Trusted Computing Platforms: Design and Applications

TRUSTED COMPUTING PLATFORMS: DESIGN AND APPLICATIONS

SEAN W. SMITH
Department of Computer Science
Dartmouth College
Hanover, New Hampshire USA

Springer

eBook ISBN: 0-387-23917-0
Print ISBN: 0-387-23916-2

©2005 Springer Science + Business Media, Inc.

Print ©2005 Springer Science + Business Media, Inc.
Boston

All rights reserved

No part of this eBook may be reproduced or transmitted in any form or by any means, electronic, mechanical, recording, or otherwise, without written consent from the Publisher

Created in the United States of America

Visit Springer's eBookstore at: http://ebooks.springerlink.com
and the Springer Global Website Online at: http://www.springeronline.com

Contents

List of Figures xiii

List of Tables xv

Preface xvii

Acknowledgments xix

1. **INTRODUCTION** **1**
   - 1.1 Trust and Computing 2
   - 1.2 Instantiations 2
   - 1.3 Design and Applications 5
   - 1.4 Progression 7

2. **MOTIVATING SCENARIOS** **9**
   - 2.1 Properties 9
   - 2.2 Basic Usage 10
   - 2.3 Examples of Basic Usage 12
   - 2.4 Position and Interests 14
   - 2.5 Examples of Positioning 15
   - 2.6 The Idealogical Debate 18
   - 2.7 Further Reading 18

3. **ATTACKS** **19**
   - 3.1 Physical Attack 21
     - 3.1.1 No Armor 22
     - 3.1.2 Single Chip Devices 23
     - 3.1.3 Multi-chip Devices 23
   - 3.2 Software Attacks 24
     - 3.2.1 Buffer Overflow 25

3.2.2 Unexpected Input 25
3.2.3 Interpretation Mismatches 26
3.2.4 Time-of-check vs Time-of-use 27
3.2.5 Atomicity 28
3.2.6 Design Flaws 29
3.3 Side-channel Analysis 30
3.3.1 Timing Attacks 30
3.3.2 Power Attacks 33
3.3.3 Other Avenues 34
3.4 Undocumented Functionality 35
3.4.1 Example: Microcontroller Memory 36
3.4.2 Example: FLASH Memory 37
3.4.3 Example: CPU Privileges 38
3.5 Erasing Data 38
3.6 System Context 39
3.7 Defensive Strategy 41
3.7.1 Tamper Evidence 41
3.7.2 Tamper Resistance 41
3.7.3 Tamper Detection 41
3.7.4 Tamper Response 42
3.7.5 Operating Envelope 42
3.8 Further Reading 42
4. FOUNDATIONS 43
4.1 Applications and Integration 43
4.1.1 Kent 44
4.1.2 Abyss 44
4.1.3 Citadel 45
4.1.4 Dyad 46
4.2 Architectures 48
4.2.1 Physical Security 48
4.2.2 Hardware and Software 49
4.3 Booting 50
4.4 The Defense Community 52
4.5 Further Reading 52

5. DESIGN CHALLENGES **55**
5.1 Context 55
5.1.1 Personal 55
5.1.2 Commercial 56
5.2 Obstacles 57
5.2.1 Hardware 57
5.2.2 Software 59
5.3 Requirements 63
5.3.1 Commercial Requirements 63
5.3.2 Security Requirements 64
5.3.3 Authenticated Execution 66
5.4 Technology Decisions 67
5.5 Further Reading 71
6. PLATFORM ARCHITECTURE **73**
6.1 Overview 73
6.1.1 Security Architecture 74
6.2 Erasing Secrets 75
6.2.1 Penetration Resistance and Detection 76
6.2.2 Tamper Response 76
6.2.3 Other Physical Attacks 77
6.3 The Source of Secrets 78
6.3.1 Factory Initialization 78
6.3.2 Field Operations 79
6.3.3 Trusting the Manufacturer 81
6.4 Software Threats 81
6.4.1 Software Threat Model 82
6.4.2 Hardware Access Locks 82
6.4.3 Privacy and Integrity of Secrets 85
6.5 Code Integrity 85
6.5.1 Loading and Cryptography 86
6.5.2 Protection against Malice 86
6.5.3 Protection against Reburn Failure 87
6.5.4 Protection against Storage Errors 88
6.5.5 Secure Bootstrapping 89
6.6 Code Loading 90
6.6.1 Authorities 91
6.6.2 Authenticating the Authorities 92

6.6.3 Ownership 92
6.6.4 Ordinary Loading 93
6.6.5 Emergency Loading 96
6.7 Putting it All Together 97
6.8 What's Next 99
6.9 Further Reading 99

7. OUTBOUND AUTHENTICATION 101

7.1 Problem 101
7.1.1 The Basic Problem 102
7.1.2 Authentication Approach 102
7.1.3 User and Developer Scenarios 103
7.1.4 On-Platform Entities 104
7.1.5 Secret Retention 104
7.1.6 Authentication Scenarios 105
7.1.7 Internal Certification 107
7.2 Theory 108
7.2.1 What the Entity Says 109
7.2.2 What the Relying Party Concludes 109
7.2.3 Dependency 110
7.2.4 Soundness 111
7.2.5 Completeness 112
7.2.6 Achieving Both Soundness and Completeness 112
7.2.7 Design Implications 113
7.3 Design and Implementation 114
7.3.1 Layer Separation 115
7.3.2 The Code-Loading Code 115
7.3.3 The OA Manager 116
7.3.4 Naming 119
7.3.5 Summary 119
7.3.6 Implementation 120
7.4 Further Reading 121

8. VALIDATION 123

8.1 The Validation Process 124
8.1.1 Evolution 124
8.1.2 FIPS 140-1 125
8.1.3 The Process 126
8.2 Validation Strategy 126

8.3 Formalizing Security Properties 129
8.3.1 Building Blocks 130
8.3.2 Easy Invariants 131
8.3.3 Controlling Code 131
8.3.4 Keeping Secrets 132
8.4 Formal Verification 134
8.5 Other Validation Tasks 136
8.6 Reflection 138
8.7 Further Reading 139

9. APPLICATION CASE STUDIES 141
9.1 Basic Building Blocks 141
9.2 Hardened Web Servers 142
9.2.1 The Problem 142
9.2.2 Using a TCP 144
9.2.3 Implementation Experience 149
9.3 Rights Management for Big Brother's Computer 152
9.3.1 The Problem 152
9.3.2 Using a TCP 153
9.3.3 Implementation Experience 154
9.4 Private Information 155
9.4.1 The Problem 155
9.4.2 Using a TCP: Initial View 157
9.4.3 Implementation Experience 158
9.4.4 Using Oblivious Circuits 160
9.4.5 Reducing TCP Memory Requirements 163
9.4.6 Adding the Ability to Update 165
9.5 Other Projects 167
9.5.1 Postal Meters 167
9.5.2 Kerberos KDC 167
9.5.3 Mobile Agents 167
9.5.4 Auctions 167
9.5.5 Marianas 168
9.5.6 Trusted S/MIME Gateways 169
9.5.7 Grid Tools 169
9.6 Lessons Learned 170
9.7 Further Reading 171

10. **TCPA/TCG** **173**
10.1 Basic Structure 175
10.2 Outbound Authentication 178
10.3 Physical Attacks 179
10.4 Applications 180
10.5 Experimentation 180
10.6 TPM 1.2 Changes 181
10.7 Further Reading 181

11. **EXPERIMENTING WITH TCPA/TCG** **183**
11.1 Desired Properties 184
11.2 The Lifetime Mismatch 184
11.3 Architecture 185
11.4 Implementation Experience 189
11.5 Application: Hardened Apache 190
11.6 Application: OpenCA 191
11.7 Application: Compartmented Attestation 193
11.8 Further Reading 194

12. **NEW HORIZONS** **195**
12.1 Privilege Architectures 195
12.2 Hardware Research 197
12.2.1 XOM 197
12.2.2 MIT AEGIS 198
12.2.3 Cerium 199
12.2.4 Virtual Secure Coprocessing 199
12.2.5 Virtual Machine Monitors 199
12.2.6 Others 200
12.3 Software Research 201
12.3.1 Software-based Attestation 202
12.3.2 Hiding in Plain Sight 202
12.4 Current Industrial Platforms 203
12.4.1 Crypto Coprocessors and Tokens 203
12.4.2 Execution Protection 203
12.4.3 Capability-based Machines 204
12.5 Looming Industry Platforms 204
12.5.1 LaGrande 204

12.5.2 TrustZone 206
12.5.3 NGSCB 206
12.6 Secure Coprocessing Revisited 208
12.7 Further Reading 209

Glossary 211

References 221

About the Author 235

Index 237

List of Figures

1.1 Secure coprocessing application structure 3

5.1 The basic hardware architecture. 68

5.2 The basic software architecture. 69

6.1 The authority tree. 91

6.2 Contents of a layer. 93

6.3 Statespace for a layer. 93

6.4 Ordinary code-load command. 94

6.5 Countersignatures. 95

6.6 Authorization of code-load commands. 95

6.7 An emergency code-load command. 97

7.1 *Epochs* and *configurations.* 105

7.2 Replacing untrusted software with trusted software creates problems. 106

7.3 Replacing trusted software with untrusted software creates problems. 107

7.4 Sketch of the proof of our outbound authentication theorem. 113

7.5 When the code-loading layer updates itself. 116

7.6 Having the certifier outlive a code change creates problems. 117

7.7 Having the certifier outlive the certified can cause problems. 118

7.8 We regenerate certifier key pairs with each code change. 118

8.1 The formal verification process, as we envisioned it before we started. 128

8.2 The “safe control” invariant. 132

8.3 The “safe zeroization” invariant. 133

8.4 The formal verification process, as it actually happened. 135

8.5 Validation documentation tools. 136
9.1 Revising the SSL handshake to use a trusted co-server. 150
9.2 A switch 160
9.3 Oblivious shuffles with a Benes network 162
11.1 Flow of protection and trust in our TCPA/TCG-based platform. 188
12.1 The standard CPU privilege structure. 196
12.2 The revised CPU privilege structure. 197

List of Tables

6.1	Hardware ratchets protect secrets.	85
6.2	Hardware ratchets protect code.	87
9.1	Performance of an SSL server with a trusted co-server.	151
9.2	Slowdown caused by adding a trusted co-server.	151

Preface

We stand an exciting time in computer science. The long history of specialized research building and using security-enhanced hardware is now merging with mainstream computing platforms; what happens next is not certain but is bound to be interesting. This book tries to provide a roadmap.

A fundamental aspect of the current and emerging information infrastructure is distribution: multiple parties participate in this computation, and each may have different interests and motivations. Examining security in these distributed settings thus requires examining which platform is doing what computation—and which platforms a party must trust, to provide certain properties despite certain types of adversarial action, if that party is to have trust in overall computation. Securing distributed computation thus requires considering the trustworthiness of individual platforms, from the differing points of view of the different parties involved. We must also consider whether the various parties in fact trust this platform—and if they should, how it is that they know they should.

The foundation of computing is hardware: the actual platform—gates and wires—that stores and processes the bits. It is common practice to consider the standard computational resources—e.g., memory and CPU power—a platform can bring to a computational problem. In some settings, it is even common to think of how properties of the platform may contribute to more intangible overarching goals of a computation, such as fault tolerance. Eventually, we may start trying to change the building blocks–the fundamental hardware—in order to better suit the problem we are trying to solve.

Combining these two threads—the importance of trustworthiness in these Byzantine distributed settings, with the hardware foundations of computing platforms—gives rise to a number of questions. What are the right trustworthiness properties we need for individual platforms? What approaches can we try in the hardware and higher-level architectures to achieve these properties? Can we usefully exploit these trustworthiness properties in computing platforms for broader application security?

With the current wave of commercial and academic trusted computing architectures, these questions are timely. However, with a much longer history of secure coprocessing, secure boot, and other experimentation, these questions are not completely new. In this book, we will examine this big picture. We look at the depth of the field: what a trusted computing platform might provide, how one might build one, and what one might be done with one afterward. However, we also look at the depth of history: how these ideas have evolved and played out over the years, over a number of different real platforms—and how this evolution continues today.

I was drawn to this topic in part because I had the chance to help do some of the work that shaped this field. Along the way, I've enjoyed the privilege of working with a number of excellent researchers. Some of the work in this book was reported earlier in my papers [SW99, SPW98, Smi02, Smi01, MSWM03, Smi03, Smi04], as documented in the "Further Reading" sections. Some of my other papers expand on related topics [DPSL99, DLP$^+$ 01, SA98, SPWA99, JSM01, IS03b, SS01, IS03a, MSW$^+$ 04, MSMW03, IS04b, IS04a].

Acknowledgments

Besides being a technical monograph, this book also represents a personal research journey stretching over a decade.

I am not sure how to begin acknowledging all the friends and colleagues who assisted with this journey. To start with: I am grateful to Doug Tygar and Bennet Yee, for planting these seeds during my time at CMU and continuing with friendship and suggestions since; to Gary Christoph and Vance Faber at Los Alamos, for encouraging this work during my time there; and to Elaine Palmer at IBM Watson, whose drive saw the defunct Citadel project turn into a thriving research and product development effort. Steve Weingart and Vernon Austel deserve particular thanks for their collaborations with security architecture and formal modeling, respectively. Thanks are also due to the rest of the Watson team, including Dave Baukus, Ran Canetti, Suresh Chari, Joan Dyer, Bob Gezelter, Juan Gonzalez, Michel Hack, Jeff Kravitz, Mark Lindemann, Joe McArthur, Dennis Nagel, Ron Perez, Pankaj Rohatgi, Dave Safford, and David Toll; to the 4758 development teams in Vimercate, Charlotte, Poughkeepsie, and Lexington; and to Mike Matyas.

Since I left IBM, this journey has been helped by fruitful discussions with many colleagues, including Denise Anthony, Charles Antonelli, Dmitri Asonov, Dan Boneh, Ryan Cathecart, Dave Challener, Srini Devadas, John Erickson, Ed Feustel, Chris Hawblitzel, Peter Honeyman, Cynthia Irvine, Nao Itoi, Ruby Lee, Neal McBurnett, Dave Nicol, Adrian Perrig, Dawn Song, and Leendert van Doorn. In academia, research requires buying equipment and plane tickets and paying students; these tasks were supported in part by the Mellon Foundation, the NSF (CCR-0209144), AT&T/Internet2 and the Office for Domestic Preparedness, Department of Homeland Security (2000-DT-CX-K001).

Here at Dartmouth, the journey continued with the research efforts of students including Alex Barsamian, Mike Engle, Meredith Frost, Alex Iliev, Shan Jiang, Evan Knop, Rich MacDonald, John Marchesini, Kazuhiro Minami, Mindy Periera, Eric Smith, Josh Stabiner, Omen Wild, and Ling Yan. My colleagues in the Dartmouth PKI Lab and the Department of Computer Science also provided invaluable helpful discussion, and coffee too.

Dartmouth students Meredith Frost, Alex Iliev, John Marchesini, and Scout Sinclair provided even more assistance by reading and commenting on early versions of this manuscript.

Finally, I am grateful for the support and continual patience of my family.

Sean Smith
Hanover, New Hampshire
October 2004