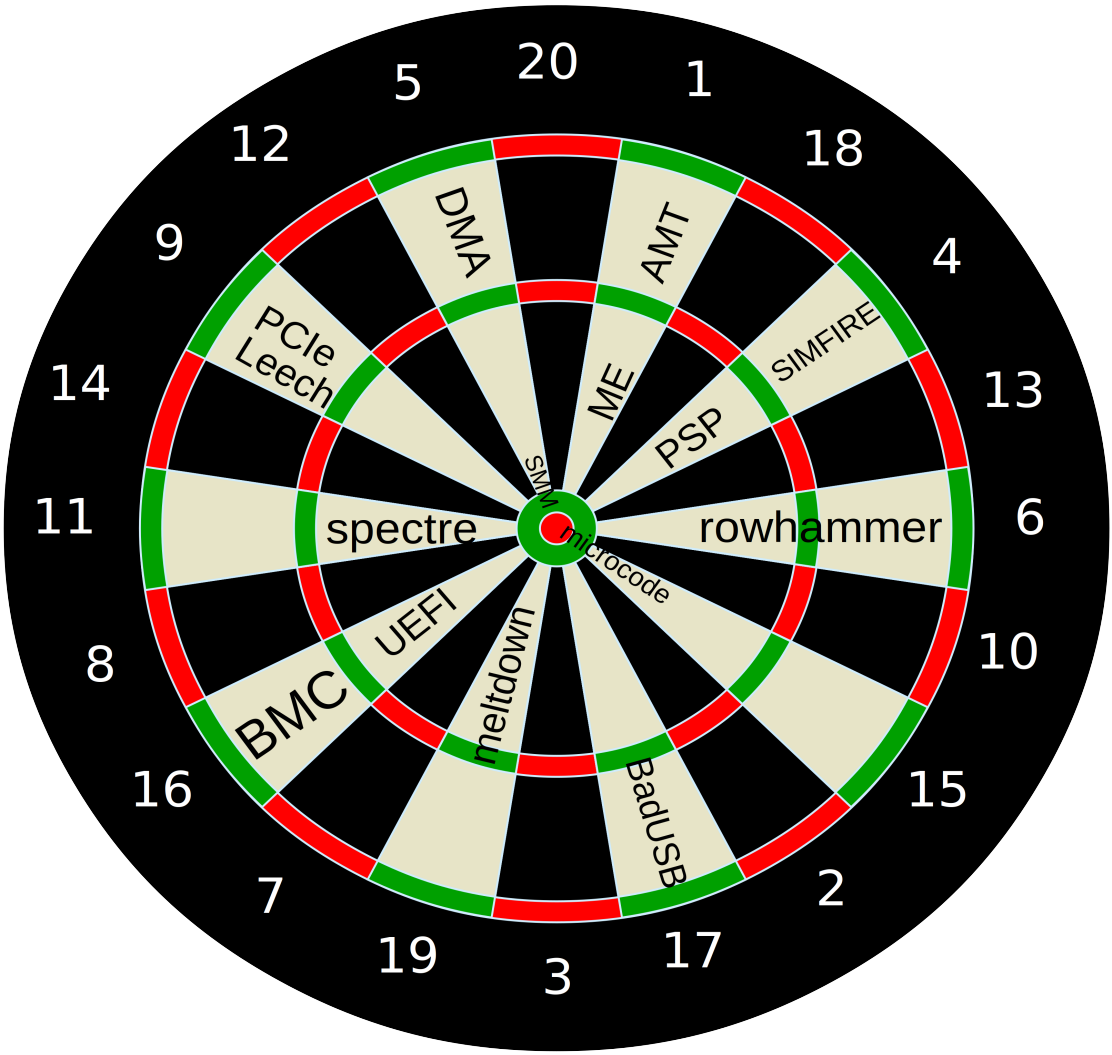
Platform Firmware Security Defense

for Enterprise
System Administrators
and Blue Teams



Paul English & Lee Fisher
PreOS Security
https://preossec.com

Firmware Security Defense for Enterprise Sysadmins and Blue Teams

Contents

Copyright

Cover

Content:

Disclaimer

Firmware is software. But unlike the operating system and application software stored on your hard disk / SSD, you can't necessarily wipe and reinstall. Since platform firmware makes the system operate, you may render your system completely inoperable when working with firmware. In this book we will try to highlight the more dangerous operations, particularly automated fuzz testing, or write/rewrites. As with most technology advice, or software - proceed with caution and at your own risk.

Note

In an enterprise environment, ideally there will be many identical or near identical systems deployed. Therefore it is feasible to test some of the more dangerous operations on a single sample before deploying to a fleet. PreOS Security strongly recommends this approach. In an enterprise environment, you are also likely to have warranty and extended support from the manufacturer/OEM. Don't hesitate to consult with the manufacturer before proceeding - in fact, this will help everyone and the ecosystem by highlighting to the manufacturers that customers care about these issues, and want reliable, secure systems.

About This Book

Footnotes

Rather than footnotes for this first edition, we've included the latest version of our Awesome Firmware Security *Github Awesome list* living document as an Appendix. Every reference that should be footnoted is included in this document.

We welcome your feedback! We'd like this book to be top notch, and neither of the authors are writers.

editor@preossec.com

1 Introduction

We're going to give you a brief introduction to modern platform firmware technologies, starting from the Intel-based, BIOS-based IBM PC. As an IT expert some of this may be material you already know, but we'll keep it brief and move on to the security implications.

Platform firmware has evolved and expanded quickly and as software it has moved towards increasing layers of abstraction and code complexity. Naturally this has increased lines of code and attack surface, which means there is also more to defend. Defense is part awareness, and part action and we'll cover both, for a System Administrator, Site Reliability Engineer or Enterprise Incident Response *Blue Team* member.

Some of these topics are also relevant to personal consumer level firmware defense, but we plan on releasing a dedicated book on that topic soon.

1.1 Defining Platform Firmware

The concept of *firmware* has multiple definitions. With mobile/embedded/IoT devices, firmware is considered all of the software (operating system and applications). A firmware update for a phone/camera/game console is quickly becoming standard practice and when such devices are expected to have an internet connection, they often include internet enabled automated or semi-automated updating capabilities. While slightly riskier than updating application software in an operating system, a firmware update on these devices typically includes checksums and multi-boot-loader capabilities to avoid bricking the device. Security on mobile/embedded/IoT devices is critical, particularly as they are often exposed to internet based attackers continuously. However, many of these firmware implementations are sophisticated enough to be considered operating systems in their own right, often an embedded version of Linux, or a variant of BSD.

Instead, for the purposes of this book we are discussing platform firmware such as UEFI, ACPI and PCI option ROMs. It is worth noting that any mobile/embedded/IoT device that is complex enough to have both an operating system and applications likely also has not just one, but several *platform firmwares* for additional controllers or processors onboard. For example, a cell phone has a primary CPU, running Android or iOS, but it also has a baseband processor, running firmware.

Platform firmware is software that makes the hardware work. Terms like UEFI, ACPI and PCI option ROMs are components of platform firmware. When hardware is powered-on, firmware code is executed, and that code helps initialize the hardware. The firmware's bootloader loads the operating system, or sometimes the next level of firmware bootstrapping, leading eventually to an operating system boot. Most platform firmware is stored on flash ROM chips. In some cases, like UEFI, firmware is also loaded as a file from a file system (foo.efi), like an operating system executable (foo.exe).

Platform firmware has evolved significantly since the original BIOS-based IBM PC. It has become more complex, has significantly increased attack surface, and has proliferated into many different firmwares installed on any given motherboard, and even more when peripherals are considered. Unfortunately even when platform firmware updates are available, system administrators are sometimes not aware of the updates, and they

are not applied. Some firmware implementations are not updatable at all, or their space limitations prevent a necessary update from being applied. Only a relatively few platform firmware implementations have the ability to connect directly to the internet for updates. Even though some have a full network stack like UEFI, to support network boot, and out-of-band management like BMC, IPMI and Redfish to support remote management, most still lack the ability to download updates automatically. Only a few manufacturers, and only specific models support automatic firmware updates via the operating system automated update mechanisms, such as Windows Update and Linux Vendor Firmware Service, and they are not all signed and encrypted. Instead, even today, updates must be provided by running software provided by each firmware provider, to check the status and apply updates if needed. It is no surprise that platform firmware updates are neglected.

For the duration of this book we'll be talking about *platform firmware* and may interchangeable simply say *firmware*.

1.2 Types of Platform Firmware

1.2.1 History - BIOS to EFI

BIOS has been around since the IBM PC, and is still present on some modern systems. BIOS was a 16-bit Real Mode codebase, used by MS-DOS. Most other operating systems only used BIOS to initialize themselves, and rely on their own drivers to access the system. BIOS was *interrupt-based*, and offered services to the OS/application: Interrupt 10h was disk I/O, Interrupt 13h was video I/O, etc. IBM PS/2 had a variant called ABIOS (Advanced BIOS), which was a protected-mode reentrant solution, resolving many of the initial technical limitations of BIOS, but it was proprietary to PS/2 and died with the platform. Initially there were no considerations for security included, but as security became a concern in the PC ecosystem, BIOS added passwords and security began to be added. Historically, there were a few BIOS implementations, by OEMs and IBVs (Indepdendent BIOS Vendors). One reason UEFI came along was to solve the technical limitations in BIOS.

ARC was a firmware technology that ran on some RISC systems, including DEC Alpha, PowerPC, MIPS, etc. The architectural design of ARC - firmware executables stored on a FAT file system volume, is very similar to the design of EFI.

1.2.2 EFI/UEFI

When Intel came out with the Itanium processor, EFI was invented as it's firmware layer, as x86-centric BIOS would not work on the Itanium. Later, Intel used EFI as an alternative to BIOS as the main platform firmware technology for their x86/x64 systems. When Apple moved from the PowerPC to the Intel processor, they adopted EFI for their firmware. Microsoft Windows eventually migrated from BIOS to using and later requiring UEFI for Windows. EFI became UEFI Unified EFI in a move towards a standardized, open source code base known as Tianocore. UEFI works on both Intel and AMD x86 and x86_64 systems. UEFI now also works on ARM systems, and is available on some ARM servers. UEFI has an architecturally-indepdendent bytecode

(EBC) that is sometimes used. EFI had little security, later UEFI added *Secure Boot* (more on this later) and code signing of executables.

1.2.3 Extensions to BIOS / UEFI

1.2.3.1 Remote Booting

Some early PCs and terminals shipped as "diskless workstations", and were designed to boot from binaries loaded remotely over a network. This feature became standardized as PXE ("Preboot eXecution Environment") and became included in many BIOS systems. As BIOS was only designed for local boot, this involved adding a full network stack, and NIC drivers into the firmware - either in the BIOS itself or in the firmware on the NIC. Current UEFI's network stack supports IPv4, IPv6, and iSCSI. In addition to PXE, UEFI also has "UEFI HTTP[S] Boot", the ability to boot over HTTP. Remote booting gives attackers a much larger attack surface, if the protocols are not properly secured and authenticated.

1.2.3.2 ACPI

Various firmware solutions for power mangement, ISA PNP (Plug 'n Play), APM, etc have evolved into ACPI. ACPI was added to BIOS to enable power management, allowing Suspend/Resume states in the power cycle. ACPI is also used by UEFI, and the ACPI specification is controlled by the UEFI Forum. Besides power management, ACPI is a general purpose firmware extensibility method, where vendors can define their own code modules (tables), containing code and/or data. ACPI has an architecturally-indepdendent bytecode (AML, ACPI Machine Language), so each OS needs to have an ACPI VM to execute ACPI code. Microsoft Windows PCs include ACPI tables that have embedded Windows executables used when Windows needs to sysprep/restore a system. Many vendors have ACPI specs with diverse uses. The modding community has long been updating ACPI tables to workaround getting Windows or macOS to work on different hardware. Attackers can potentially use the same techniques as modders to update ACPI to include their malware.

1.2.3.3 Beyond the PC: embedded Linux bootloaders

U-Boot and *coreboot* are popular boot loaders on embedded Linux systems. When used on Intel systems, these boot loaders run a BIOS payload. Coreboot and U-Boot can also invoke a UEFI-based payload – on Intel or ARM systems. Some embedded solutions nest payloads, coreboot and U-Boot together. To complicate things even more, traditional UEFI firmware can be built to use coreboot as the UEFI initialization layer (and not a payload). So payload-based boot loaders may have a combination of firmware technologies.

ARM can run coreboot, U-Boot, or less commonly, UEFI. Two examples of ARM-centric firmware security technologies include *TrustZone* and *ARM Trusted Firmware*.

Google Android uses coreboot with Verified Boot. Google ChromeOS systems use coreboot with Verified Boot. On Intel ChromeOS systems, coreboot loads a BIOS payload, not a UEFI-based solution. While this BIOS payload has limited security, it is

being protected by coreboot's Verified Boot. ChromeOS often uses either TPM on Intel or TrustZone on ARM as backing trust store.

1.2.4 Beyond the CPU: other processors

1.2.4.1 TPM / fTPM

A TPM chip was added to some PCs to provide a *root-of-trust*, but many BIOS systems don't use it. The TrustWorthy Computing Group maintains standards for TPM and related protocols. A TPM chip is designed to be a store for secrets that is hard to access. In addition to a hardware-based TPM, there is also a *firmware TPM*, a software (firmware) implementation that works on an alternate processor, such as the Intel ME processor as Intel AMT-based firmware. Presumably any firmware TPM-hosting processor is not as hard to access as the *tamper-resistant* TPM. A recent example of a vulnerability in dedicated TPM hardware is the Infineon TPM vulnerability known as *ROCA*, impacting specific chips with weak key generation algorithms. Some manufacturers integrating the affected Infineon TPM chips refused to supply a fix for this bug.

1.2.4.2 Management Processors

Intel ships it's Management Engine (ME) integrated on the CPU die, as does AMD with their Platform Service Processor (PSP). Apple ships their T2 processor as a discrete component. In each case, the management processor can run applications, such as an application to provide remote management capabilities (such as the Intel AMT), fTPM functionality and many more.

1.2.4.3 Graphics Processor Unit (GPU)

A Graphics Processor Unit (GPU) was added to the PC to augment video performance. Now GPUs are used as general purpose processors (GPGPU). Some high-end systems have arrays of GPUs. Typically the GPU adapter attaches via PCIe (or in some cases Thunderbolt), therefore to be seen at boot time it requires an *option ROM*.

1.2.4.4 Network Interface Card (NIC)

Initially PCs did not include network adaptors, but now almost every computing device has one, often many NICs. While some NICs are relatively simple, some technologies such as *TCP offloading* require the NIC to have a full general purpose processor, and fairly complex firmware to bootstrap and sometimes a full-blown operating system to operate. In cellular technology, the cellular NIC, or modem contains a powerful, typically black box firmware known as the *baseband*. In this domain it is important to include and consider WiFi, Bluetooth and NFC technologies, as each provides networking or network-like access to a machine, and may be on by default or have associated platform firmware.

1.2.4.5 BMC (Baseboard Management Controller)

The baseboard management controller is typically a standalone system in it's own right with CPU, RAM, storage and networking, designed to managed the primary host system, often used in blade servers. Typically these are designed to be powered on even when the host system is powered off, and provide remote control to system administrators. BMC is a generic term, and there are many branded implementations, including IPMI, DMTF, SMASH, DASH and Redfish. Historically these controllers were built as add-on products that plug into the motherboard, but due to their usefulness, there are BMC-like features built directly into the CPU main die - even if it means effectively shipping an entire additional CPU and operating system inside the primary CPU - in this case, *Intel ME*, specifically *Intel AMT* running as an application on Intel ME and *AMD PSP*.

1.2.4.6 Blade Servers

When the PC evolved from a desktop to a server, a rack of blades added a new trust level. A rack of multiple blade servers has a Service Processor on the rack which controls the blade servers, the new root of trust for all of the blades. Attackers who control the Service Processor control all of the blades.

1.2.4.7 Smartphones

A modern phone is a powerful PC, with platform firmware, and embedded OS/appllications. In addition, a phone also includes a baseband processor and it's firmware. These processors and their firmware are usually trade-secret closed-source technology, so it is difficult to say how to defend them. Pessimistically, attackers probably have access to these trade secrets, and can use this processor to host their malware, invisible to the main system's CPU and firmware.

1.2.4.8 Hypervisors

One way to secure a trusting bare-metal CPU from attackers is to have a background mangement mode processor. Another way is to virtualize a system, adding new layers of security against attackers. With the hypervisors, virtualized hardware now includes virtualized firmware. Compared to some bare-metal targets, hypervisors may be easier targets for attackers. The hardware and firmware components are OS-level processes, the firmware is more accessable on disk rather than on flash and there is usally a remote-access way instead of requiring physical access to a bare-metal system. For defenders, attacks against VMM process may be more observable using OS-level SIEMs and logging tools, and defenders at least pay attention to OS-level interactions.

1.2.5 Beyond System Firmware: Peripheral Firmware

Peripheral firmware expands system firmware. These days, there are basically two kinds of peripherals, cards you insert into slots of the motherboard (ISA, ISA-PNP, EISA, PCMCIA, PCI, PCIe, etc.), and devices (hard disks, SSDs, monitors, printers, scanners, etc.) that you connect via ports (USB, Thunderbolt serial, Firewire, etc.). The lines are blurring as one can attach PCIe devices externally (via Thunderbolt), and sometimes

skip a layer, such as the SAS/SATA layer with NVMe devices directly attached to PCIe. Microprocessors (and RAM,ROM, NVRAM) have grown in capabilities - low power operation, more transistors, higher complexity and requirements for peripheral devices have increased, so naturally more powerful processors (and more RAM, ROM, NVRAM) are included in all peripherals, along with more lines of code to support the increased features and therefore a greater attack surface.

1.2.5.1 PCIe and Thunderbolt

When BIOS was first deployed, the main peripheral card bus interface was ISA. These days it is PCIe. When a hardware vendor adds features to their cards (network/disk/video adapters, etc.) they need to extend the features that BIOS provides. Cards extend BIOS via Option ROMs, aka Expansion ROMs, firmware contained on the card that extends the BIOS. The BIOS method of extending it's functionality is by hooking interrupts. Thunderbolt is basically PCIe via ports, instead of motherboard slots, and has similar option ROM driver security issues, and it is easier for attackers to attach devices to ports than open up systems and add cards to the motherboard.

We've already addressed GPU and NIC firmware, but it is not uncommon for PCIe adapters to attempt to offload work from the main CPU by embedding a CPU of their own. RAID adapters are also in this category, but any PCIe device that is sufficiently high throughput (or high-end/expensive) is a candidate.

1.2.5.2 USB

USB-based devices have their own firmware stack. Attackers have multiple ways to use the USB firmware to make USB device act in interesting ways. The firmware drivers on a USB device can claim to be a network device, a local storage device, or many other options that can give entry to a overly-trusting operating system or end-user that blindly inserts USB devices that people give them.

For devices connected via ports, there is firmware on the device, as well as often firmware inside modern cables (USB, Thunderbolt, etc.). An attacker can attach a rogue PCIe or USB device to a system with many attack vectors: USB devices like the Hak5 USB Rubber Ducky device is common approach for USB, PCILeech is one for PCIe. Recent OS updates include defenses such as disabling ports during suspend/resume states.

1.2.6 Peripherals of Peripherals

1.2.6.1 Storage Devices

Storage devices are typically attached to a peripheral bus, and there is a controller on the system side, such as a SAS or SATA host bus adapter, or RAID controller, which may have it's own firmware. But every storage device also has firmware on the device itself. Unfortunately there is no better term for this than "controller." In the case of spinning disks, the controller is necessary to map the complex and mutable phsyical disk locations (blocks) to a fairly uniform presentation of logical blocks for the operating system, and to communicate via standardized SAS and/or SATA commands, as well as the SMART management protocol. Solid-state storage devices operate as collections of discrete flash chips, so in addition to mapping blocks, and hiding / remapping bad

blocks, SSD controllers typically also do an internal variant of RAID for additional data resiliency, and manage a rotating buffer of space for wear levelling. Both spinning disks and SSDs often add features such as encryption. Every additional feature requires additional code, with additional attack surface.

1.3 Securing the Boot Loader

When the system firmware attempts to load the operating system, there are various points where an attacker can exploit a BIOS or UEFI system. Various technologies have been added to help secure the boot processs.

Secure Boot is a feature of UEFI that uses PKI to secure some phases of the boot process. With Secure Boot disabled, a user can be tricked into running UEFI-based malware, or an insecure operating system. Apple's MacOS does not use Secure Boot, but provides a different variant of boot security as Apple has been evolving their EFI independently from UEFI. With the introduction of their T2 security processor, Apple introduced a Secure Boot of their own, distinct from UEFI/EFI Secure Boot. Windows and Linux can be configured to use Secure Boot, and Microsoft has strongly encouraged OEMs to migrate to UEFI v2.x in order to use the Secure Boot feature. UEFI Secure Boot can be configured not only to protect the boot process, but also kernel extensions and even end user applications. At sufficient scale, it makes sense to consider keying Secure Boot for your enterprise, and it may make sense to utilize it's protection fully to the application level.

Verified Boot is a feature of coreboot, on Android and ChromeOS systems, which is conceptually similar to Secure Boot: if Verified Boot is diabled, attackers can subvert the boot process.

Measured Boot uses TPM measurements to hash each phase of the boot process in order to find malware. *Trusted Boot* uses Intel TXT measurements to hash each phase of the boot process for a similar result.

Intel Boot Guard is a silicon-level technology from Intel that protects the boot process before Secure Boot starts. Once you enable Boot Guard on a system, you cannot disable it.

1.4 Managing Systems and Firmware

Various technologies were developed to help administer server systems without sending IT staff to a data center. Fully physically separate IP-KVM and IP power switch solutions can be more secure, but are also more expensive, so for systems which may need remote management, *out of band* (theoretically) but fully integrated solutions were developed. In order to control a system when it is ostensibly powered off, these solutions require some power and network access to allow the management processor to work. Often these solutions are integrated into, or adjacent to the firmware, or very low level access, effectively below ring zero and run firmware of their own, which has a fairly large attack surface.

IPMI was an early standardization effort in this space, providing built-in, or relatively cheap add-on remote management capabilities via an add-on board. A common security

issue with IPMI is that it is often shipped enabled from the manufacturer with a weak standardized password. While many machines with IPMI have a dedicated management NIC, as the technology has become cheaper and more integrated it is more common to discover IPMI enabled on the primary, standard NIC, or even multiple NICs that can easily be plugged into non-isolated networks.

It is always best to use a separate and isolated network for this sort of management, but in some cases this may be difficult to achieve, particularly as these technologies have started appearing in desktops and laptops.

Intel Active Management Technology (AMT) is firmware that runs on the Intel ME processor, designed to provide remote management capabilities, and sold as a relatively cheap business class upgrade to consumer-oriented CPUs. AMT provides a web service based remote management system. Recently a vulnerability was found where an attacker could logon and the AMT password-checking code would accept any password, systems were vulnerabile from 2009 to 2017 if their AMT network traffic was attacker-accessible, and the commont configuration for AMT is to enable it on the primary, always-on NIC rather than a segregated management network.A major vulnerability was also recently discovered with the Intel ME processor and operating system itself. AMD systems provide a similar technology called Platform Security Processor (PSP). Apple recently added a T2 processor to provide some similar functions to their iMac Pro, and Macbook Pro.

DMTF SMASH and DASH is somewhat similar to IPMI. Multiple AMD systems often use DASH (for Desktops) and SMASH (for Servers).

DMTF Redfish is a standardized replacement for IPMI. It provides an HTTP-based JSON ReSTful management interface.

There are multiple similar vendor-centric technologies, like: HP iLO, Dell DRAC, Sun/Oracle ILOM, AMI MegaRac, etc. UEFI applications, drivers and services can also be run before the UEFI bootloader application loads the operating system.

1.5 Summary

This chapter attempted to summarize some of the more important computer firmware technologies and some of their security issues. For more information, see the NIST SP 800-193 specification's enumeration of platform firmware, as well as the CHIPSEC Project's HAL driver subcomponents (each one is an area of interest for security researchers). Computer hardware and firmware keeps evolving and there are multiple layers of technology involved. Attackers are learning to use firmware as their host, defenders need to learn to protect their computer firmware.

2 Threats Against Firmware

For almost every type of firmware, proof of concept exploits exist. Many types of active firmware exploits have been discovered in the wild, and others have been referenced in document leaks. As firmware increases in sophistication and complexity, so does the available attack surface. Operating systems, while still widely vulnerable, are better understood and protected than firmware by system administrators and security professionals. In some cases, conventional protections and existing security best practices are insufficient to protect against firmware level attacks.

UEFI replaces the older BIOS in most modern Intel/AMD based systems. While there were BIOS based attacks, these typically needed to be written in assembly language by an attacker with a relatively deep understanding of of the target BIOS and system to which it applied. The attack would not necessarily be portable between the targetted system and any other make or model of system, even from the same manufacturer. UEFI is a very standardized platform, allowing many manufacturers to leverage a powerful open source core system called Tianocore https://tianocore.org which in many regards resembles a full blown operating system. Many thousands of lines of code are common among UEFI implementations, with the primary differentiation between systems being standardized modules utilizing a common API, known in UEFI as *protocols*. Modules can be written for UEFI are typically written in C, but UEFI supports several additional languages including Python and even Javascript, so there is less need for low level assembly programming skill. So UEFI presents a much larger attack surface than BIOS. Fortunately UEFI is also frequently implemented along side some significant security improvements such as Secure Boot. But like most security improvements, technologies such as Secure Boot add additional work to manage a system, so they are often disabled.

While UEFI is a distinct case and notably more standardized than other system firmware implementations, some of the same issues apply in other places. The decreased cost, increased availability and power of ARM and other similar microprocessors makes it easier and often cheaper to implement an interface by building in C, or a higher level language for controllers built into PCIe cards, hard disks and SSDs. Often firmware can be updated from the operating system level, allowing field fixes instead of returns for repair (RMA), but also allowing deeply implanted malware, starting with an operating system level compromise. ROMs that can only be updated by the manufacturer are no guarantee of protection, of course - as mistakes can be made, such as returning a laptop to the wrong customer with secrets like *Computrace* (a sort of LoJack for firmware) for a different customer embedded.

Almost all firmware, including every in-production version of UEFI contains closed-source compiled binary blobs, and every step in the supply chain often adds additional binary blobs to a complete system. It is easy to think of a system as being from a particular manufacturer, but even if they actually manufacture the system as opposed to just putting their name on it, there is typically a chain of Original Design Manufacturer (*ODM*) and Indendent Hardware Vendor (*IHV*) upstream of the Original Equipment Manufacturer (*ODM*), not to mention the manufacturers of discrete compontents such as network and disk controllers.

If you're familiar with the principles of open source software for bug and security transparency, you won't find much transparency in the firmware world. Bugs with security implications are likely to remain hidden for much longer, and malware can be

added at any stage when binary blobs are added.

2.1 Types of Firmware Threats

At the firmware level, it is possible to execute almost every type of attack that may be more familiar from the higher level software world. In the most direct case, malware which requires an operating system can be inserted by compromised firmware into the operating system while running, or directly onto non-volatile storage. More subtly, firmware can operate entirely outside of the operating system - in storage (eg: disk, SSD) firmware corrupting or exfiltrating data in transit to/from storage, or operating in a peripheral, enabling a camera while leaving the indicator light disabled.

Examples of all of the following, plus every additional normal software attack have been shown to impact firmware:

- Race Conditions
- Buffer Overflows
- Privilege Escalation
- Parsing Errors
- Misconfiguration

2.1.1 Intel Taxonomy

Intel presented at Black Hat 2017 on firmware security, and included an excellent firmware bug taxonomy - along with an evaluation of the frequency of bug occurrances.

While these could be viewed as variations on standard software bugs, it is useful to consider them as special cases from a firmware perspective. All credit to Intel for this list:

- Inconsistent power transition checks
- Bad initialization state (secrets kept in memory, assumption of certain system state)
- S3 boot script
- Race Condition
- Enabling protection mechanisms (including disabling/enabling certain hardware elements, including wrong module launching order)
- Time of check / time of use (TOCTOU) race conditions
- Trusting input (from less privileged entities in the platform, or having different assets configured by different thraet vectors)
- Measurement failures
- Not measuring certain modules
- Accepting signed updates, but for the wrong platform due to using the same keys
- Accepting to load certain modules in the wront time (when certain assets were not properly protected)
- Failing to identify a measurement error/problem
- Platform capability not properly configured
- Locks not set, devices not properly initialized, features not disabled, etc

- Security of meaningful assets exposed to untrested entities
- eg: UEFI variables
- Hardware misbehavior
- Wrongly defined architecture, micro-architecture, bad design, etc
- Platform components behaving differently than specified

While this list is built with UEFI primarily in mind, it could equally apply to any firmware on a system.

2.2 Firmware Threat Examples

2.2.1 Hacking Team's UEFI Rootkit

UEFI is extremely powerful. Compared to BIOS, UEFI has many more features, and is extensible, which presents a larger attack surface than the older BIOS and smaller boot firmware. Hacking Team was revealed to have a working exploit during the 2015 leaks of their documents. Hacking Team's attack worked against Insyde's UEFI implementation, and as revealed required physical access to the target machine to boot into the UEFI shell and load attack code. The code itself simply injected (or re-injected, if necessary) Microsoft Windows based malware, so would not work against a non-Windows operating system. However, it was shown that Hacking Team provided support in the instance that the attack failed, such as on a non-Insyde implementation of UEFI. Porting the attack code to another UEFI implementation would be a relatively simple matter for an adversary like Hacking Team. Remote installation of the malware would also be fairly easy to manage if necessary, perhaps requiring a reboot into UEFI shell, possibly installed as a UEFI update from within the operating system by another piece of malware. The primary goal of this type of attack is to ensure persistence, even in the event of an operating system reinstall.

2.2.2 BadUSB

The most obvious attack via USB, is a thumb drive with an autoexec, or other malware file on it. When the file is automatically executed, or opened by a user, the malware is implanted into the operating system at the application level. While more complex and challenging, there are multiple firmware attacks from USB devices, given how easy it is for attackers to create their own drivers that lie to the computer about the USB device's function. Devices like the Hak5 USB Rubber Ducky are easily available to attackers.

Attacks on USB device firmware were demonstrated in 2014 for devices using Phison controllers. Attackers can modify a USB storage device to act as an HID (keyboard/mouse input), or to create hidden partitions within the storage. This attack is fairly specific to Phison controllers and would require a similarly sophisticated attacker to build a custom attack for other USB devices.

2.2.3 PCIe & Thunderbolt DMA Attacks

Exposing the PCIe bus to external devices creates a new attack vector for attackers with physical access. A generic device designed for this type of attack is called the PCI Leech

and when plugged into the PCIe bus can exfiltrate data directly from system RAM using DMA, or modify the contents of that RAM to insert malware. The Thunderstrike attack for Macs infects the *PCIe option ROM* - a firmware blob present on many PCIe devices. Using Thunderstrike, an attacker could infect a particular Thunderbolt based device, then whenever that device is used on a new system it would infect the new host system.

2.2.4 Equation Group Hard Disk Firmware Attack

Inserting malware on the disk controller firmware itself allows the attacker to bypass disk encryption, read or corrupt data in real time as it is read from or written to the disk or SSD. This technique could also be used to inject OS level malware by simply corrupting an executable on-disk. The Equation Group was discovered to have used this attack.

3 Firmware Defense

3.1 Ideal Solution

The ultimate solution to the firmware problem is a huge challenge. Many individual designers and manufacturers for even a single motherboard need to be convinced to participate, and to follow best practices and advice from organizations such as NIST. Doing so would require them to spend money on systems development in a very low margin business. Firmware has much lower visibility to end users, and much slower update cycles. Designers and manufacturers would also need to switch from a history of closed source / binary blob and IP protection mindset, to one that is more transparent, if not completely open source, for auditability purposes.

Some experts, such as Joanna Rutkowska of Invisible Things Labs propose to separate and isolate firmware from hardware so that it can be tracked for changes, and to create a stateless computer.

While so many industry currents are against improvement, as a system administrator wielding purchase decision making or influencing power, you do have the power to influence things for the better. In fact, if the people influencing large scale purchasing decisions do not agitate for improvement, it will certainly never happen.

3.2 Real Solution

The practical solutions available to you today involve some modifications to your existing practices, such as hardware lifecycle management, in which security, let alone firmware security may not have been much of a concern to date. Additionally, there are small tweaks to normal security best practices that you are probably already doing to add and adjust for firmware concerns.

The next chapter will go into more depth, and in our end notes we'll have a comprehensive (but succinct) checklist you can use for your organization.

Below is a quick but incomplete summary to start thinking about the issue:

- Follow general security best practices as applied to firmware, including:
- Lock down firmware settings
  - Set password(s)
  - Disable unneeded service(s) (eg: remote management, PXE)
  - Enable Secure Boot - and, in larger deployments consider deploying/managing your own Secure Boot keys
- Add Firmware To Hardware Lifecycle
- Ensure that you are purchasing the most secure hardware you can, given other constraints (budget, etc)
- Before deploying, lock down firmware settings, dump and hash
- When IT puts hands on hardware, check and test firmware
  - for necessary updates
  - dump and hash
  - check hashes
  - use CHIPSEC, FWTS and other tools

- Before disposal, ensure hardware is appropriately wiped - including firwmare. Particularly secrets stored in eg: TPM, changing firmware passwords, and Secure Boot keys.
- Follow NIST Guidelines for firmware
- Use CHIPSEC, Firmware Test Suite (FWTS) and other tools

3.3 Follow Security Best Practices

As an IT professional, you probably already know and follow far more security best practices than the average non-technical individual. Some of these may not apply to firmware at all – but remember, firmware is software, and often it is software with a full network stack and continuous network access. Most existing practices require relatively small, logical modifications to incorporate firmware security also.

The list that follows may miss some, and the authors would welcome suggestions for incorporation into future editions of this book.

Our starting point is the excellent article "5 Tips for Protecting Firmware From Attacks" with input from Yuriy Bulygin and John Loucaides of Eclypsium, formerly of Intel Advanced Threat Research (ATR), and the NIST SP 800-147 standard.

3.3.1 Understand and Communicate That the Threat To Firmware Is Real: Technical Staff & Management Education

By reading this book you'll understand the threat better than most of your peers. Share this book, and share what you've learned with peers and managers. The most immediate step is to ensure that you are aware of which firmware your systems are using, and applying manufacturer firmware updates in a timely manner, which will take cooperation and effort from your IT and security departments.

As these firmware implementations do not typically have internet enabled automatic updating, this task falls to you. In most enterprise environments, large fleets of identical machines make this easier but it is still a large task to track versions of firmware, update availability and deployment. You already have practices for managing OS and app updates such as ensuring that updates get applied in a timely way, and completely, particularly if you allow users to defer updates. Ideally you can integrate your firmware update management with this process, if not the actual firmware updates themselves.

3.3.2 Think Broadly About Firmware

Remember that almost every component of a every modern computing system has firmware. This includes not only laptops and servers with their built in components such as on-board WiFi adapters or hard drives, but also components you may think of as entirely peripheral such as monitors. Remember to include computing devices that are not strictly computers such as network switches and routers. Maintain awareness that almost every electronic device these days is some form of computer, or has some form of computer attached. If not, it probably will soon.

3.3.3 Practice Security Basics

Most firmware can only be modified with physical access, or with root/administrator privileges, so it is important to protect these privileges. Follow defense-in-depth with firewalls, sandboxes, IDS/IPS and OS and application updates.

3.3.4 Physical Security

Physical security is an entire dedicated subject unto itself, with expertise and knowledge since the dawn of human agriculture. So naturally a lot of this skill lies entirely outside of the information technology world. Comprehensively covering physical security, even as it relates to firmware security is far outside the scope of this book. Fundamentally, physical security is about access - ensuring only authorized individuals have access, and recording access, both by authorized individuals and even unauthorized individuals such as during a breakin.

Physical access is one of the more important aspects of firmware security, as many of the types of firmware we are discussing can only be modified with physical access to the machine containing the target firmware, such as through a JTAG port on a motherboard or physical presence at the console during a reboot.

3.3.4.1 Physical Security for Servers

In the case of servers, physical access is typically highly controlled, to best practices. If not, there are many reasons in addition to firmware security to go through physical security steps. It should be noted that physical access protections are necessary but not sufficient for servers, as on most modern systems many firmware ROMs can be modified remotely directly (in the case of remote access IP KVM, IPMI, iLOM, BMC, and Redfish) and more likely indirectly via the operating system and OS level tools.

3.3.4.2 Physical Security for Endpoints

Client endpoint systems including desktops, laptops and mobile devices seem like an impossible problem to solve regarding physical security. Never the less, there are best practices for managing this problem, and in most cases there is plenty of room for improvement in simply following best practices, both for managing physical access to mobile machines, and training the people carrying them.

The most important consideration for firmware security for endpoints is unknown access. The most obvious example is the classic *Evil Maid Attack*, wherein an unattended laptop left in a hotel room is physically accessed by an attacker posing as a hotel employee.

Travel requirements, currently or in the future, may encourage or require people to put their laptops in checked luggage, giving access to airport security and baggage handlers. Even if you are not concerned about state-level threats, some countries may encouage or facilitate corporate espionage at the border, or during a trip, in order to gain access to trade secrets and itellectual property, for a competitive advantage.

The standard security advice for endpoints, such as:

1. Require strong passwords.

2. Use a strong firmware password.
3. Do not give users local administrator access.
4. Use full-disk encryption.
5. Use Secure Boot.

is not sufficient. Once a laptop has been physically handled by an attacker, all bets are off.

There is a more subtle version called *insider affiliate*. A laptop in Alice's possession that regularly leaves the secured corporate office environment could easily be accessed by a friend or family member without Alice's knowledge, while she has for the most part, retained physical possession of the laptop and does not think of it in the same context as the classic Evil Maid attack for a laptop left alone at an unfamiliar location. This drives some important suggestions:

1. Always have plenty of vetted clean (including firmware updates and security scans) shelf-spare laptops on hand so you do not have to interrupt Alice's productivity too much with a swap.
2. Develop, and contiuously improve the laptop swap process to keep it as efficient as possible.
3. Consider adding a laptop swap after any foreign travel.
4. Make firmware scans for updates and security a regular part of any mobile deployment, as you would with viruses and malware. Daily or multiple times per day would not be inappropriate.

The Linux Foundation Workstation Security checklist is helpful in this domain, with specific guidance for firmware considerations, and most advice is equally applicable to other operating systems.

3.3.4.3 Physical Security for Transit

You might find yourself needing to ship machines of any type including servers, laptops, workstations, tablets, etc. Often this will need to be done quickly, for some sound business reason. Physical tamper resistence and detection for shipping packages is well established technology but it is worth some consideration if for example your business typically ships pallets of fidget spinners and you're asking your shipping department to send computer equipment. Intercepting hardware in transit to implant spyware / malware is a known attack pattern (known as *interdiction*), and if you're using full disk encryption or the attacker wants persistence, then firmware is a prime target, particularly given the attacker may be able to get hands on the hardware for long enough to access JTAG ports on the board. So a pre-shipping set of firmware updates, imaging and hashing is in order, along with a post-shipping verification of the same elements.

3.3.5 Golden Images (and hashes)

This may be something you thought you left behind in your IT career for OS installs, but in the firmware world this is still highly relevant. Some firmware implementations may be designed or expected to never change in their service lifetime, even if their contents are updatable via eg: a JTAG port. The manufacturer may only expect to update them during a warranty return, for example or have just added updatability as a just-in-case feature. Even when firmware might be expected to be updated fairly frequently such as UEFI, they have very slow update cycles relative to the operating system and app updates to which you've become familiar.

So, make golden images, and current grade (SHA-256 or better) hashes of as many different firmware ROMs as possible on your systems when you first purchase them, and periodically validate these on every system. Be aware that changing variable contents (eg: boot device order in UEFI) will change the hash. In field use, this sort of change should not be very frequent, and ideally golden imaging and hashing would happen on at least the same frequency. These images and hashes can be stored along side similar auditability information you already save.

3.3.6 Security Scans

Tools such as the open source CHIPSEC project, from Intel, and Firmware Test Suite (*FWTS*) from Ubuntu Linux allow deep security scans of firmware. Ideally every brand new system based on an Intel chip with UEFI firmware should pass every test in CHIPSEC. As a sysadmin, use your purchasing leverage to pressure your vendors about this issue. Meanwhile, you can at least run all of the CHIPSEC tests, and record the information for a new system, and then periodically thereafter. You can use your record of CHIPSEC failures to influence future purchasing decisions also.

3.3.7 End User and Non-Technical Staff Education

Most enterprises are not currently doing nearly enough infosec education of their staff, but physical security of mobile machines is a must. As your current user education probably does need improvement, why not include some training on physical security? If your employer is effectively or actually asking staff to take their laptops offsite, be sure and emphasize a "blameless" approach to a physical compromise incident - it is not the employee's fault that their job demands that their laptop gets exposed to non-employees. An incident is an opportunity for everyone to learn how to improve security, and to refine your incident response plan. Also take the opportunity to advertise your extremely low friction and speedy process for swapping out their possibly compromised laptop with a fully functional replacement.

3.3.8 Add Firmware To Hardware Lifecycle

The most essential component of firmware security by far is adding firmware to the hardware lifecycle. The lifecycle described in detail in NIST SP 800-147 can be used to help augment your existing hardware lifecycle. Current hardware lifecycles typically have little to nothing to do with security. The focus with a traditional hardware lifecycle

is strictly management of the hardware assets as capital assets. At most there is a concern for tracking assets, including if possible, discouraging and detecting when a hardware asset, such as a laptop is stolen outright or goes missing through accident or negligence. Even in the case of theft or loss, the concern for the loss of an expensive asset may dominate over the potential IT security implications.

Adding firmware security to the hardware lifecycle should not involve significant extra labor. The foremost issue is to make golden images and hashes of platform firmware with each brand new system, at the earliest step after unboxing. Ideally these hashes would be verified against hashes provided by the manufacturer, although currently manufacturers do not reliably provide hashes for verification. At this one critical juncture it is possible to capture a baseline. After installing or configuring software on the system, and most especially after connecting the system to a network it is no longer possible to be as certain of the baseline.

The next step in refinement is to add in firmware checks at various points in the lifecycle of the hardware. As system firmware is in fact, software, ideally it would be checked with the same frequency as the operating system and applications, much like anti-virus. This is not currently feasible for the most part, though PreOS Security software should be able to help very soon. In the meantime, why not add firmware re-dumps and scans whenever a system is re-enters IT department control from the wild? It is particularly important to add these checks whenever a system has had a physical access compromise, or when a compromise is suspected.

While a new concern for firmware security may highlight issues with the current hardware lifecycle process, remember that hardware lifecycle, like most IT procedures and practices can always be improved, and there are likely other benefits to improving it. If your CFO hasn't been pestering you about asset tracking recently, you never know when he or she will start, or you'll get a new CFO that insists on it overnight. Why not start following better practices now?

3.4 NIST Guidline Summaries

NIST provides some excellent guidance on firmware security, working with some of the top experts in the industry. The documents are a much longer read than this book, but we try to provide a useful summaries for you. For further reading, it is suggested that you read the abstracts and executive summaries of each of these documents. If you are going to read just one all the way through, start with the most up to date, and comprehensive but still in draft NIST SP 800-193.

3.4.1 NIST SP 800-147 BIOS Protection Guidelines

The overall focus of NIST SP 800-147, published in 2011 is the older BIOS firmware, commonly used on x86 and x86_64 PC systems. However the advice can easily be extended and applied to any platform firmware, particularly those which permit field updates.

Firstly, BIOS updates should use digital signatures to prevent installation of BIOS update images that are not authentic. Secondly, if possible physical presence of an operator should be required for authorized BIOS update installation. There should

be integrity protection features in and around the BIOS to prevent unintentional or malicious updates to the BIOS, including preventing other system components from bypassing the authenticated update mechanism.

The focus of this document might appear to be entirely on OEMs and manufacturers, however it is the responsibility of the system administrator or security professional to ensure that systems purchased and used are compatible with these requirements in such a way that they can actually be enforced. The system operators will also be responsible for integrating these requirements with the hardware lifecycle in such a way that, for example IT staff come into physical contact with mobile systems regularly enough to make the required updates in a timely fashion.

3.4.2 NIST SP 800-147b BIOS Protection Guidelines for Servers

The gaps in NIST SP 800-147 regarding more complex servers, such as blades and network devices with services processors with multiple levels of BIOS or UEFI firmware both for each blade and the chassis itself are addressed in NIST SP 800-147b. In quick summary, all the same protections should apply at each level, including protection from unauthorized updates between each system component.

3.4.3 NIST SP 800-155 BIOS Integrity Measurement Guidelines

In NIST SP 800-155, the concept of roots of trust and secure transmission of integrity measurements are added. While this document is aimed entirely at OEMs, OS vendors and security application vendors, it is worth understanding the capabilities and limitations of systems as a whole, and as implemented in the systems you manage. You may also want to understand this document in order to press your system vendors on these topics.

3.4.4 NIST SP 800-193 Platform Firmware Resiliency Guidelines

The newest document in the area of firmware security, NIST SP 800-193, generalizes discussion to all platform firmware explicitly and includes guidance for OEMs and component/device suppliers to ensure that all system firmware consider and implement security principles. Once again - if you are going to just read one of the NIST guidelines, we recommend you start with SP 800-193.

4 Defense Tools and Examples

4.1 Tools: CHIPSEC for UEFI, BIOS, SMM, ACPI

CHIPSEC is open source software designed at Intel Security Advanced Threat Research, and is one of the authoritative solutions in the platform firmware space. CHIPSEC not only permits dumping and hashing of firmware images, but also deep inspection of loadable modules, variables and other attack surface on the complex UEFI platform firmware. CHIPSEC provides a deep test suite, including brute force security techniques such as fuzzing. While CHIPSEC is extremely complete, it is highly architecture-dependent and works best on Intel x86/x86_64 motherboards from the last decade or so.

CHIPSEC is a tool that anyone integrating or selling Intel-based systems should be using, all the way to any retail vendor that boxes their own systems. While one wouldn't expect Best Buy to unbox a system from an OEM and test it with CHIPSEC before selling it, VARs should be validating that a brand new system passes all CHIPSEC tests. This is not something that happens today, but it is something that you as someone who purchases (or provides purchasing guidance) for many systems can insist upon.

CHIPSEC can be used for both live and offline analysis. In order to do offline analysis, a copy of a ROM (typically called a *ROM.bin* file) is needed. CHIPSEC can generate a *ROM.bin* file for offline analysis, but other tools such as FlashROM, Google Pawn and Apple eficheck can as well.

CHIPSEC is a good example of software that should be available (and open source) for all platform firmware, most particularly those that are designed to be updatable rather than flashed to ROMs.

4.2 Tools: PCIe

Solution options start to get messy for PCIe firmware. Most PCIe firmware are called "option ROMs" or "expansion ROMs" and as the name implies, for simpler PCIe cards they are often ROM chips which can only be updated by replacing the chip entirely. Physical access to the system would be required to execute this attack.

As systems become more integrated, standalone PCIe devices such as add-in cards, and Thunderbolt devices have moved up the value chain and are more complex devices, often with software updatable ROMs. Nobody wants to throw away a $1000 GPU because of a code error in the initialization ROM, or the often complex device operating firmware.

Using Linux, the PCIe bus can be inspected using *lspci*. PCIe option roms can be dumped using *sysfs*, and *fwupd* for Linux and Windows Update firmware updates for Windows are the beginnings of a managed system to provide automatic updates. But be aware that these methods may be far from comprehensive - the option ROM to ensure the GPU functions at boot is only a tiny fraction of the firmware on a modern GPU. Currently only proprietary tools from the GPU vendor can address the firmware on a modern GPU, and the same goes for NICs and RAID adapters.

4.3 Tools: Many More

Many more tools are listed in our Appendix *Awesome Firmware Security*. It is a living document, so be sure and read the latest version to keep up to date, or consider subscribing to updates.

5 Help Improve Firwmare Security For Everyone

As a system administrator, or other IT operations professional, you are uniquely positioned to improve firmware security throughout the IT industry. Of course you can and should assist with awareness, educating fellow IT professionals and management as opportunities arise - perhaps even by sharing this book. However the greatest lever for change is the purchasing power you wield. No individual consumer can ever have the types of conversations with system VARs and OEMs that you can on a regular basis. Your average salesperson will not necessarily know what you're talking about, but for most purchases you will have access to a more technical sales engineer. It is your responsibility as a system administrator to guide your employer towards the best IT purchases which may not necessarily be the cheapest available. It is your job to balance all of the factors involved for high dollar purchases that are a major capital expenditure for your employer which will be used by your fellow employees for many years.

All we advocate in this book is that you consider hardware and firmware level security as one of those many purchasing factors, and ask questions and use leverage with your vendors accordingly. Consider opening up your process to additional vendors which prioritize firmware security by providing verification hashes and signed updates via standardized mechanisms (Windows Update *and* LVFS fwupd), and full open source. At least express concerns for all of the closed source, unverifiable firmware components included with your current vendors hardware.

Like many issues that face IT professionals, you may not make an immediate improvement for your own career or work environment, but you can make a positive change for future IT professionals everywhere, and more broadly for everyone. There may be no system today that you can purchase that comprehensively provides software tools, verifiability and security for all the firmware with which it ships. But that does not need to be the case in a few years.

Manufacturers and OEMs face their own challenges, with the biggest being the very small margin on computer hardware. All improvements must somehow be extracted from that small margin. Firmware presents a special challenge as manufacturers are unlikely ever to touch a system again once it has shipped, so field updates need to be supported. There will probably be some push back if you start demanding change, but it is worth the effort, and should pay back in the rest of your IT career, and for your fellow IT professionals.

6 Conclusion

Platform firmware provides a platform for attackers to install persistent malware out of view of current conventional anti-virus and malware detection software. Firmware has increased in complexity and capabilities, and therefore also in attack surface, and awareness of the problem remains low.

As a system or network administrator, or member of a blue team you've increased your awareness of this problem and now you can help educate others, begin to implement defenses and pressure your VARs and OEMs for more security compliant hardware.

6.1 The PreOS Security Solution

The first step is integrating imaging and hashing of all possible platform firmwares into your hardware lifecycle immediately after unboxing, and periodically throughout the life of machines, particularly after a suspected physical compromise or OS level malware. PreOS Security makes this process and more easy with our software client and web dashboard, and we'll work with you to integrate our client with your single-pane system of choice.

You can also stay up to date on the latest developments in firmware security with a concise summary through our quarterly newsletter:

https://preossec.com/newsletter

We'd like to continuously refine the content of this book, so if you've got feedback, send it to us at:

editor@preossec.com

Appendix A: Awesome Firmware Security

https://github.com/PreOS-Security/awesome-firmware-security/

Awesome Firmware Security is a curated list of platform firmware resources, with a focus on security and testing. Created by PreOS Security.

ObDisclaimer: Firmware is software. But, while you can wipe and reinstall software on your hard drive, it is possible to brick your system when working with the firmware. Use care, and proceed at your own risk.

NOTE: IoT / embedded operating system security is not included, unless they happen to overlap with platform security, such as Intel AMT, AMD PSP, Redfish, IPMI, BMC, OpenBMC. There are already awesome IoT/embedded operating system lists. eg: Awesome IoT

Technologies and Terminology

Each of these technologies are awesome in their own right, and we'll make a standalone awesome list for them at some point. Meanwhile, they form our index.

- ACPICA - The ACPI Component Architecture Project (ACPICA) provides a collection of cross-platform ACPI tools, such as acpidump.
- ACPI - ACPI is a platform firmware technology, originally intended to replace Plug and Play, MP, and Advanced Power Management. The UEFI Forum owns the spec and maintains an awesome list of ACPI-related documents.
- ACPICA - The ACPI Component Architecture Project (ACPICA) provides a reference implementation, and a collection of cross-platform ACPI tools, such as acpidump.
- ARC - ARC (Advanced Computing Environment) is a platform firmware technology used by early Windows NT non-Intel systems. The design of ARC was influential to the design of UEFI: firmware images on a hard disk partition, pointed to by variables.
- BIOS - BIOS is a platform firmware technology initially used on the Intel-based IBM PC. It is an 8086 Real Mode technology. Intel has said that they will end-of-life BIOS-based platform firmware by 2020, replacing it with UEFI. Intel and a few IBVs have closed-source BIOS implementations. BIOS used to be the main firmware technology on Microsoft Windows PCs, until Windows started requiring UEFI.
- SeaBIOS - The primary open source BIOS implementation.
- coreboot - coreboot is a platform firmware technology, originally called LinuxBIOS. It loads payloads such as SeaBIOS, UEFI, among others. Widely used in embedded systems. Coreboot is used by Google on ChromeOS systems, using coreboot Verified Boot for additional security.
- Direct Memory Access - DMA allows certain hardware subsystems, most notably PCIe to access main system RAM, independent of the central processing unit (CPU). Attackable by rogue hardware such as PCIleech. The primary protection is iommu hardware and operating system support.
- Heads - Heads is a platform boot firmware payload that includes a minimal Linux that runs as a coreboot or LinuxBoot ROM payload to provide a secure, flexible

boot environment.

- Independent BIOS Vendor - An Independent BIOS Vendor (IBV) provides an integrated firmware solution to OEMs/ODMs. With UEFI replacing BIOS, some IBVs now refer to themselves as IFVs, Independent Firmware Vendors. Some OEMs will outsource their consumer-class device firmware to IBVs, and do their own firmware for their business-class devices. Examples include:
- AMI
- Insyde
- Phoenix
- Intel Boot Guard - Intel Boot Guard is a firmware security technology that helps secure the boot process before UEFI Secure Boot takes place. Once Boot Guard is enabled, it cannot be disabled and prevents the installation of replacement firmware such as coreboot.
- JTAG - JTAG is a hardware interface to chips that allows access to the firmware. It is used by firmware engineers during devlopment, and by Evil Maid attackers when the vendor leaves the JTAG interface exposed in consumer devices.
- LAVA - LAVA is an automated validation architecture primarily aimed at testing deployments of systems based around the Linux kernel on ARM devices, specifically ARMv7 and later.
- LinuxBoot - LinuxBoot is a platform firmware boot technology that replaces specific firmware functionality like the UEFI DXE phase with a Linux kernel and runtime.
- Management Mode - Management Mode is term used by UEFI to refer to both Intel SMM and ARM TrustZone. A privileged execution mode of the CPU.
- Management systems are implemented on a separate processor, and often a dedicated network interface, facilitating out of band access and control. In some cases such as Intel ME and AMD PSP the management processor is on the same die as the primary CPU. These systems often use a full embedded OS, such as BSD or Linux.
- AMD PSP - The AMD PSP (Platform Security Processor) is a security processor on AMD systems, which runs firmware applications such as fTPM.
- Apple T2 - System management controller, image signal processor, SSD controller and secure enclave for encrypted storage and secure boot for the imac pro.
- Baseboard Management Controller A BMC is an interface to manage server firmware, including applying updates. OpenBMC is the main open source BMC implementation.
- DASH - DMTF DASH is an out-of-band firmware management specification for desktops. Intel AMT is a compliant implementation of DASH, as is AMD SIMFIRE.
- Intel AMT - Intel AMT is a platform firmware management technology on Intel systems, running on the Intel ME processor as an application. AMT provides services such as remote KVM, power control, bare-metal OS restore and re-imaging, and remote alerting.
- Intel ME - Intel ME is a management and security processor on Intel systems, which runs Intel Active Management Technology AMT, Advanced Fan Speed Control, Boot Guard & Secure Boot, Serial over LAN and firmware-based TPM (fTPM). Appears to run a variant of MINIX.
- IPMI - IPMI is a platform firmware management technology, typically on Intel or AMD server systems. Often implemented as an embedded Linux. While

widely-used, the modern replacement for IPMI is Redfish.

- OpenBMC - The OpenBMC project is a Linux distribution for embedded devices that have a BMC.
- Redfish - DMTF Redfish is an out-of-band firmware management technology, replacing IPMI
- SMASH - DMTF DASH is an out-of-band firmware management specification for servers, similar to DASH.
- Measured Boot - Intel technology using TCG TPMs to secure the boot process.
- Microcode - Microcode is a form of firmware for the CPU. Systems need microcode updates just like they need platform firmware updates, and OS updates.
- NIST - a standards-setting body for the US government. Has several security for design and operations relating to firmware in Documentation, Books and Training
- Original Equipment Manufacturer - An OEM builds and sells original hardware.
- Original Design Manufacturer - An ODM builds hardware and sells them to OEMs.
- Operating System Vendor - An OSV is an Operating System Vendor, which includes firmware/OS interactions.
- Option ROM - An Option ROM, aka an Expansion ROM, aka OpROM, aka XROM, is the firmware 'blob' of a PCI/PCIe device. An Option ROM is terminology from BIOS era, when a card would hook the BIOS platform firmware and add additional functionality for the new card. An Option ROM is a BIOS/UEFI driver on the card's flash. A card may need multiple drivers, one for each architecture and one for each platform firmware type (BIOS+x86_64, BIOS+ARM, UEFI+x86_64, UEFI+ARM, etc). Option ROMs do not account for all of the firmware on such a device, as the operating firmware for the device function such as RAID, or TCP offloading may be entirely separate.
- PCIe - PCIe is the interface for PC boards. PCIe devices include Option ROMs of firmware. The device may have a processor invisible to the system board, it is difficult to fully trust PCIe hardware.
- Secure Boot - Secure Boot is a term often associated with UEFI Secure Boot, an optional security feature of UEFI that helps secure the boot process. It does not require a TPM. Besides UEFI, other firmware technologies also use the term Secure Boot, sometimes in lower case. The Apple EFI-based Secure Boot implementation is not the same as the Secure Boot technology used by Windows/Linux systems.
- SMM - Systems Management Mode (SMM) is a processor mode in Intel and AMD systems, separate from Real and various Protect Modes, that gives full control of the processor. SMM-hosted applications, such as malware, is invisible to the normal Protect Mode-based code.
- SPI - SPI is an interface to accessing the firmware. Used by vendors during development, and used by attackers if left enabled in consumer products.
- Trusted Execution Environment - also known as Secure Execution Environment (SEE). An example of a hypervisor or other technology that constrains firmware to be more secure. ARM TrustZone is an example of a SEE.
- Thunderbolt - a external peripheral hardware interface developed by Intel and Apple. Combines PCIe, DisplayPort and DC power.
- Tianocore - Tianocore is the home to the UEFI Forum's open source implementation. Vendors use this code, along with closed-source drivers and value-added code.
- TrustZone - TrustZone (TZ) is a firmware security technology used on ARM systems, a form of TEE/SEE, called Management Mode by UEFI.

- TPM - A TPM is the root-of-trust for many platform firmware implementations, such as Intel/AMD BIOS and UEFI systems. TPM is defined by the Trustworthy Computing Group (TCG). There are discrete TPM chips, as well as "soft" firmware TPM implementations called fTPM, provided by eg: Intel ME, AMD PSP.
- Trusted Boot - Trusted Boot is a firmware security technology from the Trustworthy Computing Group, which uses TPMs to help secure the boot process.
- Trustworthy Computing Group - Trustworthy Computing Group (TCG) is an industry trade group that controls the TPM and related specifications.
- U-Boot - U-Boot loads payloads such as SeaBIOS, UEFI, among others. U-Boot and coreboot are widely used in embedded systems.
- UEFI - UEFI is a platform firmware technology initially created by Intel, now used by Intel, AMD, ARM, and others, which was initially designed for the Intel Itanium and as a replacement for BIOS. UEFI is also EFI. UEFI-based platform firmware technology is often referred to as BIOS, with the older BIOS called Legacy Mode or CSM (Compatibility Support Mode).
- UEFI DBX - The UEFI DBX UEFI Secure Boot blacklist file contains the latest UEFI Secure Boot PKI blacklist/expired keys. Check your vendor documentation to see how your system's vendor tools work to obtain and apply this to your system; if the vendor has no tools, ask them to provide them.
- UEFI Forum - The UEFI Forum is an industry trade group that controls the UEFI and ACPI specifications, the UEFI SCT tests, and provides the Tianocore open source UEFI implementation.
- USB - Universal Serial Bus (USB) is an industry standard for external peripheral devices. USB devices can be configured to be multiple devices, and rogue USB hardware like Hak5's Rubber Ducky can trick naive operating systems.
- Verified Boot - Verified Boot is a firmware security technology from Google, that helps secure the boot process. Roughly equivalent to Secure Boot.
- Android Verified Boot - Android version of Verified Boot
- ChromeOS Verified Boot - ChromiumOS and ChromeOS version of Verified Boot.

Threats

- BadBIOS - BadBIOS is the alleged firmware malware reported by Dragos.
- Evil Maid Attack - The Evil Maid attack is perhaps the most well-known firmware attack, where the victim leaves their sstem unattended and an attacker has some period of time with physical access to the system, for them to install firmware-level malware. For example, person leaves their laptop in their hotel room while out for dinner, and the attacker is posing as hotel room service.
- Hacking Team UEFI Malware - acking Team is a company that sells exploits to governments and others. Amongst their offerings is a UEFI-based firmware attack for Windows PCs. The Hacking Team malware is one of the few existing known public UEFI blacklisted by CHIPSEC.
- Fish2 IPMI Security - a compilation of information about poor and/or insecure IPMI implementations.
- PCI Leech - PCILeech is PCI-based rogue hardware used to attack PCI interfaces of systems. Defense is iommu in combination with operating system iommu support.

- Rowhammer - Rowhammer is a new form of memory-based security attacks against systems. Defense is ECC memory.
- ThinkPwn - ThinkPwn is a UEFI malware PoC that originally targets ThinkPad systems. The ThinkPwn malware is one of the few existing known public UEFI blacklisted by CHIPSEC. Thinkpwn.efi is included in FPMurphy's UEFI Utilities, one malware binary amongst other useful tools, be careful if using those tools.
- USB Rubber Ducky - a Rubber Ducky is an example of rogue USB hardware, which lets the user configure the system to trick naive operating systems into thinking it is any number of devices.

Tools

ObDisclaimer: Reading about firmware is one thing, but using these tools can be dangerous. You can brick your system - proceed with caution and at your own risk.

Open Source

NOTE: For security and safety purposes, open source software is auditable and verifiable. But beware that working with firmware is more dangerous than working with software installed on a hard disk you can wipe and reinstall. You can brick your system! Proceed with caution and at your own risk.

- ACPICA tools - provides tools and a reference implementation of ACPI.
- acpidump - Cross-platform OS-present tool from ACPICA to dump and diagnose ACPI tables.
- BIOS Implementation Test Suite - The Intel BIOS Implementation Test Suite (BITS) provides a bootable pre-OS environment for testing BIOSes and in particular their initialization of Intel processors, hardware, and technologies. It includes a CPython compiled as a raw BIOS application.
- DarwinDumper - DarwinDumper is an open source project which is a collection of scripts and tools to provide a convenient method to quickly gather a system overview of your OS X System.
- Eclipse UEFI EDK2 Wizards Plugin - This Eclipse plugin helps EDK2 developers use the Eclipse IDE with CDT for doing UEFI development.
- EFIgy - Duo Security's EFIgy is an open source Apple Mac-centric tool that checks if the system has the most up-to-date EFI firmware.
- Firmadyne - Firmadyne is an automated and scalable system for performing emulation and dynamic analysis of Linux-based embedded firmware.
- Firmware.re - Firmware.RE is a free service that unpacks, scans and analyzes almost any firmware package and facilitates the quick detection of vulnerabilities, backdoors and all kinds of embedded malware.
- GRUB - GRUB is a Multiboot boot loader. It compiles as a BIOS or a UEFI application.
- Linux Shim - The Shim is a UEFI boot loader, which loads another UEFI boot loader, perhaps with a different license, and signed by another vendor. There are multiple forks of Shim in the wild.
- Fedora Guide to UEFI Secure Boot Shim

- Linux Stub - The Linux kernel can be built so that the kernel is both a BIOS and a EFI boot loader.
- CHIPSEC - CHIPSEC is a security tool created by Intel, to test the security posture of Intel BIOS / UEFI. Currently the only tool that can check for multiple public firmware security vulnerabilities.
- eficheck - sadly lacking an awesome link for this, as this tool is only available on recent versions MacOS, and not documented at https://apple.com. Verifies UEFI integrity and security.
- Firmware Test Suite - FirmWare Test Suite (FWTS) is a collection of firmware tests created by Canonical, the Ubuntu Linux OSV, to help test a system for defects that will cause Ubuntu problems. FWTS is a suite of dozens of tests, for multiple technologies. The UEFI Forum recommends FWTS as the main ACPI test resource. FWTS is a command line tool for Linux, and includes an optional CURSES UI, and an optional FWTS-live live-boot distribution. FWTS is included in Intel's LUV Linux distribution.
- FlashROM - FlashROM is a Linux/BSD-centric utility a utility for identifying, reading, writing, verifying and erasing flash chips. It is designed to flash BIOS/EFI/coreboot/firmware/optionROM images on mainboards, network/graphics/storage controller cards, and various other programmer devices. Partial Windows support is available.
- Golden Image - A golden image is the vendor's original binaries for the firmware. The term is also used for OS images. Better vendors provide images and tools to reset used hardware/grey market acquisitions to a known state. Before trusting any downloaded binary, such as a golden image, it should be compared to a hash. Most vendors do not provide a hash for their images.
- Linux UEFI Validation - LUV is a Linux distro created by Intel to test UEFI implementation of OEMs. It bundles CHIPSEC, FWTS, and other firmware tests. LUV is available in binary form as LUV-live, a live-boot distribution.
- Linux Vendor Firmware Services - aka: LVFS or fwupd, a firmware update service for Linux OEMs. AWESOMELY provides a standardized system. OEMs that use this are taking Linux compatibility and security seriously. On Microsoft Windows, a similar approach works through Windows Update.
- Microsoft Windows Update - surprise - Windows Update is awesome! In addition to doing OS software-level updates, Windows Update can do firmware updates via standardized capsules. These updates must be verified by the firmware / hardware vendor, and can be EV signed.
- Pawn - Google Pawn is a Linux-centric online firmware tool that dumps the platform firmware image to a file, for later offline analysis.
- PhoenixTool - PhoenixTool is a third party freeware to manipulate (U)EFI and few leagcy bios based firmware blobs.
- rEFInd - rEFInd is the successor to rEFIt, a UEFI boot loader that lets you select multiple operating systems.
- RU.EFI - RU.EFI is a third-party freeware firmware tool that has multiple features. It works as a MS-DOS or UEFI Shell utility.
- RWEverything - RWEverything (RWE) is a third-party freeware firmware tool that has multiple features. The tools works on Windows. The CHIPSEC tool, if the CHIPSEC Windows kernel driver is not loaded, can use the RWE kernel driver.
- Sandsifter - Sandsifter is an x86 fuzzer.

- UEFI Utilities - UEFI Utilities is a collection of UEFI Shell utilities that provide system diagnostic information. (It also includes a copy of ThinkPwn.efi, be careful.)
- UEFI Firmware Parser - UEFI Firmware Parser examines firmware 'blobs', mainly UEFI ones.
- UEFITool - UEFITool is a GUI program that parses firmware 'blobs', mainly UEFI ones. In addition to the UEFITool Qt GUI tool, the UEFITool source project also includes a handful of non-GUI command line tools, including UEFIDump. UEFITool has two source trees to be aware of, master and new-engine.
- Visual UEFI - Visual UEFI is a plugin for Visual Studio that lets Visual Studio users do UEFI EDK2 development without having to know the details of the EDK2 build process, which is not like the Visual Studio build process.
- zenfish IPMI tools - IMPI security testing tools by Dan Farmer of SATAN fame.

Closed Source

SPECIAL NOTE: Closed source software is not auditable or verifiably safe or secure. The risk of malware is higher, and with firmware the risk of bricking your system is always present. These are potentially "awesome" tools, but this list does not constitute a recommendation for use. Proceed with caution and at your own risk.

Documentation, Books and Training

- Beyond BIOS - Beyond BIOS: Developing with the Unified Extensible Firmware Interface, Third Edition. Book on UEFI by Intel and other UEFI Forum members. Originally published by Intel Press.
- Darkreading Firmware Security Tips - This article, which has input from the Intel CHIPSEC team, gives basic high-level guidance for firmware security. Start with this, before digging into the NIST documents.
- Firmware Security Blog - Source of firmware security and development news and information, with a focus on UEFI-centric platform firmware. (DISCLAIMER: One of the awesome-firmware authors, and PreOS employee is the Firmware Security blogger.)
- Firmware Security Twitter List - Jacob Torrey hosts this list on Twitter, which contains many of the core firmware security researchers.
- Hardware Security Training - The Hardware Security Training company is a collection of multiple hardware/firmware security trainers.
- Harnessing the UEFI Shell - Harnessing the UEFI Shell: Moving the Platform Beyond DOS, Second Edition. Book on UEFI by Intel and other UEFI Forum members. Originally published by Intel Press.
- Intel Security Training - training from the CHIPSEC team at Intel Advanced Threat Research (ATR) team of Intel Security. The documents are an AWESOME source of information about Intel hardware/firmware security threats, focusing on UEFI and related technologies.
- IPMI Security Best Practices - best practices for IPMI security from Dan Farmer. In need of an update. Most would apply to Redfish, or any OOB management technology.

- Linux Foundation Workstation Security Policy - The Linux Foundation has a collection of IT Policies, for Linux systems, it includes some firmware security guidance.
- Linux on UEFI - Linux on UEFI Roderick W. Smith has an online book with information on UEFI and Linux, showing how to use multiple boot loaders.
- Low Level PC Attack Papers - an awesome timeline of hardware/firmware security research.
- NIST - firmware guidance documents. These are awesome. Start with SP 800-193
- SP 800-147 - an older document, aimed primarily at BIOS.
- SP 800-147b - an addition to SP 800-147 specifically for servers.
- SP 800-155 - note this standard is still in draft status, but it is still quite useable
- SP 800-193 - note this standard is still in draft status, but quite useable and the most modern of all the documents. Start reading here.
- NSA Common Criteria for PC BIOS Protection - This 2013 Common Criteria Standard Protection Profile (PP) for PC firmware. Addresses the primary threat that an adversary will modify or replace the BIOS on a PC client device and compromise the PC client environment in a persistent way. There aren't any firmware solutions taht meet this profile, but reading the threat model is useful background.
- One-Stop Shop for UEFI Links - One-Stop Shop for UEFI/BIOS Specifications/Tools Maintained by UEFI.Tech Community
- Open Security Training Intro to BIOS & SMM - Introduction to BIOS and SMM course materials.
- Rootkits and Bootkits - This is the only book on firmware security at the time, writen by firmware security experts.

Appendix B: Platform Firmware Security for Enterprise Guidance Summary Checklist

A checklist summary of book text and references, particularly applicable NIST standards. When you pick up this list, you'll partway through the *Hardware Lifecycle* for all of your current hardware. We recommend you simply start implementation at the beginning (Policies and Procedures), and apply the entire checklist to your next naturally scheduled hardware purchase. Then, as time permits, proceed through the process for older hardware, starting with hardware in the most sensitive positions eg:

- Officer, Finance and Legal laptops
- Servers subject to *PCI*, *HIPAA*, etc

This may trigger some unscheduled purchases for security reasons, but the older hardware can be reused in less sensitive applications.

Meta: Policies and Procedures

[] Review and update existing corporate security policies to include firmware security.

[] Review and update existing corporate security procedures to include firmware security.

Pre-Acquisition Research Phase

Include hardware and firmware in threat modeling. Think broadly about firmware. A computer includes multiple firmware images: microcode, multiple platform firmware images (UEFI or BIOS, ACPI, etc.), firmware on each adapter card, firmware on each peripheral and its adapter cables. Besides 'bare-metal' firmware, virtualization technologies have virtual firmware drivers that can also be attacked; virtual firmware has it's own set of attacks, some shared with bare-metal firmware, some unique to the VMM implementation, some shared with traditional app-level attacks. In addition to firmware 'blobs' stored in flash and NVRAM, UEFI stores firmware as files on the FAT-based disk partition, EFI System Partition (ESP), potentially editable by attackers on the ACL-less FAT volume.

Plan to purchase secure and securable systems and to secure the entire system at all levels: hardware, firmware, and software. Read entire guidance document and plan for implementation.

Educate technical staff to the firmware threat. Secure systems for non-technical users, and train them about evil maid attacks - particularly with mobile compute devices.

Add firmware to Hardware Lifecycle: System Administrators should integrate NIST 147, which provides Secure Firmware Lifecycle, to augment existing enterprise hardware lifecycle. Digital Forensic and Incident Response (DFIR) teams should expand their checkslists to include contacts with firmware vendors, OEMs, IBVs, NIST 147 is a starting point.

Ensure availability of, test and learn to use vendor specific firmware tools, including golden image hashes, and signed updates.

Update news and training sources for firmware, including general bug fixes, and critical security updates. Ensure that coverage and updates are fit to purpose.

Request security data from vendors. Source this data yourself eg: acquire a single example of a new model and ensure it passes CHIPSEC security tests.

Set company policy: * For security features that must be available in newly-acquired systems. eg: Secure Boot, Verified Boot, TPMv2, CHIPSEC tests. * Against grey market hardware. * For security features that must be configured in new systems.

Consider currently deployed systems. Are they fit to purpose? Should some new hardware purchases be accelerated, and older systems redeployed to less sensitive applications?

Provisioning Phase

Configure firmware security settings based on company policy. Eg, enable/disable VT/TPM/TXT, WakeOn features, firmware/BIOS password, firmware network ability, Secure Boot, etc.

Ensure all vendor firmware is up-to-date, with up-to-date keys for any signed code.

Register endpoint identity and BIOS integrity information in system inventory

During initial system acquisition, make your own initial 'golden image' of the platform firmware, save the file for future image comparisions, as initial baseline. If vendor provides their own golden image, compare your image with their image, or hash.

Use built-in hardware/firmware security features: Use existing vendor firmware security features, TPM, UEFI Secure Boot, Verified Boot, Measured Boot, Trusted Boot, Heads, etc.

Use signed firmware images: For UEFI, along with Secure Boot, only use firmware that has been properly signed. Understand the code signing process and where gaps may exist – eg, some certs are given to closed-source binaries, no source code audit permitted.

Operations and Maintenance Phase

Keep firmware up-to-date: In addition to OS and app software, also update the firmware, ideally via signed, automated mechanisms such as Windows Update, Linux *fwupd (LVFS)*.

Apply latest Secure Boot blacklisted key database from UEFI.org and Microsoft.com.

Verify firmware updates with hashes.

Perform periodic firmware security scans. Download firwmare blobs and check for unexpected changes with hashes.

Consider "IT hands on" interactions with end user equipment an opportunity to run automated firmware-level security tests.

Monitor all network traffic from/to firmware.

Audit OS-level code which instigates firmware updates. If hardening OS/applications, consider restricting access to tools that perform firmware updates to authorized accounts.

During a security incident, use traditional malware analysis techniques to look for firmware-level malware, and add firmware specific tools such as UEFITool, ACPItool.

During a security incident, the response should include re-obtaining firmware images, and re-running firmware security tests (eg, CHIPSEC), to compare with previous images/results.

Watch vendor security advisory sites of all vendors in supply-chain, for security advisories, and new detection tools to use. Main CPU, TPM vendor, OS vendor, OEM, all IHVs, etc.

Recovery (Incident Response) Phase

Use forensic tools to compare current rom image to previous scans for changes.

Re-run security tests, compare to last-known-good results, look for signs of an attacker. For example, SPI protections were disabled, but now are enabled.

If system is compromised with OS/app-level malware, it may have updated firmware. Validate the entire system, firmware and OS/level tests. Restore a system using both firmware and OS 'golden images'.

After a security incident, Incident Response will need additional time to cleanse a system of firmware-level malware. In some cases, the system may be bricked and not recoverable.

After a security incident, the IR team postmortem analysis should include which vendors have inadequate tools/images for firmware recovery, and discard those systems, or redeploy them to less important roles.

Disposition Phase

Restore firmware to factory reset before disposition. Before disposing of a system, in addition to sanitizing disk media, reset the firmware to a known-good state, using golden image from vendor.

Sanitize any PII in firmware before disposition. Before disposing of a system, ensure that any PII stored in firmware, user authentication data, TPM secrets, etc. are reset. Ask vendor how to restore any PII stored by firmware.

Appendix C: Author Biographies

Paul English

Paul English is CEO of PreOS Security Inc. Paul has Bachelors in Computer Science from Worcester Polytechnic Institute obtained in 1998. And Paul has been a UNIX & Linux system administrator and wearer of many other IT hats since 1996. More recently he has managed a few people while still racking the occasional server. From 2014-2017, Paul was a Board member for the League of Professional Systems Administrators (https://lopsa.org), a non-profit professional association for the advancement of the practice of system administration.

LinkedIn: https://www.linkedin.com/in/englishpaul/

Twitter: [@penglish_PreOS](https://twitter.com/penglish_PreOS)

Lee Fisher

Lee is CTO of PreOS Security Inc. He started his IT career as a sysadmin, on VAX/VMS and AT&T Unix systems, working nights to pay for his way through college. Lee spent many years at Microsoft on multiple systems product teams, including releasing 'Debugging Tools for Windows' (the new Windbg), the 'Windows NT HAL Kit', and the 'Windows NT IFS Kit'. Lee co-ran the Microsoft Porting Lab, where he helped most of the OEMs and IHVs with their driver port to Windows NT. Lee brought a variety of open source projects – such as NCSA Mosaic – to Microsoft campus to help them with the Windows ports of their code. Lee worked in multiple roles in the IT industry, but prefers to write security/system tools in C. Prior to starting PreOS, he was consulting with a few specialty kernel drivers for a few friend's companies.

Lee spoke at a few dozen conferences, including: Security BSides Portland'16, LinuxFest NorthWest, ToorCon Seattle, and Microsoft WinHEC.

Lee's personal blog can be found at: https://FirmwareSecurity.com/

LinkedIn: https://www.linkedin.com/in/lee-fisher-b80b14143/

Twitter: [@leefisher_PreOS](https://twitter.com/leefisher_PreOS)

Appendix D: Errata

Errata

Improvements

- More illustrations and diagrams
- motherboard with names/arrows pointing to all the attack vectors
- hardware lifecycle flow chart
- add direct connect PCI, USB, JTAG attack vector examples