Android 应用逆向分析方法研究

张志远[1], 万月亮[2], 翁越龙[2], 糜波[2]

（1. 广东省东莞市公安局，广东东莞 523000;2. 北京锐安科技有限公司，北京 100044）

摘　要：Android 移动设备和应用逐渐兴起，对 Android 应用进行逆向分析，有助于分析程序的工作原理，从而查明应用是否有害，是否威胁到设备使用人的信息安全。文章从分析流程、静态分析、动态调试、反逆向手段等几个方面阐明了如何逆向分析一个 Android 应用，以实现应用程序安全可控的目的。

关键词：Android；应用；逆向分析

中图分类号：TP309　　文献标识码：A　　文章编号：1671-1122（2013）06-0065-04

Research on Reverse Analyzing of Android Application

ZHANG Zhi-yuan[1],WAN Yue-liang[2] ,WENG Yue-long[2],MI bo[2]

(1. *Dongguan Municipal Public Security Bureau, Dongguan Guangdong* 523000,*China;*

2. *Run Technolog Co., Ltd.Beijing,Beijing* 100044,*China*)

Abstract: Nowadays, Android portable devices and applications are gradually rising. Reverse analyzing the Android application, we can see how the program runs, whether it is harmful and threatens to the privacy of the owner. This article will expound how to reverse analyze an Android application from analysis flow, static analysis, debug, anti-reverse and other aspects.

Key words: Android; application; reverse analyzing

近年来，使用 Android 操作系统的移动设备逐渐兴起，相应的应用软件也不断出现。除了 Google 官方的 Google Play 外，用户还可以从安卓市场、机锋市场等处下载自己感兴趣的应用。另外，各大网站、论坛也都提供 Android 应用的下载。然而公众在享受 Android 应用带来的生活和工作上的便利的同时，也不得不面对这样的现实，很多应用在悄悄地窃取用户手机中的私密信息。2013 年的“ 3 · 15 ” 晚会就对一些危险的 Android 应用进行了曝光，所以对 Android 应用进行全面的权限和行为上的分析是非常必要的。Android 应用的开发通常是通过 JAVA 语言和 C 语言完成的。JAVA 语言负责界面、功能等部分，但是由于 JAVA 语言运行效率较低，所以涉及高速算法的部分通常由 C 语言来完成。JAVA 语言生成由虚拟机 Android Dalvik 可以识别的中间代码，而 C 语言则形成 Android 系统的 so 库文件。本文将对这两部分代码进行分析和研究。

1 分析流程

Android 应用的安装程序是以 .apk 为后缀的文件。分析之前需要把它拆解开来，我们使用的工具是 Apktool，这是 Google 的一个 Java 开源项目，可以从 http://code.google.com/p/android-apktool/下载得到，运行过程需要安装 JRE（Java Runtime Evironment）。

1.1 拆解安装包

利用 Apktool 可以把 APK 安装包拆解成资源、配置文件、SMALI 代码文件等，如图 1 所示。具体命令格式为：

apktool d[ecode] [OPTS] <file.apk> <out_dir>

通常使用以下命令拆解 APK 安装包：apktool d -d <file.apk> <out_dir>。

加入 -d 参数是解码安装到调试模式，为便于以后调试使用，其他参数使用方法可以参见 Apktool 的命令帮助。

Android 应用不包含由C语言编写的 so 库文件，如果包含则拆解完成后，目录中将出现 lib 目录，so 库文件则包含其中，如图 2 所示。

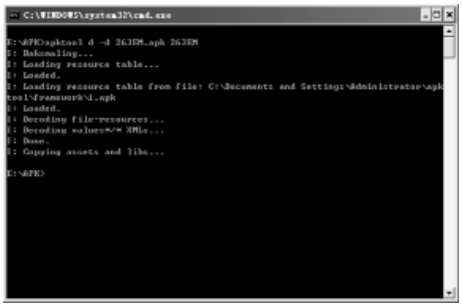

图1 利用Apktool拆解APK安装包

图2 拆解得到的文件

1.2 分析和修改代码

用 Apktool 把安装包拆解成逐个的文件后，就可以分析其中代码了。其中，AndroidManfifest.xml 文件包含了该应用的一些配置信息，其中包含该应用所需要申请的 Android 系统的权限，如访问通讯录、访问存储卡、访问 Internet、修改系统配置、得到精确的位置信息等。对此文件的分析可以初步理清该应用的一些行为。接下来，分析具体的代码 , 包括由 Java 语言生成的 smali 代码和由 C 语言生成的 native 代码，后面将会详细介绍分析的方法。

1.3 重新制作安装包

修改完文件，需要重新打包成 APK 安装包，依然使用 Apktool 工具 , 如图 3 所示。命令格式为 ：

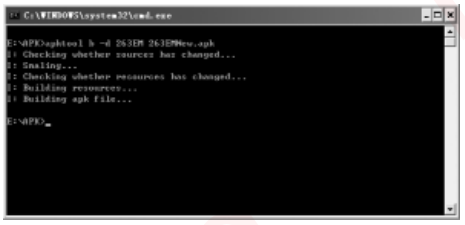

图3 重新打包成APK文件

apktool b[uild] [OPTS] [<out_dir>] [<file.apk>]

通常使用以下命令拆解 APK 安装包 ：

apktool d-d <out_dir> <file.apk>

加入 -d 参数是制作安装到调试模式，为方便以后调试用。

重新打包后还不能安装，还需要对安装包进行数字签名，使用工具 signapk.jar，该工具是 Android 源码包中的签名工具。利用这个工具进行签名，我们需要使用 Android 源码中提供的签名文件 testkey.pk8 与 testkey.x509.pem，它们位于 Android 源码的 build/target/product/security 目录，新建 signapk.bat, 内容为 ：java-jar ” %~dp0signapk.jar ” ” %~dp0testkey.x509.pem ” “ %~dp0testkey.pk8 ” %1 signed.apk。签名完毕的 APK 就可以安装到 Android 设备或者虚拟机里了。

2 静态分析

2.1 静态分析SMALI代码

通过 apktool 拆解开的文件中包含 SMALI 代码，这就是运行在 Android 虚拟机 Dalvik 中的汇编代码。下面就简单介绍一下 SMALI 的语法。

先来看一段实例代码 ：

```
 # virtual methods
.method protected onActivityResult(IILandroid/content/Intent;)V
        .locals 2
        .parameter
        .parameter
        .parameter
        .prologue
        .line 179
        if-nez p1, :cond_0

        .line 180
        new-instance v0, Landroid/os/Message;

        #v0=(UninitRef);
        invoke-direct {v0}, Landroid/os/Message;-><init>()V

        .line 181
        .local v0, sendmsg:Landroid/os/Message;
        #v0=(Reference);
        iget-object v1, p0,
Lcom/em/mobile/activity/EmSaveActivity;->uiHandler:Landroid/os/Handler;

        #v1=(Reference);
        invoke-virtual {v1, v0}, Landroid/os/Handler;->sendMessage(Landroid/
os/Message;)Z

        .line 183
        .end local v0        #sendmsg:Landroid/os/Message;
        :cond_0
        #v0=(Conflicted);v1=(Conflicted);
        return-void
.end method
```

这是一个完整的 SMALI 代码函数。函数的末尾是函数返回值类型。对于 SMALI 的数据类型详见表 1。

表1 SMALI数据类型表

名称	类型	备注
V	Void	只能用于返回值类型
Z	Boolean	
B	Byte	
S	Short	
C	Char	
I	Int	
J	long(64位)	
F	Flat	
D	double(64位)	

函数中局部变量从 v0 开始，v0、v1、v2……，函数入口参数从 p0 开始，p0、p1、p2……，其中 p0 是 this 指针。类名引用以“ L ” 开头，例如 Landroid/os/Message。

函数调用以 invoke 开始，脚本调用如下 ：

invoke-virtual {v1, v0}, Landroid/os/Handler;->sendMessage(Landroid/os/Message;)Z

v1、v0 分别为函数调用的两个函数，其中 v1 为 android/os/Handler 类对象指针，v0 为函数 sendMessage 的入口参数，函数返回值是 BOOL 型。此外，还可以通过工具 dex2jar 将 .apk

文件转换成 .jar 文件，dex2jar 的官网是 http://code.google.com/p/dex2jar。 通过此工具得到 .jar 文件后，可以使用工具 jd-gui 浏览反编译的 java 代码。

2.2 静态分析Native代码

Android 系统运行在 ARM 系列的处理器，Native 代码即是 ARM 指令集代码，如图 4 所示。下面就 ARM 处理器做简要介绍：

图4 jd-gui浏览Java反编译代码

通常 ARM 处理器一共有 37 个寄存器，其中有些寄存器是特定的 CPU 模式下才能使用。R0-R15 是常用的寄存器，其中 R15 是指令寄存器，又称 PC 寄存器，相当于 X86 的 EIP 寄存器。R14 也称作子程序连接寄存器（Subroutine Link Register）或连接寄存器 LR。当执行 BL（相当于 X86 的 CALL 指令）子程序调用指令时，R14 中得到 R15 的备份，子程序返回时，R14 的值又传给 R15。R13 是堆栈寄存器，又称 SP 寄存器，相当于 X86 的 ESP 寄存器。CPSR 为标志寄存器。ARM CPU 的指令长度都是固定的。CPU 分为两种状态，第一种为 ARM 状态，此时处理器执行 32 位对齐的 ARM 指令；第二种为 Thumb 状态，此时处理器执行 16 位的对齐的 Thumb 指令。现在移动设备多数都是 ARM 状态下的 32 位对齐的。由于指令集的内容较多，本文就不展开叙述了。

3 动态调试

3.1 动态调试SMALI代码

动态调试 SMALI 代码需要 Apktool1.4.1 及以前的版本，因为 Google 在之后的版本已经不支持动态调试。

欲调试 SMALI 代码，需要下列环境：

1）JAVA 运行时库。

2）Apktool 1.4.1。

3）Android SDK(包含 DDMS（Dalvik Debug Monitor）和 Android 模拟器)。

4）NetBeans 7.0。

调试步骤如下：

1）按照第 1 部分的分析流程拆解 APK 包 SMALI 代码和资源文件，并重新打包成 APK 文件，并对 APK 包做数字签名。

2）在 NetBeans7.0，新建项目，选择打开已有源码，把 SMALI 代码导入到 NetBeans 环境中。

3）将新打包的 APK 文件安装到模拟器中，并且运行起来。

4）运行 DDMS，并且选中应用的进程，如图 5 所示。

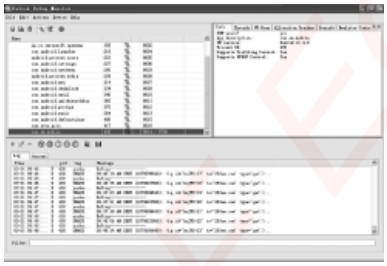

图5 在DDMS里选择要调试的进程

5）在 NetBeans 选择“ 调试 ”→“ 连接调试器 (A)…… ”，按照图 6 所示填写，点击确定。

图6 配置连接到调试器的参数

6）在 SMALI 代码上设置断点。在 NetBeans 选择 调试 ”→“ 新建断点 ”，“ 断点类型 ” 选择“ 行 ”。当程序运行到此时即会中断，然后可以在“ 调试 ” 菜单里选择“ 步进 ”、“ 步过 ” 或者“ 继续运行 ” 等。

3.2 动态调试Native代码

调试 Android Native 代码需要 IDA PRO 6.1 及以上的版本。具体步骤如下：

1) 把 IDA PRO 目录下 android_server 传到 android 设备或模拟器中，并修改文件属性：

adb push android_server /data/local/tmp/

adb shell 进入模拟器

cd /data/local/tmp/

chmod 755 android_server

2) 使用 root 权限运行 android_server，否则后面会看到的进程很少：

./android_server

看到监听端口 23946。

3) 在 windows 控制台下转发 windows 到模拟器或者手机端口，运行命令：

adb forward tcp:23946 tcp:23946

4) IDA PRO 中配置调试参数：

在 IDA PRO 中选择“ Debugger ” —“ process options ”，在弹出的对话框中，hostname 填写 localhost，port 则填写 23946。

5) 在 IDA PRO 中选择“ debugger ” —“ attach to process ”，

在弹出的列表中选择需要调试的进程。

3.3 对抗静态分析

Android 应用中，JAVA 语言的部分是必不可少的，然而 JAVA 语言编写代码容易被反编译成源代码。Google 在 Android2.3 的 SDK 里加入了 ProGuard 代码混淆工具。使用 Eclipse+ADT 开发 Android 应用软件时，会生成 project.properties 和 proguard.cfg 两个配置文件。在 project.properties 文件中添加 proguard.config=proguard.cfg。

在 IDE 自动生成的 proguard.cfg 中已经为开发者添加了一些类名和方法，如下所示：

```
-optimizationpasses 5
-dontusemixedcaseclassnames
-dontskipnonpubliclibraryclasses
-dontpreverify
-verbose
-optimizations !code/simplification/arithmetic,!field/*,!class/merging/*
-keep public class * extends android.app.Activity
-keep public class * extends android.app.Application
-keep public class * extends android.app.Service
-keep public class * extends android.content.BroadcastReceiver
-keep public class * extends android.content.ContentProvider
-keep public class com.android.vending.licensing.ILicensingService
-keepclasseswithmembernames class * {
        native <methods>;
}
-keepclasseswithmembernames class * {
        public <init>(android.content.Context, android.util.
AttributeSet);
}
-keepclasseswithmembernames class * {
        public <init>(android.content.Context, android.util.
AttributeSet, int);
}
-keepclassmembers enum * {
        public static **[] values();
        public static ** valueOf(java.lang.String);
}
-keep class * implements android.os.Parcelable {
        public static final android.os.Parcelable$Creator *;
}
```

开发者还有需要混淆的类或方法，可以按照上述格式自行添加。经过混淆后，用 jd-gui 查看到的代码如图 7 所示。

图7 混淆后的JAVA代码

3.4 对抗动态调试

为了调试代码，我们在重新编译 APK 文件时，会在 AndroidManifest.xml 文件中的 Application 标签下加入 android:debuggable=”true”，这条语句的作用是让应用程序可以调试，这时在程序中可以检测 android:debuggable 配置项，具体代码如下：

```
if ((get Application Info().flags & Application.FLAG_DEBUGGABLE) != 0){
        Log.e(“ e.com.droider.antidebug ”,“ 程序被修改为可
调试状态 ”);
        android.os.Process.killProcess(android.os.Process.
myPid());
}
```

上述代码中，程序首先检测 android:debuggable 项是否为“ true ”, 如果是，则输出调试信息，并杀掉自己的进程。

3.5 防止重编译

当 APK 文件被重新编译后，它的数字签名是会发生改变的，可以通过检测数字签名的变化来发现 APK 文件是否被重编译。Android SDK 中提供了检测软件签名的方法，可以调用 PackageManager 类的 getPackageInfo() 方法，为第 2 个参数传入 PackageManager.GET_SIGNATURES，返回的 PackageInfo 对象的 signatures 字段是软件发布时的签名，但是这个签名的内容较长，可以使用签名对象的 hashCode() 方法获取 Hash 值。

开发者可以将该 Hash 值存放在服务端，客户端取得签名对象的 Hash 值发往服务器进行校验，服务器返回校验是否通过的结果，根据此结果，客户端决定继续运行还是程序终止。

4 分析实例

我们以微信云通讯录插件作为实例，实际分析其数据加密的算法。微信以“ 通讯安全助手 ” 的功能提供通讯录云备份的功能，即把手机上的通讯录数据备份到云服务器上，我们简要分析其上传和下载数据的加密机制的过程。首先通过拆包和重新打包、加载数字签名，把新的 APK 包安装到模拟器里。然后找到其获取通讯录数据的代码，逆流而上，寻找其数据是如何加密的。我们分析发现，它的数据加密首先是通过 ZLIB 进行加密处理，然后利用 XXTEA 算法进行加密。不幸的是，这里的加密密钥为固定值，也就是说，只要我们从中间截获网络数据，即可还原出通讯录的明文。看来所谓的“ 通讯安全助手 ” 其实也并不安全。

5 结束语

Android 系统是美国 Google 公司发布的一套开源的操作系统，如今已经被广泛应用在手机、平板电脑、车载智能设备中。随着 Android 系统的人气不断上升，Android 系统中的恶意软件、病毒、木马等也开始出现，窃取隐私、恶意扣费、破坏系统等威胁正在蔓延。研究 Android 应用的逆向分析方法，有利于认清程序的各类行为，分辨应用危险的动作，从而能保护系统的安全。（责编　程斌）

参考文献：

[1] 段钢 . 加密与解密（第三版）[M]. 北京 : 电子工业出版社，2008.

[2] 佘志龙，陈昱勋，郑名杰，陈小凤，郭秩均 .Google Android SDK 开发范例大全 (第 2 版)[M]. 北京 : 人民邮电出版社，2010.

[3] 钱林松，赵海旭 . C++ 反汇编与逆向分析技术揭密 [M]. 北京 ：机械工业出版社，2011.

[4] 戴维斯 , D.W.，W.L. Price. 计算机网络安全 [M]. John Wiley & Sons. 1989.

[5] 丰生强 . Android 软件安全与逆向分析 [M]. 北京 : 人民邮电出版社 ,2013.

[6] Stallings, W. 密码学和网络安全 ：原理和实践 [M]. Prentice Hall，2006.

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

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
24. 掌握 Android 7.0 新增特性 Quick Settings

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

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

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

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

Programming:

1. 计算机软件基础数据结构 - 算法