



Building Secure Firmware

Armoring the Foundation of the Platform

Jiewen Yao
Vincent Zimmer

Apress®

Building Secure Firmware

**Armoring the Foundation
of the Platform**

**Jiewen Yao
Vincent Zimmer**

Apress®

Building Secure Firmware: Armoring the Foundation of the Platform

Jiewen Yao
Shanghai, China

Vincent Zimmer
Issaquah, WA, USA

ISBN-13 (pbk): 978-1-4842-6105-7
<https://doi.org/10.1007/978-1-4842-6106-4>

ISBN-13 (electronic): 978-1-4842-6106-4

Copyright © 2020 by Jiewen Yao and Vincent Zimmer

This work is subject to copyright. All rights are reserved by the Publisher, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, reuse of illustrations, recitation, broadcasting, reproduction on microfilms or in any other physical way, and transmission or information storage and retrieval, electronic adaptation, computer software, or by similar or dissimilar methodology now known or hereafter developed.

Trademarked names, logos, and images may appear in this book. Rather than use a trademark symbol with every occurrence of a trademarked name, logo, or image we use the names, logos, and images only in an editorial fashion and to the benefit of the trademark owner, with no intention of infringement of the trademark.

The use in this publication of trade names, trademarks, service marks, and similar terms, even if they are not identified as such, is not to be taken as an expression of opinion as to whether or not they are subject to proprietary rights.

While the advice and information in this book are believed to be true and accurate at the date of publication, neither the authors nor the editors nor the publisher can accept any legal responsibility for any errors or omissions that may be made. The publisher makes no warranty, express or implied, with respect to the material contained herein.

Managing Director, Apress Media LLC: Welmoed Spahr
Acquisitions Editor: Susan McDermott
Development Editor: Laura Berendson
Coordinating Editor: Jessica Vakili

Distributed to the book trade worldwide by Springer Science+Business Media New York, 233 Spring Street, 6th Floor, New York, NY 10013. Phone 1-800-SPRINGER, fax (201) 348-4505, e-mail orders-ny@springer-sbm.com, or visit www.springeronline.com. Apress Media, LLC is a California LLC and the sole member (owner) is Springer Science + Business Media Finance Inc (SSBM Finance Inc). SSBM Finance Inc is a **Delaware** corporation.

For information on translations, please e-mail booktranslations@springernature.com; for reprint, paperback, or audio rights, please e-mail bookpermissions@springernature.com.

Apress titles may be purchased in bulk for academic, corporate, or promotional use. eBook versions and licenses are also available for most titles. For more information, reference our Print and eBook Bulk Sales web page at <http://www.apress.com/bulk-sales>.

Any source code or other supplementary material referenced by the author in this book is available to readers on GitHub via the book's product page, located at www.apress.com/978-1-4842-6105-7. For more detailed information, please visit <http://www.apress.com/source-code>.

Printed on acid-free paper

*To my lovely wife, Wenjun Zeng. Thank you for your constant
encouragement and unconditional support!*

—Jiewen Yao

*To my parents; to my wife, Jan; and to my daughters, Ally and Zoe.
Your support and love are rays of sunshine that warm my days.*

—Vincent Zimmer

Table of Contents

About the Authors	xix
About the Technical Reviewer	xxi
Organization and What Is Covered	xxiii
Acknowledgments	xxv
Preface	xxvii
Foreword	xxix
Part I: Overview	1
Chapter 1: Introduction to Firmware	3
Similarity Between Firmware and Software	3
Distinction Between Firmware and Software	5
Introduction to Firmware Security	6
Firmware Resiliency	7
Firmware Measurement and Attestation	8
Secure Device Communication.....	8
Introduction to Variants of Host Firmware	9
Industry Standards.....	10
Boot Flow and Phased Handoff.....	11
Introduction to Non-host Firmware.....	13
Introduction to Device Firmware.....	14
Summary.....	14
References.....	15

TABLE OF CONTENTS

- Chapter 2: Proactive Firmware Security Development..... 17**
 - Requirement Phase..... 17
 - Security Requirements 17
 - Threat Model and Architecture Phase 18
 - Threat Model Analysis 18
 - Security Architecture and Design 31
 - Real Example..... 36
 - Development Phase 49
 - Secure Coding Practice 49
 - Security Unit Test..... 49
 - Security Code Review..... 49
 - Test Phase..... 55
 - Fuzzing 55
 - Static Code Analysis 56
 - Dynamic Code Analysis 56
 - Vulnerability Scan..... 57
 - Release and Maintenance Phase 57
 - Security Incidence Response 57
 - People Education 58
 - Before the Project Starts 58
 - After the Project Release..... 58
 - Fallacy and Pitfall..... 58
 - Security Technology..... 58
 - Security Process..... 59
 - Summary..... 60
 - References..... 61
- Part II: Security Architecture..... 65**
- Chapter 3: Firmware Resiliency: Protection 67**
 - Resiliency Building Block..... 67
 - Immutable ROM 69

Integrity	69
Confidentiality.....	69
Case Study.....	70
Attack and Mitigation	72
Updatable Firmware.....	73
Authenticated Update Mechanism	73
Case Study.....	74
Integrity Protection	93
Case Study.....	94
Non-bypassability.....	94
Attack and Mitigation	94
Configurable Data	108
Case Study.....	108
Attack and Mitigation	110
Summary.....	111
References.....	111
Chapter 4: Firmware Resiliency: Detection	115
Secure Boot.....	115
Detectability	115
Version.....	117
Policy Revocation	117
Non-bypassability.....	117
Additional Capability.....	118
Case Study.....	118
Attack and Mitigation	149
Secure Configuration	155
Detectability	156
Attack and Mitigation	157
Summary.....	158
References.....	158

TABLE OF CONTENTS

- Chapter 5: Firmware Resiliency: Recovery 163**
 - Image Recovery 163
 - RTRec Selection and Recovery Policy 163
 - Recovery Image Selection 165
 - Recovery Image Location 165
 - Case Study..... 166
 - Attack and Mitigation 177
 - Configuration Recovery..... 178
 - Recovery Configuration Selection 178
 - Attack and Mitigation 180
 - Watchdog 181
 - Summary..... 182
 - References 182

- Chapter 6: OS Resiliency 185**
 - Protection..... 186
 - Automated Update..... 186
 - Detection..... 186
 - Image Signing..... 186
 - Case Study..... 186
 - Recovery 193
 - Automated Recovery 193
 - Case Study..... 193
 - Summary..... 195
 - References 195

- Chapter 7: Trusted Boot 197**
 - Static Root-of-Trust for Measurement (SRTM)..... 198
 - Trusted Platform Module (TPM) 198
 - TPM Device Type..... 201
 - Measurement Report..... 202
 - Attestation 203

S3 Resume	206
Device Identifier Composition Engine (DICE)	207
Case Study.....	210
Attack and Mitigation	224
Dynamic Root-of-Trust for Measurement (DRTM).....	226
CPU-Based Dynamic Root-of-Trust.....	227
S3 Resume	229
DEC's Execution Environment.....	230
Hot Plug.....	231
Case Study.....	232
Attack and Mitigation	237
TCG Memory Overwrite.....	242
Secure MOR.....	244
MOR for Storage	244
Attack and Mitigation	245
TCG Physical Presence Configuration	246
TCG Storage	246
Summary.....	246
References.....	246
Chapter 8: Device Security	257
Device Firmware Resiliency.....	257
Secure Device Communication	258
Authentication and Measurement	258
Secure Communication Channel	259
Device Identifier Composition Engine (DICE)	260
Case Study.....	260
Attack and Mitigation	267
Device Attack Prevention	268
Device Identifier Data	268
Attack and Mitigation	269
Direct Memory Access (DMA).....	270

TABLE OF CONTENTS

- Case Study..... 275
- Attack and Mitigation 283
- Message Signaled Interrupt (MSI) 286
- Case Study..... 291
- Attack and Mitigation 297
- Server RAS (Reliability, Availability, and Serviceability)..... 300
- Case Study..... 301
- Attack and Mitigation 305
- Summary..... 307
- References..... 307
- Chapter 9: S3 Resume 313**
- Threat Model..... 315
- LockBox 317
 - Case Study..... 319
 - Attack and Mitigation 332
- Summary..... 333
- References..... 333
- Chapter 10: Access Control 335**
- Boot Access Control 337
 - What a User Knows: Password 337
 - What a User Has: Token 342
 - What a User Is: Biometrics 342
 - Other Considerations 342
 - Case Study..... 343
 - Attack and Mitigation 345
- TEE Access Control 347
- Feature Configuration Control 347
 - User Physical Presence 347
 - UEFI Variable..... 347
 - Case Study..... 348
 - Attack and Mitigation 349

Device Configuration Control	349
Physical Presence	350
Secure Console.....	350
Case Study.....	351
Attack and Mitigation	355
Storage Access Control.....	356
Hard Drive Password	356
Fast Boot Impact	357
Unlock in a Warm Reset	357
Auto Unlock in S3	357
Runtime D3 impact.....	358
Password Update.....	359
User Password vs. Master Password	359
Retry Count.....	359
Hard Drive Freeze	359
Secure Console.....	359
Case Study.....	360
Attack & Mitigation	371
Network Access Control.....	372
Case Study.....	372
Attack & Mitigation.....	374
Device Access Control.....	375
Case Study.....	375
Attack & Mitigation.....	377
Summary.....	378
References.....	378
Chapter 11: Configuration.....	383
UEFI Variables	383
Integrity Protection	384
Availability Protection	406
Confidentiality Protection	416

TABLE OF CONTENTS

- Attack and Mitigation 421
- UEFI PI Firmware Volume 427
- UEFI PI PCD (Platform Configuration Data)..... 427
- Summary..... 429
- References..... 430
- Chapter 12: Security Model 433**
- Confidentiality 433
 - Bell-LaPadula 434
- Integrity..... 436
 - Biba Model..... 436
 - Clark-Wilson Model 438
- Others 440
- Application 441
 - Trusted Computing Group Security Model 441
 - Firmware Resiliency (Protection, Detection, Recovery)..... 443
- Summary..... 456
- References..... 456
- Chapter 13: Virtual Firmware 459**
- New Threats in the Guest Domain..... 460
 - Case Study..... 461
 - Attack and Mitigation 473
- Device Interface 476
 - Case Study..... 477
 - Attack and Mitigation 481
- Special Feature 482
 - Case Study..... 483
- Summary..... 487
- References..... 487

Part III: Security Development	493
Chapter 14: Firmware Secure Coding Practice	495
Basic Security Practice	495
Secure Coding Practice	495
Secure Design Practice	517
Boot Firmware Secure Design Practice	522
Advanced Secure Coding Topic	525
Side Channel Attack	525
Fault Injection.....	545
High-Risk Area in Firmware	551
External Input	552
Race Condition	553
Hardware Input.....	554
Secret Handling	555
Register Lock.....	556
Secure Configuration.....	556
Replay/Rollback.....	557
Cryptography	558
Summary.....	559
References.....	559
Chapter 15: Compiler Defensive Technology	571
Break the Exploit.....	573
Stack Check.....	573
Code Protection	577
Address Space Layout Randomization	579
Control Flow Guard	580
Hardware-Based Control Flow Guard	589
Speculative Load Hardening.....	594
Eliminate the Vulnerability	598
Static Analyzer.....	598
Address Sanitizer	598

TABLE OF CONTENTS

- Hardware-Based Address Sanitizer 605
- Uninitialized Data Check..... 608
- Arithmetic Check 613
- Summary..... 618
- References..... 618
- Chapter 16: The Kernel 625**
- Break the Exploitation..... 628
 - Code Protection 628
 - Address Space Layout Randomization 634
 - Control Flow Guard 642
 - Address Sanitizer 644
- Contain the Damage..... 654
 - User Mode/Supervisor Mode..... 654
 - Virtual Machine Monitor 659
 - Trusted Execution Environment..... 669
 - System Partitioning 670
- Summary..... 672
- References..... 672
- Chapter 17: Trusted Execution Environment 681**
- CPU-Based TEE 681
 - X86 SMM 681
 - ARM TrustZone..... 707
 - Intel SGX 716
 - AMD SEV 718
 - Intel TDX 719
 - IBM Z 719
 - RISC-V Keystone..... 719
 - RISC-V/ARM-M MultiZone..... 720
 - Standards, Frameworks, and SDKs 721
 - Summary 722

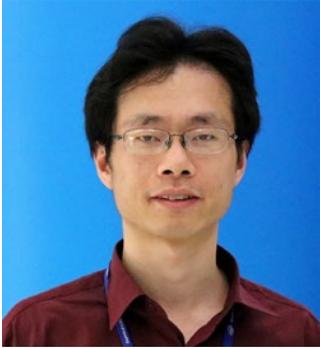
Coprocessor-Based TEE	723
Intel Converged Security and Management Engine (CSME)	724
Apple Secure Enclave Processor (SEP).....	727
Google Titan.....	727
Microsoft Azure Sphere: Pluton	728
Summary.....	729
References.....	732
Chapter 18: Silicon Secure Configuration.....	745
Flash Lock.....	745
BIOS Write Protection	745
BIOS Region Selection	747
SPI Region Lock.....	749
SPI Region Access Control.....	750
SMM Lock.....	751
SMRAM Address Alias Lock	754
SMRR.....	759
SMM Code Access Check	759
Global SMI Lock	760
IOMMU	761
IOMMU Protection for DRAM.....	762
IOMMU Protection for SRAM.....	762
Silicon Support for DMA Prevention	762
Summary.....	764
References.....	765
Chapter 19: Cryptography	767
Modern Cryptography	767
Cryptography Usage in the Firmware	770
Algorithm Recommendation	772
Some Concepts.....	772
Challenge in the Firmware	787
Attack and Mitigation	790

TABLE OF CONTENTS

- Quantum Safe Cryptography..... 792
 - Security Challenge..... 792
 - Quantum Safe Algorithm 794
 - Quantum Cryptography..... 805
 - Algorithm Recommendation 809
 - Preparation for the Future 811
- Summary..... 813
- References..... 814
- Chapter 20: Programming Language..... 825**
 - Assembly Language..... 825
 - C Language..... 828
 - Rust..... 828
 - Rust Security Solution 830
 - Unsafe Code 847
 - Current Project 850
 - Limitation..... 851
 - Others 852
 - Summary..... 852
 - References..... 852
- Part IV: Security Test 855**
 - Chapter 21: Security Unit Test 857**
 - Security Unit Testing Plan 857
 - Advanced Security Unit Testing..... 860
 - Fuzzing 861
 - Symbolic Execution 866
 - Formal Verification 868
 - Design for Security Test 870
 - Summary..... 872
 - References..... 872

Chapter 22: Security Validation and Penetration Test	875
Security Validation Plan	875
Penetration Test Plan	886
Summary.....	888
References.....	889
Chapter 23: Maintenance	891
Mitigation Strategy and Tactics.....	891
Supply Chain for the Firmware Components	894
Firmware Component Certificate and Manifest.....	894
Firmware Attestation Data.....	896
Vulnerable Firmware Detection.....	899
Antivirus for Firmware	900
Firmware Update	900
Summary.....	901
References.....	902
Index.....	907

About the Authors



Jiewen Yao is a principal engineer in the Intel Architecture, Graphics, and Software Group. He has been engaged as a firmware developer for over 15 years. He is a member of the UEFI Security Sub-Team (USST) and the Trusted Computing Group (TCG) PC Client sub-working group. He has presented at industry events such as the Intel Developer Forum, UEFI Plugfest, and RSA Conference. He worked with coauthor Vincent Zimmer to publish 30 “A Tour Beyond BIOS” technical papers for tianocore.org and firmware.intel.com. He holds 40 US patents.



Vincent Zimmer is a senior principal engineer in the Intel Architecture, Graphics, and Software Group. He has been engaged as a firmware developer for over 25 years and leads the UEFI Security Sub-Team. He has presented at industry events such as the Open Source Firmware Conference, LinuxFest Northwest, Intel Developer Forum, UEFI Plugfest, Open Compute Project Summit, Black Hat Las Vegas, BSides Seattle, ToorCon, and CanSecWest. In addition to collaborating with Jiewen Yao on many whitepapers, he has coauthored several books on firmware, papers, and over 400 issued US patents.

About the Technical Reviewer

Tim Lewis is the CTO and head of the Office of Security and Trust for Insyde Software. With over 30 years of BIOS experience, he has served on the UEFI board of directors and chaired the UEFI Security Sub-Team (USST). He is an active member of his local CS2AI chapter near his home in Folsom, California.

Organization and What Is Covered

Currently, we already have books to separately introduce the topics of firmware and security. The purpose of this book is to link these areas together and provide the best practices in the security development of firmware.

The whole book consists of four parts:

Part I: Overview

Chapter 1 includes a brief overview of the firmware in a system.

Chapter 2 describes a proactive development lifecycle for the firmware. We will introduce a general threat model for the firmware and use a real example to demonstrate how to do threat model analysis.

Part II: Security Architecture

Chapter 3 describes the first aspect of the firmware resiliency – protection.

Chapter 4 describes the second aspect of the firmware resiliency – detection.

Chapter 5 describes the third aspect of the firmware resiliency – recovery.

Chapter 6 extends the resiliency concept from the firmware to the operating system (OS).

Chapter 7 introduces the trusted boot concept, besides the firmware resiliency in Chapters 3, 4, and 5.

Chapter 8 focuses on the security aspects of the device firmware, including secure device communication and device attack prevention.

Chapter 9 discusses S3 resume – a special boot path in the system firmware.

Chapter 10 presents the access control technology in the firmware, including boot access control, device access control, feature configuration control, and so on.

Chapter 11 examines the confidentiality, integrity, and availability of the firmware configuration data.

Chapter 12 provides a brief introduction for the security model and maps the Clark-Wilson model to existing firmware features.

Chapter 13 explains the virtual firmware, which may include a new threat model.

ORGANIZATION AND WHAT IS COVERED

Part III: Security Development

Chapter 14 introduces the general secure coding practices for firmware.

Chapter 15 discusses the compiler defensive technologies which may be used in the firmware.

Chapter 16 describes the possible firmware kernel hardening technologies.

Chapter 17 compares different trusted execution environments (TEEs), including system management mode (SMM) and ARM TrustZone.

Chapter 18 shows the silicon secure configuration, such as flash lock and system configuration lock.

Chapter 19 explains the cryptography usage in the firmware space, including an introduction for post-quantum cryptography.

Chapter 20 introduces the Rust language – a type-safe language choice for firmware development.

Part IV: Security Test

Chapter 21 discusses the security unit test, including fuzzing and symbolic execution.

Chapter 22 describes the security validation and penetration test plan.

Chapter 23 recaps the mitigation strategies – eliminate the vulnerability, break the exploitation, contain the damage, and reduce the attack window – and focuses on how to maintain the firmware in the lifecycle.

Whom Is This Book For?

The target audience of this book includes firmware architects, developers, and validation engineers. We assume the reader has basic knowledge on the following domains:

- 1) Computer system architecture, such as X86, ARM, PCI bus, and so on
- 2) Operating systems or embedded systems
- 3) Programming language – C language
- 4) Firmware design and development experience, such as EDK II, coreboot, ARM Trusted Firmware, and so on

Acknowledgments

We would like to acknowledge our technical reviewer and Apress editors. We would also like to acknowledge the many engineers and collaborators with whom we have worked in the industry, standards groups, and open source communities.

Preface

In 1988, the Morris worm was released from the lab onto the Internet. It was the first time that a computer virus had gained significant mainstream media attention. People started to realize that a software flaw was vital, which changed the perception of security on the Internet. Ten years later, in 1999, the Chen Ing-hau (CIH) virus brought another big impact because the Basic Input/Output System (BIOS) flash chip was attacked and the impacted system could not boot at all. A technician was required to reprogram the BIOS. This was probably the first wave of firmware security. The firmware developers started to invent the idea of locking the flash region in response to CIH.

The second big wave was around 2009. Invisible Things Lab released a series of BIOS attacks and presented in the Black Hat conference. These attacks included the REMAP base address register (BAR) attack, Bitmap (BMP) file attack, system management mode (SMM) callout, SMM cache poisoning, and Authenticated Code Module (ACM) hijack. The firmware developers realized that locking the flash and system management RAM (SMRAM) was far from enough. Similar to the software development process, the firmware development domain also needs to apply the security development lifecycle (SDL).

The third wave of firmware security commenced in 2014, where more and more firmware attacks appeared in different hacking conferences. This next wave of attacks included exploits against the capsule image, setup variable, SMM communication, memory mapped input/output (MMIO) BAR in SMM, Speed Racer, Direct Memory Access (DMA), S3 boot script, and Unified Extensible Firmware Interface (UEFI) secure boot policy - lots of firmware features became the attack point. It is just like the old saying, "If there is a will, there is a way." The firmware engineers started to invent new technologies and processes to harden the firmware design and development.

The war of defense and attack never ends. Today, firmware security has become more and more important. The National Institute of Standards and Technology (NIST) published the documents SP800-193 - Platform Firmware Resiliency Guidelines and SP800-155 - BIOS Integrity Measurement Guidelines. The Trusted Computing Group (TCG) created the Cyber Resilient (CyRes) workgroup to focus on three principles of

PREFACE

resilience (protecting, detecting, and recovering), the Device Identifier Composition Engine (DICE) workgroup to explore the security principles for the Internet of Things (IoT) ecosystem, and the Attestation workgroup to provide a framework for attestation. The Distributed Management Task Force (DMTF) created the Secure Protocol and Data Model (SPDM) for device firmware authentication and measurement, as well as key exchange. The Internet Engineering Task Force (IETF) created the Software Updates for Internet of Things (suit) working group to define a firmware update solution for small IoT devices and Remote Attestation Procedures (rats) working group to standardize the formats and procedures for the assertions about the system components. The cloud companies invented different platform Root-of-Trust (RoT) techniques, such as Microsoft Cerberus, Google Titan, and Amazon Nitro. The device vendor started building the device root-of-trust to report the device certificate. In this fourth wave, every firmware engineer needs to understand the aspects of firmware security - from industry standards, architecture, design, and implementation up to the final stages of test and validation.

Foreword

When Saudi Aramco was attacked in August 2012 by the Shamoon computer virus, it was a wake-up call to the entire industry. Saudi Aramco is one of the largest companies in the world and holds the second largest amount of crude oil reserves. A shutdown of Saudi Aramco for a prolonged amount of time would have had a devastating impact on the oil economy, and this could have plunged the world into an economic recession.

The Shamoon virus was by today's standards a relatively simple virus. It overwrote the master boot record on the hard disk with garbage, rendering the system unbootable. While the infestation was effective – reportedly more than 30,000 systems got impacted – the much-feared secondary effect of the global oil supply chain hiccup didn't materialize. That was in part because the hard disk is an “easy” to service component and Aramco was able to replace the hard disks in time to avoid any serious impact.

However, this attack made folks in the industry and intelligence communities around the world wonder: What would have happened if Shamoon went after a non-serviceable component, like the flash chips where the system firmware is stored? This is much harder to repair on-site, and it would have required the systems to be shipped back to the manufacturer. This would have taken a lot more time, and in that case a global oil supply shortage could not have been avoided, potentially even triggering a global recession.

This incident prompted a cross-industry/cross-government collaboration that eventually resulted in a set of recommendations from the National Institute of Standards and Technology (NIST). The first set of recommendations, BIOS Protect Guidelines (NIST SP800-147), described requirements such as

- BIOS update authentication: The firmware needs to be signed by the proper authority.
- Integrity protection: The firmware cannot be modified, either unintended or maliciously.
- Non-bypassibility: There is no way to bypass the authenticated firmware update mechanism.

FOREWORD

This specification was followed up in 2018 by another NIST publication, Platform Firmware Resiliency Guidelines (NIST SP800-193). This specification extended the previous one by focusing on recovery. Its central tenet was simple: providing protection against unauthorized firmware updates is not enough. The firmware is a complex piece of software, and it will, probabilistically, contain bugs that can be exploited by an attacker. How do you quickly and seamlessly recover if this happens?

Before you consider this problem to be a PC-only threat, think again. With the proliferation of IoT devices in homes and factories, and SCADA devices to control the national grid, firmware security has become a key part of the world's critical infrastructure protection. Just imagine the harm a malicious attacker could do by taking over the firmware of pipeline controllers in a gas line in Siberia, Russia, during the winter. You'll be hard-pressed to send someone out there to fix the problem.

The Saudi Aramco attack and its resulting NIST secure firmware recommendations solidified the importance of firmware security in the industry. Those recommendations built on years of academic and industrial research in this area. I was personally involved in some of these efforts, ranging from IBM's physical secure coprocessor (IBM 47xx) projects and TCG/TCG frameworks to secure hypervisor research and working with other industry partners on the foundations of NIST SP800-147.

The authors Jiewen Yao and Vincent Zimmer have bundled together into this book their combined years of experience in developing secure firmware and building resilient systems. Every computer, big or small, starts with firmware, and if that's compromised, all is lost, so following the authors' guidance is as relevant today as it was in 2012.

—Leendert van Doorn
Redmond, WA, USA

PART I

Overview

Firmware is a special type of software. There is a secure software development lifecycle defined for software. These lifecycle practices can be applied to the firmware world.

CHAPTER 1

Introduction to Firmware

This chapter will provide an overview of system firmware. Although the space of implementations of system firmware is quite broad, details that relate to the secure construction of firmware will be discussed.

Similarity Between Firmware and Software

Firmware vs. Embedded System vs. OS Kernel vs. OS Application

Firmware is the lowest layer of software on the platform. Modern firmware, such as UEFI Platform Initialization-based firmware, like EDK II, U-Boot (Universal Boot Loader), coreboot, Open Power skiboot, and so on, are predominately written in C with a small amount of assembly language code. This code is often stored in a non-volatile storage container bound to the platform. The relationship of firmware to the rest of a platform stack is shown in Figure 1-1.

Human User
GUI
Application
Middleware
Libraries
OS Drivers
OS Kernel
OS HAL
Hypervisor
Firmware
Hardware

Figure 1-1. *Stack from Hardware to User*

Given this C-based provenance, the C code can be susceptible to the class of attacks that afflict higher-level software. These attacks include memory safety issues, involving the variants of buffer overflow, such as stack overflow, heap overflow, and integer overflow. In addition, control flow attacks against C code in the application or OS space can be repurposed against system firmware. Beyond memory issues, other aspects of attack that can occur against the firmware include confidentiality concerns, such as stealing secrets. Beyond that, the firmware often participates in the root-of-trust flow for a system, so integrity considerations are of importance since any unauthorized code flows in the platform can deny the platform promise of that feature. The platform boot

can include accessing the network, so considerations of network security can also be applied to the platform firmware. And there are few, if any, platforms that have only a single central processing unit (CPU) core in the system on a chip (SOC), so the firmware must support multiprocessing (MP) and defend against the various classes of attacks, such as race conditions, that inhere in this application domain. Finally, the platform firmware must also defend against other classes of attacks, such as side channels, confused deputy, and time-of-check/time-of-use (TOC/TOU) attacks.

Given these exposures listed, the firmware may have similar platform hardening strategies, albeit with implementations customized for the domain. These include hardening tactics such as stack cookie checks, data execution protection (DEP), address space layout randomization (ASLR), control flow guard/integrity (CFG/CFI), code signing enforcement check, sandbox with interpreter, access control (user authentication and authorization), network security, and cryptography suitable in the firmware execution environment.

Beyond the defenses, the firmware may have similar software security validation strategies, but with different implementations than higher-level software regimes. These validation approaches can include static code analysis, dynamic code analysis (address sanitizer, ASan), fuzzing, symbolic execution, and formal verification when possible.

Distinction Between Firmware and Software

Although firmware is typically written in a higher-level language like C, it often has special requirements. These requirements begin with the environment. Specifically, firmware has a size limitation that is driven by the small ROM size and small RAM size in microcontrollers, only having SRAM or cache that can be used before the DRAM is ready, Management Mode (MM) size leveraging stolen system memory, and limited stack and heap size.

Additional limitations include the execution-in-place (XIP) code of early code. Namely, some code executes in the ROM. One aspect of this is the ROM code, wherein some code has no writable global variable in the data section. And for this early code, such as the UEFI Platform Initialization PI, Slim Bootloader stage 1, and coreboot romstage, there is no memory management, including no virtual memory, although page tables may be used.

Beyond memory management, there are challenges with execution isolation in early code flows. For example, ring separation might or might not be available. As such,