Basic Security Principles	7
2.2.2	Reference Monitor and Its Implementations	8
2.2.3	Execution Monitors	9
2.2.4	Security Automata	10
2.3	Edit Automata	11
2.3.1	Model of the Edit Automata	11
2.3.2	Edit Automata Formal Definition	12
2.4	Inlined Reference Monitors as Program Rewriting	14
2.4.1	Program Rewriting	14
2.4.2	Formal Definition of a Target	14
2.4.3	Modeling Program Rewriting to Enforce Security Policies	15
2.5	Revisiting Types of Enforceable Security Policies	16
2.6	Outline	16
3	Survey over Runtime Security-Enforcement Systems	17
3.1	Ideal Feature List	18
3.1.1	General Ideal IRM Features Sub-List	18
3.1.2	Policy Language Ideal Features Sub-List:	19
3.1.3	Rewriter Ideal Features Sub-List	20
3.2	Execution Monitors Systems	21
3.2.1	SASI	21
3.2.2	PoET	25
3.2.3	Naccio	28
3.2.4	Polymer	32

3.2.5	SPoX	36
3.3	Outline	39
4	Proposed Runtime Security Enforcement Mechanism Using JavaMOP	45
4.1	Runtime and Monitoring Verification Systems	46
4.2	Monitoring Oriented Programing (MOP) and JavaMOP Instance . .	46
4.2.1	JavaMOP Syntax	47
4.2.2	JavaMOP Language Features	50
4.2.3	JavaMOP Rewriter	52
4.3	Using JavaMOP as Inlined Reference Monitors	53
4.3.1	Access Control	54
4.3.2	Addition of the Around Join Point	54
4.3.3	Separation of Duties	58
4.3.4	SQL Injection	60
4.3.5	Wall Policies	60
4.4	Policy Composition	63
4.5	Relating JavaMOP with IRM Systems	66
4.5.1	JavaMOP Categorization	67
4.6	Outline	68
5	Proposed Meta-Policies Conflict Resolution Framework using Finite State Transducers	73
5.1	Introduction	74
5.1.1	Properties about Monitors Conflicts Resolutions	76
5.2	Finite State Transducers	77
5.2.1	Definition of Finite State Transducers (FST)	77
5.2.2	Sequential Transducers	78
5.2.3	More about FST	78
5.2.4	Closure Properties of FST	79
5.3	Modeling of Monitors as FST	80
5.3.1	Events Definition	80
5.3.2	Monitors Framework-Modeling Architecture	81
5.3.3	Modeling of a Program Monitor TM_{pgm}	81
5.3.4	Modeling Monitor Transducers in TM_{net} Monitor	83
5.3.5	Identifying Potential Conflicts	84
5.3.6	Conflict Resolution with TM Composition	84
5.3.7	Modeling Monitors Not in Conflict	86

5.3.8	Other Composition Usages	86
5.3.9	Example	88
5.4	Outline	88
6	Proposed JavaMOP Extension	91
6.1	Why Extending Specifications	91
6.2	Parser and Syntax	92
6.2.1	Extending Features & BNF Syntax	92
6.2.2	Process	94
6.2.3	LL Parsers and JavaCC	96
6.3	Implementation	97
6.3.1	Using Double-Dispatch Visitors	97
6.4	Features by Examples	97
6.4.1	Extending Multiple Specification	97
6.4.2	Extending Abstract Events and Local Monitors	98
6.4.3	Event Pointcut	99
6.4.4	Handler Pointcut and Overriding Handlers	100
6.4.5	Extending Parameters	100
6.5	Outline	100
7	Performance Measurements	109
7.1	JavaMOP Chinese Wall Performance	109
7.2	DaCapo Benchmark Performance Results	110
7.2.1	JavaMOP Performance Results on DaCapo Benchmark	111
7.2.2	JavaMOP Performance Results Against Polymer and SPoX	112
8	Conclusion and Future Work	117
8.1	Conclusion	117
8.2	Future Work	118
A	Extended JavaMOP Specification Examples	119
B	FST Monitor's Framework Symbols	129
	Bibliography	131
	List of Abbreviations	137

List of Figures

2.1	Approaches to reference monitor [7]	8
5.1	Target Transducer Monitor TM_{pgm}	82
5.2	Transducer Monitor for No System Calls Specification	88
5.3	File Network Wall TM	89
6.1	Creation of Extended Abstract Syntax Tree	95
6.2	Part of the Extended Abstract Syntax Tree Class Diagram.	102
7.1	Chinese Wall Performance Chart	111
7.2	JavaMOP Performance Chart on DaCapo Benchmark	115

List of Tables

3.1	SASI- General Features	21
3.2	SASI- SAL- Policy Specification Language Features	23
3.3	SASI- Rewriter Features	24
3.4	PoET- General Features	25
3.5	PoET- PSLange- Policy Specification Language Features	28
3.6	PoET- Rewriter Features	28
3.7	Naccio- General Features	28
3.8	Naccio- Policy Specification Language Features	32
3.9	Naccio- Rewriter Features	33
3.10	Polymer- General Features	33
3.11	Polymer- Policy Specification Language Features	34
3.12	Polymer- Rewriter Features	36
3.13	SPoX- General Features	36
3.14	SPoX- Policy Specification Language Features	38
3.15	SPoX- Rewriter Features	38
3.16	Overall Comparison Results	43
4.1	JavaMOP- General Features	47
4.2	JavaMOP- Policy Specification Language Features	52
4.3	JavaMOP- Rewriter Supported Features	53
4.4	Combination of behavior of two policies, A and B	64
7.1	ChineseWall Performance Results.	110
7.2	JavaMOP Performance Results on DaCapo Benchmark.	112
7.3	Instrumented Events Statistics on DaCapo	113
7.4	DaCapo Performance Results for JavaMOP, SPoX and Polymer. . .	114

Listings

3.1	SAL: No message sent after reading a file	22
3.2	PSLang: Allow 10 windows opened at most at the same time	27
3.3	Naccio Resource Specification of a File Resource	30
3.4	Naccio Policy- Limit Write	31
3.5	Naccio Platform Interface	31
3.6	Polymer: File Writer Interface File	35
3.7	Polymer: Part of Check Mail Attachments Policy	40
3.8	Polymer: Action File	41
3.9	SPoX:No Network Send after Read	42
4.1	JavaMOP Syntax	48
4.2	JavaMOP Specification (Safe Lock)	49
4.3	Disable System Calls	54
4.4	Around Join Point Syntax	55
4.5	Restrict System Calls	56
4.6	Restrict Executable File Creation	57
4.7	Restrict Hidden File Access	58
4.8	Separation of Duties	59
4.9	SQL Injection	61
4.10	The Chinese Wall Policy in JavaMOP	70
4.11	File Network Wall	71
4.12	Conflict Manager	71
4.13	Parametric Conflict Manager	72
6.1	JavaMOP Extended Specification Syntax using BNF Grammar	93
6.2	Conditional File Network Wall	98
6.3	Audit Abstract Specification	103
6.4	Restrict File Creation	104
6.5	DisableNonMailNetworkUses event pointcut and event definition overloading	105
6.6	DisableNonMailNetwork- Version 2 Uses binded events pointcut	106
6.7	Disable Non Mail Network	106
6.8	Disable Mail Connections for Guest Users	107
A.1	Conditional File Network Wall	120
A.2	Audit Abstract Specification	121
A.3	Restrict File Creation	122