**北京航空航天大学计算机学院**

**硕士学位论文中期检查报告**

**论文题目：**面向移动设备的cache攻击关键技术研究

**专 业：**软件工程

**研究方向：**软件工程

**研 究 生：**李勃

**学 号：**SY1506402

**指导教师：**姜博副教授

**北京航空航天大学计算机学院**

2017年8月30日

目 录

[**1 论文工作计划** 1](#_Toc491596792)

[1.1 论文研究背景和意义 1](#_Toc491596793)

[1.2 论文研究目标 2](#_Toc491596794)

[1.3 论文研究内容 3](#_Toc491596795)

[*1.3.1 实现精确计时方式 3*](#_Toc491596796)

[*1.3.2 实现有效的驱逐策略 4*](#_Toc491596797)

[*1.3.3 实现高效攻击方式 6*](#_Toc491596798)

[*1.3.4 实现keylogger工具 6*](#_Toc491596799)

[*1.3.5 实现AES密钥攻击 7*](#_Toc491596800)

[**2 已完成的工作** 8](#_Toc491596801)

[2.1 通过POSIX实现精确计时 8](#_Toc491596802)

[2.2 实现evict + reload和prime + probe 8](#_Toc491596803)

[2.3 访存操作对probe时间的影响 9](#_Toc491596804)

[2.4 Prime + probe异步Aes攻击实验 12](#_Toc491596805)

[2.5 实现keylogger攻击 14](#_Toc491596806)

[2.6 实现AES攻击并获取第一轮密钥 18](#_Toc491596807)

[**3 关键技术及难点** 20](#_Toc491596808)

[3.1 确定驱逐策略 20](#_Toc491596809)

[3.2 探索异步攻击的方式 20](#_Toc491596810)

[3.3 探索对AES进行攻击的方式 21](#_Toc491596811)

[**4 下一阶段工作计划** 21](#_Toc491596812)

[4.1 存在的问题 21](#_Toc491596813)

[4.2 尚未完成的工作 21](#_Toc491596814)

[4.3 解决问题的技术思路或措施 21](#_Toc491596815)

[4.4 下一阶段计划 21](#_Toc491596816)

[**5 主要参考文献** 23](#_Toc491596817)

# 论文工作计划

## 论文研究背景和意义

随着移动互联网近些年突飞猛进的发展，手机等移动设备已经成为我们生活必不可少的一部分。它在给我们生活带来便利的同时，也给我们到来了潜在的威胁，尤其是在手机能方便管理资产，存储着各种隐私的今天。因此，随着科技的进步，安全问题也越发得到人们的重视。虽然手机等设备的开发厂商以及Android系统的开发公司采取了各种措施来保护用户的隐私安全，包括可信执行环境，虚拟内存管理，权限管理等。然而，由于电脑，手机等设备结构设计的缺陷，并随着安全领域不断的研究，一些手机设备的安全漏洞和攻击方式逐渐的显露出来，其中包括cache旁路攻击。Cache旁路攻击是通过探测程序执行时cache的hit或miss，访存时间的多少等信息来进行攻击的方法。其中攻击程序与被攻击程序没有直接的交互，他们并行执行在同一个核或不同的核上，访问各自的地址空间，因此攻击时不需要多余的权限。

在最近10年中，基于x86平台的CPU cache攻击得到了人们越来越多的重视。Kocher[19]和Kelsey[18]等人提出了通过分析高速缓冲存储器在运行时泄露出的旁路信息来破解计算机中的加密算法的方法。这一思想在计算机安全相关人员的重视下得到快速的发展。在最近几年不断提出了在x86平台上的cache攻击方法，并在监测用户键盘输入，AES T-table加密破解方面得到了实验，也证明了cache攻击的有效性。比如，Page等人提出了一种针对DES加密技术的cache模型攻击的方法，将DES密钥的搜索空间从56位降低到32位。Tsunoo[34]通过分析DES查表索引以及Cache的访问特征首次实现了针对DES的Cache 攻击，并在一台600-MHz Pentium III 的个人计算机上，通过223个明文样本成功获取DES 全部密钥。Bernstein[38]在强制消除网络传输时延条件下实现了一种针对OpenSSL[27]中AES 的远程时序驱动计时攻击，其中加密服务端采集AES 加密时间。Bonneau[5]提出了一种利用密码程序在加密中的内部数据访问冲突导致的Cache 命中和失效信息来进行密码分析的方法。Percival[30]提出多线程间共享Cache 存储器访问引发的漏洞，它不仅提供了线程间一个简单、高带宽的隐秘通道，也给恶意线程监视其他线程提供了入口，使得恶意线程能够窃取加密密钥，他依次设计实现了一种针对RSA 的计时攻击。Osvik[29]借鉴Percival信息采集方法实现了多例针对AES 的Cache 计时攻击。Neve[24][25][26]将Osvik攻击[33]切入点转移到最后一轮，提出了一种新的AES 最后一轮访问驱动的Cache计时攻击方法。可见，通过这些漏洞，攻击者不仅可以获得用户键盘输入等的隐秘信息，还能获得价值更高的密钥等信息，进而威胁到用户的财产安全。

然而，由于Android等移动设备的体系结构与Intel x86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攻击相关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的手机上的有效的跨核 cache攻击方法。Moritz Lipp et al. [21] 提出了通过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，TEE相关的共享库可能泄露的漏洞，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。

## 论文研究目标

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。

## 论文研究内容

### 实现精确计时方式

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。

* 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\*/

};

* 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

* 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。

### 实现有效的驱逐策略

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。

### 实现高效攻击方式

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。

### 实现keylogger工具

通过cache攻击能够在Android平台上获取使用共享库的用户输入信息，例如用户敲击键盘的输入。对用户键盘攻击时，首先剖析用户在输入特定按键时共享库地址的cache hits和cache misses的情况，并统计出一个cache攻击模板矩阵。每个按键均对应着一个矩阵，因此若矩阵可分，则可通过这些矩阵将用户输入区分开。在探测阶段则通过用户输入记录的cache hits和cache misses的矩阵推测出用户的输入。

为了执行对用户按键输入的cache模板攻击，攻击者需将按键相关的共享库或共享二进制文件映射到其地址空间中。通过使用共享库，攻击者能够绕过操作系统对其他app运行时数据访问的限制，并且能够在非特权模式的情况下展开攻击。

### 实现AES密钥攻击

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。

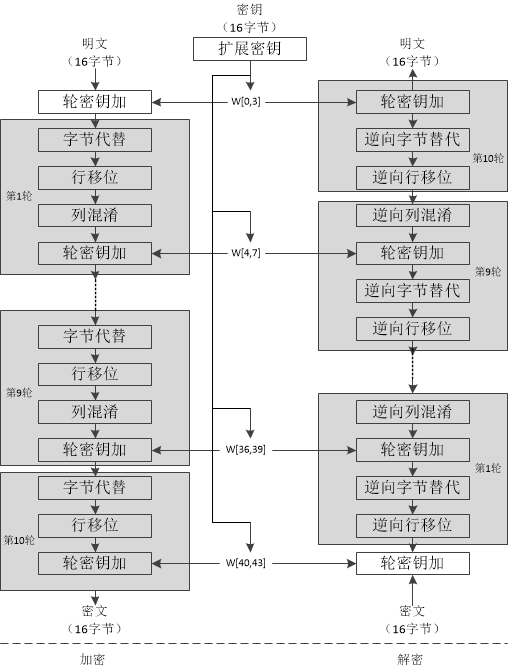


图1 AES加密步骤

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。

# 已完成的工作

## 通过POSIX实现精确计时

"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\*/

};

## 实现evict + reload和prime + probe

为了确定访存与访问cache的区别，并求出访存与访问cache的时间阈值。设计了这组实验，实验过程分为两组，第一组实验测量cache命中的时间，测量方法如下：

* 测量访cache时间：

access\_memory(address);//将address中的数据加载到cache中

uint64\_t time = get\_timing\_start()//获取当前系统时间

access\_memory(address);//访问address

uint64\_t delta = get\_timing\_end(session) - time;//计算访存时间

* 测量访内存时间：

evict(address);//将地址所在cache中的数据驱逐出去

uint64\_t time = get\_timing\_start()//获取当前系统时间

access\_memory(address);//访问address

uint64\_t delta = get\_timing\_end(session) - time;//计算访存时间

为了避免偶然因素对实验结果造成的影响，通过增加测量次数的方式来减少误差，分别对访cache和访内存进行50000次实验，并分别将其众数作为所测时间。实验结果如下：

Cache access time: 765ns

Memory access time: 1000ns

Threshhold: 883ns

## 访存操作对probe时间的影响

本节通过实验探究实际应用攻击场景下set的cache line miss对probe操作时间的影响。实验目标为Lenovo K51c78（ARM v8-a处理器， cache共有512个set，每个set为16路，每路大小为64B），被攻击程序为一个简单app，点击该app上的ACCESSSET按钮将执行针对相关set的访存操作，该程序的截图如2所示。



图2 被攻击程序界面

为了实现异步攻击，攻击程序为另一个app，该app基于POSIX计时方式，使用prime + probe攻击方式的，攻击程序的界面如图3所示。

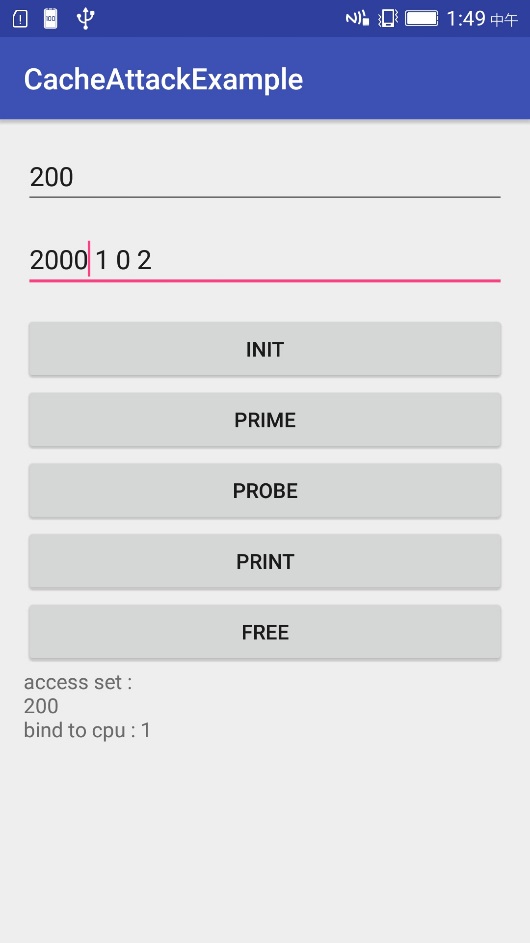


图3 攻击程序界面

点击prime操作后将对第一个输入栏中的set进行evict操作，点击probe操作将探测prime阶段读入cache set中的数据是否还在该set中。

具体的实验过程如下：

1. 设置cachesetaccess程序访问的set号，本实验设置为200，及点击accessset按钮后该app将通过一条访存语句占用该set的一个cache line。
2. 设置攻击程序的配置信息，在第一个edittext文本框中输入待探测的set号，第二个edittext文本框中输入重复探测的次数以及访问模式等信息。
3. 点击init按钮，攻击程序找到待探测set的相关地址并将其存储到数组中
4. 点击prime按钮，执行将制定set的eviction操作，将相关地址中的数据读入该set中
5. 切换到cachesetaccess程序，点击accessset按钮执行访存操作
6. 切换回攻击程序后点击probe按钮并返回probe时间
7. 重复过程4到6，知道重复次数达到所设置的值
8. 重复一次整个过程，不过这次切换到cachesetaccess后，不点击accessset按钮
9. 记录下实验结果并分析

由于需要排除系统切换或攻击程序本身访存操作对prime阶段占用的cache set的影响。因此需要每次对尽可能少的cache set做prime probe操作，并对多个set进行实验。本实验仅仅对单个set进行prime probe操作，set号为150，并将有访存操作的probe时间与没有访存操作的probe时间进行对比。实验结果如图4所示。

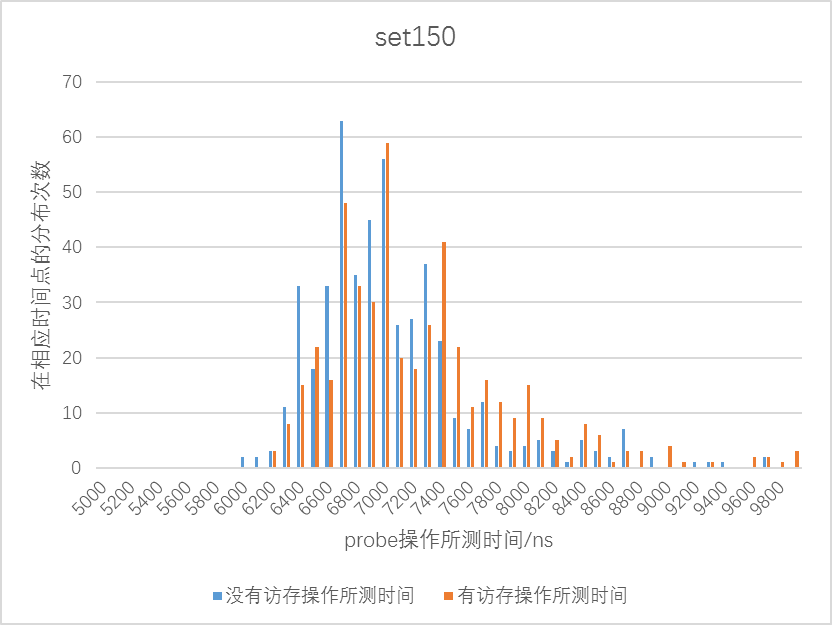


图4 set150实验数据

其中横坐标表示probe操作所花时间，单位为纳秒。纵坐标为probe操作所得时间在横坐标相应的时间点上统计的次数。其中蓝色条块表示没有执行访存操作所得结果，红色条块为相对的执行访存操作所测结果。根据图5.3.1可看出，红色条块的分布情况为蓝色条块分布向右偏移300ns的结果，及访存操作增加了probe操作的时间，且该时间大致等于一次cache miss（1000ns）与cache hit（765ns）的时间差。因此验证了当前实现的prime + probe攻击方式能够有效的被用于实现异步的cache 攻击。

## Prime + probe异步Aes攻击实验

为验证prime + probe能够探测到AES操作所占用的cache set，本实验使用同上节实验相同的目标机和攻击程序，异步的对一个能够执行Aes加密的app进行攻击实验。该app的界面截图如图5所示。

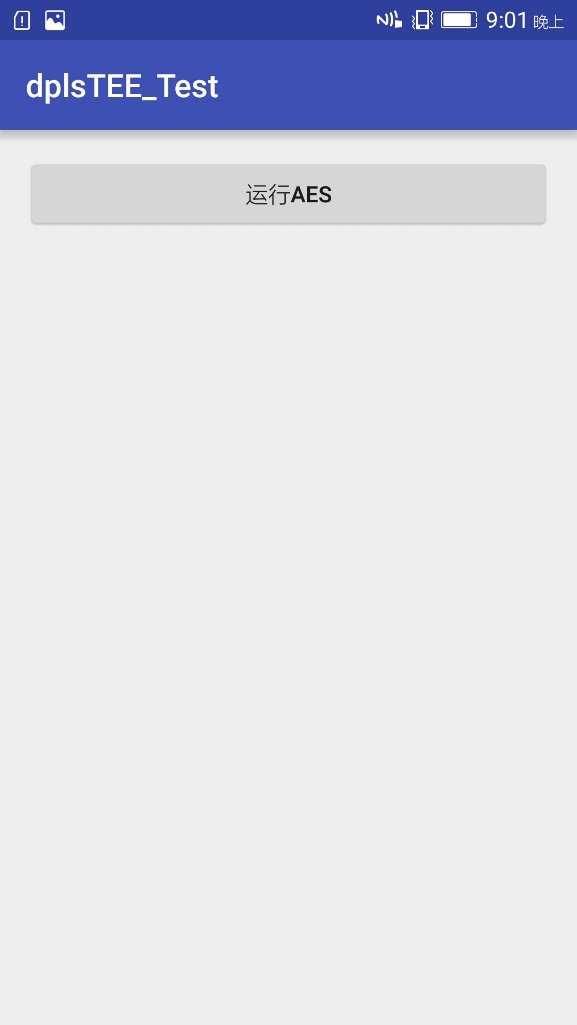


图5 AES程序

为了检测prime + probe攻击的有效性，针对多个set进行试验，通过在AES的查表过程中进行插桩，打印出执行AES加密操作时访问的cache set号，将set分为两类，一类为可能未占用set，表示AES操作没有显示说明占用的set，另一类为占用set，表示AES操作中通过插桩打印出执行过程中占用了的set。其中可能未占用set又可分为两类，未占用set和隐式占用set，因为在AES操作中仅仅打印出部分访问的set号，且AES或系统切换的访存操作访问的set并未打印出来。实验结果3个图如下所示，图6表示显示占用的set，图7和图8均表示可能未占用set。

图6 显示占用set

图7 可能未占用set

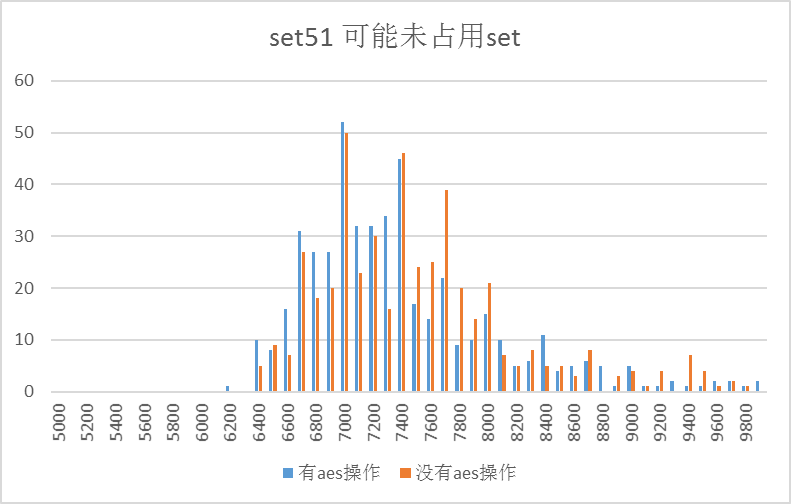


图8 可能未占用set

其中横坐标表示probe操作所花时间，单位为纳秒。纵坐标为probe操作所得时间在横坐标相应的时间点上统计的次数。其中蓝色条块表示没有执行访存操作所得结果，红色条块为相对的执行访存操作所测结果。通过图6可看出，prime + probe检测到了Aes中的访存操作对cache造成的影响，其实验结果相较于没有Aes操作的结果向后偏移了一个cache hit和cache miss的时间差。而图7中的有Aes操作和没有Aes操作的统计结果类似，表示在执行AES操作的过程中没有访问到该set，因此probe计时相同。图8中操作的set虽然为可能未访问的set，但有AES操作的结果相对于没有AES的统计结果有一个向右的偏移。出现这种情况是因为本实验并没有将执行AES中的所有访存操作访问的set都打印出来，而该set应该属于执行AES操作访问到的其他相关set。综述可得实现的prime + probe能够有效的对AES T-table进行异步攻击。

## 实现keylogger攻击

通过cache攻击能够在Android平台上获取使用共享库的用户输入信息，例如用户敲击键盘的输入。对用户键盘攻击时，首先剖析用户在输入特定按键时共享库地址的cache hits和cache misses的情况，并统计出一个cache攻击模板矩阵。每个按键均对应着一个矩阵，因此若矩阵可分，则可通过这些矩阵将用户输入区分开。在探测阶段则通过用户输入记录的cache hits和cache misses的矩阵推测出用户的输入。具体操作流程如下：

|  |  |
| --- | --- |
| 启动用户数据按键模拟程序并用Cache攻击获取数据的算法 | |
| 输入：  s：需要模拟的用户输入事件集合  input\_simulator：用户输入按键事件模拟程序  input\_simulator(c) 表示模拟用户输入按键事件c  cache\_attack：Cache攻击程序 | |
| 输出：无 | |
| 1:  2:  3:  4:  5:  6:  7: | **for** c **in** s  **if** ISRUNNING(input\_simulator) **then**  KILL(input\_simulator)  MULTIRUN(input\_simulator(c))  RUN(cache\_attack)  **if** ISRUNNING(input\_simulator) **then**  KILL(input\_simulator) |

上面的伪代码中ISRUNNING函数检测程序是否在系统中运行，KILL函数杀死特定程序所在的所有进程，RUN函数在本进程中启动某个程序，MULTIRUN另外创建一个进程来运行特定的程序。

数据的处理模块可以通过调用以下函数逐步实现：

|  |  |  |
| --- | --- | --- |
| 执行次序 | 函数名称（包含模块名） | 函数功能 |
| 1 | ***main.getlogfile*** | 给定文件名列表，从Android系统的目录下拉取送所有的文件log文件到指定的本地文件夹目录中 |
| 2 | ***database.initdatabase*** | 新建新的Sqlite3数据库 |
| 3 | ***evaluate.getdatas*** | 从指定的文件夹中读取适合的log文件并解析、清洗并初步处理其数据，得到各个地址的平均值、最大值、最小值等，并将得到的结果存储到特定的sqlite3数据库中 |
| 4 | ***evaluate.makematrixes*** | 通过sqlite3中的数据筛选出一些特征明显的地址并计算其阈值 |
| 5 | ***evaluate.getkeyaddressesandmatrix*** | 通过已经筛选好的地址的相关数据以及其阈值获得相关矩阵 |
| 6 | ***evaluate.writeconfigurationfile*** | 将获得的相关矩阵以特定的格式写入文件中，以备下一个模块的监测程序进行读取 |

在本模块的实现过程中，主要采用了Python的数据处理库Pandas。Pandas是Python语言的一个高效，易用的数据分析开源库，用户可以利用其方便快捷地处理数据大量的数据，并其处理的数据能够十分容易地转换成CSV，数据库文件，Excel，数据图表等多种表现形式。

在数据的清洗过程中，采用了Grubbs检验法来清除异常的数据。在一般的Grubbs检验法中，每一次清除异常数据应该选择所给数据中的最大值和最小值作为两个被怀疑的异常数据。而在本模块的实际实现过程中发现在实际获取的数据中，异常数据往往是远大于其他值的值。实际获得的数据的最小值一般为0（填充过的数据）或者1（原始数据），且其一般处在正常的数据分布范围内，不属于异常数据也。故在算法的具体实现中可以只考虑该数据的最大值是否是异常数据，以达到优化程序运行的目的。

使用Grubbs清理一组数据中的异常数据的算法用伪代码表示如下：

|  |  |
| --- | --- |
| 使用Grubbs检验法清除异常数据的算法 | |
| 输入：s：可能含有异常数据的数据集合。 | |
| 输出：s: 已经清理过异常数据的数据集合。 | |
| 1:  2:  3:  4:  5:  6:  7:  8: | **while** true  **if** s.length < 3 **then**  **break**  **if** (s.max - s.ave)/ s.std > getGP(s.length) **then**  s.remove(s.max)  **else**  **break**  **return** s |

其中的s.length, s.max, s.ave, s.std分别表示集合s的元素个数，最大值，平均值和样本标准差，getGP(s.length)可获得元素个数s.length对应的Grubbs临界值。

在具体数据阈值的计算过程中，采用了优化的KMeans聚类算法，设置K=2将每个地址的数据分为两类，然后计算值较大类的最小值和值较小类的最大值的平均值作为该地址的阈值。

通过单一地址监测来获得用户输入按键事件的测试结果是比较理想。如图9所示是对某个地址的监测数据，该地址的阈值为30，可以从该地址可以推断出在测试后期进行了多次特定用户事件的触发。然而事实上的情况是在测试后期进行了5次连续的按键事件，这均可以在图像横轴区间为[2906,3072]的图像中得到验证。

图9 单一地址显示单一按钮事件

对多个地址全部按键事件的检测,测试实验设计为：以间隔1s的方式逐个输入英文句子“this is just a test”，检测攻击程序能否识别用户的输入事件。

在初期的设计与实现过程中采用了使用尽量少的地址进行监测的方案，其监测的地址为：0x18580、0x15c0、0x10980、0x13940、0x1b80 、0x10940、0x27c0

然而其结果并不理想。

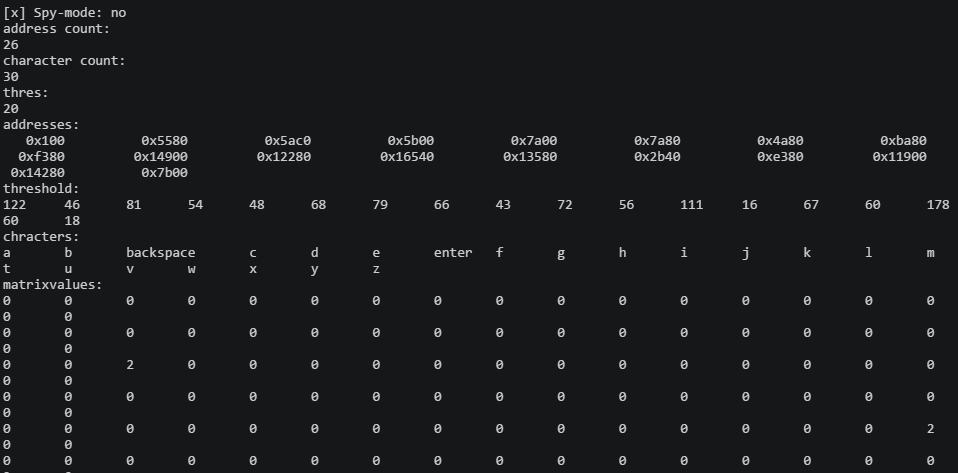


图 10 程序执行截图

之后采用了使用特定地址判定单一用户输入按键事件的方案，得到了较为理想的测试结果：基本上每个按键输入事件均能够被检测出来，但是仍然存在误报和结果不准确的情况，但这在实际的应用中是具有较大意义的，能够极大地缩小密钥等可能取值的空间。程序执行时如图10所示。

## 实现AES攻击并获取第一轮密钥

本文使用prime+probe的方式对Aes进行攻击，攻击主要分为第一轮和第二轮攻击，第一轮攻击主要根据可以通过密钥和明文获取第一轮t-table的访问索引，及给一个16字节的密钥k = (k0,… ,k15)，在Aes加密过程中会扩展到10轮的内部密钥，其中r=1,…,10，每一轮密钥被划分为4个字，每个字4个字节，表示为，其中第一轮扩展密钥时访问的T-table索引可由p^k求得，及明文的每个字节与密钥的每个字节相异或得到的就是第一轮攻击时访问到的table索引，其中第0、4、8、12字节求得的索引访问的是table0，第1、5、9、13访问的是table1，2、6、10、14访问的是table2，3、7、11、15访问的是table3。而第二轮加密访问到的索引比较负责，通过数学推到，可得到第二轮访问索引如下：

⊕⊕⊕⊕2•⊕⊕3•⊕⊕⊕

⊕⊕⊕⊕3•⊕⊕⊕⊕⊕⊕

⊕⊕⊕⊕⊕⊕⊕⊕⊕⊕⊕⊕1

⊕⊕⊕⊕⊕⊕⊕⊕⊕⊕⊕⊕

其中x2表示中间密钥的第三个字节，x5,、x8、x15同理，其中x2对应的是table2的索引，x5对应的是table1的索引，x8对应的是table0的索引，x15对应的是table3的索引，这些索引是实现Aes攻击的关键。本文通过假设检验的方式来进行Aes攻击，及遍历所有密钥可能的取值。根据假设的取值和密钥可以计算出相应访问的索引，再和实际访问索引进行对比则可获得该假设的可信度。

也就是首先枚举密钥可能的取值，也就是对于密钥中的每一个字节，从0到255有256种可能，且16个字节得同时进行猜测，及控制变量法。对于每一次猜测，可以获得一个度量分数m，根据m的高低则可判断哪个猜测组合的可能性最高，即可猜测出Aes的密钥。其中度量分数主要是通过获取与该猜测密钥k在Aes t-table上对应关系决定的。密钥k与明文p相异或所得的值就是Aes在执行期间在t-table的查找索引，而table的查找索引对应到cache中的不同set中，最后反映出的就是对cache中set进行prime和probe操作统计时间所反映出来的时间差异。本文通过使用ks检验的方式来获取度量分数，通过ks检验来判断两个分布是否相同。其中第一个分布的样本获取的是prime之后紧接着probe所测得时间，其中第二个样本获取的是prime之后执行Aes加密操作后对table所映射到的set进行的probe操作所获取的时间。此时如果ks检验得到这两个样本总体属于同一个分布，则表示在执行Aes期间Aes并没有访问到这个set。KS-检验（Kolmogorov-Smirnov test） -- 检验数据是否符合某种分布Kolmogorov-Smirnov是比较一个频率分布f(x)与理论分布g(x)或者两个观测值分布的检验方法。其原假设H0:两个数据分布一致或者数据符合理论分布。D=max| f(x)- g(x)|，当实际观测值D>D(n,α)则拒绝H0，否则则接受H0假设。KS检验与t-检验之类的其他方法不同是KS检验不需要知道数据的分布情况，可以算是一种非参数检验方法。当然这样方便的代价就是当检验的数据分布符合特定的分布事，KS检验的灵敏度没有相应的检验来的高。在样本量比较小的时候，KS检验最为非参数检验在分析两组数据之间是否不同时相当常用。

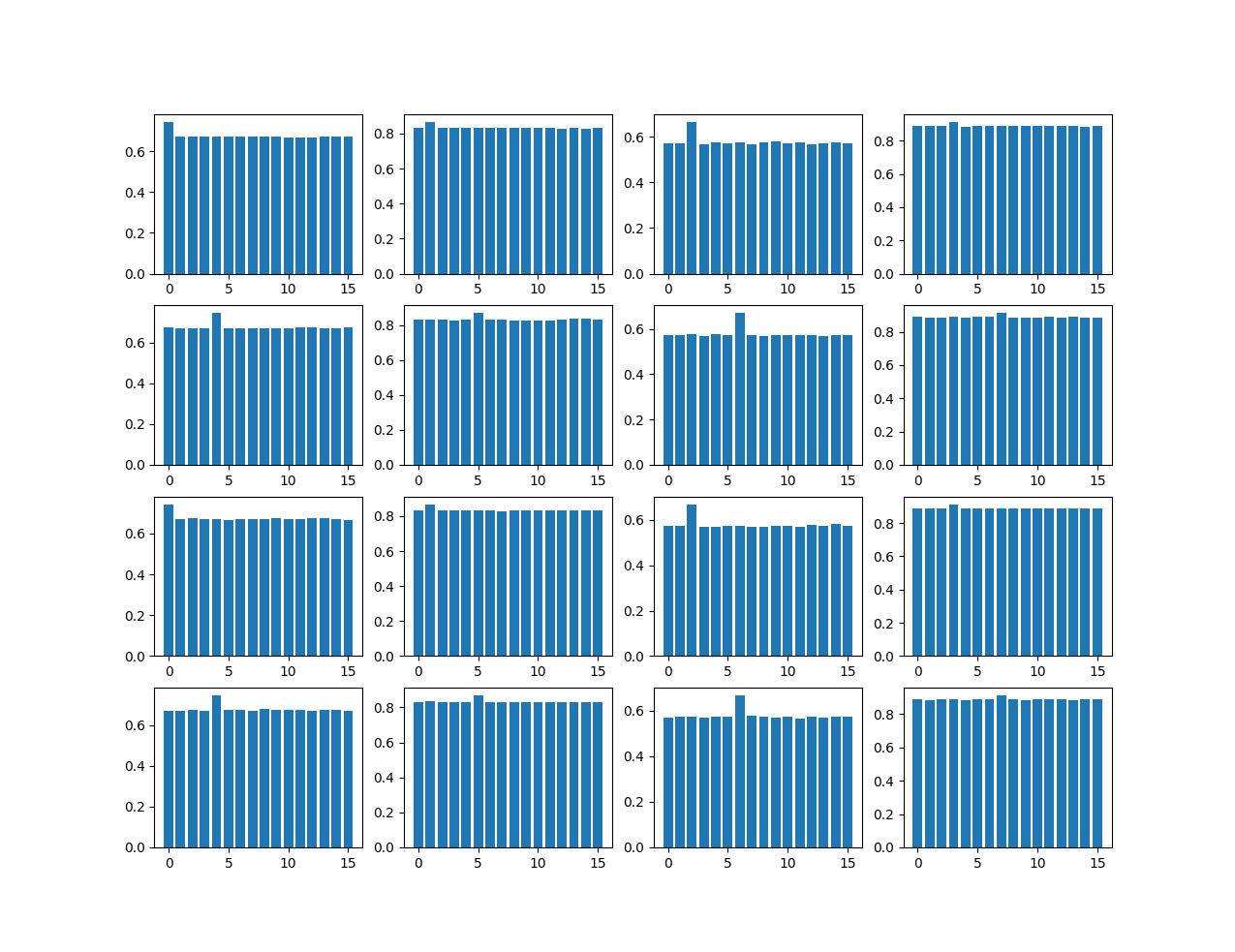
本文对Aes进行了第一轮攻击，并能够成功获取到密钥中每一位的前4个位，结果如下所示：

图 11 Aes第一轮攻击结果

在实验中使用到的密钥k=（0x00,0x01,0x02,0x03,0x04,0x05,0x06,0x07，0x00,0x01,0x02,0x03,0x04,0x05,0x06,0x07），入托11所示，能够比较明显的获取到密钥的前4位。

而对于密钥的后4位来说，在第二轮攻击中，也是通过假设验证的方式来获取密钥值。方法与第一轮攻击类似。不同之处在于第一轮攻击每一个密钥字节与明文相异或能独立的获取在table上的一个索引，比如p1^k1可得出Aes在加密第一轮访问table1的索引，也就确定了将table1对于的数据缓存到cache中相应set中的其中一个line中。而第二轮攻击中table中的任意索引都有可能与密钥中的多个字节相关，比如x2 = S[p[0] ^ k[0]] ^ S[p[5] ^ k[5]] ^ int(GF256(2) \* GF256(S[p[10] ^ k[10]])) ^ int(GF256(3) \* GF256(S[p[15] ^ k[15]])) ^ S[k[15]] ^ k[2]是第二轮访问到table2的一个索引，可见该索引与k0，k5，k10以及k15都相关，需要将k0，k5，k10，k15值进行统一遍历才能获得一个索引，其中每一密钥字节还有4个位需要猜测，总共16位，一共需要65535次猜测并验证才能获取到密钥值。这样由于猜测次数较多，Aes后8轮对table的访问容易对猜测结果造成影响。也是由于这个原因，第二轮获取后4位密钥值目前效果不佳。

# 关键技术及难点

## 确定驱逐策略

驱逐策略关系着cache set能够快速高效的驱逐，不仅仅要将L2 cache相关set中的所有line中的数据驱逐，也要讲L1 cache中的相关数据驱逐到内存中。然而由于大多数ARM处理器使用的L2 cache伪随机替换策略，要在连续的访存中估计cache set中的数据显得困难。虽然在不支持flush刷新的情况下可以使用相关地址来驱逐set，通过大量访问相关地址达到驱逐目的地址到内存的目的，但这不仅会增加时间和空间存储的开销，也会引入探测阶段的偏差（由于不清楚是哪些地址占用了set）。由此在确定机型上找到快速高效的驱逐策略是难点之一。

## 探索异步攻击的方式

Cache攻击的攻击程序与被攻击程序的交互方式是攻击能否成功的关键，然而在这个方向目前还没有太多相关的研究。Evict + Reload、Prime + Probe等方法只是探测cache set状态的工具，还需确定具体的攻击过程。由于在evict或prime过后需要被攻击程序执行待攻击的事件，之后再有reload或probe操作来获取cache的旁路信息，然而在这期间由于系统访存或攻击程序自身访存操作可能会对evict或prime阶段占用的cache set中的部分甚至全部line，导致探测阶段误认为待攻击程序访问了该cache set，因此如何处理系统噪音造成的影响也是攻击成功的关键。

## 探索对AES进行攻击的方式

由于AES加密过程涉及到4种操作：字节替代、行移位、列混淆和轮密钥加。加解密中每轮的密钥分别由初始密钥扩展得到。算法中16字节的明文、密文和轮密钥都以一个4\*4的矩阵表示。在对AES的T-table实现进行密钥攻击时，需要熟悉攻击过程以及每个过程中对那个table进行查找，并挖掘出可利用的内存位置，提取出AES可供利用的数学特性，这些均为AES攻击中的难点。

# 下一阶段工作计划

## 尚未完成的工作

由于keylogger目前仅是探测到有可能被点击的用户输入，噪音影响比较严重，对同一个地址进行监听时有可能有多个用户时间均处于可疑状态，未能明显的将多个用户事件区分开。下一阶段打算进一步减少噪音等干扰因素的影响，缩小结果的范围。

对于Aes攻击，在第二轮攻击获取密钥每个字节后4位的过程中，由于后8轮加密对table的访问使得攻击结果辨识度不高，基本table映射的每个set均被访问到，因此目前结果并不理想，接下来打算提高辨识度，获取密钥的所有位数据。

Aes攻击第二轮需要将密钥中4个字节为一组进行猜测，由于第一轮攻击已经获取到前4位，所以总共有（8-4）\* 4=16位进行猜测，共有6万多个可能性，对所有可能的组合进行检验，需要较多的时间，且由于受到后8轮加密对table访问的影响，恢复结果不理想。仅恢复Aes密钥每个字节的前4位，后四位还没有恢复。

## 解决问题的技术思路或措施

对于keylogger攻击，在监测程序用使用多个子进程分别对不同的地址进行监测，由父进程对多个进程的监测结果进行分析整合。由于多个进程的运行过程中不会相互影响，所以这样可能会有比在同一进程中轮流进行监测不同地址有更好的结果，可以使用机器学习的方法来获得较为理想的相关矩阵，机器学习能够很好地学习用户输入和库文件地址访问之间的关系，并不需要人为地设计相关矩阵的生成算法，其结果可能更为理想。

对于Aes攻击来说[28]，当前使用ks检验值来判断特定set在Aes加密过程中有没有被访问到，并对多次样本的ks值取平均值来获取一次猜测的度量分。由于ks检验的区分度较高，两个分布稍有不同就会对ks值造成较大的起伏，因此可以对ks值做处理得到较平和的度量分数。或取多次样本ks值的众数来确定一次猜测的度量分，排除极端样本对统计结果的影响。

## 下一阶段计划

下一阶段的工作计划如下表1所示：

表 1 研究计划表

|  |  |
| --- | --- |
| 规划 | 安排 |
| 2017.09 ~ 2017.10 | Keylogger工具完善，提高获取用户输入的精度 |
| 2017.10 ~ 2017.11 | 完善Aes攻击，实现对密钥后4字节的获取 |
| 2017.10 ~ 2017.11 | 撰写并发表小论文 |
| 2017.12 ~ 2018.02 | 整理材料，并撰写毕业论文，准备毕业答辩 |

# 主要参考文献

1. O. Acıiçmez, W. Schindler, Çetin K. Koç, Cache Based Remote Timing Attack on the AES[M]// Topics in Cryptology – CT - RSA 2007. Springer Berlin Heidelberg, 2007:271-286.
2. O. Aciicmez, Advances in side-channel cryptanalysis : microarchitectural attacks[J]. 2006.
3. D. J. Bernstein. Cache-timing attacks on AES[J], Vlsi Design IEEE Computer Society, 2005, 51(2):218 - 221.
4. A. Bogdanov, T. Eisenbarth, C. Paar, et al, Differential Cache-Collision Timing Attacks on AES with Applications to Embedded CPUs[M]// Topics in Cryptology - CT-RSA 2010. Springer Berlin Heidelberg, 2010.
5. J. Bonneau, I. Mironov, Cache-collision timing attacks against AES[J]. 2006, 4249:201-215.
6. P. Dan, Theoretical Use of Cache Memory as a Cryptanalytic Side-Channel.[J]. Journal of Arid Environments, 2002, 2002(10):393-446.
7. J. F. Gallais, I. Kizhvatov, Tunstall M, Improved Trace-Driven Cache-Collision Attacks against Embedded AES Implementations[C]// Information Security Applications -, International Workshop, Wisa 2010, Jeju Island, Korea, August 24-26, 2010, Revised Selected Papers. 2010:243-257.
8. J. F. Gallais, I. Kizhvatov, Error-Tolerance in Trace-Driven Cache Collision Attacks[J]. Cosade, 2011:222--232.
9. D. Gruss, C. Maurice, S. Mangard, Rowhammer.js: A Remote Software-Induced Fault Attack in JavaScript[M]// Detection of Intrusions and Malware, and Vulnerability Assessment. 2016.
10. D. Gruss, C. Maurice, K. Wagner, et al, Flush+Flush: A Fast and Stealthy Cache Attack[M]// Detection of Intrusions and Malware, and Vulnerability Assessment. Springer International Publishing, 2016.
11. D. Gruss, R. Spreitzer, S. Mangard, Cache template attacks: automating attacks on inclusive last-level caches[C]// Usenix Conference on Security Symposium. USENIX Association, 2015:897-912.
12. D. Gullasch, E. Bangerter, S. Krenn, Cache Games -- Bringing Access-Based Cache Attacks on AES to Practice[J]. 2011, 2010(1):490-505.
13. B. Gülmezoğlu, M. S. İnci, G. Irazoqui, et al, A Faster and More Realistic Flush+Reload, Attack on AES[J]. 2015.
14. G. Irazoqui, T. Eisenbarth, B. Sunar, Cross Processor Cache Attacks[C]// The, ACM. 2016:353-364.
15. G. Irazoqui, T. Eisenbarth, B. Sunar, S$A: A Shared Cache Attack That Works across Cores and Defies VM Sandboxing -- and Its Application to AES[J]. 2015:591-604.
16. G. Irazoqui, M. S. Inci, Eisenbarth T, et al, Know Thy Neighbor: Crypto Library Detection in Cloud[J]. Proceedings on Privacy Enhancing Technologies, 2015, 1(1):25-40.
17. G. Irazoqui, M. S. Inci, T. Eisenbarth, et al, Lucky 13 Strikes Back[C]// ACM Symposium on Information, Computer and Communications Security. ACM, 2015.
18. J. Kelsey, B. Schneier, D. Wagner, et al, Side Channel Cryptanalysis of Product Ciphers.[C]// Computer Security - ESORICS 98, European Symposium on Research in Computer Security, Louvain-La-Neuve, Belgium, September 16-18, 1998, Proceedings. 1998:97--110.
19. P. C. Kocher, Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems[C]// International Cryptology Conference on Advances in Cryptology. Springer-Verlag, 1996:104--113.
20. P. C. Kocher, Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems[M]// Advances in Cryptology — CRYPTO ’96. Springer Berlin Heidelberg, 1999:104--113.
21. M. Lipp, D. Gruss, R. Spreitzer, et al, ARMageddon: Cache Attacks on Mobile Devices[J]. Mundo Electrónico, 2016, 6(1):págs. 60-65.
22. F. Liu, Y. Yarom, Q. Ge, et al, Last-Level Cache Side-Channel Attacks are Practical[J]. IEEE Symposium on Security & Privacy, 2015:605-622.
23. C. Maurice, C. Neumann, O. Heen, et al, C5: Cross-Cores Cache Covert Channel[M]// Detection of Intrusions and Malware, and Vulnerability Assessment. 2015:46-64.
24. M. Neve, J. P. Seifert, Z. Wang, A refined look at Bernstein's AES side-channel analysis[C]// ACM Symposium on Information, Computer and Communications Security. ACM, 2006:369-369.
25. M. Neve, J. P. Seifert, Advances on Access-Driven Cache Attacks on AES[C]// Selected Areas in Cryptography, International Workshop, SAC 2006, Montreal, Canada, August 17-18, 2006 Revised Selected Papers. 2006:147-162.
26. M. Neve, Cache-based Vulnerabilities and SPAM analysis[J]. Doctor Thesis Ucl, 2006.
27. OpenSSL the open-source toolkit for SSL/TLS. 2005. http://www.openssl.org/
28. Y. Oren, V. P. Kemerlis, S. Sethumadhavan, et al, The Spy in the Sandbox: Practical Cache Attacks in JavaScript and their Implications[J]. Computer Science, 2015.
29. D. A. Osvik, A. Shamir, E. Tromer, Cache Attacks and Countermeasures: The Case of AES[J]. Lecture Notes in Computer Science, 2005, 2005:1-20.
30. C. Percival, Cache missing for fun and profit[J]. Proc of Bsdcan, 2005.
31. R. Spreitzer, T. Plos, Cache-Access Pattern Attack on Disaligned AES T-Tables[C]// Constructive Side-Channel Analysis and Secure Design. 2013:200-214.
32. R. Spreitzer, T. Plos, On the Applicability of Time-Driven Cache Attacks on Mobile Devices[C]// Network and System Security. 2013:656-662.
33. E. Tromer, D. A. Osvik, A. Shamir, Efficient Cache Attacks on AES, and Countermeasures[J]. Journal of Cryptology, 2010, 23(1):37-71.
34. Y. Tsunoo, T. Saito, T. Suzaki, et al, Cryptanalysis of DES Implemented on Computers with Cache.[J]. Proc of Ches Springer Lncs, 2003, 2779:62-76.
35. M. Weiß, B. Heinz, F. Stumpf, A Cache Timing Attack on AES in Virtualization Environments[M]// Financial Cryptography and Data Security. Springer Berlin Heidelberg, 2012:314-328.
36. Y. Yarom, K. Falkner, FLUSH+RELOAD: a high resolution, low noise, L3 cache side-channel attack[C]// USENIX Security Symposium. 2014:719-732.