安全固件开发最佳实践

关于

本文档由云安全行业峰会 (CSIS) 制作。CSIS 是一组云服务提供商，其使命是在开发最佳安全解决方案的愿景和方法上保持一致。该小组包括来自顶级云服务提供商的成员，作为一个行业团队合作，并发展一种协调的方法来提高从组件到系统再到解决方案的云安全性。英特尔为团队提供便利。

有关 CSIS 章程和范围的更多信息，请访问www.cloudsecurityindustrysummit.org

我们感谢以下对文档的贡献者（按字母顺序列出）：

- Ben Stoltz, Google
- Matt King, Oracle
- Nathan House, Rackspace
- Paul McMillan, Netflix
- Rob Wood, NCC Group
- Stuart Yoder, Arm
- Sumeet Kochar, Lenovo
- Shawn Chang, HardenedLinux
- Tobias Langbein, ZKB
- Yigal Edery, Kameleon

文档目的和范围

本文档提倡固件开发实践，这将提高安全性和可靠性。本指南通常对软件项目很有用，但是，这里的重点是大型数据中心使用的固件。本文档由 CSIS 供应链工作组的成员与开放计算项目 (OCP) 安全工作组的成员合作编写。

虽然本文档无意提出具体的实施要求，但引用的许多要点都说明了应遵循的最佳实践，以保持这种所需的安全级别。这些建议并不意味着禁止。可以考虑满足要求的替代解决方案，并鼓励超越它们。

在各种 NIST 标准（例如 800-147B、800-193）中提出了实现服务器弹性的特定固件要求，并且在 CSIS 立场文件(下载链接[^1])中也有解决。

开发固件项目的范围包括但不限于：

- 存储在非易失性存储器中的软件，用于处理低级硬件初始化和事件（例如上电、复位事件）。固件还可以实现产品的主要运行时特性。
- 用于诊断、刷写等的工具。
- 操作系统或引导加载程序的驱动程序能够使用硬件/固件，包括 Option ROM。
- 嵌入在外围硬件设备中的固件。

这些安全开发的最佳实践将适用于所有上述和相关的开发。

OCP-Security 涵盖了与维护固件完整性和确保加载正确固件有关的固件要求，包括：

- 建立和维护系统的信任根 (RoT)。
- 固件映像的签名和验证以及安全启动的证明。

- 固件更新过程和程序。

本文档确实包括一些针对云提供商特定主题的评论，尽管此处未涵盖固件安全的这些方面，以避免重复和混淆。

在本文档中，应遵循的最佳实践标有复选框 (□)。这是为了让"只想把正确的事情做好"的读者更容易理解。

分叉此文档

本文档包含跨多个用例的建议，并从许多来源提取信息，包括作者的经验。请将其用作满足您自己的要求、设计和其他项目特定需求的资源。

欢迎对本文档做出更多贡献。请提交您的建议，或分叉并创建您自己的定制版本。

为什么固件安全很重要

固件是计算机系统、设备和相关基础设施的重要威胁媒介。如果设备开机时执行的第一个代码遭到破坏，那么整个系统就不能也不应该被认为是安全的。固件可能会因恶意攻击或无意中受到损害。

固件攻击可以通过破坏现有功能或通过用恶意代码替换预期的固件来发起，就像用 UEFI rootkit 注入系统的情况一样。这些此类情况通常通过增加平台的完整性和弹性来解决和最小化，并且已在我们之前的 CSIS 立场文件中提出。

固件也可能有代码漏洞，这可能允许攻击者远程访问设备，或通过调用可利用的运行时固件服务并在更高特权的上下文中执行任意代码来获得更高的特权。

当硬件或软件无法保护固件（例如，操作系统的写入操作）时，固件可能会被无意覆盖。

我们必须始终能够完全相信硬件只会运行预期的安全固件。这代表了最基本的信任根。实现这一目标需要固件开发人员提高关注度和安全意识，这是本文档的核心。

威胁模型

对系统的安全目标有一个共同的理解是很重要的。正式的威胁模型允许就系统用例的保护进行合理的讨论。随着整体系统的复杂性和贡献者数量的增加或考虑新的用例，威胁模型变得更加重要。那些超出特定项目范围的威胁也必须记录在案。

在本文中，我们专注于提高对给定时间系统上运行的代码和配置的信心。可以破坏这种信心的条件或事件被视为威胁。

例子

对于处理元件 (CPU) 或其他可编程逻辑设备（例如 CPLD、FPGA），实现初始软件加载和执行的掩膜 ROM 可能容易受到恶意或畸形外部输入（例如存储在 SPI 闪存上的代码和配置）的攻击芯片、硬件配置引脚或通过调试接口（例如 JTAG）进行操作。

Asset	**Threat**	**Mitigation**
初始程序加载完整性（掩码 ROM）	恶意外部输入（SPI flash）	检查并强化掩模 ROM 代码

初始程序加载完整性（掩码 ROM）	恶意外部输入（硬件配置引脚状态）	根据准确的元件数据表进行原理图审查
初始程序加载完整性（掩码 ROM）	恶意外部输入 (JTAG)	电路板逻辑以及其他操作和软件环境更改，以在生产环境中禁用JTAG

(参见 “OCP 常见安全威胁”，OCP Security Wiki[^2])上正在进行的工作

捕获安全异常

我们行业中产品（或多个）时间表或可用资源的实际情况通常会导致设计中的权衡。因此，在下一个项目进行之前，通常会看到在安全方面妥协并忘记更大目标的生产周期。危险迹象也可能出现在周期的不同阶段，产品团队似乎对安全威胁已经习以为常或麻木不仁，这可能与即使是轻微的员工流失一样令人担忧，因为后者需要重新发现安全威胁，而且通常在计划的后期改变代价高昂，妥协的可能性更大。

为了打破这个循环，建议建立一个正式的“安全例外”制度。安全异常清楚地记录了特定实现与所需实现之间的差距。在某些情况下，安全异常会识别出在将产品投入生产环境之前必须纠正的缺陷。其他人可能有可以实施的缓解措施。但是，请务必捕获差距作为输入以帮助未来的产品开发。

签名的意义

本节提供了一些关于签名固件和相关工件的意义和价值的背景知识。它不涉及签名操作的细节或提供清单方法。签名算法的选择、密钥强度以及在各种设备上启动时签名检查的细节不在讨论范围之内。

在撰写本文时，开放计算项目的安全工作组正在进行一项补充工作。在他们的论文中，他们提到了超出范围的项目，其中一些在此处解决：

OpenCompute 项目贡献，IBM 白皮书，“固件代码签名的最佳实践”[^3]

> “超出本文范围的是信任链要求、固件开发实践、供应链和制造过程，以及证书的使用、证书颁发机构和受信任的应用程序商店。同样超出本文范围但需要注意的是，建立对审计日志的信任。审计日志是否受到保护以防篡改？审计日志是否可以从一开始就被伪造？是否有自动程序来检测与签署服务器审计日志相关的异常活动？

人们需要了解特定签名的含义。在信任范围最松散的一端，签名仅用于满足策略要求（例如，系统不会在没有签名的主启动映像的情况下启动）并且仅用作使用前的完整性检查。在退化的情况下，签名密钥是众所周知的，签名与等效强度哈希算法（例如 SHA256）具有相同的含义，确保位被正确读取，但没有说明它们是可信的。

签名应该是对人们在可信赖软件中寻找的所有品质的证明。也就是说，签名与以下证据相关联：

- 访问与签名相关的秘密和签名的能力受到严格控制和充分理解。
- 从输入到构建过程到输出的映射必须是确定性和可重现的。
- 构建和签名过程是可审计的；本身已知来源的环境、工具链和其他输入，以及用于执行构建的人员或角色帐户都被记录为该过程的一部分。
- 应用签名的人员和流程了解已签名工件的来源。
- 当需要该信息时，签名的工件很容易映射回它们的出处。

审核签约活动

签名日志应该根据已知的构建和发布进行审计。任何使用与已知版本不对应的有价值密钥签名的工件都可能导致轮换密钥和其他操作。

可重现的构建

可重现的构建表明供应商对构建过程有足够的控制权，并且了解该过程的输入是什么。通常需要一个允许通过唯一标识符引用和检索源代码主体的源代码控制系统，例如Git tags[^4]。快照或足够灵活的变更管理系统可用于跟踪工具链、二进制 blob（如预编译的第 3 方库或微码）以及构建环境的其他元素。请注意，二进制 blob 需要显着的信心飞跃并且不鼓励。

可重现的构建通过向开发人员、测试人员和其他人保证他们正在处理的构建产品实际上源自指定的源代码来提高质量。当需要重现和检测难以发现的错误时，此属性非常有价值，因为开发人员可以相信他们所做的检测更改是其测试构建中的唯一变量。例如，开发人员应该能够将构建一分为二（请参阅https://git-scm.com/docs/git-bisect）以隔离引入错误的更改。

签名的不同目的

在构建产品（固件、配置、发行说明等）的生命周期中有几个点需要建立或更新信任。通常，单个签名操作涵盖所有情况，但有时将它们分开是有用的。一个例子可能是一个超大规模的云服务提供商，它为系统设置密钥以仅尊重所有者/运营商自己的签名。固件来自使用该制造商密钥签名的制造商。在该固件可用于所有者/运营商的数据中心之前，制造商的签名必须替换为所有者/运营商的签名。

签名的有用时间，以及该签名的含义：

- **在固件开发期间构建工件** - 所有构建和签名过程都会定期执行，但开发团队可以访问密钥。最终用户系统不接受所使用的密钥。签名绑定输入和工件。

- **构建用于发布的工件** - 所生成产品的出处和测试/验证可以得到证明，并被认为适合在开发团队之外发布。
  除了包含启动时间、安装时间或其他签名的任何工件之外，还需要对整个工件集合或描述这些工件的清单进行签名。作为威胁的例子；攻击者能否插入 README 文件，指示所有者/操作员以不安全的方式进行安装或替换最终破坏系统的脚本？

- **交付** - 所有者/运营商将通过某种机制（ftp 服务器、网络服务器、电子邮件、从检索到的源和指定构建环境的完整复制）接受交付，并且必须能够检查供应商在发布和/或发布的每个元素上的签名作为一个整体。

- **部署** - 所有者/运营商，尤其是在任何大型数据中心，将拥有用于部署更新固件、配置、FPGA 比特流等的自动化程序。这些程序可能有自己的签名流程或参考供应商的签名。通常情况下，所有者/运营商不希望部署来自供应商的更新，而无需经过他们自己的质量控制流程。例如，新固件可能与系统的其他部分或数据中心操作有负面交互，这些交互在生产中测试之前不会显现出来。单独的安装验证机制或等效机制可以用来禁止尚未批准的更新。

- **诊断** - 在支持部署的设备和诊断现场问题时，如果无法在开发硬件上重现问题，则可能需要带有自定义仪器的固件。在签署此类特权构建时，应注意防止它们在比预期更广泛的系统上使用，以防这些构建被泄露。这可以通过将构建绑定到正在调试的特定硬件实例来实现（例如，添加检查以确保将预期的序列号烧入 CPU）。

- **监控** - 所有者/运营商想知道已应用更新并且以前的版本已从数据中心删除。可以加密证明其当前固件和配置的设备提高了信任级别。

安全启动和安全固件更新

SecureBoot 和 Secure Updates 从平台安全的角度来看很重要，因为它们为设备上固件的完整性提供了高度保证。但是，这些超出了编写安全固件的范围。它们受 OCP-Security 保护。

这通常通过多种机制实现：

- 一种安全引导机制，可在引导之前验证预期映像的签名。
- 一种用于固件更新的安全机制，包括诸如具有在生效之前验证每个更新的更新策略、完全清除旧固件以支持新更新的能力以及防止攻击者恢复到已知-的回滚预防机制等功能。坏固件。
- 一种用于安全维护和证明设备内部信任链的机制，包括在设备上实施固件测量。测量应包括固件的加密安全散列和非唯一配置数据（设备唯一配置值，如序列号可以，在某些情况下应该被排除在外）。

所有这些都在 OCP 安全工作组的章程内，并在那里涵盖。CSIS 希望避免重复并支持这项工作。OCP-Security 文档可以在这里找到： https: //www.opencompute.org/wiki/Security

固件开发最佳实践

本节概述了软件开发最佳实践的子集，重点关注与固件开发相关的问题。

设计

从设计阶段开始，将安全性集成到固件启动的最早阶段时是最好的。在编写一行代码之前，“捕获”将导致安全问题的设计问题并在设计过程中更正它们更便宜、更容易。

- ☐ 应在设计时和进行更改时对所有组件（作为整个系统）执行威胁建模，以了解预期的信任边界以及如何减轻潜在的漏洞利用。
- ☐ 应记录信任边界假设。
- ☐ 最小特权，去特权化。
  - ☐ 锁定关键 SPI 区域以防止更新，直到下次重新启动。
  - ☐ 阻止访问固件后续阶段不需要的密钥材料。
  - ☐ 将 SMM / SMI 处理程序的使用限制在最低限度。
- ☐ 限制预签名验证攻击面。
  - [ ]避免在签名验证之前对图像或元数据进行复杂的解析（例如，在签名之前加密/压缩，而不是在加密/压缩操作之前签名）
  - ☐ 在成功验证（启用时）之前，不要公开特权调试、制造或诊断功能。
- ☐ 充分了解硬件设计以正确编程和配置安全启动顺序。
- ☐ 为安全相关事件和包括常见故障（例如硬件初始化失败）实施事件日志。

输入验证

“输入”用于描述不受固件本身控制的任何实体直接或间接提供的任何命令或数据。这包括供应商驱动程序和实用程序提供给设备的所有命令和数据，这些可以被修改或替换，以及来自其他硬件或固件系统组件的交易，这些组件可能是恶意的或欺骗的，包括 SPI 闪存和其他持久性存储器和文件系统。这是对最终用户直接发起的请求的补充。

- ☐ 外部输入，包括配置数据，在使用前必须始终经过清理或验证。
- ☐ 规范化字符串和字符。
- ☐ 过滤或转义可能允许目录遍历攻击等的特殊字符。
- ☐ 验证更新和闪存中的元数据，以确保格式错误的元数据不能用作攻击媒介。

内存安全

可利用漏洞的最常见来源可能是允许攻击者破坏内存并使用它来执行任意代码的错误。遵循一些基本规则可以显着降低此类事件发生的可能性。

- [ ] 使用内存安全实践。
  - [ ] 初始化所有变量。
  - [ ] 始终 边界检查[^5] 缓冲区和数组。
  - [ ] 检查整数溢出和下溢。
  - [ ] 使用安全的字符串和缓冲函数。
  - [ ] 内存不应该既可写又可执行 (W^X[^6])。
- [ ] 考虑使用内存安全语言。
  - [ ] 例如：Rust，或用于嵌入式系统的更安全的用户空间运行时，例如 https://github.com/u-root/u-root。
- []启用适用的内存损坏漏洞利用缓解措施，例如：
  - [ ] ASLR - 地址空间布局随机化[^7]。
  - [ ] 堆栈保护[^8]。
  - [ ] MRR - 内存范围寄存器。
  - [ ] 考虑在不同的内存区域（如堆栈、堆和代码）之间使用故障诱导保护页。
  - [ ] 考虑使用控制流完整性[^9] 保护（例如：[shadow stack](https://github.com/tianocore/edk2/commit/0aac2f777a688a146050bed47753e2dcf801d3c7))

并发

现代设备包含多个处理核心，或者采用对称多处理能力，或者采用分布式非对称模型，最常见的是两者兼而有之。并非所有处理核心都将在相同的安全上下文中运行固件。在与任何共享资源（如共享内存、共享硬件寄存器或共享外围设备和协处理器）交互的此类系统上运行的固件必须注意防止各种竞争条件。这可能发生在低特权子系统可以与共享资源交互而高特权子系统正在使用它的情况下。一些最佳实践包括：

- [ ] 在可能的情况下利用硬件互锁来防止对共享资源的多个并发访问。
- [ ] 在验证和使用外部提供的数据之前制作本地副本，以避免使用时间检查时间 (TOCTOU) 问题。也可能需要任何间接数据结构的深拷贝。

源代码管理

强大的源代码控制系统对于实现可重现的构建和更改审核至关重要。

可接受的源代码控制系统将包括以下功能或启用以下活动：

- [ ] 维护提交历史，包括提交者的身份和提交的意图。（例如，通过要求提交签名[^10])
- [ ] 政策执行，包括：
  - [ ] 签入前的代码审查。
  - [ ] 触发自动化测试挂钩并生成测试报告。
  - [ ] 考虑 CI/CD - 持续集成/持续部署。
- [ ] 与问题跟踪集成。
  - [ ] 错误数据库应该将错误链接到相关的源代码更改。
- [ ] 能够复制任何面向外部的构建，目的是发布有针对性的安全修复程序。

安全代码审查

定期对固件源代码进行安全审查并由安全团队和/或外部各方进行检查是固件开发的重要组成部分。

- ☐ 至少在以下事件中必须定期检查固件是否存在安全问题：
  - ☐ 发布前。
  - ☐ 更新前。
  - ☐ 发现问题时追溯以前的版本。（例如变异狩猎）
- ☐ 考虑偶尔邀请其他部门、客户或其他第三方的员工进行代码审查，以获得全新的视角，并迫使开发团队重新审视代码开发期间所做的假设和决策。

第三方库

作为正在开发的固件的一部分，通常需要使用第三方库（包括开源库）。并且可能成为安全问题的根源，因此跟踪这些问题并在使用它们时遵循一些基本规则非常重要。

- ☐ 审查第 3 方提供商的安全开发实践，以确保它们达到或超过本文档中概述的安全实践。
- ☐ 记录所有第 3 方代码的使用。至少包括对库和版本的引用，并以最佳实践记录所使用的特定功能，尤其是对于较大的库。
- ☐ 尽可能避免使用源代码不可用的预编译二进制文件。当必须使用此类二进制文件时，记录它们并通过跟踪这些二进制文件的来源并验证其包含的哈希来验证其可信度。
- ☐ 在可能的情况下，实施特权分离或隔离第 3 方二进制组件以限制它们与软件其余部分的交互（例如内核强制的特权分离等）
- ☐ 维护包含的库和相关版本的更新列表，以帮助跟踪漏洞、安全更新和适当的及时缓解。
- ☐ 当发现问题并源自第三方来源时，需要跟进以了解影响和补丁制作。

测试

安全开发的一个重要部分包括测试安全问题。以下做法将有助于在发布固件之前发现安全问题。

- ☐ 运行静态分析[^11] 工具来识别代码中的问题。
- ☐ 执行代码的模糊测试[^12]。
- ☐ 衡量测试套件的代码覆盖率[^13]，并以可接受的水平为目标。
  - ☐ 例如关于可接受指标的讨论 此处[^14]。
- ☐ 考虑对安全关键代码部分使用形式化方法（更多详细信息此处[^15]）
- ☐ 自动安全测试作为签入的一部分
- ☐ 测试威胁模型识别的所有威胁向量并包括外部依赖项。

捕获安全异常

上下文在上一节中提供。以下是建议：

- ☐ 记录所有安全异常并在下一个版本开发周期中考虑它们。
- ☐ 制定流程来审核跨版本的安全进度。

构建与编译

自然地，实际在生产中运行的代码是编译后的代码。因此，确保构建的完整性并优化编译以确保安全性非常重要。

- ☐ 可重现构建（参见 签名的不同目的 部分的解释）。
  - ☐ 在安全可靠的环境中运行构建过程。

    - ☐ 记录所有构建依赖项，包括编译器工具链、库和构建脚本。
    - ☐ 确保二进制文件可以由多方相同地构建（参见 https://reproducible-builds.org/docs/ 和 https://wiki.debian.org/ReproducibleBuilds）。
- ☐ 编译器设置。
    - ☐ 如果不是最大警告级别，则应设置为高，警告被视为错误。一旦这些设置到位，应根据需要更正错误。（例如，如果使用 Microsoft Visual Studio，/W4 /WX 是适合使用的编译器开关）。
    - ☐ 与固件构建相关的编译器和汇编器选项应该在调试文件中启用足够强大的源代码哈希排放。（例如，如果使用 Microsoft Visual Studio 编译器和汇编器，请使用 /ZH:SHA_256）。
- ☐ 归档官方版本的构建工件。
    - ☐ 必须保留与公开可用的固件版本相对应的所有固件版本工件。这应该包括调试符号文件和映射文件，以便能够调查在现场发现的问题、更改列表、构建日志文件等。

调试挂钩

调试挂钩是必要的，一些调试机制的用例在维护系统信任方面可能变得非常复杂。没有涵盖所有用例的最佳实践。

一些可能适用的设计选择包括：

- ☐ 在生产版本中禁用调试挂钩。
- ☐ 确保调试挂钩不会以违反设备特定安全性的方式重新打开。（例如，不要创建容易受到潜在攻击的可利用的高权限“调试模式”。）
- ☐ 需要设备标识符和随机数由安全的、记录的、授权服务签名以启用调试功能。如果设备被重置，随机数将变得无效。
- ☐ 要求设备的信任链作为启用调试功能的一部分而失效。这可能意味着设备无法从那时起参与正常的生产活动，这可能还包括清除敏感状态和密钥。

源代码访问

授予固件用户访问源代码的权限有助于提高该代码的安全级别，并增加对固件安全状况的信任。

- ☐ 尽可能提供源代码以供第 3 方审查。这有助于提高代码的安全性。
- ☐ 可以通过开源或通过有针对性地向主要客户披露源代码来提供代码。特别是对于云提供商，不共享源代码通常意味着云提供商不会信任固件。
- ☐ 在需要代码机密性（IP，知识产权）的情况下，考虑共享固件的关键部分，例如与安全启动、固件更新和加密实现相关的代码。
- ☐ 如果无法提供代码，则考虑由可信任的第三方进行审核，以确保固件的信任级别。
- ☐ 为确保在需要为产品的整个生命周期开发安全补丁时源代码可用，请考虑将源代码置于托管中以对冲业务连续性风险。

验证/合规

为了能够证明固件开发遵循了正确的安全最佳实践，通常需要为公开发布的固件构建提供以下工件的证据：

- ☐ 独立（非固件开发团队）安全审查/渗透测试。
- ☐ 测试和评论报告。
- ☐ 构建日志。
- ☐ 测试日志。
- ☐ 修订历史/更改日志。

安全配置

许多固件实现支持运行时配置。配置设置可能会更改固件环境的安全参数、禁用硬件和软件安全功能、启用调试功能和已知不太安全的旧功能等。因此，配置对于固件安全与固件本身一样重要.

安全配置的最佳实践包括：

- ☐ 在使用基于硬件的配置的情况下，首选支持熔丝/一次性可编程 (OTP) 配置的组件，因为带有带电阻器的外部配置引脚更容易受到攻击/修改。
    - ☐ 当使用保险丝存储配置数据时，确保保险丝/OTP 功能允许更改可能需要在设备生命周期内更新的设置（例如：设备所有权、故障分析/调试状态）
- ☐ 对于基于闪存的配置，配置应在使用前进行签名和验证。还考虑包括设备身份以防止配置被复制和在其他设备中重复使用（即：应避免使用未签名的标志）
- ☐ 默认配置状态应该是安全的，只要可能（即：不包括制造初始配置）
- ☐ 避免故障打开语义，例如使用特定值指示更安全状态的位置[^16]，或空白设置指示"不安全"的位置[^17]
- ☐ 实施多重配置检查（例如，在每次使用时，并定期检查）以确保配置值不会在运行时更改，例如，通过故障注入攻击

启用遗留标准的弃用

许多遗留标准在设计时并未考虑到安全性。这些标准通常允许未经身份验证的固件加载。更新通常包含已知漏洞 (CVE) 的历史记录，通常会对平台构成风险。

但是，这些标准通常还没有准备好完全弃用，并且仍在生产中使用。随着时间的推移，行业将转向更安全的基础，需要进行过渡。

从云提供商的角度来看，能够完全禁用非常重要，理想情况下甚至不包括遗留标准的代码。如果必须包括支持，这些系统在禁用时应该拒绝对这些子系统的请求，并终止处理这些输入的进程。下面列出了这些已知的有风险的。

传统 BIOS 引导

传统 BIOS 不支持安全启动，这意味着固件在加载期间未经过身份验证。

以下要求解决了这种威胁：

- ☐ 支持具有安全启动支持的新启动机制（例如 UEFI SecureBoot、Linux Boot、Verified Uboot）
- ☐ 对所有固件进行签名，以便验证其真实性。
- ☐ 允许所有者/操作员禁用对传统启动的支持。

IPMI

负责 IPMI 标准的小组要求[^18] 业界找到合适的替代品并弃用 IPMI。

> "没有计划或预期对 IPMI 规范进行进一步更新。IPMI 发起人鼓励设备供应商和 IT 经理考虑更现代的系统管理界面，它可以为现有数据中心提供更好的安全性、可扩展性和功能，并在必要的平台和设备。DMTF 的 Redfish 标准（来自 dmtf.org/redfish）就是这样一个接口的例子。"

IPMI 具有已知的可利用漏洞历史记录（请参阅 Intel IPMI CVE 报告[^19]，http:// fish2.com/ipmi/），以及依赖于基于共享秘密的身份验证，如果泄露，系统就会暴露在外。

建议采取以下缓解措施：

- ☐ 使客户能够删除/禁用 IPMI。
- ☐ 对于必须支持 IPMI 的系统，在顶部启用额外的身份验证层（例如 SSH）
- ☐ 始终使用最新的 IPMI 实现和最新的安全修复

不安全的网络协议

在许多情况下，固件必须支持通过网络启动或更新等功能。这使其面临风险，例如未经验证的图像下载和执行、加载未经验证的选项 ROM 以访问网络，或来自同一第二层网络（DNS、DHCP、TFTP）上的恶意服务器的拒绝服务。

建议采取以下缓解措施：

- ☐ 当需要网络引导或基于网络的更新时，实施最新的安全协议（例如使用最新的 TLS 版本，在 PXE 引导中使用 HTTPS 而不是 TFTP）
- ☐ 考虑支持替代的本地引导路径（例如 BMC，如果值得信赖，提供安装到主机本地引导块存储设备的方法，而不是依赖网络引导。）
- ☐ 支持禁用任何此类基于网络的功能，以防所有者/运营商不需要。

支持工具

为支持相关组件、固件或平台而开发的工具、实用程序和驱动程序也应符合安全最佳实践。它们的开发应该保持安全性和类似的质量一致性。

软件开发最佳实践

- ☐ 固件更新和诊断实用程序应遵循安全开发生命周期 (SDL) 最佳实践，因为它们通常最终用于生产环境。
    - ☐ 对于 Windows，请查看 Microsoft Driver Security Checklist[^20]
- ☐ 应尽可能遵守最小权限原则[^21]（例如映射 IO 资源后删除 root）
- ☐ 应该只访问与目标设备关联的内存或 IO 资源，而不是任意内存或 IO 资源（过度访问会导致 [本地权限提升漏洞]（https://eclypsium.com/2019/08/10/screwed -驱动程序签名密封交付/)[^22])。
- ☐ 用于生产环境的实用程序
    - ☐ 应该在处理之前检查来自设备的响应是否格式正确。表现出错误行为的设备不应通过对命令提供意外或不合规格的响应而导致实用程序崩溃或失败。
    - ☐ 应该能够离线或在隔离环境中运行。
    - ☐ 以二进制形式提供的实用程序不得要求禁用内核保护以防止运行未签名的代码。这意味着必须对设备驱动程序进行签名。
        - ☐ Windows 实用程序应该经过 WHQL 认证。
        - ☐ EFI 实用程序应由 e UEFI 证书颁发机构签名。
        - ☐ 对于 Linux，请遵循 [内核模块签名指南](https://www.google.com/url?q=https://www.kernel.org/doc/html/v5.0/admin-guide/ module-

signing.html&sa=D&ust=1556214662946000&usg=AFQjCNHTrVj2pyDaMI7odz1O9_LzZ2CHrw)[^23]

    - [ ] 遵循实用程序包的发行版包管理器指南中概述的包签名程序。
    - [ ] 如果第 3 方分发，请提供加密包验证的替代方法，例如 GNU Privacy Guard 分离签名。

- [ ] 调试、诊断、制造和测试实用程序
    - [ ] 不应包含任何启用额外调试功能的秘密。
    - [ ] 应要求获得特权调试功能的授权：
        - [ ] 必须绑定到设备的特定实例。
        - [ ] 应该是可撤销的。
        - [ ] 应产生审计跟踪。

文档

- [ ] 实施行业指定固件更新命令（例如：NVMe、ATA 或 SCSI）的实用程序应记录任何偏差或应用固件更新所需的其他操作。
- [ ] 建议记录设备驱动程序访问的所有内存和 IO 资源。
- [ ] 应正确记录所有实用程序子命令和选项。
- [ ] 非标准接口协议和扩展应该被充分记录以允许进行安全测试
- [ ] 建议提供实用程序源代码，包括：
    - [ ] 调试符号
    - [ ] 代码头 -[ ]部分源代码总比没有好

环境特定要求

- [ ] UEFI
    - [ ] 如果安装在 FS0 以外的位置，工具必须正常运行：\（UEFI 映射表中列出的第一个文件系统）。
- [ ] 操作系统
    - [ ] 未来平台将不支持传统模式启动，使基于 DOS 的工具无法使用。固件更新和诊断工具不能依赖于 DOS 环境。
- [ ] Linux
    - [ ] 非 EOL 内核版本遵循您的产品预期支持时间表。
    - [ ] 驱动模块支持，无论是其后的初始化。空间限制或安全性也可能是仅支持内核编译驱动程序的一个因素。
    - [ ] 在支持的内核驱动程序接口中添加适当的访问控制策略（例如：没有世界可访问的文件、SELinux 策略等）
    - [ ] 内核驱动程序必须使用所有适当的 uaccess[^24] 方法来移入和移出数据内核，在需要的地方制作本地副本以避免竞争条件。
    - [ ] 内核驱动程序的文件系统接口必须暴露最小的攻击面。审核所有暴露的接口（/proc、/sys、/debugfs、/dev 等）

    - ☐ 实现内核驱动程序时，ioctl[^25] 接口容易出错，应该避免（*file_operations* read/write 是首选）。
    - ☐ 在 debugfs[^26] 中公开的接口应该在生产构建中移除。
    - ☐ 如果驱动程序不在上游 Linux 中，请包括源代码、内核头文件和安全示例以供使用。

发布后流程

产品发布只是产品生命周期中的一个里程碑，从内部交付到让实际客户使用您的产品。从安全角度来看，您有责任继续寻找漏洞，并负责管理一个流程，即使在发现新漏洞时也能让客户保持安全。ISO/SEC 标准 29147 和 30111 为如何管理这些流程提供了一套详细而详尽的要求。以下清单应该可以帮助您朝着正确的方向开始。

主动寻找漏洞和利用

- ☐ 根据测试部分中描述的做法，定期查找任何新版本/补丁中的问题
- ☐ 为客户、安全研究人员或其他此类外部来源建立安全流程，以报告他们发现的漏洞；该流程必须提供一种报告问题的方式，而无需强制参与错误赏金计划或以其他方式对披露条款设置任何先决条件。确保此过程易于发现（例如 security.txt[^27]）。
- ☐ 考虑提供漏洞赏金计划[^28] 以进一步鼓励安全研究人员和用户社区帮助查找和修复安全问题。这包括发布成功发现、评估和报告风险所需的足够先决条件和信息。

问题分类和风险评估

- ☐ 报告问题时，实施调查和严重性分类并发现所需的修复。
- ☐ 能够为特定的发行版本发布有针对性的补丁。
- ☐ 最好使用一种公开可用的框架，根据攻击媒介、利用的难易程度、可能的影响等因素，对发现的问题的威胁级别进行分类。
    - 例如 CVSS[^29] 可以帮助计算每个威胁的风险级别。
    - 例如 STRIDE[^30] 和 DREAD[^31] 模型可以帮助评估威胁和风险级别。
- ☐ 假设紧迫性，直到证明并非如此。

漏洞披露和补丁可用性

- ☐ 遵循披露者同意的负责任的披露[^32] 做法和时间表，并为补丁发布提供明确定义的 SLA。
- ☐ 当问题被确定为可利用时——必须及时通知客户，并给予足够的时间自行准备/修补。
- ☐ 一旦您意识到一个活动事件（即已知“在野”被利用的问题）——必须立即通知使用您产品的客户。
- ☐ 优先向云提供商尽早披露是必不可少的，因为他们经常因高价值而成为目标。
- ☐ 当安全问题（由您或其他人）公开时，确保报告/打开 CVE (https://cve.mitre.org/)。这有助于您无法通知的客户跟踪适用于他们的事件。
- ☐ 确保披露与它们适用的特定发布版本相关联，并且专门发布修复程序来解决漏洞。尽可能避免将安全修复程序与其他功能和增强功能捆绑在一起。
- ☐ 通过行业标准机制和打包使固件更新可发现，并考虑集成到 LVFS 或 Windows 更新等分发机制中。

时间表和 SLA

- ☐ 建立承诺的时间表，以便在发现漏洞和发生活动事件时进行披露。这个时间表应该给客户足够的时间来准备/打补丁。确切的时间表可能会有所不同，但请考虑几小时和几天，而不是几周和几个月（例如 https://cyber.dhs.gov/bod/19-02/ 和 https://www.cyberessentials.ncsc.gov.uk/requirements-基础设施）

组件特定要求

大多数硬件平台都包含许多硬件设备或组件，需要与之交互以提供额外的硬件功能。该接口是通过组件固件启用的，固件可以提供对硬件功能的直接访问或加载其他接口以访问其他功能的方法。本节概述了这些接口和组件的详细信息。

UEFI

统一可扩展固件接口是通过 BIOS 或引导 ROM 访问的引导接口，它提供了访问较低级别设备及其相关接口的标准。UEFI 提供了大量的底层硬件访问，如果开发得当，可以确保固件完整性。这方面的指导方针包括：

- [ ] 系统必须通过配置安全检查，例如：
  - CHIPSEC[^33] 在 Intel x86 平台上
  - 相关 FWTS 测试[^34]
- [ ] 并包括对以下方面的支持：
  - [ ] 安全启动（或等效）
    - 硬件 SRTM 或 DRTM
    - DBx（已知漏洞的关键黑名单）
  - [ ] BIOSGuard（或等同物）
  - [ ] 不可绕过的认证更新机制 (NIST 800-147)[^35]
  - [ ] Option-ROM 禁用功能。
  - [ ] PKI 创建和管理。
  - [ ] 通过视觉通知禁用安全启动。
  - [ ] 具有特权角色和安全访问的凭证授权。

###核心启动

coreboot[^36] 是一个扩展的固件平台，可在多种硬件架构（x86、RISC-V、ARM、PPC 等）上提供快速安全的启动体验。coreboot 实现了芯片组支持的大多数安全功能，但默认情况下不启用它们。Verified boot 和 measured boot 需要通过 payload 来实现，eg: heads。

支持 coreboot 的平台应遵循以下准则：

- [ ] 系统必须通过配置安全检查，例如：
  - [ ] 对于 Intel x86 平台，CHIPSEC 推荐的安全设置和功能[^37]
  - [ ] 相关 FWTS 测试
- [ ] 实测启动支持：
  - [ ] Heads[^38] 作为支持 TPMv1.2 的系统的核心引导负载实现
  - [ ] 基于 Vboot 的实现[^39] 适用于支持 TPMv2 的系统
- [ ] DRTM（x86 平台上的英特尔 TXT）
  - [ ] 内存清除[^40]
  - [ ] 测量[^41] 在 BIOS/SINIT ACM 中

选项-ROM

选项 ROM 包含支持预操作系统功能所需的固件级驱动程序。它们由外围卡提供或集成到主板的引导闪存中。其中包括 PXE 启动所需的网络驱动程序、视频显示驱动程序和存储驱动程序。\

不建议使用旧版 BIOS 选项 ROM。

- ☐ UEFI DB 信任的签名。（参见 UEFI 规范 v2.8[^42]，第 32.4.1 节）
- ☐ 可以通过凭据禁用、锁定或设置为只读作为可配置的安全措施。
  - 例如，请参阅 Microsoft 的“UEFI 验证选项 ROM 指南[^43]”。请记住，只有最严格的“安全启动”配置才可能有助于缓解供应链攻击。 如果不需要重建 UEFI 固件映像，可以使用“Fiano”[^44] 等工具注入其他已知良好的选项 ROM。如果嵌入选项 ROM 以提高安全性，则还必须重新签署 UEFI 映像以实现安全引导。

BMC

底板管理控制器 (BMC) 为服务器提供带外管理服务。

BMC 在安全方面的历史不佳，因为它们最初的威胁模型严重依赖于隔离的管理网络和行为良好的系统管理员。该模型是 BMC 在可能的范围内提供等同于系统管理员实际存在的访问权限。

随着裸机托管等服务的出现，有必要支持一种威胁模型，其中主机操作系统可能是恶意的，BMC 为服务提供商提供唯一可扩展的方式来将主机恢复到可信赖的状态。例如，通过 BMC 进行的管理必须能够在将主机移交给下一个客户之前强制执行主机的引导环境并随后恢复良好的已知状态。BMC 隐式信任主机操作系统的模型与裸机托管服务不兼容。

几乎所有 BMC 都运行具有固定软件堆栈的 Linux 的某些变体，即 BMC 仅运行其固件映像中包含的代码。下面的大纲提供了有关关键兴趣点的一般指导。

- ☐ BMC 应该运行支持良好的操作系统版本（例如 长期支持版本[^45]）
- ☐ 使用强化的内核设置（例如：KSPP 指南[^46]）
- ☐ 记录所有可用接口：
  - ☐ IPMI[^47] 界面（请注意本节中的建议），即不要使用）
  - ☐ SMASH CLP[^48] 通过 ssh
  - ☐ Redfish[^49]（[安全详细信息](http://redfish.dmtf.org/schemas/DSP0266_1.7.0.html#security - 细节-a-id-安全-细节-a-)[^50]）
  - ☐ 基于 SSL (443) 的 Web 界面，具有从 80 禁用或重新路由的能力
- ☐ 提供禁用不需要的接口的能力：
  - ☐ 远程登录
  - ☐ SSH
  - ☐ 不安全的网络 (80)
  - ☐ 用于连接和分离主机媒体的媒体管理服务（特定于平台）
  - ☐ IPMI
- ☐ 提供禁用或限制服务功能的能力：
  - ☐ 更喜欢基于角色的访问限制。
  - ☐ 首选通过 Active Directory、Open Directory、LDAP、EDir、Radius 等进行远程身份验证。

外设固件

不直接由主机处理器执行的固件（例如嵌入在外围硬件中并负责其初始化的固件），以及由外围设备提供给主机执行的固件(例如[上一节]中提到的Option ROM）(#option-roms)) 还必须遵守最佳实践以确保系统安全。主

机操作系统驱动程序开发通常需要此固件和相关的主机运行命令。一般指导如下：

- [ ] 外设固件应遵循 SecureBoot 签名验证流程，就像平台固件一样
- [ ] 实施外围固件证明，如 OCP-Security 所定义，为主机平台提供外围固件完整性状态的可见性。
- [ ] 应用于外围固件的安全启动策略可能包括：
  - [ ] 不要将控制转移到没有通过强签名检查的代码，从而"阻塞系统/子系统"，*or*
  - [ ] 放弃特权或对密钥的访问，并在将控制权转移到未通过签名检查的代码之前记录异常。这允许无人值守的恢复过程尝试系统恢复，然后重新启动到受信任的状态。或者，
  - [ ] 设计具有主机或 BMC 单方面访问权限的硬件，以审核和编程外围设备。这将问题推到信任链的较早阶段，但如果需要外围设备作为引导设备，则可能会使系统操作复杂化。
- [ ] 考虑威胁模型中存在恶意 PCIe 设备的可能性。
  - [ ] 是否需要配置 IOMMU？何时以及如何？
- [ ] 必须列出所有可用的命令，包括不支持的命令。支持的命令必须完整记录。
- [ ] 需要记录所有用于访问外围固件功能的可用接口。
- [ ] 如果核心功能不需要，则提供禁用单个接口的能力。

参考

- CII 徽章 , https://github.com/coreinfrastructure/best-practices-badge
- NIST 800-193, https://csrc.nist.gov/publications/detail/sp/800-193 /草稿
- CSIS 立场文件对 800-193 的回应，https://www.cloudsecurityindustrysummit.org /document/firmware-integrity-in-the-cloud-data-center.pdf
- SAFECode , https://safecode.org/publications/
- 引导固件代码审查指南, https://edk2-docs.gitbooks.io/ edk-ii-secure-code-review-guide/code_review_guidelines_for_boot_firmware/
- ISO 27000 , http://www.iso27001security.com/html/iso27000.html
- OWASP 安全软件开发生命周期项目, https://www.owasp.org/index.php/OWASP_Secure_Software_Development_Lifecycle_Project
- IOMMU 针对 I/O 攻击的保护：漏洞和概念证明，https://link.springer。 com/article/10.1186/s13173-017-0066-7
- ISO/SEC 29147，https://www.iso。 org/obp/ui/#iso:std:iso-iec:29147:ed-2:v1:en
- ISO/SEC 30111, https://www.iso.org/standard/53231.html
- 固件代码签名的最佳实践 https://www.opencompute.org/documents/ibm-white-paper-best-practices-for-firmware-code-signing

修订记录

1 - 2019-06-18

CSIS SCWG 的初始版本

1.1 - 2019-10-27

添加了社区反馈，包括以下内容：

- 与签名前验证检查相关的要求
- 内存安全要求
- 多进程系统中代码并发的要求

- 安全配置要求
- 核心引导支持的组件特定要求
- 改进发布后流程的要求
- 进一步阅读的额外有用参考
- 文档中的许多较小的编辑和改进
- 公认的新社区贡献者。

文档现已发布在 OCP GitHub 上。那里最受欢迎的是额外的贡献。

引用的 URL

[^1]:

> https://cloudsecurityalliance.org/artifacts/firmware-integrity-in-the-cloud-data-center/

[^2]:

> https://www.opencompute.org/wiki/安全

[^3]:

> http://files.opencompute.org/oc/public.php?service=files&t=f4171bae8c7a32f05b0401378ee08483&download

[^4]:

> https://git-scm.com/book/en/v2/Git-Basics-Tagging

[^5]:

> https://en.wikipedia.org/wiki/Bounds_checking

[^6]:

> https://en.wikipedia.org/wiki/W%5EX

[^7]:

> https://en.wikipedia.org/wiki/Address_space_layout_randomization

[^8]:

> https://en.wikipedia.org/wiki/Stack_buffer_overflow#Protection_schemes

[^9]:

> https://en.wikipedia.org/wiki/Control-flow_integrity

[^10]:

> https://help.github.com/en/articles/about-required-commit-signing

[^11]:

> https://www.owasp.org/index.php/Static_Code_Analysis

[^12]:

> https://www.owasp.org/index.php/Fuzzing

[^13]:

> https://en.wikipedia.org/wiki/Code_coverage

[^14]:

> https://en.wikipedia.org/wiki/Code_coverage#In_practice

[^15]:

> https://en.wikipedia.org/wiki/Formal_methods

[^16]:

> https://recon.cx/2017/brussels/resources/slides/RECON-BRX-2017-Breaking_CRP_on_NXP_LPC_Microcontrollers_slides.pdf

[^17]:

> http://faq.riffbox.org/content/3/71/en/how-to-erase-frp-factory-reset-protection-using-riff-box-generic-instrucions.html

[^18]:

> https://www.intel.com/content/www/us/en/servers/ipmi/ipmi-home.html

[^19]:

> https://www.cvedetails.com/product/30635/Intel-Ipmi.html?vendor_id=238

[^20]:

> https://docs.microsoft.com/en-us/windows-hardware/drivers/driversecurity/driver-security-checklist

[^21]:

> https://en.wikipedia.org/wiki/Principle_of_least_privilege

[^22]:

> https://eclypsium.com/2019/08/10/screwed-drivers-signed-sealed-delivered/

[^23]:

> https://www.kernel.org/doc/html/v5.0/admin-guide/module-signing.html

[^24]:

> https://github.com/torvalds/linux/blob/master/include/linux/uaccess.h

[^25]:

> https://lwn.net/Articles/428140/

[^26]:

> https://lwn.net/Articles/429321/

[^27]:

> https://securitytxt.org/

[^28]:

> https://en.wikipedia.org/wiki/Bug_bounty_program

[^29]:

> https://www.first.org/cvss/

[^30]:

> https://en.wikipedia.org/wiki/STRIDE_（安全）

[^31]:

> https://en.wikipedia.org/wiki/DREAD_(risk_assessment_model)

[^32]:

> https://en.wikipedia.org/wiki/Responsible_disclosure

[^33]:

> https://github.com/chipsec/chipsec

[^34]:

> https://wiki.ubuntu.com/FirmwareTestSuite/Reference

[^35]:

> https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-147.pdf

[^36]:

> https://doc.coreboot.org/

[^37]:

> https://github.com/hardenedlinux/Debian-GNU-Linux-Profiles/blob/master/docs/harbian_fw/harbian_chipsec.md

[^38]:

> https://github.com/osresearch/heads

[^39]:

> https://github.com/coreboot/coreboot/blob/master/Documentation/security/vboot/measured_boot.md

[^40]:

> https://github.com/coreboot/coreboot/blob/master/Documentation/security/memory_clearing.md

[^41]:

> https://github.com/coreboot/coreboot/blob/master/Documentation/security/intel/txt.md

[^42]:

> https://uefi.org/sites/default/files/resources/UEFI_Spec_2_8_final.pdf

[^43]:

> https://docs.microsoft.com/en-us/windows-hardware/manufacture/desktop/uefi-validation-option-rom-validation-guidance

[^44]:

> https://github.com/linuxboot/fiano

[^45]:

> https://www.kernel.org/

[^46]:

> https://kernsec.org/wiki/index.php/Kernel_Self_Protection_Project/Recommended_Settings

[^47]:

> https://www.intel.com/content/www/us/en/servers/ipmi/ipmi-technical-resources.html

[^48]:

> https://www.dmtf.org/standards/smash

[^49]:

> https://www.dmtf.org/standards/redfish

[^50]:

> http://redfish.dmtf.org/schemas/DSP0266_1.7.0.html#security-details-a-id-security-details-a-