Research of Precise Timing on Android Devices

DENG Liu-yu-qin[1,a],CAI Hong-Liu[1,b], CHEN Cai-sen[2,c], XUE Ting-mei[1,d],

YU Xi[1,e]

[1]Department of Information Engineering, Academy of Armored Forces Engineering, Beijing 100072, China

[2]Department of Science Research, Academy of Armored Engineering, Beijing 100072, China

[a]xtdengliu260@163.com,[b]caihongliu@163.com,[c]caishenchen@163.com, [d]tmxue@163.com, [e]568571668@qq.com

Keywords: Precise Timing; Android; ARM; CCNT

Abstract: Precise timing is an extremely important part in the cache timing attack. We usually use the time stamp counter (TSC) to timing the cache access on the x86 platform. To overcome the differences between the ARM platform and x86 platform, we show in this paper about how to implement precise cache timing on Android devices by using the cycle counter register (CCNT register), which is a part of the performance-monitor register (PMR). To show the timing result directly on the device screen, wemade an android application.

Foreword

Cache timing attack is one of the side channel attack, this kind of attack has been proved to be efficient to RSA, AES encryption[1][2]. But most of the cache timing attack achievement is about the PC(x86 platform), the cache timing attack based on the mobile devices just started in the recent years. With the rapid development of mobile internet, the security of the mobile devices with arm processor is more and more important now. So in this article we focused on the differences between the x86 processor cache and arm processor cache and made some conclusion about it.

The rest of the paper is organized as follows: Section II described related work. In Section III we discussed how we have investigated the method to precise timing on ARM devices. Section IV we have implemented the method in section III and showed the experiment results. In section V we concluded the paper.

Related work

Introduction of Cache timing attack

In the past, attacker's attack on the password system is based on the algorithm of password system and brute force the cipher text. This method takes time and space andits attack efficiency is low. If the system properly use advanced password such as AES, the encryption key is very difficult to be cracked. Personnel in these conditions, therefore, developed a new method---the side channel attack (SCA). SCA is refers to the use of information leaked during the cryptographic hardware or software system encryption process to recover the key. These information conclude time, power consumption, etc. SCA research has made certain achievements in the PC.

© 2015. The authors - Published by Atlantis Press

Cache timing attack is a side channel attack which makes use of the timing differences in the cache "hit" and cache "miss". In the process of cache access, whether the target information is in the cache memory can make a difference. If the target information is in the cache memory, we call it a cache "hit". If information is not in the cache memory, we call it a cache "miss". Because of the cache working mechanism, cache "hit" will be faster than cache "miss". According to this, we can make use of it, and monitor the hardware encryption process, and combine with the encryption method, then get the secret key. The cache timing attack can be divided in to three kinds: time-driven, trace-driven and access-driven cache timing attack. Take access-driven cache timing attack for example, the model of access-driven cache timing attack are showed in figure 1 and the details of it are as follows:

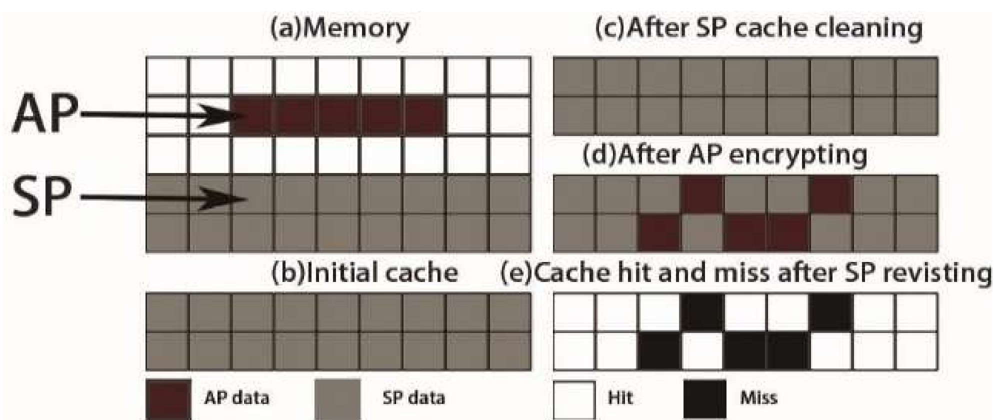

Fig 1. access-driven cache timing attack model.

(1) Cache information collection: when the encryption process (AP) is running, we let the spy process running at the same time. We call the memory block occupied by AP and SP in the main memory to ACB and SCB respectively. The size of ACB and SCB are showed in Picture1 (a), the size of cache are showed in Picture1 (b). Before the AP run, we start the SP first, so that the SP data will occupy all the cache. Then trigger the AP process, this action may eject some SP datafrom the cache, this is showed in Picture1 (d). Then, trigger the SP again, the ejected data will show a cache miss and other data will show cache hits. By comparing the time difference, we can get the accessed cache line collection and the not accessed cache line collection during the encryption.

(2) Cache information analysis: according to the collected side-channel information, we can get a collection of possible index value and a collection of impossible index value by using the mapping relationship of T table cache set and table index.

Timing method of the cache timing attack

On the x86 platform, which always employs the Intel CPU, we implement the precise timing by access Time Stamp Counter (TSC). This 64 bit counter is used for counting clock cycles. So after every clock tick this counter is incremented. To access the time stamp counter, we have to do it through the RDTSC instruction [3].This instruction saves the lower 32 bits to the EAX register and the higher 32 bits to the EDX register. On the 64bit system, we use RAX and RDX registers respectively. But the upper 32 bits of the register are cleared. Table 1 shows the usage of the RDTSC on the 64 bit system. There is a problem is that because of the Intel processor's out of order execution, this instruction will not be executed by the exact same order in the source code. This instruction also may wait for the previous instructions have been executed and then execute the RDTSC instruction. So this problem can distort the timing measurement, and cause the precise timing fail.

Table 1

# of NOPs	CPU frequencies		
	800MHz	1600MHz	2670MHz
16384	120	60	36
32768	120	60	36
65536	120	60	36

On the ARM platform, precise timing is different from the x86 platform. For the processors in the ARM devices are not with the same architecture as the x86 devices processors, which are belong to the Cortex-A8, Cortex-A9 architecture, the precise timing method should base on the registers and instructions of the ARM processors. On the ARM devices, we implement precise timing by Cycle Counter register (CCNT register), which is belongs to the Performance-Monitor Register. The details about the precise timing of the ARM devices will be talked about in the next section.

Precise timing on the ARM devices

In this section, we will introduce how to implement precise timing on the ARM devices step by step.

Android NDK

For implement precise timing on android devices has to monitor the hardware, we have to use the C language to implement this function. The existing android developing environment is Eclipse and Android Studio. These two android developing environment uses Java programming language to develop android project. To monitor and measure the hardware, we have to use the ASM and C language.

Android NDK (Native Development Kit) is a toolset that allows you to implement parts of your app using native-code languages such as C and C++. For certain types of apps, this can be helpful so you can reuse existing code libraries written in these languages [4]. As a developer, we should know that we should balance the benefits and drawbacks of the NDK. The NDK can help us program C/C++ in our project, but it also increases the complexity of our project. Typical good candidates for the NDK are CPU-intensive workloads such as game engines, signal processing, physics simulation, and so on. Precise timing is a workload CPU-intensive so NDK is the toolset that we have to use.

At first, we should download the newest version of NDK, and then start to use it. To use the NDK, we should at first define the JNI port in the java activity and then build it. Then find the .classes files to use javah command to make them to .h files(head files). Then we can write the C files according to these head files. Once we get the C files, we should write android make file to tell Android NDK how to make these files. With the make file, Android NDK can make the C files and head files to .so files(library files). Then we copy the .so file to the /lib files of the android studio project files and define the location in the build.gradle files of the project, the android app can call the functions in the C files.

Get the full authority of the mobile phone

To access the CCNT register, we should get the full authority of the mobile phone. So we have to root the mobile devices first. To install kernel module on the device, at first we need to download the kernel sources for the mobile devices we have. Then we need to compile the attached source code(enableccnt.c) to a kernel module. Then we can get a kernel module file(a .ko file) and we can

up load this file to our phone. Then we should load the kernel module in order to enable user-mode access to the performance counters. Therefore , connect with the ADB shell to the phone(the ADB shell comes with the Android NDK) and execute: insmod enableccnt.ko. Note that after a re-boot of the smartphone, the kernel module is unloaded and we need to load it again.

Use the CCNT to implement precise timing

After loading the kernel module and have user access to the cycle count register(user mode program on Android should be allowed) and then we should be able to execute the two functions get_cyclecount(), and init_perfcounters(). Use these two functions in the right location, we can get the cycle counts between two functions' executing. Combined with the CPU frequencies, we can calculate the time between the two functions' executing.

Experiment and results

In this section, we introduce the details of the experiment and show the measurements of experiment.

Introduction of experiment details

To show the results of the measurements, we develop an android application and in this application, we designed three button, the first button is to show the latest count numbers when we press the button1, the second button is to show the count numbers during the operations we designed. The third button is to initialize the timing function.

The operations to implement precise timing

At first, we should root the experiment device, then link the device to the PC to install module as showed in figure2 After install enableccnt.ko to the mobile devices, we should initialize CCNT by using asm code as showed in figure 3.

```
C:\windows\system32\cmd.exe - adb  shell
Microsoft Windows [版本 6.2.9200]
(c) 2012 Microsoft Corporation。保留所有权利。

C:\Users\D.L>adb shell
shell@android:/ $ su
su
root@android:/ # insmod /sdcard/enableccnt.ko
insmod /sdcard/enableccnt.ko
root@android:/ #
```

Fig 2 insmod enableccnt. ko.

```
value = 1;

// peform reset:

  value |= 2;     // reset all counters to zero.
  value |= 4;     // reset cycle counter to zero.
  value |= 16;

// program the performance-counter control-register:
asm volatile ("MCR p15, 0, %0, c9, c12, 0\t\n" :: "r"(value));

// enable all counters:
asm volatile ("MCR p15, 0, %0, c9, c12, 1\t\n" :: "r"(0x8000000f));

// clear overflows:
asm volatile ("MCR p15, 0, %0, c9, c12, 3\t\n" :: "r"(0x8000000f));
```

Fig3 Initialize CCNT.

And we can access the CCNT by using asm code "asm volatile ("MRC p15, 0, %0, c9, c13, 0\t\n": "=r"(overhead));"[5].

Results of the experiments

By writing according C language codes and asm language code in c to generate the lib files, we can import these lib files in Android Studio to use these functions writing in C and then we can access CCNT. As introduced in 4.1, we showed the results in an application by press button. As showed in figure 4.

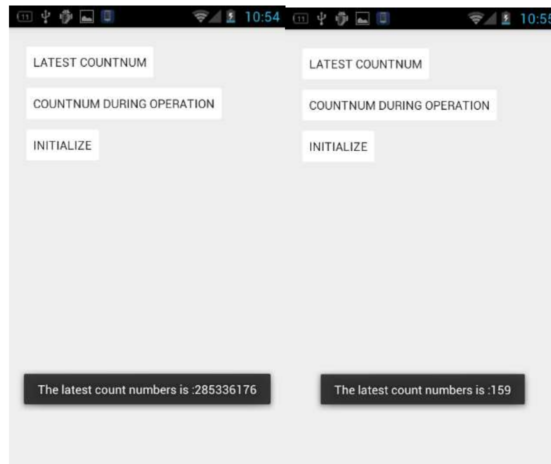

Fig 4 CCNT count results.

We have count the operation time for 100 times and results are showed in figure 5.

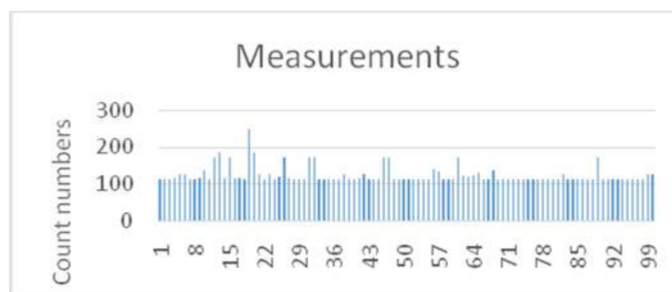

Fig 5 Analysis of measurements.

As showed in figure 6 The operation of reading a existed 10*10 two-dimensional array costs 110 cycles to 246 cycles, but the shortest and with the highest frequency is 110. So we guess this operation costs 110 cycles. For the CPU frequency of the experiment device is 1GHz, so we can calculate the time and get the results of $1.074*10^{-4}$ s.

Conclusion

In this paper, we have shown the method to implement precise timing on an android device with ARM processor. As we can see in this paper, because of the difference in the count register between the two kinds of processor. On the x86 devices which uses Intel processors, because of the system characteristic, we can easily get the full authority of the devices and use the Time Stamp Counter to implement precise timing. So that cache timing attack on the x86 platform always use the TSC. But for the safety reasons, the mobile devices with ARM processors are not as easy as the x86 devices to get the root authority. But if we want to implement precise timing, we have to get the root authority. So the experiments on in this paper have to get the root authority first and have to upload kernel source to the device. After doing so, we can get the real-time count number of CCNT by call two functions. And then we can calculate the precise timing between two calls with the frequency of the CPU.

In fact, compile the enableccnt.ko for our mobile devices is difficult because any small mistake will affect the whole compile process. And edit the jni functions to meet our needs is also a hard way to realize it. As we can see in section IV, the use of CCNT can help us implement precise timing on Android devices, this can help us in the cache timing attack of the Android devices.The advantage of our method is that we can get the right way to get extremely precise timing on the

ARM platform, the accuracy can reach to 1 ns. But we can also see some abnormal time in the results, this is because of the interruption of other process.So in the future research, we should aim at overcome the interruption of other process and combine the precise timing with the AES encryption.

Acknowledgement

This research was financially supported by the National Natural Science Foundation of China (Grant No.61402528).

References

[1] Herath U, Alawatugoda J, Ragel R G. Software implementation level countermeasures against the Cache timing attack on advanced encryption standard[J]. arXiv preprint arXiv:1403.1322, 2014.

[2] JJTian, YZ Kou, CS, Chen et, al.Cache timing attack to RSA with sliding window algorithm[J] Computer Engineering, 2011, 37(17): 99-101,104.

[3] Intel. Intel 64 and IA-32 Architectures Software Developer's Manual, Volume 2 (2A,2B & 2C): Instruction Set Reference, A-Z, Dec. 2011.

[4] Information on http://developer.android.com/tools/sdk/ndk/index.html.

[5] Information on http://stackoverflow.com/questions/3247373/how-to-measure-program-execution-time-in-arm-cortex-a8-processor.