IOP Conf. Series: Materials Science and Engineering **322** (2018) 052008 doi:10.1088/1757-899X/322/5/052008

Key Technologies of Phone Storage Forensics Based on ARM Architecture

Jianghan Zhang and Shengbing Che*

School of Central South University of Forestry and Technology, Hunan Changsha,China 410004

*Corresponding author e-mail: cheshengbing727@163.com

Abstract. Smart phones are mainly running Android, IOS and Windows Phone three mobile platform operating systems. The android smart phone has the best market shares and its processor chips are almost ARM software architecture. The chips memory address mapping mechanism of ARM software architecture is different with x86 software architecture. To forensics to android mart phone, we need to understand three key technologies: memory data acquisition, the conversion mechanism from virtual address to the physical address, and find the system's key data. This article presents a viable solution which does not rely on the operating system API for a complete solution to these three issues.

1. Data acquisition

Android system is running the kernel of Linux, which means the Android operate system is running a Davik virtual machine on the Linux, and Android applications running on the Davik virtual machine. So, you cannot get a real, effective and reliable memory data in the topside, and you must get it from the driver level. Before the Linux kernel 2.6, memory data can be read from the device node "/ dev / mem". But in the subsequent version, the node can only read the limited data rather than all the running memory space data. And Android Linux kernel version has been more than 2.6 in popular. If you want to read the whole Android memory data, you must compile a drive to read the memory in the first.

As we told, Android is running on top of the Linux kernel, the driver is related to kernel. So, compile the Android system driver is the same as compile of Linux system driver.

By analyzing the "/ dev / mem" driver source code can be found that source code read memory data is through the physical memory mapped into virtual memory, then, use the mapped virtual memory pointer copy the data function "copy_to_user" to buffer space which in the user space.

For low physical memory, you can use the macro "__va ()" to achieve the mapping from physical address to the virtual address, the parameters of the macro passed a physical address value, the return is a virtual address pointer..

For high memory, you can't use the macro "__va ()" to achieve the mapping from physical address to the virtual address, you should map through the "kmap ()" function, and use "kunmap ()" to reclaim the mapped memory. The physical memory is managed in pagination, with a page size of 0x1000 bytes and a page number starting at 0. "Kmap ()" to pass the parameters of the page structure, page

structure can be created by "pfn_to_page ()"function and pass the parameters for the page number. Each times calling Kmap() can only mapped one page of virtual memory space.

If the physical address is on other devices not on the range of running memory, you need to use the "ioremap () function to map the physical address to run memory space. This function receives two parameters, one is the physical address value, the other one is the memory size which needs use iounmap () to recycle memory.

To determine whether the physical memory is high memory or low memory, you can call PageHighMem () to determine, and the parameters which be transmited is page structure.

To determine whether the physical memory is on the range of running memory, we can use the function “page_is_ram” to determine in a purely computer Linux, the parameters which be transmit is the page number. But, through a large number of experiments found that there is no function symbol (page_is_ram) in android system in the Linux kernel, so we cannot directly use the symbol when we compile the driver. Fortunately, there is a file named “/proc/kallsyms” in the Android system that records the address information for the kernel symbol. You can use the grep pipe to filter out the need to find the symbolic address, where you need to find the symbol named page_is_ram, as showed in Figure 1-1.

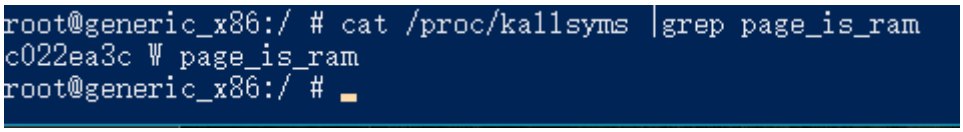

Figure 1-1. View the symbolic address of page_is_ram.

In summary, read a number bytes of data for a physical memory address, you should determine whether the physical address in the range of running memory at first, otherwise using ioremap and iounmap to map and reclaim memory. If this address is running in the range, then determine the physical address in the high memory range or low memory range, if the address is in the high memory range, through the kmap and kunmap to map and reclaim memory, otherwise use of __va mapping address.

2. Conversion mechanism for virtual address and physical address

ARM-based chips and x86-based chips in the virtual address to the physical address conversion is not the same way. The virtual address to physical address translation under the ARM architecture is included in first level and secondary index. If bit1 of the 32-bit physical address found by the first level index address is 1, that’s only include first level index. If bit1 is 0 and bit0 is 1, which shows it had secondary index. If both bit1 and bit0 are 0, it means that is an invalid address.

First level index has two modes, Supersection and Section. If the value of bit 18 is 1, it needs to be converted to the final physical address by the Supersecction mode. Otherwise use the Section mode.

Secondary index also has two modes, Large Page and Small page. If the 32-bit physical address found by the secondary index address is 1, it needs to be converted to the final physical address by the conversion method of Small page mode. If bit1 is 0 and bit0 is 1, Conversion method to convert to the final physical address needs large page mode. If both bit1 and bit0 are 0, it means the address is an invalid address.

2.1 First level index physical address calculation

The index number First level tableIndex(L1 TableIndex) in the page table is based on the virtual address for 12 bit values of bit20 to the bit31. Since the address length is 4 bytes, the L1 TableIndex multiplied by 4 and plus the address of page directory, we can get the first level index physical address.

2.2 The compute mode of the secondary index physical address

The index number Second level tableIndex(L2 TableIndex) in the secondary index table is based on the virtual address for 12 bit values of bit20 to the bit31. Since the address length is 4 bytes, L2 TableIndex multiplied by 4 and plus the address of secondary index, we can get the secondary index

physical address.

2.3 Supersection conversion method

Supersection belongs to the first level index mode, which can extend the 32-bit virtual address to the 40-bit physical address. The final converted physical address consists of four parts, bit0 to bit 23 are Supersection Index spliced by the virtual address of bit0 to bit23, bit 24 to bit 31 are SupersectionBA(BA is the abbreviation for Basses Address) spliced by the first level index address of bit24 to bit31, bit 32 to bit 35 are Extentd BA spliced by the first level index address of bit0 to bit23, Bit36 to bit 39 are Extentd BA spliced by the first level index address of bit5 to bit8.

2.4 Section conversion method

The section belongs to the first level index mode, the final converted physical address consists of two parts, bit0 to bit19 are SupersectionIndex spliced by the virtual address of bit0 to bit19, bit20 to bit31 are Section basse address spliced by the first level index address of bit20 to bit31.

2.5 Large Page conversion mode

Large Page belongs to the Secondary index mode. Get the secondary index physical address value, set the property of bit 0 to bit15 is 0, and plus the PageIndex in virtual address of the bit 0 to bit15, to get the final physical address.

2.6 Small Page conversion method

Small Page belongs to the Secondary index mode. Get the secondary index physical address value, set the property of bit 0 to bit11 is 0, and plus the PageIndex in virtual address of the bit 0 to bit15, to get the final physical address.

3. Analyzing the key data of the system

The kernel of Android is Linux. The key point is to analyze the descriptor task of all process if you want to forensics the memory of Android mobile phone. Because all the processes in memory are linked by a two-way circular list, you can find all the process descriptors by searching the list as long as you get a process descriptor(task_struct). In the kernel, the first process init_task is derived, so you can directly extract the address operation to obtain init_task virtual address in the driver, and then use __pa () convert virtual address to a physical address to get the physical address of init_task. Init_task is a variable of the tast_struct type, which has a tasks member that is a double-linked circular list node, and it can find the previous or next process descriptor.

It will inevitably involve the virtual address to the physical address in the process of analysis, __ pa () can only convert a limited system virtual address, the process of virtual address to physical address __pa () is impossible. It must through the conversion mechanism of the virtual address to the physical address to convert in the second section.

The page directory base (pgd / cr3) is a support in the virtual address to the physical address conversion mechanism of the second secction, without it cannot do any conversion work, the page address can get in the kernel variable init_mm pgd members, Is also a virtual address, because it is a kernel variable, so you can convert to a physical address through __pa ().

In this way, with the page directory base address, with the physical address of the first process descriptor, by reading the tasks of the first process descriptor, obtain to the next process of the process descriptor virtual address, and then converted into physical address, get the next process of the process descriptor physical address, and then read the next process descriptor, and so on, and ultimately all the process descriptor can be found.

Analysis the URL information opened by android.browser, you can start from files members of the task_struct, it is a files_struct type structure, there is a structure members fdt that is fdtable type, the max_fds members of fdtable represent the number of opening files, the fd members of fdtable is a pointer to the start address of the file pointer array. With max_fds and fd, you can analyze all the open file information of the process. Analysis web site information of the process, you can scan the process

of all mapped physical memory to achieve. Non-system processes generally have their own pgd, the value can be parsed from the mm_struct type structure member mm of the task_struct, what the mm_struct pgd members saved is the process of page directory base virtual address, through the virtual address to the physical address of the conversion method you can analysis any virtual address . mmap members of mm_struct record the process of memory allocation, by analyzing it can know the physical memory information that the process has been mapped out, scanning the physical memory that has been mapped out , then you can analyze the process of browsing the URL traces.

4. Results and conclusion

The subject in this experiment is Google's Android emulator, Android 5.0, ABI is armeabi-v7a, kernel version of Linux 3.4.67.

The page base address (Pgd / cr3) read from the drive is 0x4000, the first process init_task process descriptor physical address is 0x4A0C78, tasks offset is 0x1C8, process comm offset is 0x2E4.

By analogy, all the process descriptors can be found out, as shown in Figure 4-1.

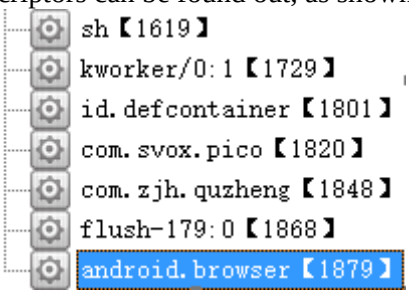

Figure 4-1 lists of process descriptors.

Here to analysis the browser process android.browser, the basic information as show in Figure 4-2. The process descriptor address is 0x2C9E8800, the base address of process page directory is 0x2CA54000, the process PID is 1879, and the parent process PID is 81.

```
Task name:android.browser
Task  PID:1879
Parent PID:81
Task   PGD:2CA54000
Task descriptor address:2C9E8800
```

Figure 4-2. Basic information for browser process android.browser.

Analysis the information of the process android.browser opened, analysis of the results and show in Figure 4-3.

```
File path:/data/com.android.browser/databases/browser2.db-wal
File system type:ext4
File struct address:2C05A480
Inode struct address:2C463E20
Dentry struct address:2C461818

File path:/data/com.android.browser/app_webview/Local Storage/https_m.baidu.com_0.localstorage
File system type:ext4
File struct address:2C09F0C0
Inode struct address:2C4652D0
```

Figure 4-3. The information of the process android.browser opened.

Analysis android.browser physical memory mapping and the results show in Figure 4-4, the total mapped physical memory size of process is 154.41MB.

```
android.browser

Memory interval node count:48
Total virtual memory:1.53GB
Total physical memory:154.41MB
```

Figure 4-4. The situation of android.browser physical memory mapping.

Using scanning to mapped physical memory, analyze the open URL traces, the analysis results show in Figure 4-5.

Figure 4-5. The URL information opened by android.browser.

All these experimental results could be enough to show that the proposed solution is effective and feasible.

5. Research prospects

The advantage of this paper is to overcome the mechanism of virtual address to physical address translation under ARM architecture and obtain the system memory data directly from the driver layer without relying on the API of the system application layer. Then, the address translation algorithm of this paper will run the system One by one analysis. System application layer API will take protective measures for the system sensitive data, and thus the data obtained is unstable, inaccurate, unreliable, for evidence, the wrong data will inevitably lead to the wrong direction, and this paper by taking bypass the application layer API directly access to memory data, to avoid the system do the possibility of hands and feet to date in the middle, then the data obtained is true, effective and reliable.

The shortcomings of this paper is the equipment to be evidence must be root, for different mobile phones, need the corresponding kernel source to support the driver of the compiler.

Driver compilation needs to correspond to kernel source to support, that is a disadvantage, and there are no solution to ideas and ways temporary. The root can analyze the system vulnerabilities, and use the loopholes was injected into Trojan horse to find way. So, the future research and development direction can start from the root.

The advantage of this paper is to overcome the mechanism of virtual address to physical address translation under ARM architecture and obtain the system memory data directly from the driver layer without relying on the API of the system application layer. Then, the address translation algorithm of this paper will run the system One by one analysis.

References

[1] Chen Qiang. The underlying interface and driver development technology explain for Android[M]. Beijing: China Railway Press,2015, pp.238-267.
[2] Junli. Android system source code analysis [M]. Beijing: China Railway Press,2015,pp.21-24.
[3] Jie Song, Lichen Dang, Zhenggang Guo, etc. Research on Security Mechanism Analysis and Application of AndroidOS Mobile Platform [J]. Computer Technology and Development, 2010(06):152-155.
[4] British ARM company. ARM_Architecture_Reference_Manual_ARMv7-AR.pdf[M]. British ARM company, 2011, pp.1324-1337.