UEFI Bootkit 模型与分析

唐文彬[1]　陈　熹[1]　陈嘉勇[1,2]　祝跃飞[1]
（解放军信息工程大学信息工程学院　郑州 450002）[1]
（中国科学院信息安全国家重点实验室　北京 100049）[2]

摘　要　分析了 UEFI Bootkit 的工作原理和关键技术；在 Harold 木马模型的基础上，给出了 UEFI Bootkit 的形式化描述；分析了 UEFI Bootkit 和木马在隐蔽技术方面的差异，建立了 UEFI Bootkit 协同隐藏的形式化模型；给出了模型的一个应用实例，理论证明了在操作系统内核启动前检测 Bootkit 比在操作系统启动完成后检测具有更好的效果；开发了一套在操作系统内核加载前就开始检测的 UEFI Bootkit 检测系统；使用检测系统进行了实际的测试，结果表明，UEFI Bootkit 检测系统具有较好的检测效果，有效地验证了模型的准确性。

关键词　UEFI，形式化，Bootkit，隐蔽技术，可信计算，检测系统
中图法分类号　TP309.5　　**文献标识码**　A

Analysis and Detection of UEFI Bootkit

TANG Wen-bin[1]　CHEN Xi[1]　CHEN Jia-yong[1,2]　ZHU Yue-fei[1]
(Information Engineering Institute, PLA Information Engineering University, Zhengzhou 450002, China)[1]
(State Key Laboratory of Information Security, Graduate University of Chinese Academy of Science, Beijing 100049, China)[2]

Abstract　This paper analyzed the work mechanism and key technology of UEFI Bootkit, expanded the definition of Trojan according to it, illustrated the differences of hiding technology between UEFI Bootkit and Trojan, built a formal model of UEFI Bootkit cooperative concealment, showed an application of the model, proved the idea that detecting Bootkit before the operating system kernel starting can obtain a better effect than after the operating system starting. We designed and developed UEFI Bootkit detection system which works before the operating system kernel starts. The detection system was used to do practical test, and the results show UEFI Bootkit detection system obtains a good effect and has the accuracy.

Keywords　UEFI, Formal description, Bootkit, Hiding technology, Trusted computing, Detection system

1　引言

随着信息技术的日益普及，恶意代码[1,2]受到了越来越广泛的关注。恶意代码技术的迅速发展，给信息安全领域不断带来新的挑战。Bootkit 是恶意代码技术较为高级的阶段。可以认为，所有在开机时比 Windows 内核更早加载，实现内核劫持的恶意代码，都能够称为 Bootkit。

关于 Bootkit 最早的研究源于 2005 年 eEye Digital Security 公司的研究人员发布的“BootRoot”项目[3]，项目中详细介绍了如何利用 Windows 启动过程侵入系统内核，通过感染主引导记录（MBR）的方式，实现绕过内核检查和启动隐身。2007 年 Black Hat 欧洲大会上，Nitin Kumar 和 Vipin Kumar 公布了 VBootKit，实现了利用启动过程入侵 Vista 内核。2009 年，Peter Kleissner 在 Black Hat 会议上发布 Stoned Bootkit，它具有较强的通用性，能够在多个版本的 Windows 操作系统上运行。

随着计算机硬件的快速发展，传统的 BIOS 逐渐成为瓶颈，其已经很难适应现代计算机硬件的发展需求。针对传统 BIOS 的弊端，Intel 公司提出了可扩展固件接口（Extensible Firmware Interface，EFI），以替代传统的 BIOS。UEFI 采用了全新的架构，其启动流程完全不同于传统的 BIOS，在 UEFI 体系结构下，传统的 Bootkit 已经不再适用。

随着 UEFI BIOS 的普及，其安全性也逐渐受到学者的关注。Next-Generation 公司的安全研究员 John Heasman 在 2007 年指出了针对 EFI 攻击的几种可能方法[4]，但并未给出具体细节。2009 年，Rafal Wojtczuk 和 Alexander Tereshkin 指出了 EFI BIOS Q45 系列主板上的一个关于开机画面像素计算上的漏洞[5]。2010 年，林伟[6]设计并实现了一个基于 UEFI 的 Bootkit，该 Bootkit 能够感染 bootmgfw.efi 文件，绕过 PatchGuard 保护机制，在操作系统启动过程中实现内核劫持。

目前已有的基于 UEFI 的 Bootkit 分析和检测工作主要

到稿日期：2011-12-21　返修日期：2012-03-21　本文受郑州市科技创新团队（10CXTD150）资助。
唐文彬（1989－），男，硕士生，主要研究方向为信息安全；陈　熹（1988－），男，硕士生，主要研究方向为信息安全；陈嘉勇（1982－），男，博士生，主要研究方向为信息安全；祝跃飞（1962－），男，教授，博士生导师，主要研究方向为信息安全。

有的 UEFI Bootkit 样本，归纳总结其工作原理，给出 UEFI Bootkit 的形式化描述；分析 UEFI Bootkit 在隐藏性方面的新特点，在此基础上建立 UEFI Bootkit 的协同隐藏模型，给出了模型的一个应用实例；理论证明在操作系统内核启动前检测 Bootkit 比在操作系统启动完成后检测具有更好的效果，依据结论开发了一套在操作系统内核启动前就开始检测的 UEFI Bootkit 检测系统。通过实验验证，在操作系统内核启动前检测具有较好的效果，有效地说明了模型的准确性。

本文第 2 节分析 UEFI Bootkit 的工作原理；第 3 节给出 UEFI Bootkit 的形式化定义，建立 UEFI Bootkit 协同隐藏模型；第 4 节给出基于 UEFI 的 Bootkit 模型的一个应用实例；最后进行总结。

2 UEFI Bootkit 工作原理分析

基于 UEFI 的 Bootkit 先于操作系统内核启动，并且要在操作系统启动完成之后获得内核态权限，因此，Bootkit 必须将代码嵌入到操作系统启动流程中，并通过控制权传递与回收实现在系统启动完成后获得内核态权限的目标。Bootkit 可以在内核加载之前的所有启动组件中嵌入代码，包括 UEFI BIOS 和 OS Loader。Bootkit 传递和回收控制权主要有 Hook ExitBootServices 和逐级 Hook 两种方式。以 Windows 7 操作系统为例，分别根据两种不同的 Hook 方式分析 Bootkit 的工作原理。

2.1 越级 Hook 方式

UEFI 提供了一套完备的启动时服务(boot service)和运行时服务(runtime service)给操作系统设计者使用，ExitBootServices 便是启动时服务中的一个重要函数。当 OS Loader 把操作系统内核加载进内存之后，会调用 ExitBootServices 来停止启动时服务。然后，OS Loader 将控制权转交给内核。通过反汇编的方法，得到具体实现代码为：

```
kernel_load(kernel_image,kname,&kd,&imem,&mmem);
close_devices();
status=BS->ExitBootServices(kernel_image,cookie);
start_kernel(kd.kentry,bp);
```

从 OS Loader 转交控制权给内核的实现代码可以看到，只要对启动时服务 ExitBootServices 进行 Hook，就可以在操作系统内核执行前再次获得控制权，从而成功劫持系统内核。以在 UEFI BIOS 中嵌入代码为例，越级 Hook 方式的 Bootkit 工作原理如图 1 所示。

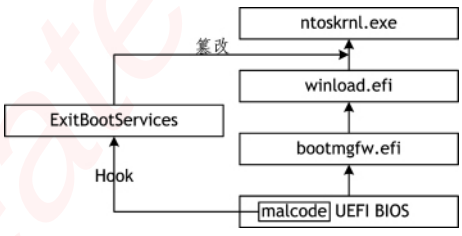

图 1 越级 Hook 工作原理

从图中可以看出，首先 Bootkit 通过刷写固件的方式将其 ExitBootServices 函数进行 Hook，当 Hook 完成之后将控制权重新转交给正常的启动流程，操作系统按照正常流程继续启动。启动到 winload.efi 阶段时，winload.efi 将内核加载进内存，然后调用 ExitbootServices 函数停止启动时服务，这时控制权回收到 Bootkit 中，Bootkit 可以任意篡改内核实现系统启动完成后获得内核态权限的目的。

2.2 逐级 Hook 方式

逐级 Hook 方式的 Bootkit 需要针对每一个启动组件都进行 Hook。以在 UEFI BIOS 中嵌入代码为例，Bootkit 在 bootmgfw.efi 加载进内存后对其 Hook，然后将控制权转交给正常启动流程，在 bootmgfw.efi 的 Hook 点执行时，再对 winload.efi 进行 Hook，之后将控制权再次转交给正常启动流程，当 winload.efi 的 Hook 点执行时，就可以篡改内核，达到获得内核态权限的目的。逐级 Hook 方式的 Bootkit 工作原理如图 2 所示。

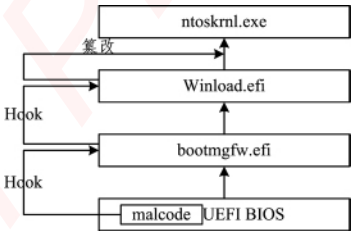

图 2 逐级 Hook 工作原理

由于逐级 Hook 需要对启动的每一个阶段都进行 Hook，因此其通用性与越级 Hook 方式相比较差。两种方式的共同点在于都需要篡改内核，来实现在操作系统启动后获得内核态权限的功能。

3 基于 UEFI 的 Bootkit 模型

3.1 木马的扩展模型

为了深入研究基于 UEFI 的 Bootkit，剖析其隐藏技术的本质，参照 Harold Thimbleby 的木马模型框架[7]，给出木马扩展模型的形式化描述。首先给出如下定义。

定义 1 操作系统内核启动后计算机的状态、程序文本和映像等统称为启动后表示，记为 R。$r\in R$，r 是有限的。

定义 2 操作系统内核启动前计算机的状态、程序文本和映像等统称为启动前表示，记为 R'。$r'\in R'$，r' 是有限的。

定义 3 $E:R\to(L\downarrow R)$，E 为表示到环境的映射。在此，通常理解为计算机的系统配置。$L\downarrow R$ 是名字到表示的部分映射，L 是名字的可数集。用户通常关心的是名字，如程序名和文件名，具体的名字属于集合 L。$L\downarrow R$ 的含义为：如果名字有定义，那么就利用名字获取其相应的表示。

定义 4 $E(r)$的域：names $r=\mathrm{dom}\ E(r)$，为可计算的和有限的，其中包含一些独立于 r 的名字。

定义 5 程序的含义是指程序在运行过程中完成了什么工作，用$[p]$表示。如果 $r\in R$，对于 r 中的程序 p，$[p]$的含义用$[\cdot]:R\to(R\to R)$表示。程序的运行将使一种表示变化为另一种表示，如系统的状态在程序的作用下变为另一种状态。

定义 6 两个程序 p 和 p' 相等是指：

$\forall r\in R,[p]r=[p']r$

由于判定两个程序相等是不可计算的，因此引入程序相似关系和大多数量词 M。

定义 7 考虑检测程序输出的过程所耗费的时间，定义 R 上小于线性时间可计算的关系为相似，用～表示。相似关系是非传递的。

定义 8 如果两个程序 p 和 p' 对大多数的输入能够产生相似结果，则认为它们是不可区分的，记作 $p\approx p'$，即：

$p\approx p'$ iff $M\, r\in R:[p]r\sim[p']r$

为方便讨论，本文再引入一些符号。

$\hat{p}$：表示 p 被木马化后的程序。

$r\xrightarrow{l}r''$：

$r\xrightarrow{l}r''$ iff $l\in$ names r and $r''=[E(r)l]r$

可以自然扩展到 $l_1,l_2,\cdots,l_n\in L^*$ 的有限序列。

定义 9 木马关系：

$ptrojan\ \hat{p}\Leftrightarrow\exists\, r\in R,[p]r\not\sim[\hat{p}]r$

该定义说明如果程序 p 和 $\hat{p}$ 存在木马关系，那么至少存在一种能够将它们区分的表示。

定义 10 木马方法是一个递归的、非空的、可枚举的关系 $T\subseteq R\times R\times R'\times R'\times L$，如果 $\langle r,\hat{r},r',\hat{r}',l\rangle\in T$，则有：

$\wedge\, r\sim\hat{r}$

$\wedge\, E(r)l\approx E(\hat{r})l$

$\wedge\, M\, t\in L^*$：

$\wedge\,[E(r)l]r\xrightarrow{t}r''$

$\wedge\,[E(\hat{r})l]\hat{r}\xrightarrow{t}\hat{r}''$

$\wedge\, r''\not\sim\hat{r}''$

该定义说明如果 $\langle r,\hat{r},r',\hat{r}',l\rangle\in T$ 是一个木马方法 T，操作系统启动后 $\hat{r}$ 拥有与 r 类似的环境，其中存在程序 l，对潜在的大多数输入参数来说，l 在两个环境中执行的情况是一样的，但最终会表现出不一致，那么这个 l 就是木马程序。

该模型将原来的木马定义扩展到 5 维，增加了操作系统启动前的表示。由上述模型可以看出：木马的定义重点关注于植入木马后系统环境会产生变化以及产生何种变化，而对于木马怎样使系统环境发生变化却没有涉及。同时，木马不会影响内核启动前各启动组件的状态，即木马都是在操作系统内核加载后才会执行。

3.2 基于 UEFI 的 Bootkit 的形式化描述

木马是在操作系统启动之后加载运行的，而基于 UEFI 的 Bootkit 在操作系统内核启动之前就已经获得控制权，并通过控制权传递，在系统启动完成后执行木马的功能。也就是说，在操作系统启动完成后 Bootkit 可以使用木马模型进行刻画，此时只需在操作系统内核启动之前对其进行定义即可。

定义 11 计算机程序 p 由程序片段 $a_1,a_2,\cdots,a_n$ 组成，

定义 12 计算机操作系统启动平台用 S 表示，平台由各个组件组成，一个组件可能由更小的组件组成，最小规模的组件可以是一段可执行的程序代码。Bootkit 化后的操作系统启动平台用 $\hat{S}$ 表示。

定义 13 控制权传递关系可以用一个二元关系表示：$\rightarrow\in S\times S,A_i\rightarrow A_j$ 表示组件 A_i 将控制权传递给 A_j，其中 A_n 表示操作系统内核获得控制权。因此控制权传递可以用下面的式子表示：

$A_1\rightarrow A_2\rightarrow\cdots\rightarrow A_n,A_i\subset S,A_i$ 启动状态的表示为 r_i'

如果两个表示 r 和 $\hat{r}$ 是木马关系而且在操作系统启动之前就获得执行，并具有传递和回收控制权的功能，那么 $r,\hat{r}$ 是名字 l 上的 Bootkit 关系。其具体定义如下。

定义 14 Bootkit 方法是一个木马方法 $V\subseteq R\times R\times R'\times R'\times L$ 并满足附加的条件，如果 $\langle r,\hat{r},r',\hat{r}',l\rangle\in V$ 则有：

$\wedge\, r\sim\hat{r}$

$\wedge\, E(r)l\approx E(\hat{r})l$

$\wedge\, M\, t\in L^*$：

$\wedge\,[E(r)l]r\xrightarrow{t}r''$

$\wedge\,[E(\hat{r})l]\hat{r}\xrightarrow{t}\hat{r}''$

$\wedge\, r''\not\sim\hat{r}''$

$\wedge\, a_0\in l$

$\exists\, i\in[1,n)$：

$\wedge\, A_i\rightarrow a_0$

$\wedge\, M\, t\in L^*$：

$\wedge\,[E(r_n')l]r_n'\xrightarrow{t}r_n''$

$\wedge\,[E(\hat{r}_n')l]\hat{r}_n\xrightarrow{t}\hat{r}_n''$

$\wedge\, r_n''\not\sim\hat{r}_n''$

该定义说明基于 UEFI 的 Bootkit 在操作系统启动之后其实质是一个木马程序。同时，Bootkit 程序 l 将程序片段 a_0 嵌入到操作系统启动过程中执行，即 Bootkit 在操作系统内核启动前就获得执行权。在内核启动前对于大多数输入来说，Bootkit 在两个环境的执行情况是一样的，但是最终会在操作系统内核获得控制权时表现出不一致。

3.3 基于 UEFI 的 Bootkit 隐藏技术

传统的木马在植入目标系统之后，会采取各种技术隐藏自身，以避免被发现。张新宇等人在文献[8]中提出了木马协同隐藏的思想，并将木马的隐藏技术分为 3 类，分别为本地隐藏、通信隐藏和协同隐藏，较好地总结了现有的木马隐藏技术。但是，对于协同隐藏中一个重要的因素－启动隐藏，文献中却并未涉及，同时由于 Bootkit 技术在当时还并未出现，因此文献[8]对文件隐藏的概括还不够全面。作者在文献[8]的基础上，总结了木马的启动隐藏技术，对比了 Bootkit 和木马在启动隐藏和文件隐藏的差异，给出了 Bootkit 协同隐藏的形式化描述。

序。2)修改系统服务,设置为通过服务加载。3)感染正常的启动程序,如 explorer.exe,随正常程序启动。木马的启动时机一般滞后于操作系统内核,被操作系统内核直接或间接加载执行。

Bootkit 在操作系统启动之后可以用木马刻画,因此,其大部分隐藏技术都与木马相似。它与木马隐藏技术的差异主要表现在启动隐藏和文件隐藏方面,如表 1 所列。从表中可以看到,在启动隐藏和文件隐藏方面,Bootkit 比木马具有更高的隐蔽性。

表 1 Bootit 与木马隐藏技术对比

	木马	Bootkit
启动隐藏	修改注册表,设置为自启动或服务启动,启动时机滞后于操作系统内核	插入或篡改启动组件如 UEFI BIOS,启动时机先于操作系统内核
文件隐藏	一般通过修改系统 API 的正常功能实现隐藏,会在文件系统下驻留文件	存放在固件或 EFI 分区中,在文件系统下一般不留痕迹

参考张新宇的木马协同隐藏模型,对 UEFI 下 Bootkit 的协同隐藏进行形式化描述。以 $\hat{p}$ 表示 Bootkit 的主程序,在操作系统启动完成后包含 $\hat{u}_1,\hat{u}_2,\cdots,\hat{u}_k$ 若干个子程序,因此 $\hat{u}_1,\hat{u}_2,\cdots,\hat{u}_k$ 都具有木马的属性。在操作系统下,每个子程序 $\hat{u}_i$ ($i\in[1,k]$)隐藏某一类属性,它们共同作用隐蔽 $\hat{p}$ 的特征。

设 $r,r',r'',\hat{r},\hat{r}',\hat{r}'',r_i,r_i',\hat{r}_i,\hat{r}_i',r_i'',\hat{r}_i''\in R,\hat{u}_i,i\in[1,k],m$ 为 $\hat{p}$ 属性的类数,$k\geqslant m$,则

$$\wedge r_i\sim\hat{r}_i \tag{1}$$

$$\wedge E(r_i)\hat{u}_i\approx E(\hat{r}_i)\hat{u}_i \tag{2}$$

$$\wedge Mt\in L^*$$
$$\wedge[E(r_i)\hat{u}_i]r_i\xrightarrow{t}r_i''$$
$$\wedge[E(\hat{r}_i)\hat{u}_i]\hat{r}_i\xrightarrow{t}\hat{r}_i'' \tag{3}$$
$$\wedge r_i''\not\sim\hat{r}_i''$$

$$\wedge r\sim\hat{r} \tag{4}$$

$$\wedge\hat{u}_i\in L^* \tag{5}$$
$$\wedge[E(r)\hat{p}]r\xrightarrow{\hat{u}_1,\hat{u}_2,\cdots,\hat{u}_i}r''$$
$$\wedge[E(\hat{r})\hat{p}]\hat{r}\xrightarrow{\hat{u}_1,\hat{u}_2,\cdots,\hat{u}_i}\hat{r}''$$

$$\begin{cases}\wedge r''\not\sim\hat{r}'', & 1\leqslant i<m\\ \wedge r''\sim\hat{r}'', & i\geqslant m\end{cases},\hat{u}_i\text{ 协同隐藏 }\hat{p} \tag{6}$$

$$\wedge a_0\in l \tag{7}$$

$$\exists i\in[1,n):$$
$$\wedge A_i\rightarrow a_0 \tag{8}$$
$$\wedge Mt\in L^*$$
$$\wedge[E(r_j')\hat{p}]r_j'\xrightarrow{t}r_j''$$
$$\wedge[E(\hat{r}_j')\hat{p}]\hat{r}_j'\xrightarrow{t}\hat{r}_j''$$

$$\begin{cases}\wedge r_j''\sim r_j''\text{ or }\wedge r_j''\not\sim r_j'', & 1\leqslant j<n\\ \wedge r_j''\not\sim\hat{r}_j'', & j=n\end{cases} \tag{9}$$

其中式(1)—式(6)表示 Bootkit 在操作系统启动完成后的协同隐藏,即木马的协同隐藏;式(7)和式(8)表示 Bootkit 在操作系统内核启动前就获得控制权;式(9)表示 Bootkit 在操作系统内核启动前的协同隐藏。

由 Bootkit 协同隐藏的形式化描述可以看出,在操作系统启动完成后,如果子程序可以隐藏 Bootkit 主程序具有的所有属性即 $i\geqslant m$,那么检测程序就无法将 $\hat{p}$ 与正常程序 p 区别开来。在操作系统内核加载之前,对于大多数输入参数而言,检测程序能以一定的概率将 $\hat{p}$ 与正常程序 p 区别开来,但是当操作系统内核获得控制权时则会以概率 1 表现出不一致。

4 基于 UEFI 的 Bootkit 模型的应用

基于 UEFI 的 Bootkit 在操作系统启动完成后一般都具有内核态的权限,为其隐藏提供了便利,因此具有较强的隐蔽性,一般的检测程序都难以将其检测出来。同时,Bootkit 先于检测程序启动,可以直接破坏检测程序正常功能的执行,实现“隐身”。

4.1 Bootkit 检测判定定理

目前,对于 Bootkit 检测大多都是在启动之后,由于 Bootkit 在启动后存在协同隐藏,可能导致检测程序无法将其与正常程序区别开来,检测效果不好。于是,在 UEFI 的 Bootkit 模型的基础上,本文给出基于 UEFI 的 Bootkit 检测判定定理。

定理 1 Bootkit 在操作系统启动完成后的检出率为 α,在操作系统内核启动前的检出率为 β,则 $\alpha\leqslant\beta$。

证明:1)如果 l 为 Bootkit 主程序,且能够在操作系统启动完成后被检出,则 l 在操作系统内核启动前能够被检出。

$\forall l$,l 是 Bootkit 主程序,且 l 在操作系统下能够被检测程序检出,那么根据协同隐藏模型的式(6)有:

$$\wedge\hat{u}_i\in L^*$$
$$\wedge[E(r)l]r\xrightarrow{\hat{u}_i}r''$$
$$\wedge[E(\hat{r})l]\hat{r}\xrightarrow{\hat{u}_i}\hat{r}''$$
$$\wedge r''\not\sim\hat{r}''$$

同时,根据式(9),在操作系统内核启动前有:

$$\wedge Mt\in L^*$$
$$\wedge[E(r_j')l]r_j'\xrightarrow{t}r_j''$$
$$\wedge[E(\hat{r}_j')l]\hat{r}_j'\xrightarrow{t}\hat{r}_j''$$
$$\wedge r_n''\not\sim\hat{r}_n''$$

即 l 也能在操作系统内核启动前被检测程序检出。

2)存在 l 为 Bootkit 主程序,不能够在操作系统启动完成后被检出,但是 l 在操作系统内核启动前能够被检出。

l 是 Bootkit 主程序,取 $i=m$,根据式(6),在操作系统启动完成后有:

$$\wedge\hat{u}_i\in L^*$$

$$\wedge[E(r)l]r \xrightarrow{u_i} r''$$

$$\wedge[E(\hat{r})l]\hat{r} \xrightarrow{\hat{u}_i} \hat{r}''$$

$$\wedge r'' \sim \hat{r}''$$

即 l 在操作系统下不能被检测程序检出。但是根据式(9),在操作系统内核启动前有:

$$\wedge Mt \in L^*$$

$$\wedge[E(r_j')l]r_j' \xrightarrow{t} r_j''$$

$$\wedge[E(\hat{r}_j')l]\hat{r}_j' \xrightarrow{t} \hat{r}_j''$$

$$\wedge r_n'' \not\sim \hat{r}_n''$$

即 l 能在操作系统内核启动前被检测程序检出。

综合 1)和 2),有 $\alpha \leqslant \beta$,证毕。

由 Bootkit 协同隐藏模型可以看到,即使 Bootkit 采用协同隐藏技术进行隐藏,在操作系统内核获得控制权时还是会表现出不一致,其一定能被检出,即 $\beta=1$。

4.2 基于 UEFI 的 Bootkit 检测实验与结果分析

由定理 1 可知,在操作系统内核启动前检测 Bootkit 将具有更好的效果。因此,采用在操作系统内核启动前进行检测的方法,参考可信计算的思想[9-11]开发了一个基于完整性校验的 UEFI Bootkit 检测系统。检测系统存储在光盘中,该系统能够在内核启动前对各个启动组件进行完整性校验,系统整体框架如图 3 所示。

图 3 检测系统整体框架

由图 3 可以看出,②、④、⑥为系统启动的执行流程,即控制权传递过程为:UEFIBIOS→检测系统→OSLoader→OS Kernel。①、③、⑤为系统的校验流,依次对各个启动组件进行可信度量。具体的过程为:首先,设置启动选项为光盘启动,则 UEFI BIOS 执行完成之后将自动加载检测系统。这时,检测系统加载 OS Loader,并判断 digist(detect,OS Loader)=expect(OS Loader) 是否成立,如果不成立,则系统被感染 Bootkit;如果成立,则 Hook OS Loader,并将控制权转交给 OS Loader。当 OS Loader 加载完 OS Kernel,执行到 Hook 点时,代码执行权限回收到检测系统中,检测系统判断 digist(detect,OS Kernel)=expect(OS Kernel)是否成立,如果不成立,则系统被感染 Bootkit;如果成立,则系统未感染 Bootkit。其中 digist(A,B)表示组件 A 对组件 B 进行摘要运算的结果,detect 表示检测系统,expect(A)表示组件 A 哈希的预期值。

由基于 UEFI 的 Bootkit 模型可知,对于感染 Bootkit 的主机,在操作系统内核获得控制权时一定会表现出不一致,即必然会存在 digist(detect,A_n)≠expect(A_n),因此,一定可以被检测系统检出。利用 Bootkit 检测系统分别对 UEFI Bootkit 进行检测,结果如表 2 所列。其中 desert.exe[12] 是插入到 UEFI 是插入到 UEFI BIOS 中的 Bootkit,采用的是逐级 Hook 方式;linbot.exe[6] 是插入到 bootmgfw.efi 中的 Bootkit,采用的是逐级 Hook 方式。从表中可以看到,开发的检测系统能够有效地检测 Bootkit,验证了基于 UEFI 的 Bootkit 模型的准确性。

表 2 UEFI Bootkit 检测结果

样本名称	代码插入位置	Hook 方式	检测结果
desert.exe	UEFI BIOS	越级 Hook	检测成功
cylon.exe	UEFI BIOS	逐级 Hook	检测成功
linbot.exe	bootmgfw.efi	逐级 Hook	检测成功

结束语 本文基于 UEFI Bootkit 的原理,分析了 UEFI Bootkit实现的关键技术。然后借鉴 Harold Thimbleby 的木马模型,给出了 UEFI Bootkit 定义的形式化描述。在 UEFI Bootkit 定义的基础上,讨论了 UEFI Bootkit 和木马在隐蔽技术上的差异,建立了 UEFI Bootkit 协同隐藏模型,并给出了模型的一个应用,证明了在操作系统内核启动前检测 Bootkit 比操作系统启动完成后检测具有更好的效果。根据定理开发了一套在操作系统内核启动前就开始检测的 UEFI Bootkit 检测系统,实验结果表明,系统具有较好的检测效果,有效地验证了模型的准确性。

参 考 文 献

[1] Cadar C, Genesh V, Pawlowski P M, et al. EXE: Automatically generating inputs of death[C]//CCS'06 Proceedings of the 13th ACM Conference on Computer and Communications Security. 2006:322-335

[2] Levine J F, Grizzard J B, Owen H L. Detecting and categorizing kernel—level rootkits to aid future detection[J]. IEEE Security and Privacy, 2006, 4(1):27-32

[3] Soeder D, Permeh R. Eeye BootRoot[EB/OL]. http://research.eeye.com/html/tools/RT20060801-7.html, 2005-09

[4] Heasman J. Implementing and Detecting an ACPI BIOS Rootkit [EB/OL]. http://www.blackhat.com/presentations/bu-usa-07/Heasman/Presentation/bh-usa-07Heasman.pdf, 2007-02

[5] Wojtczuk R, Tereshkin A. Attacking Intel BIOS[EB/OL]. http://www.blackhat.com/presentations/bh-usa-09/WOJTCZUK/BHUSA09-Wojtczuk-AtkIntelBios-SLIDES.pdf, 2009-08

[6] 林伟. 基于 UEFI BIOS 的 Bootkit 技术研究[D]. 郑州:信息工程大学, 2010

[7] Thimbleby H, Anderson S, Cairns P. A framework for medelling trojans and computer virus infection[J]. The Computer Journal, 1998, 41(7):444-458

[8] 张新宇,卿斯汉,马恒太,等. 特洛伊木马隐藏技术研究[J]. 通信学报, 2004, 25(7):153-159

[9] 沈昌祥,张焕国,冯登国. 信息安全综述[J]. 中国科学 E 辑:信息科学, 2007, 37(1):129-150

[10] 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, 48(8):1332-1349

[11] 黄强. 基于可信计算的终端安全体系结构研究[D]. 武汉:海军工程大学, 2007

[12] 朱瑜. Bootkit 检测技术研究[D]. 郑州:信息工程大学, 2010

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
30. 总线协议中的 CRC 及其在 SATA 通信技术中的应用
31. 基于 FPGA 的 SATA 硬盘加解密控制器设计
32. Modbus 协议在串口通讯中的研究及应用
33. 高可用的磁盘阵列 Cache 的设计和实现
34. RAID 阵列中高速 Cache 管理的优化

35. 一种新的基于 RAID 的 CACHE 技术研究与实现
36. 基于 PCIE-104 总线的高速数据接口设计
37. 基于 VPX 标准的 RapidIO 交换和 Flash 存储模块设计
38. 北斗卫星系统在海洋工程中的应用
39. 北斗卫星系统在远洋船舶上应用的研究
40. 基于 CPCI 总线的红外实时信号处理系统
41. 硬件实现 RAID 与软件实现 RAID 的比较
42. 基于 PCI Express 总线系统的热插拔设计
43. 基于 RAID5 的磁盘阵列 Cache 的研究与实现
44. 基于 PCI 总线的 MPEG2 码流播放卡驱动程序开发
45. 基于磁盘异或引擎的 RAID5 小写性能优化
46. 基于 IEEE1588 的时钟同步技术研究
47. 基于 Davinci 平台的 SD 卡读写优化
48. 基于 PCI 总线的图像处理及传输系统的设计
49. 串口和以太网通信技术在油液在线监测系统中的应用
50. USB30 数据传输协议分析及实现
51. IEEE 1588 协议在工业以太网中的实现
52. 基于 USB30 的设备自定义请求实现方法
53. IEEE1588 协议在网络测控系统中的应用
54. USB30 物理层中弹性缓冲的设计与实现
55. USB30 的高速信息传输瓶颈研究
56. 基于 IPv6 的 UDP 通信的实现
57. 一种基于 IPv6 的流媒体传送方案研究与实现
58. 基于 IPv4-IPv6 双栈的 MODBUS-TCP 协议实现
59. RS485CAN 网关设计与实现
60. MVB 周期信息的实时调度
61. RS485 和 PROFINET 网关设计
62. 基于 IPv6 的 Socket 通信的实现
63. MVB 网络重复器的设计
64. 一种新型 MVB 通信板的探究
65. 具有 MVB 接口的输入输出设备的分析
66. 基于 STM32 的 GSM 模块综合应用
67. 基于 ARM7 的 MVB CAN 网关设计
68. 机车车辆的 MVB CAN 总线网关设计
69. 智能变电站冗余网络中 IEEE1588 协议的应用
70. CAN 总线的浅析 CANopen 协议
71. 基于 CANopen 协议实现多电机系统实时控制
72. 以太网时钟同步协议的研究
73. 基于 CANopen 的列车通信网络实现研究
74. 基于 SJA1000 的 CAN 总线智能控制系统设计
75. 基于 CANopen 的运动控制单元的设计
76. 基于 STM32F107VC 的 IEEE 1588 精密时钟同步分析与实现

77. 分布式控制系统精确时钟同步技术
78. 基于 IEEE 1588 的时钟同步技术在分布式系统中应用
79. 基于 SJA1000 的 CAN 总线通讯模块的实现
80. 嵌入式设备的精确时钟同步技术的研究与实现
81. 基于 SJA1000 的 CAN 网桥设计
82. 基于 CAN 总线分布式温室监控系统的设计与实现
83. 基于 DSP 的 CANopen 通讯协议的实现
84. 基于 PCI9656 控制芯片的高速网卡 DMA 设计
85. 基于以太网及串口的数据采集模块设计
86. MVB1 类设备控制器的 FPGA 设计
87. MVB 接口彩色液晶显示诊断单元的显示应用软件设计
88. IPv6 新型套接字的网络编程剖析
89. 基于规则的 IPv4 源程序到 IPv6 源程序的移植方法
90. MVB 网络接口单元的 SOC 解决方案
91. 基于 IPSec 协议的 IPv6 安全研究
92. 具有 VME 总线的车载安全计算机 MVB 通信板卡
93. SD 卡的传输协议和读写程序
94. 基于 SCTP 的 TLS 应用
95. 基于 IPv6 的静态路由实验设计
96. 基于 MVB 的地铁列车司机显示系统研究
97. 基于参数优化批处理的 TLS 协议
98. SSD 数据结构与算法综述

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
23. Vxworks 下的冗余 CAN 通讯模块设计
24. WindML 工业平台下开发 S1d13506 驱动及显示功能的实现
25. WindML 中 Mesa 的应用
26. VxWorks 下图形用户界面开发中双缓冲技术应用
27. VxWorks 上的一种 GUI 系统的设计与实现
28. VxWorks 环境下 socket 的实现
29. VxWorks 的 WindML 图形界面程序的框架分析
30. VxWorks 实时操作系统及其在 PC104 下以太网编程的应用
31. 实时操作系统任务调度策略的研究与设计
32. 军事指挥系统中 VxWorks 下汉字显示技术
33. 基于 VxWorks 实时控制系统中文交互界面开发平台
34. 基于 VxWorks 操作系统的 WindML 图形操控界面实现方法
35. 基于 GPU FPGA 芯片原型的 VxWorks 下驱动软件开发
36. VxWorks 下的多串口卡设计
37. VxWorks 内存管理机制的研究
38. T9 输入法在 Tilcon 下的实现
39. 基于 VxWorks 的 WindML 图形界面开发方法
40. 基于 Tilcon 的 IO 控制板可视化测试软件的设计和实现
41. 基于 VxWorks 的通信服务器实时多任务软件设计
42. 基于 VXWORKS 的 RS485MVB 网关的设计与实现
43. 实时操作系统 VxWorks 在微机保护中的应用
44. 基于 VxWorks 的多任务程序设计及通信管理
45. 基于 Tilcon 的 VxWorks 图形界面开发技术
46. 嵌入式图形系统 Tilcon 及应用研究
47. 基于 VxWorks 的数据采集与重演软件的图形界面的设计与实现
48. 基于嵌入式的 Tilcon 用户图形界面设计与开发
49. 基于 Tilcon 的交互式多页面的设计
50. 基于 Tilcon 的嵌入式系统人机界面开发技术
51. 基于 Tilcon 的指控系统多任务人机交互软件设计
52. 基于 Tilcon 航海标绘台界面设计
53. 基于 Tornado 和 Tilcon 的嵌入式 GIS 图形编辑软件的开发
54. VxWorks 环境下内存文件系统的应用
55. VxWorks 下的多重定时器设计

56. Freescale 的 MPC8641D 的 VxWorks BSP
57. VxWorks 实验五[时间片轮转调度]
58. 解决 VmWare 下下载大型工程.out 出现 WTX Error 0x100de 的问题
59. 基于 VxWorks 系统的 MiniGUI 图形界面开发
60. VxWorks BSP 开发中的 PCI 配置方法
61. VxWorks 在 S3C2410 上的 BSP 设计
62. VxWorks 操作系统中 PCI 总线驱动程序的设计与实现
63. VxWorks 概述
64. 基于 AT91RM9200 的 VxWorks END 网络驱动开发
65. 基于 EBD9200 的 VxWorks BSP 设计和实现
66. 基于 VxWorks 的 BSP 技术分析
67. ARM LPC2210 的 VxWorks BSP 源码
68. 基于 LPC2210 的 VxWorks BSP 移植
69. 基于 VxWorks 平台的 SCTP 协议软件设计实现
70. VxWorks 快速启动的实现方法[上电到应用程序 1 秒]

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
24. 掌握 Android 7.0 新增特性 Quick Settings
25. Android 应用逆向分析方法研究
26. Android 操作系统的课程教学
27. Android 智能手机操作系统的研究
28. Android 英文朗读功能的实现
29. 基于 Yocto 订制嵌入式 Linux 发行版
30. 基于嵌入式 Linux 的网络设备驱动设计与实现
31. 如何高效学习嵌入式
32. 基于 Android 平台的 GPS 定位系统的设计与实现
33. LINUX ARM 下的 USB 驱动开发
34. Linux 下基于 I2C 协议的 RTC 驱动开发
35. 嵌入式下 Linux 系统设备驱动程序的开发
36. 基于嵌入式 Linux 的 SD 卡驱动程序的设计与实现
37. Linux 系统中进程调度策略
38. 嵌入式 Linux 实时性方法
39. 基于实时 Linux 计算机联锁系统实时性分析与改进
40. 基于嵌入式 Linux 下的 USB30 驱动程序开发方法研究
41. Android 手机应用开发之音乐资源播放器
42. Linux 下以太网的 IPv6 隧道技术的实现
43. Research and design of mobile learning platform based on Android
44. 基于 linux 和 Qt 的串口通信调试器调的设计及应用
45. 在 Linux 平台上基于 QT 的动态图像采集系统的设计
46. 基于 Android 平台的医护查房系统的研究与设计
47. 基于 Android 平台的软件自动化监控工具的设计开发
48. 基于 Android 的视频软硬解码及渲染的对比研究与实现
49. 基于 Android 移动设备的加速度传感器技术研究
50. 基于 Android 系统振动测试仪研究
51. 基于缓存竞争优化的 Linux 进程调度策略
52. Linux 基于 W83697 和 W83977 的 UART 串口驱动开发文档
53. 基于 AT91RM9200 的嵌入式 Linux 系统的移植与实现
54. 路由信息协议在 Linux 平台上的实现
55. Linux 下 IPv6 高级路由器的实现

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
17. Windows CE 环境下无线网卡的自动安装
18. 基于 Windows CE 的可视电话的研究与实现
19. 基于 WinCE 的嵌入式图像采集系统设计
20. 基于 ARM 与 WinCE 的掌纹鉴别系统
21. DCOM 协议在网络冗余环境下的应用
22. Windows XP Embedded 在变电站通信管理机中的应用
23. XPE 在多功能显控台上的开发与应用
24. 基于 Windows XP Embedded 的 LKJ2000 仿真系统设计与实现
25. 虚拟仪器的 Windows XP Embedded 操作系统开发
26. 基于 EVC 的嵌入式导航电子地图设计
27. 基于 XPEmbedded 的警务区 SMS 指挥平台的设计与实现
28. 基于 XPE 的数字残币兑换工具开发
29. Windows CENET 下 ADC 驱动开发设计
30. Windows CE 下 USB 设备流驱动开发与设计
31. Windows 驱动程序设计
32. 基于 Windows CE 的 GPS 应用
33. 基于 Windows CE 下大像素图像分块显示算法的研究
34. 基于 Windows CE 的数控软件开发与实现

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
12. 基于 PowerPC 双核处理器嵌入式系统 UBoot 移植
13. 基于 PowerPC 的雷达通用处理机设计
14. PowerPC 平台引导加载程序的移植
15. 基于 PowerPC 嵌入式内核的多串口通信扩展设计
16. 基于 PowerPC 的多网口系统抗干扰设计
17. 基于 MPC860T 与 VxWorks 的图形界面设计
18. 基于 MPC8260 处理器的 PPMC 系统
19. 基于 PowerPC 的控制器研究与设计
20. 基于 PowerPC 的模拟量输入接口扩展
21. 基于 PowerPC 的车载通信系统设计
22. 基于 PowerPC 的嵌入式系统中通用 IO 口的扩展方法
23. 基于 PowerPC440GP 型微控制器的嵌入式系统设计与研究
24. 基于双 PowerPC 7447A 处理器的嵌入式系统硬件设计
25. 基于 PowerPC603e 通用处理模块的设计与实现
26. 嵌入式微机 MPC555 驻留片内监控器的开发与实现
27. 基于 PowerPC 和 DSP 的电能质量在线监测装置的研制
28. 基于 PowerPC 架构多核处理器嵌入式系统硬件设计
29. 基于 PowerPC 的多屏系统设计
30. 基于 PowerPC 的嵌入式 SMP 系统设计
31. 基于 MPC850 的多功能通信管理器
32. 基于 MPC8640D 处理系统的技术研究
33. 基于双核 MPC8641D 处理器的计算机模块设计
34. 基于 MPC8641D 处理器的对称多处理技术研究

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
16. 基于 ARM 的嵌入式 Linux 无线网卡设备驱动设计
17. ARM S3C2440 Linux ADC 驱动
18. ARM S3C2440 Linux 触摸屏驱动
19. Linux 和 Cortex-A8 的视频处理及数字微波传输系统设计
20. Nand Flash 启动模式下的 Uboot 移植
21. 基于 ARM 处理器的 UART 设计
22. ARM CortexM3 处理器故障的分析与处理
23. ARM 微处理器启动和调试浅析
24. 基于 ARM 系统下映像文件的执行与中断运行机制的实现
25. 中断调用方式的 ARM 二次开发接口设计
26. ARM11 嵌入式系统 Linux 下 LCD 的驱动设计
27. Uboot 在 S3C2440 上的移植
28. 基于 ARM11 的嵌入式无线视频终端的设计
29. 基于 S3C6410 的 Uboot 分析与移植
30. 基于 ARM 嵌入式系统的高保真无损音乐播放器设计
31. UBoot 在 Mini6410 上的移植
32. 基于 ARM11 的嵌入式 Linux NAND FLASH 模拟 U 盘挂载分析与实现
33. 基于 ARM11 的电源完整性分析
34. 基于 ARM S3C6410 的 uboot 分析与移植
35. 基于 S5PC100 移动视频监控终端的设计与实现
36. UBoot 在 AT91RM9200 上的移植简析
37. 基于工控级 AT91RM9200 开发板的 UBoot 移植分析

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
13. 基于 UEFI 固件的恶意代码防范技术研究
14. MIPS 架构计算机平台的支持固件研究
15. 基于 UEFI 固件的攻击验证技术研究
16. 基于 UEFI 的 Application 和 Driver 的分析与开发
17. 基于 UEFI 的可信 BIOS 研究与实现
18. 基于 UEFI 的国产计算机平台 BIOS 研究
19. 基于 UEFI 的安全模块设计分析
20. 基于 FPGA NiosⅡ的等精度频率计设计
21. 基于 FPGA 的 SOPC 设计
22. 基于 SOPC 基本信号产生器的设计与实现
23. 基于龙芯平台的 PMON 研究与开发
24. 基于 X86 平台的嵌入式 BIOS 可配置设计
25. 基于龙芯 2F 架构的 PMON 分析与优化
26. CPU 与 GPU 之间接口电路的设计与实现
27. 基于龙芯 1A 平台的 PMON 源码编译和启动分析
28. 基于 PC104 工控机的嵌入式直流监控装置的设计
29. GPGPU 技术研究与发展
30. GPU 实现的高速 FIR 数字滤波算法
31. 一种基于 CPUGPU 异构计算的混合编程模型
32. 面向 OpenCL 模型的 GPU 性能优化
33. 基于 GPU 的 FDTD 算法
34. 基于 GPU 的瑕疵检测
35. 基于 GPU 通用计算的分析与研究
36. 面向 OpenCL 架构的 GPGPU 量化性能模型
37. 基于 OpenCL 的图像积分图算法优化研究
38. 基于 OpenCL 的均值平移算法在多个众核平台的性能优化研究
39. 基于 OpenCL 的异构系统并行编程
40. 嵌入式系统中热备份双机切换技术研究
41. EFI-Tiano 环境下的 AES 算法应用模型
42. EFI 及其安全性研究
43. 基于 UEFI Shell 的 PreOS Application 的开发与研究

Programming:

1. 计算机软件基础数据结构 - 算法
2. 高级数据结构对算法的优化
3. 零基础学算法
4. Linux 环境下基于 TCP 的 Socket 编程浅析
5. Linux 环境下基于 UDP 的 socket 编程浅析
6. 基于 Socket 的网络编程技术及其实现
7. 数据结构考题 - 第 1 章 绪论
8. 数据结构考题 - 第 2 章 线性表
9. 数据结构考题 - 第 2 章 线性表 - 答案
10. 基于小波变换与偏微分方程的图像分解及边缘检测
11. 基于图像能量的布匹瑕疵检测方法
12. 基于 OpenCL 的拉普拉斯图像增强算法优化研究
13. 异构平台上基于 OpenCL 的 FFT 实现与优化
14. 数据结构考题 - 第 4 章 串
15. 数据结构考题 - 第 4 章 串答案
16. 用 IPv6 编程接口实现有连接通信的方法

FPGA / CPLD:

1. 一种基于并行处理器的快速车道线检测系统及 FPGA 实现
2. 基于 FPGA 和 DSP 的 DBF 实现
3. 高速浮点运算单元的 FPGA 实现
4. DLMS 算法的脉动阵结构设计及 FPGA 实现
5. 一种基于 FPGA 的 3DES 加密算法实现
6. 可编程 FIR 滤波器的 FPGA 实现
7. 基于 FPGA 的 AES 加密算法的高速实现
8. 基于 FPGA 的精确时钟同步方法
9. 应用分布式算法在 FPGA 平台实现 FIR 低通滤波器
10. 流水线技术在用 FPGA 实现高速 DSP 运算中的应用
11. 基于 FPGA 的 CAN 总线通信节点设计
12. 基于 FPGA 的高速时钟数据恢复电路的实现
13. 基于 FPGA 的高阶高速 FIR 滤波器设计与实现
14. 基于 FPGA 高效实现 FIR 滤波器的研究

15. FPGA 的 VHDL 设计策略
16. 用 FPGA 实现串口通信的设计
17. GPIB 接口的 FPGA 实现