基础架构安全弹性技术指南草案（固件安全篇）

alpha 预览版

Version 2022.5.17

前言

本指南的其他内容可能涉及专利，本文件的发布机构不承担识别这些专利的责任。
本指南起草单位：赛博堡垒（HardenedVault），中科院软件所
本指南主要起草人：Shawn Chang（赛博堡垒），张楠(中科院软件所)

本指南专家组成员：
韩文报，海南大学计算机与网络空间安全学院
李清宝，信息工程大学
张先国，中电科第三十研究所系统工程部
霍玮，中科院信工所
魏强，信息工程大学
方华，中国网络安全联盟
杨秋松，中科院软件所
黄辰林，国防科技大学计算机学院
钟益民，上交所技术有限责任公司
冯涛，上交所技术有限责任公司
成松林，北京战儒科技有限公司
高咏涛，中国船舶集团有限公司第七〇五研究所
吴龙飞，大唐高鸿信安
李冰，浪潮工业互联网产业股份有限公司

简介

技术指南目标

从抽象层级的角度来看，现代计算系统的架构顶层是软件，由操作系统和应用程序构成，尽管它们提供了用户所使用的大部分功能，但它们依赖于底层的支撑。本文档总体性地称其为平台。平台包括硬件和固件组件，它们是初始化组件、启动系统以及提供由硬件组件实现的运行时服务所必需的。平台固件的设计与实现，及其相关配置对于计算系统的可信度至关重要，固件中的大部分在系统架构中具有高权限，由于固件是系统运行所必需的，维修此固件可能具有挑战性，针对平台固件的攻击可以造成巨大影响，比如当处于 SPI Flash 中的固件永久受损时必须重新编程烧写，这对

于数据中心管理来说会造成极大的成本，更糟糕的情况是高级恶意攻击可能尝试向固件中注入持久的恶意软件、修改关键的底层服务以破坏其运行、窃取数据或者以其他方式影响计算机系统的安全状态。

平台固件已经经历了快速的进化和扩展，如同软件那样，它朝着抽象层级更多的和代码更加复杂的方向发展，这增加了代码行数以及攻击面，同时有更多的方面需要防护。固件的威胁是显而易见的，2021 年已经分配 CVE 编号的平台固件漏洞有数百之多，本文档的目标是为机构，企业中的数据中心管理人员，系统管理员，安全工程师在面对高度复杂的固件威胁时提供一份技术指南作为风险评估的参考。

受众

本文档的受众包括计算机系统平台设备厂商，其中包括客户端、服务器和网络设备的制造商，也可为小型机房和数据中心的运营机构提供采购，部署，技术评估等方面的参考，技术指南假设读者理解计基本系统安全知识和计算机架构。

适用性和范围

固件是一种特殊类型的软件，主要用于硬件底层的交互和控制，固件的范畴过于广泛，不同领域的固件牵涉的安全防护的问题差异较大，本文档适用于通用计算机系统，比如服务器，笔记本和台式机的上运行的固件，在某些特定场景下（例如下一代固件架构）本文档素材也适用于嵌入式平台，但这不在本文档的讨论范围。

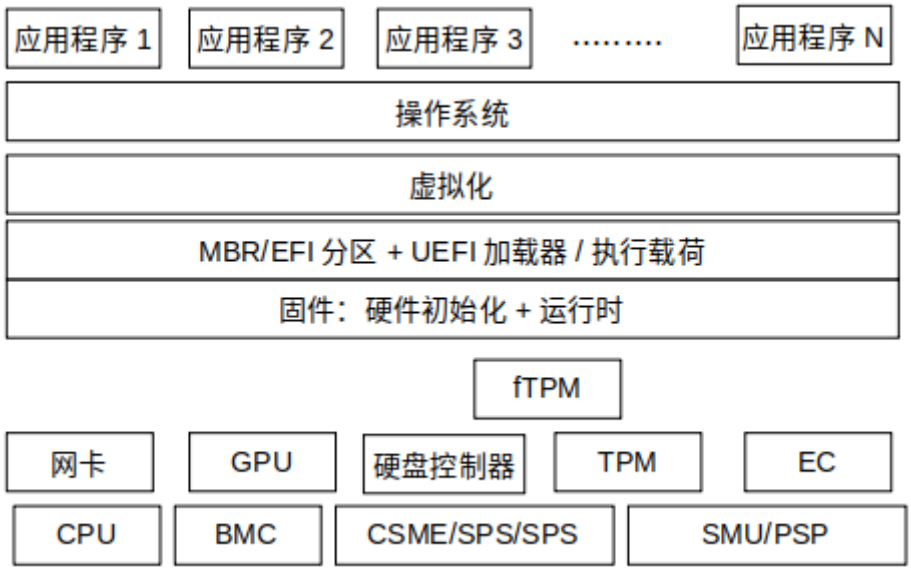
平台固件定义

本文档将探讨平台固件（platform firmware）安全，并且可能会可互换地简称为固件（firmware）。固件（firmware）这一概念拥有多种定义，在手机/嵌入式/物联网设备的上下文中，大多时候固件指所有软件（操作系统和应用程序），手机/相机/游戏机的固件更新通常是整个基础软件的更新，这比通用计算设备上单独更新应用程序会具备更高的风险，在这些设备上的固件更新通常包括校验和以及多重引导程序能力以避免使得设备异常不可用（例如固件损毁“变砖”），在这个背景下，嵌入式领域的固件大多时候由 Linux 或者 BSD 变种构成的独立的操作系统。与之相反，本文档讨论固件的范围限于 UEFI、ACPI 、PCI OptionROM、CSME 等平台固件，值得注意的是，任何复杂到同时拥有操作系统和应用程序的移动/嵌入式/物联网设备很可能不仅仅拥有一种固件，而是拥有若干种平台固件（platform firmware）用于板载的微控制器或者处理器，例如，一台物联网网关设备可能由一个运行 Linux 系统主处理器、一个用于安全验证的处理器和若干微控制器组成。

平台固件是一种特殊的软件，当硬件加电时，固件代码便被执行，该代码初始化硬件，之后固件的引导程序加载操作系统，或者有时是下一层级的执行载荷用于加载操作系统，大多数平台固件存储于 SPI Flash 中，在某些案例中，例如 UEFI，固件也可加载来自文件系统中的文件（hellov.efi），如同操作系统中的可执行文件（hellov.exe/a.out）那样。

自从 BIOS 年代的 IBM PC 开始平台固件经过了多年的演进后变得异常复杂，这极大地增加了攻击面，并且衍生版本造成的碎片化对于供应链也造成极大的风险，而大量的数据中心的管理中对于

固件作为软件资产的管理以及相关安全风险评估都是缺失的，更糟糕的是，有些固件实现是完全不可更新的，只有相对少数的平台固件实现拥有直接连接到互联网以获取更新的能力，即使某些实现（UEFI）拥有完整的网络栈以支持网络启动，或者带外管理功能以支持远程管理（BMC、IPMI 和 Redfish 等），但大多数实现仍然缺乏自动下载更新的能力。只有少数制造商和特定硬件型号支持通过操作系统的自动更新机制来自动更新固件，例如 Windows Update 和 Linux Vendor Firmware Service，需要注意的是有些固件供应商并不提供签名和加密。



BIOS/UEFI

BIOS 来自于 IBM PC 的年代，广泛用于 MS-DOS 的 16 位实模式，大多数其他操作系统仅将 BIOS 用于初始化其自身，并且依赖于它们自身的驱动程序来访问系统，BIOS 是基于中断（interrupt-based）的，并且为操作系统/应用程序提供服务：中断 13h 为磁盘 I/O、中断 10h 为视频 I/O 等，IBM PS/2 开发了 ABIOS（高级 BIOS）的变种，它是一种保护模式下可重入（reentrant）的解决方案，解决了 BIOS 的众多原始技术限制，其独占 PS/2 平台并且随着它一起消亡了。最初，BIOS 并没有包括任何安全考虑，然而随着安全性成为 PC 生态系统中的话题，BIOS 加入了口令，安全特性也开始被添加。

EFI 作为 Intel 为 Itanium 处理器架构开发的启动固件，后来 EFI 也用于 x86/x64 系统，由于苹果和微软的青睐，EFI 最终通过开源项目 Tianocore 而完成了标准化，即 UEFI（统一 EFI），EFI 几乎没有安全特性，其 UEFI 添加了安全启动（Secure Boot）（以及后来基于它的更多特性）和可执行文件的代码签名，UEFI 还具备完整的网络协议栈，这使得可以通过 PXE 方式进行远程操作。

ACPI 作为现代固件的一部分也被加入到 BIOS 中以启用电源管理，允许加电循环中的待机/唤醒状态，ACPI 同样也被 UEFI 使用，并且其 ACPI 规范由 UEFI 论坛控制，除了电源管理功能外，ACPI 是一种通用目的的扩展固件的方法，厂商可以在其中定义它们自己的代码模块（表），其中包含了代码和/或数据，ACPI 拥有一种架构独立的字节代码（AML，即 ACPI 机器语言），因此每种操作系统都需要拥有 ACPI VM 以执行 ACPI 代码，这在一定程度上增加了攻击平面。

基板管理控制器（BMC）/ 管理引擎（CSME）/PSP（平台安全处理器）

基板管理控制器通（BMC）是独立于 CPU、内存、外存和网络的系统，设计目的是为管理宿主系统，通常它们被设计为即使在宿主系统关机的情况下仍然保持运行，并且为系统管理员提供远程控制。BMC 是一个通用术语，有不同的实现包括 IPMI、DMTF、SMASH、DASH 和 Redfish。历史上，Intel CSME 和 AMD PSP 也可以作为类似 BMC 的带外管理设备，只是其为了其他特性考虑的缘故变得比独立 BMC 更加复杂，而这也成为了现代服务器/桌面系统的最大风险之一，本文档后续有章节详细介绍 CSME 风险评估。

网卡（NIC）

NIC 不论是独立的还是作为 SoC 的一部分而集成的，大多数现代客户端和服务器平台拥有至少一块 NIC（有线或者无线），并且可能拥有多块，包括多种类型（有线、Wi-Fi、蜂窝），受到攻击的 NIC 固件可以作为对系统中的其他漏洞的利用的跳板，被用于窃取数据、作为中间人等，除了由微控制器运行的固件以外，NIC 可能还包含在启动过程中加载并且由宿主处理器执行的扩展只读存储器（ROM）固件，NIC 的扩展 ROM 固件同样受到保护是至关重要的，扩展 ROM 固件可以同宿主处理器启动固件一同存储（对于集成 NIC 的情况），或者对于作为扩展卡的情况，可以独立存储于 NIC 本身。

图形处理器（GPU）

GPU 是作为客户端平台中的主要“输出型”人体学接口设备（HID）的设备。在某些情况下，GPU 也可以被用作“通用处理单元-GPGPU”以支持高性能计算。GPU 可以作为对系统中的其他漏洞的利用的跳板，除了由微控制器运行的固件部分以外，GPU 可能包含在启动过程中加载并且由宿主处理器执行的 Option ROM 及 EFI 固件。而此类固件可以同宿主处理器的启动固件一同存储（对于板载 GPU、CPU 集成 GPU 的情况），或者对于独立 PCI/PCIe 扩展卡的情况，固件通常独立存储于 GPU Flash 本身。

嵌入式控制器（EC）/ Super I/O（SIO）

EC 通常与移动平台（笔记本、变形本、平板）相关联，而 SIO 通常与基于桌面的平台（台式机、基于桌面的工作站）相关联。这并非普遍确切，但是通常足够确切以确定在某种类型的客户端系统中能够找到 EC 还是 SIO。EC 或者 SIO 通常控制平台中的诸如键盘、指示灯、风扇、电池监测/充电、散热监测等功能。此外，它通常是平台中的首个执行代码的系统板载设备，甚至会使得宿主处理器处于重置状态，直到 EC / SIO 准备就绪以允许宿主处理器获取它的首行宿主处理器固件代码。

可信平台模块（TPM/fTPM）

TPM 芯片被添加到某些 PC 上以提供可信根（root of trust），但是很多 BIOS 系统并未使用它。Trustworthy Computing Group （TCG）维护 TPM 标准以及相关协议，TPM 芯片被设计为存储难于访问的机密信息，除了基于硬件的 TPM 以外，还有固件 TPM（firmware TPM），这是一种软件

（固件）实现，运行于另一块处理器之上，例如在 Intel CSME 或者 AMD PSP 平台固件中的实现。

硬盘（HDD）/ 固态硬盘（SSD）

HDD 或者 SSD 代表了当前用于存储大量数据的传统平台中的最先进技术。这些设备与主机控制器（HC）相耦合。在 HDD 或者 SSD 内部，微控制器及其相关联的固件被用于执行将数据从平台的主内存发送至大容量存储设备的实际存储操作。对 HDD 或者 SSD 的固件的攻击同样可以用作对系统中的其他漏洞的利用的跳板，也可以被用于攻击用户和/或平台数据。

PCI(e) 和雷电

当 BIOS 最初被开发之时，主要的外设板卡总线接口是 ISA，而现在则是 PCI(e)。当硬件厂商向它们的板卡（网络/磁盘/视频适配器等）中添加特性时，它们需要扩展 BIOS 所提供的那些特性。板卡通过 OptionROM，即板卡所包含的用于扩展 BIOS 的固件来扩展 BIOS。BIOS 扩展其功能的方法是挂钩中断。雷电基本上是通过接口而非主板上的插槽的 PCI(e)，因而具有类似的 OptionROM 驱动程序的安全问题，并且对于攻击者而言，将设备连接到雷电接口上相对于打开系统机箱向主板上添加板卡更加容易，基于 USB3380 和 FPGA 实现的 PCILeech 以及 Thunderclap 是常用于 PCI(e)的攻击方法。

USB

基于 USB 的设备拥有它们自己的固件栈。攻击者有多种方式来利用 USB 固件以使得 USB 设备以有趣的方式运作。USB 设备上的固件驱动程序可以声明其为网络设备、本地存储设备或是众多其他选项，这些选项可以使其骗过过于轻信任何东西的操作系统或是盲目地插入他人为其提供的 USB 设备的最终用户。

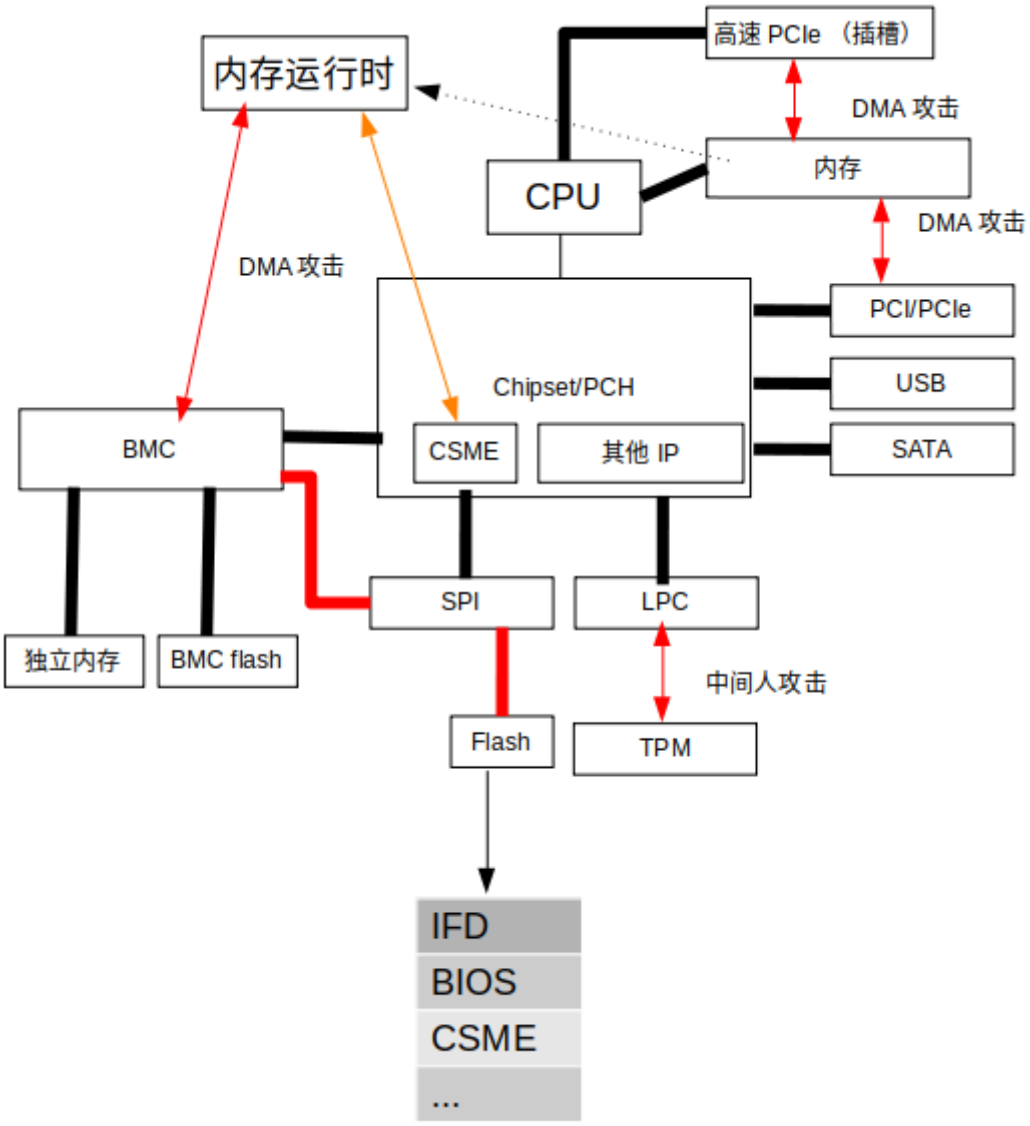
在通过接口连接、包含固件的设备之外，现代的线缆（USB、雷电等）内部通常也包含固件。攻击者能够以带有多种攻击向量的方式将一块恶意 PCI(e) 或者 USB 设备连接到系统上：使用诸如 Hak5 USB Rubber Ducky 等设备是常用于 USB 的攻击方法。最近的操作系统更新包括诸如在待机/唤醒状态下禁用接口的防护措施。

固件威胁模型

平台固件涉及到多个硬件部分，不同的硬件所运行的固件都有自身特定的攻击面，而一部现代计算机系统上运行了无数的独立硬件的组合，这些独立硬件都可以看作是独立的计算机（或者图灵机），这产生了大量的攻击面，攻击者会攻击不同的组件以获得更高的权限，一旦成功会带来诸多的好处：

* 固件持久化，可以固件启动的多个阶段进行植入，以 UEFI 固件为例，从早到晚的植入阶段有 PEI，DXE 和 ESP（EFI 系统分区），越早阶段的植入难度会更高，好处则是一旦成功则可以在满足特定条件下达到更好的隐蔽性，可以为攻击者后续的行动进行支持。

* 破坏信任链条，攻击者拥有更多的权限可以进一步的攻击可信计算体系，比如 UEFI secure boot 的大多实现中都会存在漏洞，某些阶段的漏洞可以直接绕过验签的过程。



常见漏洞以及攻击方法介绍

BIOS/UEFI

针对 BIOS/UEFI 的威胁模型主要涉及：

- SMM 漏洞利用（SMM 权限提升）
- UEFI 恶意固件植入及基于 SMM 的持久化攻击
- 供应链攻击
- 恶意外围设备
- Secure boot bypass
- 错误的 PCH/CPU 安全性配置

系统管理模式(System Management Mode，SMM)是 Intel CPU 的模式之一，通常也称为 Ring-2 ，因为它的特权级高于内核级以及虚拟化层。SMM 拥有自己的内存区段,叫作 SMRAM,它可以阻止其他执行模式的代码对这个区域的访问。SMM 早期引入时最初的目标不是提供安全特性而是用来处理计算机特定的需求，比如高级电源管理(APM，已被 ACPI 代替)以及调试作用。现在它也被用来保护对含有 UEFI 代码的 SPI 闪存的写入访问。它可以被看作是一个"安全区域"，与 ARM 上的 Trust Zone 类似。SMM 的代码、内存区域、权限等，由 UEFI 进行初始化和设置。

针对 SMM 的威胁主要集中在 SMM 漏洞利用和基于 SMM 的持久化攻击。利用 SMM 漏洞，可以实现提权、系统瘫痪甚至对 Flash 芯片的任意写入。而基于 SMM 的持久化攻击可以从深层破坏一个信息化系统，可以实现无磁盘文件、无内存代码的远控；从 IO 层面对键盘监听等功能，同时没有有效的安全防御软件。

SMM（System Management Mode）漏洞是指利用 SMM 相关逻辑缺陷、安全弱点、漏洞可以获得 SMM 执行权限，如缓冲区溢出、SMM 指针跨界、S3 脚本等漏洞。从而可以访问 SMRAM，执行只有 SMM 权限才能执行的代码、指令。SMM 包括 Ring 0 的所有特权，具有访问全部内存和外设硬件的最高特权级。由于 SMM 模式的特殊性，只能通过系统管理中断（System Management Interrupt，SMI）进入 SMM 模式，而 SMI 是一个特殊的中断，只能通过软件的同步或者硬件事件的异步机制引发。一旦 CPU 检测到 SMI 中断，将立即切换到 SMM 模式并跳转到预先定义的 SMM 入口处，同时保存现场。

常见的 SMM 漏洞主要包括：

- SMM-callout
- SWSMI 参数检查不完备
- S3 引导脚本表漏洞

  下面简要介绍 UEFI S3 引导脚本表漏洞。

UEFI S3 引导脚本表漏洞

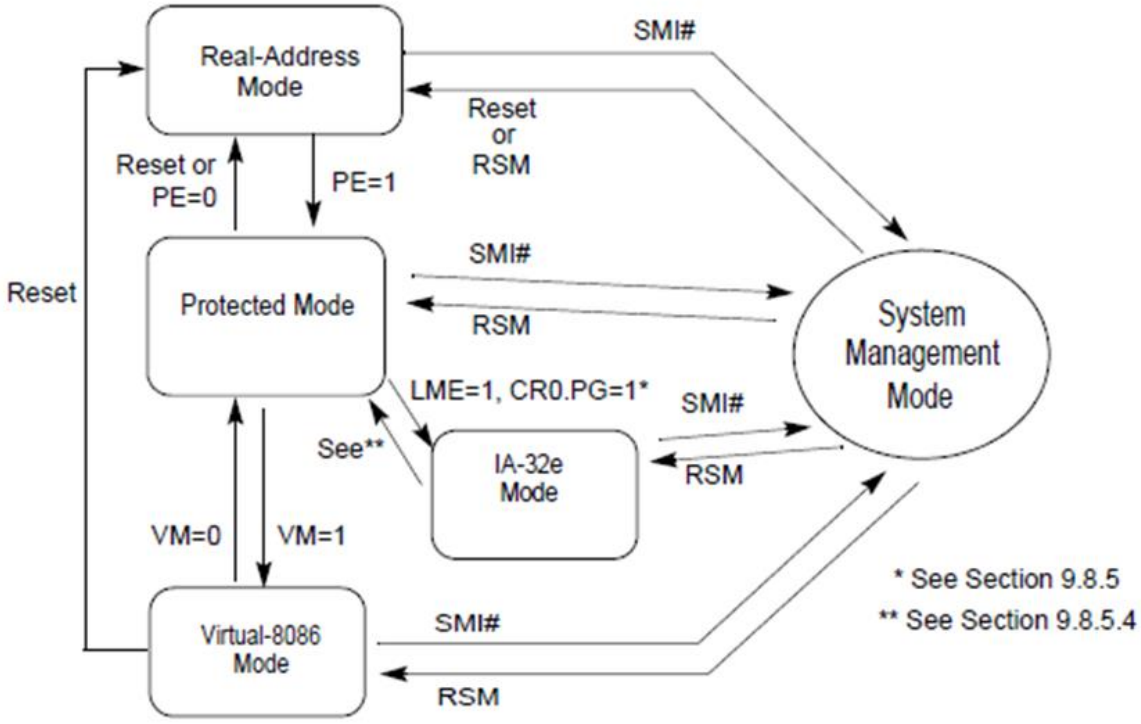


图 处理器工作模式之间的转换

如上图所示，SMM 可以从任一"普通"模式访问。SMM 支持 16 bit，32 bits 和 64 bit 模式，这使得它成为所有其他模式的备份。在普通模式和 SMM 之间的转换是由系统管理中断(SMI)触发时产生的。当该中断触发时,处理器切换到 SMM 模式：它首先将现在 CPU 的所有状态（含寄存器、标志位）保存到名为"保存状态(Saved State)"的内存区域(确保之后能够恢复)，然后切换包括指令指针在内的上下文，使其执行 SMRAM 中的代码。

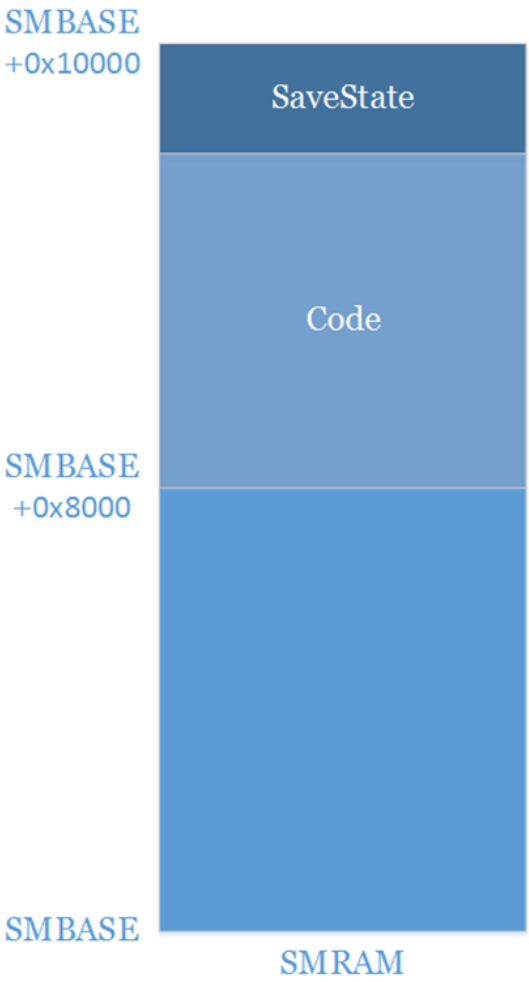


图 SMRAM 示意图

SMRAM 是一段 UEFI 固件保留给 SMM 模式使用的物理内存区域。SMRR 寄存器可以保护该区域，使得"普通"模式代码无法访问,也可以使得 DMA 方式无法访问它。SMBASE 是一个必须在该区域范围内的地址，它用来确定当切换到 SMM 时保存状态存储的位置，以及指令指针应该设置在哪个位置。每个核心都有一个 SMBASE(为了防止多个核心同时切换模式时会覆盖写入彼此的保存状态),并且在 SMRAM 内部没有对 SMBASE 地址位置的限制。

UEFI 引导计算机启动时除正常引导路径外，ACPI 兼容的现代计算机固件还为“S3 级别睡眠恢复模式”执行单独的引导路径。UEFI 固件中有一个特殊的数据结构，叫做 UEFI S3 boot script table（S3 引导脚本表），它用于在正常引导路径期间保存系统寄存器值，并在 S3 恢复期间快速恢复这些寄存器值。

按照安全性设计，S3 引导脚本表理应保存在 SMRAM 中，防止普通用户和进程访问。但在某些计算机 BIOS 实现中，UEFI 引导脚本表存储在普通物理内存中，或者由于配置错误，导致 Ring 0

用户进程可以访问 SMRAM 区域的 S3 脚本表，这些为 S3 漏洞创造了触发条件。因此，如果攻击者能够从正在运行的操作系统中修改 S3 脚本表，并触发 S3 睡眠，将有可能覆盖**负责平台安全性的某些系统寄存器的值**。如前述章节所介绍的 SMM_BWP、BLE、FlockDN 等寄存器值。

因此，一些厂商的固件其 S3 脚本表可被某种方式修改，同时在执行引导脚本表之前没有锁定其中的某些寄存器，这种类型的缺陷被称为“UEFI 引导脚本表漏洞”。成功利用 UEFI 引导脚本表漏洞可以突破计算机的 Flash 安全机制。

DMA 物理威胁介绍

DMA（直接内存访问）是 PCI 总线（包括 PCIe）最强大的功能之一，直接内存访问技术允许外设在不依赖 CPU 的情况下访问 RAM。DMA 大幅提高了数据传输性能，但也带来了安全问题。但这是预期之内的。因为它允许用户、外设在操作系统甚至 CPU 没有察觉的情况下直接访问内存。随着多年来 DMA 功能不断的普及，攻击面也随之增加。利用 DMA 功能可以实现对系统任意物理内存的读写访问，导致系统解锁、实时植入恶意代码等攻击行为。这些攻击已成为不容忽视的攻击向量。为了解决这些问题，Intel/AMD 等公司为 CPU 实现了一个 IOMMU 机制。IOMMU，即 IO 内存管理单元。它可以决定什么设备可以访问物理内存的哪一部分，从而避免利用 DMA 机制对系统造成的攻击。

Intel VT-d 、AMD-Vi 以及 ARM 的 SMMU 都是 IOMMU 的一种实现技术。VT-d（VT for Directed I/O）通过更新设计的 IOMMU 架构，实现了多个 DMA 保护区域，最终实现了 DMA 虚拟化。这个技术也叫做 DMA Remapping，VT-d 已经成为 intel 的 IOMMU 主要实现，本指南建议用户打开 BIOS Setup 中的 VT-d（或其他等同技术），防止 DMA 攻击。

网卡（NIC）

涉及网卡的弱点、漏洞大多数是关于驱动程序的，利用这类漏洞，可以实现提权。也有一部分是涉及 NVM 操作，可导致网卡资源不可用（拒绝服务）。如：CVE-2020-8690、CVE-2020-8691、CVE-2020-8692、CVE-2020-8693、CVE-2020-24492/93/94/95/96/97/98

恶意固件案例

恶意固件名称	公开披露年份	预计投放使用年份	感染类型	操作系统	开发商
Vector-EDK	2015	2014	DXE	?	HackingTeam
DerStarke	2016	早于 2014	DXE	Windows/ Linux/macOS	CIA/Vault7
QuarkMatter	2016	早于 2014	ESP	Windows/ Linux	CIA/Vault7
LoJaX	2018	2017 或更早	DXE	Windows	APT28
TrickBot	2020	2017	DXE	Windows	N/A
FinSpy	2021	2011	MBR/ESP	Windows/ Linux/macOS	N/A
ESPecter	2021	2012/2020	MBR/ESP	Windows	N/A

从目前公开的恶意固件样本中可以清晰的看到大致的持久化流程：检查固件安全设置是否正确，如果未正确设置（比如没有写保护）就直接写入恶意模块到 SPI flash 中，如果正确设置的情况下则使用漏洞利用（例如 CVE-2014-8273）后进而完成恶意模块植入，早期固件恶意植入目标主要是 MBR 和 ESP，但随着攻击门槛的不断降低也大量出现了 DXE 甚至 PEI 阶段的植入。

堡垒悖论

根据上述公开的信息，企业或者机构依然难以得出固件的实际风险仅限于此的结论，因为无法得出确切结论的原因主要是受限于高级防护领域的堡垒悖论魔咒，即在构建系统安全防护体系过程中的复杂性导致威胁建模的不准确。假设一台高级防护节点经过了从应用层，内核层，固件层，芯片层以及密码工程的纵深加固后上线：

1）假设攻击者的目标是拿到内核权限并且植入内核 rootkit，那攻击者攻破了应用层和内核层防护后已经达到目的，自然不会去攻击下一层，即使之后通过取证和应急响应拿到了相关攻击的证据，也只能说明攻击者具备攻击内核层的能力，并不能证明攻击者没有攻陷虚拟化，SMM 以及 CSME 的能力，这种情况会让企业或者机构的决策者出现盲区。

2） 假设攻击者已经攻陷了 UEFI/SMM 固件层并且成功持久化，如果攻击者没有进行大规模行动而只是定向打击高价值目标，那此类攻击由于设计上的“不可访问”等逻辑，也难以发现，而在这个层面取证的难度和成本大幅增加，并且缺少有效方法。

3）假设攻击者已经攻陷了包括 CSME 在内的 PCH 上大部分 IP 并成功持久化，到这个阶段大部分检测和取证手段都失效，这个阶段通常企业和机构决策者会认为系统是安全的。

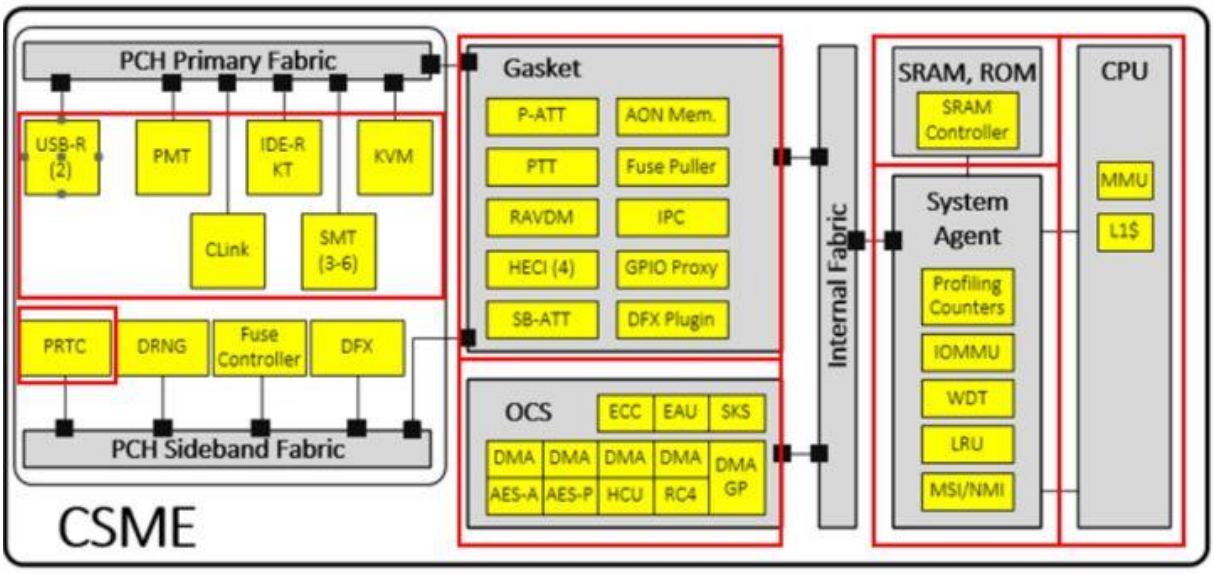
Intel CSME 风险评估

Intel CSME（早期名为 ME（Management Engine），管理引擎）自从发布起长期存在争议，争议内容主要涉及后门和漏洞两个方面，近年来这些争议也让开始引起数据中心运营方的注意，本章节从技术的角度探讨企业或者机构针对 CSME 的风险评估所涉及的方面。

ME 介绍和版本演化

早在 2008 年，Intel 就将 ME（管理引擎）引入芯片组，它比操作系统具有更高的特权，这意味着操作系统级别的安全检测方案对 ME 几乎是无效的，更重要的是，绝大部分用户都不知道他们的计算机中有一个独立的“小电脑”存在，在安全研究方面，早在 2009 年 Alexander Tereshkin 和 Rafal Wojtczuk 证明了植入 Ring -3 rootkit 的可能性（注：在此上下文中，“Ring”的概念为，应用：Ring 3，Linux 内核：Ring 0，虚拟化：Ring -1，BIOS/UEFI/SMM：Ring -2，Intel ME： Ring -3）。

另外一项有趣的研究 Patrick Stewin 于 2014 年发布的 DAGGER，遗憾的是，大多数人在过去 10 多年中都不知道 ME 的存在之广泛，直到 2018 年 Google 决定在部分机器上干掉 ME 才引起了业界的关注。Intel 在 KabyLake 这一代处理和芯片组发布时给 ME 有了新的命名：CSME（Converged Security and Management Engine），这或许是因为同代芯片组把 TXT 和 BootGuard 这两个安全特性绑定在一起的原因。ME/CSME 通常是指 x86 桌面平台的 PCH 中的一个独立 IP，运行过程中也会使用到其他 PCH 中的 IP，值得注意的是任何 PCH 中的 IP 都非常重要，一旦被攻陷则会陷入堡垒悖论：

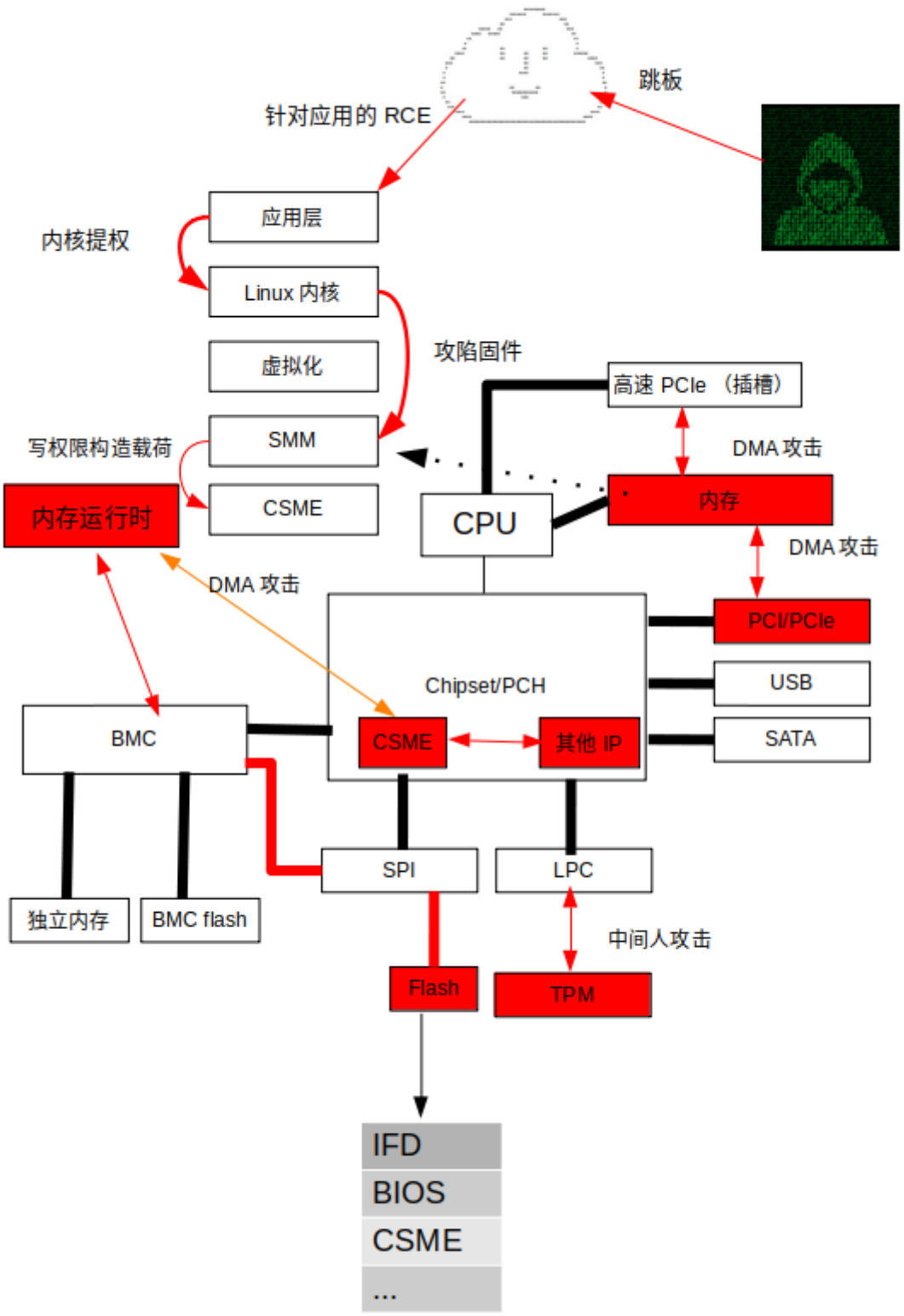


ME 主要应用于 x86 桌面系统，TXE 和 SPS（Server Platform Service）分别是嵌入式和服务器平台的实现。ME/CSME 的发展中其硬件和软件都经过了不断的变化和更新：

	ME	ME	ME	ME	ME
版本	1.x – 5.x	6.x – 10.x	11.x	12.x	15.x
硬件核心	ARCTangent-A4	ARCompact	Quark	Quark	Quark(?)
指令集	ARC (32-bit)	ARCompact(32/16)	x86(32-bit)	(32-bit) x86	x86 (32-bit)
防御特性	N/A	N/A	NX	SMEP	CET/CFI
操作系统	??	ThreadX	MINIX	MINIX	MINIX
对应SPS版本	SPSv1.x	SPSv2.x – v3.x	SPSv4.x	SPSv5.x	SPSv6.x

CSME 威胁模型和攻击面

从攻击面的角度，本指南提供两种极端的攻击路径，第一种是攻击者成功构建多层级打击链最终攻陷 CSME，第二种攻击者已经掌控 CSME 的前提下针对宿主机发起攻击：



1）从主机（Host）CPU 运行的系统上构建打击链深入到 Intel CSME 并且植入持久化，这是一个很长的攻击链条：

* Ring 3 的漏洞利用（webshell 或者应用程序的远程利用）获得普通执行权限
* 利用 Ring 0 内核漏洞进行提权，这里值得注意的是虽然操作系统内核早已经不是 2007 年“Attacking the CORE”上下文的那个”CORE“但依旧非常重要，因为内核是通向更底层的入口

* 从目前攻击样本来看不需要过多关注 Ring -1 虚拟化层。
* 通过绕过芯片组防护机制或者物理攻击攻陷 Ring -2 的固件，比如 coreboot/UEFI/SMM，以达到获得写 SPI flash 的能力。
* 触发 Ring -3（CSME）早期启动阶段（RBE，kernel 等在>=CSMEv11 版本中无法关闭的模块）或者 CSME 代码模块的 0day 或者已知漏洞（比如 SA-00086）以获得 CSME 完整控制权，攻击者可以使用 CSME 作为跳板开启 VISA 访问 PCH 的内部接口。

2）从 CSME 攻击主机（Host）系统，攻击场景如下：
* 攻击者具备 1)的能力前提下，可以直接替换厂商固件并在 IOMMU 开启前具备执行 RS1 DMA 操作的能力，这等同于给了 CSME 上帝模式可以任意读写主机侧操作系统（Linux，Windows，OSX 等）内存。
* OEM 厂商在固件中有意或者无意的错误配置 PSF（Primary Scalable Fabric）以辅助 CSME 拥有上帝模式。
* CSME 使用 IMRs 直接绕过 IOMMU/VT-d 进入上帝模式。

值得企业用户注意的是，以上不论哪种攻击场景发生后会给取证工作带来巨大的麻烦，这也是从一开始就需要关注固件安全的原因。另外，针对上帝模式的机型覆盖率作为风险评估的因素之一涉及多个维度（公开材料，NDA 授权以及逆向工程），本指南建议企业或者机构可以咨询专业机构。

警告：小心调试和测试你的 PoC，任何 CSME 相关的测试和微调都可能导致 PCH 烧毁。

禁用 CSME

自由软件/固件社区和安全研究社区一直在寻找各种关闭 CSME 的方法，针对普遍机型的通用方法大致如下：

类型	结果	时间
代码删除	Core 2之后x86平台移除所有模块会导致每30分钟重启一次	2010
中和	最小化删除代码模块	2016
关闭	设置altmedisable/HAP位，CSME硬件初始化完成后自动关闭	2017
关闭	通过HECI/HMRFPO指令运行时关闭	2018

应该关掉 CSME 吗?

这个问题本身涉及到多个因素，你是否信任 Intel？你认为 Intel 和 OEM 厂商谁植入后门的概率更高？有一些用户非常重视隐私，数字自由的权利或者选择不信任 Intel 甚至是 OEM 厂商，他们

愿意承受比普通用户更高的成本去干掉 CSME，而大多数用户对 Intel 依然持有”信仰的飞跃“。从 CSO 和安全工程师的角度，漏洞是风险的主要因素，那有两个方面必须考虑：

* CSME 的自身防护能力
* 有多少 CSME 提供的安全特性是企业整体方案必须的

第一个方面相对容易量化：

版本	防御机制	风险
<=CSMEv11/SPSv3	NX	高
<=CSMEv14/SPSv5	NX/SMEP	中
>=CSMEv15/SPS6	NX/SMEP/CET	低

NX（不可执行的栈）最早出现于 2000 年的 PaX 中，SMEP 是让攻击者在漏洞利用过程中没办法直接在内核中执行用户空间的代码，CET 是一种 TigerLake 开始引入的硬件 CFI（代码流完整性），Intel 在 v15/SPSv6 的硬件平台中也加入了支持。虽然 Intel CSME 是一个无法被全面审计的黑盒，黑客和安全研究者通过对部分代码模块的逆向找到一些有趣的发现，比如架构综述，NSA 的隐藏开关，IDLM 模块等。开源带来的透明度并不等同于安全，没有源代码的情况下依然可以进行更为耗时的二进制审计。另一方面，闭源也并不代表安全，通常闭源软件的质量差于开源软件，但 CSME 则是一个例外，CSMEv15 引入了更多防护机制的同时也把模糊测试加入了开发流程，不论开源还是闭源软件都会受益于这些技术。

第二个方面，CBnT 可以用于构建信任根，SGX 可以提供飞地计算方案，如果关掉了 CSME 那你会失去这些安全特性。当然，可以通过其他信任基和加强数据中心重要机型的物理防护管理来进行弥补，而这也是一个不断选择的过程。

使用 AMD 平台是否更安全？

AMD 平台上有一个类似 CSME 的实现叫 PSP（Platform Security Processor），PSP 也会辅助 AMD 平台完成一些安全特性，比如机密计算方案 SEV（Secure Encrypted Virtualization-Encrypted ），PSP 相对 CSME 的相对优势在于其安全特性的设计和实现的复杂度低于 Intel，在企业整体安全防护方面上并无优势。

CSME 工具集

企业或者机构在评估 CSME 过程中有一些公开的辅助工具可以使用：

工具	ME类型	用途
me_cleaner	ME/TXE/SPS	删除代码模块以及开启altmedisable/HAP位
intelmetool	ME	获取ME以及BootGuard信息
me-disable	ME	HECI指令运行时关闭ME
spsInfo	SPS	获取SPS信息
mei-amt-check	ME	
mmdetect	ME/SPS	检测ME制造模式状态
ifdtool	ME	综合型工具

固件安全防护体系

UEFI 类别的安全机制介绍

BIOS/UEFI 攻击的主要焦点之一在于针对主板 Flash 芯片的任意读写。如果攻击者通过某种方法获得了针对主板 Flash 芯片的任意读写权限，则很容易实现针对系统的持久化攻击。

现代计算机系统从 CPU、PCH 设计/IBV/ODM/OEM 等层面对主板 Flash 芯片采取了一些保护措施。主要包括：

- SMM_BWP/BLE/BIOSWE -- Global Flash Write Protection
- BIOS Range Write (PRx) Protection
- SMRAMC Protection
- BIOS Flash Descriptor Read/Write Access Protection
- BootGuard 完整性保护
- BIOSGuard 完整性保护
- 及 OEM 厂商的保护措施（主要是完整性保护措施及身份验证方式）

SMM_BWP/BLE/BIOSWE -- Global Flash Write Protection

BIOS_CNTL 是 PCH 中的一个寄存器，包含 BIOSWE/BLE/SMM_BWP 等标志位，对应多种保护机制。他们之间又有一些逻辑关联性。

BIOSWE/BLE 是早期的 Flash 保护机制，其中：

* BIOSWE 定义：当 BIOSWE 置位时，系统允许对 SPI flash 进行写入访问。而如果 BLE bit 代表对 BIOSWE bit 的锁定，当 BLE=0 时可任意设置 BIOSWE 数值。

* BLE 定义：

当 BLE=0：可任意修改 BIOSWE 数值。

当 BLE=1：应用程序（非 SMM 代码）尝试置位 BIOSWE 时（通常是驱动程序），这个动作将触发一个 SMI#。SMI 关联的 SMM handler 将决定这个置位动作是否合法。如果非法（发现来自应用程序）则 SMM handler 立即取消置位，使得 BIOSWE=0。

BIOS_CNTL—BIOS Control Register (LPC I/F—D31:F0)

Offset Address:	DCh	Attribute:	R/WLO, R/W, RO
Default Value:	00h	Size:	8 bit
Lockable:	No	Power Well:	Core

Bit	Description
7:5	Reserved
4	**Top Swap Status (TSS)** — RO. This bit provides a read-only path to view the state of the Top Swap bit that is at offset 3414h, bit 0.
3:2	**SPI Read Configuration (SRC)** — R/W. This 2-bit field controls two policies related to BIOS reads on the SPI interface: Bit 3- Prefetch Enable Bit 2- Cache Disable Settings are summarized below: **Bits 3:2** — **Description** 00b — **No prefetching, but caching enabled.** 64B demand reads load the read buffer cache with "valid" data, allowing repeated code fetches to the same line to complete quickly 01b — **No prefetching and no caching.** One-to-one correspondence of host BIOS reads to SPI cycles. This value can be used to invalidate the cache. 10b — **Prefetching and Caching enabled.** This mode is used for long sequences of short reads to consecutive addresses (i.e., shadowing). 11b — **Reserved. This is an invalid configuration**, caching must be enabled when prefetching is enabled.
1	**BIOS Lock Enable (BLE)** — R/WLO. 0 = Setting the BIOSWE will not cause SMIs. 1 = Enables setting the BIOSWE bit to cause SMIs. Once set, this bit can only be cleared by a PLTRST#
0	**BIOS Write Enable (BIOSWE)** — R/W. 0 = Only read cycles result in Firmware Hub I/F cycles. 1 = Access to the BIOS space is enabled for both read and write cycles. When this bit is written from a 0 to a 1 and BIOS Lock Enable (BLE) is also set, an SMI# is generated. This ensures that only SMI code can update BIOS.

图 BIOS_CNTL(BLE)写保护机制

然而 BIOSWE/BLE 存在设计或实现缺陷，可通过条件竞争的方式攻击 BIOSWE/BLE Flash 保护机制。因此，Intel 等 CPU 厂商又实现了一种保护机制：SMM_BWP，是对 Flash 保护的升级。SMM_BWP 同样位于 PCH 芯片组的 BIOS_CNTL 寄存器中：bit 5。

SMM_BWP 具体定义：按照 Intel 规范的说明，它提供一项写保护能力，确保 BIOS 区域仅在所有处理器核心都处在 SMM 模式中，且 BIOSWE=1 时才可以被写入。

BIOS_CNTL—BIOS Control Register (LPC I/F—D31:F0)

Offset Address:	DCh	Attribute:	R/WLO, R/W, RO
Default Value:	20h	Size:	8 bits
Lockable:	No	Power Well:	Core

Bit	Description
7:6	Reserved
5	**SMM BIOS Write Protect Disable (SMM_BWP)**—R/WL. This bit set defines when the BIOS region can be written by the host. 0 = BIOS region SMM protection is disabled. The BIOS Region is writable regardless if processors are in SMM or not. (Set this field to 0 for legacy behavior). 1 = BIOS region SMM protection is enabled. The BIOS Region is not writable unless all processors are in SMM and BIOS Write Enable (BIOSWE) is set to '1'.
4	**Top Swap Status (TSS)**—RO. This bit provides a read-only path to view the state of the Top Swap bit that is at offset 3414h, bit 0.
3:2	**SPI Read Configuration (SRC)**—R/W. This 2-bit field controls two policies related to BIOS reads on the SPI interface: Bit 3 – Prefetch Enable Bit 2 – Cache Disable Settings are summarized below: **Bits 3:2** — **Description** 00b — **No prefetching, but caching enabled.** 64B demand reads load the read buffer cache with "valid" data, allowing repeated code fetches to the same line to complete quickly. 01b — **No prefetching and no caching.** One-to-one correspondence of host BIOS reads to SPI cycles. This value can be used to invalidate the cache. 10b — **Prefetching and Caching enabled.** This mode is used for long sequences of short reads to consecutive addresses (that is, shadowing).
1	**BIOS Lock Enable (BLE)**—R/WLO. 0 = Transition of BIOSWE from '0' to '1' will not cause an SMI to be asserted. 1 = Enables setting the BIOSWE bit to cause SMIs and locks SMM_BWP. Once set, this bit can only be cleared by a PLTRST#.
0	**BIOS Write Enable (BIOSWE)**—R/W. 0 = Only read cycles result in Firmware Hub or SPI I/F cycles. 1 = Access to the BIOS space is enabled for both read and write cycles. When this bit is written from a 0 to a 1 and BIOS Lock Enable (BLE) is also set, an SMI# is generated. This ensures that only SMI code can update BIOS.

图 BIOS_CNTL(SMM_BWP)写保护机制

BIOS Range Write(PRx) Protection

在 UEFI 规范和 PCH 实现中，还将 BIOS/UEFI 数据区域分为多个区段（Range），并提供了 PRx

寄存器及相应的权限设置，用来保护这些区段的读写权限。这样设计的目的是：更加灵活的对 UEFI 数据区域进行权限管理，以免粗糙的权限粒度造成易用性和安全性的失衡。下图为 Intel PCH 手册中的简要描述：

Table 5-60. Flash Protection Mechanism Summary

Mechanism	Accesses Blocked	Range Specific?	Reset-Override or SMI#-Override?	Equivalent Function on FWH
BIOS Range Write Protection	Writes	Yes	Reset Override	FWH Sector Protection
Write Protect	Writes	No	SMI# Override	Same as Write Protect in previous ICHs for FWH

图 PRx 区段保护寄存器

PR0—Protected Range 0 Register (SPI Memory Mapped Configuration Registers)

Memory Address: SPIBAR + 74h　　Attribute: R/W
Default Value: 00000000h　　Size: 32 bits

This register can not be written when the FLOCKDN bit is set to 1.

Bit	Description
31	**Write Protection Enable** — R/W. When set, this bit indicates that the Base and Limit fields in this register are valid and that writes and erases directed to addresses between them (inclusive) must be blocked by hardware. The base and limit fields are ignored when this bit is cleared.
30:29	Reserved
28:16	**Protected Range Limit** — R/W. This field corresponds to FLA address bits 24:12 and specifies the upper limit of the protected range. Address bits 11:0 are assumed to be FFFh for the limit comparison. Any address greater than the value programmed in this field is unaffected by this protected range.
15	**Read Protection Enable** — R/W. When set, this bit indicates that the Base and Limit fields in this register are valid and that read directed to addresses between them (inclusive) must be blocked by hardware. The base and limit fields are ignored when this bit is cleared.
14:13	Reserved
12:0	**Protected Range Base** — R/W. This field corresponds to FLA address bits 24:12 and specifies the lower base of the protected range. Address bits 11:0 are assumed to be 000h for the base comparison. Any address less than the value programmed in this field is unaffected by this protected range.

图 PRx 区段保护寄存器

此外，为了保护 PRx 区段寄存器在设定后不被随意修改，还设定了针对 PRx 寄存器的保护机制：在 PCH 的 HSFS 寄存器(Hardware Sequencing Flash Status Register)中，存在的 FLOCKDN bit。

FLOCKDN bit 定义：

当 FLOCKDN=0 时，PRx 区段寄存器可任意修改；

当 FLOCKDN=1 时，PRx 区段寄存器被锁定，读写权限无法修改。直到下一次 Reset#。

HSFS—Hardware Sequencing Flash Status Register (SPI Memory Mapped Configuration Registers)

Memory Address: SPIBAR + 04h　　Attribute: RO, R/WC, R/W
Default Value: 0000h　　Size: 16 bits

Bit	Description
15	**Flash Configuration Lock-Down (FLOCKDN)** — R/W/L. When set to 1, those Flash Program Registers that are locked down by this FLOCKDN bit cannot be written. Once set to 1, this bit can only be cleared by a hardware reset due to a global reset or host partition reset in an Intel ME enabled system.

图 FLOCKDN 保护寄存器定义

正确的 SMM_BWP/BLE/PRx 保护状态设定如下图所示（红框中，需置位为 1）：

```
[*] running module: chipsec.modules.common.bios_wp
[x][ ================================================================================
[x][ Module: BIOS Region Write Protection
[x][ ================================================================================
[*] BC = 0x00000AAA << BIOS Control (b:d.f 00:31.5 + 0xDC)
    [00] BIOSWE            = 0 << BIOS Write Enable
    [01] BLE               = 1 << BIOS Lock Enable
    [02] SRC               = 2 << SPI Read Configuration
    [04] TSS               = 0 << Top Swap Status
    [05] SMM_BWP           = 1 << SMM BIOS Write Protection
    [06] BBS               = 0 << Boot BIOS Strap
    [07] BILD              = 1 << BIOS Interface Lock Down
[+] BIOS region write protection is enabled (writes restricted to SMM)

[*] BIOS Region: Base = 0x00400000, Limit = 0x00FFFFFF
SPI Protected Ranges
------------------------------------------------------------
PRx (offset) | Value    | Base     | Limit    | WP? | RP?
------------------------------------------------------------
PR0 (84)     | 00000000 | 00000000 | 00000000 | 1   | 0
PR1 (88)     | 00000000 | 00000000 | 00000000 | 1   | 0
PR2 (8C)     | 00000000 | 00000000 | 00000000 | 0   | 0
PR3 (90)     | 00000000 | 00000000 | 00000000 | 0   | 0
PR4 (94)     | 00000000 | 00000000 | 00000000 | 0   | 0

[!] None of the SPI protected ranges write-protect BIOS region
```

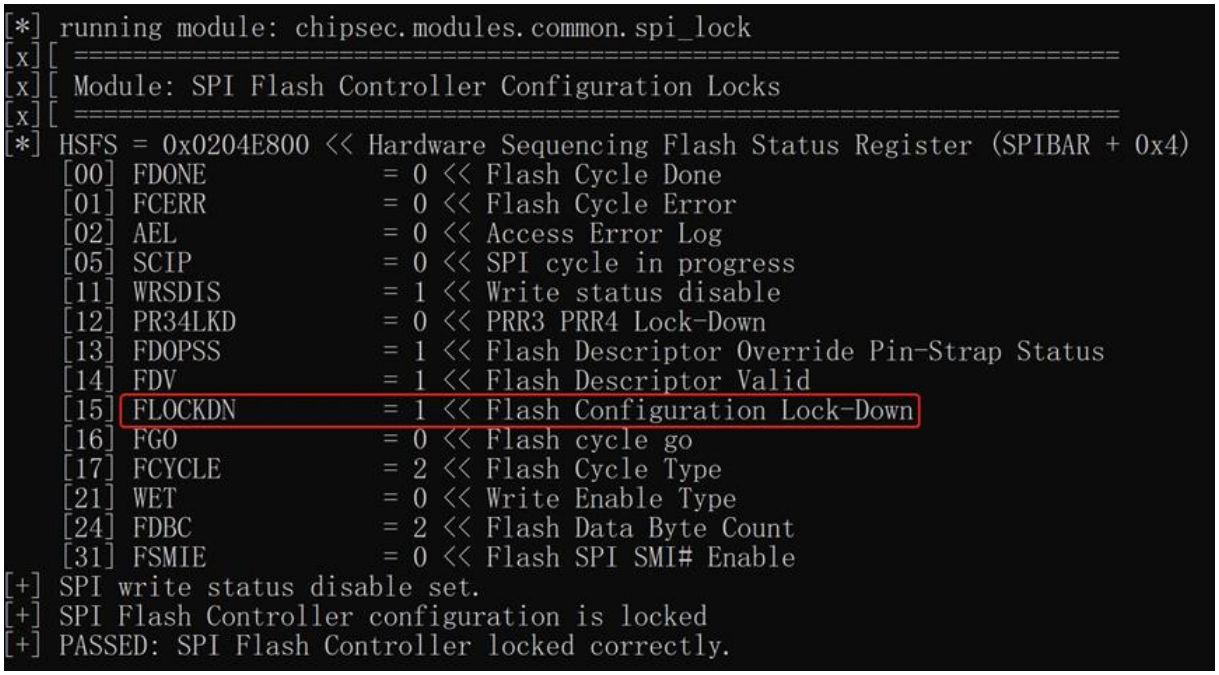


图 FLOCKDN 正确设定

BIOS Flash Descriptor Read/Write Access Protection

早期的 BIOS 镜像是 legacy 模式，在 legacy BIOS 镜像文件里面主要就是 BIOS 二进制代码和 logo 等数据。在 UEFI 普及后，且在 Intel ICH8 PCH 后，Intel 加入了针对 ME 的支持，引入 Descriptor mode。从此 SPI Flash 里面不再只有 BIOS 代码，而简单来说变成了如下结构：

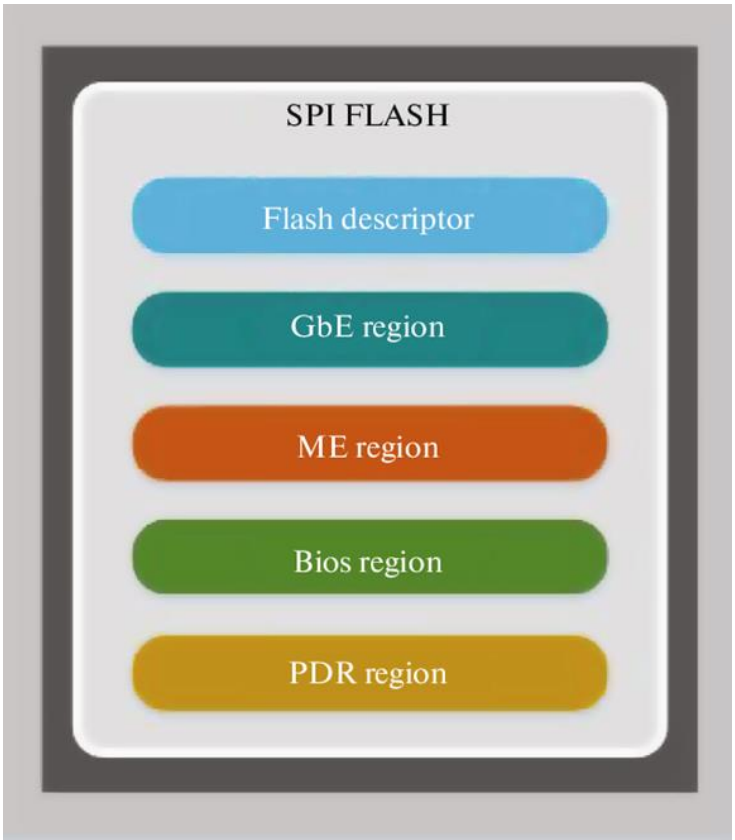


图 BIOS Flash 模块组成

如果是笔记本或其他手持设备，则通常还会有 EC 模块，即：Flash description、GbE、 ME、BIOS、PDR、EC 6 个分区。其中，BIOS 分区就是 UEFI 代码和数据所在的区域，它只是整个 Flash 或 Flash 镜像的一部分，而其中 Flash Descriptor（闪存描述符，简称 FD）定义了各个分区的读写权限。Flash Descriptor 是在所有基于 Intel 平台的 SPI 闪存芯片上的数据结构。它包含诸如为闪存镜像的每个区域分配的空间、每个区域的读写权限、供应商特定数据的保留空间、芯片组配置参数等信息。Flash Descriptor 的固定大小为 4KB (0x1000h)，根据平台生成，大致由以下部分组成：

图 Flash Descriptor 结构定义

较旧的平台使用 Flash Descriptor v1（非 Intel 命名），它最多可支持 8 个 SPI 区域。更新的平台（>= 100 系列或 APL 架构）使用 Flash Descriptor v2-3，它可以支持多达 16 个 SPI 区域。

其中，我们在本指南中关注的是 Master 部分。它包含 flash 的硬件安全设置，授予每个区域的读/写权限。每个 FD 权限访问条目的大小为 0x4 字节，并采用 Little Endian 顺序（每个字节从右到左读取）。在 FD v1/v2 中，仅分别使用前两个或三个字节，其余未使用。第一个值是写访问，第二个是读访问。每个读取和/或写入位**表示特定分区对其他分区的访问**。如图所示：

Bits	Description
31:29	Reserved, must be zero
28	**Platform Data Region Write Access.** If the bit is set, this master can erase and write that particular region through register accesses.
27	**GbE Region Write Access.** If the bit is set, this master can erase and write that particular region through register accesses.
26	**ME Region Write Access.** If the bit is set, this master can erase and write that particular region through register accesses.
25	**Host CPU/BIOS Master Region Write Access.** If the bit is set, this master can erase and write that particular region through register accesses. Bit 25 is a don't care as the primary master always has read/write permissions to it's primary region
24	**Flash Descriptor Region Write Access.** If the bit is set, this master can erase and write that particular region through register accesses.
23:21	Reserved, must be zero
20	**Platform Data Region Read Access.** If the bit is set, this master can read that particular region through register accesses.
19	**GbE Region Read Access.** If the bit is set, this master can read that particular region through register accesses.
18	**ME Region Read Access.** If the bit is set, this master can read that particular region through register accesses.
17	**Host CPU/BIOS Master Region Read Access.** If the bit is set, this master can read that particular region through register accesses. Bit 17 is a don't care as the primary master always has read/write permissions to it's primary region
16	**Flash Descriptor Region Read Access.** If the bit is set, this master can read that particular region through register accesses.
15:0	**Requester ID.** This is the Requester ID of the Host processor. This must be set to 0000h.

图 Flash Descriptor 中 Master 字段的官方定义

以假设的 FD v1 某区域为例，它对其他 7 个区域具有这样的读/写权限：

Access	A (Bit 0)	B (Bit 1)	C (Bit 2)	D (Bit 3)	E (Bit 4)	F (Bit 5)	G (Bit 6)	H (Bit 7)
E Write	Yes	No	Yes	No	Yes	Yes	No	No
E Read	Yes	No	Yes	Yes	Yes	Yes	Yes	Yes

其中，A-H 代表 8 个区域，假设当前区域为 E，那么上图就标识了 E 区对其他区域的读写访问权限。“可访问”由 1 表示，“不可访问”为 0。因此，假定区域 E 的写入权限等于 10101100b = 0xAC，而假定区域 E 区的读取权限等于 10111111b= 0xBF。由于在 FD v1 中仅使用前两个字节，因此假定区域 E 的最终访问权限条目为：0xACBF0000。以此类推。

如果想在所有重要区域（BIOS、ME、GbE、EC）的 Flash Descriptor 上启用完全的读/写访问

权限，则须将它们设置为 FD v1 的 0xFF 或 FD v2-3 的 0xFFF。如下所示：

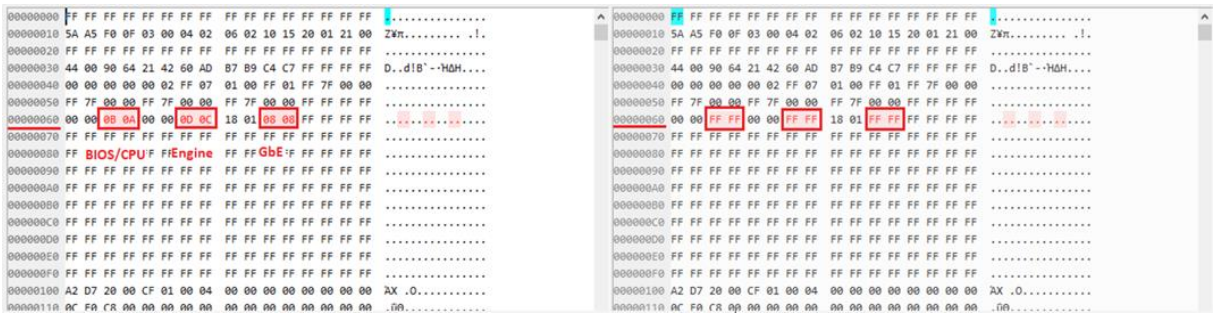


图 Flash Descriptor 读写访问权限

然而，这种设定是不安全的。可以使得某区域对其他重要区域进行读写，会造成系统固件级瘫痪甚至持久化控制的风险。

正确的设定应该是将其设定为适当的访问权限。如下图所示：

```
Reading HSFSTS register... Flash Descriptor: Valid

    --- Flash Devices Found ---
    GD25B128C     ID:0xC84018     Size: 16384KB (131072Kb)

    --- Flash Image Information --
    Signature: VALID
    Number of Flash Components: 1
        Component 1 - 16384KB (131072Kb)
    Regions:
        DESC       - Base: 0x00000000, Limit: 0x00000FFF
        BIOS       - Base: 0x00400000, Limit: 0x00FFFFFF
        CSME       - Base: 0x00001000, Limit: 0x003AFFFF
        GbE        - NOT PRESENT
        PDR        - NOT PRESENT
        EC         - Base: 0x003B0000, Limit: 0x003FFFFF
    Master Region Access:
        BIOS       - ID: Read: 0x000F, Write: 0x000A
        CSME       - ID: Read: 0x000D, Write: 0x0004
        GbE        - ID: Read: 0x0009, Write: 0x0008
        EC         - ID: Read: 0x0101, Write: 0x0100
```

图 配置正确的 Flash Descriptor

由此可见，当前计算机系统中，对 Flash 的保护是全方位的，多角度的。攻击者若想实现主板 Flash 芯片的任意读写操作，需要绕过多重写保护机制。因此，需要设定好已有的保护机制。

SMRAM Protection

SMM[见 SMM 章节]的可执行代码和数据存在于 SMRAM 中，当 SMRAM 被锁定时，操作系统或用户模式软件的任何代码无法访问它。通常，系统固件（传统 BIOS 或 UEFI）将 SMM 代码复制到

SMRAM 并在平台初始化期间将其锁定。

SMRAM 可以位于兼容内存段 （CSEG）、高内存段 （HSEG） 或内存顶部段 （TSEG）系统(物理)内存区域中。CSEG 是 SMRAM 的默认区域，位于非缓存物理内存的固定地址范围 A0000h:BFFFFh（与 VGA 内存重叠）。 CSEG 主要由传统 BIOS 开发人员使用，现代系统可以使用（并实际使用）SMRAM 的其他位置：HSEG 或 TSEG。

SMRAM 控制寄存器(System Management RAM Control，SMRAMC) 控制 SMRAM 在 CSEG/HSEG/TSEG 区域中的存在位置及从权限低于 SMM 的执行模式对它的访问。这是它的位的描述：

Size: 8	**Default Value: 02h**	
Bit Range	**Acronym**	**Description**
7	RSVD	Reserved.
6	D_OPEN	When D_OPEN = 1 and D_LCK = 0, the SMM DRAM space is made visible even when SMM decode is not active. This is intended to help BIOS initialize SMM space. Software should ensure that D_OPEN = 1 and D_CLS = 1 are not set at the same time.
5	D_CLS	When D_CLS = 1, SMM DRAM space is not accessible to data references, even if SMM decode is active. Code references may still access SMM DRAM space. This will allow SMM software to reference through SMM space to update the display even when SMM is mapped over the VGA range. Software should ensure that D_OPEN = 1 and D_CLS = 1 are not set at the same time.
4	D_LCK	When D_LCK=1, then D_OPEN is reset to 0 and all writeable fields in this register are locked (become RO). D_LCK can be set to 1 via a normal configuration space write but can only be cleared by a Full Reset. The combination of D_LCK and D_OPEN provide convenience with security. The BIOS can use the D_OPEN function to initialize SMM space and then use D_LCK to "lock down" SMM space in the future so that no application software (or even BIOS itself) can violate the integrity of SMM space, even if the program has knowledge of the D_OPEN function.
3	G_SMRAME	If set to '1', then Compatible SMRAM functions are enabled, providing 128KB of DRAM accessible at the A_0000h address while in SMM. Once D_LCK is set, this bit becomes RO.
2:0	C_BASE_SEG	This field indicates the location of SMM space. SMM DRAM is not remapped. It is simply made visible if the conditions are right to access SMM space, otherwise the access is forwarded to DMI. Only SMM space bewteen A_0000h and B_FFFFh is supported, so this field is hardwired to 010b.

系统固件在平台初始化期间设置 SMRAMC 值并锁定寄存器——所有字段都变为只读，直到下一次系统重启（注意热重启和冷启动可能执行路径不同）。在正确配置的系统上，D_LCK 必须为 1，D_OPEN 必须为 0，这意味着只有以 SMM 模式运行的代码才能访问 SMRAM 内存。G_SMRAME 字段控制 CSEG 的存在，C_BASE_SEG 负责 HSEG 和 TSEG。在我的硬件上，C_BASE_SEG 是只读的，预定义值是 010b。

因此，OEM/ODM 厂商应正确配置 SMRAMC 值，以免系统遭受 SMM 攻击。正确设定的 SMRAMC 如下图所示：

```
[x][ ================================================================================
[x][ Module: Compatible SMM memory (SMRAM) Protection
[x][ ================================================================================
[*] PCI0.0.0_SMRAMC = 0x1A << System Management RAM Control (b:d.f 00:00.0 + 0x88)
    [00] C_BASE_SEG        = 2 << SMRAM Base Segment = 010b
    [03] G_SMRAME          = 1 << SMRAM Enabled
    [04] D_LCK             = 1 << SMRAM Locked
    [05] D_CLS             = 0 << SMRAM Closed
    [06] D_OPEN            = 0 << SMRAM Open
[*] Compatible SMRAM is enabled
[+] PASSED: Compatible SMRAM is locked down
```

图 正确设定的 SMRAMC 案例

系统管理范围寄存器 (SMRR: System Management Range Registers) 是一对 MSR 寄存器，包括：IA32_SMRR_PHYSBASE 和 IA32_SMRR_PHYSMASK，它们只能由 SMM 代码修改。因为 HSEG 和 TSEG 内存是可缓存的，因此，必须配置 SMRR 寄存器以保护它免受 SMM 缓存中毒攻击。

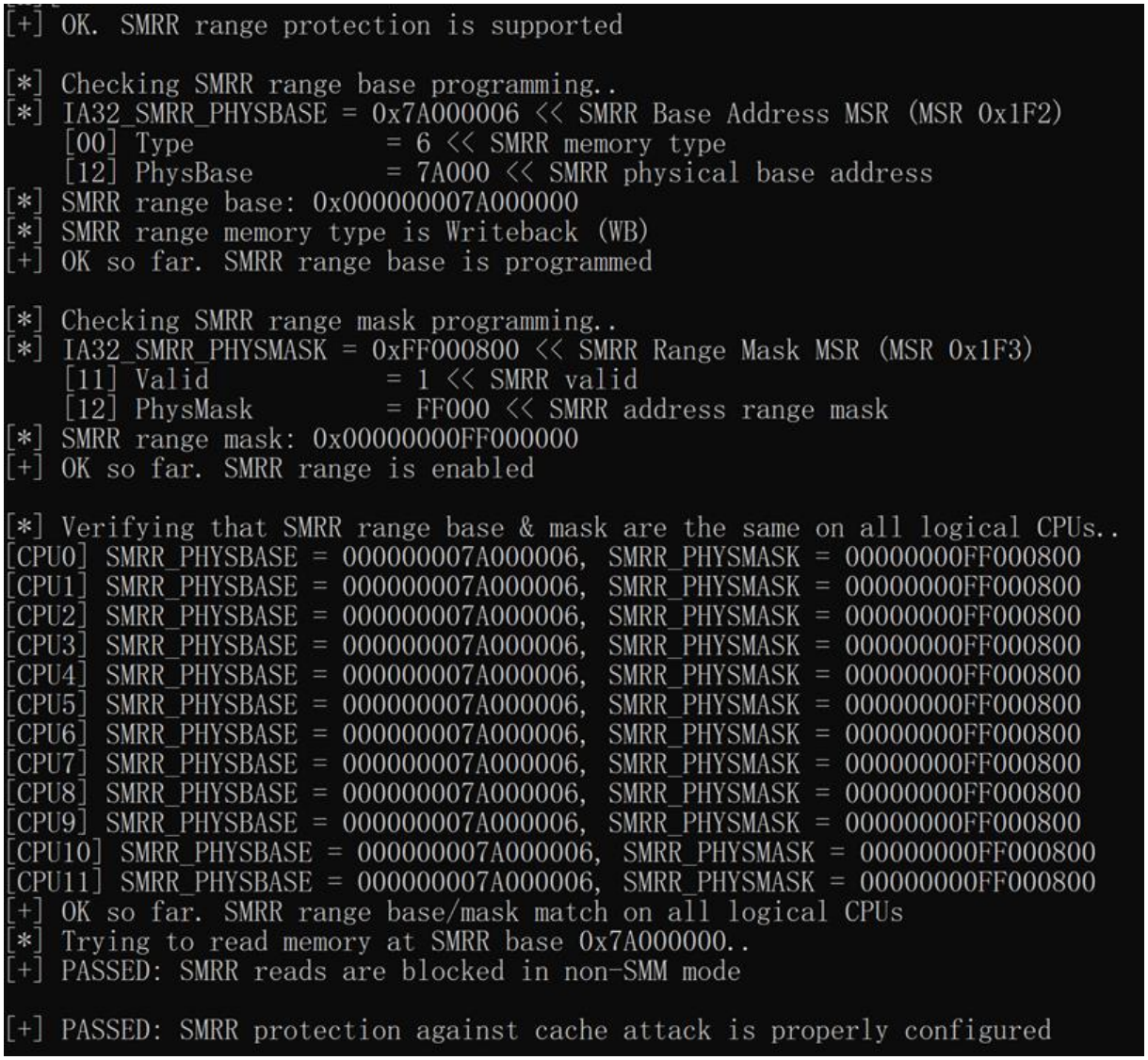


图 SMRR 寄存器正确配置示例

Boot Guard 完整性保护

为了解决 BIOS/UEFI 镜像非授权篡改的问题，英特尔提出了 Boot Guard 安全机制。Boot Guard 是英特尔第 4 代 Core 微架构(Haswell)中引入的一种技术，旨在提供底层固件(UEFI)防护保障，使其免于被恶意篡改。Boot Guard 提供了对 BIOS/UEFI 镜像进行签名验证的能力。

该技术允许 OEM 厂商将使用公钥签名算法的公钥部分烧入 CPU，未经厂商签名的 BIOS 固件将会被 CPU 拒绝执行。

Boot Guard 提供了两种模式：

- Verified Boot 模式下会验证固件签名，且将完全拒绝未通过验证固件的运行。

- Measured Boot 模式下将启动过程的信息记录到 TPM（可信任平台模组） 中，交由操作系统去做后续进一步处理。

Boot Guard 的模式是由 OEM 在出厂前决定的。而几乎所有的笔记本厂商在搭载 Broadwell 笔记本出厂前都将 Boot Guard 设为了第一种模式。

Intel Boot Guard 安全机制引入了一个叫做 ACM（Authenticated Code Module）的组件，它是一组 Intel 提供的，用来验证 BIOS 固件的二进制数据。

同时，Boot Guard 也把 BIOS 分为两个部分：IBB 和 OBB。IBB（Initial Boot Block）一般只包括 PEI 和 SEC 阶段，OBB（OEM boot block）则是剩余的阶段。

CPU 在出厂的时候，ME 是处于 Manufacture mode，这时主板厂商(ODM/OEM)可以通过工具将厂商的公钥部分熔合(Fuse)到 PCH 中的 NVRAM 中。并用签名服务器对 UEFI 镜像用私钥进行签名。

而 Boot Guard 的大致工作原理为：在系统启动的时候，ACM 紧接着 ME 启动，ACM 从 ME 获取安全策略和密钥，之后为厂商初始启动块(Initial Boot Block, IBB)设置一个安全的执行环境。ACM 用在 PCH 中熔合的公钥验证 UEFI 是否由其对应的私钥签名，如果通过，就跳到 Reset Vector 执行，如果签名校验失败，则系统拒绝执行 BIOS 代码，通常表现为死机。

ACM 也是放在 Flash 上，ACM 是由 Intel 用”ACM 私钥”进行签名，而每次开机由 Intel 的 microcode 来验证签名正确与否。

需要注意的是，Intel B 系列处理器去掉了 BOOT Guard。2018 年出品的 i7-8700B、i5-8500B 都去掉了 BOOT Guard。Intel B 系列处理器只会被用于一体机或者迷你机设备上。

Boot Guard 是在 Haswell 这一代 CPU 引入的安全特性。在这里，建议使用 Haswell 及之后的 CPU，以便带来针对 UEFI 固件完整性的保护。

BIOS Guard 完整性保护（PFAT）

Intel BIOS Guard（PFAT）是 intel 芯片组中 BIOS Flash 保护功能的扩充技术，目标为应付恶意代码对 BIOS Flash 日益增加的威胁。它可防止在未经生产商授权下篡改 BIOS 数据，可在遭遇攻击后将 BIOS 恢复至原有的正常状态。

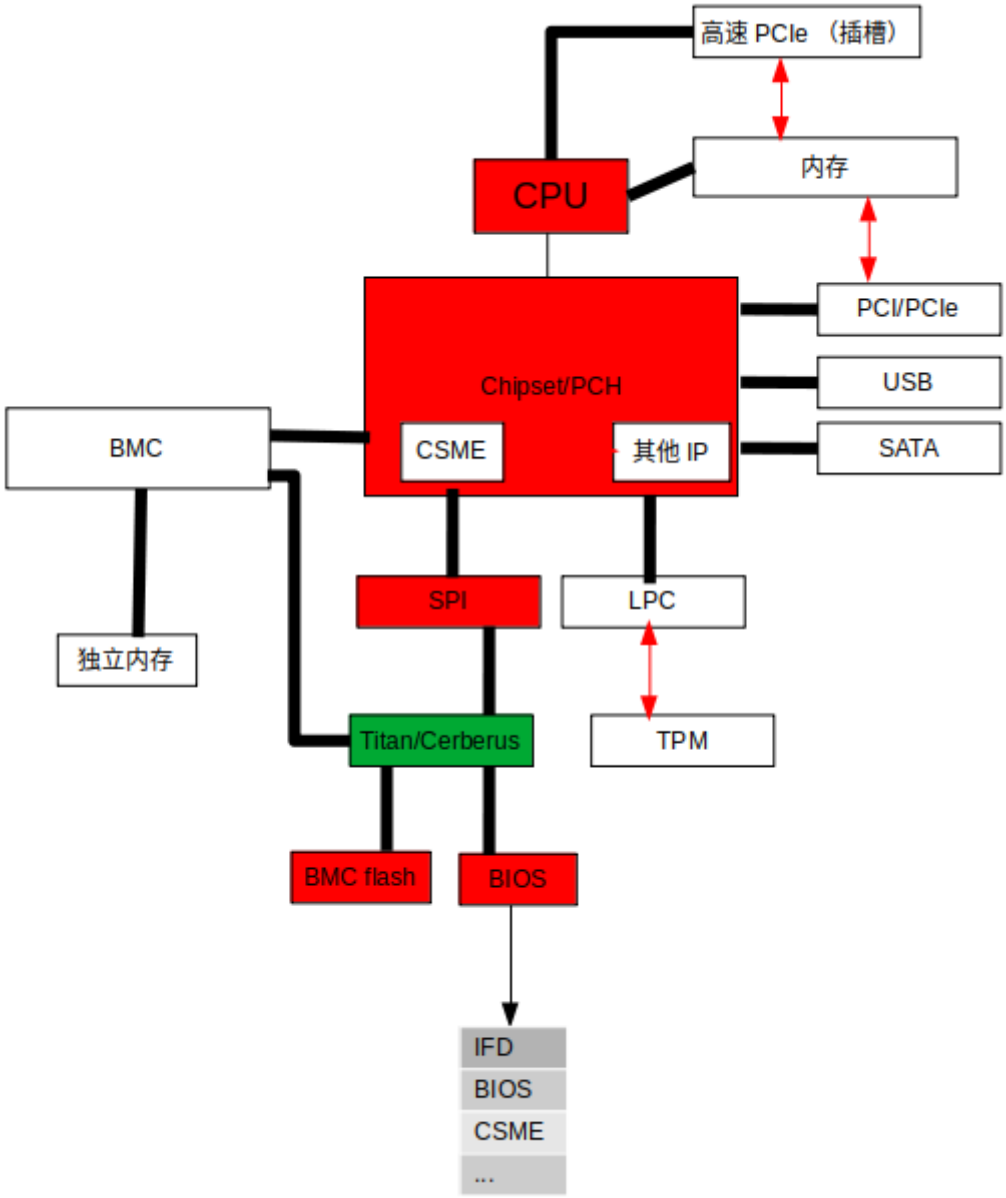
BIOS Guard 可在用户写入主板 Flash 时，对 UEFI 镜像进行验签。如果验签通过，则继续处理，否则拒绝继续更新主板 Flash。BIOS Guard 也是在 Haswell 这一代 CPU 引入的安全特性。建议使用 Haswell 及之后的 CPU。

信任根

Titan 和 Cerberus

在服务器平台上，有更多的固件需要加入到安全启动的范畴，例如基板管理控制器 （BMC）、网络接口卡 （NIC）、RAID 控制器、非易失性快速存储器 （NVMe）存储设备等。常规的安全启动解决方案无法验证所有设备的固件，因为安全启动逻辑仅验证部分固件的完整性和签名，所以需要一种新的解决方案来验证电路板上的所有固件以满足 NIST SP 800-193 关于弹性三部曲的要求，虽然服务器 BMC 可以做到访问主板上的其他固件，但 BMC 其他的功能过多，这个背景下诞生了一些中间层劫持（Interposer）的方案，比如 Google Titan，微软主导的 Cerberus 以及更为复杂的 AWS Nitro 体系。

Titan 是 Google 为 GCP（Google 云平台）开发的硬件信任根解决方案，其开源版本 OpenTitan 提供了一个参考实现。Cerberus 是源于微软早年贡献给 OCP（开放计算项目）的 Olympus 项目后的一个衍生的开放硬件安全项目。Google Titan 和 Cerberus 使用一个安全处理器对 SPI 总线和主板硬件设备之间进行拦截，作为信任根对固件（UEFI，OptionROMs，BMC，Intel ME/SPS 以及其他外设的固件）的签名合法性和完整性进行检查，这一过程对于主 CPU 是无感知的。



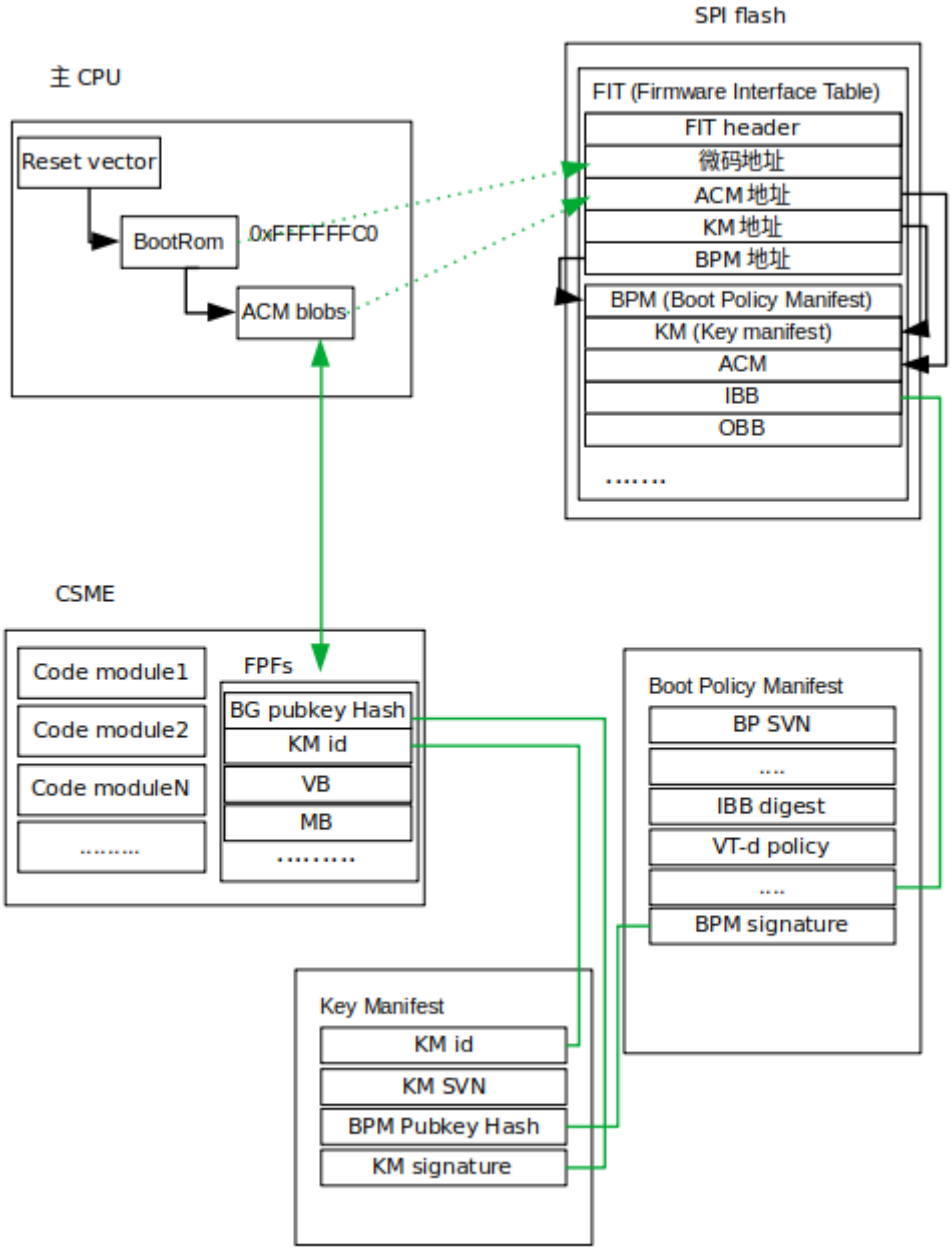
AWS Nitro Security Chip

亚马逊为 AWS 服务器开发了 Nitro 系统，其中 Nitro 安全芯片作为和 Titan/Cerberus 类似的硬件信任根方案，但 Nitro 包括固件规则更新在内的整个执行流都是单向的，唯一更新的途径是基于 Nitro 控制卡，由于 Nitro 系统的复杂性和非通用性，本指南不作过多描述。

Intel Boot Guard

Boot Guard（BG）是 Intel 为 x86 平台提供的信任根方案，完成 BG 的开箱之后，启动过程中传统 x86 的 reset vector 所在的 0xfffffff0 将不再是执行第一条指令的地方，CSME 会先行启动后对 ACM（Authenticated Code Module）进行验签，ACM 是 x86 平台中 Intel 系统中常见的方案（其他 ACM 比如 BIOS Guard ACM，TXT ACM 等），ACM 通常是一段由 Intel 签名的小型二进制程序，微码加载后也会对 ACM 进行验签，而 Boot Guard 系统中的二级密钥由 OEM 厂商控制，ACM 的验签成功后从 CSME 的 FPFs（熔丝）中读取开箱时烧写的 OEM 公钥散列后对 OEM 的 IBB（initial

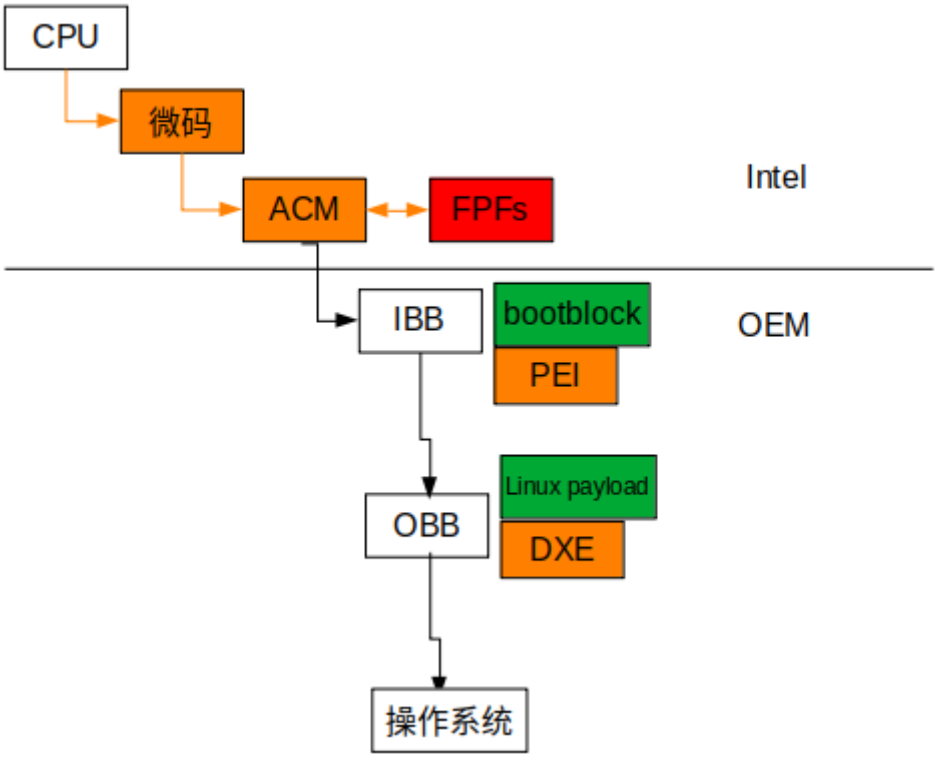
boot block）进行验签，之后继续对其他固件部分进行验签。



Boot Guard 定义了一组结构用于存放签名信息：

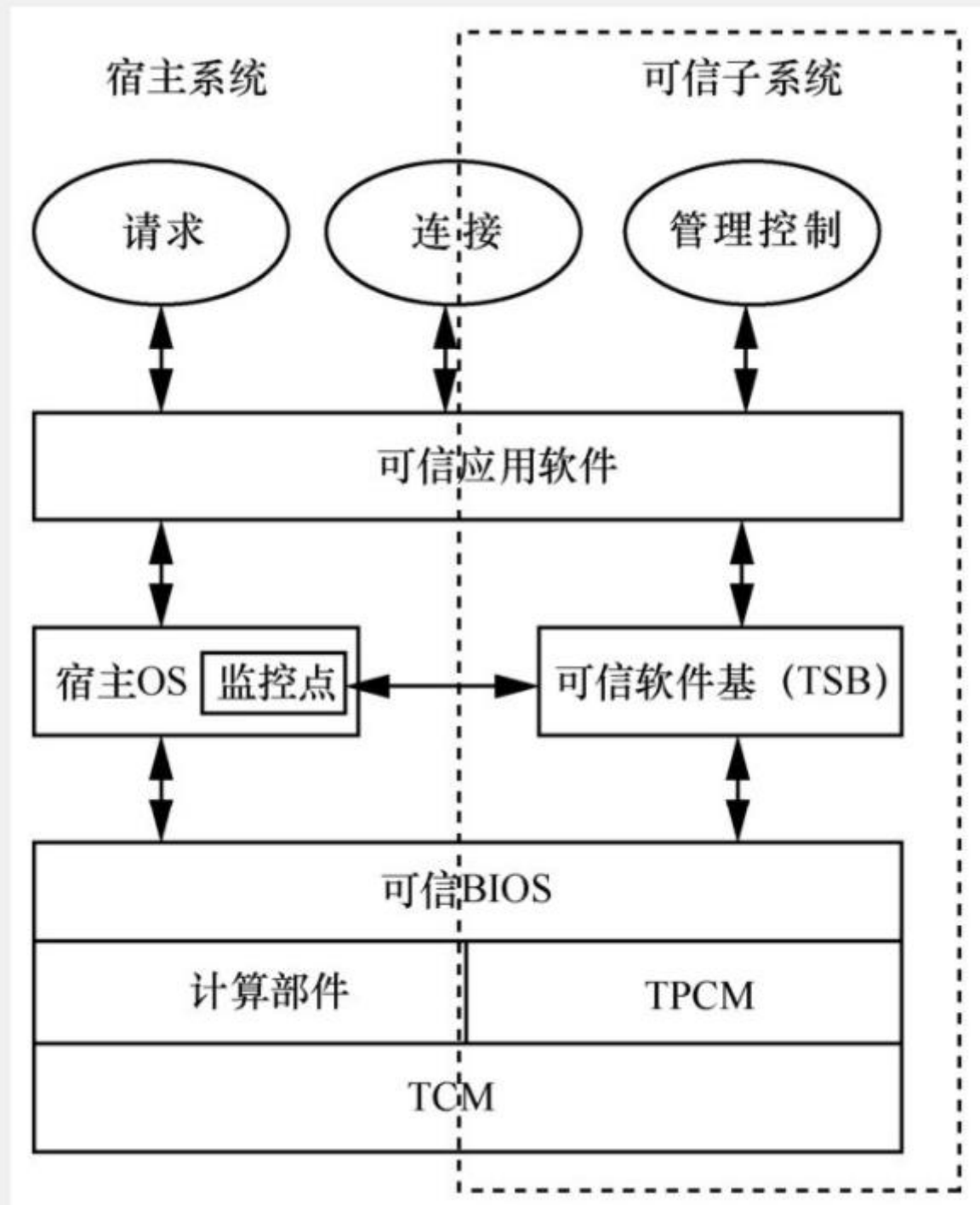
* FIT（固件接口表）：固件镜像的信息表，FIT 地址在 0xFFFFFFC0，FIT 记录了 ACM，BPM 和 KM 的地址信息。
* BPM（启动策略清单）：记录 IBB 的摘要以及被 KMK（Key Manifest Key）签名。
* 密钥摘要：记录开箱时写入 CSME 熔丝的 BG 公钥摘要。

整个 BG 启动前半部分是基于 Intel 签名的 ACM 的验签，后半部分则是 OEM 厂商可控：

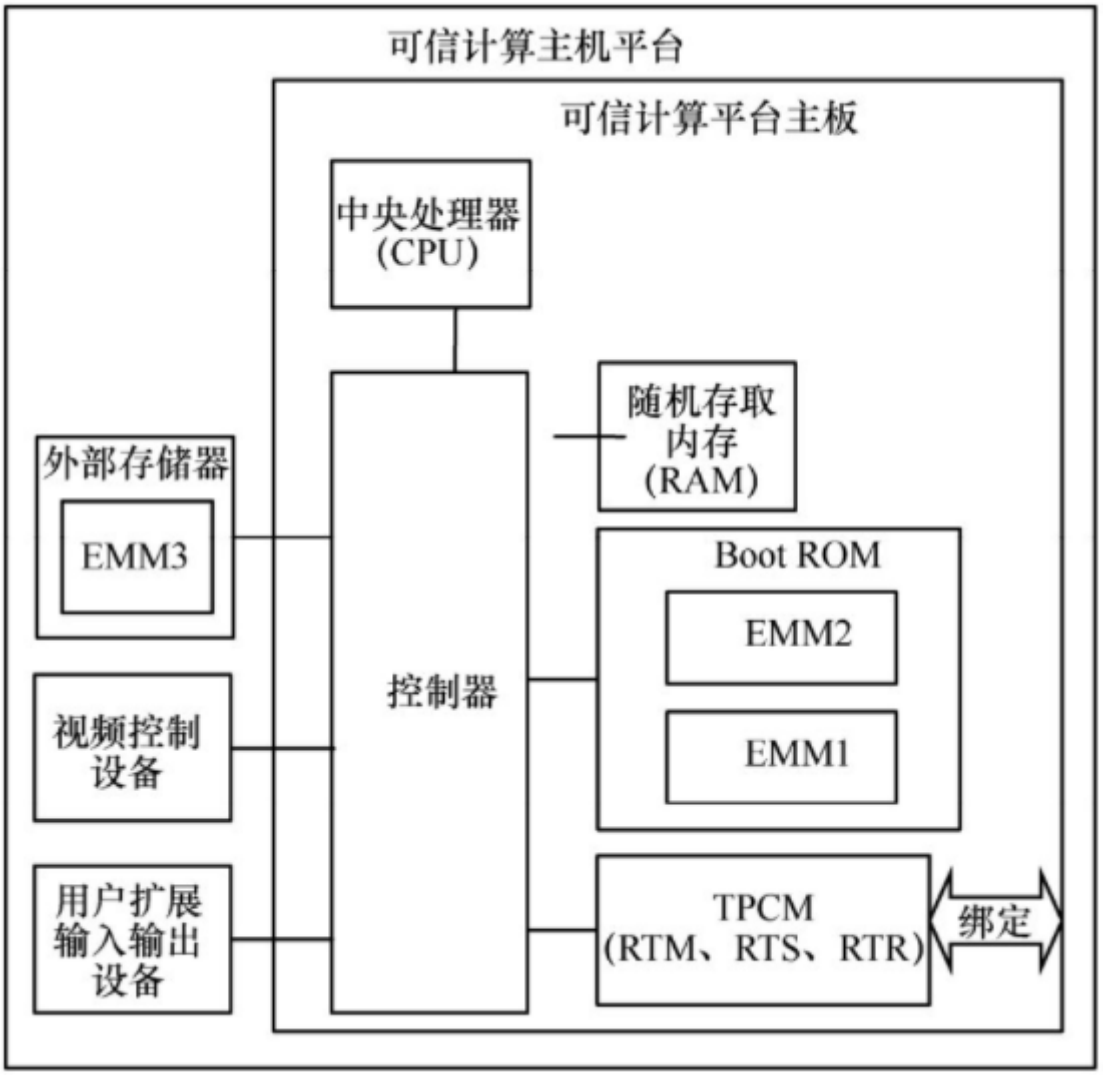


可信 3.0

可信 3.0 是中国独立于 TCG 规范的可信计算标准，目前标准化工作（GB/T 40650-2021）已经完成并于 2022 年 5 月开始实施，其中 TPCM 可以作为信任根的实现：

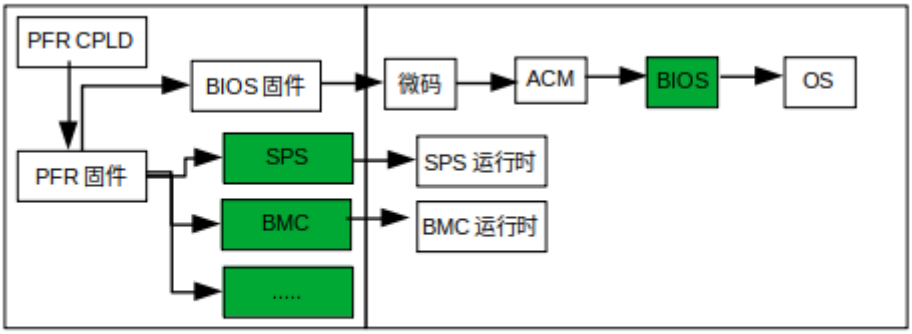


有厂商建议 TEE (Trusted Execution Environment,可信执行环境)采用一个单独固定的计算核心运行安全可信相关的系统和软件，其余 3 个计算核心分配给 REE(Rich Execution Environment，富 / 用户执行环境)运行用户操作系统和应用,二者通过软中断和共享内存的方式进行通信，同时在内存、IO、非易失性存储等方面也进行资源隔离。TEE 中的 TPCM 相关安全可信软件主动对 REE 中运行的用户系统和应用进行动态度量,并根据策略设置,对异常情况进行报警或控制。



PFR (Platform Firmware Resiliency)

PFR（平台固件弹性）是 Intel 为了让 x86 服务器符合 NIST SP800-193 的方案，PFR 的设计类似 Google Titant 和 Cerberus，信任根方案 PFR CPLD 基于 CPLD（Complex Programmable Logic Device）的信任根实现，也用于检测其他固件的完整性，这里也扮演中间层劫持（Interposer）的作用，PFR CPLD 会针对 BIOS flash 中的 BIOS 段和 SPS（服务器版本的 CSME），BMC flash，SMBus（PSU 电源供应单元等组件）进行检测。

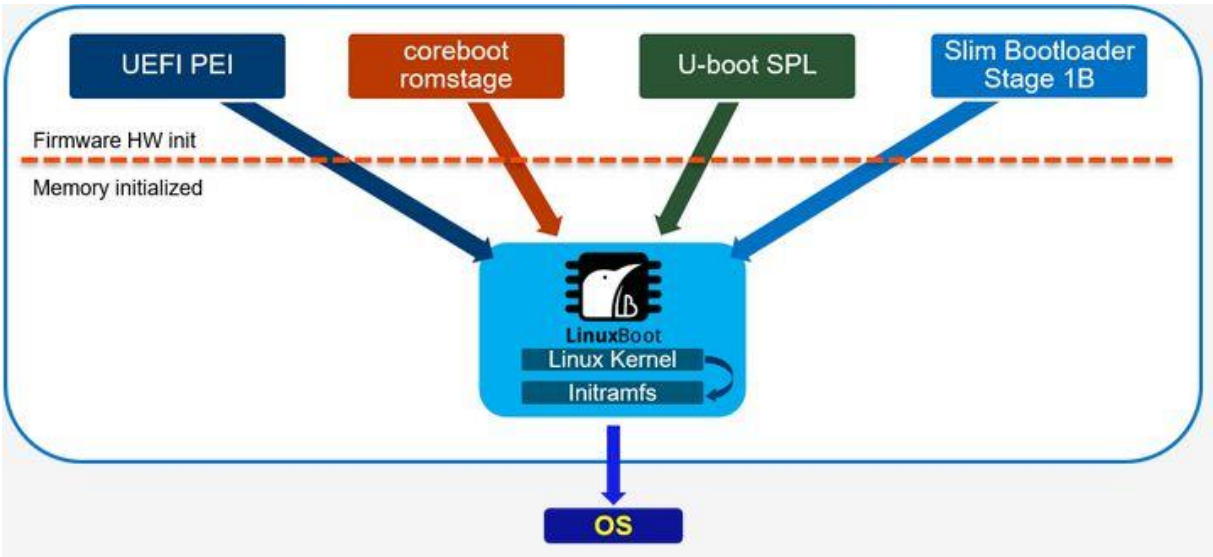


冷启动　　热启动

“下一代”固件背景介绍

1999 年冬天，洛斯阿拉莫斯国家实验室的研究 Ron Minnich 发起了一个叫 LinuxBIOS 的项目，其目标是用自由软件去替代私有的固件，LinuxBIOS 设计思路是让尽量少的代码做硬件初始化的工作，当硬件初始化完成后就加载一个基于 Linux 内核的执行载荷，这个项目后来更名为 coreboot，今天的 coreboot 支持除了 Linux 以外的多种执行载荷，这种架构也成为了 2020 年代当业界重新审视固件问题和探讨”下一代“固件时的重要基础，这或许是必然性和偶然性并存所致。

进入 2010 年代后，业界开始探索各种固件安全的方案，UEFI 中被称为”Secure Boot”是一种以验证签名构建信任链条的 VerifiedBoot 实现，而基于 TPMv1.2/v2.0 的 MeasuredBoot 则因为 Intel 芯片组对 TXT 支持限制以及 TCG 标准中的其他可信计算特性难以在 UEFI 中实现，2016 年，对冲基金 Two Sigma 的研究员 Trammell Hudson 借鉴了 LinuxBIOS 的思路，基于 Linux 开发了一个名为 heads 的执行载荷并用 coreboot 进行加载，把所有包括部分可信计算在内的安全特性都放进了载荷中，这样以低成本的方式解决了 UEFI 生态持续数年都难以解决的问题，业界继续在 1999 年版本的“下一代固件”架构的基础上前行。2017 年，Google 和 heads 社区联合开发了名为 NERF 的执行载荷，NERF 采用了保留 PEI 和最小化 DXE 的模式去兼容 OEM 的 UEFI 固件，但有一些机型的重定向问题难以解决，NERF 另一个特性是用户态基于 Go 运行时，有趣的是，NERF 的尝试让基于 Linux 的执行载荷方案进入到了一个更标准化的阶段：LinuxBoot。



下一代固件安全特性

下一代固件由于硬件初始化和执行载荷在架构上做到了分离，这使得安全特性的设计和实现变得更容易，企业和机构可以更灵活的根据需求去制定自己的策略。

VerifiedBoot

此项特性在 UEFI 厂商的市场宣传中被称为”Secure Boot”，只允许在正常启动过程中加载已签名的内核以及附带数据（例如，initrd）以启动，由于在 UEFI 中签名应与内核存储在同一文件中，因此需要专用工具（例如，shim），并且签名过程难以操作。而在 下一代固件中，签名验证过程可以由捆绑的 gnupg 工具完成，包含公钥的专用 gnupg 钥匙链可以捆绑到 固件载荷的 initrd 中，操作系统的内核、initrd 和引导配置将由存储在智能卡中的相应私钥进行签名，签名将作为独立文件存储在引导分区中，而签名过程也可以在密钥管理的操作系统中完成从而无需修改内核文件本身。如果签名失效，自动启动过程将被中断，目前的下一代固件实现都可以为用户提供 shell 以便手动启动操作系统或修复签名链条。而从信任根的角度，可以把信任链条基于 BootGuard 甚至 PFR 建立。

MeasuredBoot

如果主板配备 TPM，coreboot 可被构建为使用它测量其加载的所有组件（包括载荷执行体），理论上讲，下一代固件可以支持 TPMv1.2 /v2.0 以及可信 3.0，如果启用了 TPM 支持，可以将一段随机秘密封印到 TPM 中一组被 coreboot 用于测量自身的 PCR 上，并在引导过程中将该随机秘密作为 TOTP 呈现出来，以便用户验证。如果固件被更改，PCR 将发生变动，之前封印的、被用户验证过的秘密不能再从 TPM 中解封，以便向用户警告固件被意外修改。这是本地证明的一种形式，如果启动过程中验证 TOTP 不方便，则可以使用远程证明。如果秘密未能解封，自动启动过程将中断，并将提供恢复 shell。

全盘加密

如果 TPM 可用，并且操作系统在启动过程中需要解锁 LUKS，则 可以在 LUKS 中添加随机密钥，并将密钥封印到 TPM 中（无论有没有密码）。如果密钥可以成功解封，固件载荷将复制操作系统的 initrd 到其 tmpfs 中，将该密钥和一条 crypttab 条目附加到复制的 initrd 中，然后用修改后的 initrd 启动操作系统，因此 LUKS 将在启动过程中由密钥解锁。添加密钥后，LUKS 头将被测量进入 PCR，这是封印解锁密钥的 PCR 之一。因此，如果固件或 LUKS 头被更改，密钥将无法解封，自动启动过程将中断，当然，如果改动是因为用户主动更新了固件或改动了 LUKS 头，只要还有其他解锁 LUKS 以添加密钥的手段，用户可以为 LUKS 生成并封印一段新密钥。

基于硬件内置密钥交换的参数加密

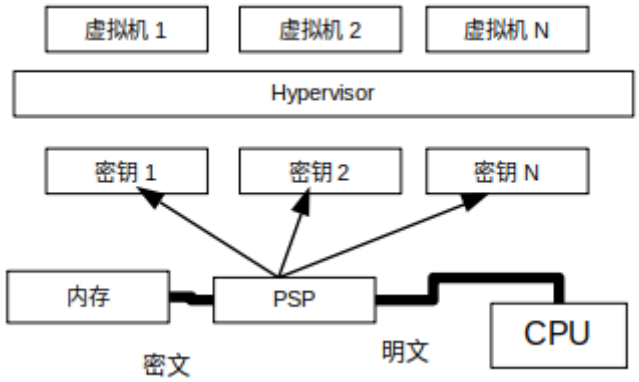
自 2010 年代以来，物理攻击成为了一个古老而重要的话题，如果关键信任基（在本例中为 TPM）遭到破坏，则可能会危及关键基础架构的安全，2011 年以及 2012 年发布的<LPC 总线劫持者指南>和<TPM 通信接口的劫持者指南>揭示了 TPM（直到现在高度依赖 LPC 总线）的攻击面。2018 年 3 月，NCC Group 展示了使用价格低于 180 美元廉价硬件设备方案发起名为 TPM Genie 的攻击，TPM Genie 可以作为 interposer 来发起中间人攻击，以伪造 EK 或简单地嗅探主机和 TPM 之间的流量，而在 2019 年，安全研究人员利用 49 美金的硬件则完成了针对 TPM 嗅探的并且提取 bitlocker 的密钥，所有的公开演示都是 PoC 级别的，这里值得注意的是真实世界的漏洞利用可以在更短的时间内完成针对 TPM 中间人劫持或者嗅探的攻击，目前大部分运行 TPM v1.2/v2.0 的实现都受到影响。下一代固件应对此类威胁可以引入一项新特性：通过会话密钥与 TPM 内部生成的密钥进行交换来加密敏感参数（例如针对明文的封印/解封操作）来免疫此类攻击。

机密计算

机密计算协会对机密计算（Confidential Computing）的定义如下：通过在基于硬件的可信执行环境 （TEE） 中执行计算来保护正在使用的数据。按照这个定义，虽然 TCG 的可信计算，飞地计算以及机密计算可以在术语上进行互换，但当前的机密计算方案更多的是指 AMD SEV 和 Intel TDX。

SEV

AMD SEV（Secure Encrypted Virtualization）发布于 2016 年，是第一款商用机密计算解决方案，用户可以使用指定密钥对虚拟机内存进行加密，针对虚拟机内存加密、密钥管理的实现是基于 PSP（平台安全处理器），加密的内存页只有虚拟机本身可以访问，而共享页的部分可以和宿主机 hypervisor 进行交互，SEV 发布后 AMD 为了保护虚拟机的寄存器状态又发布了 SEV-ES，SEV/SEV-ES 支持前两代的 EPYC 服务器平台，而 SEV-SNP 则只支持第三代 EPYC 平台。



企业平台固件检查列表

固件和硬件的生命周期是同步的，本指南针对从硬件采购前的调研（技术评估，风险评估等），根据威胁模型以及风险评估的综合结论进行芯片安全特性的开箱以及部署，运营和维护过程所需要注意的问题以及硬件和固件生命周期终结后的事项给予一些检查列表。一旦企业或者机构确定了该列表后所制定的策略和程序后，应该将其应用到任意一批新增的硬件采购中，并且在必要的时候（例如高价值数字资产）也应该将其应用于存量的硬件中，列表应该优先应用于对企业或者机构高价值的硬件，例如：

* 需要高级防护应用需求，例如密钥分发，分布式存储元数据索引，金融规则回测的服务器等。
* 符合安全合规（等保 2.0、PCI、HIPAA 等标准）的服务器
* 高层管理、财务和法务部门的笔记本

企业或者机构出于安全性的原因可能触发某些计划外的采购，然而旧硬件可以在不那么敏感的应用中重复使用。企业或者机构在执行所有阶段的过程中所需要的工具最好有开源实现，因为审计和开箱类工具的透明度对于整体安全防护非常重要，如果没有开源实现的情况下，企业即使愿意签署 NDA 但厂商也拒绝分享源代码的话，企业可以考虑更换供应商。

调研以及采购阶段

将硬件和固件纳入威胁模型对于企业来说是至关重要的，一台计算机（服务器，桌面或者笔记本）包含多种固件镜像：微码、BIOS（UEFI，coreboot，以及基于 FSP 开发的其他固件）、每一块适板卡上的固件以及每一个外设的固件。企业需要针对所有硬件对应的固件进行风险评估，可参考本指南之前介绍的固件类型，威胁模型，防护体系等议题进行整体技术评估。

在调研和采购阶段本指南的检查列表如下：

* 请求厂商提供固件以和硬件安全数据信息，企业安全部门可以自行验证，例如，获得一台新型号的硬件样机并且确认它通过了 CHIPSEC 安全测试。
* 确保厂商可以提供固件工具（开箱或者审计用途），测试并且学习使用它们，包括黄金镜像散列值，以及经过签名的更新。
* 企业威胁情报系统涵盖固件，包括普通的 bug 修复，以及重要安全更新。

* 针对固件威胁的内容为技术部门和非技术部门的员工进行培训，比如教育他们诸如邪恶女仆攻击这类通用攻击类型的内容。将固件加入到硬件生命周期管理也非常重要，另外，数字取证与应急响应（DFIR）团队应该扩展他们的检查列表以包括联系固件厂商、OEM 以及独立固件供应商。

* 罗列采购需求中的机型所涉及到的固件类别，作为数字资产的一部分进行生命周期管理。

* 制定企业采购策略：

- 关于所购买的新系统必须拥有的安全特性，例如，验证启动、基于 TPMv2.0 或者可信 3.0 的可信测量启动、信任根、DRTM（动态测量）、CHIPSEC 测试等
- 拒绝购自灰色市场的硬件以避免被提前植入后门的风险
- 罗列新系统所必须配置的安全特性

另外，已部署的系统需要进行风险评估，是应该加速新硬件的采购呢，还是将旧系统部署到不那么敏感的应用中或者制定中短期加固方案。

开箱以及部署阶段

基于企业的安全策略对固件安全特性进行开箱操作，例如，启用或禁用 TPM/TXT、CSME、唤醒特性、固件/BIOS 口令、固件的网络能力、安全启动、信任根、全盘加密等特性。

确保所有厂商固件和签名密钥为最新的签名密钥，在系统清单中注册机器身份和 BIOS 完整性信息，在最初的系统采购过程中，制作企业平台固件初始黄金镜像（基于 CHIPSEC 或者其他方案），保存该文件作为初始基线以用于将来的比对，如果厂商提供了它们自己的黄金镜像，企业依然需要用自己测试的散列值结果同它们的进行比对。

使用经过签名的固件镜像，对于企业自己签名的固件镜像，不论是 UEFI 还是下一代固件，保护密钥都是至关重要，企业需要制定密钥签名的严格措施。理解代码前面以及可能出现的风险，比如某些闭源的二进制文件被签名但无法进行源代码审计。

分级保护

企业不同类型的数字资产重要程度不同，对于运行不同重要性的机器也应该区别对待，以面对不同等级的威胁，本指南给出一个根据业务的重要性制定不同级别的安全建议，高级防护的机型（重要和关键）里需要注意的是负责密钥管理和存放平台固件开箱工具集的机器不论是笔记本还是服务器都必须对应最高级别，另外开箱所涉及的工具建议优先选择开源实现以降低后门风险。

安全特性	普通 (Normal)	重要 (Important)	关键 (Critical)
Linux 安全基线与合规（STIG，CIS，等保 2.0）	有	有	有
Linux 内核安全基线	有	有	有
Linux 内核高级加固	N/A	有	有
安全验证启动	N/A	UEFI secure boot	coreboot
固件基础风险评估	N/A	CHIPSEC	CHIPSEC
DMA 攻击防护	N/A	可选	有，IOMMU/SMMU
安全测量启动	N/A	N/A	有，TPM/TCM
基于信任基全盘加密	N/A	N/A	有
信任基物理攻击防护	N/A	N/A	有
多信任基支持(含信任根)	N/A	N/A	CBnT/TPM/TPCM
硬件级错误注入防护	N/A	N/A	有
DRTM	N/A	N/A	有，TXT/SKINIT
关闭 CSME	N/A	可选	可选，根据版本和场景

运营和维护阶段

保持固件为最新：除了操作系统和应用程序外，也进行固件更新，理想情况是通过经过签名的、自动化的机制，例如 Windows Update、LVFS（Linux Vendor Firmware Service）等，如果企业采用传统的固件特性，应该持续更新自 uefi.org 或者 microsoft.com 的最新安全启动黑名单密钥数据库，利用散列值验证固件更新。

执行周期性的固件安全检测，并且监控具有网络协议栈的固件的网络流量，审计那些建议固件更新的操作系统层级代码，执行固件更新的工具限制为只有认证帐户可以访问应该成为强制访问控

制的一部分。在安全事故中，使用传统恶意软件分析技术以查找固件层级的恶意软件，并且添加固件专用工具，诸如 UEFITool、ACPItool 、基于 radare2，IDA Pro 和 Ghidra 的逆向插件等。

在安全事故中，响应措施应当包括重新获取固件镜像以及重新运行固件安全测试（CHIPSEC 等），以便同之前的镜像/结果进行比对。

查询供应链中的所有厂商的厂商安全咨询站点以获得关于安全性的建议，以及可供使用的新的检测工具，包括主 CPU 厂商、TPM 厂商、操作系统厂商、OEM 厂商、独立固件供应商以及独立硬件制造商等。

事故响应阶段

使用取证工具比较当前的镜像和之前的扫描结果以查找更改，重新运行安全测试，同最后一次已知正确的结果进行比对，以查找攻击者留下的足迹，例如，SPI 保护曾经被禁用，但是现在被启用了。如果系统被操作系统/应用程序层级的恶意软件攻击，它可能已经更新了固件，验证整个系统，进行固件和操作系统层级的测试。利用固件和操作系统的黄金镜像恢复系统。

安全事故之后，应急响应团队将会需要额外的时间以清除一台系统中的固件层级的恶意软件。在某些情况下，系统可能会变砖并且不能恢复，事后分析报告应该包含哪些厂商未能提供足够的工具/镜像以用于固件恢复，应该逐步淘汰这些系统，或者将其重新部署到相对次要的应用中。

废弃处理阶段

废弃处理之前，将固件恢复至出厂状态。废弃处理一台系统之前，除了清除磁盘介质外，将固件重设至一种已知正确的状态，使用厂商或者企业自己设定的黄金镜像，废弃处理一台系统之前，确保存储于固件中的任何配置、用户认证数据、TPM 机密等都被重置。

总结

固件的运行级别高于操作系统所在的 RING 0，如果用户的威胁模型中包含攻击者持久化这一项，那固件安全就无法忽视，这个领域的攻防越来越受到业界的关注，美国国家标准与技术研究院（NIST）早在 2011 年 4 月即发布了 NIST SP 800-147：BIOS 保护指南，2011 年 12 月发布了 NIST SP 800-155：BIOS 完整性度量指南，2014 年 8 月发布了 NIST SP 800-147B：服务器 BIOS 保护指南，以及目前最重要的固件安全合规指南：2018 年 5 月发布了 NIST SP 800-193：BIOS 平台固件弹性指南。2021 年 5 月 26 日，美国网络安全和基础设施安全局（CISA）在 RSA 2021 大会上公开了 VBOS（操作系统以下的漏洞）计划，其目的是推动连同操作系统在内以及更底层组件的安全防护，虽然过去的 15 年固件层面的安全对抗从未停止，VBOS 计划是第一次把隐蔽战争放上了桌面。2022 年 2 月 23 日，美国国防部发布美国信息与通信技术的关键供应链评估报告以回应 EO14017，报告中直接指出固件是最薄弱的环节。

欧洲方面，虽然并没有像美国一样有系统性的固件安全合规指南，但从几个方面可以看到其重视程度，欧盟委员会自从 2014 年发起的“地平线 2020”计划中资助了大量的开源芯片和固件安全项目以提升生态的透明度进而更好的实施弹性（防护，检测和恢复）方案，德国联邦信息安全办公

室（简称 BSI）多次公开的提到对开放固件的支持和跟进工作，2019 年，BSI 认证了德国老牌安全厂商 genua GmbH 的网络安全产品（编号：BSI-DSZ-CC-1085-2019）支持开放固件体系实现固件安全特性。2021 年 10 月，欧盟委员会开启了针对消费级市场强制固件安全更新的立法程序，未来可能会扩展到其他领域比如工业 4.0 以及服务器等。

在全球高级威胁防护的大趋势下，平台固件属于整体防御中核心的环节之一，本指南供中国企业和机构参考。

附录 1：技术术语

- ACPICA，ACPI 组件架构项目（ACPICA）提供了一系列跨平台的 ACPI 工具，诸如 acpidump
- ACPI，ACPI 是一种平台固件技术，它最初是要用于替代即插即用、MP 和高级电源管理。UEFI 拥有其规范并且维护一份与 ACPI 相关的文档的良好列表
- ACPICA，ACPI 组件架构项目（ACPICA）提供了一种参考实现以及一系列跨平台的 ACPI 工具，诸如 acpidump
- BIOS，BIOS 是一种最初用于基于 Intel 的 IBM PC 的平台固件技术。它是一种 8086 实模式技术。Intel 曾经表示它们将会在 2020 年结束基于 BIOS 的平台固件的生命周期，代之以 UEFI。Intel 和少数 IBV 拥有闭源的 BIOS 实现。BIOS 曾经是微软 Windows PC 的主要固件技术，直到 Windows 开始强制要求 UEFI
- SeaBIOS,传统 BIOS 调用的开源实现，需要在硬件初始化完成后运行，SeaBIOS 可以被 coreboot、UEFI DXE 在物理机上启动，或在虚拟机环境中运行
- coreboot，coreboot 是一种最初称为 LinuxBIOS 的平台固件技术，它可以加载诸如 SeaBIOS、GRUB、UEFI 及其他执行载荷，下一代固件的基础架构，coreboot 被用于 Google ChromeOS 系统以及其他高级防护需求的服务器场景
- 直接内存访问，DMA 允许某些硬件子系统，最为明显的是 PCI(e) 以访问主系统的内存，独立于中央处理器（CPU）。它可以被诸如 PCIleech 等恶意硬件攻击。主要的防护机制是 IOMMU/SMMU 硬件和操作系统支持
- VaultBoot，VaultBoot 是一种具备下一代平台启动固件安全特性的载荷，它包括一个作为 coreboot 或者 UEFI 的载荷运行的最小化 Linux 以提供一种安全、灵活的启动环境
- TrenchBoot，支持 AMD 的 DRTM 方案 SKINIT 的开源实现
- 独立 BIOS 厂商，独立 BIOS 厂商（IBV）为 OEM/ODM 提供集成式固件解决方案，随着 UEFI 取代 BIOS，某些 IBV 现在改称自己为 IFV（独立固件厂商），某些 OEM 仍然将其消费者级别的设备固件外包给 IBV，而为其商用级设备制作自己的固件。范例包括：
  - AMI
  - Insyde
  - Phoenix
- Intel Boot Guard，Intel Boot Guard 是一种固件安全技术，它在 UEFI 安全启动发生之前帮助保护启动过程，一旦 Boot Guard 被启用，它将不能被禁用
- JTAG，JTAG 是芯片的硬件接口以允许访问固件，它被固件工程师用于开发，以及被邪恶女仆攻击者在厂商在其消费者设备中将 JTAG 接口遗留为暴露状态时使用
- LAVA，LAVA 是一种自动化的验证架构，主要目的是测试基于 ARM 设备上的 Linux 内核构建的系统的部署，特别是对于 ARMv7 及更新的设备
- LinuxBoot，LinuxBoot 是一种平台固件启动技术，它将诸如 UEFI DXE 阶段的特定固件功能替换为 Linux 内核和运行时

- 管理模式， 管理模式是 UEFI 所使用的词语，用以指代 Intel SMM 和 ARM TrustZone，即 CPU 的一种特权执行模式
- 管理系统实现于一块独立的处理器上，并且通常是在专用网络接口上，以有利于带外访问和控制，比如 Intel CSME 和 AMD PSP，其运行定制过的 MINIX 和 Linux 系统
- AMD PSP，AMD PSP（平台安全处理器）是 AMD 系统上的一块安全处理器，它可以运行诸如 fTPM 等固件应用程序
- 基板管理控制器，BMC 是用于管理服务器固件，包括应用更新的接口。OpenBMC 和 u-bmc 是主要的开源 BMC 实现
- DASH，DMTF DASH 是一组用于台式机的带外固件管理规范，Intel AMT 是一种符合 DASH 的实现，AMD SIMFIRE 也是
- Intel AMT，Intel AMT 是 Intel 系统上的一种平台固件管理技术，它作为应用程序运行于 Intel CSME 处理器上，AMT 提供诸如远程 KVM、电源控制、裸机操作系统恢复与重建镜像以及远程预警等服务
- Intel CSME， Intel CSME 是 Intel 系统上的一块管理和安全处理器，它可以运行 Intel 主动管理技术（AMT）、高级风扇速度控制、Boot Guard 和安全启动、通过局域网的串口以及基于固件的 TPM（fTPM）
- IPMI，IPMI 是一种平台固件管理技术，通常位于 Intel 或者 AMD 的服务器系统上，通常被实现为一种嵌入式的 Linux 系统，尽管广泛使用，IPMI 的现代替代品是 Redfish
- OpenBMC，OpenBMC 项目是一个用于拥有 BMC 的嵌入式设备的 Linux 发行版
- u-bmc，相对激进的开源 BMC 方案，尝试用 gRPC 替代 IPMI
- Redfish，DMTF Redfish 是一种带外固件管理技术，用于取代 IPMI
- SMASH，DMTF SMASH 是一组用户服务器的带外固件管理规范，与 DASH 相似
- 微码，微码是一种用于 CPU 的固件形式，系统需要微码更新，如同它们需要平台固件更新以及操作系统更新那样
- 原始设备制造商，一家 OEM 构建并销售原始硬件
- 原始设计制造商，一家 ODM 构建硬件并且将其销售给 OEM
- 操作系统厂商，OSV 是一家操作系统厂商，它包括固件/操作系统的交互
- OptionROM，OptionROM，又称为扩展 ROM、OpROM、XROM，是一块 PCI/PCIe 设备上的固件 blob，OptionROM 是来自 BIOS 时代的术语，当时一块板卡将会与 BIOS 平台固件挂钩，并且为新的板卡添加额外功能，OptionROM 是位于该板卡的闪存中的 BIOS/UEFI 驱动程序。一块板卡可能需要多种驱动程序，分别用于每一种架构以及每一种平台固件类型（BIOS+x86_64、BIOS+ARM、UEFI+x86_64、UEFI+ARM 等）。OptionROM 并不构成这样的设备上的全部固件，由于用于诸如 RAID 或者 TCP 工作量卸载等设备功能的操作固件可能是与之完全分离的
- PCIe，PCIe 是 PC 主板的接口。PCIe 设备的固件包括 OptionROM。此设备可能拥有一块对于系统主板不可见的处理器，难以完全信任 PCIe 硬件
- 安全启动，安全启动是一个经常与 UEFI 安全启动相关联的词语，后者是 UEFI 的一种可选安全特性，用于帮助防护启动过程，它并不需要 TPM。除了 UEFI 以外，其他固件技术也会使用安全启动这一词语，有时是小写。苹果的基于 EFI 的安全启动实现不同于用于 Windows/Linux 系统的安全启动技术
- SMM，系统管理模式（SMM）是 Intel 和 AMD 系统中的一种处理器模式，区别于实模式和不同的保护模式，它赋予了对于处理器的完全控制权。由 SMM 托管的应用程序，诸如恶意软件，对于通常的基于保护模式的代码不可见
- SPI，SPI 是一种用于连接外部设备的接口。在 x86 平台上一般被用于连接存储平台固件的闪存芯片。它在开发过程中被厂商使用，并且也可以被攻击者使用，如果它在消费者产品中仍然保持开启状态
- 可信执行环境， 也称为安全执行环境（TEE）,它是虚拟机监视器或者其他技术的一个范例，通过限制固件以使其更加安全，ARM TrustZone 是 TEE 的一个范例

- 雷电，由 Intel 和苹果开发的一种外部外设硬件接口，它合并了 PCIe、DisplayPort 和直流电源
- Tianocore，Tianocore 是 UEFI 论坛的开源实现的起源，厂商将其代码同闭源驱动程序以及增值代码一起使用
- TrustZone，TrustZone（TZ）是用于 ARM 系统的一种固件安全技术，它是一种形式的 TEE/SEE，后者被 UEFI 称为管理模式
- TPM，TPM 是众多平台固件实现的可信根，诸如 Intel/AMD 的 BIOS 和 UEFI 系统，TPM 由可信计算团伙（Trustworthy Computing Group，TCG）定义，有独立的 TPM 芯片，也有称为 fTPM 的“软”固件 TPM 实现，它们由诸如 Intel CSME、AMD PSP 等实现
- 可信启动，可信启动是一种来自可信计算小组的固件安全技术，它使用 TPM 来帮助防护启动过程
- 可信计算小组，可信计算小组（Trustworthy Computing Group，TCG）是一个业界贸易组织，它控制着 TPM 及其相关规范
- U-Boot，U-Boot 可以加载诸如 SeaBIOS、UEFI 等其他负载，U-Boot 和 coreboot 广泛应用于嵌入式系统
- UEFI，UEFI 是一种最初由 Intel 创建，现在被 Intel、AMD、ARM 及其他系统所使用的平台固件技术，它最初是为 Intel Itanium 而设计并且作为 BIOS 的替代品的，UEFI 也是 EFI，基于 UEFI 的平台固件技术通常也称为 BIOS，其中的老旧 BIOS 称为传统模式（Legacy Mode）或者 CSM（兼容性支持模式，Compatibility Support Mode）
- UEFI DBX，UEFI DBX UEFI 安全启动黑名单文件包含最新的 UEFI 安全启动 PKI 黑名单/过期密钥。检查您的厂商文档以查看您的系统厂商的工具如何工作以获取并应用此文件到您的系统；如果厂商并未提供工具，请要求它们提供
- UEFI 论坛，UEFI 论坛是一个业界贸易组织，它控制着 UEFI 和 ACPI 规范以及 UEFI SCT 测试，并且提供开源的 UEFI 实现 Tianocore
- USB，通用串行总线（USB）是一种用于外部外设的业界标准。USB 设备可以被配置为多种设备，并且诸如 Hak5 的 Rubber Ducky 等恶意 USB 硬件可以戏弄天真的操作系统

附录 2：工具

- acpidump，来自 ACPICA 的跨平台、存在于操作系统中的工具以转储并且诊断 ACP 表
- ACPICA 工具，提供工具和 ACPI 的一种参考实现
- BIOS 实现测试套件，Intel BIOS 实现测试套件（BITS）提供了一个可启动的预操作系统环境以测试 BIOS，特别是它们对 Intel 处理器、硬件和技术的初始化。它包括一个编译为原生 BIOS 应用程序的 CPython
- tpm2-tools，tpm2.0 工具集实现，可为本地和远程证明实现提供基础
- go-tpm，go 语言的 TPM 实现
- go-attestation，go 的可信计算远程证明实现
- Keylime，基于 python 和 rust 实现的远程证明管理和部署系统
- DarwinDumper，DarwinDumper 是一个开源项目，它是一系列脚本和工具，以提供一种便捷的方法来快速采集关于您的 OS X 系统的系统概要
- Eclipse UEFI EDK2 向导插件，此 Eclipse 插件帮助 EDK2 开发者使用带有 CDT 的 Eclipse 集成开发环境以进行 UEFI 开发
- EFIgy，Duo Security 的 EDIgy 是一个以苹果 Mac 为中心的开源工具，以检查系统是否拥有最新的 EFI 固件
- Firmadyne,Firmadyne 是一个自动化的可扩展系统，用于对基于 Linux 的嵌入式固件进行模拟

和动态分析

- Firmware.re，Firmware.RE 是一个免费的服务，它可以解包、扫描并且分析几乎任何固件包，并且有助于快速监测漏洞、后门和所有类型的嵌入式恶意软件
- GRUB，GRUB 是一种多重启动的引导程序。它可被编译为 BIOS 或 UEFI 应用程序、以及 coreboot 载荷
- Linux Shim，Shim 是一种 UEFI 引导程序，它会加载另一个 UEFI 引导程序，可能具有不同的许可证，或者由其他厂商签名。有多种已公开的 Shim 分叉
- Fedora 关于 UEFI 安全启动 Shim 的指南
- HardenedLinux 关于 UEFI 安全启动 Shim 指南
- Linux Stub，Linux 内核可以被如此构建，以使得它同时是 BIOS 和 UEFI 引导程序
- CHIPSEC，CHIPSEC 是一种由 Intel 创建的安全工具，以测试 Intel BIOS/UEFI 的安全状态。它是当前唯一能够检查多种公开的固件安全漏洞的工具
- eficheck，此工具仅存于最近版本的 macOS 中，并且没有在 https://apple.com 以文档形式记载。它可以验证 UEFI 完整性和安全性
- 固件测试套件，固件测试套件（FWTS）是由 Ubuntu Linux OSV，Canonical 创建的一系列固件测试工具，以帮助测试系统中将会导致 Ubuntu 出现问题的缺陷。FWTS 是用于多种技术的数十种测试的套件。UEFI 论坛推荐 FWTS 作为主要的 ACPI 测试资源。FWTS 是一个用于 Linux 的命令行工具，并且包括可选的 CURSES UI 以及可选的 FWTS-live live 启动发行版。FWTS 包含在 Intel 的 LUV Linux 发行版中
- FlashROM，FlashROM 是一个以 Linux/BSD 为中心的工具，用于识别、读取、写入、验证和擦除闪存芯片。它被设计为烧录主板、网卡、显卡、存储控制器其他不同编程设备上的 BIOS/EFI/coreboot/固件/选项 ROM 镜像等，有对于 Windows 的部分支持
- fiedka，固件镜像可视化和编辑工具
- 黄金镜像，黄金镜像是厂商的原始固件二进制文件，这一词语也被用于操作系统镜像。更好的厂商提供镜像和工具以便将使用过的硬件或是购自灰色市场的硬件重置为一种已知状态。在信任任何下载到的二进制文件，诸如黄金镜像之前，它应该同散列值进行比对。大多数厂商并不提供它们的镜像的散列值
- Linux UEFI Validation，LUV 是一种由 Intel 创建的 Linux 发行版以测试 OEM 的 UEFI 实现，它捆绑了 CHIPSEC、FWTS 及其他固件测试。LUV 可通过 LUV-live 的二进制形式获得，这是一个 live 启动的发行版
- Linux 厂商固件服务，也称为 LVFS 或者 fwupd，是一种用于 Linux OEM 的固件更新服务，它极好地提供了一种标准化的系统。使用它的 OEM 正在严肃地对待 Linux 的兼容性和安全性。在微软 Windows 上，一种类似的机制通过 Windows Update 来工作
- Huffman11，Huffman 解码
- me_cleaner，清除 CSME 非必须的代码模块
- me-tools，解压 CSME 的代码模块
- MEAnalyzer，静态分析 CSME 信息
- MCExtractor，Intel，AMD 和 VIA 微码提取工具
- unME12，Intel CSME 12 解包工具
- intelmetool，CSME 运行时信息显示
- PSPTool，AMD PSP 静态分析工具的开源实现
- 微软 Windows Update，除了提供操作系统软件层级的更新以外，Windows Update 还可以通过标准化的胶囊提供固件更新。这些更新必须被固件/硬件厂商验证，并且可以被 EV 签名
- Pawn，Google Pawn 是一种以 Linux 为中心的在线固件工具，它可以将平台固件镜像转储为文件，以供以后进行离线分析
- PhoenixTool，PhoenixTool 是一种第三方免费软件以操作（U）EFI 和少数基于传统 BIOS 的

固件 blob

- rEFInd，rEFInd 是 rEFIt 的继任者，一种允许您选择多种操作系统的 UEFI 引导程序
- RU.EFI，RU.EFI 是一种拥有多种功能的第三方免费固件工具，它以类似 MS-DOS 或者 UEFI Shell 工具的方式工作
- RWEverything，RWEverything（RWE）是一种拥有多种功能的第三方免费固件工具。此工具工作在 Windows 上。如果 CHIPSEC 的 Windows 内核驱动程序未被加载，则 CHIPSEC 工具可以使用 RWE 内核驱动程序
- Sandsifter，Sandsifter 是一种 x86 模糊测试工具
- converged-security-suite，针对 Intel 的 DRTM 方案的工具实现
- UEFI Utilities，UEFI Utilities 是一系列 UEFI Shell 工具的集合，以提供系统诊断信息（它也包含一份 ThinkPwn.efi 副本，当心）
- UEFI 固件语法解释器，UEFI 固件语法解释器检查固件 blob，主要是 UEFI 的固件 blob
- UEFITool，UEFITool 是一种对固件 blob 进行语法检查的图形用户界面程序，主要是对于 UEFI 固件 blob。除了 UEFITool Qt 图形用户界面工具以外，UEFITool 源代码项目也包括若干非图形用户界面的命令行工具，包括 UEFIDump
- radare2，开源的逆向工具集
- efiSeek ，逆向 UEFI 固件的 Ghidra 插件
- UEFI_RETool，逆向 UEFI 固件工具，支持 radare2 和 IDA Pro
- efiXplorer ，逆向 UEFI 固件的 IDA Pro 插件
- meloader，CSME 加载器的 IDA Pro 插件
- smutool，dump 出 AMD SMU 的工具
- Visual UEFI，Visual UEFI 是 Visual Studio 的一种插件，使得 Visual Studio 用户能够进行 UEFI EDK2 开发而无需知道 EDK2 的构建过程细节，而这种构建过程与 Visual Studio 并不相似
- zenfish IPMI 工具，由 SATAN fame 的 Dan Farmer 提供的 IPMI 安全测试工具

引用

AMD 产品安全：

https://www.amd.com/en/corporate/product-security

Intel 安全中心：

https://www.intel.com/content/www/us/en/security-center/default.html

浪潮安全中心：

https://en.inspur.com/en/security_bulletins/index.html

联想安全中心：

https://support.lenovo.com/us/en/product_security/home

Vector-EDK - Hacking Team

https://github.com/hackedteam/vector-edk

DerStarke

https://wikileaks.org/ciav7p1/cms/page_3375125.html

QuarkMatter

https://wikileaks.org/ciav7p1/cms/page_21561431.html

TrickBoot

https://eclypsium.com/2020/12/03/trickbot-now-offers-trickboot-persist-brick-profit/

LoJax

https://www.eset.com/fileadmin/ESET/US/resources/datasheets/ESETus-datasheet-lojax.pdf

FinSpy

https://securelist.com/finspy-unseen-findings/104322/

ESPecter:

https://www.welivesecurity.com/2021/10/05/uefi-threats-moving-esp-introducing-especter-bootkit/

HardenedVault 白皮书：赛博堡垒锻造之路

https://zhuanlan.zhihu.com/p/406518045

Info about CSME

https://github.com/hardenedlinux/firmware-anatomy/blob/master/hack_ME/me_info.md

The Intel® Converged Security and Management Engine IOMMU Hardware Issue - CVE-2019-0090 and CVE-2020- 0566

https://www.intel.com/content/dam/www/public/us/en/security-advisory/documents/cve-2019-0090-whitepaper.pdf

Nightmares(Meltdown/Spectre/L1TF) never goes away

https://hardenedlinux.github.io/system-security/2018/08/16/meltdown_spectre_l1tf.html

A Memory Encryption Engine Suitable for General Purpose Processors

https://eprint.iacr.org/2016/204.pdf

Titan silicon root of trust for Google Cloud

https://keystone-enclave.org/workshop-website-2018/slides/Scott_Google_Titan.pdf

The Nitro Project - Next Generation AWS Infrastructure

https://old.hotchips.org/hc31/HC31_T1_AWS_Nitro_Hot_Chips_20190818-2.pdf

OpenTitan

https://github.com/lowRISC/opentitan

Project Cerberus

https://github.com/opencomputeproject/Project_Olympus/tree/master/Project_Cerberus

OpenBMC

https://github.com/openbmc/openbmc

u-bmc

https://github.com/u-root/u-bmc

Building Secure Firmware: Armoring the Foundation of the Platform

https://www.amazon.com/Building-Secure-Firmware-Armoring-Foundation/dp/1484261054

Platform Firmware Security Defense for Enterprise System Administrators and Blue Teams

https://preossec.com/products/Platform-Firmware-Security-Defense-for-Enterprise-System-Administrators-and-Blue-Teams.pdf

pcileech

https://github.com/ufrisk/pcileech

Thunderclap: Exploring Vulnerabilities in Operating System IOMMU Protection via DMA from Untrustworthy Peripherals

https://thunderclap.io/thunderclap-paper-ndss2019.pdf

Intel Boot Guard

https://edk2-docs.gitbook.io/understanding-the-uefi-secure-boot-chain/secure_boot_chain_in_uefi/intel_boot_guard

Open source cache as ram with Intel Bootguard

https://9esec.io/blog/open-source-cache-as-ram-with-intel-bootguard/

可信计算 3.0 工程初步 第二版

https://item.jd.com/12482068.html

GB/T 40650-2021 信息安全技术 可信计算规范 可信平台控制模块

http://openstd.samr.gov.cn/bzgk/gb/newGbInfo?hcno=18C7FD6FBC935007E0ACEA03DDF28AD0

从端到云基于飞腾平台的全栈解决方案白皮书

https://www.phytium.com.cn/Site/theme/Uploads/20210519/%E4%BB%8E%E7%AB%AF%E5%88%B0%E4%BA%91%E5%9F%BA%E4%BA%8E%E9%A3%9E%E8%85%BE%E5%B9%B3%E5%8F%B0%E7%9A%84%E5%85%A8%E6%A0%88%E8%A7%A3%E5%86%B3%E6%96%B9%E6%A1%88%E7%99%BD%E7%9A%AE%E4%B9%A6.pdf

TPM Genie

https://raw.githubusercontent.com/nccgroup/TPMGenie/master/docs/NCC_Group_Jeremy_Boone_TPM_Genie_Whitepaper.pdf

AMD Secure Encrypted Virtualization (SEV)

https://developer.amd.com/sev/