

# VU Research Portal

## Using information flow tracking to protect legacy binaries

Slowinska, J.M.

2012

### **document version**

Publisher's PDF, also known as Version of record

[Link to publication in VU Research Portal](#)

### **citation for published version (APA)**

Slowinska, J. M. (2012). *Using information flow tracking to protect legacy binaries*. [PhD-Thesis - Research and graduation internal, Vrije Universiteit Amsterdam].

### **General rights**

Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

- Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
- You may not further distribute the material or use it for any profit-making activity or commercial gain
- You may freely distribute the URL identifying the publication in the public portal

### **Take down policy**

If you believe that this document breaches copyright please contact us providing details, and we will remove access to the work immediately and investigate your claim.

### **E-mail address:**

[vuresearchportal.ub@vu.nl](mailto:vuresearchportal.ub@vu.nl)



# Contents

<b>Acknowledgements</b>	<b>vii</b>
<b>Contents</b>	<b>ix</b>
<b>List of Figures</b>	<b>xiii</b>
<b>Publications</b>	<b>xv</b>
<b>1 Introduction</b>	<b>1</b>
1.1 The Problem . . . . .	2
1.2 Goals . . . . .	4
1.3 Contributions . . . . .	5
1.4 Organisation of the Thesis . . . . .	6
<b>2 Background - Memory Corruptions</b>	<b>7</b>
2.1 Buffer Overflows . . . . .	7
2.1.1 Buffer Overflows – the Mechanism . . . . .	8
2.1.2 Control-diverting versus Non-control-diverting Attacks . . . . .	8
2.1.3 Stack Smashing Attacks . . . . .	10
2.1.4 Other Attacks Corrupting Return Addresses . . . . .	11
2.1.5 Pointer Subterfuge . . . . .	12
2.1.6 Heap Smashing . . . . .	13
2.1.7 Non-control-diverting Attacks . . . . .	18
2.2 Format String Attacks . . . . .	22
2.3 Conclusions . . . . .	23
<b>I Dynamic Taint Analysis for Attack Detection: Usefulness and Limitations</b>	<b>25</b>
<b>3 Dynamic Taint Analysis for Attack Detection</b>	<b>29</b>
3.1 Introduction . . . . .	29
3.2 Basic Dynamic Taint Analysis . . . . .	30

3.2.1	Definition and Policies . . . . .	30
3.2.2	Popular Solutions . . . . .	32
3.2.3	Limitations . . . . .	36
3.2.4	Extensions to Basic Dynamic Taint Analysis . . . . .	38
3.3	Evaluating the Practicality of Pointer Tainting . . . . .	39
3.3.1	Definition . . . . .	41
3.3.2	Test Environment . . . . .	42
3.3.3	Problems with Pointer Tainting: Experimental Results . . . . .	44
3.3.4	Problems with Pointer Tainting: Analysis . . . . .	45
3.3.5	Containment for LPT <i>and</i> FPT: EBP/ESP Protection . . . . .	50
3.3.6	LPT-specific Containment Techniques . . . . .	51
3.3.7	FPT-specific Containment Techniques . . . . .	54
3.3.8	How Bad Are Things? . . . . .	57
3.3.9	Other Pointer Tainting Approaches . . . . .	59
3.3.10	Conclusions . . . . .	60
<b>4</b>	<b>Prospector: Buffer Overflows Analysis</b>	<b>61</b>
4.1	Introduction . . . . .	61
4.2	Related Work . . . . .	64
4.3	Attacks and Factors Complicating the Analysis . . . . .	65
4.4	Design . . . . .	67
4.4.1	Argos . . . . .	68
4.4.2	Dealing with Advanced Heap Overflows . . . . .	68
4.4.3	Dealing with Malformed Messages in Heap Overflows . . . . .	69
4.4.4	Age Stamps . . . . .	69
4.4.5	Additional Indicators . . . . .	70
4.4.6	Example Explained . . . . .	70
4.4.7	Formal Specification of the Properties of Tainted Data and Gaps . . . . .	72
4.4.8	Analysis . . . . .	73
4.4.9	Signature Generation . . . . .	76
4.4.10	Double-free Errors . . . . .	78
4.5	Implementation Details . . . . .	79
4.5.1	Monitoring Process Switches from the Hardware . . . . .	79
4.5.2	Heap Protection . . . . .	80
4.5.3	Prospector Tagging . . . . .	81
4.5.4	AgeStamp Wrapping . . . . .	81
4.5.5	Stale Red Markers . . . . .	82
4.6	Evaluation . . . . .	82
4.6.1	Effectiveness . . . . .	83
4.6.2	Analysis Performance . . . . .	85
4.6.3	Signatures Checking Performance . . . . .	86
4.7	Discussion . . . . .	88



4.8	Conclusions . . . . .	89
<b>5</b>	<b>Hassle: How to Deal with Encrypted Channels?</b>	<b>91</b>
5.1	Introduction . . . . .	91
5.2	Architecture . . . . .	92
5.2.1	Retainting . . . . .	94
5.2.2	Interposition Details . . . . .	96
5.3	Signature Generation . . . . .	97
5.4	Filters . . . . .	97
5.5	Results . . . . .	98
5.6	Related Work . . . . .	99
5.7	Conclusions . . . . .	100

## **II Beyond Dynamic Taint Analysis: Protecting Legacy Binaries against Memory Corruption Attacks** **101**

<b>6</b>	<b>Howard: Dynamic Data Structure Excavation</b>	<b>107</b>
6.1	Introduction . . . . .	107
6.2	Related Work . . . . .	109
6.3	Recovery by Access Patterns: Challenges . . . . .	112
6.4	Howard Design and Implementation . . . . .	114
6.4.1	Function Call Stack . . . . .	114
6.4.2	Pointer Tracking . . . . .	115
6.4.3	Multiple Base Pointers . . . . .	116
6.4.4	Array Detection . . . . .	117
6.4.5	Final Mapping . . . . .	121
6.4.6	Partial Recovery of Semantics . . . . .	122
6.5	Comments and Limitations . . . . .	122
6.6	Optimisation and Obfuscation Techniques . . . . .	124
6.6.1	Compiler Optimisations . . . . .	124
6.6.2	Obfuscation Techniques . . . . .	126
6.6.3	Ongoing Work . . . . .	127
6.7	Applications . . . . .	129
6.7.1	Binary Analysis with Reconstructed Symbols . . . . .	129
6.7.2	Recovering High-level Data Structures . . . . .	131
6.8	Evaluation . . . . .	133
6.9	Conclusions . . . . .	137
<b>7</b>	<b>BodyArmour for Binaries</b>	<b>139</b>
7.1	Introduction . . . . .	139
7.2	What to Protect: Arrays and Array Accesses . . . . .	142
7.2.1	Extracting Arrays and Data Structures . . . . .	142
7.2.2	Array Accesses and Pointer Manipulations . . . . .	144

7.3	Code-coverage and Problems to be Expected . . . . .	144
7.4	<i>BA-objects mode</i> : an Object-level Protection . . . . .	145
7.4.1	What is Permissible? What is not? . . . . .	145
7.4.2	Colours . . . . .	146
7.4.3	Protection by Colour Matching . . . . .	146
7.4.4	Pointer Subtraction: What if the Code is Colour Blind? . . . . .	146
7.4.5	Stale Colours and Extra Measures to Rule out False Positives . . . . .	147
7.4.6	Why False Positives are not Possible . . . . .	147
7.5	<i>BA-fields mode</i> : a Colourful Armour . . . . .	149
7.5.1	What is Permissible? What is not? . . . . .	149
7.5.2	Shaded Colours . . . . .	150
7.5.3	Protection by Colour Matching . . . . .	151
7.5.4	What if the Code is Colour Blind? . . . . .	152
7.5.5	Why We do Not See False Positives in Practice . . . . .	153
7.5.6	Are False Positives Still Possible? . . . . .	154
7.6	Efficient Implementation . . . . .	155
7.6.1	Updated Layout of ELF Binary . . . . .	156
7.6.2	Instrumentation Code . . . . .	156
7.7	Evaluation . . . . .	157
7.8	Related Work . . . . .	159
7.9	Discussion . . . . .	161
7.10	Conclusions . . . . .	163
<b>8</b>	<b>Conclusions</b>	<b>165</b>
	<b>References</b>	<b>169</b>
	<b>Summary</b>	<b>189</b>
	<b>Samenvatting</b>	<b>193</b>