Surreptitious Software

Obfuscation, Watermarking, and Tamperproofing for Software Protection

Christian Collberg
Jasvir Nagra
Contents

Preface xv
About the Authors xxv
Acknowledgments xxvii

1 What Is Surreptitious Software? 1
1.1 Setting the Scene 1
1.2 Attack and Defense 6
1.3 Program Analysis 7
  1.3.1 A Simple Reverse Engineering Example 9
1.4 Code Obfuscation 13
  1.4.1 Applications of Code Obfuscation 16
  1.4.2 Obfuscating Transformations 20
  1.4.3 Black Hat Code Obfuscation 26
1.5 Tamperproofing 32
  1.5.1 Applications of Tamperproofing 33
  1.5.2 An Example 35
1.6 Software Watermarking 36
  1.6.1 An Example 38
  1.6.2 Attacks on Watermarking Systems 41
1.7 Software Similarity 43
  1.7.1 Plagiarism 43
  1.7.2 Software Forensics 44
  1.7.3 Birthmarking 45
  1.7.4 A Birthmarking Example 47
1.8 Hardware-Based Protection Techniques 49
  1.8.1 Distribution with Physical Token 49
  1.8.2 Tying the Program to the CPU 50
  1.8.3 Ensuring Safe Execution Environment 51
  1.8.4 Encrypted Execution 52
  1.8.5 Physical Barriers 54
1.9 Discussion 55
  1.9.1 Reasons to Use Software Protection . . 55
  1.9.2 . . and Reasons Not To 56
  1.9.3 So Which Algorithms Should I Use? 57

1.10 Notation 58

2 Methods of Attack and Defense 59
  2.1 Attack Strategies 60
    2.1.1 A Prototypical Cracking Target 61
    2.1.2 What's the Adversary's Motivation? 63
    2.1.3 What Does the Adversary Get to Crack? 65
    2.1.4 What's the Adversary's Attack Methodology? 68
    2.1.5 What Tools Does the Adversary Use? 72
    2.1.6 What Techniques Does the Adversary Use? 72
    2.1.7 Discussion 83

  2.2 Defense Strategies 86
    2.2.1 Notation 87
    2.2.2 The cover Primitive 90
    2.2.3 The duplicate Primitive 93
    2.2.4 The split and merge Primitives 96
    2.2.5 The reorder Primitive 100
    2.2.6 The map Primitive 101
    2.2.7 The indirect Primitive 104
    2.2.8 The mimic Primitive 106
    2.2.9 The advertise Primitive 108
    2.2.10 The detect-respond Primitive 110
    2.2.11 The dynamic Primitive 112
    2.2.12 Discussion 113

  2.3 Discussion 114
    2.3.1 What Do We Need from Attack and Defense Models? 114
    2.3.2 How Do We Use the Models to Devise Algorithms? 115

3 Program Analysis 117
  3.1 Static Analysis 118
    3.1.1 Control Flow Analysis 119
    3.1.2 Data Flow Analysis 127
    3.1.3 Data Dependence Analysis 132
    3.1.4 Alias Analysis 134
    3.1.5 Slicing 141
    3.1.6 Abstract Interpretation 143

  3.2 Dynamic Analysis 145
    3.2.1 Debugging 146
    3.2.2 Profiling 161
3.2.3 Tracing 163
3.2.4 Emulation 168

3.3 Reconstituting Source 170
3.3.1 Disassembly 172
3.3.2 Decompilation 180

3.4 Pragmatic Analysis 190
3.4.1 Style Metrics 191
3.4.2 Software Complexity Metrics 193
3.4.3 Software Visualization 195

3.5 Discussion 198

4 Code Obfuscation 201

4.1 Semantics-Preserving Obfuscating Transformations 202
4.1.1 Algorithm OBFCF: Diversifying Transformations 203
4.1.2 Algorithm OBFTP: Identifier Renaming 209
4.1.3 Obfuscation Executives 212

4.2 Definitions 217
4.2.1 Potent Obfuscating Transformations 219
4.2.2 Efficient Obfuscating Transformations 222
4.2.3 Stealth 222
4.2.4 Other Definitions 224

4.3 Complicating Control Flow 225
4.3.1 Opaque Expressions 225
4.3.2 Algorithm OBFWHKD: Control-Flow Flattening 226
4.3.3 Introducing Aliasing 229
4.3.4 Algorithm OBFCTJ_bogus: Inserting Bogus Control Flow 235
4.3.5 Algorithm OBFLDK: Jumps Through Branch Functions 239
4.3.6 Attacks 242

4.4 Opaque Predicates 246
4.4.1 Algorithm OBFCTJ_pointer: Opaque Predicates from Pointer Aliasing 247
4.4.2 OBFWHKD_opaque: Opaque Values from Array Aliasing 250
4.4.3 Algorithm OBFCTJ_thread: Opaque Predicates from Concurrency 251
4.4.4 Breaking Opaque Predicates 253

4.5 Data Encodings 258
4.5.1 Encoding Integers 261
4.5.2 Encoding Booleans 266
4.5.3 Encoding Literal Data 269
4.5.4 Encoding Arrays 272

4.6 Breaking Abstractions 277
4.6.1 Algorithm OBFWCsig: Merging Function Signatures 277
4.6.2 Algorithm OBFCTJ_class: Splitting and Merging Classes 279
4.6.3 Algorithm $\text{OB}F_{\text{DMRVSL}}$: Destroying High-Level Structures 281
4.6.4 Algorithm $\text{OBFAJV}$: Modifying Instruction Encodings 293

4.7 Discussion 298

5 Obfuscation Theory 301
5.1 Definitions 304
5.2 Provably Secure Obfuscation: Possible or Impossible? 307
  5.2.1 Turing's Halting Problem 308
  5.2.2 Algorithm $\text{REAA}$: De-obfuscating Programs 311
5.3 Provably Secure Obfuscation: It’s Possible (Sometimes)! 313
  5.3.1 Algorithm $\text{OBFLBS}$: Obfuscating with Point Functions 314
  5.3.2 Algorithm $\text{OBFNS}$: Obfuscating Databases 322
  5.3.3 Algorithm $\text{OBFPP}$: Homomorphic Encryption 324
  5.3.4 Algorithm $\text{OBFCEJO}$: Whitebox DES 329
5.4 Provably Secure Obfuscation: It’s Impossible (Sometimes)? 335
  5.4.1 A General Obfuscator 336
  5.4.2 Obfuscating Learnable Functions 340
  5.4.3 Proving that Obfuscation Is Impossible 341
  5.4.4 Discussion 343
5.5 Provably Secure Obfuscation: Can It Be Saved? 344
  5.5.1 Overcoming Impossibility 346
  5.5.2 Definitions Revisited: Make Obfuscation Interactive 346
  5.5.3 Definition Revisited: Make Obfuscation Non-Semantics Preserving 349
5.6 Discussion 354

6 Dynamic Obfuscation 357
6.1 Definitions 360
6.2 Moving Code Around 362
  6.2.1 Algorithm $\text{OBFKMN}M$: Replacing Instructions 362
  6.2.2 $\text{OBFAG}_{\text{swap}}$: Self-Modifying State Machine 366
  6.2.3 $\text{OBFMAMDSB}$: Dynamic Code Merging 376
6.3 Encryption 383
  6.3.1 $\text{OBFCKSP}$: Code as Key Material 385
  6.3.2 $\text{OBFAG}_{\text{crypt}}$: Combining Self-Modification and Encryption 392
6.4 Discussion 398

7 Software Tamperproofing 401
7.1 Definitions 405
  7.1.1 Checking for Tampering 406
7.1.2 Responding to Tampering 410
7.1.3 System Design 410
7.2 Introspection 412
7.2.1 Algorithm TPCA: Checker Network 414
7.2.2 Generating Hash Functions 418
7.2.3 Algorithm TPHMST: Hiding Hash Values 423
7.2.4 The Skype Obfuscated Protocol 431
7.2.5 Algorithm REWOS: Attacking Self-Hashing Algorithms 435
7.2.6 Discussion 439
7.3 Algorithm TPTCJ: Response Mechanisms 440
7.4 State Inspection 444
7.4.1 Algorithm TPCVCPSJ: Oblivious Hash Functions 447
7.4.2 Algorithm TPJIV: Overlapping Instructions 450
7.5 Remote Tamperproofing 453
7.5.1 Distributed Check and Respond 454
7.5.2 Solution Strategies 454
7.5.3 Algorithm TPZG: Slicing Functions 455
7.5.4 Algorithm TPJSPDK: Measuring Remote Hardware 459
7.5.5 TPCNS: Continuous Replacement 462
7.6 Discussion 464

8 Software Watermarking 467
8.1 History and Applications 468
8.1.1 Applications 468
8.1.2 Embedding a Mark in Audio 472
8.1.3 Embedding a Mark in an Image 474
8.1.4 Embedding a Mark in Natural-Language Text 475
8.2 Watermarking Software 478
8.3 Definitions 480
8.3.1 Watermark Credibility 482
8.3.2 Attacks 484
8.3.3 Watermarking vs. Fingerprinting 485
8.4 Watermarking by Permutation 486
8.4.1 Algorithm WMMDM: Reordering Basic Blocks 488
8.4.2 Renumbering 490
8.4.3 Algorithm WMQP: Improving Credibility 491
8.5 Tamperproofing Watermarks 494
8.5.1 Algorithm WMMC: Embedding Media Watermarks 495
8.6 Improving Resilience 498
8.6.1 Algorithm WMSHKQ: Statistical Watermarking 498
8.7 Improving Stealth 505
8.7.1 Algorithm \textsc{wmMIMIT}: Mapping Instructions 505
8.7.2 Algorithm \textsc{wmVVS}: Watermarks in CFGs 506
8.7.3 Algorithm \textsc{wmCC}: Abstract Interpretation 516
8.8 Steganographic Embeddings 522
8.8.1 Algorithm \textsc{wmASB}: The Compiler as Embedder 523
8.9 Splitting Watermark Integers 526
8.9.1 Splitting a Large Mark into Small Pieces 527
8.9.2 Redundant Watermark Pieces 528
8.9.3 Sparse Codes for Increased Credibility 531
8.10 Graph Codecs 533
8.10.1 Oriented Parent-Pointer Tree 534
8.10.2 Radix Graphs 534
8.10.3 Permutation Graphs 535
8.10.4 Planted Plane Cubic Trees 536
8.10.5 Reducible Permutation Graphs 536
8.11 Discussion 537
8.11.1 Embedding Techniques 539
8.11.2 Attack Models 539

9 Dynamic Watermarking 541
9.1 Algorithm \textsc{wmCT}: Exploiting Aliasing 546
9.1.1 A Simple Example 547
9.1.2 Recognition Problems 549
9.1.3 Increasing Bitrate 551
9.1.4 Increasing Resilience to Attack 557
9.1.5 Increasing Stealth 561
9.1.6 Discussion 564
9.2 Algorithm \textsc{wmNT}: Exploiting Parallelism 565
9.2.1 Embedding Watermarking Widgets 569
9.2.2 Embedding Example 574
9.2.3 Recognition 577
9.2.4 Avoiding Pattern-Matching Attacks 579
9.2.5 Tamperproofing Widgets 580
9.2.6 Discussion 581
9.3 Algorithm \textsc{wmCCDKHLS}_{paths}: Expanding Execution Paths 583
9.3.1 Encoding and Embedding 584
9.3.2 Recognition 590
9.3.3 Discussion 591
9.4 Algorithm \textsc{wmCCDKHLS}_{hf}: Tamperproofing Execution Paths 592
9.4.1 Embedding 593
9.4.2 Recognition 595
9.4.3 Tamperproofing the Branches 596
9.4.4 Discussion 597
9.5 Discussion 598

10 Software Similarity Analysis 601

10.1 Applications 602
10.1.1 Clone Detection 603
10.1.2 Software Forensics 605
10.1.3 Plagiarism Detection 608
10.1.4 Birthmark Detection 610
10.2 Definitions 611
10.2.1 Similarity Measures 612
10.3 $k$-gram-Based Analysis 616
10.3.1 $SSWA_{WINNOW}$: Selecting $k$-gram Hashes 616
10.3.2 $SSWA_{MOSS}$: Software Plagiarism Detection 619
10.3.3 $SSM_{kgram}$: $k$-gram Java Bytecode Birthmarks 623
10.4 API-Based Analysis 625
10.4.1 $SSTNMM$: Object-Oriented Birthmarks 626
10.4.2 $STONMM$: Dynamic Function Call Birthmarks 629
10.4.3 $SSDL$: Dynamic $k$-gram API Birthmarks 630
10.5 Tree-Based Analysis 631
10.5.1 $SEFM$: AST-Based Clone Detection 631
10.6 Graph-Based Analysis 635
10.6.1 $SSKH$: PDG-Based Clone Detection 636
10.6.2 $SLCHY$: PDG-Based Plagiarism Detection 640
10.6.3 $SSM_{wpp}$: Dynamic Whole Program Birthmarks 641
10.7 Metrics-Based Analysis 644
10.7.1 $SSKK$: Metrics-Based Clone Detection 645
10.7.2 $SLM$: Metrics-Based Authorship Analysis 646
10.8 Discussion 652

11 Hardware for Protecting Software 655

11.1 Anti-Piracy by Physical Distribution 657
11.1.1 Distribution Disk Protection 658
11.1.2 Dongs and Tokens 664
11.2 Authenticated Boot Using a Trusted Platform Module 670
11.2.1 Trusted Boot 671
11.2.2 Taking Measurements 673
11.2.3 The TPM 676
11.2.4 The Challenge 677