基于 UEFI 固件的恶意代码防范技术研究

付思源[a]，刘功申[b]，李建华[a]

(上海交通大学 a. 电子工程系；b. 信息安全工程学院，上海 200240)

摘　要：统一可扩展固件接口(UEFI)缺乏相应的安全保障机制，易受恶意代码的攻击，而传统的计算机安全系统无法为固件启动过程和操作系统引导过程提供安全保护。针对上述问题，设计基于 UEFI 的恶意代码防范系统。该系统利用多模式匹配算法实现特征码检测引擎，用于在计算机启动过程中检测与清除恶意代码，并提供恶意启动项处理及系统内核文件备份等功能。实验结果证明，该系统能为固件及上层操作系统提供完善的安全保护，且代码尺寸小、检测速度快，恶意代码识别率高。

关键词：基本输入输出系统；统一可扩展固件接口；固件；恶意代码；特征码匹配；多模式匹配

Research of Malicious Code Defense Technology Based on UEFI Firmware

FU Si-yuan[a], LIU Gong-shen[b], LI Jian-hua[a]

(a. Department of Electronic Engineering; b. School of Information Security Engineering, Shanghai Jiaotong University, Shanghai 200240, China)

【Abstract】Unified Extensible Firmware Interface(UEFI) faces a grim challenges of malicious code attack. The traditional computer security software can not provide security for the firmware and operating system boot process. In order to solve the problem, this paper designs a malicious code defense system based on UEFI firmware. By using the multi-pattern matching algorithm, a signature detecting engine under UEFI environment is implemented, which provides functionally of malicious code detect, boot option analysis and firmware and operating system kernel backup. Experimental results prove that the system can effectively resist malicious code with small code size and low costs to meet the firmware's need of flash size and fast boot.

【Key words】Basic Input Output System(BIOS); Unified Extensible Firmware Interface(UEFI); firmware; malicious code; signature matching; multi-pattern matching

DOI: 10.3969/j.issn.1000-3428.2012.09.035

1　概述

近年来，随着病毒、木马、网络蠕虫等恶意代码攻击技术的不断成熟，仅依赖于操作系统级别的安全防范机制已经不能满足信息安全技术的发展需求，计算机安全体系必须进一步向下延伸到系统固件层甚至硬件层。固件作为计算机处理器最早执行的软件，在嵌入式系统、个人计算机和服务器上广泛存在，其安全性直接影响上层操作系统和整个计算机安全体系。针对固件安全问题的研究已经成为信息安全领域研究和关注的新热点。

基本输入输出系统(Basic Input Output System, BIOS)作为第 1 代计算机固件架构，自 1981 年设计至今被广泛应用于计算机系统中。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。以 1999 年大规模爆发的 CIH 病毒为代表，恶意代码对系统的入侵开始向下延伸到固件层。文献[1-2]描述了利用 APCI、PCI 的漏洞实现在 BIOS 中植入 Rootkit 恶意代码的技术。文献[3]利用固件系统管理模式(System Management Mode, SMM)植入代码攻击 Intel 的 TXT(Trusted Execution Technology)安全技术。由于固件处于计算机系统的底层，当 BIOS 受到上述威胁攻击时，会导致计算机操作系统在加载之初就被入侵者所控制，从而给发现和清除这些威胁带来极大的困难。

2003 年，Intel 对外公布了其制定的新一代固件标准，用以替代传统的 BIOS。随后，联合业界建立了统一可扩展固件接口(Unified Extensible Firmware Interface, 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，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，C 语言风格的参数堆栈传递方式，具有操控所有硬件资源的能力，较传统的 BIOS 开发更为便利和灵活。由 Tiano 社区负责的 EDK II 项目是根据最新 UEFI 标准开发和维护的开源实现平台。本文在此基础上对基于 UEFI 固件的恶意代码防范技术进行了研究。

2　UEFI 固件的安全需求

随着 UEFI 固件逐渐替代传统的 BIOS，针对其安全性问题的研究也越来越多。最新版本的 UEFI 标准[4]中加入了关于可信启动、数字签名和数字摘要等服务的定义，这些定义符合可信计算组织制定的可信平台规范[5]，可用于固件执行过程中的完整性检查和身份认证。文献[6]利用上述 UEFI 标准中的服务接口，给出了一种基于数字签名和数字摘要的安全启动模型。文献[7]通过在系统中引入 TPM 芯片作为可信测量根，在预启动环境中构建操作系统运行前的可信链，实现了一种基于 UEFI 平台的可信固件。这些研究成果主要集中在可信计算领域，以数字认证技术为核心，通过禁止固件中

危害。在实际应用中，这种方法存在以下缺陷：

(1)缺乏灵活的安全管理机制。UEFI 固件相对于传统 BIOS 的优势之一在于其灵活性和开放性，而基于信任链的数字认证技术极大地限制了这方面的发展。UEFI 平台下现有的数字认证技术通过禁止未知代码的运行来保障固件的安全，因此，任何代码(包括 UEFI 固件本身和上层操作系统的加载程序)都必须经过上级认证后加入信任链才能够运行。系统一经部署使用，即使是系统用户确认安全的固件代码也无法安装和运行。这种限制虽然能够保证固件自身的安全，但由于管理复杂，整个系统的灵活性和扩展性较差，因此不能适应环境变化的需求。

(2)缺乏对上层操作系统的保护。目前的研究成果只着眼于保护 UEFI 固件本身的安全，而未考虑将 UEFI 纳入整个计算机系统的安全防范体系中。从对传统 BIOS 的攻击技术来看，入侵者攻击系统固件的目的通常不仅仅是破坏系统固件或简单地阻止操作系统的启动，而是利用固件漏洞向系统中植入恶意代码，进而在操作系统加载之初获得系统的控制权，实现对上层操作系统的入侵。因此，禁止未知代码修改固件中的关键数据项仅能保证系统的启动流程不被破坏，难以检测和识别针对上层操作系统的攻击。

因此，UEFI 固件需要更加完善的安全防范机制。

3 基于 UEFI 固件的恶意代码防范系统

UEFI 固件的灵活性和开放性给计算机安全技术带来了新的发展空间。相比传统的 BIOS，UEFI 更像一个微操作系统，可在操作系统载入之前操控所有的硬件资源，且具有实现更复杂逻辑运算的能力。本文将恶意代码防范技术引入固件层，以基于特征码的恶意代码检测引擎为核心，设计并实现一个基于 UEFI 固件的恶意代码防范系统，为整个计算机系统提供完整的安全防范工作。

根据最新版本的 UEFI 及相关标准，计算机系统的启动流程分为 5 个阶段，依次为 EFI 预启动(Pre-EFI Initialization, PEI)、驱动运行环境(Driver Execution Environment, DXE)、Boot 设备选择(Boot Device Select, BDS)、操作系统(Operating System, OS) Boot 和 OS 运行环境 Runtime。本文提出的恶意代码防范系统运行于 UEFI 环境下，为 UEFI 和上层操作系统提供相应的调用接口。在计算机启动的不同阶段，系统分别提供相应的安全检测与防范措施，如图 1 所示。

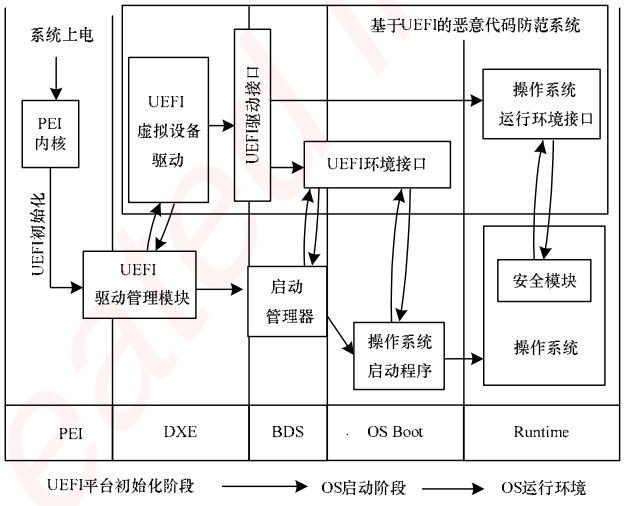

图 1 基于 UEFI 固件的恶意代码防范系统

3.1 PEI 阶段

PEI 阶段主要负责基本的上电初始化工作，如 CPU 和内存的初始化。这部分固件代码的安全检查工作由 PI 标准中规定的 SEC 模块专门负责，不在本文研究范围之内。

3.2 DXE 阶段

DXE 阶段是 UEFI 驱动程序的运行环境，也是 UEFI 固件运行的主要阶段。在本阶段开始时，UEFI 内核的 Driver Dispatcher 将枚举所有外围设备的驱动程序，并判断其相互依赖关系，依次载入系统。所有 UEFI 驱动成功加载后，UEFI 固件即完成启动准备工作，可继续运行其他 UEFI 应用程序或开始引导操作系统。

本文提出的恶意代码防范系统以 UEFI 环境下虚拟设备驱动程序的形式实现，在 DXE 阶段的初期加载到 UEFI 运行环境中。在随后过程中，UEFI 内核可利用本系统提供的恶意代码检测功能检查其他设备驱动程序和 UEFI 应用程序，根据检测结果判断目标代码是否存在安全威胁，并采取相应的防范措施，能够有效地弥补基于信任链的数字认证技术在灵活性上的不足。

3.3 BDS 阶段

BDS 阶段是 UEFI 环境中启动管理器运行的阶段，主要负责发现和选择正确的操作系统引导设备，根据系统策略或用户的选择调用相应的系统引导程序。目前 UEFI 固件选择操作系统的安全策略主要依据 2 点：

(1)操作系统存放的物理位置，如来自本地硬盘、可移动存储设备(U 盘、光盘)或网络启动；

(2)引导程序(即操作系统的入口程序)能否通过数字签名和认证。

由此可见，当前 UEFI 固件对操作系统的选择和判断是非常简单的。

本系统在 UEFI 调用操作系统引导程序之前加入对上层操作系统内核的完整性检查功能。由于操作系统的核心程序具有稳定性且数量有限，因此系统提取这些文件的数字摘要，并与之前完整性检查记录中的数据进行比对，判断核心文件是否被破坏或篡改。必要时可利用 UEFI 的网络接口提取远程备份文件完成操作系统的修复工作，确保操作系统核心的安全性和完整性。

3.4 OS Boot 阶段

UEFI 启动管理器调用操作系统引导程序之后，系统的控制权由 UEFI 固件转移到操作系统，计算机进入操作系统引导阶段。在本阶段，操作系统仍然可以调用 UEFI 环境下的所有服务。当操作系统加载完成后，UEFI 所占用的大部分系统资源将被释放，计算机启动完成，进入操作系统运行环境。

在此阶段中，上层操作系统可调用本系统的服务接口，根据事先拟定的安全策略和恶意代码检测引擎的检查结果，清除操作系统启动过程中的恶意代码程序。操作系统启动完成后，本系统的主要功能已经完成，将释放恶意启动项处理和完整性检查模块所占用的内存资源，供上层操作系统使用。

3.5 Runtime 阶段

Runtime 是操作系统启动完成后的正常运行过程，此时大部分 UEFI 资源已被释放，仅保留少量必要模块存放在内存中，供操作系统调用。

本文系统在操作系统运行环境中可选择性地驻留内存，配合上层安全防范软件进行工作。在操作系统运行环境中，某些恶意程序能够寄生在操作系统的核心服务中，或设法先

操作系统中的安全软件即使能够检测到，也难以直接进行彻底有效的清除，只能通过反复重启或由用户手动启动到安全模式下进行清理。在加入本系统后，上层安全软件在检测到恶意代码后，可将相关文件信息和处理办法利用本系统提供的服务接口传递到 UEFI 固件层，尝试在 UEFI 环境下进行清理，或由固件将这些信息保存下来，在下一次计算机启动过程的 OS Boot 阶段进行清理工作。

由于系统的所有模块都在 UEFI 环境下运行，将整个系统的恶意代码防范工作提前到计算机固件启动的初期阶段，因此能够在恶意代码启动之前完成查杀，保证操作系统的正常启动及上层安全软件的顺利运行。同时由于本系统的设计完全符合 UEFI 标准，向上层系统提供统一的调用接口，因此不受系统平台架构的限制，可用于任何支持 UEFI 固件的系统环境。

4 系统实现及性能分析

4.1 虚拟设备驱动的结构

为了尽早在 UEFI 运行环境中启动，系统设计为虚拟设备驱动程序的形式，且仅依赖于文件系统、网络接口、数字认证和部分 UEFI 内核服务，系统依赖要求极低，因此，在 UEFI 驱动管理模块判断驱动程序加载顺序时具有极高的优先级，在 DXE 阶段初期即可进行加载。系统内部以基于特征码的恶意代码检测引擎为核心，主要包括数据管理和恶意代码防范 2 个模块，其结构如图 2 所示。

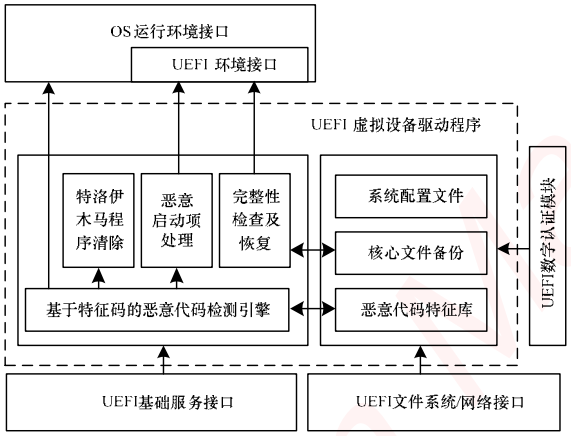

图 2 虚拟设备驱动实现结构

系统数据管理主要包括系统配置文件、核心文件备份和恶意代码特征库 3 个部分。其中，配置文件存储程序配置信息、远程服务器信息及完整性检查记录，这部分数据需存储在本地硬盘的 UEFI 隐藏分区中，对于无盘工作站可压缩后作为 UEFI 变量存储在主板的 Flash 芯片内。恶意代码特征库和文件备份数据可保存在远程服务器中，通过 UEFI 提供的网络服务接口进行维护和更新。为保证不被其他恶意程序篡改，系统程序文件和所有数据文件均经过数字签名和加密后存储，受到 UEFI 数字认证模块的保护。

恶意代码防范模块是系统的核心，以基于特征码的检测引擎为基础，提供恶意代码检测、特洛伊木马程序清理、恶意启动项处理功能。在启动阶段，系统还将对 UEFI 固件和操作系统中的核心文件进行完整性检查，计算其 MD5 摘要信息并与之前的记录进行比对，以确认核心程序没有被篡改。

由于系统需要对 DXE 阶段之后的整个计算机运行过程提供安全保护，因此需要对外提供 2 套调用接口。在 DXE、BDS 和 OS Boot 阶段，系统将注册为 UEFI 系统服务，以其他所有 UEFI 程序和操作系统的引导程序调用。在操作系统运行环境中，根据 UEFI 标准的规定，将调用接口进行封装，利用 UEFI Runtime Service 中的 Update Capsule 交换机制实现与上层操作系统的信息传递。

4.2 恶意代码检测引擎的实现

恶意代码的检测通常分为 3 种类型：基于特征码的检测，基于语义的检测和基于行为的检测，本文系统采用基于特征码的检测方法，具有针对性强、误报率低、开销较小的优点，非常适用于固件运行环境。特征码是对样本进行分析得到的一串或一组二进制序列，用以标识恶意代码的身份，在检测中将样本与特征码进行匹配，若匹配成功，则认定样本受到恶意代码的感染。

系统以经典的 AC(Aho-Corasick)[8]多模式匹配算法为基础实现恶意代码检测引擎。AC 算法在进行匹配之前先对模式串(特征码)集合进行预处理，生成树状的有限状态机，之后只需要对样本进行一次扫描就可以精确匹配到目标模式串。这种算法的优点在于匹配速度快，对平均长度为 n 字节的模式串集合进行匹配操作的时间复杂度仅为 $O(n)$，但空间复杂度较高，在预处理过程中构建查询树需要创建和维护 256^n 个状态节点。

通常特征码的平均长度为数十至上百字节，在实验中发现，由于 UEFI 环境下内存操作效率的限制，维护数量如此巨大的状态节点使特征库的预处理时间远远超过了可接受的范围。为此，本文对 AC 算法进行改进，利用每个特征码的前 m 个字节构建查询树，即利用有限状态机匹配模式串的前 m 字节后得到较小的目标模式串集合，再通过精确匹配得到目标结果。

实验证明，在实际运行环境中，根据平台的内存操作效率和特征码集合的平均长度选择合适的 m 值(通常为 3~5)，可有效平衡预处理和匹配操作所需的时间，使整个系统的效率达到较高的水平。

4.3 实验验证与性能分析

由于实验的目的是测试 UEFI 平台下恶意代码防范系统的功能和运行效率，特征码的分析提取技术不是本文研究的重点，因此在实验中使用 Clam AntiVirus 网站提供的 1 万条病毒特征码(总长 973 KB)作为恶意代码库，并采用 VX heaven 网站上收集的其中 68 个病毒文件(共 7.53 MB)作为恶意代码样本。在 DXE 阶段，以主板固件中的全部 130 个 EFI 程序(3.82 MB)和上述恶意代码样本作为待测样本；在操作系统启动阶段，以 Windows 操作系统 System32 文件夹下选取的 200 个可执行文件(30.61 MB)和上述恶意代码样本作为待测样本。实验系统基于 EDKII 平台开发，运行实验的操作系统为 Windows XP，硬件环境为 Intel Tunnel Mountain(DQ57TM)系主板，酷睿 3.2 GHz 处理器，1 GB 内存。

本文系统编译后的 EFI 程序文件为 88 KB，可置入主板的 Flash 芯片中运行。实验平台在加入系统前后的启动效率如图 3 所示。在 UEFI 固件运行环境中，本文系统对测试样本的漏报率为 10.3%。恶意代码特征库加载的平均时间为 2 201 ms，在 DXE 阶段对固件中所有待测样本进行检测所消耗的时间为 321 ms，在 BDS 阶段对操作系统中所有待测样本进行检测所消耗的时间为 2 397 ms。实验结果表明，系统在对计算机启动性能产生较小影响的情况下，能够在 UEFI 环境中成功实现恶意代码的检测与防范。

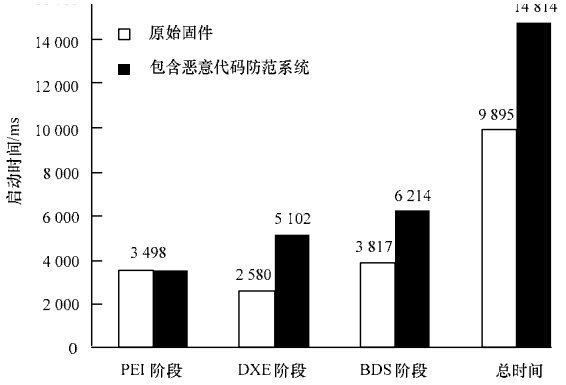

图 3　本文系统对计算机启动效率的影响

5　结束语

恶意代码防范是信息安全领域的难点，其典型问题之一是某些恶意代码会先于安全软件加载到系统，优先控制操作系统，导致安全软件不能对其有效地清除。

本文提出的恶意代码防范技术不仅能够用于检测固件环境中的恶意代码，而且可以在操作系统的整个加载过程持续提供防范工作，保证操作系统和上层安全软件的正常载入和运行，有效地解决了上述问题。下一步的研究工作主要有以下 2 点：

(1)继续研究针对 UEFI 固件的恶意代码行为特征；

(2)进一步选择和优化基于 UEFI 的恶意代码检测算法，降低对计算机启动性能的影响。

[1] Heasman J. Implementing and Detecting an ACPI BIOS Rootkit[C]//Proc. of BLACKHAT'06. Washington D. C., USA: [s. n.], 2006.

[2] Heasman J. Implementing and Detecting an PCI Rootkit[C]//Proc. of BLACKHAT'07. Washington D. C., USA: [s. n.], 2007.

[3] Wojtczuk R, Rutkowska J. Attacking Intel Trusted Execution Technology[C]//Proc. of BLACKHAT'09. Washington D. C., USA: [s. n.], 2007.

[4] The Unified EFI Forum. Unified Extensible Firmware Interface Specification Version 2.3.1[EB/OL]. (2011-04-08). http://www.uefi.org.

[5] Trusted Computing Group. TCG PC Specific Implementation Specification Version 1.1[EB/OL]. (2003-08-04). http://www.trustedcomputing.org.

[6] Zimmer V J. Platform Trust Beyond BIOS Using the Unified Extensible Firmware Interface[C]//Proc. of SAM'07. Las Vegas, USA: [s. n.], 2007.

[7] Zhou Zhenliu, Xu Rongsheng. BIOS Security Analysis and a Kind of Trusted BIOS[C]//Proc. of the 9th International Conference on Information and Communications Security. Zhengzhou, China: [s. n.], 2007: 427-437.

[8] Aho A V, Corasiek M J. Efficient String Matching: An Aid to Bibliographic Search[J]. Communications of the ACM, 1975, 18(6): 333-343.

编辑　张　帆

(上接第 116 页)

(6)目前大多数的图像置乱加密算法都只针对方形图像，本文算法既能对方形图像进行置乱，又能对矩形图像进行置乱。

4　结束语

传统基于像素位置改变的置乱算法不能改变像素的灰度分布特征，在基于像素值改变的置乱算法中，低维的变换不能改变像素的灰度分布特征，高维的变换迭代周期长、解密代价高，基于单组和双组混沌系统的置乱算法存在加密强度不够、密钥空间小、留有轮廓等问题。本文利用多组混沌序列和矩阵拉直算子解决上述问题，通过对彩色数字图像的 RGB 矩阵进行拉直和置乱，使人们无法识别原始图像所表达的真正含义。理论分析和实验结果表明，该算法具有较大的密钥空间，无需迭代，能同时实现像素位置和像素值的双置乱，对密钥很敏感，能改变图像的灰度特征，且置乱后的图像纹理细，颗粒均匀。

参考文献

[1] 王冬梅, 金一庆. 双偶阶幻方变换数字图像的半周期[J]. 浙江大学学报, 2005, 32(3): 274-276.

[2] Shen Jianbin, Jin Xiaogang, Zhou Chuan. A Color Image Encryption Algorithm Based on Magic Cube Transformation and Modular Arithmetic Operation[C]//Proceedings of the 6th Pacific Rim Conference on Multimedia. Jeju Island, Korea: [s. n.], 2005: 270-280.

[3] Han Fengling, Hu Jiankun, Yu Xinghuo. A Biometric Encryption Approach Incorporating Fingerprint Indexing in Key Generation[C]//Proceedings of International Conference on Biometrics. Kunming, China: [s. n.], 2006: 675-681.

[4] 林雪辉, 蔡利栋. 基于 Hilbert 曲线的数字图像置乱方法研究[J]. 中国体视学与图像分析, 2004, 9(4): 224-227.

[5] 黄外斌, 张　憧, 董光昌. 基于 Arnold 变换的图像逆置乱算法[J]. 高校应用数学学报, 2008, 23(1): 99-104.

[6] Qi Dongxu, Zhou Jiancheng, Han Xiaoyou. A New Class of Transformation and Its Application in the Image Transformation Covering[J]. Science in China: Series E, 2000, 43(3): 304-312.

[7] 杨礼珍, 陈克非. 变换矩阵的阶及两种推广 Arnold 变换矩阵[J]. 中国科学: E 辑, 2004, 32(4): 151-161.

[8] 邵利平, 覃　征, 衡星辰, 等. 基于高维矩阵变换的雪崩图像置乱变换[J]. 中国图象图形学报, 2008, 13(8): 1429-1436.

[9] 邓绍江, 张岱固, 濮忠良. 一种基于混沌的图像置乱算法[J]. 计算机科学, 2008, 35(8): 238-240.

编辑　张　帆

嵌入式资源免费下载

总线协议：

1. 基于 PCIe 驱动程序的数据传输卡 DMA 传输
2. 基于 PCIe 总线协议的设备驱动开发
3. CANopen 协议介绍
4. 基于 PXI 总线 RS422 数据通信卡 WDM 驱动程序设计
5. FPGA 实现 PCIe 总线 DMA 设计
6. PCI Express 协议实现与验证
7. VPX 总线技术及其实现
8. 基于 Xilinx FPGA 的 PCIE 接口实现
9. 基于 PCI 总线的 GPS 授时卡设计
10. 基于 CPCI 标准的 6U 信号处理平台的设计
11. USB30 电路保护
12. USB30 协议分析与框架设计
13. USB 30 中的 CRC 校验原理及实现
14. 基于 CPLD 的 UART 设计
15. IPMI 在 VPX 系统中的应用与设计
16. 基于 CPCI 总线的 PMC 载板设计
17. 基于 VPX 总线的工件台运动控制系统研究与开发
18. PCI Express 流控机制的研究与实现
19. UART16C554 的设计
20. 基于 VPX 的高性能计算机设计
21. 基于 CAN 总线技术的嵌入式网关设计
22. Visual C 串行通讯控件使用方法与技巧的研究
23. IEEE1588 精密时钟同步关键技术研究
24. GPS 信号发生器射频模块的一种实现方案
25. 基于 CPCI 接口的视频采集卡的设计
26. 基于 VPX 的 3U 信号处理平台的设计
27. 基于 PCI Express 总线 1394b 网络传输系统 WDM 驱动设计
28. AT89C52 单片机与 ARINC429 航空总线接口设计
29. 基于 CPCI 总线多 DSP 系统的高速主机接口设计

VxWorks：

1. 基于 VxWorks 的多任务程序设计
2. 基于 VxWorks 的数据采集存储装置设计
3. Flash 文件系统分析及其在 VxWorks 中的实现
4. VxWorks 多任务编程中的异常研究
5. VxWorks 应用技巧两例
6. 一种基于 VxWorks 的飞行仿真实时管理系统
7. 在 VxWorks 系统中使用 TrueType 字库
8. 基于 FreeType 的 VxWorks 中文显示方案
9. 基于 Tilcon 的 VxWorks 简单动画开发
10. 基于 Tilcon 的某武器显控系统界面设计
11. 基于 Tilcon 的综合导航信息处理装置界面设计
12. VxWorks 的内存配置和管理
13. 基于 VxWorks 系统的 PCI 配置与应用
14. 基于 MPC8270 的 VxWorks BSP 的移植
15. Bootrom 功能改进经验谈
16. 基于 VxWorks 嵌入式系统的中文平台研究与实现
17. VxBus 的 A429 接口驱动
18. 基于 VxBus 和 MPC8569E 千兆网驱动开发和实现
19. 一种基于 vxBus 的 PPC 与 FPGA 高速互联的驱动设计方法
20. 基于 VxBus 的设备驱动开发
21. 基于 VxBus 的驱动程序架构分析
22. 基于 VxBus 的高速数据采集卡驱动程序开发

Linux：

1. Linux 程序设计第三版及源代码
2. NAND FLASH 文件系统的设计与实现
3. 多通道串行通信设备的 Linux 驱动程序实现
4. Zsh 开发指南-数组
5. 常用 GDB 命令中文速览
6. 嵌入式 C 进阶之道
7. Linux 串口编程实例
8. 基于 Yocto Project 的嵌入式应用设计
9. Android 应用的反编译
10. 基于 Android 行为的加密应用系统研究
11. 嵌入式 Linux 系统移植步步通
12. 嵌入式 CC++语言精华文章集锦
13. 基于 Linux 的高性能服务器端的设计与研究
14. S3C6410 移植 Android 内核

15. Android 开发指南中文版
16. 图解 Linux 操作系统架构设计与实现原理（第二版）
17. 如何在 Ubuntu 和 Linux Mint 下轻松升级 Linux 内核
18. Android 简单 mp3 播放器源码
19. 嵌入式 Linux 系统实时性的研究
20. Android 嵌入式系统架构及内核浅析
21. 基于嵌入式 Linux 操作系统内核实时性的改进方法研究
22. Linux TCP IP 协议详解
23. Linux 桌面环境下内存去重技术的研究与实现

Windows CE：

1. Windows CE.NET 下 YAFFS 文件系统 NAND Flash 驱动程序设计
2. Windows CE 的 CAN 总线驱动程序设计
3. 基于 Windows CE.NET 的 ADC 驱动程序实现与应用的研究
4. 基于 Windows CE.NET 平台的串行通信实现
5. 基于 Windows CE.NET 下的 GPRS 模块的研究与开发
6. win2k 下 NTFS 分区用 ntldr 加载进 dos 源代码
7. Windows 下的 USB 设备驱动程序开发
8. WinCE 的大容量程控数据传输解决方案设计
9. WinCE6.0 安装开发详解
10. DOS 下仿 Windows 的自带计算器程序 C 源码
11. G726 局域网语音通话程序和源代码
12. WinCE 主板加载第三方驱动程序的方法
13. WinCE 下的注册表编辑程序和源代码
14. WinCE 串口通信源代码
15. WINCE 的 SD 卡程序[可实现读写的源码]
16. 基于 WinCE 的 BootLoader 研究

PowerPC：

1. Freescale MPC8536 开发板原理图
2. 基于 MPC8548E 的固件设计
3. 基于 MPC8548E 的嵌入式数据处理系统设计
4. 基于 PowerPC 嵌入式网络通信平台的实现
5. PowerPC 在车辆显控系统中的应用

6. 基于 PowerPC 的单板计算机的设计
7. 用 PowerPC860 实现 FPGA 配置
8. 基于 MPC8247 嵌入式电力交换系统的设计与实现
9. 基于设备树的 MPC8247 嵌入式 Linux 系统开发
10. 基于 MPC8313E 嵌入式系统 UBoot 的移植
11. 基于 PowerPC 处理器 SMP 系统的 UBoot 移植

ARM：

1. 基于 DiskOnChip 2000 的驱动程序设计及应用
2. 基于 ARM 体系的 PC-104 总线设计
3. 基于 ARM 的嵌入式系统中断处理机制研究
4. 设计 ARM 的中断处理
5. 基于 ARM 的数据采集系统并行总线的驱动设计
6. S3C2410 下的 TFT LCD 驱动源码
7. STM32 SD 卡移植 FATFS 文件系统源码
8. STM32 ADC 多通道源码
9. ARM Linux 在 EP7312 上的移植
10. ARM 经典 300 问
11. 基于 S5PV210 的频谱监测设备嵌入式系统设计与实现
12. Uboot 中 start.S 源码的指令级的详尽解析
13. 基于 ARM9 的嵌入式 Zigbee 网关设计与实现
14. 基于 S3C6410 处理器的嵌入式 Linux 系统移植
15. CortexA8 平台的μC-OSⅡ及 LwIP 协议栈的移植与实现

Hardware：

1. DSP 电源的典型设计
2. 高频脉冲电源设计
3. 电源的综合保护设计
4. 任意波形电源的设计
5. 高速 PCB 信号完整性分析及应用
6. DM642 高速图像采集系统的电磁干扰设计
7. 使用 COMExpress Nano 工控板实现 IP 调度设备
8. 基于 COM Express 架构的数据记录仪的设计与实现
9. 基于 COM Express 的信号系统逻辑运算单元设计

10. 基于 COM Express 的回波预处理模块设计
11. 基于 X86 平台的简单多任务内核的分析与实现
12. 基于 UEFI Shell 的 PreOS Application 的开发与研究