**中图分类号：TP311**

**论文编号：10006SY1306320**



硕士学位论文

**面向移动设备的cache攻击关键技术研究**

作者姓名 李勃

学科专业 计算机系统结构

指导教师 姜博副教授

培养院系 计算机学院

**TODO**

**A Dissertation Submitted for the Degree of Master**

**Candidate：Li Bo**

**Supervisor：****Prof. Long Xiang**

School of Computer Science and Engineering

Beihang University, Beijing, China

**中图分类号： TP311**

**论文编号：10006SY1306320**

硕 士 学 位 论 文

**面向移动设备的cache攻击关键技术研究**

作者姓名 李勃 申请学位级别 工学硕士

指导教师姓名 姜博 职 称 副教授

学科专业 计算机系统结构 研究方向 程序调试技术

学习时间自 2015 年 9 月 10 日 起至 2018年 3 月 日止

论文提交日期 2018 年 月 日 论文答辩日期 2018年 3 月 日

学位授予单位 北京航空航天大学 学位授予日期 年 月 日

关于学位论文的独创性声明

本人郑重声明：所呈交的论文是本人在指导教师指导下独立进行研究工作所取得的成果，论文中有关资料和数据是实事求是的。尽我所知，除文中已经加以标注和致谢外，本论文不包含其他人已经发表或撰写的研究成果，也不包含本人或他人为获得北京航空航天大学或其它教育机构的学位或学历证书而使用过的材料。与我一同工作的同志对研究所做的任何贡献均已在论文中作出了明确的说明。

若有不实之处，本人愿意承担相关法律责任。

学位论文作者签名：        日期： 年 月 日

学位论文使用授权书

本人完全同意北京航空航天大学有权使用本学位论文（包括但不限于其印刷版和电子版），使用方式包括但不限于：保留学位论文，按规定向国家有关部门（机构）送交学位论文，以学术交流为目的赠送和交换学位论文，允许学位论文被查阅、借阅和复印，将学位论文的全部或部分内容编入有关数据库进行检索，采用影印、缩印或其他复制手段保存学位论文。

保密学位论文在解密后的使用授权同上。

学位论文作者签名： 日期： 年 月 日

指导教师签名： 日期： 年 月 日

摘 要

待写

**关键词：** cache攻击

**Abstract**

Todo

**Key words**: Cache attack

**目 录**

**目 录** iii

**图 目** v

**表 目** viii

第一章 绪论 1

1.1 研究背景及意义 1

1.2 Cache攻击研究现状 2

1.3 研究内容及目标 2

1.3.1 实现精确计时方式 3

1.3.2 实现有效的驱逐策略 4

1.3.3 实现高效攻击方式 6

1.3.4 实现keylogger工具 7

1.3.5 实现AES密钥攻击 7

1.4 本文研究限制 9

1.5 本文的组织结构 9

第二章 相关技术以及原理 11

2.1 Cache结构 11

2.1.1 CPU Caches 11

2.1.2 全相联映射 12

2.1.3 组相联映射 12

2.1.4 Cache替换策略 13

2.1.5 Tag以及Index 13

2.1.6 Cache包含性 14

2.2 AES加密 15

2.3 计时方式 15

2.4 Cache攻击 15

2.4.1 Evict+Time 16

2.4.2 Prime+Probe 16

2.5 本文研究限制 18

2.6 本章小结 18

第三章 驱逐方案的设计 19

3.1 攻击策略 19

3.2 驱逐策略 20

3.3 AES攻击方式 20

3.4 异步攻击 20

3.5 关键技术解决方案 20

3.6 本章小结 20

第四章 AES攻击验证 21

4.1 AES同步攻击 21

4.2 AES异步探索 21

4.3 本章小结 21

第五章 预防攻击措施 22

5.1 攻击漏洞 22

5.2 预防攻击措施 22

5.3 实验结果 22

5.4 本章小结 22

总结与展望 23

工作与研究总结 23

对未来工作的展望 23

参考文献 25

附 录 26

攻读硕士学位期间得到的学术成果 27

致 谢 28

**图 目**

图 1 记录系统和重放系统在执行重放中扮演的角色 3

图 2 执行方案的分类 4

图 3 SMP-Revirt使用CREW协议实现对页面访问冲突顺序的记录 5

图 4 FDR硬件架构图 7

图 5 ACoreOs调试器工作原理 14

图 6 ACoreOs操作系统组成及各部分关系图 15

图 7 宿主机与目标机的连接关系图 16

图 8 ACoreOs任务状态转换图 17

图 9 不可抢占的优先级调度 18

图 10 可抢占的优先级调度 19

图 11 时间片轮转调度 20

图 12 导致系统出错的执行 24

图 13 相同的执行，加入了软件探针，"躲过"了故障 24

图 14 任务相应执行代码 26

图 15 并行访问信号量时任务A先获取到操作权 27

图 16 并行访问信号量时任务B先获取到操作权 27

图 17 执行重放方案整体设计图 29

图 18 执行重放执行流程 31

图 19 信息记录模块的分离式设计 34

图 20 单核版本日志过滤模块设计框架图 37

图 21 多核版本日志过滤模块设计框架图 38

图 22 单核版本重放模块设计框图 42

图 23 多核版本重放模块设计框图 43

图 24 互斥信号量对多核任务执行影响 44

图 25 循环体中的PC指针无法唯一标识程序执行的当前位置 45

图 26 触发任务切换因素 46

图 27 信息记录模块实现框图 50

图 28 信息记录模块开启执行流程 50

图 29 ACoreOs\_semaphore\_obtain接口探针插桩比对实例 56

图 30 中断信息插桩探针对比实例 57

图 31 rdtsc()函数插桩实现 57

图 32 环形缓冲区结构图 58

图 33 写入控制模块执行流程图 59

图 34 伪消息队列设计框图 61

图 35 缓冲区管理模块执行流程图 61

图 36 日志上传模块执行流程图 63

图 37 日志解析模块工作流程 65

图 38 日志解析模块实现框图 66

图 39 日志过滤模块实现 67

图 40 单核版本的日志过滤模块执行流程 68

图 41 多核版本的日志过滤模块执行流程 70

图 42 单核版本执行重放模块的执行流程 72

图 43 CPU管理模块工作流程 73

图 44 多核版本执行重放模块执行流程 76

图 45 测试环境架构图 79

图 46 同步事件测试程序运行结果 81

图 47 同步事件测试程序执行结果分析结果 82

图 48 同步事件测试程序执行时序图 83

图 49 日志中任务创建事件信息的提取和托管 84

图 50 重放模块根据日志创建任务上下文 84

图 51 重放模块根据同步事件信息设置断点并启动相应任务 85

图 52 同步事件设置的断点命中后上下文比对过程 85

图 53 到达切换点后任务切换过程 86

图 54 生产者消费者程序重放执行输出结果 87

图 55 数据流测试程序运行结果 88

图 56 数据流测试程序记录得到的日志分析结果 89

图 57 重放阶段创建重放任务上下文 90

图 58 任务1重放过程中断点设置 90

图 59 任务1重放中断点命中数据回写 90

图 60 数据流测试程序重放执行输出结果 91

图 61 中断测试程序执行阶段输出结果 92

图 62 中断测试程序记录得到的日志分析结果 92

图 63 中断事件重放过程 93

图 64 中断测试程序的重放输出结果 94

图 65 多核上生产者消费者测试程序执行结果 95

图 66 多核上生产者消费者任务日志分析结果 96

图 67 多核重放模块创建对应任务上下文 97

图 68 多核下事件重放过程 98

图 69 多核上生产者消费者被重放输出结果 98

图 70 插桩开销实验结果 99

**表 目**

表 1 现有执行重放方案对比 7

表 2 软件调试方法分类 13

表 3 任务创建事件信息记录探针 51

表 4 任务启动事件信息记录探针 51

表 5 信号量操作信息记录探针 52

表 6 消息队列事件信息记录探针 52

表 7 上下文切换事件信息记录探针 53

表 8 调度指示信息记录探针 53

表 9 中断打断任务上下文信息记录探针 54

表 10 数据流信息记录探针 54

表 11 CPU索引信息记录探针 54

表 12 缓冲区管理任务对各个消息处理过程 62

表 13 Java语言和目标系统数据结构对应关系 64

表 14 过滤多余事件信息算法伪代码 68

表 15 日志合并算法伪代码 69

表 16 过滤多余异步事件算法伪代码 70

表 17 上下文比对算法伪代码 74

表 18 实验测试程序概述 80

# 绪论

## 研究背景及意义

从第一台手机诞生到现在，伴随着移动互联网突飞猛进式的增长，手机等移动设备已经成为我们日常生活中不可或缺的一部分。它在给我们的生活带来便利的同时也对我们的隐私带来了潜在的威胁，尤其是手机存储着包含账号、密码等用户私密信息的现在。因此，随着技术的发展，隐私安全问题也越来越得到人们的重视。虽然移动设备开发厂商以及Android、Windows等操作系统采取了很多安全措施来保护用户的隐私，包括虚拟内存管理、权限管理、可执行环境等技术。然而，由于系统硬件的一些固有缺陷，一些针对计算机以及移动设备的攻击方式逐渐显现出来，其中就包括本文所研究的cache攻击。其攻击方式为基于cache hit与cache miss的时间差异来探测被攻击程序执行过程中的内存及cache的访问情况，并依次获取被攻击程序私密信息。由于攻击程序和被攻击程序之间没有直接的交互，并可以运行在不同的核上，访问各自的内存区域，因此不需要额外的权限就能进行攻击，因此是一种强大的攻击方式。

期初，在Intel x86平台上的Cache攻击得到了科研工作者的重视，并有很多学者从事这一方向的研究。Kocher[1]和Kelsey[2]提出了通过分析高速缓冲存储器cache在运行时泄露出的信息来破解计算机加密算法的方法。基于这些思想，近些年不断的有人提出在Intel x86上进行cache攻击的方法，并成功的实现了监听用户输入、破解加密算法等攻击，也反向证明了Cache攻击是有效的。比如，Page et al.[41]提出了一种破解DES加密技术的Cache攻击模型，并将DES密钥的搜索空间从56位降低到了32位。Tsunoo[26]随后提出通过分析DES加密过程的查表索引以及其Cache访问特性的方式，首次实现了针对DES密钥的Cache攻击，并在一台600-MHZ的Pentium III个人电脑上通过223个明文成功的获取了DES全部密钥。Bernstein[28]在消除网络传输延迟的条件下实现了针对OpenSSL[29]中AES加密的时间驱动攻击，其中，服务端使用AES加密算法加密数据。Percival[36]提出了多线程间共享Cache缓存可能产生的漏洞，该漏洞其他线程提供一个简单的，高带宽的隐秘铜套，通过使用该漏洞，恶意程序能够监视其他线程，窃取被攻击程序加密算法的密钥等信息。依此Percival设计并实现了一种针对RSA加密算法的攻击方式。鉴于Percival提出的方法，Osvik[37]设计了一种针对AES加密技术的Cache计时攻击方式。Neve[38]则将Osvik的切入点转移到AES加密过程中的最后一轮，提出一种新的最后一轮驱动的Cache攻击方法。通过这个漏洞，攻击者能够轻易的获取到包含用户输入、加密算法密钥等在内的私密信息，进而威胁到用户的人生财产安全。

然而，由于Android等移动设备的结构与Intel x86设备的结构有很大的区别，在指令集合、cache组织方式以及cache替换策略等与cache攻击密切相关的结构也有很大的不同。因此，直到最近为止，才有人提出在非root的手机上执行有效的cache攻击方法。Moritz Lipp et al. [24] 提出了通过在移动设备上进行prime + probe， flush + reload， evict + reload以及flush + flush对ARM处理器的跨核攻击模型，并且不需要root权限。这些模型能够有效的探测到在被攻击程序运行时cache无意间泄露出来的信息，通过对这些信息进行统计分析，并将其作用于cache攻击的模型即可提取用户的私密信息。其中最典型的攻击模型为cache模板攻击，该模型在探测阶段不断的探测待攻击时间执行时共享库地址空间的加载情况，形成一个cache模板矩阵，该矩阵对应着某一事件执行时共享库各个地址的访问情况。在通过该模板矩阵进行攻击时，探测用户执行事件时共享库地址的加载情况，并与cache模板矩阵进行对比，进而分析出用户执行的操作。通过近些年的研究，cache旁路攻击已经被认可为一种强大的攻击方法，在研究攻击方法的同时也提出了一些修复漏洞的方法，包括在Android 6.0.1中修复的 /proc/self/pagemap 的访问权限。通过控制在非特权模式下用户线程对自己存储空间页表映射表的权限，能够有效的抑制攻击程序通过共享库执行cache模板攻击。

基于这些背景，本题旨在发掘更多在移动设备端的cache攻击方式，以及有效的攻击模式，主要针对用户输入、高级加密技术（AES）以及可信执行环境，探索并实现有效的cache攻击。挖掘与用户输入和AES加密有关的可能发生信息泄露的漏洞，进一步针对攻击提出一些应对措施，从而促进移动设备端安全化进程，使用户能够更加安全，更加放心的使用手机。

## 研究内容及目标

## 本文研究限制

## 本文的组织结构

本文共分为五章，具体内容如下：

第一章 绪论。首先介绍了课题的研究背景和意义，接着介绍了执行重放技术的基本概念，原理和应用，并对此分析了近些年国内外对执行重放技术的研究工作，最后给出了本文的研究目标及内容。

第二章 相关技术以及原理。首先分析了插桩技术、软件调试技术，特别应用于本文中的交叉调试技术，接着详细介绍了本文的目标操作系统ACoreOs的内部实现机制，最后给出了执行重放过程中所面对的挑战和问题。

第三章 执行重放方案的设计。首先指出了重放本文目标系统上应用程序存在的问题，然后根据存在的问题具体给出了相应的解决方案，接下来，给出了本文提出的执行重放整体设计框图和各个模块的大致设计思路。最后，本文针对相关挑战给出了相应的解决方案。

第四章 执行重放方案的实现。本章分三部分，分别给出了执行重放方案中三大模块：信息记录模块、日志解析过滤模块、重放模块的实现细节。

第五章 系统验证。详细介绍了对本文提出的执行重放方案的测试和验证。并对引入的插桩开销进行了度量。

总结与展望。首先对本文的工作进行了总结，然后指出来本文工作不足，最后给出了对后续工作的展望。

# 相关技术以及原理

## Cache结构

cache攻击就是攻击者针对cache结构，设计程序通过cache获取被攻击程序的内存访问情况，据此分析出用户的私密信息，例如用户的按键输入、秘钥等。熟悉cache结构就成为设计攻击程序或对其进行防卫的必要条件，本章主要包含CPU cache的详细介绍、cache组织、cache与内存的映射关系以及cache攻击技术等信息。

### CPU Caches

当代计算机以及各种移动设备的CPU性能不仅仅依赖与其时钟周期，还受到其指令集以及和其他设备的交互的影响。因为内存应该尽可能快的向CPU提供其中存储的数据，因此如下图所示的层次图被设计出来缩小内存与CPU的速度差异。通常情况下来说，从cache中读取数据要比从内存中读取快很多倍，因此称之为缓存。

由于访问速度越贵的内存价格越昂贵，内存架构组织成金字塔型的层次结构在访问速度和价格之间做折中，越靠近CPU的存储访问速度越快，价格也越昂贵，越靠近内存的存储价格越便宜，相应的访问速度也越慢。因此访问缓存中的数据要比访问主存中的速度要快很多。

为了性能方面的考虑，通常情况下L1缓存直接与CPU中获取指令以及加载和存储数据的核心逻辑相连。对于冯诺依曼体系结构的计算机，通常情况下只使用一个cache用于存储指令和数据，然而哈佛体系结构的计算机则分别有一个指令缓存I-Cache和一个数据缓存D-Cache,顾名思义I-Cache用于缓存指令而D-Cache用户缓存数据，并且指令缓存和数据缓存通过两条总线可以同时传输数据，因此数据读取速度更快一些。

程序更倾向于访问已经访问过的地址以及附近的地址，比如在一个循环中，相同的代码被一遍又一遍的执行，这也被称之为程序运行的局部性原理。因此，为了提高程序的运行速度，需要尽可能的将之后需要访问的指令及数据提早缓存到cache中，间接提高访存速度。对一些追求实时性的硬件及程序cache使得读取指令或数据的时间存在不确定性，会导致问题的发生。

Cache仅仅缓存了主存中的一部分数据或指令，因此cache必须能够记录下这部分数据的地址以及其相关的内容。当CPU想要加载或修改某一地址的数据时，它首先去L1 cache中查找，看所需要的数据在不在缓存中。如果不在缓存中，则CPU必须到更低一级的缓存或主存中去查找数据，以确保指令流水能够顺利的执行。出自性能方面的考虑，每次会从内存中将一个内存块缓存到cache中，也就是一个cache line，其中每个内存块包含B个Bytes，这也是cache缓存的最小单位。因此当一个地址被缓存到cache中时，其附近地址的数据也被一同缓存到cache中，根据程序的局部性原理，这些数据很有可能是程序不久将要访问到的，因此能够减少从内存中获取数据频率，增加程序的运行速度。每个cache line有一个tag，表示什么地址的数据被缓存到cache中。

### 全相联映射

有很多种cache实现的方式，其中最简单的就是直接相联映射。在一个基于直接映射方式实现的cache中，主存中的每个内存块只能唯一对应cache中指定位置的set中，这种对应方式是不能改变的，也就是说主存中的数据要么只能被缓存到cache中相应的set中，要么就是仅存储在内存中。由于主存大小远远大于cache容量，因此会有多个内存块对应到同一个cache set的情况。下图展示了一个拥有128个set，每个set 64字节，总容量为4KB的cache与内存的映射情况。

为了在cache中获取指定地址的数据，首先获取地址的index位，将地址中的tag位与cache相应index位置数据的tag位进行比较，如果tag位相等，表示cache命中，则根据offset位将响应位置的数据传输到cache中或修改改位置的数据。如果tag位不相等，则表示cache缺失，及所查找的数据不在这一级cache中，进一步将从下一级cache或从主存中获取数据到cache中，替换掉之前缓存的数据并跟新tag。

### 组相联映射

如果替换策略能够选择cache中的任何块存储数据，则这样的cache结构称为全相联映射。现在的cache被组织为若干个set，其中每个set包含固定数目的cache line，每个cache line对应内存中的一个数据块，这种组织方式被称为组相联映射。如果组相联映射结构中一个cache set包含N个cache line，则将其称之为N-way组相联。因此，每个内存地址映射到其中一个cache set中，并且可以缓存在这个set中的任意一个cache line中。任意两个能偶映射到同一个cache set的address成为是相互关联的。关联的地址总会竞争同一个cache set中的line，并由驱逐策略决定被驱逐的line。为了方便理解，如下描述本文用于实验的Lenovo K51c78测试机的L2 cache，其cache总容量为512KB，总共包含512个set，cache与内存的映射关系为16路组相联，因此cache index位为9位。

### Cache替换策略

在组相联结构的cache中，当cache set中每个cache line都缓存数据或当cache确实发生时，缓存的代码或数据必须从cache中驱逐到内从中以缓存新的数据或指令。决定驱逐哪个cache line的算法就被称之为替换策略。替换策略必须决定cache set中的哪些数据在将来不会被使用到。最近最少替换算法LRU总会替换掉最近没有被使用过的cache line中的数据。然而ARM结构的处理器的cache通常使用称之为pseudo-LRU伪随机替换策略，虽然也有ARM处理器使用LRU替换算法作为cache的替换策略，然而由于性能方面的考虑，在实际中通常使用伪随机替换策略。伪随机替换策略Pseudo-random将根据伪随机数生成器决定驱逐cache set中的哪个cache line。

### Tag以及Index

CPU能够分别使用基于虚拟地址或物理地址的虚拟index或物理index。通常虚拟索引cache比物理索引cache要块，因为虚拟索引在cache查找时不要将虚拟地址转换为物理地址。但使用虚拟索引会导致相同的物理地址被缓存到不同的cache line中，这会导致性能的下降。为了唯一的标记缓存到指定line中的地址，通常使用tag标签来进行标记。同样，tag也能时虚拟的或者物理的。可能的搭配当时的优缺点如下所示：

VIVT-虚拟索引，虚拟标签

索引和标签都是使用的虚拟地址，由于在查找cache时不需要地址转换，因此查询速度较快。然而，由于虚拟标签不是唯一的，共享内存可能在cache中缓存多次

* PIPT-物理索引，物理标签

索引和标签都是使用的物理地址，由于查找cache时需要将虚拟地址转换为物理地址，因此速度较慢，不过共享内存仅会缓存一份到cache中。

* PIVT-物理索引，虚拟标签

物理地址用于索引，标签使用的虚拟地址，这种组合内有任何好处，因为查找时既需要将虚拟地址转换为物理地址，并且共享内存也可能在cache中缓存多份

* VIPT-虚拟索引，物理标签

虚拟地址用于索引，物理地址用于标签，这种组合相较PIPT的优势在于通过并行查找TLB转换物理地址，然而只有当物理地址转换完毕后才能比较tag标签。

### Cache包含性

为了提高cache命中的几率，CPUs通常使用多级缓存L1～Ln，其中越靠近CPU的缓存速度越快，价格也越贵，容量也越小。就像先前描述的那样，本文实现基于的目标机Lenovo k51c78的cache分为两层，L1 cache结构为4路组相联，大小为32KB，共有128个sets，而L2 cache结构为16路组相联，大小为512KB，共有512个sets。此外目标机利用了哈佛体系cache结构的变种，及L1 cache分为指令cache和数据cache，但它们使用同样的地址空间。每个核均有一个L1 cache，由于k51c78共有4个核心，因此共有4个L1 cache，以及4核公用的1个L2 cache。

如果从cache中读取一个字，则存在与cache中的该字应该与该字对应在内存中的内容相等。然而，当有核执行存储指令企图修改数据时，在写操作执行之前先要查找对应的数据是否缓存在cache中，如果cache命中，则有如下两种策略：

* 写回策略：

写会策略将数据写到cache中，因此会存在cache与内存数据不一致的问题。当没有将被写过的数据写会内存时这种方式没有什么问题，但当cache set中cache line都被占用，并且驱逐策略决定将当前cache line中的数据驱逐到内存中时，就需要判断该line中是否存有被修改过但还没有来得及写到内存中的数据。其中判断依据是以cache line的额外字段描述的，及脏位，若脏位被置为1，则表示该line存有还未写到内存中的数据，这时需要先将line中的数据写会到内存中，再写入要缓存的数据。

* 写穿策略：

写穿策略在CPU执行存储命令时将要更新的数据同时写到cache和内存中，保持主存和缓存数据的一致性，由于每次写操作都需要执行写存操作，而内存访问速度远远慢于访问cache的速度，因此执行速度较慢。

对于包含多层缓存的处理器，cache需要决定将数据缓存到那一层中，因此有几种不同的存储策略：

* 包含式缓存：

对于低级别的缓存来说，一个高级别的缓存是包含式的，当缓存在低级别缓存中的数据同事也缓存在高级别的缓存中。因此，对于包含式缓存，低级别缓存中的数据时高级别缓存中数据的一个子集。

* 排除式缓存：

当一个cache line仅仅能保存在两级缓存的其中一级中时，该cache被成为时排除式的。

* 非包含式缓存：

如果一个缓存既不是包含式的也不是排除式的，则这种缓存结构称为是非包含式缓存。

现代的因特尔CPUs的最后一级cache是包含式的，AMD CPUs的最后一级cache是排除式的，而大多数ARM CPUs的最后一级cache是非包含式的。然而ARM Cortex-A53/Cortex-A57 CPUs 的最后一级cache是包含式的。

## AES加密

AES是美国国家标准技术研究所NIST旨在取代DES的21世纪的加密标准。AES加密数据块分组长度必须为128比特，密钥长度可以是128比特、192比特、256比特中的任意一个（如果数据块及密钥长度不足时，会补齐）。AES加密有很多轮的重复和变换。大致步骤如下：1、密钥扩展（KeyExpansion），2、初始轮（Initial Round），3、重复轮（Rounds），每一轮又包括：SubBytes、ShiftRows、MixColumns、AddRoundKey，4、最终轮（Final Round），但最终轮没有MixColumns。

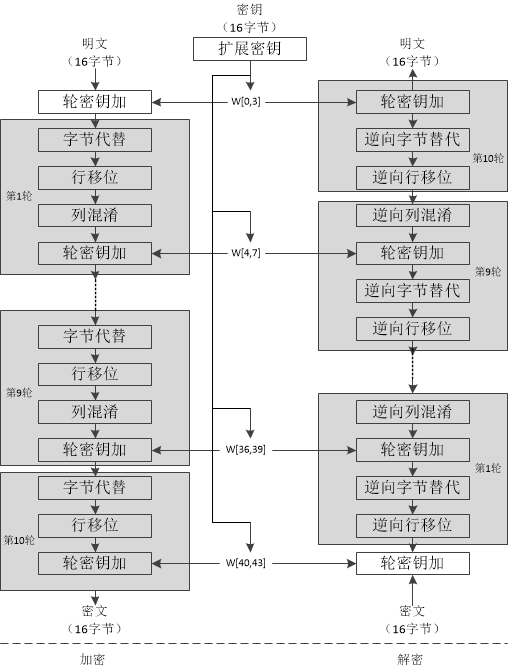


图1 AES加密步骤

Flush + Reload， Evict + Reload以及Prime + Probe可以用来对T-table实现的AES加密算法进行攻击。Moritz Lipp et al. [24] 阐述了跨核同步T-table AES的可能性，及由攻击者触发加密算法的执行。但异步的跨核攻击更具通用型，由于异步执行会引入不受控的内存访问，从而污染cache状态，因此实现的难度教同步cache攻击要大。本题旨在实现针对T-table AES的跨核的异步cache攻击的实现。

本题的研究目标为在目标机（Lenovo K51c78）实现精确的计时方式，并得到快速高效cache驱逐策略，实现有效cache攻击策略，并实现针对用户输入的keylogger攻击、实现端到端的基于AES T-table的cache攻击，并探索对可信执行环境TEE的有效攻击方式。

上述所提到的Evict + Time、Prime + Probe、Flush + Reload以及Evict + Reload等方法虽然都有Intel x86平台下的实现。将他们与有效的时间源计时方式相结合，能够有效的将cache hits和cache misses的情况区分开，从而能够用于实现平台上的cache攻击。然而，由于ARM平台的指令集、cache组织结构以及cache的多级包含关系于x86平台有很大的区别。且cache的替换策略也与x86平台上的最近最少使用算法不一致，ARM所使用的伪随机替换算法增加了cache set的操作难度，驱逐操作的完整性在很大程度上是随机的，因此造成的系统噪音也是ARM移动平台cache攻击研究进度缓慢的原因之一。之前在x86平台上的攻击方式在Android客户端上不一定能发挥作用。此外，之前一些针对于ARM平台的cache攻击将被攻击程序和攻击程序集成到一个程序中，即由攻击程序触发待攻击的加密。由于这种方式能够有效的减少系统切换对占用了的cache set的影响，并且能够缩短驱逐出set到执行被攻击程序的时间，因此能够有效的减少系统噪音，得到更加明显的实验数据。然而在实际的攻击过程中，攻击程序和被攻击程序通常是不同的程序，攻击过程中引入的系统噪音会导致时间测量出现误差，需要对时间测量，驱逐策略等方面进行改进来优化对cache的操控。基于这些思想，本题主要的研究方向为面向移动设备的cache攻击技术研究，实现evict + reload以及prime + probe等工具，并实现对诸如key logger、高级加密（AES）或可信执行环境（TEE）的cache 攻击

## 计时方式

## Cache攻击

cache边信道攻击主要利用从内存加载数据与从缓存加载数据时间上的差距泄露的信息来进行攻击的。由于读取缓存在cache中的数据要比从内存中获取数据要快很多，通常从cache中读取数据只需要不到1ns，而从内从中获取数据要几十到几百纳秒的时间，大了2个数量级，因此能够通过判断指定的数据是否缓存在cache中来推断其最近有没有被访问过。其中泄露出来的访问信息是潜在的漏洞，特别对于加密算法，因为其可能导致秘钥被破解。

早在1996年，Kocher[43]提出了通过度量加密操作所需的总时间来破解加密系统的想法，Kocher也成为第一个提出通过CPU cache泄露的信息在不读取加密相关数据情况下破解加密系统的人。4年后Kelsey et al.[40] 提出了旁信道攻击的概念，并且断言基于cache命中率对Blowfish[71]以及CAST[14]等使用沙盒S-boxes的加密算法进行攻击时可行的。Page[64]和Tsunoo et al.[82]基于Page提出的理论对数据加密标准(DES)进行攻击。同样也有针对高级加密技术AES的cache攻击的研究，比如Bernstein[9]提出了针对AES加密技术的著名的cache-timing攻击，Neve[57]及neve et al.[59]对cache-timing进行了深入的分析。

本节将描述几种有效的cache攻击模式，这些攻击模式在Intel x86平台上能够有效的获取cache泄露的信息，包括Evict+Time，Prime+Prime，Evict+Reload等。

### Evict+Time

2005年，Percival[66]和Osvik et al.[63]提出了能够更有效的从CPU cache获取私密信息的方法，Osvik et al.还规范了这两中概念，及Evict+Time以及Prime+Probe，将在随后讨论。它们的基本思想都是判断cache中的哪些set被被攻击程序访问过。

Evict+Time算法：

1.测量被攻击程序的执行时间

2.驱逐cache中指定的set

3.再次测量被攻击程序的执行时间

Evict+Time算法可以用来决定指定的cache set在被攻击程序执行期间有没有访问到。首先，测量得到被攻击程序的执行时间t1，在下次测量被攻击程序执行时间之前，指定cache set i中的数据被驱逐到内存中，并测量得到被攻击程序的执行时间t2。最后通过比较t1与t2的区别来判断被攻击程序在执行过程中有没有访问到指定的set。若t1小于t2，则表示驱逐set i中的数据增加了被攻击程序的执行时间，也就是驱逐操作将被攻击程序之前缓存在该cache set中的数据驱逐到内存中，也就是被攻击程序执行过程中需要访问到set i。

Osvik et al.[63]和Tromer et al.[81]提出Evict+Time能够对基于OpenSSl实现的AES实现强有力的攻击，并且在攻击过程中不需要加密过程中的明文以及密文的信息。

### Prime+Probe

第二种攻击模式是由Osvik et al.提出来的，并将其称之为Prime+Probe。与Evict+Time相同，通过Prime+Probe攻击者能够判断cache中的某些set有没有被被攻击程序访问到。

Prime+Probe算法：

1.占用指定的cache sets

2.执行被攻击程序

3.检测cache中的哪些sets仍然被占用

Prime+Probe算法主要由3个步骤组成，首先，攻击程序通过其内存空间的数据占用一个或多个指定的cache set，对于使用LRU替换策略的cache，通过连续读取能够映射到指定set的与该set所包含的cache line数量相等的数据，则可将该set之前缓存的数据全部驱逐到内存中，且需要确保读取的数据在攻击程序的内存空间中。此后，执行被攻击程序，被攻击程序执行过程中的访存操作会占用cache中的某些set的某些line，并根据替换算法替换之前存在的部分数据。最后，攻击判断在第一阶段读取到cache中的数据是否还缓存在cache中。

上图展示了攻击的详细过程，网格代表一个8路组相联（列）的cache，共有6个sets（行），且假设攻击者旨在检测被攻击程序在执行过程中对cache set 4的占用情况。在步骤a中，攻击程序通过连续访问能够映射到set 4中的数据来占用set 4中的8个cache line，由于该cache使用的组相联的映射方式，连续两个能够映射到该set中的地址之间的间隙为6×B字节，B为一个cache line所包含的字节数，为了将一个cache set中的所有数据清除干净，至少需要读与cache set所包含的line数相等的相关地址，本例中则至少需要读8个相关地址，记为m1,...,m8。在执行万步骤a后，cache set 4中的所有line均缓存着攻击程序的数据，由于该cache为8路组相联，因此可以将缓存在该cache set中的数据表示为b1、b2、b3、b4、b5、b6，每个字母表示一组cache line大小的数据，并由蓝色标识。

在步骤b中，被攻击程序执行，在其执行过程中，由于不断的执行指令并访问数据，为了提高访存速度，操作系统会将这些数据所在的数据块缓存到cache中，具体缓存到那个cache set中是由数据的起始地址决定的，而具体存储到哪个cache line中是由相应的替换算法决定的。如图所示，被攻击程序执行过程中访问的部分数据对应的cache set也为set 4，且set 4中的所有line均被占用，因此cache根据替换策略替换了该set中的部分line。在这一过程中缓存的数据使用红色标识，如上图所示。

在步骤c中，攻击程序再次访问步骤a中访问过的地址m1,…,m8，并判断这些在步骤a中缓存到cache set中的数据是否还保留在cache中。判断的依据就是当再次访问这些数据时，若访问时间快，则表示数据依然在cache中，若访问时间较慢，则表示数据在被攻击程序执行期间被驱逐到了内存中，需要再次从内存中获取数据，因此导致较长的访问时间。

## 本文研究限制

## 本章小结

# 驱逐方案的设计

在过去的几十年里，得益于摩尔定律CPU的运算速度以每年大概60%的速度提升，然而内存的读写速度每年的提升幅度在7-9%之间，使得本来就慢得多的内存越来越跟不上CPU的速度，到现在两者之间已经有了很大的差距，对现在的计算机处理器，访问在一级缓存中的数据只需要0.3ns，而访问在内存中的数据需要50到150ns的时间，大约降低了2到3个数量级。为了解决CPU与内存的速度不匹配问题，现代计算机及移动设备在CPU与内存间放置多级缓存，在缓存命中的情况下大幅降低访存时间。但也引入了一些问题，及缓存缺失时会导致更长的访问时间。因此，Cache被设计为特定的访问模式以降低cache的miss几率，如全相联映射、组相联映射等，针对特定的cache模式，根据内存与cache的映射关系能够轻易的对其进行攻击，获取被攻击程序执行时的缓存信息。

现代处理器使用一级或多级缓存的组相联缓存，从上到下内存容量逐渐增加，访问速度逐渐下降，其中每一级由S个cache set，每个cache set中包含W个cache line，每个cache line能够容纳B字节的数据，因此cache的总容量为B×W×S个字节。其中cache line是cache缓存的基本单位，当cpu读取的数据不在缓存中时，系统产生一个中断，并从内存中获取一个内存块的数据缓存到cache中，再从cache中读取数据到cpu，其中内存块block的大小与cache line的大小相同。对采用组相联映射的cache，一个cache set包含W个cache line，当cache miss发生时，从内存中获取B字节的内存块缓存到cache中，set索引号为内存块起始地址a mod [W × B]，并根据cache使用的替换策略替换掉（驱逐）set中指定的cache line。常见的替换策略由最近最少使用算法LRU、随机替换算法等，若使用LRU算法，则cache set中最近没有被访问过得cache line被驱逐到内存中去。

现代处理器有多到3级的缓存及L1到L3，其中L1的访问速度最快，容量最小，当L1查找失败时到L2查找，相应的L2的访问速度较L1要慢，容量比L1要高，L3同理，但L1到L3的访问速度比访问内存速度要快很多。为了简化攻击过程，本文不区分访问L1和访问L2、L3的区别，只区分cache命中和cache缺失。

## 计时方式设计

一个精确的计时方式是cache攻击的前提，它需要将cache hits和cache misses区分开来，供攻击者获取待攻击程序运行导致的cache状态的变化。并针对不同的攻击对象获取不同的相关信息，比如对共享库的攻击需要获取共享库相关地址在被攻击程序运行期间的cache缓存状况。然而获取这些状况的前提都是拥有能够准确区分cache hits和cache misses的能力。Moritz Lipp et al.[24]虽然提出了几种非特权的计时方式，包括perf\_event\_open、POSIX的clock\_gettime函数以及专用的线程计时器。但这些接口并非对所有Android版本，对所有处理器都开放，因此需要对待攻击机型确定能精确且能稳定测量的时间源，以此提供对cache攻击的支持。

除了通过读取寄存器获取CPU周期来度量时间外，还有其他3种可选方式来度量访存或访问cache的时间。

1. clock\_gettime系统调用

"clock\_gettime"是基于Linux C语言的时间函数，他可以精确到纳秒。

#include<time.h>

int clock\_gettime(clockid\_t clk\_id， struct timespec \*tp);

clk\_id : 检索和设置的clk\_id指定的时钟时间。

CLOCK\_REALTIME:系统实时时间，随系统实时时间改变而改变，即从UTC1970-1-1 0:0:0开始计时，中间时刻如果系统时间被用户改成其他，则对应的时间相应改变

CLOCK\_MONOTONIC:从系统启动这一刻起开始计时，不受系统时间被用户改变的影响

CLOCK\_PROCESS\_CPUTIME\_ID:本进程到当前代码系统CPU花费的时间

CLOCK\_THREAD\_CPUTIME\_ID:本线程到当前代码系统CPU花费的时间

struct timespec

{

time\_t tv\_sec; /\* 秒\*/

long tv\_nsec; /\* 纳秒\*/

};

1. Perf性能分析工具

Perf是内置于Linux内核源码树中的性能剖析(profiling)工具。它基于事件采样原理，以性能事件为基础，支持针对处理器相关性能指标与操作系统相关性能指标的性能剖析。常用于性能瓶颈的查找与热点代码的定位。

从Linux Kernel2.6.31版本开始，Linux内核开始提供\_\_NR\_perf\_event\_open的系统调用。使用这个系统调用我们可以像使用文件一样打开一个Performance counter，通过设置不同的参数让这个Performance Counter统计不同的软件或硬件事件，然后就可以向读文件一样来读取这些事件的统计结果。

static struct perf\_event\_attr attr;

attr.type = PERF\_TYPE\_HARDWARE;

attr.config = PERF\_COUNT\_HW\_CPU\_CYCLES;

fd = syscall(\_\_NR\_perf\_event\_open， &attr， 0， -1， -1， 0);

PERF\_COUNT\_HW\_CPU\_CYCLES：统计CPU周期数

1. 线程计时模拟器

如果没有足够精确的计时接口可用，攻击者可以通过运行一个循环自增一个全局变量的线程来得到CPU周期的一个倍数估计值。因为一次循环自增操作可看做是由固定次数个时间周期组成的，所测时间的整数倍即为所对应的时钟周期，因此能够将cache hits和cache misses区分开来，从而也可以用于cache攻击。

## 驱逐策略设计

为了将地址从cache中驱逐到主存中，在Intel x86平台可以使用非特权的clflush指令。虽然ARM平台也提供了类似的cache操作工具，但在非特权模式下不允许使用。驱逐的第二种方式为相关地址的访问，主要原理为读取大量的能够映射到制定cache set的地址，以此来将该cache set中之前存储的数据替换到主存中。虽然读取大量的地址能够大概率的保证将关联set中的数据都驱逐出内存，但大量的访存操作不仅仅会增加驱逐所花的时间，存储相关地址的内存也会增大，而且由于cache伪随机替换策略的影响，驱逐干净cache set后很难了解cache set中存储哪些相关地址，会对probe阶段的探测结果产生影响，从而影响攻击结果。

除此之外，还需将L1 cache中相关set中的数据也驱逐到内存中。因此找到快速且可靠的驱逐方式是至关重要的。驱逐是否成功可以通过探测待驱逐的地址是否仍在cache中来判断。

Gruss et al. 发现了有三个因素对驱逐的成功率有影响，并将其作为可调整的驱逐策略参数：

1. 只有在能够映射到同一个cache set中的地址的cache hits和cache misses会对驱逐的成功率有非负的影响。这可以通过在相关地址中添加能够映射到其他cache set的地址来验证，并可以发现随机的非关联地址不会对平均的成功率产生影响。因此驱逐策略的有效性依赖于驱逐set的大小。
2. 此外，对于cache来说地址是不可区分的，因此访问模式被定义为一个地址系列，比如，其中每个标号代表一个不同的地址，这个系列定义了地址访问的一个先后次序，先访问等等。如果这个模式定义在一个循环中，则每个循环中访问的不同地址数会对驱逐策略的有效性产生影响。
3. ARM平台的cache替换策略倾向于驱逐最近添加到cache line中的数据，因此需要重复的访问相同的地址来保证地址被保存在cache中。比如，将驱逐序列从到缩短了超过33%的执行时间，并且增加了驱逐率。此外，在一定次数的重复之外，再增加访问次数不会增加驱逐率，或许还会更差。

基于这些观察，Gruss et al. 定义了依赖于cache以及cache的替换策略的三个可调整参数的驱逐模式，以供不同设备的调整以得到最佳的驱逐策略。

|  |  |
| --- | --- |
| 驱逐算法 | |
| 输入：  N：待驱逐set中可以存放的不同地址数  D：每个循环访问的不同地址数  A：每个循环每个地址的访问次数 | |
| 输出：无 | |
| 1:  2:  3:  4:  5:  6: | for i = 0; i < N – D; i++ do  for j = 0; j < A; j++ do  for k = 0; k < D; k++ do  Access(i + k)访问第i + k个相关地址  end for  end for  end for |

因此，为了保证驱逐的成功率，需要对具体的设备做大量的实验以得到快速有效的驱逐模式。再此上才能保证行而有效的cache攻击。

## 攻击方式设计

在能够获取精确的系统时间来对驱逐操作计时以及快速高效可靠的驱逐方式的基础上，还需考虑攻击程序对待攻击程序的攻击交互方式。之前的一些实验结果是在攻击程序触发待攻击事件的条件下测量的，比如，攻击程序首先驱逐一个cache set，随后触发待攻击事件，最后再探测是否还占有cache set中的cache line来判断待攻击事件有没有占用到该cache set。在此情况下，通常能得到理想的实验数据，因为由一个程序触发攻击和待攻击动作，在攻击过程中保证攻击和待攻击程序在同一个核上运行（实际上是同一个程序），并且减少程序切换对随后需要访问的已占用set中cache line的部分甚至全部的驱逐。因此，在实际的攻击过程中需要考虑到系统切换和人为因素对cache set状态造成的影响。

在使用prime + probe操作对某使用AES加密算法实现的程序进行攻击的过程中，为了避免系统切换或人为因素对cache set状态造成的影响。由于攻击程序与加密程序不能合并在一个程序中，并且系统切换造成的影响无法消除（在攻击程序与加密程序切换执行的过程中一定需要操作系统的调度），在对每个set进行prime + probe操作测量访存时间时都引入系统调度这一因素，并将系统调度假定为一个常量，则所获取的时间为probe所测时间+系统调度导致的时间增量。由于每次set操作都引入了这一时间增量，cache hits和cache misses仍能通过threshold时间增量区分开来。

至于人为触发prime或probe操作对结果造成影响这一因素可以通过程序或脚本替代手工操作来解决。

## 异步攻击设计

## 关键技术解决方案

## 本章小结

# AES攻击验证

## AES同步攻击

## AES异步探索

## 本章小结

# 预防攻击措施

在前两章中，本文主要介绍了基于AcoreOs的执行重放方案的设计和详细的实现步骤。针对本文提出的执行重放方案，本章节主要从功能方面对方案的可用性和正确性进行验证。本章先是对提出的方案中各个模块的功能进行正确性验证，最后通过比对目标程序在记录执行阶段和重放执行阶段的结果来验证本方案对于多核特定应用的有效性和正确性。

## 攻击漏洞

## 预防攻击措施

## 实验结果

## 本章小结

本章完成了对提出的执行重放的正确性的验证，并量化了记录执行阶段所引入的插桩探针对系统的影响。首先，本章选择了几个典型的测试样例来分别验证了该执行重放方案对单核上同步、异步、数据流以及多核上控制流事件的正确重放，测试样例的正确重放说明该执行重放方案可应用于此类测试程序的重放。接着，本文通过比对插桩前后程序的运行时间，量化了记录执行阶段引入的探针对系统整体的影响，从测试结果可以看出引入的探针的确会对系统性能造成一定的影响。

# 总结与展望

## 工作与研究总结

当前，几乎所有的应用程序都借助了多线程、多任务技术，来最大化利用系统提供的性能。这些技术的引入给程序开发阶段的调试提出了很大的挑战，循环调试技术作为软件开发过程中的关键技术，可以辅助软件开发人员在软件开发阶段修复很多错误，然而，循环调试方法对于多线程、多任务的应用程序的某些错误的调试不再有效，而执行重放技术作为一项热门技术可以对多线程、多任务应用执行重放，从而辅助调试器对此类应用调试。现有的执行重放技术大多数都是针对于普通操作系统及其应用的执行重放，不能直接应用于多核实时操作应用程序的执行重放。因此研究用于多核实时操作系统下的应用程序执行重放技术是特别有意义的。

## 对未来工作的展望

上述主要讲述了本文的研究内容和取得的成果，但本文提出的执行重放方案并不完整，还存在一些问题需要解决，后续还有一部分工作需要完善和优化。

1. 本文虽然实现了对多核实时系统上应用程序的执行重放，但由于使用软件方式记录多核交叉内存访问顺序开销非常大，对系统的实时性影响特别大，无法满足系统实时性要求，从而本文未解决多核交叉内存访问问题，对于存在多核交叉内存访问的应用程序无法正确重放。
2. 本文提出的重放方案借助了交叉调试环境中的调试器功能，通过在目标系统上设置断点来进行确认任务的切换点，这种方式需要频繁的调用交叉调试器接口从目标机获取任务的相关信息，特别是中断等异步事件的重放，每次获取的信息量较大，导致获取速度慢，而且中断发生位置的不确定性，对于发生在循环体内的中断事件，比对操作将占用大部分时间，重放时间也会大大增加，后续需要找寻相应的解决策略来减少中断事件重放所需要的时间。
3. 目前探针都是源代码级别的，插桩需要花费一定的时间，插桩位置也需要专业的人员才能完成，后续可开发自动化插桩工具，自动化完成对系统提供的编程接口的插桩记录以及对数据流事件的插桩记录。
4. 对于多核交叉内存访问导致的数据竞争，由于记录的开销太大，无法满足系统的要求，本文中没有给出切实有效的解决方案，不过根据已有的研究表明可借助硬件仿真器的支持，完成对交叉内存访问的记录，并完成相应的重放工作，由于本实验环境缺少硬件仿真器支持，并没有验证硬件仿真器对于解决多核内存交叉访问的有效性。

# 参考文献

1. Chen Y., Zhang S., Guo Q., et al. Deterministic Replay: A Survey[J]. ACM Computing Surveys (CSUR), 2015, 48(2): 17-63
2. Konuru R., Srinivasan H., Choi J.D. Deterministic replay of distributed java applications[A]. Proceedings of the 14th International Symposium on Parallel and Distributed Processing [C], 2000: 219-227
3. Pokam G., Pereira C., Dannse K., et al. Architecting a chunk-based memory race recorder in modern CMPs[A]. Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture[C], 2009: 576-585

# 附 录

同步事件重放样例代码分为两部分：生产者任务执行代码和消费者任务代码。

# 攻读硕士学位期间得到的学术成果

[1] **Chunlei Ma**, Xiang Long, Bo Jiang, Zhenyu Zhang. Replay Debugging of Real-Time Vxworks Applications[A]. International Workshop on Dependable Software and Applications(DSA)[C],IEEE, 2015: 568-573.(Accepted).

# 致 谢

岁月流逝，我在北京航空航天大学两年半的硕士研究生生涯即将结束。回首两年半学习和生活，不禁感触颇多。从2013年刚刚以一个新生的身份踏入北航校园至今临近毕业的两年半时光里，我曾激情万分，对知识充满渴望，对新学校充满好奇，然而也有过迷茫，迷茫是否跟随社会潮流追风逐浪，还是潜心目前的技术研究；迷茫学校里所学的技能是否能通过社会对我们的检验，跻身于技术达人行列；迷茫两年半的付出是否能够得到回报，找到一份满意的工作。两年半的时光虽然即将结束，然而我在这两年半中的收获却跟随自己一生终生受益，在北航里，我学会了如何学习、如何工作、如何与师兄师弟们相处交流，也让我找到了自己的方向和目标，更让我体会到了在技术一流的学府中，应该如何虚心受教，自我鞭策，自我总结，不断进步。为此，我心中充满了对各位老师、同学、家人的感激之情。

首先衷心感谢我的导师龙翔教授，感谢龙老师两年半以来对我的严格要求和耐心指导。在攻读硕士研究生学位期间，我非常荣幸能够成为龙老师的学生，跟随龙老师进行学术研究。龙老师不仅学时渊博、治学严谨，而且为人和蔼，平易近人，热爱生活，热爱技术，是我一生学习的楷模。

此外，要感谢姜博教授，他在我整个硕士研究生生涯中给予我无私的指导和督促。姜老师经验丰富、知识渊博、思维敏锐，是科研项目的带头人。每次科研上遇到疑惑或难点，姜老师总会耐心引导，不厌其烦的为我解答讲解，姜老师的深入讲解不仅帮助我解决了难题而且也完善了我整个的知识体系，进一步加深了对知识的理解，同时，也提高了我发现问题、分析问题、解决问题的动手能力。姜老师对学术的无限执着、对真理的不断追求、对自己的严格要求也是我今后步入社会学习的榜样。

感谢万寒老师一直以来指导我的工作和学习，也感谢万老师在实验室的管理工作，给我营造了一个良好的学习和工作环境。同时，非常佩服她的敬业精神，为整个实验室的良好运营做出了巨大的贡献。

感谢和我一起学习工作的郭兰、李力、马凯，感谢他们在学习和生活上给予的无私鼓励和关怀，正是他们的鼓励才使我坚持下来顺利完成了这个课题！同时，也要感谢实验室的学弟学妹们，他们的到来为实验室注入了新的血液，新的活力，他们乐观、活泼、开朗的个性一直感染我，给我的科研生活平添了很多色彩。研究生阶段我最大的收获是能够进入嵌入式系统实验室，从事自己最感兴趣的方向，大家在科研项目上同舟共济，生活生互帮互助，给实验室营造了非常好的氛围。与大家在一起的美好时光，必将是我人生中的一笔财富，也是我学生时代美好的记忆。在这里，我也祝愿实验室的所有学弟学妹都能顺利毕业，找到满意的工作，跻身于技术达人行列。

特别要感谢的是我的父母，感谢他们的养育之恩，他们虽然没有什么文化，地位卑微，但深知读书是农村孩子唯一好的出路，在养育我25年的漫长岁月里，父母给了我无微不至的关怀，激励我发奋图强，不断进取。他们的付出和期盼是我能迈出农村进入大学最大的动力源。

最后，感谢各位老师在百忙之中抽出宝贵的时间对我的论文进行评阅和审议，谢谢您们。