

电子科技大学

UNIVERSITY OF ELECTRONIC SCIENCE AND TECHNOLOGY OF CHINA

硕士学位论文

MASTER THESIS



论文题目 基于固件文件系统的 UEFI 安全机制研究

学科专业 信息安全

学 号 201321060638

作者姓名 房 强

指导教师 范明钰 教 授

分类号 _____ 密级 _____

UDC ^{注 1} _____

学 位 论 文

基于固件文件系统的 UEFI 安全机制研究

(题名和副题名)

房 强

(作者姓名)

指导教师	范明钰	教 授
	电子科技大学	成 都

(姓名、职称、单位名称)

申请学位级别 硕士 学科专业 信息安全

提交论文日期 2016.3.15 论文答辩日期 2016.5.20

学位授予单位和日期 电子科技大学 2016 年 6 月

答辩委员会主席 _____

评阅人 _____

注 1: 注明《国际十进分类法 UDC》的类号。

A Research on UEFI Security Mechanism Based on Firmware File System

A Master Thesis Submitted to
University of Electronic Science and Technology of China

Major: **Information Security**

Author: **Fang Qiang**

Supervisor: **Prof. Fan Ming Yu**

School: **School of Computer Science and Engineering**

独创性声明

本人声明所呈交的学位论文是本人在导师指导下进行的研究工作及取得的研究成果。据我所知，除了文中特别加以标注和致谢的地方外，论文中不包含其他人已经发表或撰写过的研究成果，也不包含为获得电子科技大学或其它教育机构的学位或证书而使用过的材料。与我一同工作的同志对本研究所做的任何贡献均已在论文中作了明确的说明并表示谢意。

作者签名：_____ 日期：_____ 年 _____ 月 _____ 日

论文使用授权

本学位论文作者完全了解电子科技大学有关保留、使用学位论文的规定，有权保留并向国家有关部门或机构送交论文的复印件和磁盘，允许论文被查阅和借阅。本人授权电子科技大学可以将学位论文的全部或部分内容编入有关数据库进行检索，可以采用影印、缩印或扫描等复制手段保存、汇编学位论文。

（保密的学位论文在解密后应遵守此规定）

作者签名：_____ 导师签名：_____

日期：_____ 年 _____ 月 _____ 日

摘 要

传统的 BIOS 由于存在程序开发及维护困难、不易扩展等缺陷,已严重制约了计算机系统的发展。针对这些弊端,Intel 等公司推出统一可扩展固件接口(Unified Extensible Firmware Interface, UEFI)以替代传统 BIOS。UEFI 固件是计算机启动的第一道程序,其安全性直接关系到整个计算机系统的安全,一旦遭到攻击,就会导致计算机在操作系统加载前就被攻击者控制。随着 UEFI 的普及,已经出现了针对 UEFI 的攻击,尤其是利用固件文件系统进行的攻击具有隐蔽性强、权限高的特点。鉴于 UEFI 潜在的安全威胁不断增多,对其安全机制研究刻不容缓。

UEFI 规范中加入了关于可信启动、数字签名等安全服务的定义,这些定义基于可信计算原理,完成 UEFI 固件的完整性校验和身份认证等功能。但是这些安全机制主要是针对第三方的检测,并不能有效阻止攻击者利用固件文件系统对 UEFI 固件进行攻击。由于 UEFI 相关文档的公开,固件文件系统的细节信息很容易被攻击者得到。攻击者利用这些信息对固件中的文件进行添加或修改,从而绕过固件安全机制对第三方驱动和应用程序的检查,直接对 UEFI 固件中的代码进行破坏和利用。

本文将 UEFI 的启动阶段与可信计算的思想相结合,通过建立 UEFI 启动过程中的信任链和基于固件文件的可信度量机制,提出基于固件文件系统的 UEFI 安全方案,解决了目前攻击者利用固件文件系统对 UEFI 固件进行攻击的问题。

本文依据 UEFI Framework 架构对 UEFI 启动过程阶段的划分,将前两个启动阶段作为可信度量根,其它阶段作为信任链的节点,从而在各阶段之间建立一条信任链。各阶段在进行控制权传递时,必须对下一阶段进行可信度量,若下一阶段可信则传递控制权,并依次将可信度量关系传递下去。UEFI 安全方案采用完整性度量机制作为可信度量方法,通过对固件文件进行完整性度量,来确认启动各阶段的可信性,从而保证整个 UEFI 启动过程的可信性。

根据 UEFI 安全方案,本文设计并实现基于 UEFI 驱动的固件安全模块。该模块使用 UEFI 驱动程序来虚拟 TPM (Trusted Platform Module) 芯片,主要由主模块、Hash 算法引擎、通信单元及存储模块组成。固件安全模块是对安全方案功能的实现,存储在 UEFI 固件中,在 UEFI 启动过程中运行并保护 UEFI 启动。本文最后通过实验对固件安全模块进行测试,验证了固件安全方案的可行性及优缺点。

关键词: 统一可扩展固件接口; 固件文件系统; 可信计算; 信任链; 安全模块

ABSTRACT

Currently, the computer hardware is developing rapidly, but it is difficult for legacy BIOS to meet the demand of modern computer hardware development owing to the uneasiness for programming & maintenance and extension. Aiming to the shortages of legacy BIOS, Intel and other companies have suggested replacing it by UEFI (Unified Extensible Firmware Interface). UEFI firmware is the first access program of the computer and its security will directly affect the security of the whole computer system. If UEFI firmware encounters attack, the computer will be controlled by the attackers before loading in the operation system, and the attacker will gain the root permission of the computer. The UEFI's population has already brought the targeted attacks, especially the attacks of strong imperceptibility and high permission through firmware file system. In consideration of more and more potential security threats to UEFI, the study on security mechanism of UEFI firmware is of great urgency.

As the substitute of legacy BIOS, UEFI's standard was added with the trusted start, digital signature and other definitions for security services which are based on the trusted platform standard to execute the integrity verifying and identity authentication of UEFI firmware platform. However, those security mechanisms mainly aimed to the detection program of the third party but cannot stop attackers from attacking UEFI firmware by making use of firmware file system. Because the related UEFI documents are disclosed, the detailed information of firmware file system is easily accessible to attackers. By making use of the information, attackers can make addition or alternation to the files in the firmware and straightly destroy or take advantages of the code of UEFI firmware by skipping the trusted compute chain's examination to the third party's drive and application program.

The thesis gives the firmware film system based UEFI security idea to combine UEFI start progress and trusted compute thought, or specifically create a trust chain in the UEFI start progress and trusted measurement mechanism of firmware files. The security idea can effectively repel attackers damaging UEFI firmware by using firmware file system.

Based on the division of UEFI start process by UEFI Framework, the first two start phases are regarded as the trusted measurement root and other phases as the nodes of

trust chain to create a trust chain linking all the phases. In the transferring process of control permission, the trusted measurement must be conducted before, and the control permission can be transferred only when the next phase is trusted. The trusted measurement relationship should be transferred one after the other. The integrity measurement mechanism is taken as the trusted measurement method in UEFI security plan, that is to say, the trustworthiness of each phase is determined via the integrity measurement to the firmware files to guarantee the whole UEFI start process with high trust degree.

In accordance with the UEFI security plan, the thesis will design and realize the firmware security module driven by UEFI. The module, composed of main module, Hash algorithm engine, communication cell and memory module, uses UEFI drive program to make virtual TPM (Trusted Platform Module) chip. The security plan function is realized via the firmware security module which is stored in UEFI firmware and can get the execution permission in UEFI start process, and also will protect the start process. In the last part the thesis, the firmware security module will be given with test to verify whether it can protect UEFI firmware files or not, so as to ensure the trustworthiness of UEFI start process.

Key words: UEFI, Firmware File System, trusted computing, trusted chain, security module

目 录

第一章 绪论	1
1.1 课题研究背景及意义	1
1.1.1 研究背景	1
1.1.2 研究意义	2
1.2 国内外研究现状	3
1.2.1 安全威胁研究现状	3
1.2.2 安全机制研究现状	4
1.3 论文主要研究内容	5
1.4 论文组织结构	5
第二章 UEFI 规范及可信计算技术研究	7
2.1 UEFI 基本框架	7
2.1.1 UEFI 系统服务	8
2.1.2 UEFI 驱动模型	9
2.1.2.1 UEFI 驱动模型简介	10
2.1.2.2 UEFI 驱动分类	12
2.1.3 UEFI 映像文件	12
2.2 UEFI Framework 结构	14
2.3 UEFI 整体安全性分析	16
2.3.1 UEFI 平台中存在的安全隐患	16
2.3.2 基于 Framework 的 UEFI 启动阶段安全性分析	17
2.4 可信计算技术	18
2.4.1 可信计算理论简介	18
2.4.2 信任链技术	19
2.4.3 可信度量机制	20
2.5 本章小结	21
第三章 UEFI 固件文件系统研究与分析	22
3.1 固件文件系统结构	22
3.2 固件设备	23
3.3 固件卷	24
3.4 固件文件	25

3.4.1 固件文件类型	25
3.4.2 固件文件格式	27
3.5 区块文件与 EFI 文件	29
3.5.1 区块文件	29
3.5.2 EFI 文件	30
3.5.2.1 EFI 文件格式	30
3.5.2.2 EFI 文件生成	31
3.6 文件遍历和访问	32
3.7 本章小结	32
第四章 基于固件文件系统的 UEFI 安全威胁研究	34
4.1 固件文件系统安全性分析	34
4.2 固件文件系统初始化	35
4.3 基于固件文件系统的攻击方法研究	37
4.3.1 UEFI Bootkit 研究	37
4.3.1.1 Bootkit 工作原理分析	37
4.3.1.2 Bootkit 代码启动方法	39
4.3.2 基于 FFS 文件的攻击方法	40
4.4 本章小结	41
第五章 基于固件文件系统的 UEFI 安全方案研究	43
5.1 基于固件文件的可信计算研究	43
5.2 UEFI 的信任链设计	44
5.2.1 UEFI 信任链的构建基础	44
5.2.2 可信度量根的确立	44
5.2.3 基于 UEFI 启动过程的信任链设计	45
5.3 可信度量方法	46
5.4 UEFI 固件安全方案设计与分析	47
5.4.1 固件安全方案设计	47
5.4.2 固件安全方案分析	50
5.5 本章小结	51
第六章 UEFI 固件安全方案的实现与验证	52
6.1 EDKII 开发环境介绍	52
6.2 固件安全模块设计	53
6.2.1 主模块	53

6.2.2 通信单元	55
6.2.3 Hash 算法引擎	56
6.2.4 存储模块	56
6.3 固件安全模块实现	57
6.4 安全方案验证及结果分析	59
6.4.1 实验环境及工具	59
6.4.2 验证方法	60
6.4.3 验证结果及分析	60
6.4.3.1 整体功能验证及分析	60
6.4.3.2 对启动过程影响验证及分析	64
6.5 本章小结	64
第七章 总结与展望	65
致谢	67
参考文献	68

第一章 绪论

1.1 课题研究背景及意义

1.1.1 研究背景

互联网的高速发展推动了整个世界的政治、经济和文化发展，给人们生活的各方面带来了巨大的便利。但是，伴随着互联网的发展，网络安全威胁也一直存在，并且安全形势日益严峻。固件作为计算机开机后执行的第一道程序，在 PC 机和服务器上广泛使用，其安全性直接影响整个计算机平台的安全。因此，计算机固件的安全问题已经成为当下研究的热点。

基本输入输出系统（Basic Input/Output System, BIOS）是一组固化在计算机主板 ROM 芯片上的程序，主要负责初始化硬件和操作系统预启动处理^[1]。BIOS 自设计应用至今已有 20 多年历史，由于当时的局限性，BIOS 的设计存在着开发及维护困难、扩展性差等不足之处^[2]，已严重制约了整个计算机系统的发展。更重要的是，传统的 BIOS 在设计之初缺少安全方面的考虑，存在诸多安全隐患^[3]。BIOS 作为系统启动的第一个程序，其在整个计算机系统的位置举足轻重，从而导致了一些木马病毒选择 BIOS 固件作为攻击目标^[4]。由于固件处于计算机系统的底层，针对 BIOS 的攻击会在操作系统加载前进行，这导致攻击者将拥有计算机系统的根权限，并且在应用层很难发现和清除。

针对传统 BIOS 的弊端，Intel 于 2003 年对外公布了其制定的新一代固件标准 EFI（Extensible Firmware Interface），用以替代传统 BIOS，并将其运用到了自己的 Itanium 平台中。传统 BIOS 使用汇编语言编写，其开发过程缺乏文档，难以理解和扩展，而 EFI 是一个可扩展的、标准化的固件接口规范。在 Intel 的推广下以及业界对 EFI 认识的不断深入，包括 Intel、AMD、微软等在内的厂家联合成立了 UEFI 联盟，并在 EFI 1.10 的基础上推出了 UEFI 规范^[5]，使用 UEFI 规范开发的 BIOS 程序称为 UEFI BIOS。UEFI 联盟于 2006 年推出 UEFI 2.0 标准和相应的 UEFI 平台初始化(PI 1.0)标准^[6]，并将其框架代码开源，以推动新固件标准的发展。UEFI 固件采用模块化设计，使用 C 语言编写，具有统一的开发标准和规范。UEFI 支持第三方程序运行，具有强大的开放性和可扩展性，其提供了开源代码和开发工具，方便用户开发丰富的应用程序。例如，开源网站 www.tianocore.org 提供了 EDK（EFI Development Kit）开发环境，该环境提供了 UEFI 框架的开源代码和大量的库函数，极大地方便了 UEFI 应用的开发和移植^[7]。

UEFI 作为一个全新的固件规范，虽然推出时间不久，但其已经在 PC 机和服务器上广泛应用。UEFI BIOS 比传统 BIOS 具有很多优点，但也存在诸多的安全隐患。UEFI 在驱动模块实现、驱动加载模式、硬盘存储方式、C 语言编写等问题上使其比传统 BIOS 面临更多的安全威胁^[8]。

近年来，国内外研究学者不断提出针对 UEFI 的攻击方法，常见的攻击方式有 Bootkit 攻击、文件感染等^{[9][10][11]}。早在 2012 年国内一些学者便对 UEFI 的安全性进行了分析，并提出了感染 OS Loader、篡改 NVRAM 变量和插入 EFI Runtime driver 三种攻击方法^[9]。2015 年 3 月份的 CanSecWest 安全大会上，两位安全专家展示了一种针对 UEFI 的攻击方式^[10]。他们利用 UEFI 漏洞将恶意程序植入 UEFI 固件中，并让恶意程序以 SMM 状态运行，从而能够以“隐身”状态偷取计算机内部资料。2015 年 Hacking Team 数据泄露事件中，通过泄露资料人们发现 Hacking Team 已经实现了对 UEFI 的 Rootkit，并将其应用于控制远程目标机器的攻击行为中^[11]。因此，在 UEFI 安全形势如此严峻的情况下，对 UEFI 的安全研究已迫在眉睫。

1.1.2 研究意义

UEFI 作为一个全新的 BIOS 规范，未来必将在 PC 机、服务器、嵌入式系统等诸多领域全面取代传统 BIOS。但是，我国对 UEFI 的研究却始终落后于西方国家。近年来随着我国大力提倡计算机和操作系统国产化，我国在软件和硬件方面的相关技术都得到了极大的突破，然而作为核心固件的 BIOS，其重要的战略地位一直未受重视。从传统 BIOS 到 UEFI，我国对其关键技术的掌握始终落后于国外水平，并且目前我国国内还没有 BIOS 制造商和 UEFI 规范制订的参与者^[2]。因此，在 BIOS 技术改革换代的趋势下，我国必须抓紧对 UEFI 的研究，弥补我国在这方面技术的缺失。

另外，在当前 UEFI 普及和应用高速发展的时期，已经出现了针对 UEFI 的攻击手段，并给人们带来了不同程度的损失^[12]。因此，必须抓紧对 UEFI 安全机制的研究，加强对 UEFI BIOS 的安全防护。UEFI BIOS 是连接硬件与整个计算机系统的纽带，是系统开机的第一道程序，也是计算机系统安全的第一道防线和系统信任链的基石。一旦 UEFI 被攻击者攻破，整个系统的大门将对攻击者敞开，攻击者将拥有根权限对整个系统的任意部分进行攻击而且不容易被发现。然而，目前国内外对 UEFI 的研究大部分是关于 UEFI 的应用和第三方程序的安全检测^[13]，对其安全性的研究不仅起步晚，而且数量少。为了 UEFI 能够健康、稳固地发展，必须加强对其安全机制的研究，找出 UEFI 存在的安全威胁并尽快完善其安全机制。

本文从 UEFI 固件文件系统^[14]入手，结合可信计算理论^[15]，通过研究 UEFI 启

动阶段的控制权传递和信任链设计,提出基于 UEFI 固件文件系统的固件安全方案,解决当前攻击者通过修改固件设备文件对 UEFI 进行攻击的问题。该方案根据可信计算的原理,以 UEFI 启动各阶段的核心模块为基础建立一条信任链,然后使用 Hash 算法对信任链的各个节点进行完整性度量,从而将信任逐级传递下去,保证整个 UEFI 平台在安全可信的环境下运行。

1.2 国内外研究现状

由于 UEFI 的可扩展性以及其丰富的网络功能,更多的应用如主机备份、远程故障诊断等功能将会集成到 UEFI BIOS 上,从而导致操作系统下的恶意代码等安全问题进一步延伸到 UEFI 固件中。因此,如何在 UEFI 架构下构建安全可信的运行环境将是未来 UEFI 规范研究的重点和热点。下面将从 UEFI 的安全威胁和安全机制两方面来说明国内外对其研究的现状。

1.2.1 安全威胁研究现状

在国内,一些研究学者对传统 BIOS 和 UEFI 的运行机制和文件结构进行了研究,并且分别针对传统 BIOS 和 UEFI 提出了恶意代码植入方式和隐藏方式^[16]。文献[9]提出了三种针对 UEFI 的攻击方法,即感染 OS Loader、篡改 NVRAM 变量和插入 EFI Runtime Driver。这三种方法通过不同的方式在 UEFI 启动过程中获得执行权限,最终都能实现劫持操作系统内核。其中插入 EFI Runtime Driver 的方法,利用 UEFI 固件文件系统的格式,将恶意代码伪装成驱动的固件文件进行加载,从而获得执行权限。

在 2015 年 3 月的 CanSecWest 安全大会上,两位安全专家展示了一种针对 UEFI 的攻击方式^[10]。他们利用 UEFI 漏洞将恶意程序植入 UEFI 固件中,并让恶意程序以 SMM 状态运行,从而能够以“隐身”状态偷取计算机内部资料。这种攻击手段非常难以通过远程手段进行,但是一旦攻击成功,防火墙、杀毒软件都无法发现这类程序,除了更新 BIOS 外都无法清除这类恶意程序。

2015 年 7 月, Hacking Team 超过 400GB 的数据被窃取并公布于网络,人们在泄露数据中发现了 Hacking Team 已经实现了对 UEFI BIOS 的 Bootkit 攻击^[11],并用于安装他们的远程控制系统(RCS)被控端到目标机器上。从资料可以看出,这种攻击方式仍然要对目标机器进行物理接触才能实现,进入 UEFI 后,安装工具首先将修改过的 UEFI 固件设备文件更新到目标机器中,然后固件中的恶意代码会从外部拷贝后门程序,并将其安装到目标机器上。

2016 年 3 月,联想的 ThinkPad 系列产品的 UEFI 固件中出现了缓冲区溢出漏

洞^[17]，其固件的实现参考 UEFI 的开源项目 EDKII，而 EDKII 的 Capsule 更新机制中存在多个整数溢出漏洞，可以导致执行任意代码。

1.2.2 安全机制研究现状

国内对 UEFI 安全机制的研究主要集中在建立可信固件平台这方面。文献[18]提出了基于 UEFI 的可信固件平台的概念和模型，并实现了可信固件在 Vista BitLocker 中的应用。文献[19]对 UEFI 的典型实现 Tiano 系统进行了分析，并结合可信计算思想，实现了基于 UEFI 的可信 Tiano 系统。文献[20]对固件文件系统进行了描述，并提出了对固件文件系统中的 EFI 文件进行数字签名认证和完整性校验的安全机制。

在国外，UEFI 厂商已逐步实现了基于可信计算的 TPM (Trusted Platform Module, 可信平台模块) 芯片并用于计算机系统中^[21]。2003 年 4 月 8 日，TCPA (Trusted Computing Platform Alliance, TCPA)重组为“可信计算组织”(Trusted Computing Group, TCG)，其提出了一个基于计算机系统启动的可信计算方案^[22]。该方案提出在硬件和操作系统之间构建可信计算平台，并将其在硬件中实现。该平台在计算机启动过程中对运行在该平台上的所有程序提供保护，从而大大的增强了系统的可信级别。目前，Intel、IBM 等国外芯片厂商都推出了自己的 TPM 芯片，并将其广泛应用于生产的笔记本电脑和台式机中。

2006 年，TCG 组织和 UEFI 联盟联合推出了将可信计算思想用于 UEFI 固件的两个规范：TCG EFI Platform Specification^[23]和 TCG EFI Protocol Specification^[24]。这两个规范弥补了关于如何规范在 UEFI 整个启动过程中构建可信任链的空白，也对如何架构安全可信的 UEFI 模型提供了方向。TCG EFI Platform Specification 根据 UEFI Framework 规定的启动顺序，规范了各个启动阶段中平台配置寄存器的使用规则 and 如何进行完整性校验。TCG EFI Protocol Specification 则是定义了一套软件借口，统一了对 TPM 芯片的操作方法。

如上所述，国内外普遍使用可信计算技术加强对 UEFI 固件的保护。可信计算的核心思想就是建立可信链，然后沿可信链传递信任和系统的控制权。由于可信计算技术已经逐渐成熟，在保护固件的安全性方面具有良好的效果，并且信任链的设计思想与 UEFI 启动过程结合紧密，因此本文采用可信计算的思想来实现 UEFI 固件安全方案。

在基于可信计算的 UEFI 安全机制研究中，国内外对可信度量的研究重点仍然集中在对第三程序的认证和对固件实体的完整性度量上，而忽略了对 UEFI 固件文件系统的保护。本文从固件文件系统的角度，对固件中的核心模块和驱动模块

文件进行保护，防止攻击者利用固件文件系统的缺点对 UEFI 固件进行破坏。

1.3 论文主要研究内容

本文主要是对 UEFI 规范和可信计算技术进行研究，分析固件文件系统的架构和安全威胁，通过解决以下关键技术问题，提出基于固件文件系统的 UEFI 安全方案，从而解决目前攻击者利用固件文件系统对 UEFI 固件进行攻击的问题。

1、对 UEFI 的框架及可信计算理论进行研究，为建立基于 UEFI 启动过程的信任链提供思路。本文从 UEFI 系统框架出发，通过 UEFI 的系统服务和驱动模型来阐述 UEFI 工作的基本原理，并分别对 UEFI 的整体安全性和启动各阶段的安全性进行分析。然后对可信计算理论进行研究，分析可信计算技术与 UEFI 启动过程的结合点，建立一条基于 UEFI 启动过程的信任链。

2、分析固件文件系统架构及其面临的安全威胁，研究攻击者利用固件文件系统进行攻击的原理。本文首先描述整个固件文件系统的结构，然后从文件系统的各个层面由外到内逐一进行分析，剖析一个固件设备文件的组成结构，并分析 UEFI 启动各阶段功能在固件设备文件中的具体实现方式。本文对固件文件系统的安全性及已有的安全校验机制进行分析，并对两种利用固件文件对 UEFI 进行攻击的方法进行研究。

3、结合可信计算理念和 UEFI 启动各阶段在固件设备文件中的实现形式，提出一个 UEFI 固件安全方案。该方案根据 UEFI 启动过程建立一条信任链，并阐述信任链的传递过程和可信度量机制，然后通过对各个启动阶段的核心固件文件和驱动程序的固件文件进行安全校验，完成对整个固件设备文件的可信度量。

4、根据安全方案，本文设计并实现基于 UEFI 驱动模型的固件安全模块。该安全模块由主模块、Hash 计算引擎、通信单元及存储模块组成。固件安全模块对 UEFI 固件安全方案进行了具体实现，通过 Hash 计算引擎对固件文件进行完整性测量，然后对比存储模块中的可信度量日志来确定固件文件的可信性。本文最后通过实验论证安全方案的可行性及优缺点，并结合实验结果分析其优缺点以便进一步地改进。

1.4 论文组织结构

本文共分为七章，各章内容安排如下：

第一章：介绍本文研究课题的背景及意义，并结合国内外研究现状，确定论文的主要研究内容。

第二章：研究 UEFI 的整体框架和 Framework 结构，然后对其系统服务和安全

性进行详细分析，最后重点研究可信计算理论。

第三章：研究 UEFI 的固件文件系统，对 UEFI 固件设备文件的各个组成部分进行详细地解析。

第四章：结合 UEFI 固件文件系统的特点，分析其存在的安全威胁，并对基于固件文件系统的攻击方法进行研究。

第五章：结合可信计算思想和 UEFI 固件文件系统的特点，提出基于固件文件系统的 UEFI 固件安全方案。

第六章：通过设计和实现固件安全模块来对 UEFI 固件安全方案进行实现，并通过测试实验对安全方案进行验证。

第七章：总结论文所做的工作，并展望下一步的研究方向。

第二章 UEFI 规范及可信计算技术研究

UEFI 是一个开放的业界标准接口规范，它定义了固件平台必须实现的一组接口和结构^[5]。UEFI 固件运行在底层硬件和操作系统之间，其启动过程的特点与可信计算理论结合紧密。本章将详细阐述 UEFI 的框架结构以及其提供的接口服务，并就其启动过程进行分析，然后分别从信任链技术、可信度量机制两方面对可信计算理论进行研究。

2.1 UEFI 基本框架

UEFI 提供了一个统一可扩展的固件平台，并针对平台特性定义了一系列接口。该平台整体处于硬件与操作系统中间，平台最上层的可扩展固件接口包含了平台提供的 API 函数、启动时服务（EFI Boot Services）、运行时服务（EFI Runtime Services）和操作系统引导程序，下层则是根据 UEFI 规范实现的平台固件。UEFI 的平台架构如图 2-1 所示：

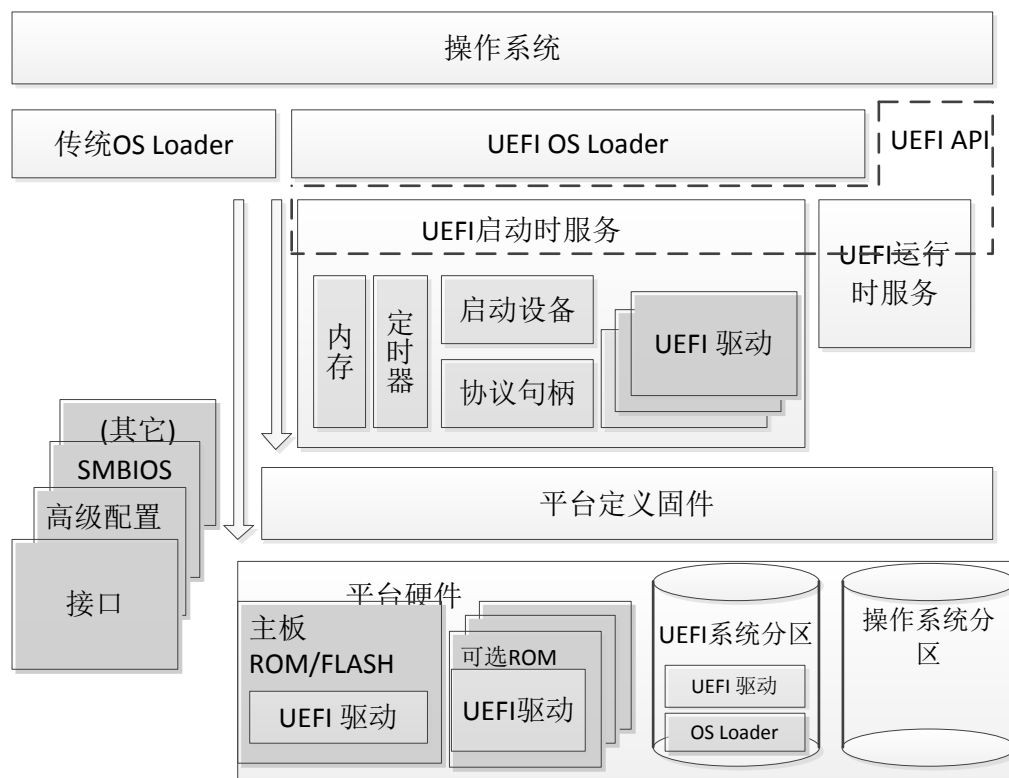


图 2-1 UEFI 系统框架图

图 2-1 描绘了 UEFI 框架中各模块的关系，UEFI 固件作为承上启下的模块，

对底层硬件进行了抽象处理，又不断地对上层的操作系统提供服务，并在不同的服务层的连接中采用了标准的接口。图中 UEFI 保留了对传统 BIOS 引导操作系统的兼容性，所以在可扩展固件接口的实现中有两种操作系统引导方式，分别为 Legacy OS Loader 和 UEFI OS Loader。

UEFI 允许操作系统预处理，实现了操作系统的引导和一些系统软件执行所需要的其它应用程序，如诊断程序、UEFI Shell、系统调试软件等^[25]，这些程序统称为 UEFI 实体（UEFI Image）。根据 UEFI 规范，UEFI Image 包含三种：UEFI 应用程序、UEFI 驱动和 OS Loaders，这些实体都是在 UEFI API 调用的基础上实现的。UEFI 系统服务是 UEFI 整个框架的重要组成部分，UEFI 驱动又是 UEFI 系统服务的主要使用者，也是 UEFI 固件功能实现的最重要的形式，因此下面将重点介绍 UEFI 系统服务和 UEFI 驱动模型。

2.1.1 UEFI 系统服务

系统服务表（EFI System Table）是 UEFI 系统服务中最重要的数据结构，它包含了 UEFI 提供的所有服务和其它描述平台配置信息的数据结构，其入口地址的指针会传递给每一个 UEFI 驱动和应用程序。从 EFI System Table 中，UEFI Images 能得到系统的配置信息^[5]，如 ACPI、SMBIOS 表等和丰富的 UEFI 服务，包括标准输入输出服务、启动时服务、运行时服务和各种各样的协议服务。这些服务以指针的方式存放在 EFI System Table 中，EFI System Table 包含内容如表 2-1 所示。

UEFI 提供了控制台标准输入输出服务，实现在系统服务表中，该服务的使用者可以通过系统服务表的指针直接进行访问。控制台输入输出服务由两个主要协议组成，分别是 EFI 简单文本输入协议和 EFI 简单文本输出协议。

启动时服务和运行时服务是 UEFI 提供的最重要的服务，它们通过各自的服务表来进行访问，这两个表的入口地址存在于 EFI System table 中，表里面可使用的服务个数和类型在每个 UEFI 的版本中有明确的规范^[5]。这两种服务通过接口函数进行定义，通常被运行在 UEFI 环境中的代码调用，这些代码包括管理设备访问程序、扩展平台功能程序、在预引导环境中运行的应用程序以及操作系统引导程序。

启动时服务主要提供 UEFI 启动时需要使用的服务，如任务调度、内存操作、事件处理以及文件管理等，这些服务是 UEFI 实现其基本功能如加电自检、初始化硬件等必须使用的。在 UEFI 启动后，每个 UEFI Image 都会被提供一个指向系统服务表的指针，通过该指针可以得到启动服务调度表以及一个访问控制台的默认句柄。此时所有的启动时服务对 UEFI Image 都是可用的，直到 OS Loader 加载完操作系统内核，调用 ExitBootServices 函数，之后 UEFI 启动时服务不可再被使用。

表 2-1 UEFI 系统服务表

指针类型	指针名称	含义
EFI_TABLE_HEADER	Hdr	系统服务表的头部，包含系统服务表的数字签名、大小和一个 32 位的 CRC 校验和
CHAR16	*FirmwareVendor	指向固件供应商的指针
UINT32	FirmwareRevision	固件版本
EFI_HANDLE	ConsoleInHandle	控制台输入设备句柄
EFI_SIMPLE_TEXT_INPUT_PROTOCOL	*ConIn	指针指向简单文本输入协议
EFI_HANDLE	ConsoleOutHandle	控制台输出设备句柄
EFI_SIMPLE_TEXT_OUTPUT_PROTOCOL	*ConOut	指针指向简单文本输出协议
EFI_HANDLE	StandardErrorHandle	控制台标准错误输出句柄
EFI_SIMPLE_TEXT_OUTPUT_PROTOCOL	*StdErr	指针指向简单文本输出协议
EFI_RUNTIME_SERVICES	*RuntimeServices	指向运行时服务的指针
EFI_BOOT_SERVICES	*BootServices	指向启动时服务的指针
UINTN	NumberOfTableEntries	缓冲区中系统配置表的数量
EFI_CONFIGURATIN_TABLE	*ConfigurationTable	系统配置表的指针

与启动时服务不同的是，运行时服务在 OS Loader 加载操作系统后仍可使用。UEFI 运行时服务主要包括 UEFI 变量服务、时钟服务、虚拟内存服务及一些杂项服务，这些服务在操作系统加载后通常提供一些辅助性的服务，并且存在很大的局限性，比如当需要保护硬件访问资源时，运行时服务功能将会在内部禁用。

2.1.2 UEFI 驱动模型

传统 BIOS 使用硬件中断、端口操作的方法来实现 BIOS 的基本功能，而 UEFI 则采用了 Driver/Protocol 的新方式来实现诸如故障检测、电源管理、操作系统引导等 UEFI 基本功能，大大提高了效率并增加了对新硬件的支持。为了支持现有的工业标准总线和未来的架构，统一各 BIOS 生产厂家的实现 UEFI 驱动时采用的技术形式，UEFI 提供了一个驱动模型，期望能够简化设备驱动的设计和执行，并减小可执行映像的大小。

2.1.2.1 UEFI 驱动模型简介

对于一个特定的平台，由固件服务程序、总线驱动程序和设备驱动程序构成的组合很可能由不同的开发厂商来生产，比如独立 BIOS 开发商(Independent BIOS Vendor, IBV)、原始设备制造商(Original Equipment Manufacturer, OEM)和独立硬件开发商(Independent Hardware Vendor, IHV)。而由这三种厂商生产的组件又必须相互协作来提供引导操作系统的服务，这时就必须建立 UEFI 驱动模型，来规范 UEFI 驱动的设计和开发^[26]。

UEFI 驱动由控制台来统一进行管理和执行，UEFI 驱动模型规定驱动不允许自行搜索可用的控制器来进行启动。在 UEFI 驱动的实现中，每一个驱动都有一个句柄与之相对应，其功能往往通过在驱动句柄上绑定一个或多个协议，调用协议里的相关函数来实现。图 2-2 展示了句柄数据库里的一个句柄及其协议被一个驱动使用时的情况。

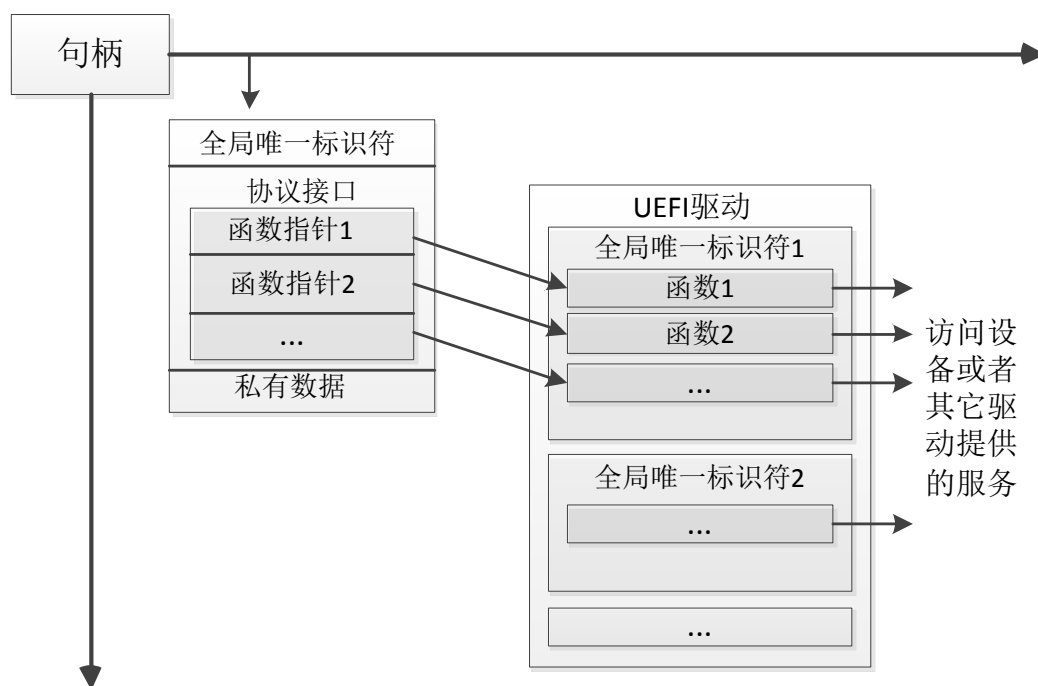


图 2-2 协议产生与使用

UEFI 的可扩展特性往往由协议的组合来实现，驱动程序和协议的关系虽然很密切，但它们显然是不同的两个概念。驱动程序是一个可执行的 UEFI 映像，它通过同时安装多种协议和句柄来实现它的功能，而 UEFI 协议是一些函数指针和数据结构体或者规范定义的 API 的集合。EFI_DRIVER_BINDING_PROTOCOL 协议是每一个遵循 UEFI 驱动模型的驱动都必须绑定的一个协议，它是控制驱动执行的核

心组件。当控制台分配一个可用的控制器时，它会调用 UEFI 启动时服务 ConnectController()根据驱动的 EFI_DRIVER_BINDING_PROTOCOL 协议来确定一个最合适的驱动进行执行。

UEFI 驱动模型中，驱动的入口程序必须遵循一些严格的规则。首先，不允许任何硬件的操作，只能选择性地安装一些协议实例到自己的驱动句柄上，其中必须安装一个 EFI_DRIVER_BINDING_PROTOCOL 实例。一个遵循 UEFI 驱动模型的驱动句柄示例如图 2-3 所示：

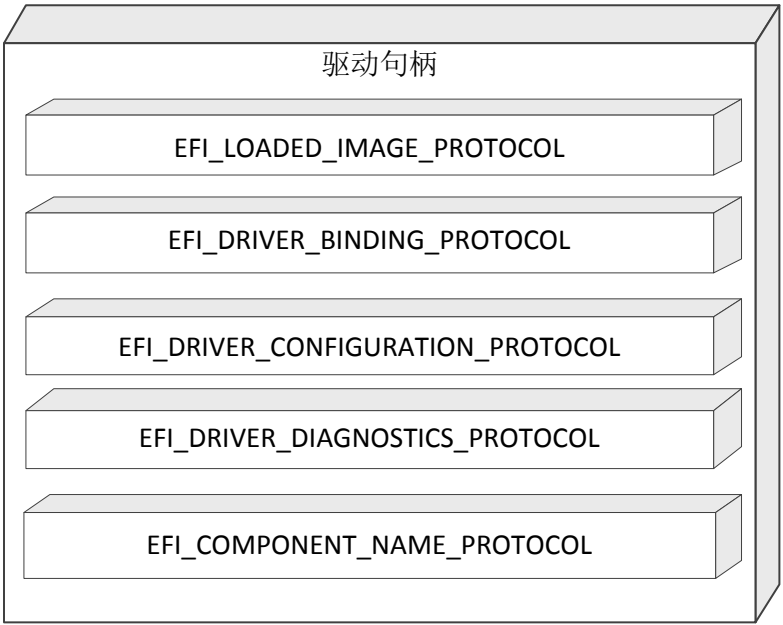


图 2-3 安装驱动协议的句柄示例

另外，驱动可以选择性地安装一些协议，如 Driver Configuration Protocol、Driver Diagnostics Protocol 等，如果驱动本身可以被卸载，它可以选择性地更新 Loaded Image Protocol 来提供自己的 Unload 接口，驱动所支持的协议及其功能参见表 2-2。

表 2-2 UEFI 驱动模型支持的协议及功能

协议名称	协议功能
驱动绑定协议	查看驱动程序是否用于特定硬件设备，开始和终止设备驱动程序运行
组件名称协议	为控制的硬件提供命名服务
驱动诊断协议	提供调试诊断硬件设备的接口
驱动配置协议	为用户提供配置驱动所控制硬件的功能

2.1.2.2 UEFI 驱动分类

UEFI 驱动是 UEFI 功能实现的主要形式，UEFI 的重要功能模块基本都是以驱动的形式存在，如电源管理模块、操作系统引导模块、CPU 管理模块等。根据 UEFI 驱动不同的功能，又可以将 UEFI 驱动划分为以下几种驱动：服务驱动、初始化驱动程序、启动时桥接驱动和 UEFI 驱动模型驱动。UEFI 驱动模型驱动又分别包含设备驱动、总线驱动及混合驱动。UEFI 的各类驱动及其含义如表 2-3 所示：

表 2-3 UEFI 驱动分类及功能

驱动类型	驱动功能
总线驱动	遵循 UEFI 驱动模型，会在句柄数据库中产生一个或者多个驱动句柄，并在句柄上安装 Driver Binding Protocol 的实例。这种类型的驱动在驱动绑定协议的 start 函数调用时会产生新的子句柄，并在子句柄上增加另外的 I/O 协议。
设备驱动	遵循 UEFI 驱动模型，与总线驱动不同的是，这种类型的驱动在 start 函数被调用时不产生子句柄，只是在现有的控制句柄上增加额外的 I/O 协议。
启动时桥接驱动	这种类型的驱动会产生一个或多个控制句柄，控制句柄包含一个 Device Path Protocol 和一个对芯片根总线提供的 I/O 服务抽象出来的协议。
混合驱动	遵循 UEFI 驱动模型，并且兼容了设备驱动和总线驱动的特点，即不仅会在现有控制句柄上增加 I/O 协议，也会产生新的子句柄。
服务驱动	这种驱动会在服务句柄上加载一个或多个协议，并且它的入口点会返回 EFI_SUCCESS。
初始化驱动	这种驱动不会产生任何句柄，也不会句柄数据库中增加任何协议，它只会进行一些初始化操作，并返回错误代码。

2.1.3 UEFI 映像文件

UEFI 映像文件是一组包含可执行代码的文件，是 UEFI 规范内所有可执行程序统称。所有的 UEFI 映像文件都包含一个 PE/COFF 文件头，具体格式遵循微软的 PE/COFF 文件格式规范，支持 IA32、X64、IPF、EBC 等处理器架构。UEFI 映像文件通常可以存储在以下几个位置：

- (1) PCI 卡上的扩展 ROM；
- (2) 系统 ROM 或者系统 flash 芯片；
- (3) 媒体设备，如硬盘、软盘、光盘、闪存驱动器等；

(4) LAN 服务器。

UEFI 映像文件通过启动时服务的 `LoadImage` 函数加载到内存。该函数首先在内存中开辟一段空间，然后通过重定位确定加载内存地址并将映像文件整体映射到开辟的内存中，最后为该映像文件在句柄数据库里注册一个绑定有 `EFI_LOADED_IMAGE_PROTOCOL` 实例的句柄。一般来说，UEFI 映像文件在编译和链接时并没有确定具体的加载位置，其位置通常在加载时通过重定位确定，从而可以将映像文件加载到任何一段系统内存中。当映像文件成功加载后，其会调用启动时服务 `StartImage`，`StartImage` 会引导处理器跳到映像文件的入口地址处执行。UEFI 映像文件的入口地址保存在文件头中，通常保存着两个重要的参数信息：UEFI 映像文件句柄和一个指向 UEFI System Table 的指针。通过这两个参数，映像文件可以获得自身的加载信息和 UEFI 系统服务。

UEFI 映像文件主要分为两大类：UEFI 应用程序和 UEFI 驱动程序。这两类程序都是通过 `LoadImage` 和 `StartImage` 执行，但是在程序退出时，应用程序会直接被卸载，驱动程序只有在出错或者调用 `UnloadImage` 的情况下才会被卸载。图 2-4 展示了 UEFI 映像文件的分层以及不同层次的映像文件之间的联系。

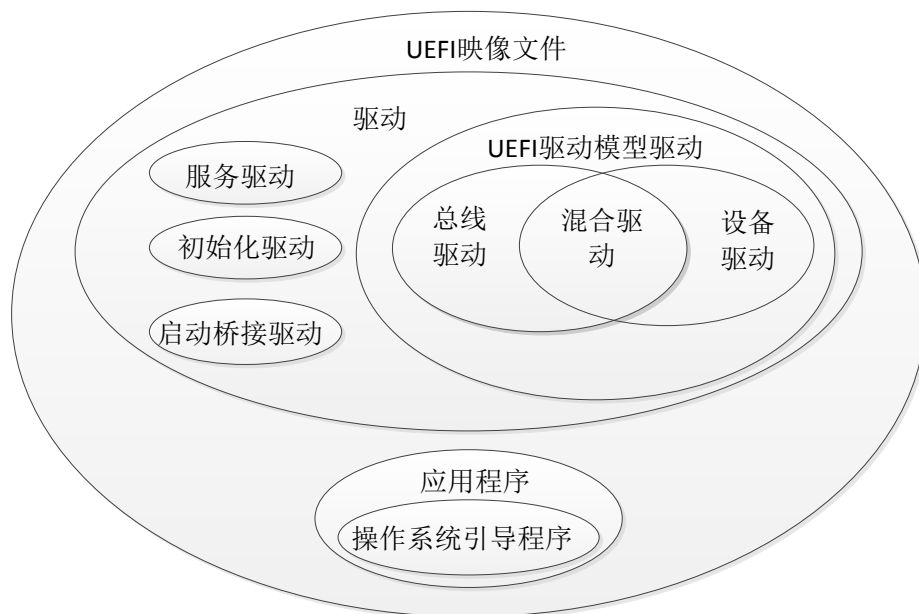


图 2-4 UEFI 映像文件分类

如图 2-4 所示，UEFI 驱动和应用程序组成了 UEFI 映像文件，它们下面又由不同类型的文件组成。其中，OS Loader 也是一个 UEFI 应用程序，与普通应用程序不同的是，OS Loader 主要功能是引导操作系统，在该程序退出后所有 UEFI 应用程序都将退出，整个计算机系统将由操作系统控制。

2.2 UEFI Framework 结构

Framework 是 Intel 对 UEFI 规范的实现方案，Intel 将该工程的开发代号命名为 Tinao，并将其部分源代码开源公布在网上。现在的 UEFI BIOS 开发厂家如 IBV、OEM、IHV 等对 BIOS 的实现都是以 Intel 所编写的 Framework 核心代码为基础，再以 UEFI 驱动模块或应用程序的形式插入一些自己编写的功能模块。Framework 采用模块化设计，根据 UEFI 规范将其启动过程分为各模块进行实现^[14]，为其它厂商开发 UEFI 固件提供了通用的架构，也大大方便了第三方厂商对平台的扩展。

Framework 作为 Intel 对新一代 BIOS 规范的具体实现，引入了现代计算机系统结构设计的思想，将整个 UEFI 执行的流程主要划分为 SEC、PEI、DXE、BDS、TSL、RT 和 AL 等 7 个阶段，如图 2-5 所示：

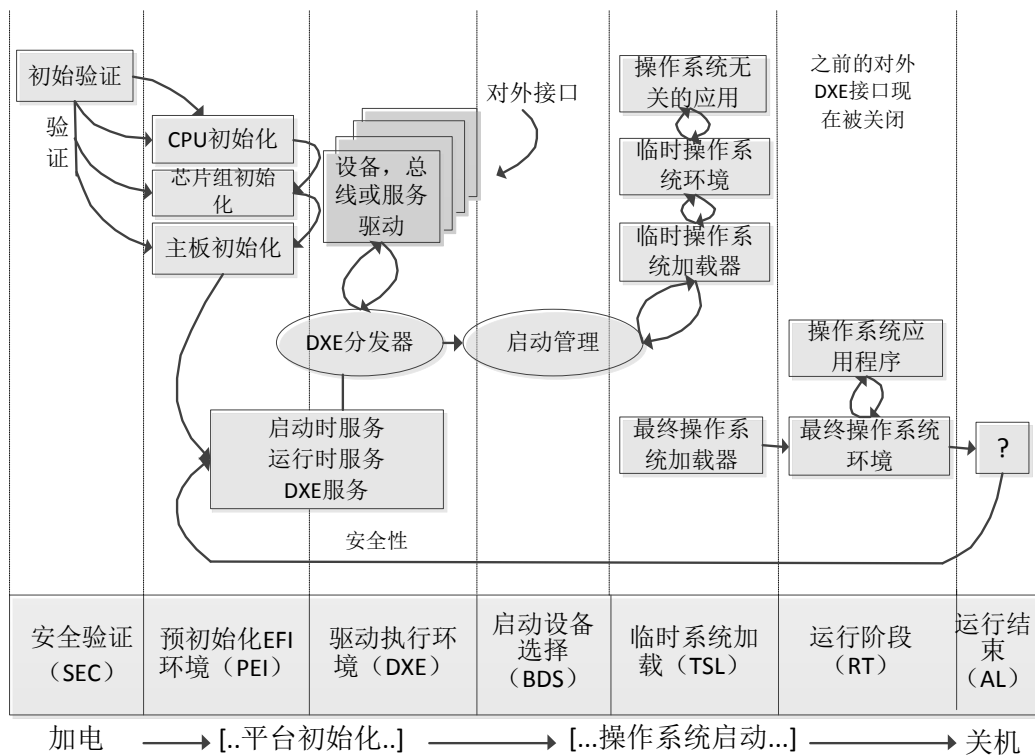


图 2-5 UEFI 启动流程

从图中可以看出整个 Framework 架构描述了从加电到平台初始化，再到操作系统引导，最后到系统结束的整个流程。下面将分阶段来介绍 Framework 的启动流程：

1、SEC (Security) 阶段

SEC 阶段是平台加电后进入的第一个阶段，是 Framework 架构的可信起点，主要是对后续阶段的代码进行安全校验。SEC 阶段代码以汇编程序为主，其首先

会刷新处理器的缓存并使其作为临时内存使用，然后使 CPU 转化为非分页的平坦保护模式。SEC 阶段是整个平台安全启动的基础，会对早期的初始化代码进行验证，确保执行代码的可信。SEC 阶段最后负责找到 PEI 阶段核心代码，检验其可靠性并向其传递 UEFI 平台控制权。

2、PEI(Pre-EFI Initialization)阶段

PEI 阶段是 UEFI 实现的核心部分，主要进行内存的完全初始化，并提供一个对 CPU、芯片组和其它平台固件的初始配置规程，最后通过 HOBs(Hand-off Blocks, 一个存储系统信息的数据结构)列表向 DXE 阶段传递整个平台的信息和控制权。UEFI 的编码实现有 99% 由 C 语言完成，因此在 PEI 阶段还需要建立内存堆栈和 C 语言的执行环境。PEI 阶段由 PEI 基础模块(PEI Foundation)和一个或多个 PEIM (Pre-EFI Initialization Module) 组成。

3、DXE (Driver Execution Environment)阶段

DXE 阶段是整个 UEFI 启动过程中最为核心的阶段，主要包含三个组件：DXE 基础模块、DXE 分发器和 DXE 驱动程序，同时提供了 UEFI 系统服务：启动时服务和运行时服务。UEFI 的大部分功能都是以驱动的形式实现，而 DXE 阶段功能就是为驱动程序提供服务、建立驱动执行环境，并且分发和调度驱动程序执行。

进入 DXE 阶段后，DXE 基础模块负责建立驱动执行环境，同时提供启动时服务和运行时服务，DXE 分发器则负责搜索所有的 DXE 驱动并按照正确的顺序来加载执行。DXE 驱动是一组由驱动开发厂商按照 UEFI 驱动模型实现的 UEFI 映像文件，用来实现初始化处理器、芯片组等功能，同时对系统服务、控制台设备和启动设备进行抽象，常见的一些 DXE 驱动有 PCI 控制驱动、显卡驱动、网卡驱动、UEB 驱动等。当所有驱动加载完成后，平台启动的准备工作也就全部完成，这时可以加载一些 UEFI 应用程序（如诊断程序、Flash 更新程序等）运行，也可以进入 BDS 阶段进行操作系统的引导工作。

4、BDS (Boot Device Selection) 阶段

BDS 阶段是 UEFI 对操作系统启动的管理阶段。BDS 阶段代码和 DXE 阶段基础模块共同组成一个控制台，控制台确定可用的启动镜像和引导设备之后会提供给用户一个启动列表，由用户选择从何种设备上启动操作系统，然后进入操作系统引导阶段。

5、TSL (Transient System Load) 阶段

TSL 阶段即操作系统引导阶段，此阶段中的控制权开始逐步移交给操作系统。此阶段中，OS Loader 会加载并初始化操作系统内核，当操作系统启动完成之后，OS Loader 会退出并且关闭 UEFI 启动时服务，之后将会进入操作系统运行阶段。

6、RT (Runtime) 阶段

RT 阶段是操作系统运行阶段，这时的 UEFI 的大部分资源都已经释放，启动时服务已经不能使用，运行时服务仍然驻留在内存中供操作系统调用。

7、AL (After Life)阶段

AL 阶段是 RT 阶段的后续阶段，当正常退出操作系统或意外关机时控制权将从操作系统回到 UEFI 固件中。

2.3 UEFI 整体安全性分析

相比传统 BIOS，UEFI 的灵活性和开放性给计算机系统带来了巨大的发展空间，但是其没有彻底消除传统 BIOS 的所面临的安全问题，并且由于 UEFI 采用高级语言实现、功能丰富、扩展性强等特点，将面临新的安全威胁。下面将从 UEFI 平台安全和 UEFI 启动阶段的安全两方面来分析 UEFI 的安全性。

2.3.1 UEFI 平台中存在的安全隐患

UEFI 使用现代的系统架构，并且在最新版本的 UEFI 规范中添加了基于数字签名的固件安全启动的概念性模型，大大增强了自身的安全性。但是由于 UEFI 平台的实现和存储缺乏完善的安全机制^[8]，仍然存在着许多安全隐患。结合 UEFI 固件平台的特点，本文总结 UEFI 平台的安全隐患存在以下几个方面：

- 1、 由于需要提供一个可扩展的接口服务，UEFI 将其系统服务全部公开，并且在其规范中明确描述。这使得攻击者可以通过阅读 UEFI 规范掌握 UEFI 系统服务的详细信息，从而根据一些特殊系统服务与操作系统启动的关联性达到劫持操作系统的目的。ExitBootServices 是 UEFI 的运行时服务，其实现和功能在 UEFI 规范中详细描述，文献[28]提出的 UEFI Bootkit 模型中，就是通过对 UEFI 运行时服务 ExitBootServices 进行 hook 获得系统的控制权。
- 2、 UEFI 可扩展性强，允许第三方厂商开发的模块在 UEFI 启动过程中得到执行。但是 UEFI 在对第三方模块进行安全认证的措施仍然不足^[29]，UEFI 规范的完全公开使得恶意的第三方模块可以轻易地绕过安全认证，从而在 UEFI 启动过程中获得执行权限。文献[9]提出的插入 EFI Runtime Driver 攻击方法便是通过在固件中插入驱动模块实现的。
- 3、 UEFI 的编码绝大部分使用 C 语言，并且 Intel 对 UEFI 实现的代码是开源的，这使得 UEFI 代码面临许多传统的针对 C 语言的攻击，其中最常见的就是缓冲区溢出攻击^[30]。C 语言中对内存的使用时非常灵活的，如果在编译时对数据边界检查不严格，向缓冲区内写入了超过其长度的数据就会造成缓冲区溢出。

更重要的是, Framework 源代码是公开的, 攻击者可以借助开发工具建立一个对源代码可调试的开发环境, 从而使攻击者在进行漏洞挖掘、漏洞调试时具有极大的便利。2016 年 3 月 2 日, 联想 ThinkPad 产品的 UEFI 中出现了缓冲区溢出漏洞, UEFI 在对 Capsule 图形进行解析时出现多个整数溢出漏洞, 可导致执行任意代码^[17]。

2.3.2 基于 Framework 的 UEFI 启动阶段安全性分析

本文在对 UEFI 启动的 DXE 和 BDS 阶段的分析过程中, 发现 UEFI 大部分功能的实现形式是 UEFI 驱动程序和应用程序, 在 DXE 阶段和 BDS 阶段驱动程序和应用程序通过固件文件模块的形式分别被加载和执行。但是, 在加载和执行这些模块时, UEFI 并没有严格的安全机制保证这些模块的安全性, 从而导致攻击者通过修改配置文件、篡改或替换驱动模块等方式获得执行权限, 甚至可以通过扰乱驱动模块的加载顺序使被攻击机器不能正常启动。

从 Framework 划分的各个启动阶段的特点来看, UEFI 固件平台在启动过程中仍然存在着诸多的安全性问题^{[8][9][27]}。下面将从几个具体的阶段来分析其安全性:

1、SEC 阶段安全性分析

SEC 阶段是整个 UEFI 平台的可信根, 是保证 Framework 架构安全启动的重要阶段。SEC 阶段执行可信度量根的核心执行代码, 并使用哈希扩展函数对 PEI 加载映像进行校验, 确保 PEI 阶段执行代码的可信。

SEC 阶段被默认是绝对安全的, 它的安全性是由厂家保障的^[8], UEFI 厂商预留接口, 确保处理器执行的第一条指令是完全可信安全的, 但是一旦该接口泄露, 将破坏整个平台的可信链。

2、DXE 和 BDS 阶段安全性分析

DXE 阶段是 UEFI 启动的核心阶段, 它提供完整的 UEFI 系统服务, 并检索固件卷中所有的驱动模块并按照正确的顺序加载执行。DXE 和 BDS 共同组成控制台, 提供给用户 EFI Shell 命令、管理和启动 UEFI 应用程序及 OS Loader 等功能。但是, UEFI 并没有对加载的驱动和应用程序进行严格的校验^[27]。

另外, UEFI 规范提供了 TCP、IP、ARP 及 DHCP 等网络协议, 这些网络协议的应用使得平台可以执行远程命令及配置操作, 而 BDS 阶段给用户提供了 EFI Shell 命令、管理和启动应用程序及 OS Loader 等功能。如果没有安全机制保证, 攻击者可能远程执行命令、修改配置文件甚至操纵操作系统的引导。

3、TSL 阶段安全性分析

TSL 阶段是操作系统引导的重要阶段, 它会按照 BDS 阶段选择的 OS Loader

来启动操作系统。然而，UEFI 并没有对 OS Loader 进行完整性校验，而是由 OS Loader 自身来完成，攻击者只要能够突破 OS Loader 的自校验函数即可篡改 OS Loader 并添加恶意代码^[9]。

2.4 可信计算技术

随着计算机技术的深入应用，计算机系统面临着诸如恶意代码攻击、信息窃取和系统非法破坏等安全威胁，这些安全威胁的根源在于没有从计算机的体系架构上建立一个可信赖机制，实现计算机平台安全可信地运行^[31]。可信计算就是在此背景下提出的一种安全技术，它通过在计算机系统中建立一条信任链和可信度量机制，使其在运行过程中具有识别代码是否可信的能力，从而能够有效地防范恶意代码的攻击。

2.4.1 可信计算理论简介

可信计算是指在计算机和通信系统中使用基于硬件的安全模块，以提高系统整体的安全性。该硬件模块即可信计算平台，其按照可信计算思想设计，旨在确保系统中数据的完整性、存储的安全性以及整个计算机平台的可信性。

可信计算的基本思想是首先在计算机系统中确立一个可信度量根，可信度量根是整个平台可信的基础，其可信性一般由物理安全、技术安全和管理安全来确保^[32]。然后再从可信度量根开始，根据控制权的传递过程建立一条信任链，该信任链从可信度量根开始，依次从固件到操作系统，再到应用程序，逐级传递信任，从而将信任扩展到整个计算机系统，保证整个系统的可信。

可信计算平台是指应用可信计算技术对计算机系统进行保护的模块，它通常通过可信赖的硬件和软件建立起可信任的计算机通信基础，从而可以在更底层进行更高级别的保护。一个可信计算平台通常由硬件平台、操作系统和应用程序三部分组成，如图 2-6 所示：



图 2-6 可信计算平台示例图

可信计算平台需建立一条基于完整性度量技术的信任链，平台从信任链的起点到终点依次进行计算确保计算机系统在运行时是安全可信的，从而对应用程序的完整性和机密性提供保护。可信平台还可对外提供如平台身份认证、完整性报

告等安全服务。在可信计算平台中，如何建立一条完整的信任链是需要解决的核心问题，下面将具体介绍信任链技术。

2.4.2 信任链技术

实现可信计算平台的关键是建立一条传递信任的链式结构，即信任链。通过构建信任链可以在系统控制权传递的过程中不断将信任传递下去，从而使可信计算平台在安全可信的环境下启动。信任链节点之间的可信性主要通过可信度量机制来判断，通常使用完整性度量作为可信度量方法，即通过对信任链节点进行完整性度量来判断该节点的可信状态。

可信度量根是信任链中首先获得控制权的节点，其可信性通常由厂商和硬件确保，用来确保整个信任链的可信，因此也将其称作核心可信度量根（core root of trust for measurement, CRTM）。TCG 在对计算机系统建立 BIOS Boot Block->BIOS->OS Loader->OS 这样的信任链，其中 BIOS Boot Block 是可信根^[33]。TCG 采用迭代计算 Hash 的方式来进行完整性度量，不断将新的 Hash 值作为完整性度量值储存在平台配置寄存器 PCR 中，计算公式如（2-1）所示：

$$New\ PCR_i = HASH(Old\ PCR_i \ //\ New\ Value), \text{其中符号} // \text{表示连接} \quad \text{公式 (2-1)}$$

TCG 给出的计算机系统的信任链模型实现了可信计算的基本思想，即从可信根开始到硬件平台，再到操作系统，最后到应用程序。如图 2-7 所示，

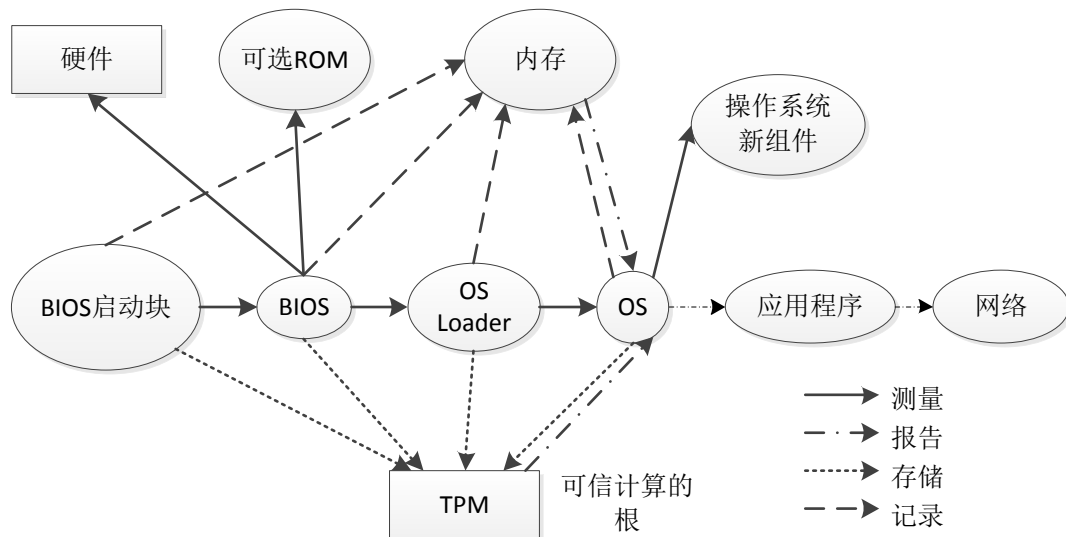


图 2-7 TCG 提出的计算机系统的信任链

TCG 对信任链建立的具体过程为：系统加电开始运行可信度量根 BIOS Boot Block，引导并验证 BIOS 的完整性，若 BIOS 可信则将控制权传递给 BIOS。然后

BIOS 对引导扇区进行完整性度量，若引导扇区可信则将控制权转交给引导扇区，这样依次类推，引导扇区度量 OS、OS 度量应用程序、应用程序度量网络，从而将信任依次传递，保证整个系统的可信性。该信任链不仅实现简单，而且与现有计算机有较好的兼容性。

2.4.3 可信度量机制

在可信计算平台启动过程中，由可信根出发根据信任的扩展机制建立一条信任链，而信任链之间的信任传递主要依靠可信度量机制。按照信任链的理论，信任链中的每一个模块在得到信任运行之前，都必须经过可信性的度量。完整性度量是现在主流的可信度量方法，它通过 Hash 函数或者消息验证码对平台模块进行完整性校验，保证系统的完整性不会遭到病毒或者木马程序的破坏。

可信度量机制由可信度量计算、可信度量存储、可信度量报告和可信度量验证四个部分组成^[34]，四者关系如图 2-8 所示。本文采用完整性度量机制作为可信度量方法，并在可信平台模块中实现。

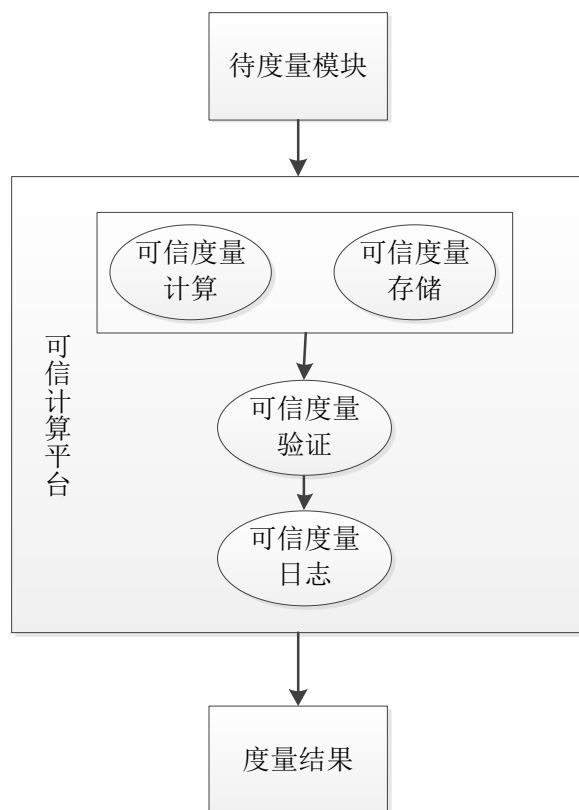


图 2-8 可信度量机制的组成

可信度量计算是指对与平台完整性相关模块进行度量，并将度量值存储于可信平台模块内部。可信计算平台在对模块进行完整性度量时，会在可信存储区内产

生用于对比的可信基准值。在完成某模块的完整性度量后，平台将度量值与基准值进行比较，如果结果相同则将信任传递给该模块。

可信度量存储主要是对可信度量中产生的日志信息和度量值进行存储，其中可信度量值通常存储在可信平台中，在信任链的传递过程中仍要参与可信度量计算。

可信度量报告是指统计可信度量存储的内容，能够提供可信的度量值和可信度量事件日志信息。

可信度量验证是指对可信度量报告中的数据按照可信度量个过程进行验证计算，从而得出当前的可信计算平台是否可信的结论。

2.5 本章小结

本章主要是对 UEFI 框架及其重要概念进行介绍，并分析了 UEFI 的安全性。首先整体分析 UEFI 的框架，展示 UEFI 框架中各个模块之间的关系。并对 UEFI 框架中三个重要组件进行了详细地介绍和分析，包括 UEFI 系统服务、UEFI 驱动模型和 UEFI 映像文件。然后本章结合图 2-5 重点分析了 UEFI Framework 架构，并对各个 Framework 中各个阶段的功能及特点进行分析，并从 UEFI 平台和 UEFI Framework 启动流程两个方面分析了 UEFI 的安全性。最后，本章分别从信任链技术、可信度量机制和可信度量模块三个方面可信计算技术进行研究。

第三章 UEFI 固件文件系统研究与分析

UEFI 作为新一代 BIOS 规范，其对固件设备文件的组织和存储也进行了明确的规定，UEFI 固件设备文件采用的文件系统被称为 UEFI 固件文件系统。本章首先对 UEFI 固件文件系统的整体架构进行描述，然后再结合 UEFI 固件文件系统对其各个组成部分进行详细地阐述。

3.1 固件文件系统结构

UEFI 联盟制定了一份平台初始化规范 (Platform Initialization Specification, PI 规范)，主要是对 UEFI 固件的实现架构、内核初始化流程、内核服务的实现方案以及 UEFI 固件设备文件在 ROM 芯片等存储器中的存储结构进行了一系列约定^[6]。这份规范的第三卷对 UEFI 固件设备文件如何在非易失性存储设备上存储进行了详细的规定。

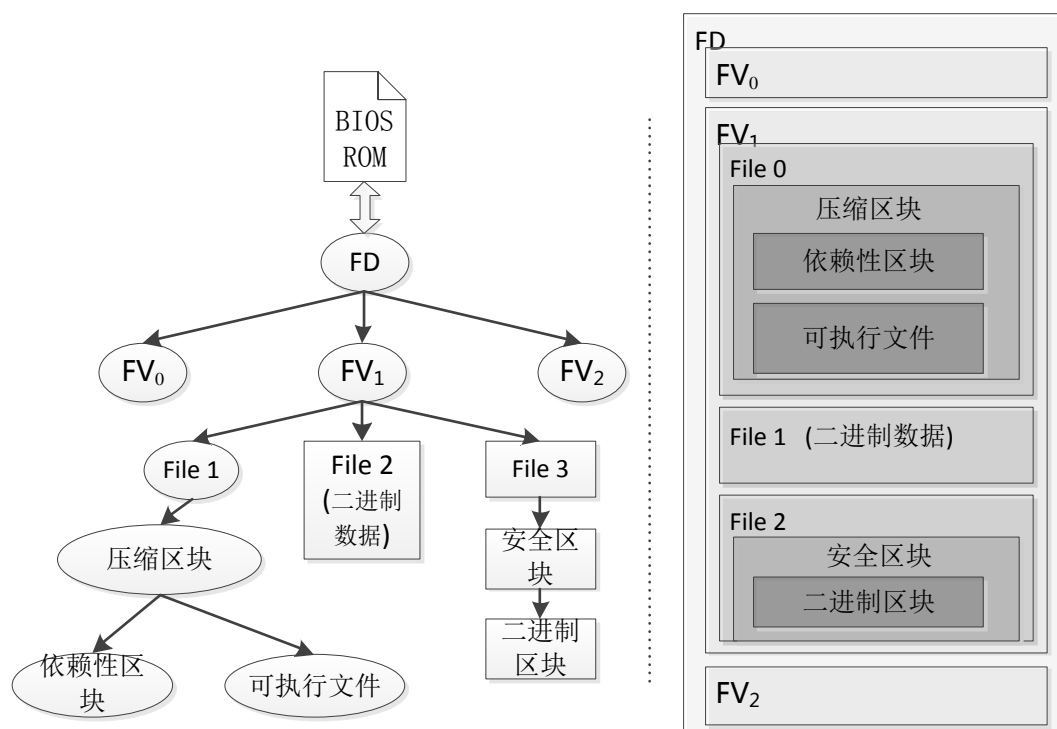


图 3-1 固件文件系统文件关系图

按照 PI 规范，UEFI 固件的实现必须支持标准的固件卷和固件文件系统格式，UEFI 固件文件系统整体上是一个层次结构，由上到下分别是固件设备(Firmware Device)、固件卷 (Firmware Volume)、固件文件 (Firmware Files) 和固件文件区块

(Firmware File Section)。从整体来看，固件设备更像是一个容器，里面依次存储着不同的固件卷，固件卷则由多个固件文件组成，最后的区块文件则是存储着要执行的代码和数据，各级文件关系如图 3-1 所示。

从图 3-1 中可以看出，整个 BIOS ROM 就是一个固件设备，一个 ROM 文件又被分成多个小的逻辑区块，即固件卷。每个固件卷都有固定的格式，主要包括固件卷头和固件卷实体。固件卷实体则由多个固件文件组成，如果固件卷有多余的空间，剩余空间将会被填充为默认数据。固件文件就是存储在固件卷上的数据或代码，一个固件文件通常由固件文件头和一个或多个固件文件区块组成。固件文件区块是相互独立的、不连续的段，每种类型的区块都存储特定的数据，比如 PE32 类型的区块存储的是代码，RAW 类型的区块则可能存储一些纯数据。固件文件区块的类型很多，有的区块甚至可以存储一整个压缩的固件卷。

3.2 固件设备

固件设备通常是指能够持久存储数据和代码的非易失性存储设备，最常见的固件设备就是 Flash 芯片，其它的还有硬盘、光盘、USB 等非易失性设备，采用 Flash 存储的 BIOS 芯片如图 3-2 所示，固件设备内存储的文件成为固件设备文件。通常情况下，一个单一的物理固件设备会被分为更小的部分以形成多个逻辑固件设备。



图 3-2 BIOS 芯片示例图

当前的计算机主板大多都是使用 Flash 芯片作为计算机固件设备，因为 Flash

芯片的良好特性能够充分满足固件文件系统的要求。比如，Flash 芯片能够按照一定的要求分成大小不同的逻辑快，并且能够按块或者扇区进行擦除，擦除后块内所有的位都能够恢复成默认值 0 或者 1^[35]。但是，虽然 Flash 芯片的易擦除和易重写的特性方便了固件开发厂商对产品的升级和维护，但是也给固件的安全性带来了一定的威胁。攻击者能够轻易地提取主板上 Flash 芯片内的固件设备文件，通过分析固件设备文件结构，通过改写或添加恶意代码的方式达到攻击目的。

3.3 固件卷

固件卷是在单个固件设备上划分出的并且连续的格式化后的块空间，可以看作是逻辑上的固件设备。一个固件设备可以是多个固件设备的集合，但是一个固件卷只能来自于一个固件设备。每个固件卷只占固件设备有效空间的一部分，它的位置再编译的时候由驱动来描述。按照固件文件系统的格式，如图 3-3 所示，每个固件卷由以下几部分组成：

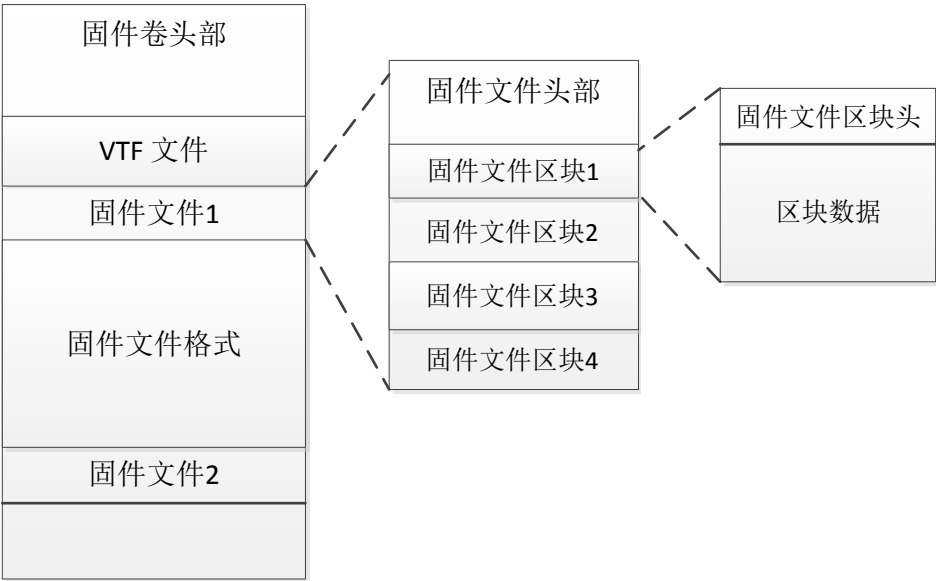


图 3-3 固件卷结构示意图

- 1. 固件卷首部。固件卷首部存储了固件卷的基本信息，包括固件卷的全局唯一标识符 (Globally Unique Identifier, GUID)、固件卷大小、固件卷签名及校验和等信息，其在 UEFI 规范中被具体定义为 EFI_FIRMWARE_VOLUME_HEADER。
- 2. 数据区。数据区即固件卷首部一下保存数据的区域，一般由一个或多个固件文件组成，固件文件在数据区中紧密排列存储，这样不仅节省了卷内的存储

空间，也保证了整个卷内的模块化存储。固件卷首部中并没有数据区中固件文件的位置信息，当固件卷需要定位某一个固件文件时，需要根据固件文件的 GUID 在整个固件卷进行遍历。

3. 一个 VTF (Volume Top File) 文件。VTF 文件也是一个固件文件，比较特殊的是 VTF 文件必须存储在每个固件卷的首部，并且每个 VTF 文件的 GUID 是固定的，在 UEFI 中被定义为 `EFI_FFS_VOLUME_TOP_FILE_GUID`，开发商可以直接进行引用。固件卷中其它的固件文件在进行插入和删除的时候必须要确认 VTF 文件位于正确的位置，文件对齐长度必须与 VTF 文件保持一致，否则会产生固件卷写入错误并导致固件卷不能被修改。

3.4 固件文件

固件文件是固件文件系统最重要的一个组成部分，固件中所有的驱动程序和应用程序必须封装成固件文件的形式才能在固件中存储^[20]。与大多数操作系统里的文件系统规则类似，以相同后缀名命名的文件会提供相似的功能，固件文件的统一后缀名为 `.ffs`，因此也可称作 FFS 文件，表示该类型文件表示提供一个驱动功能或应用功能。FFS 文件在固件卷里只有一级目录，是不会有子目录存在的。对 FFS 文件的通用操作有：检测固件卷以确定文件的类型、打开文件、读取文件、关闭文件等。

FFS 文件是固件卷中存储代码和数据的地方，一般由一个 FFS 文件头部和数据区组成，数据区一般存储着一个或者数个区块文件。它包含以下属性：

1. 名称：每个 FFS 文件都用一个 GUID 来作为唯一标识，一个固件卷中的 FFS 文件名称必须是唯一的，不能重复，但不同固件卷里可能包含相同文件名的文件。因此，在固件里检索文件时要以固件卷和文件名来进行配对。
2. 类型：FFS 文件总共有 4 类文件类型，每类又包含不同的文件类型，这 4 类类型即一般类型、OEM 类型、Debug 类型和固件卷定义类型。
3. 对齐长度：如果固件卷头部的对齐方式没有被设置为 `EFI_FVB2_WEAK_ALIGNMENT`，则该固件卷内的每个 FFS 文件的数据必须按照固件卷的格式和大小对齐。
4. 大小：每个 FFS 文件都有大小，包含 FFS 文件首部长度以及数据区内区块文件的长度。

3.4.1 固件文件类型

为了使 FFS 文件能够被固件卷服务接口统一识别，固件文件系统定义了一系

列的文件类型。FFS 文件的类型一般是由 FFS 文件的功能决定的，比如 EFI_FV_FILETYPE_DXE_CORE 类型表示 FFS 文件是 DXE 核心固件文件，通常是实现 DXE Foundation 的功能，而 EFI_FV_FILETYPE_DRIVER 表示的是驱动文件，DXE 阶段时 DXE 分发器会识别这类文件并且加载运行。FFS 文件的类型在 FFS 文件头部以一个字节来表示，具体的文件类型描述如表 3-1 所示^[6]：

表 3-1 固件文件类型及描述

固件文件类型	数值	文件类型描述
EFI_FV_FILETYPE_RAW	0x01	二进制数据文件
EFI_FV_FILETYPE_FREEFORM	0x02	段数据文件
EFI_FV_FILETYPE_SECURITY_CODE	0x03	SEC 阶段核心固件文件
EFI_FV_FILETYPE_PEI_CORE	0x04	PEI 阶段核心固件文件
EFI_FV_FILETYPE_DXE_CORE	0x05	DXE 阶段核心固件文件
EFI_FV_FILETYPE_PEIM	0x06	PEI 阶段的驱动文件
EFI_FV_FILETYPE_DRIVER	0x07	UEFI 驱动文件
EFI_FV_FILETYPE_COMBINED_PEIM_DRIVER	0x08	PEI 阶段和 DXE 阶段都可以执行的驱动文件
EFI_FV_FILETYPE_APPLICATION	0x09	应用程序文件
EFI_FV_FILETYPE_SMM	0x0A	加载到 SMRAM（系统管理内存）中的驱动文件
EFI_FV_FILETYPE_FIRMWARE_VOLUME_IMAGE	0x0B	固件卷文件
EFI_FV_FILETYPE_COMMBINED_SMM_DXE	0x0C	DXE 阶段和 SMM 阶段都可以执行的驱动文件
EFI_FV_FILETYPE_SMM_CORE	0x0D	SMM 阶段的核心固件文件

UEFI 平台在检索到固件文件后，会通过固件文件头部的类型数值获得该固件文件的类型信息，开发厂商在开发固件文件时，也要选择相应的固件文件类型进行编译。下面将对一些重要的文件类型进行说明：

1. EFI_FV_FILETYPE_APPLICATION

该类型的固件文件包含了一个或者多个 PE32 格式的映像文件，这些文件能够使用 UEFI 启动时服务 LoadImage()进行加载但不能被 DXE 分发器检索加载。EFI_FV_FILETYPE_APPLICATION 类型的固件文件必须至少包含一个 EFI_SECTION_PE32 类型的区块文件，并且该区块文件的封装没有任何限制。

2. EFI_FV_FILETYPE_DRIVER

此类固件文件是 DXE 阶段的驱动文件，在 DXE 阶段被 DXE 分发器识别并加载运行。该类文件至少要包含一个 EFI_SECTION_PE32 类型的区块文件。

3. EFI_FV_FILETYPE_PEIM

该类文件是 PEI 阶段的驱动文件，它的数据区一般由一个或多个 PEIM 组成。PEIM 是 PEI 执行阶段由 PEI 基础模块识别并加载的功能代码，用于完成 PEI 阶段的功能。EFI_FV_FILETYPE_PEIM 类型的固件文件必须包含并且只能包含一个区块文件。

4. EFI_FV_FILETYPE_COMBINED_PEIM_DRIVER

该类型的固件文件包含了在 PEI 阶段执行的代码，也包含了一个可以被 DXE 分发器加载运行的一个 PE32 映像文件，即该类型的文件在 PEI 阶段和 DXE 阶段都会被加载执行。这样不仅保证了 PEI 阶段和 DXE 阶段的代码共享，也减少了固件卷的存储空间，同时也可以将一系列 PEIM 打包放在同一个文件中。

5. EFI_FV_FILETYPE_PEI_CORE

该类文件是 PEI 基础模块实现的核心文件，按照 Framework 架构，该类型文件的执行代码在 UEFI 启动过程中要受到 SEC 阶段的完整性校验，主要功能是检索 PEIM 并且加载执行。

6. EFI_FV_FILETYPE_DXE_CORE

该类文件是 DXE 基础模块的核心实现文件，当 PEI 阶段执行结束后，该类型文件开始执行建立 DXE 基础服务，如 UEFI 启动时服务和运行时服务。该类文件只能包含 EFI_SECTION_PE32 类型的可执行区块。

7. EFI_FV_FILETYPE_RAW

此类文件不包含区块文件，而是作为原始数据文件来处理。使用这类文件必须要提前对其格式和内容充分了解，因为这类文件只是用于存储数据，对用户来说没有任何规则可用。

8. EFI_FV_FILETYPE_SECURITY_CODE

该类文件包含的数据和代码是由 SEC 阶段使用的，主要是对整个平台的安全性进行验证。该类文件的格式没有具体的定义，而是根据不同平台架构需求由 UEFI 开发厂商来决定。

3.4.2 固件文件格式

FFS 文件是 UEFI 平台功能实现的基本形式，在 UEFI 启动过程中，平台通过解析 FFS 文件的格式获得 FFS 文件的信息，然后加载执行。固件文件系统对 FFS

文件的格式进行了详细规定，文件大小小于 16Mb 的 FFS 文件格式如图 3-4 所示：



图 3-4 固件文件格式

从图 3-4 可以看出，所有的 FFS 文件都是以一个 FFS 文件头和文件数据区组成，文件的数据区一般包含一个或多个的区块文件。FFS 文件按照固件卷的对齐方式对齐，一般以 4 个字节对齐。文件大小大于 16Mb 的 FFS 文件在 FFS 文件头和数据区之间增加了 4 个字节的空间用于保存额外增加的数据大小。FFS 文件头由以下几部分组成：

1. 文件名称（Name）：该部分是由一个 8 字节的 GUID 组成，它是 FFS 文件的唯一标识。
2. 文件校验和（IntegrityCheck）：紧跟 GUID 的是 2 个字节的文件校验和，第一个字节是 FFS 文件头的检验和，第二个字节是整个 FFS 文件的校验和。
3. 文件类型（Type）：FFS 文件的类型用一个字节来表示。
4. 属性（Attributes）：一个字节的属性位用来表示 FFS 文件的属性，UEFI 规范定义的属性有 FFS_ATTRIB_LARGE_FILE、FFS_ATTRIB_FIXED、FFS_ATTRIB_DATA_ALIGNMENT、FFS_ATTRIB_CHECKSUM。
5. 大小（Size）：FFS 文件头中用 3 个字节来存储 FFS 文件大小，包括 FFS 文件头和数据区大小，该数值并不是按 4 字节对齐后大小而是文件的实际大小。
6. 状态（State）：一个字节的标识用于跟踪文件从创建到删除整个过程中的文件状态，如擦除错误，写入错误等。

3.5 区块文件与 EFI 文件

区块文件是一般存放在 FFS 文件数据区内，它与 EFI 文件关系密切。EFI 文件是 UEFI 映像文件的基本格式，大部分驱动文件和应用程序都是 EFI 可执行文件。EFI 可执行文件不能直接放入固件卷中，它们必须封装为 FFS 文件才可以在固件卷中存储。下面将分别介绍区块文件和 EFI 文件。

3.5.1 区块文件

在固件文件系统中，区块文件存储在 FFS 文件中，区块文件彼此之间互不相关。区块文件只有两个属性即文件类型和大小，因此区块文件在固件中的存储非常简单，也非常容易定位。

虽然区块文件有很多种类型，但其总体上分为两类，即封装区块和叶子区块。二者的区别是封装区块可以包含其它类型的区块包括叶子区块，而叶子区块只能包含特定的文件。如果将一个包含了很多区块的 FFS 文件看作一个树形结构，如图 3-5 所示，将 FFS 文件本身看作树的根节点，则封装区块则是 E_0 、 E_1 、 E_2 等节点，叶子区块则是 L_0 、 L_1 、 L_2 、 L_3 等叶子节点。从图中可以看出，封装区块可以和叶子区块在同一层进行存储，但是封装区块本身不能作为树的叶子节点存在，它可以包含封装区块，也可以包含叶子区块。

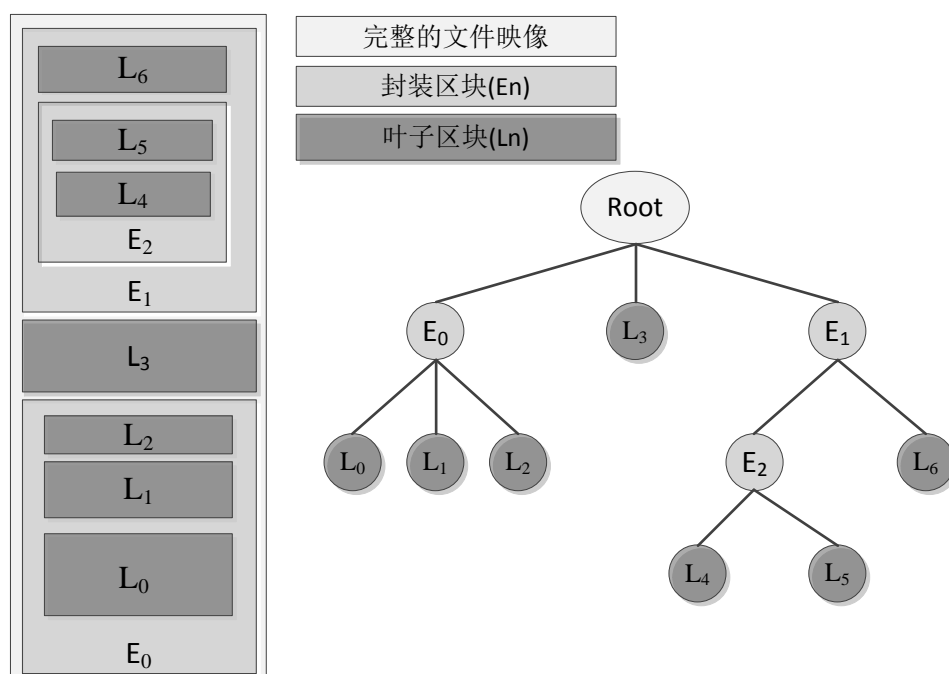


图 3-5 FFS 文件中区块文件的存储结构

UEFI 规范中描述了标准固件文件区块的存储细节，每个区块文件以区块首部

开始，后面存储不同类型的数据，比如 `EFI_SECTION_COMPRESSION` 类型的区块数据区往往存放另一个区块的压缩，`EFI_SECTION_PE32` 类型的区块数据区则存放 `PE32+` 类型的 `EFI` 文件。大小小于 16Mb 的区块的具体存储格式如图 3-6 所示：

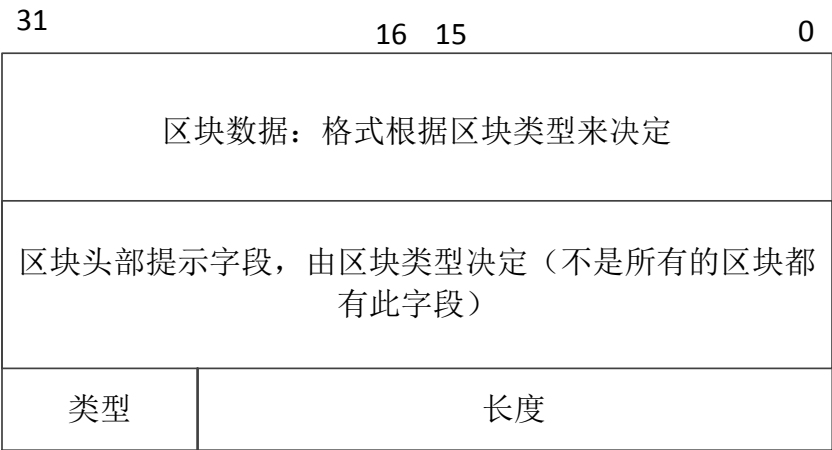


图 3-6 区块文件格式

从图 3-6 可以看出，区块首部只有两个内容，即 3 字节的文件长度以及一个字节的文件类型，之后是区块首部的提示内容，这部分由区块文件类型来具体定义。最后是区块的数据区，区块文件遵从固件卷的对齐方式，一般按 4 字节对齐。大于 16Mb 的区块文件只是在区块首部和数据区中间预留 4 个字节，用来存储文件多出 16Mb 的数据的大小。

区块文件是固件文件系统中最底层的部分，同时也可能存储另一个 `FFS` 文件的压缩。区块文件最常见的是存储着 `UEFI` 中的 `EFI` 可执行文件，用于完成 `UEFI` 启动的各项功能，但也常作为存储数据的文件。总之，区块文件是固件文件系统中底层也最复杂的文件，它的功能多种多样，具体的功能可以参照区块文件头的文件类型获得。

3.5.2 EFI 文件

3.5.2.1 EFI 文件格式

`EFI` 文件是 `UEFI` 映像文件的基本格式，是 `UEFI` 规范中的可执行文件，后缀名为 `.efi`。与 `Windows` 中的 `EXE` 等可执行文件类似，`EFI` 文件遵循微软的 `PE/COFF` 文件格式规范。不同的是，`EFI` 文件是 64 位的可执行文件，其具体的存储形式按照 `PE` 文件的扩展版本 `PE32+` 规范。`PE32+` 与 `PE` 文件的结构基本相同，只是删除了 `PE` 头部数据存储的基址，并把文件中的地址从 32 位扩展到 64 位。`PE32+` 文件的格式如图 3-7 所示：

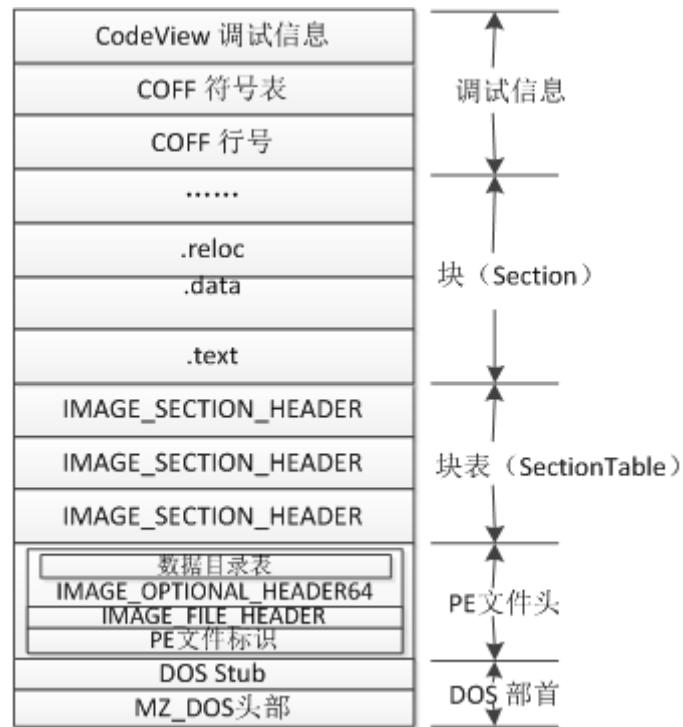


图 3-7 PE32+文件格式图

3.5.2.2 EFI 文件生成

UEFI 架构支持第三方的开发，各个厂商可以按照 UEFI 规范分别开发不同的 EFI 驱动程序和应用程序。Intel 提供了 EDK(EFI Developer Kit, EFI 开发工具包) 供 UEFI 开发者使用，现在常用的是 EDK 的改进项目 EDKII，EDKII 是 UEFI 基础框架的开源版本，允许开发、调试和测试 UEFI 驱动和应用程序。

EDKII 中提供了大量的库及协议供 UEFI 开发者使用，开发者只需按照 EDK 使用规则实现.h 源文件、.c 源文件及源文件信息文件.inf 文件，然后将这三者添加进 EDKII 中的平台文件中进行编译即可，EFI 文件的生成如图 3-8 所示：

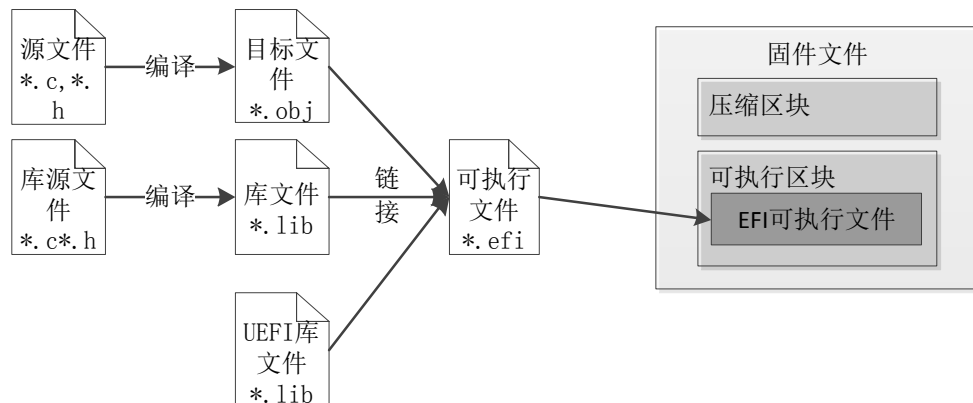


图 3-8 EFI 文件的生成过程

在编译成功之后，会在 EDKII 平台的编译文件夹内生成 EFI 文件。但是 EFI 文件不能直接在固件卷中存储，必须先将其封装为区块文件，然后将区块文件封装为 FFS 文件才能保存在固件卷中。EDKII 同时也提供了 EFI 文件的封装程序供开发者使用，如 GenSec.exe、GenFfs.exe、EfiRom.exe 等。UEFI 开发厂商使用这些工具将 EFI 文件封装为 FFS 文件，再将不同的 FFS 文件按照功能划分存储在不同的固件卷中，最后按照固件文件系统的格式将多个固件卷组装成一个 ROM 文件即可在固件设备内使用。

3.6 文件遍历和访问

在 UEFI 启动过程中，UEFI 平台要执行存储在固件卷中的各个 FFS 文件来实现硬件初始化，操作系统引导等功能，这就需要对文件进行遍历和访问。按照 UEFI 固件文件系统规定，在可写的 FFS 驱动初始化之前，SEC 阶段、PEI 阶段以及早期的 DXE 阶段代码必须能够遍历并且执行 FFS 文件。但是，由于之前可能由于断电或其它系统错误（比如一个固件卷中有两个文件具有同样的 GUID）导致 FFS 文件不一致，因此在使用 FFS 文件之前必须按照一组规则来对其进行初始化。SEC 阶段、PEI 阶段和早期的 DXE 只读 FFS 文件是没有权限对文件系统进行恢复或者修改的，如果由于文件损坏导致 UEFI 平台不能正常启动，UEFI 平台将启动恢复机制。

UEFI 提供了系统服务来确保平台对固件中文件的访问和执行。在 Framework 的 PEI 阶段，UEFI 平台通过 PEI 服务表获取固件文件相关的服务，常见的有 FFSFindNextFile 函数、FFSFindFileByName 函数和 FFSGetFileInfo 函数等。在 DXE 阶段，PEI 阶段中发现的文件会通过 HOB 产地给 DXE 分发器，分发器会在初始化自身的时候进行处理。当 DXE 分发器需要扫描新的文件时时，则使用 UEFI 提供的 EFI_FIRMWARE_VOLUME2_PROTOCOL 协议来操作 FFS 文件，常见的有 ReadFile 函数、ReadSection 函数、WriteFile 函数和 GetNextFile 函数等。FFS 文件的唯一标识是文件 GUID，DXE 分发器通过 GUID 对 FFS 文件进行遍历，检索在 DXE 阶段需要加载的驱动。DXE 分发器检索到所有 DXE 驱动的 FFS 文件后，其会搜索依赖性关系块，并使用这些信息为驱动设定一个加载顺序。

3.7 本章小结

本章研究了 UEFI 固件文件系统，对固件文件系统中的组成部分进行了详细描述。首先对固件文件系统的整体架构进行了分析，然后在该架构的基础上，分别对固件设备、固件卷、固件文件和区块文件的形式、类型及存储格式进行了描

述。区块文件的数据区最常存储的是 EFI 文件，本章对 EFI 文件的格式及生成方式也进行了描述，为后面对 EFI 文件进行操作提供基础知识，最后分析了 UEFI 平台对固件中的文件进行遍历和访问的方法。

第四章 基于固件文件系统的 UEFI 安全威胁研究

由于固件中的恶意代码具有权限高、隐蔽性强、难清除等特点，固件已经成为恶意代码攻击的重要目标。固件文件系统作为 UEFI 固件代码的组织 and 存储形式，在恶意代码的攻击过程中被频繁利用^[36]。本章将分析固件文件系统的安全性，并对基于固件文件系统的 UEFI 安全威胁进行研究。

4.1 固件文件系统安全性分析

随着针对固件的攻击方式的增多^[37]，UEFI 固件文件系统的安全性也显得尤为重要。由于 UEFI 固件文件系统的架构以及具体内容在 UEFI 联盟推出的 PI 规范中进行了详细地描述，所以攻击者能够通过阅读文档轻易获得固件设备文件的组织和存储方式，从而利用固件文件系统的弱点对固件进行攻击。本文从 UEFI 固件文件系统的角度，提出其存在的几个安全缺陷：

- 1、按照 UEFI 规范，UEFI 驱动以 FFS 文件的格式存储在固件卷中，并在 UEFI 启动过程中获得执行。但是固件文件系统并未对固件中 FFS 文件的添加和删除进行限制，攻击者可以通过 UEFI 开发环境编译然后结合译出包含恶意代码的 FFS 驱动文件，并将其插入固件卷中，然后结合固件可擦写的特性将修改后的文件写入固件。UEFI 没有判断 FFS 文件是否安全就进行加载执行，从而使伪造的驱动文件在 UEFI 启动过程中获得执行权限。文献[9]提出的插入 EFI Runtime Driver 攻击方法便是通过添加 FFS 文件的方式获得执行权限的。
- 2、固件卷中的 FFS 文件格式信息完全公开，使得 UEFI 的攻击者可以在绕过固件文件系统完整性保护的基础上任意修改 FFS 文件中的内容。比如，FFS 文件头中的文件校验和的含义及生成方式在 PI 规范中详细描述，攻击者只需在修改 FFS 文件内容后将 FFS 文件头内的文件大小、文件校验和等信息重新计算填写即可绕过 FFS 文件的完整性保护。本文中的基于 FFS 文件的攻击方法便利用 FFS 文件的这一特点，通过修改 FFS 文件头中的校验和绕过固件文件系统的完整性保护。
- 3、UEFI 映像文件采用的 EFI 格式实质上就是 Windows 中的 PE32+文件格式。由于 PE 文件的广泛应用和业界对 PE 文件格式的深入研究，针对 PE 文件的攻击方式如 PE 文件病毒等都比较成熟，这些成熟的攻击方法完全可以用到对 UEFI 映像文件的攻击当中。本文中的基于 FFS 文件的攻击方法将通过在 EFI 文件内添加区块，并修改 EFI 文件的执行入口来获得执行权限。

综上所述,虽然 UEFI 固件文件系统能够确保 UEFI 模块化开发、可扩展性好等优点,但其对固件文件的安全性并没有太多的考虑,导致文件系统存在着诸多的安全缺陷。因此,必须加强对 UEFI 固件文件系统的安全研究并完善其安全机制,否则会给整个 UEFI 平台的安全性留下安全隐患。

4.2 固件文件系统初始化

UEFI 固件文件系统提供了一套初始化机制来确保无论什么原因导致 FFS 文件发生崩溃,文件系统都能够及时检测到并及时修复^[38]。固件文件有三种基本操作:创建、更新和删除。在进行这三种文件操作时,必须严格遵守 UEFI 固件文件系统的规范,以免文件损坏。

文件更新通常用于对固件文件系统的修复,当固件卷中已存在的文件崩溃时,就必须创建新的文件,用新文件代替旧文件来修复崩溃文件。在文件更新的过程中,对固件来说任何时候新文件和旧文件都只有一个文件有效,旧文件中 FFS 头部的 `EFI_FILE_MARKED_FOR_UPDATE` 标志位被设置为有效状态。文件更新过程如图 4-1 所示:

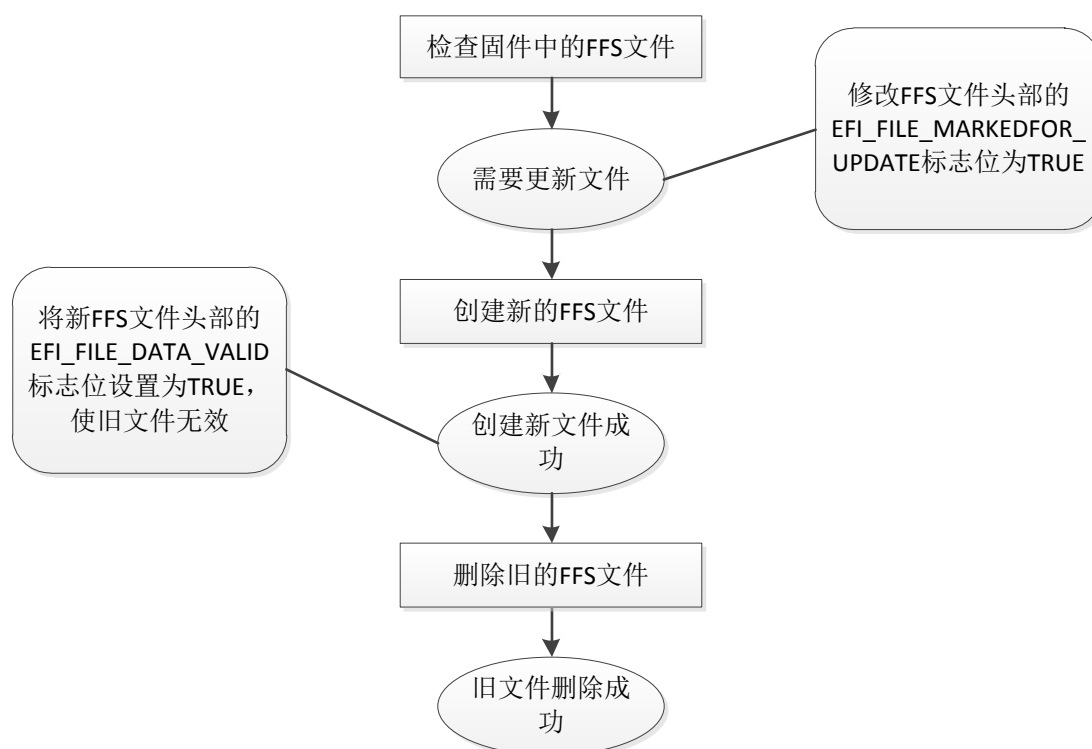


图 4-1 固件文件更新流程

FFS 文件的状态位用来确保 FFS 文件的完整性不会受到损害,比如在 FFS 文

件操作时发生断电时，状态位能够记录文件操作状态以备文件恢复。在初始化期间，如果某个故障导致固件卷崩溃，UEFI 的恢复机制将会启动。在固件文件系统初始化开始时，UEFI 将会执行一个 FvCheck 函数来检查固件卷是否损坏，FvCheck 代码执行流程如图 4-2 所示。

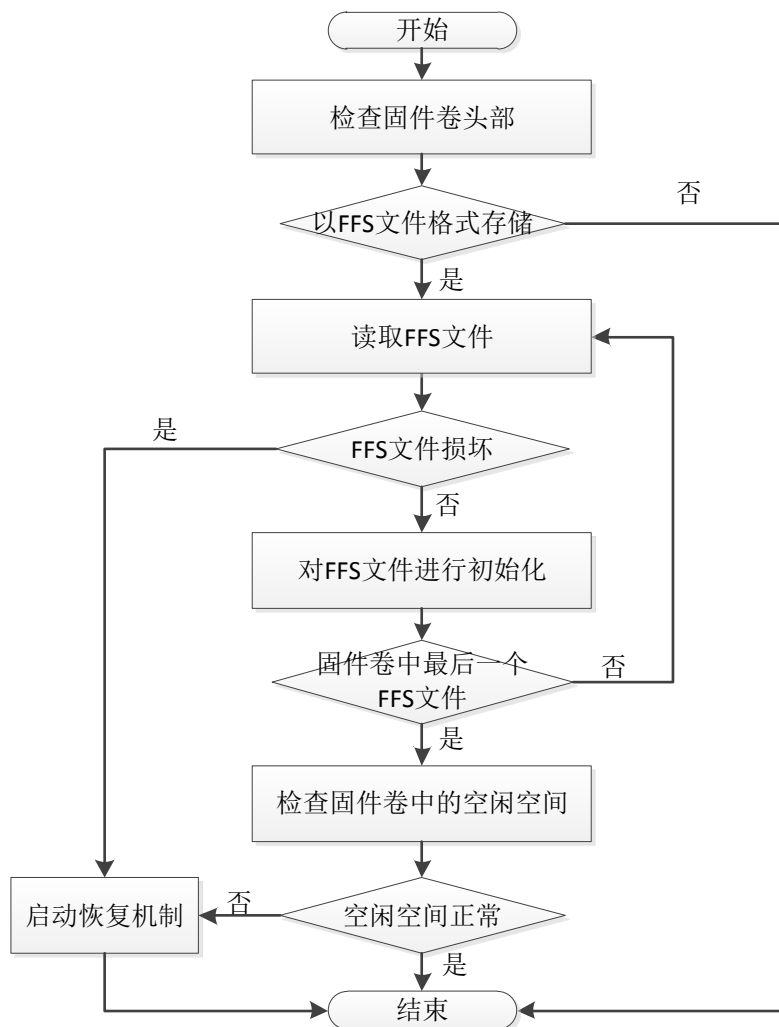


图 4-2 固件卷的初始化流程图

在 UEFI 启动过程中，UEFI 平台对固件中的文件进行频繁的文件操作。一旦文件操作失败，很容易导致文件损坏，文件系统必须及时检测到并修复文件。文件损坏的原因一般可以分为三类，即一般损坏、擦除损坏和写损坏。明显由于存储介质的随机损坏导致的文件损坏称为一般损坏，通常是由于存储媒体设备设计不当或者设备老化引起的。这种损坏是非常敏感的，文件中的一个位的改变都有可能导致文件损坏。因此，优秀的固件设备不仅要有良好的系统设计，而且还要有可靠的存储介质，这对避免发生一般文件损坏有非常好的效果。

擦除损坏是指由于文件正在擦除时发生了电源故障或者其它系统故障导致的

文件损坏。大多数 FFS 文件都会实现一个机制来回收已删除的文件并将空闲的空间进行合并，如果此类操作未完成也会导致文件损坏。

与擦除损坏类似的是，写损坏是指在写文件时发生故障导致的文件损坏，这类文件损坏会使文件系统处于一个不稳定的状态。

在固件文件系统初始化时，以上的文件损坏都会被检测到，并且根据文件损坏的性质，UEFI 会启动相应的恢复策略。FFS 文件的状态位顺序足够检测出所有的写损坏，但是要检测出所有的文件损坏，还必须结合 FFS 文件头中的完整性校验和进行确认。

4.3 基于固件文件系统的攻击方法研究

目前在计算机系统安全研究方面，基于底层固件的攻击方法已经成为研究的热门课题，由于固件是在操作系统之前启动的，相比操作系统层面的攻击，固件层的攻击有以下特点：

- 1、获得的权限高。由于固件是在操作系统启动前就已经获得了执行权限，如果固件层被攻击者攻破，攻击者将具有整个系统的最高权限，不仅可以控制操作系统的加载，也可以全程监控并操作计算机，窃取系统中的信息。
- 2、隐蔽性强。目前市面上的杀毒软件、恶意代码检测工具等安全产品都运行在操作系统层面下，即使能够监控到操作系统内核，也依然无法对固件中的恶意代码进行检测。基于固件的恶意代码攻击只有在固件中增加安全模块，实施恶意代码检测。
- 3、难以彻底清除。由于固件中的恶意代码一般都保存在主板上的 Flash 芯片内，一般基于操作系统的安全软件只能清除恶意代码在操作系统下产生的文件，无法对固件进行清理。要想彻底清除固件中的恶意代码，只有通过工具对固件芯片重新刷写才能实现。

由于固件层攻击具有以上这些特点，在 UEFI 成为新一代 BIOS 标准后，针对 UEFI 固件的攻击手段得到了广泛的研究。针对固件的攻击手段，基本上都不可避免地要利用固件文件系统的弱点，进行固件文件的操作，下面将介绍两种基于固件文件系统的 UEFI 攻击方法。

4.3.1 UEFI Bootkit 研究

4.3.1.1 Bootkit 工作原理分析

Bootkit 是指攻击代码在开机时比操作系统内核更早加载，实现内核劫持的技术^[39]。基于 UEFI 实现的 Bootkit 即在 UEFI 平台启动过程中，通过在 BIOS 芯片中

嵌入代码 HOOK UEFI OS Loader，在 OS Loader 加载完操作系统内核后，通过控制权的回收获得内核的控制权限。

UEFI 提供了一套启动时服务和运行时服务供 UEFI 开发者使用，ExitBootServices 便是启动时服务中的一个重要函数。基于 Framework 架构，TLS 阶段是操作系统引导的阶段，在此阶段，OS Loader 会加载并初始化操作系统内核，当操作系统启动完成时，OS Loader 调用启动时服务 ExitBootServices 来停止启动时服务并将控制权交给操作系统。因此，如果对 ExitBootServices 进行 HOOK，就可以在 OS Loader 加载操作系统内核之后再次获得控制权，从而成功获得操作系统内核执行权限^[40]。UEFI Bootkit 工作原理如图 4-3 所示：

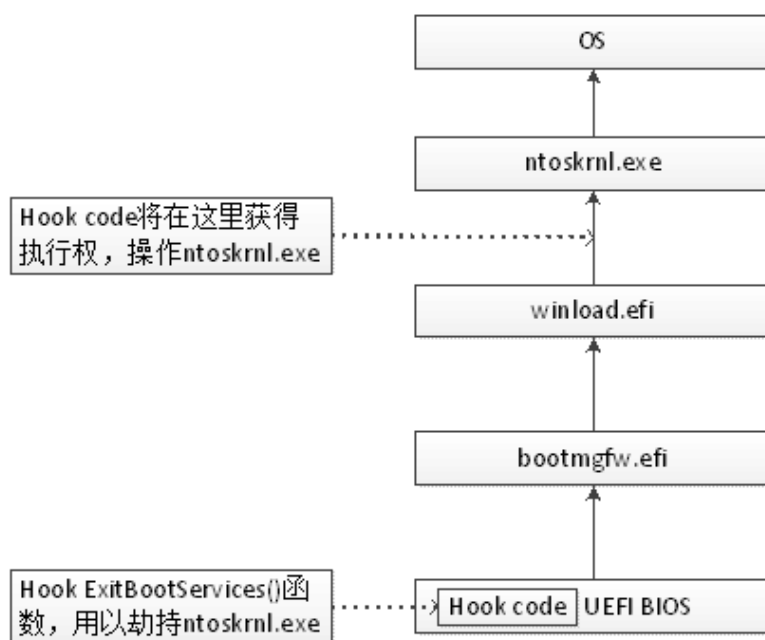


图 4-3 UEFI Bootkit 原理图

如图 4-3 所示，首先将 BootKit 代码添加到固件设备文件中，然后通过刷写固件的方式让其在 UEFI 启动过程中获得执行权限，其中 BootKit 代码完成对 ExitBootServices 函数的 Hook 工作。当 Hook 工作完成之后将控制权重新转交给正常的启动流程，使 UEFI 能够正常加载操作系统。以 Windows7 为例^[28]，UEFI 进入 TSL 阶段后，操作系统引导程序会调用 winload.efi 将内核程序 ntoskrnl.exe 装载到内存，然后 winload.efi 调用 ExitBootServices 函数停止 UEFI 启动，这时 Bootkit 代码通过之前的 Hook 重新获得了系统控制权，并可以利用内核进行下一步攻击。

Bootkit 方法还有很多，比如不直接 Hook 启动时服务 ExitBootServices 函数，而是在 UEFI 启动到操作系统加载的过程中，对 OS Loader、Winload 等文件进行一步步地 Hook，最终也可以达到劫持操作系统内核的目的，但此种方法操作比较

复杂，容易出错，效率也较低^[41]。另外一种方法就是直接攻击 OS Loader 文件，绕过 OS Loader 自校验机制，通过替换或者修改 OS Loader 的方法达到攻击目的^[28]。

4.3.1.2 Bootkit 代码启动方法

从 UEFI Bootkit 原理可知，要想实现 Hook UEFI 启动时服务 ExitBootServices 的功能，就必须让嵌入 BIOS 芯片里的 Bootkit 代码在 TLS 阶段调用 OS Loader 之前获得执行权限。按照 UEFI Framework 架构，DXE 阶段会搭建驱动执行环境，然后 DXE Dispatcher 会检索并加载 DXE 驱动到内存执行，但 UEFI 并未对 DXE 驱动进行安全校验。因此，只要将 Bootkit 代码实现为一个标准的 DXE Runtime 驱动（Runtime 驱动执行后会一直驻留在内存中，不会被清理），并且存放在固件中，在 DXE 阶段就会获得执行权限。

根据 UEFI 固件文件系统，UEFI 固件中的驱动程序以 FFS 文件的格式保存在固件卷中。将 Bootkit 代码实现为一个标准的 DXE Runtime 驱动（，并将该驱动的 EFI 可执行文件封装为 FFS 文件，然后将 FFS 文件插入固件卷中，从而在 UEFI 启动的 DXE 阶段中加载，获得执行权限，进而完成 HOOK UEFI OS Loader 的工作^[9]。插入 DXE Runtime 驱动后的 UEFI 启动流程如图 4-4 所示。

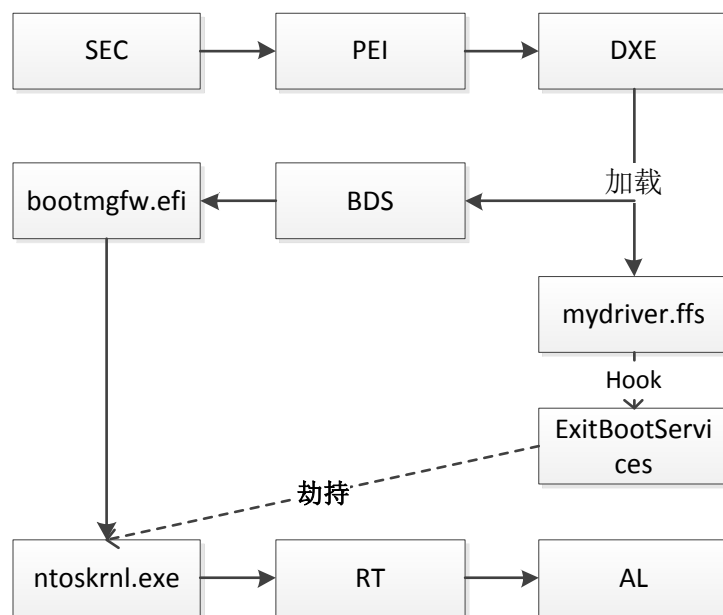


图 4-4 插入驱动后的 UEFI 启动流程

如图 4-4 所示，以 Win7 操作系统为例，当 UEFI 启动到 DXE 阶段时，需要加载 DXE 驱动完成初始化，因此 mydriver.ffs 作为 DXE runtime 驱动模块也被成功加载执行，此时 mydriver.ffs 完成 HOOK ExitBootServices 函数的功能。当 DXE 分发器加载完所有的 DXE 驱动之后，UEFI 启动进入 BDS 阶段，DXE 和 BDS 组

成的控制台将会选择并加载操作系统引导程序 `Bootmgfw.efi`。`Bootmgfw.efi` 进行加载操作系统内核 `ntoskrnl.exe` 工作，之后调用 `ExitBootServices` 函数结束 UEFI 启动服务，这时控制权回收到 `mydriver.ffs` 中，这时驱动中的代码可以操作内存中的操作系统内核，完成对操作系统内核的劫持。

FFS 文件模块的生成及添加是这种代码启动方法的关键。`Bootkit` 代码经过编译后得到驱动的 EFI 文件，此时需要结合区块文件及 FFS 文件的格式将 EFI 文件先封装为区块文件，再将区块文件封装为 FFS 文件，最后使用工具将 FFS 文件添加进固件卷。从固件文件系统的角度来看，固件卷及 FFS 文件格式的信息公开使得攻击者能够轻易地分析和添加 FFS 文件，并且由于固件文件系统并未对新加入的 FFS 文件进行安全分析，使得整个固件设备文件的安全性受到了威胁。另外，BIOS 芯片的可擦写特性虽然便于对 BIOS 的维护和升级，但也给攻击者将包含恶意代码的文件写入 BIOS 芯片提供了方法。

4.3.2 基于 FFS 文件的攻击方法

FFS 文件是 UEFI 固件功能实现的基本模块，包括 PEI 基础模块、电源管理等重要功能的实现模块都是以 FFS 文件格式存放在固件卷中。FFS 文件通常包含一个或者多个区块文件，其中最常见的是 EFI 文件的封装。由于 EFI 文件遵循 PE32+文件格式，因此可以通过修改 EFI 文件的执行入口获得代码执行权限。

基于 FFS 文件的攻击方法是通过修改 UEFI 启动过程中执行的 FFS 文件来获得执行权限。比如 `ACPI.ffs` 是 UEFI 中高级配置与电源接口的实现，其在 UEFI 启动过程中必定会得到执行。因此，可以按照 FFS 文件格式将 `ACPI.ffs` 中的可执行文件 `ACPI.efi` 文件提取出来并在其中加入恶意代码，使用 `LordPE` 工具打开 `ACP.efi` 文件，文件信息如图 4-5 所示：

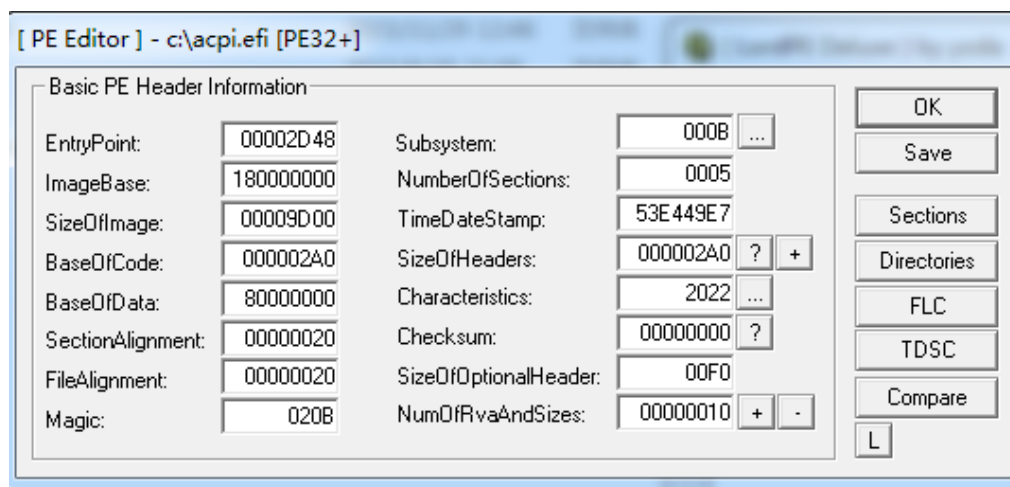


图 4-5 EFI 文件信息

如图 4-5 所示，使用 LordPE 等 PE 修改工具可以获得文件执行入口和区块信息。按照 PE 文件格式规范，并利用以上信息，可以在 ACPI.efi 文件中重新开辟区块，然后将恶意代码放入该区块，并将 ACPI.efi 文件的执行入口改为新开辟区块的首地址。在恶意代码结束的地方，需要将执行权限交给原入口地址以确保 UEFI 正常启动。将 ACPI.efi 文件修改结束后，需要按照区块文件和 FFS 文件的格式修改区块文件和 FFS 文件首部相关信息和校验值，再用修改后的 ACPI.ffs 文件替换原来的 ACPI.ffs 文件。具体的实现过程如图 4-6 所示：

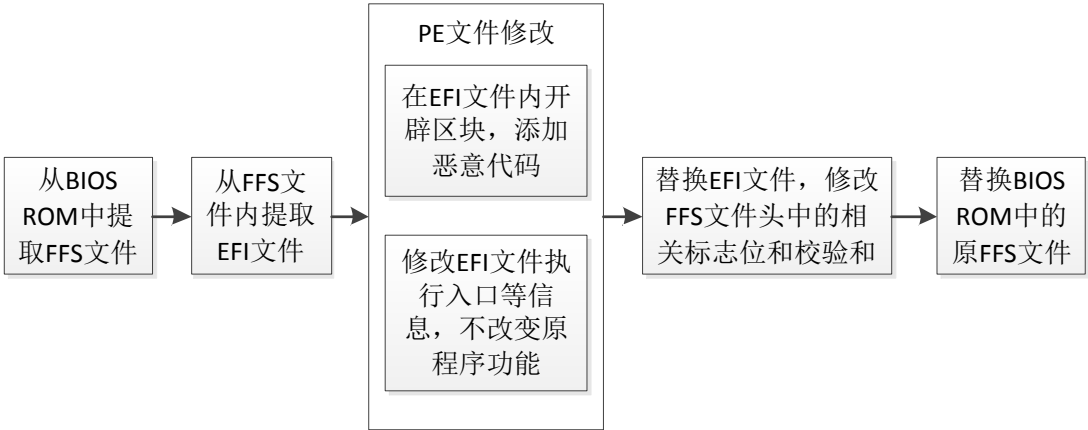


图 4-6 FFS 文件攻击过程

如图 4-6 所示，文件修改的目的是将恶意代码嵌入 UEFI 启动时的重要功能模块，利用 EFI 文件遵循 PE 文件格式的特点，通过开辟区块、修改执行入口的方法使嵌入的恶意代码获得执行权限。在基于 FFS 文件的攻击中，必须对固件文件系统、FFS 文件格式以及 PE 文件格式具有深入的了解，才能确保文件修改的方法不出差错。并且由于固件文件系统的完整性校验机制，必须要对 FFS 文件头的文件大小和校验值等信息进行修改，才能确保修改后的 FFS 文件有效，能够成功替换原始的 FFS 文件。

总之，固件文件系统及 FFS 文件格式等信息的公开使固件文件系统面临更加严峻的安全形势。随着对固件文件系统的研究不断深入，也必将有更多的攻击方式利用固件文件系统的缺陷威胁 UEFI 的安全。因此，必须加强对 UEFI 平台安全机制的研究，完善固件文件系统的安全机制。

4.4 本章小结

本章主要研究了 UEFI 平台基于固件文件系统的安全威胁。首先对 UEFI 固件文件系统的安全性进行了分析，并列出了其存在的几个安全隐患。然后对固件文

件系统的初始化过程和完整性保护方法进行了探讨。最后，本章介绍了两种基于固件文件系统的攻击方法，分析了 UEFI Bootkit 的原理和利用 FFS 文件在开机过程中获得执行权的方法，并介绍了基于 FFS 文件的攻击方法。总之，UEFI 固件文件系统仍存在诸多安全缺陷，需进一步完善其安全机制。

第五章 基于固件文件系统的 UEFI 安全方案研究

出于 UEFI 安全性的考虑，最新版本的 UEFI 规范也加入了关于可信启动、数字签名等安全服务的定义，用于 UEFI 固件平台的完整性校验和身份认证等^[42]。但是这些安全机制只能在 UEFI 启动过程中防止恶意代码伪装成第三程序入驻系统，不能阻止直接利用固件文件对 UEFI 进行的攻击^[43]。本章将基于可信计算的思想，设计基于 UEFI 启动的信任链和可信度量机制，提出基于固件文件系统的 UEFI 固件安全方案。

5.1 基于固件文件的可信计算研究

UEFI 固件是 UEFI 规范的具体实现形式，华硕、技嘉等主板厂商通常都以 Flash 芯片作为固件存储设备，并按照 UEFI 固件文件系统的格式存储固件代码和数据^[44]。根据固件文件系统架构可知，UEFI 的核心功能如 UEFI 服务、启动管理等以及重要的驱动程序均以固件文件，即 FFS 文件的形式存储在固件设备内。固件设备内的 FFS 文件示例如图 5-1 所示：

卷	索引	文件名	原始容量	标识
00	00		0001FFB8	CEF5B9A3-476D-497F-9FDC-E9814
01	00		000000D8	B52282EE-9B66-44B9-B1CF-7E504
01	01		0000102C	414D94AD-998D-47D2-BFCD-4E882
02	00		00000428	FD44820B-F1AB-41C0-AE4E-0C555
03	00	DummyMSOA	0000004F	DE498C70-1EDA-466B-ABCF-DD3AB
03	01		00008424	7BB28B99-61BB-11D5-9A5D-00902
04	00	IntelSaGopDriver	00000C76	5C266089-E103-4D43-9AB5-12D70
04	01	IntelGopDriver	0000BABE	5BBA83E6-F027-4CA7-BFD0-16358
04	02		00005018	17088572-377F-44EF-8F4E-B09FF
04	03	FvOnFv2Thunk	0000056B	5007A40E-A5E0-44F7-86AE-662F9
04	04	CpuDxe	000026D8	E03ABADF-E536-4E88-B3A0-B77F7
04	05	FileSystem	000047CA	93022F8C-1F09-47EF-BBB2-5814F
04	06		0000883A	DAC2B117-B5FB-4964-A312-0DC7
04	07		000003B3	9221315B-30BB-46B5-813E-1B1BF
04	08	CORE_DXE	00029982	5AE3F37E-4EAE-41AE-8240-35465
04	09	Runtime	000099CA	CBC59C4A-383A-41EB-A8EE-4498A
04	0A	OCMR_CPU_POWER_MANAGEMENT_DXE	0000135F	86F61BDF-5BFD-46D3-B0F9-E4372
04	0B	AsusPTTDxe	00001502	17689034-F11B-468B-8CC4-E114C
04	0C		0000002E	00F026EF-6E70-4764-AA82-37A47

图 5-1 固件设备文件示例图

根据第二章对 UEFI Framework 架构的研究可知，UEFI 启动过程被分为 7 个阶段，每个阶段都有核心服务和驱动程序加载，这些核心服务和驱动程序以 FFS 文件的形式存储在固件卷中，如 DXE 阶段的核心功能通常实现为 CORE_DXE.ffs 文件，CPU 初始化的功能实现为 CpuInitPei.ffs 文件。因此，本文通过保护 UEFI 固

件设备中的各阶段的核心 FFS 文件和驱动 FFS 文件来防止攻击者通过修改 FFS 文件获得执行权限。

固件设备文件主要由代码和数据组成。UEFI 各阶段核心 FFS 文件和驱动 FFS 文件在开机过程中都会获得执行权限，它们不仅包含文件执行时需要的数据，也包含有一个或多个 EFI 可执行文件。在 UEFI 启动过程中，FFS 文件被加载，EFI 可执行文件便获得执行权限。每个 FFS 文件都有一个 16 个字节的 GUID，GUID 是文件的唯一标识符。为了防止攻击者通过添加新的驱动 FFS 文件获得执行权限，本文不仅对 FFS 文件进行完整性校验，还会比对 FFS 文件的 GUID 来判断是否有新的 FFS 文件获得执行权限。

5.2 UEFI 的信任链设计

5.2.1 UEFI 信任链的构建基础

可信计算技术是通过建立一条信任链，并对信任链建立一个可信度量机制来对计算机系统进行保护。建立基于 UEFI 的可信计算平台，关键是建立信任链。通过构建信任链可以在 UEFI 启动过程中传递系统控制权的的同时不断传递信任，从而保证整个 UEFI 启动的可信。

TCG 提出的对计算机系统保护的信任链为：BIOS Boot Block->BIOS->OS Loader->OS，该信任链以计算机开机启动的过程为基础，逐级进行保护^[45]，其建立基础在于计算机启动过程中控制权在信任链节点之间的传递。因此，建立基于 UEFI 的信任链，必须从 UEFI 固件的功能和控制权在 UEFI 中的传递过程来入手。

计算机系统自加电以后便进入 UEFI 固件执行固件代码，UEFI 主要完成计算机自检、硬件初始化及引导操作系统等功能。按照 Framework 架构，UEFI 启动过程分为 SEC 阶段、PEI 阶段、DXE 阶段、BDS 阶段、TSL 阶段、RT 阶段和 AL 阶段等 7 个阶段，计算机进入 UEFI 固件执行后，系统控制权将在这 7 个阶段之间进行传递。

本文根据 TCG 提出的计算机系统信任链，以 7 个启动阶段为基础，在 UEFI BIOS 启动过程中建立信任链。通过分析各个阶段完成的功能，研究各阶段之间控制权的传递过程，从而在各阶段之间建立信任关系。

5.2.2 可信度量根的确立

按照可信计算的基本思想，建立信任链首先要确立一个可信度量根，然后从可信度量根开始建立一条信任链，逐级扩展。可信度量根必须是可信的，其可信性由物理安全、技术安全和管理安全共同确保，是整个信任链可信的基础，也是

整个可信计算平台可信的保障^[46]。

建立 UEFI 的信任链，首先要确立在其启动过程中的可信阶段，并将其作为信任链的可信度量根。SEC 阶段是平台加电后进入的第一个阶段，主要进行加电自检，验证硬件信息。同时，SEC 阶段会对早期的初始化代码进行验证，并使用哈希函数对 PEI 核心代码进行校验，保证其执行代码的可信。

SEC 阶段被默认为绝对安全的，其安全性由 UEFI 固件生产厂商保障，厂商生产 UEFI 固件时会预留接口，确保处理器执行的第一条指令是完全可信的^[47]。PEI 阶段主要进行 CPU、内存和主板的初始化工作并建立 C 语言的执行环境，其核心代码的安全性在 SEC 阶段已被检验，因此 PEI 阶段也是可信的。综上所述，SEC 阶段和 PEI 阶段的核心代码安全可信，可以作为 UEFI 启动过程的可信度量根。

5.2.3 基于 UEFI 启动过程的信任链设计

确立可信度量根之后，则要从可信度量根出发，建立一条基于 UEFI 启动过程的信任链。按照 UEFI 启动过程，在 PEI 阶段之后，DXE 阶段搭建 UEFI 基础服务和驱动执行环境，并检索固件卷内的所有 DXE 驱动进行加载。DXE 阶段最后与 BDS 阶段共同组成控制台，管理 UEFI 驱动和应用程序的加载。BDS 阶段执行完毕后进入 TSL 阶段，此阶段中 OS Loader 加载操作系统内核，并调用相关服务退出 UEFI 启动，将系统控制权交给操作系统，之后便进入操作系统运行阶段。因此，可以建立如图 5-2 示的信任链。

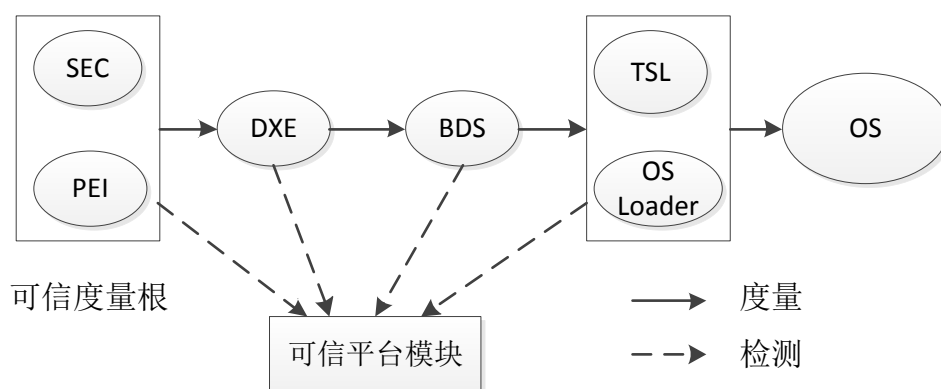


图 5-2 UEFI 的信任链设计

该信任链以 SEC 阶段和 PEI 阶段代码为可信根，进入 PEI 阶段初始化完毕后，会首先对 PEI 阶段的驱动程序进行可信度量，然后对 DXE 阶段代码及 DXE 阶段加载的驱动程序进行可信度量，若 DXE 阶段可信则对 BDS 阶段代码和驱动程序进行检验，依次类推，对 TSL 阶段和 OS Loader 进行验证，直到进入操作系统运

行阶段，UEFI 启动过程结束，整个基于 UEFI 的信任链就完整建立，信任关系也从 UEFI 传递到操作系统。

5.3 可信度量方法

可信度量方法是指在信任链传递的过程中判断信任链节点是否可信的方法，经常采用密码学中的完整性度量、数字签名认证等机制。本文经过分析，采用完整性度量机制作为 UEFI 信任链的可信度量方法，并对 FFS 文件的 GUID 进行检查。

完整性度量是现在主流的可信度量方法，它通过 Hash 函数或者消息验证码对平台模块进行完整性校验，从而保证模块的完整性。本文采用安全散列算法 SHA-1 来实现完整性度量机制。SHA-1 算法是利用一个散列函数对长度小于 2^{64} 位的消息进行运算，产生一个 160 位的消息摘要，然后用消息摘要验证消息的完整性。

GUID 是 FFS 文件的全局唯一标识符，每个 FFS 文件都以一个 GUID 来标识，固件文件系统通过 GUID 对 FFS 文件进行检索和访问。本文通过对 FFS 文件的 GUID 进行检查，防止攻击者通过添加 FFS 文件的方式获得执行权限。

可信度量方法分为可信度量计算、可信度量存储、可信度量日志和可信度量验证。基于 UEFI 的可信度量方法主要是对每个阶段的 FFS 文件进行可信度量，每个阶段的可信度量过程为：遍历该阶段的核心 FFS 文件和驱动 FFS 文件，并对每个 FFS 文件进行如下图 5-3 所示的可信度量。

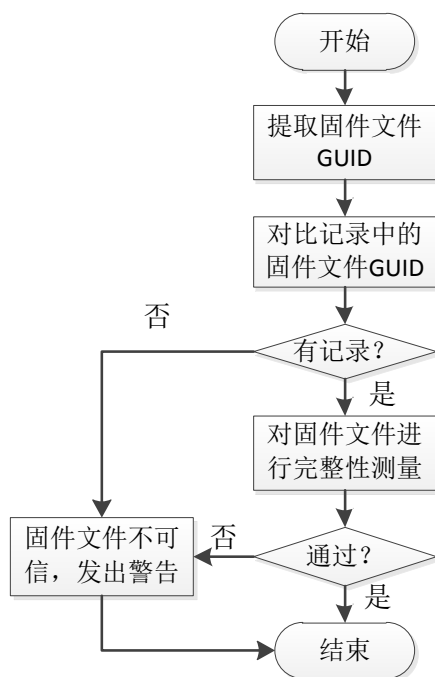


图 5-3 FFS 文件的可信度量过程

如图 5-3 所示，首先提取 FFS 文件的 GUID，并与可信度量存储模块内的初始 GUID 进行对比，若没有该记录，则认为该 FFS 文件是由第三方添加的，文件内容不可信。若有记录，则认为是固件内的原有 FFS 文件，并对其进行 Hash 计算，将 Hash 计算结果和文件的 GUID 生成可信度量日志发送给可信度量验证模块进行验证。可信度量验证模块将日志中的散列值与可信存储模块内的该 GUID 对应的散列值进行比对，若一致则认为未遭到修改，若不一致则认为该文件不可信。

根据 UEFI 信任链的设计，可信度量主要完成对 SEC 阶段和 PEI 阶段、DXE 阶段、BDS 阶段、TSL 阶段、各阶段驱动程序以及 OS Loader 的完整性测量。完整性测量在固件安全模块内部进行，每对一个 FFS 文件完成可信度量后，固件安全模块都会生成完整性度量日志，并对度量结果进行验证。整个可信度量过程中，一旦有一个 FFS 文件不可信，将导致整个 UEFI 启动过程的不可信，可信度量机制将发出错误信息并终止启动，从而保证整个平台的安全性。

5.4 UEFI 固件安全方案设计与分析

5.4.1 固件安全方案设计

结合 UEFI 信任链以及完整性度量方案，本文将从 FFS 文件的角度出发，提出 UEFI 整体的安全方案。该安全方案主要包括方案的实现形式以及安全模块的启动点、信任链节点的选择、安全方案的流程。

1、实现形式

UEFI 启动的 PEI、DXE 和 BDS 阶段都有驱动程序加载执行，按照固件文件系统，这些驱动程序大部分以 FFS 文件的格式储存在固件卷中，并随着 UEFI 的启动加载运行。本文采用 UEFI 驱动程序的形式模拟 TPM 芯片的功能，实现一个固件安全模块来验证安全方案的功能。该驱动程序的 EFI 可执行文件主要实现安全方案的功能，但必须需要封装成 FFS 文件才能插入固件卷运行。固件安全模块即驱动程序封装后的 FFS 文件，本文将固件安全模块插入固件卷中，保证固件安全模块在 UEFI 启动过程中执行，第六章将对其进行具体的设计和实现。

2、安全模块的启动点选择

固件安全模块在 UEFI 启动过程中执行，启动各阶段的可信度量工作也均在 UEFI 启动过程中完成。本文将固件安全模块的可执行文件实现为 PEI 驱动，从而在 UEFI 启动到 PEI 阶段时获得执行权限，即固件安全模块在 PEI 阶段启动并初始化。固件安全模块初始化后一直运行，由于信任链节点之间的传递过程会造成文件的变化，因此只在进入下一个信任链节点之前固件安全模块才会启动度量机制。

3、信任链节点的选择

由 5.2 这一节的内容可以看出，固件安全方案将 UEFI 启动阶段进行划分，主要设置了三个可信度量点，即 DXE 阶段、BDS 阶段和 TSL 阶段。这三个阶段是 UEFI 启动的核心阶段，由于需要加载大量 UEFI 实体文件，攻击者通常利用这三个阶段对 UEFI 固件进行攻击。由于 SEC 阶段和 PEI 阶段是可信度量根，不再对其核心代码进行可信度量，而 TSL 阶段之后计算机系统进入操作系统控制阶段，不属于 UEFI 启动过程，本文也不再对后面过程进行可信度量。

4、安全方案的流程

UEFI 固件安全方案根据 UEFI 的信任链，结合可信度量机制对各个信任节点进行可信度量，最终实现对整个 UEFI 启动过程的可信度量，整体方案如图 5-4 所示：

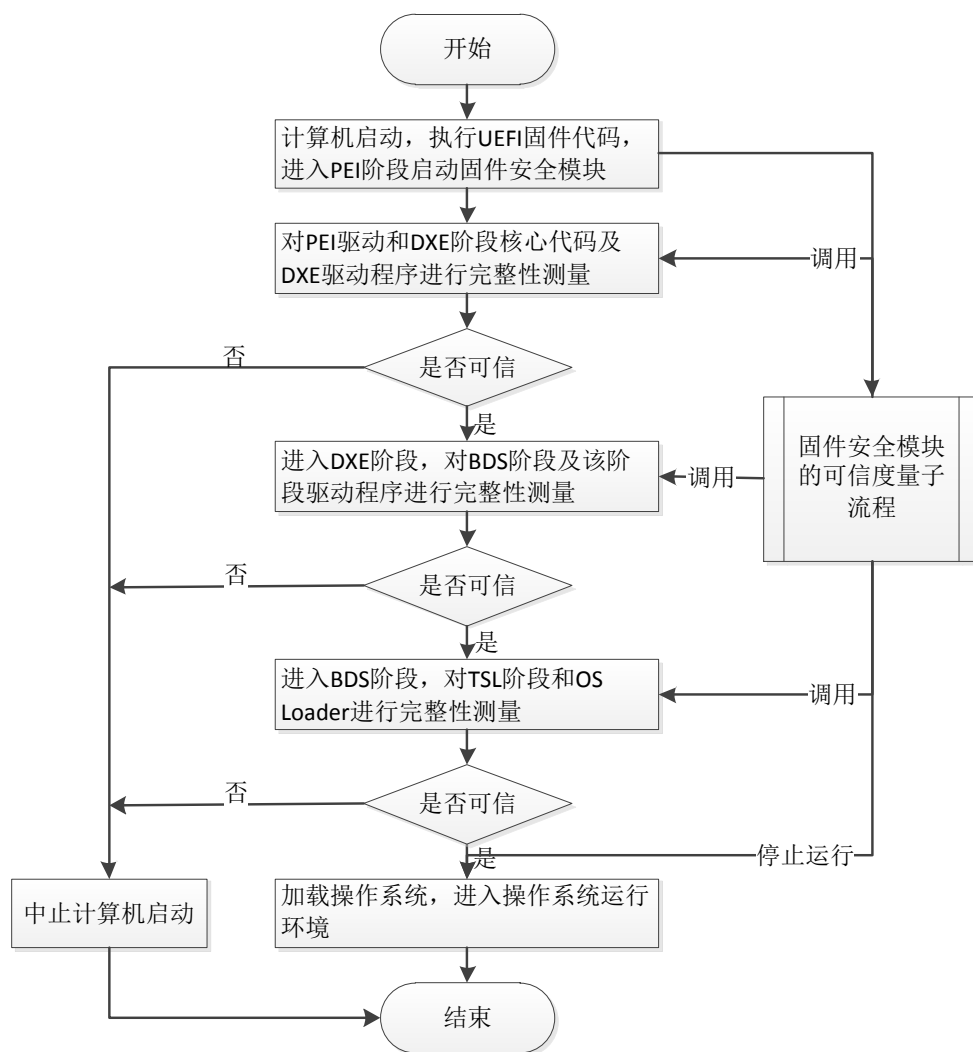


图 5-4 UEFI 固件安全方案流程图

按照 UEFI 启动过程，SEC 阶段和 PEI 阶段作为信任链的可信根，首先获得执

行。SEC 阶段上电自检、初始化硬件，然后对 PEI 阶段代码进行校验，然后便进入 PEI 阶段执行。PEI 阶段是 UEFI 实现的核心部分，通过加载执行 PEI 驱动来进行内存的完全初始化。固件安全模块作为一个 PEI 驱动，初始化工作也在这个阶段完成，并对 DXE 阶段进行了可信度量。

进入 DXE 阶段后，在控制权转交给 DXE 阶段之前，固件安全模块需对 DXE 阶段执行代码进行完整性度量。DXE 阶段建立 UEFI 基础服务和驱动执行环境，并且分发和调度驱动程序执行，DXE 阶段核心模块和驱动程序模块均以 FFS 文件格式存储在同一个固件卷内。

固件安全模块首先对该固件卷内的 FFS 文件进行遍历，并提取出 FFS 文件的 GUID 与记录文件中的 GUID 进行比对，若 GUID 不能一一对应，则认为该固件卷遭到了破坏，该阶段不可信。由于 DXE 阶段驱动需要 DXE 核心代码建立起驱动执行环境之后才能加载运行，因此先对 DXE 阶段核心 FFS 文件进行完整性度量，若通过完整性度量则对 DXE 阶段加载的驱动文件进行完整性度量，将度量值存储在固件安全模块存储区内，并将完整性度量过程用日志记录下来。对 DXE 阶段的安全检测流程如图 5-5 所示：

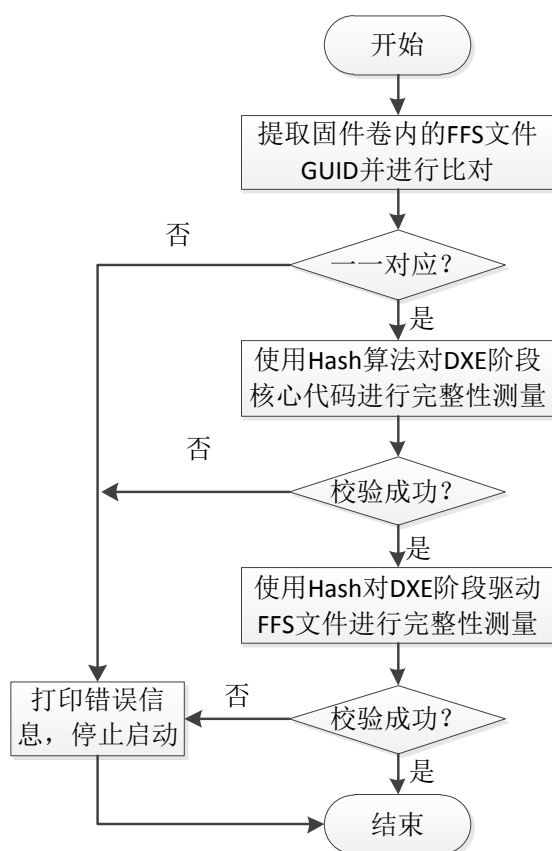


图 5-5 DXE 阶段的可信测量流程

BDS 阶段建立控制台并提供了 UEFI Shell 环境，此阶段 UEFI 平台可以从外部设备加载 UEFI 驱动和应用程序，固件安全模块在 BDS 阶段代码执行前对其核心代码进行完整性度量。

通过 UEFI 信任链可知，BDS 阶段之后是 OS Loader 加载操作系统的阶段，在 OS Loader 接收控制权之前必须对其进行完整性度量。按照 UEFI 规范，OS Loader 其实是一个 UEFI 应用程序，由于其担负着引导操作系统的重要作用，常常成为恶意代码攻击的首选目标，常见的针对 OS Loader 的攻击便有文件感染、替换等方式。因此不仅要加载 OS Loader 的代码进行完整性度量，也要对 OS Loader 文件本身进行完整性校验。

5.4.2 固件安全方案分析

固件安全方案通过对 UEFI 启动过程的各阶段建立信任链，并对信任链的各节点进行可信度量从而来确保整个启动过程的安全可信。下面将从安全方案的功能有效性和对正常启动的影响两方面进行分析。

1、安全方案的有效性分析

本文根据第四章的两种基于固件文件系统的攻击方法的原理，分析固件安全方案的功能的有效性。

从 UEFI Bootkit 原理可知，要想实现 Hook UEFI 启动时服务 ExitBootServices 的功能，就必须让嵌入 BIOS 芯片里的 Bootkit 代码在 TLS 阶段调用 OS Loader 之前获得执行权限。Bootkit 代码启动方式是通过向固件设备文件内插入 EFI Runtime Driver，在 DXE 阶段检索 DXE 驱动时加载包含 Bootkit 代码的驱动执行，从而在 UEFI 启动过程中获得执行权限。其中，插入的 EFI Runtime Driver 采用 FFS 文件的格式，由攻击者将包含 Bootkit 代码的 EFI 可执行文件封装成 FFS 文件。根据安全方案的可信度量机制，在进入 DXE 阶段之前，固件安全模块要对 DXE 阶段核心 FFS 文件和驱动 FFS 文件的 GUID 进行检查。由于 Bootkit 代码所在的 FFS 文件是由攻击者生成的，其 GUID 在固件安全模块的存储区内必然没有记录，按照固件安全方案的可信度量方法，固件安全模块在比对文件 GUID 的时候便能检测出该攻击方式。

基于 FFS 文件的攻击方法是通过修改 UEFI 启动过程中执行的 FFS 文件来获得执行权限。固件内的许多 FFS 文件比如 DXE 核心 FFS 文件、电源管理模块 FFS 文件等在 UEFI 启动过程中都会执行，这类 FFS 文件的区块文件都包含有一个或多个 EFI 可执行文件。攻击者通过对 EFI 文件进行增加区块、修改执行入口的方式将恶意代码加入 EFI 文件进行执行。由于固件文件系统自身完整性机制是将完整

性测量结果与 FFS 文件头的校验结果比较，攻击者可以通过修改 FFS 文件校验和的方式通过固件文件系统的完整性校验。但是固件安全方案通过对 FFS 文件进行完整性测量，并将测量结果与存储区内原始信息进行对比。由于 FFS 文件内容被攻击者修改，其完整性测量必然改变，这也必然会被固件安全模块检测到。

2、对正常启动的影响分析

固件安全方案的实现形式是一个驱动程序的 FFS 文件模块，在 UEFI 启动过程中启动并对固件进行保护。由于其在 UEFI 启动的过程中需要频繁地访问固件中的 FFS 文件，并进行大量的 Hash 计算，这个过程占用了计算机大量的资源。同时，由于在 UEFI 启动的每个阶段都要进行可信度量，增加了 UEFI 启动的时间。总之，UEFI 固件安全方案保证了启动的可信性，降低了整个计算机系统启动的效率。

综上所述，UEFI 固件安全方案基于可信计算的原理，在 UEFI 启动各阶段建立信任链，并对各阶段核心模块和驱动模块的 FFS 文件进行可信度量。固件安全模块将完整性度量值存储在开辟的内存空间中，并生成可信度量日志，用于验证各阶段的可信性。总之，UEFI 固件安全方案从 FFS 文件的角度进行可信度量，保证了 UEFI 固件整体的安全性。但是由于该方案的实现在 UEFI 启动阶段，其大量的计算和文件操作降低了 UEFI 启动的效率。

5.5 本章小结

本章通过对可信计算技术的研究，设计了基于可信计算的 UEFI 安全方案。首先对 UEFI 固件设备文件和可信计算的原理的结合点进行研究，重点分析了 UEFI 启动过程各阶段的核心模块在固件设备文件中的实现。然后根据可信计算的原理，结合 UEFI 的启动过程，设计出基于 UEFI 启动过程的信任链。本章提出使用 SHA-1 算法作为完整性度量方法，通过对比固件中 FFS 文件的 GUID 和散列值来确认其是否可信。本章依据 UEFI 的信任链以及可信度量方法，提出了 UEFI 固件平台的整体安全方案并对其进行分析，该方案从 FFS 文件的角度，对各阶段核心模块和驱动模块的 FFS 文件进行可信度量，弥补了固件文件系统的缺点，加强了对 UEFI 固件的保护。

第六章 UEFI 固件安全方案的实现与验证

6.1 EDKII 开发环境介绍

EDKII (EFI Development Kit) 是目前使用最广泛的 UEFI 开发平台, 其提供了大量的类库和 API 函数供开发者使用, 也提供了一系列的开发实例供 UEFI 学习者参考。借助 EDKII 的开源框架, 开发者可以非常便利地进行 UEFI 驱动程序和应用程序的开发和调试, 也可以利用其中的工具包仿真生成 UEFI 固件文件。

EDKII 从 EDK 发展而来, EDKII 提供了更多的类库, 并且扩展了 PCD 机制。EDKII 开发平台常用的开发包有 BaseTools、Build、IntelFrameworkModulePkg、MdeModulePkg、MdePkg 等^[48], 如图 6-1 所示:

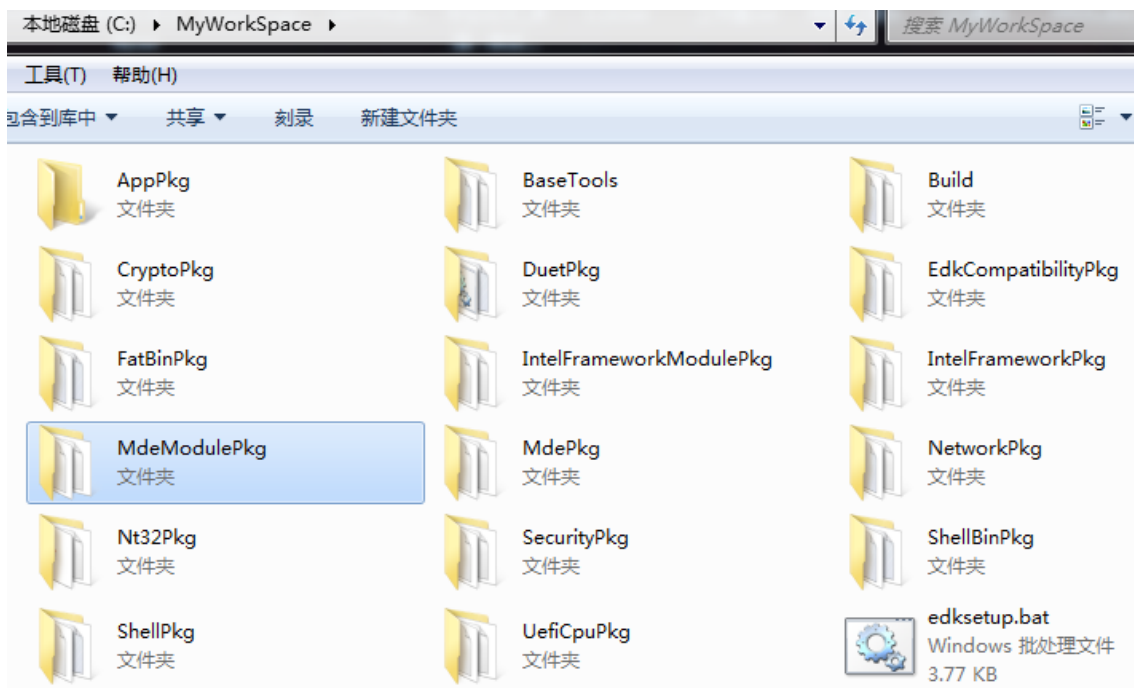


图 6-1 EDKII 开发环境

下面将简要介绍重要开发包的作用:

BaseTools: 存放编译环境的配置文件、编译工具以及一系列 UEFI 映像文件生成工具, 常用的有 EfiSec.exe、EfiFfs.exe、EfiRom.exe。

Build: 用于存放各个平台生成的可执行文件、编译文件及中间文件, 本文开发的驱动程序的 EFI 文件即在此目录下面。

IntelFrameworkModulePkg: 包含了 Intel 提供的 UEFI Framework 框架的模块实现的源码, 如固件卷的实现、控制台的实现等。

MdeModulePkg: 包含了 UEFI 底层应用模块、UEFI 内核以及标准服务接口实现的实例。它将系统底层函数封装成一系列 UEFI 开发库, 从而使用户能够方便地进行自定义模块开发, 本文所实现的驱动模块便是在此目录下进行。

MdePkg: 提供了 UEFI 规范、PI 规范等标准和 UEFI 协议的声明文件以及所有通用的底层库函数。

EdkCompatibilityPkg: EDK 代码库, 保证该版本的代码能够兼容以往的 EDK 版本和传统 BIOS。

在 EDKII 环境中, 开发驱动程序和应用程序首先要在相关的开发包中建立一个文件夹作为驱动开发环境, 文件夹中包含驱动的源文件和配置文件, 然后编译整个开发包, 在 Build 目录下即可得到驱动程序的 EFI 可执行文件。其中源文件为.h 和.c 文件, 主要用来实现程序的主要功能, 配置文件为.inf 文件, 主要用来对程序的格式进行定义以及对引用的库和程序入口进行说明。

6.2 固件安全模块设计

基于上章提出的 UEFI 固件安全方案, 本章设计 UEFI 固件安全模块, 并在 EDKII 环境下对其进行实现。由于驱动程序在 UEFI 固件中广泛使用并具有统一的标准, 本文采用驱动程序来实现固件安全模块的功能, 这样大大增强了固件安全模块的灵活性和扩展性。固件安全模块主要由以下几部分组成: 主模块、通信单元、Hash 算法引擎及存储模块, 如图 6-2 所示,

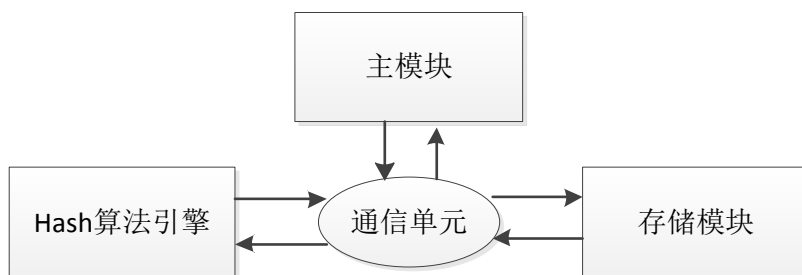


图 6-2 固件安全模块的设计

其中, 主模块是固件安全模块的主要功能的实现, 又由文件操作模块、日志生成模块和验证模块三部分组成。**Hash 算法引擎**是对完整性度量方法 **SHA-1** 算法的实现, 存储模块完成对测量日志和结果的存储, 通信单元则是负责在这三个模块中传递消息。下面将详细介绍这几个组成部分。

6.2.1 主模块

主模块是固件安全模块的主要组成部分, 主要功能是完成对 UEFI 固件中的文

件进行遍历和读取、生成完整性测量日志以及对完整性测量结果进行验证等。主模块主要分为文件操作模块、日志生成模块和验证模块，工作流程如图 6-3 所示，



图 6-3 主模块的工作流程图

- (1) 文件操作模块对 FFS 文件进行遍历，并读取 FFS 文件内容将其通过通信单元发送给 Hash 算法引擎进行计算；
- (2) Hash 算法引擎计算完毕后，将文件的 GUID 和对应的 Hash 值通过通信单元返回给文件操作模块；
- (3) 文件操作将完整性测量结果发送给日志生成模块用来生成完整性测量日志；
- (4) 日志生成模块根据完整性测量结果生成完整性测量日志，并发送给验证模块进行完整性验证；
- (5) 验证模块从存储区内读取存储文件中的记录信息，并将日志中的 FFS 文件 GUID 及测量结果与存储文件内的记录信息进行对比，如果文件 GUID 和测量结果一致，则对完整性测量结果进行验证，若通过验证则认为该 FFS 文件可信并进行下一个文件的可信度量，如果不一致或者未找到 FFS 文件的 GUID，则可信度量不通过，验证模块将调用 UEFI 服务发出错误信息。
- (6) 将生成的完整性日志存储在固件安全模块的存储模块内。

文件操作模块使用 PEI 服务表和 EFI_FIRMWARE_VOLUME2_PROTOCOL 协议对固件中的文件进行遍历和读取，该协议包含了对固件卷的文件级别的抽象以及固件卷的一些属性报告和配置服务，具体定义如下：

```

typedef struct_EFI_FIRMWARE_VOLUME2_PROTOCOL {
    EFI_FV_GET_ATTRIBUTES    GetVolumeAttributes;
    EFI_FV_SET_ATTRIBUTES    SetVolumeAttributes;
    EFI_FV_READ_FILE        ReadFile;
    EFI_FV_WRITE_FILE       WriteFile;
    EFI_FV_GET_NEXT_FILE    GetNextFile;
    UINT32    KeySize;
    EFI_HANDLE    ParentHandle;

```

```

EFI_FV_GET_INFO    GetInfo;
EFI_FV_SET_INFO    SetInfo;
} EFI_FIRMWARE_VOLUME2_PROTOCOL;

```

在实现基于固件安全模块的驱动时，需要将该协议绑定在驱动的句柄上，然后通过调用协议中的函数对固件卷中的文件进行遍历和读取。驱动首先调用 `ReadFile` 函数将固件卷中的一个 FFS 文件读取到缓冲区中，对文件内容进行完整性测量。若通过完整性测量则调用 `GetNextFile` 函数对下一个文件进行验证。

日志生成模块将文件 GUID 和完整性度量结果汇总生成完整性测量日志，并将其交给验证模块进行完整性验证，完整性测量日志采用自定义结构体 `EFI_TPM_LOG` 实现，定义如下：

```

typedef struct {
    EFI_GUID FileGuid;
    UINT8 FileType;
    CHAR16 *Checksum;
    BOOLEAN IsSignature;
    BOOLEAN IsTrusted;
} EFI_TPM_LOG;

```

主模块对每一个 FFS 文件进行可信度量后，都将生成一个 `EFI_TPM_LOG` 类型的日志信息。验证模块负责对日志信息中的文件 GUID 和完整性测量结果进行验证。结构体成员 `IsTrusted` 默认为 `False`，若通过可信度量，验证模块将其改为 `True`，并将日志信息存储在存储模块内。

6.2.2 通信单元

通信单元主要负责统一各模块之间的信息格式，完成固件安全模块内部各模块之间的信息传递。根据固件安全模块内部信息传递的需求，本文通过自定义驱动协议 `EFI_TPM_COMMUCATION_PROTOCOL`，使用协议下的函数接口完成各模块的消息传递，具体协议设计如下：

```

typedef struct _EFI_TPM_COMMUCATION_PROTOCOL {
    EFI_TPM_HANDLE Msg;
    UINT8 MsgType;
    CHAR8 *MsgBuf;
    EFI_TPM_READ_MESSAGE    Read;
    EFI_TPM_WRITE_MESSAGE   Write;
}

```

```
EFI_TPM_CLEAR_MESSAGE    Clear;
} EFI_TPM_COMMUCATION_PROTOCOL;
```

其中,Msg 是消息句柄,MsgType 用来标识消息类型,在固件安全模块内部主要有文件消息、日志消息、消息摘要等。MsgBuf 是一段 CHAR8 类型的缓冲区,用来存储消息内容,固件安全模块内部各模块可以根据消息类型对该缓冲区进行转换。该协议另外提供了三个函数供调用模块对消息进行处理。

6.2.3 Hash 算法引擎

Hash 算法引擎主要是对 SHA-1 算法的实现,用于对 UEFI 启动各阶段核心模块和驱动模块进行完整性测量。SHA-1 算法由美国 NIST 设计,它对输入长度小于 2^{64} 比特的消息,对消息按 512 比特的分组为单位进行处理,输出 160 比特的 Hash 值。由于 SHA-1 算法运用比较广泛,本文不再对算法的实现进行描述,并将 SHA-1 算法实现为 UEFI 开发平台的一个库,从而方便主模块的调用,其中最主要的是实现 Hash 算法服务协议,自定义协议如下:

```
typedef struct _EFI_HASH_PROTOCOL{
    EFI_HASH_CREATE CreateHash;
    EFI_HASH_COMPUTE ComputeHash;
    EFI_HASH_DELETE DeleteHash;
    EFI_HASH_GET_VALUE GetValue;
    EFI_HASH_GET_PARAMETER GetPara;
}EFI_HASH_PROTOCOL;
```

该协议提供 Hash 运算的接口函数,用于完成 Hash 运算功能。其中,CreateHash 函数用来生成一个 Hash 对象并进行初始化,ComputeHash 函数被 Hash 对象调用来对消息进行 Hash 运算,DeleteHash 用于销毁 Hash 对象,GetValue 函数和 GetPara 函数分别用来获取 Hash 值及 Hash 运算的参数。

6.2.4 存储模块

固件安全模块在 EFI 分区上建立存储文件 TpmStore.dat 文件作为固件安全模块的存储模块,用来存储 FFS 文件的初始 GUID 和完整性测量值等记录信息和完整性测量日志等信息。

UEFI 支持 FAT 格式的文件系统,并提供了 EFI_SIMPLE_FILE_SYSTEM_PROTOCOL 协议对磁盘分区进行操作,该协议主要使用接口 OpenVolume 为文件 I/O 范文打开一个设备卷,并返回该卷的 UEFI 文件协议实例的指针,然后利用该

指针使用 UEFI 文件协议，文件协议定义如下：

```
typedef struct _EFI_FILE_PROTOCOL {
    UINT64 Revision;
    EFI_FILE_OPEN Open;
    EFI_FILE_CLOSE Close;
    EFI_FILE_DELETE Delete;
    EFI_FILE_READ Read;
    EFI_FILE_WRITE Write;
    EFI_FILE_GET_POSITION GetPosition;
    EFI_FILE_SET_POSITION SetPosition;
    ...
} EFI_FILE_PROTOCOL;
```

EFI 分区是操作系统分区预留的分区，主要保存操作系统加载程序和一些驱动程序，采用 FAT 文件系统。本文通过 UEFI 文件协议的 Open 函数在 EFI 分区上建立存储文件，并使用 Read、Write 等函数对存储文件内容进行读取和更新。

6.3 固件安全模块实现

本文使用 EDKII 开发平台，开发基于固件安全模块的驱动程序，并将驱动程序的 EFI 文件封装成可以存储在固件卷中的 FFS 文件，然后将固件安全模块的 FFS 文件模块添加进真实的 UEFI BIOS 固件中进行验证。

固件安全模块在 PEI 阶段启动，因此要将固件安全模块实现为 PEI 阶段的驱动。本文在 MdeModulePkg\Universal 目录下建立 Tpmdriver 文件夹，在该文件夹下建立 Tpmdriver.c 和 Tpmdriver.inf 文件，Tpmdriver.c 源文件用于对固件安全模块的功能进行实现，Tpmdriver.inf 文件对源文件进行定义和说明，文件内容如下：

```
[Defines]

INF_VERSION      = 0x00010005
BASE_NAME        = Tpmdriver    //程序模块名称
FILE_GUID        = 27B89F18-ABB3-427A-8329-AAD8C84D88F0
MODULE_TYPE      = PEIM        //程序类型为 PEI 驱动
VERSION_STRING   = 1.0
ENTRY_POINT      = TpmdriverEntry //程序入口函数，与源文件一致

[Sources]

Tpmdriver.c      //程序源文件
```

```
[Packages]                                //驱动程序使用到下面两个组件
MdeModulePkg/MdeModulePkg.dec
MdePkg/MdePkg.dec

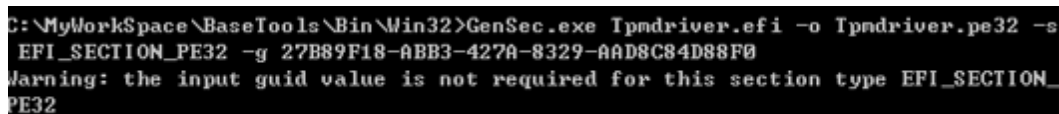
[LibraryClasses]
UefiLib                                  //UEFI 底层函数库
PeimEntryPoint                          //PEIM 入口函数实现
DebugLib                                //调试库
MemoryAllocationLib                    //提供接口函数进行内存操作
PeiServicesLib                          //PEI 提供的接口服务
TpmHashLib                              //自定义库，为程序提供 Hash 运算功能

[Protocols]
gTpmdriverProtocolGuid                  //驱动协议
gTpmdriverServiceBindingProtocolGuid    //驱动绑定协议
gEfiTpmCommicationProtocolGuid          //自定义通信协议
```

该文件是编译驱动程序的必要文件，其中 **Defines** 区域是对驱动程序的名称、属性等进行规定，最后一项 **ENTRY_POINT** 规定了源文件中的程序入口函数为 **TpmdriverEntry**，函数名必须与源文件中实现的一致。**Sources** 区域用于说明编译时的所有源文件，包括.h 文件和.c 文件。**Packages** 区域用于说明链接时用到的库，**Protocols** 区域则用来存放程序使用的协议的 **Guid**。

驱动程序的源文件 **Tpmdriver.c** 是对固件安全模块功能的实现，由于 Hash 算法引擎已被实现为 **TpmHashLib** 库，可以直接进行引用，因此 **Tpmdriver.c** 需要完成对主模块、通信单元和存储模块的编码，本文不再一一叙述。

通过 **EDKII** 的编译程序将源码编译，在 **Build** 目录下即可得到驱动程序的可执行文件 **Tpmdriver.efi**。但是，**EFI** 文件不能直接在固件卷中存储和运行，必须将其封装为 **FFS** 文件。**EDKII** 提供了 **EfiSec.exe** 和 **EfiFfs.exe** 两个工具来完成封装。首先需要使用 **EfiSec.exe** 将 **EFI** 文件封装为区块文件，如图 6-4 所示：



```
C:\MyWorkspace\BaseTools\Bin\Win32>GenSec.exe Tpmdriver.efi -o Tpmdriver.pe32 -s
EFI_SECTION_PE32 -g 27B89F18-ABB3-427A-8329-AAD8C84D88F0
Warning: the input guid value is not required for this section type EFI_SECTION_
PE32
```

图 6-4 固件安全模块封装为区块文件

将 **Tpmdriver.efi** 文件封装为 **Tpmdriver.sec** 文件后，利用 **EfiFfs.exe** 工具再将 **Tpmdriver.sec** 文件封装为 **Tpmdriver.ffs** 文件，如图 6-5 所示。



图 6-5 固件安全模块封装为 FFS 文件

生成的 Tpmdriver.ffs 文件即最终的驱动程序模块，也就是固件安全模块。将该文件添加到 UEFI 固件卷中，如图 6-6 所示。通过对 UEFI BIOS 重新刷写，在下次开机启动到 UEFI PEI 阶段时，该模块便会获得执行，并对 UEFI 固件进行可信度量，保护 UEFI 固件的安全。

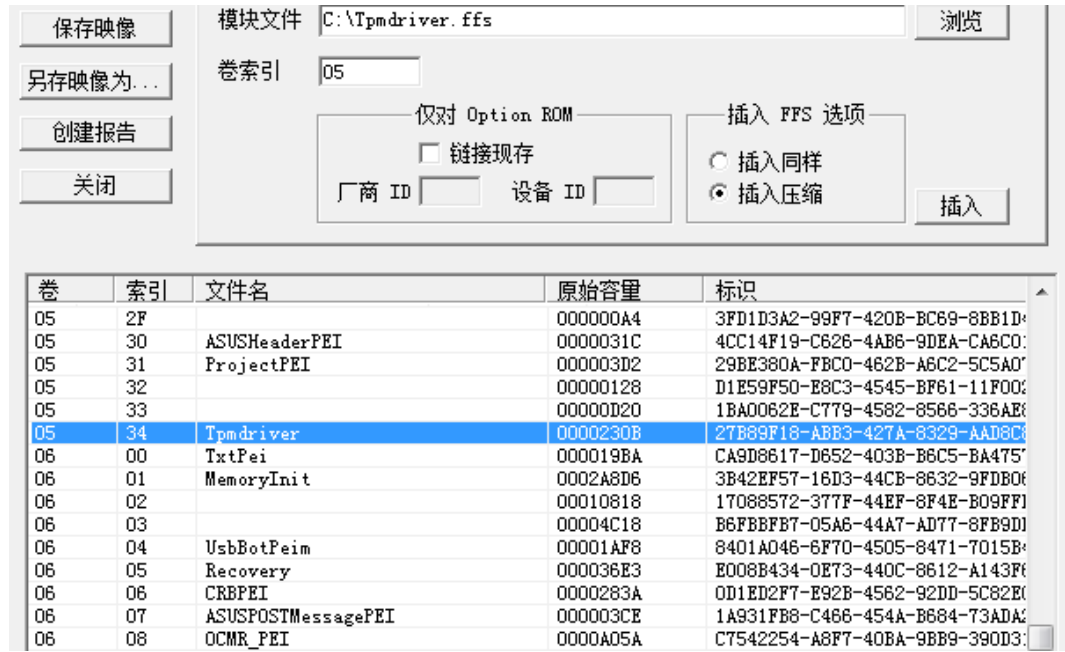


图 6-6 固件安全模块示意图

6.4 安全方案验证及结果分析

6.4.1 实验环境及工具

本次测试使用华硕 H81M 主板，该主板上使用 AMI 的 UEFI BIOS 芯片，固件存储设备为 Flash 芯片。测试系统为 Windows7 64 位操作系统。

测试工具为 MMTTool、编程器和 Checklog.efi，其中 MMTTool 是 AMI 公司推出的 BIOS 文件修改器，用来对 UEFI BIOS 的 ROM 文件进行修改。编程器为可编程的集成电路写入的工具，用来对 BIOS 芯片进行刷写。为了查看固件安全模块存储区内的记录信息，本文在 EDKII 环境下开发了 Checklog.efi 工具，用来在 UEFI Shell 环境下查看固件安全模块的存储记录。该工具使用 UEFI 提供的文件操作协议，可

以对 EFI 分区内存存储文件进行读写。

6.4.2 验证方法

本文将从安全方案的整体功能和对正常启动的影响两方面进行验证，验证方法如下：

1、对整体功能的验证

第一，根据第四章基于 FFS 文件的攻击方法的原理，将 UEFI 固件设备文件 H81M.rom 中的 ACPI.ffs 文件提取出来。根据 FFS 文件格式，将 ACPI.ffs 模块的 EFI 可执行文件 ACPI.efi 提取出来并进行修改。在 ACPI.efi 文件中加入一个新的区块，此区块的功能是打印出一条字符串“恶意代码已经获得执行！”，打印完毕后将执行权限交给 EFI 文件的原执行入口。然后将原 EFI 文件的执行入口改为新增区块的地址，从而达到在开机过程中打印字符串并不影响正常启动的攻击效果。攻击效果达到后，再将包含攻击代码的 FFS 文件插入到包含固件文件模块的 H81MSEC.rom 文件中，重新刷写 UEFI 固件，验证固件安全模块能否检测到这种攻击。

第二，分别对固件设备文件 H81M.rom 文件的 6 个固件卷中的不同 FFS 文件进行添加和修改，验证固件安全模块的功能。由于在固件设备文件内删除文件将导致计算机不能正常开启，用户只需重新刷写 UEFI 固件即可修复，因此这种对固件的攻击方式没有意义，本文的安全方案不对这种方式进行检测。

2、对 UEFI 启动效率影响的验证

本文分别对正常 UEFI 固件和包含固件安全模块的 UEFI 固件进行开机启动时间的统计，分析两者之间的差别，从而测试固件安全模块对计算机开机的影响。

6.4.3 验证结果及分析

6.4.3.1 整体功能验证及分析

第一，首先验证固件安全模块对基于 FFS 文件攻击方法的检测效果。将 H81M.rom 文件中的 ACPI 模块的 FFS 文件提取出来命名为 ACPI.ffs。然后根据 FFS 文件格式将其包含的 EFI 可执行文件提取出来，使用 PEditor 工具增加区块并将包含打印功能的恶意代码加入该区块中。将修改后的 EFI 文件与 ACPI.ffs 中的 EFI 文件替换，然后根据固件文件系统对 FFS 文件的完整性校验机制，修改 ACPI.ffs 文件的校验和，至此包含恶意代码的 FFS 文件生成完毕。删除 H81M.rom 文件中的 ACPI 模块，将修改后的 ACPI 模块命名为 MyACPI.ffs，插入到原始文件 H81MSEC.rom 中，如图 6-7 所示：

卷	索引	文件名	原始容量	标识
04	CB	AsusPostErrDxe	00001588	54AB7A17-AD08-4F86-83C2-4CF396
04	CC	ASUS_EUPSxSMI	00000E3A	D36DDD2D-1C66-4210-B77A-2FD9F9
04	CD	ASUS_EUPPEI	00000460	7DADBC98-6489-4D1C-907A-8EE24C
04	CE	GPIOROUTSmm	00000D54	5FB82B92-E5F6-417B-904E-6F617C
04	CF	EpuHwModePei	000032D6	362C7275-4D8F-4607-8D8F-28893A
04	D0	EpuHwModeDxe	00000CDA	21E34727-3881-4DEE-8020-D8908A
04	D1	ProjectSxSMI	000011AB	3F78CB8D-72EE-414E-B023-DACA0C
04	D2	ProjectDXE	000008D2	010216CD-9C09-4EB5-B7DA-DOA286
04	D3	MyACPI	00002308	16D0A23E-C09C-407D-A14A-AD0581
05	00	TxtPei	000019BA	CA9D8617-D652-403B-B6C5-BA4757
05	01	MemoryInit	0002A8D6	3B42EF57-16D3-44CB-8632-9FDB06
05	02		00010818	17088572-377F-44EF-8F4E-B09FF1
05	03		00004C18	B6FBBFB7-05A6-44A7-AD77-8FB9D1
05	04	UsbBotPeim	00001AF8	8401A046-6F70-4505-8471-7015B8
05	05	Recovery	000036E3	E008B434-0E73-440C-8612-A143F6
05	06	CRBPEI	0000283A	0D1ED2F7-E92B-4562-92DD-5C82E0
05	07	ASUSPOSTMessagePEI	000003CE	1A931FB8-C466-454A-B684-73ADA2
05	08	OCMR_PEI	0000A05A	C7542254-A8F7-40BA-9BB9-390D3C
05	09	Recovery	00000000	642DE777-E212-42ED-81CC-1B1B5F

图 6-7 修改后的 ROM 文件

将包含恶意代码的 H81M.rom 文件通过编程器刷写到固件中，在开机过程中可以看到控制权被攻击代码劫持，执行了 ACPI.efi 文件中新增区块中的代码，如图 6-8 所示：



图 6-8 插入代码获得执行权限示意图

然后，将修改后的 ACPI 模块文件 MyACPI.ffs 插入到包含有固件安全模块的 H81MSEC.rom 文件中，重新刷写 UEFI BIOS，开机过程中效果如图 6-9 示，在开

机过程中, 固件安全模块检测固件中文件的 ACPI 模块完整性遭到破坏, 发出错误信息, 并停止开机启动。

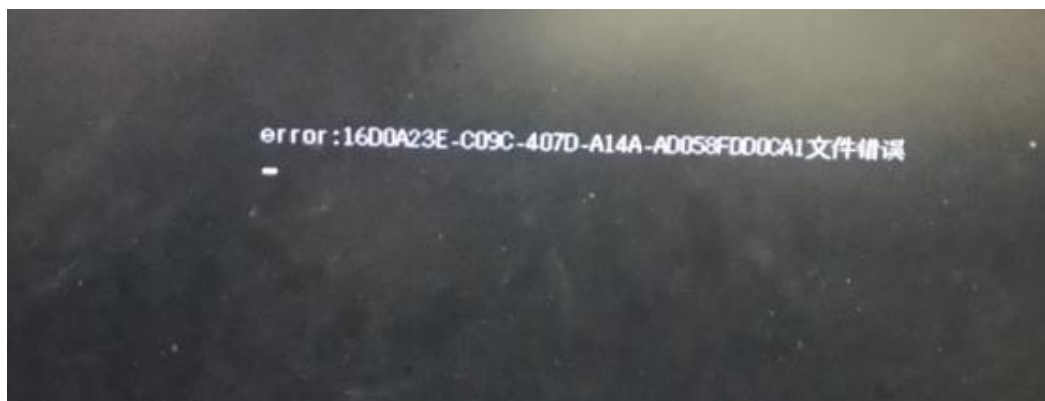


图 6-9 固件安全模块检测结果

固件安全模块将固件中的 FFS 文件信息保存在 EFI 分区的存储文件 TpmStore.dat 中。由于 EFI 分区不可见, 本文在 EDKII 环境下开发了工具 Checklog.efi, 该工具在 UEFI Shell 环境下运行, 能够检索 EFI 分区, 并读取 EFI 分区内的存储文件。使用 Checklog.efi 工具打开 EFI 分区中的存储文件 TpmStore.dat 文件, 可以存储文件存储的 FFS 文件及完整性测量结果, 如图 6-10 所示:

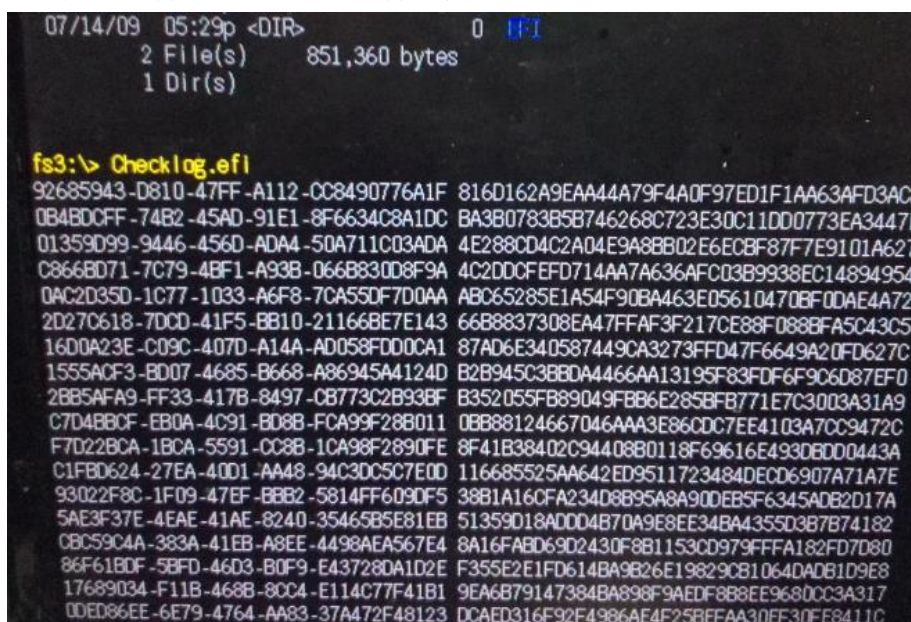


图 6-10 存储模块文件内容示意图

从以上结果可以看出, 修改后的 ACPI 模块文件 MyACPI.ffs 在开机过程中获得了执行。基于 FFS 文件的攻击方式通过修改 ACPI 模块内的 EFI 可执行文件文件, 使恶意代码在 UEFI 启动过程中获得了执行权限。但是固件安全模块通过可信

度量的方式，在开机过程中检测出 GUID 为 16D0A23E-C09C-407D-A14A-AD058FD00CA1 的文件不可信，及时终止了 UEFI 启动，阻止了恶意代码的执行。

第二，分别对各个固件卷中的不同 FFS 文件进行添加和修改，验证固件安全模块的功能。本文通过对 H81M.rom 文件中的 6 个固件卷进行 FFS 文件的添加和修改，验证固件安全模块的功能，首先对 FFS 文件修改的验证结果如表 6-1 所示，无文件名的 FFS 文件，使用文件 GUID 来表示。

表 6-1 FFS 文件修改的检测

固件卷	文件名	能否修改	检测结果
FV ₀	CEF5B9A3-476D-497F-9FDC-E98143E0422C	否	无
FV ₁	B52282EE-9B66-44B9-B1CF-7E5040F787C1	否	无
FV ₂	FD44820B-F1AB-41C0-AE4E-0C55556EB9BD	否	无
FV ₃	7BB28B99-61BB-11D5-9A5D-0090273FC14D	否	无
FV ₄	CORE_DXE	能	能够检测
FV ₄	ACPI	能	能够检测
FV ₅	CORE_PEI	否	无
FV ₅	PeiGfxDriver	能	能够检测

然后在各固件卷内添加包含恶意代码的 FFS 文件，重新刷写 BIOS 文件，检测结果如表 6-2 所示：

表 6-2 FFS 文件添加的检测

固件卷	能否添加 FFS 文件	恶意代码是否执行	检测结果
FV ₀	否	否	无
FV ₁	否	否	无
FV ₂	否	否	无
FV ₃	是	否	无
FV ₄	是	能	能够检测
FV ₅	是	能	能够检测

从表 6-1 可以看出，只要 FFS 文件能够被修改，在 UEFI 启动过程中都能被固件安全模块检测出来，由于 FV₀、FV₁、FV₂ 和 FV₃ 这四个固件卷中的 FFS 文件不符合修改条件，因此无法检测。而 CORE_PEI 模块是 PEI 阶段的核心 FFS 文件，其受到 SEC 阶段的保护，因此也不能进行修改。

从表 6-2 可以看出 FV₄ 和 FV₅ 能够添加包含恶意代码的 FFS 文件并都获得了

执行，但固件安全模块都能够进行有效检测。 FV_0 、 FV_1 和 FV_2 是固件厂商实现预留模块的卷，并且空间有限，不能插入 FFS 文件。 FV_3 虽然能够插入 FFS 文件，但不能获得执行权限，因此固件安全模块对此没有检测结果。

总之，经过对固件安全模块的整体功能进行验证，可以看到固件安全模块能够正常运行，并在开机过程中对 UEFI 启动过程对 FFS 文件进行可信测量，从而保护 UEFI 启动过程的可信。如果 UEFI 启动过程中的某一阶段不可信，固件安全模块将会停止启动整个计算机系统并调用 UEFI 服务打印错误信息。

6.4.3.2 对启动过程影响验证及分析

由于固件安全模块在 UEFI 启动过程中运行，并在每一阶段进行可信度量，不可避免地会对 UEFI 启动效率进行了影响，本文分别对正常开机时间和包含固件安全模块后的开机时间进行统计，如表 6-3 所示：

表 6-3 开机时间的平均统计

是否包含固件安全模块	测试次数	平均时间
否	100	15.36s
是	100	34.03s

从上表可以看出，在 UEFI 固件中加入固件安全模块后，计算机开机时间明显增长。由于固件安全模块采用驱动程序的方式实现，运行在 UEFI 启动过程中。在 UEFI 启动的每一个阶段，固件安全模块都要进行大量的文件操作和 Hash 计算，耗费了 UEFI 启动的时间，从而导致开机时间变长。因此，固件安全模块在对 UEFI 固件保护的同时，也产生了额外的时间开销，降低了 UEFI 启动的效率。

6.5 本章小结

本章首先介绍了目前使用最广泛的 UEFI 开发平台 EDKII，对该平台的几个重要开发包进行了说明。本章对基于可信计算的 UEFI 安全方案进行了实现，通过设计和实现固件安全模块，对 UEFI 安全方案的功能进行实现。固件安全模块由主模块、通信单元、Hash 算法引擎和存储模块组成，通过 UEFI 驱动程序的形式实现。本章最后对固件安全模块的功能进行了验证，分析了测试结果的正确性以及固件安全模块对 UEFI 启动效率的影响。

第七章 总结与展望

UEFI 作为新一代 BIOS 技术, 采用了现代化的系统架构和开发模式, 克服了传统 BIOS 开发和维护困难等缺点。但在当今网络安全事件频发、计算机系统攻击手段多样化的形势下, UEFI 和传统 BIOS 同样面临相当严峻的安全威胁。UEFI 采用现代的系统结构和数字签名等机制, 在某些程度上降低了自身的安全风险, 但是仍然无法阻止 Bootkit、固件设备文件修改等方式对 UEFI 的攻击。因此, 必须加强 UEFI 的安全机制研究, 完善 UEFI 的整体安全机制。

本文在对 UEFI 框架及其重要概念研究的基础上, 从 UEFI 固件文件系统的角度, 分析当前 UEFI 面临的安全威胁, 然后结合可信计算的理论提出了 UEFI 固件安全方案, 有效地防止了攻击者通过对固件设备文件的修改达到攻击 UEFI 的目的。主要工作具体体现在以下几个方面:

1、对 UEFI 框架及重要概念进行研究, 分析了 UEFI 系统服务、驱动模型和映像文件。然后结合 UEFI Framework 架构, 对 UEFI 启动各阶段的工作及其安全性进行了研究。最后对可信计算技术进行研究, 分析可信计算和 UEFI 启动过程的结合点, 为之后建立 UEFI 信任链提供了思路。

2、对 UEFI 固件文件系统进行重点研究, 并分析了当前固件文件系统所面临的安全威胁。本文结合固件文件系统结构, 从不同层次对固件文件系统进行了详细地分析, 并研究了 UEFI 固件如何对固件中的文件模块进行访问和执行。当前许多针对 UEFI 的攻击方法都是利用固件文件系统的弱点, 对 UEFI 固件中的文件进行修改和替换。本文分析了固件文件系统的安全性, 并研究了 UEFI Bootkit 的攻击方法和原理, 提出了基于 FFS 文件对 UEFI 固件进行攻击的方法。

3、利用可信计算原理, 提出了基于固件文件系统的 UEFI 固件安全方案。可信计算技术是研究 UEFI 保护机制的热点技术, 但是当下的研究大多是在建立可信计算平台, 利用 TPM 芯片对开机过程进行保护, 并未对 UEFI 固件设备文件进行过多保护。本文结合可信计算原理和 UEFI Framework 架构, 利用 Hash 算法对固件设备文件中的 UEFI 启动各阶段的核心固件文件和驱动固件文件进行完整性度量, 完成对 UEFI 启动过程的可信度量, 从而弥补了固件文件系统的缺点, 从固件文件的角度加强了 UEFI 固件的安全。

4、结合 UEFI 固件安全方案, 本文设计和实现了基于 UEFI 驱动程序的固件安全模块。该安全模块由主模块、Hash 计算引擎、通信单元及存储模块组成。固件安全模块对 UEFI 固件安全方案进行了具体实现, 通过 Hash 计算引擎对 FFS 文

件进行完整性测量，然后对比存储模块中的完整性测量日志来确定 FFS 文件的可信性。

本文紧紧围绕固件文件系统对 UEFI 的安全机制进行研究，提出了基于可信计算的 UEFI 安全方案，一定程度上增强了 UEFI 的安全性，但是仍存在很多不足，需在以后的工作中进一步完善和优化：

1、本文提出的 UEFI 固件安全方案是在 UEFI 固件启动时，对各个阶段执行代码进行完整性度量。当固件中的代码完整性遭到破坏后，安全模块将终止计算机系统的启动，而没有对被破坏代码的备份和恢复机制，这一点还需进一步完善。

2、固件安全模块是在 UEFI 启动过程中运行，并进行了大量的文件操作和哈希运算，在对 UEFI 固件保护的同时，也产生了额外的时间开销，降低了 UEFI 启动的效率。

3、本文采用 UEFI 驱动程序的形式实现固件安全模块，相对于硬件实现的方式虽然增强了灵活性和扩展性，但是安全性和效率不如后者。在今后的工作中，计划采用硬件和软件结合的方式，实现支持可扩展驱动模块的 TPM 芯片。

致 谢

时间飞逝，三年的硕士研究生的学习即将结束。在本文最后，我要衷心地感谢这期间帮助我的人，谢谢他们的关心和支持。

首先，衷心感谢我的导师范明钰教授。范老师不仅知识渊博、治学严谨，并且为人谦和、工作认真，不仅在工作方面而且在为人方面都给我树立了良好的榜样。在科研工作中，范老师给我们创造了良好的科研环境，提供了丰富的学习资料，并耐心指导和解决我们遇到的问题，提高了我解决问题的能力和专业知识的积累。在本文的选题、撰写和修改中，范老师给予我精心的指导和极大的支持，从而使我能够顺利完成论文的研究和写作。再次感谢范老师在各方面给我的帮助和支持。

其次，要感谢实验室的王老师以及所有同学。王老师为我们提供了良好的科研环境，并在生活方面给我们很大的帮助。感谢实验室内的同学对我学习和生活上的帮助，三年来的一起学习和生活让我难以忘怀。

感谢我的家人和朋友在我求学道路上的支持。我的父母是我求学道路上的坚强后盾，他们给了我一个和谐的家庭，排除了我求学路上的后顾之忧，他们的关怀和鼓励让我能够克服困难一直进步。感谢我的两个姐姐，她们的关心和帮助让我对未来充满信心。感谢肖秀芹对我的支持，她不仅在生活学习中给我鼓励，在论文写作过程中的监督和帮助也让我受益匪浅。

最后，感谢所有参加本文评审和答辩的老师，谢谢你们抽出宝贵的时间对我的论文进行评审和指导。

参考文献

- [1] Compaq, Phoenix, Intel. BIOS Boot Specification v1.01[OL]. <http://www.phoenix.com/pages/bios-updates-from-esupportcom.1998>
- [2] 周洁,谢智勇,余涵等. 基于 UEFI 的国产计算机平台 BIOS 研究[J]. 计算机工程, 2011, S1: 355-358.
- [3] 杨培,吴灏,金然. BIOS 安全防护技术研究[J]. 计算机工程与设计,2008,15:3840-3842+3914.
- [4] 池亚平,许盛伟,方勇. BIOS 木马机理分析与防护[J]. 计算机工程,2011,13:122-124.
- [5] UEFI Spec Working Group. Unified Extensible Firmware Interface Specifications 2.3. 1 Errata B[C]. UEFI Forum, 2012,120-376
- [6] UEFI Spec Working Group. Platform Initialization Specification1.3[C]. UEFI Forum, 2013.
- [7] 吴松青, 王典洪. 基于 UEFI 的 Application 和 Driver 的分析与开发[J]. 计算机应用与软件, 2007, 24(2): 98-100.
- [8] 刘冬, 文伟平. EFI 及其安全性分析[J]. 信息网络安全, 2009 (5): 32-34.
- [9] 唐文彬, 祝跃飞, 陈嘉勇. 统一可扩展固件接口攻击方法研究[J]. 计算机工程, 2012, 38(13): 99-101,111.
- [10] 王强.计算机界的埃博拉病毒, 你的电脑还好吗?[OL].<http://www.leiphone.com/news/201503/HivKkuD2K5Clcr67.html>.2015-3-26
- [11] 360 安全播报.Hacking Team 使用 UEFI BIOS Rootkit 长期控制目标系统[OL].<http://bobao.360.cn/news/detail/1772.html>.2015-7-15
- [12] M.Nystrom,M.Nicoles,V.Zimmer. UEFI Networking and Pre-OS Security[J]. Intel Technology Journal, 2011 (15): 80-1-1.
- [13] 王吉发. 基于 UEFI 的病毒扫描引擎的设计与实现[D].哈尔滨工程大学,2011,12-26
- [14] 陈楣, 周振柳, 许榕生. 基于 Framework 的固件卷结构的分析[J]. 计算机工程与应用, 2007, 43(15): 86-88.
- [15] 徐明迪,张焕国,赵恒等. 可信计算平台信任链安全性分析[J]. 计算机学报,2010,07: 1165-1176.
- [16] 姜子峰. BIOS 陷门关键技术研究[D].解放军信息工程大学,2013,19-43
- [17] 联想服务与支持.UEFI EDK2 Capsule 更新漏洞[OL].http://support1.lenovo.com.cn/lenovo/wsi/htmls/detail_20160302182754681.html.2016-3-2
- [18] 朱贺新. 基于 UEFI 的可信 BIOS 平台研究与应用[D].西安科技大学,2008,8-16
- [19] 曾颖明, 谢小权. 基于 UEFI 的可信 Tiano 设计与研究[J]. 计算机工程与设计, 2009 (11):

- 2645-2648.
- [20] 黄海彬. 基于 EFI 固件文件系统的平台安全策略研究与实现[D].上海交通大学, 2010,26-37
- [21] V.Zimmer. Platform Trust Beyond BIOS Using the Unified Extensible Firmware Interface[C]. Security and Management. 2007: 400-405.
- [22] TCG.TCG Specification Architecture Overview[OL]. <https://www.trustedcomputinggroup.org>
- [23] TCG.TCG EFI Platform Version 1.00[OL]. <https://www.trustedcomputing.group.org>
- [24] TCG.TCG EFI Protocol Version 1.00[OL]. <https://www.trustedcomputing.group.org>
- [25] V.Zimmer, M.Rothman, S.Marisetty. Beyond BIOS: Developing with the Unified Extensible Firmware Interface[M]. Intel Press, 2010,36-95
- [26] 韩德强,马骏,张强. UEFI 驱动程序的研究与开发[J]. 电子技术应用, 2014,05:10-13+17.
- [27] D.LIU, W.WEN. EFI and Security Analysis of EFI Framework[J]. Netinfo Security, 2009, 5: 020.
- [28] 唐文彬, 陈熹, 陈嘉勇等. UEFI Bootkit 模型与分析[J]. 计算机科学, 2012, 39(10):308-312.
- [29] Zimmer V J, Rothman M A. System and method to secure boot uefi firmware and uefi-aware operating systems on a mobile internet device (mid): U.S. Patent Application 12/165,593[P]. 2008-6-30.
- [30] 付思源, 刘功申, 李建华. 基于 UEFI 固件的恶意代码防范技术研究[J]. 计算机工程, 2012, 38(9): 117-120.
- [31] 冯登国,秦宇,汪丹等. 可信计算技术研究[J]. 计算机研究与发展, 2011,08:1332-1349.
- [32] 陈楠,王震宇,窦增杰等. 可执行可信软件安全性分析技术研究[J]. 计算机工程与设计,2010,22:4802-4805.
- [33] 沈昌祥,张焕国,王怀民等. 可信计算的研究与发展[J]. 中国科学:信息科学,2010,02:139-166.
- [34] 段晨辉. UEFI BIOS 安全增强机制及完整性度量的研究[D].北京工业大学,2014,41-52
- [35] 刘佳, 辛晓晨, 沈钢纲,等. 基于 UEFI 的 Flash 更新的开发研究[J]. 计算机工程与设计, 2011, 32(1): 114-117.
- [36] 付思源. 基于统一可扩展固件接口的恶意代码防范系统研究[D].上海交通大学,2011,36-46
- [37] Wojtczuk R, Kallenberg C. Attacking UEFI boot script[C]//31st Chaos Communication Congress(31C3),[http://events.ccc.de/congress/2014/Fahrplan/system/attachments/2566/original/venamis whitepaper. pdf](http://events.ccc.de/congress/2014/Fahrplan/system/attachments/2566/original/venamis%20whitepaper.pdf) . 2014.
- [38] S.E.Jones, E.Khoruzhenko. Parametric Build of UEFI Firmware: U.S. Patent Application 12/756,437[P]. 2010-4-8.

- [39] Wei-hua HUO. A Secure Module Analysis of UEFI[J]. Information Security and Communications Privacy, 2008, 7: 040.
- [40] Allievi A. UEFI technology: say hello to the Windows 8 bootkit[J]. 2012:34-56
- [41] Wilkins R, Richardson B. UEFI Secure Boot in Modern Computer Security Solutions[C]. Tech. rep, UEFI Forum, 2013,156-232
- [42] J.Yao, V.Zimmer, R.Rangarajan, et al. White Paper A Tour Beyond BIOS Using the Intel® Firmware Support Package with the EFI Developer Kit II[J]. 2014:452-486
- [43] Bottomley J, Corbet J. Making uefi secure boot work with open platforms[J]. Linux Foundation white paper, 2011:15-27
- [44] 李战. 基于数字签名的 UEFI BIOS 安全更新机制的研究与实现[D]. 北京工业大学, 2014,9-24
- [45] T.Rossow . TPM 2.0, UEFI and their Impact on Security and Users' Freedom[J]. 2013:33-56
- [46] 刘东丽. 基于 UEFI 的信任链设计及 TPM 驱动程序实现[D]. 华中科技大学, 2011,17-24
- [47] Cai-xia WGYAN. Trusted Chain Based on UEFI BIOS[J]. Microcomputer Information, 2011, 5: 067
- [48] 崔莹,辛晓晨,沈钢纲. 基于 UEFI 的嵌入式驱动程序的开发研究[J]. 计算机工程与设计,2010,10:2384-2387