The Storage Protection of Block Device in Android

Zhao-Wei Wang[1,2], Quan-Xin Zhang[1,2,a,*], Lei Long[1,2], Zi-Jing Cheng[3,b] and Yu-An Tan[1,2]

[1] *School of Computer Science and Technology, Beijing Institute of Technology, Beijing 100081, China*

[2] *Beijing Engineering Research Center of Massive Language Information Processing and Cloud Computing Application, Beijing 100081, China*

[3] *Space Star Technology CO., Ltd. China Academy of Space Technology, Beijing, China.*

[a]*zhangqx@bit.edu.cn*, [b]*951677563@qq.com*

[*]*Corresponding author*

Abstract

Data protection technology based on operation redirection has not been implemented in Android operating system. To achieve the goal of operation redirection, the research of Android storage subsystem needs to be done. By modifying the block device driver and rebuilding the kernel, all block device I/O requests are monitored and redirected. The experiment results demonstrate that the redirection-based storage protection is feasible in Android system. The potential improvement is discussed at the end of the paper.

Keywords: Android; Storage protection; Block device driver; Data redirection

Introduction

In recent years, Android has become the most popular operating system for mobile devices. However, Android doesn't provide users with protection for storage media. Users can directly modify or delete files and even format the flash media without restriction. Unintentional operations will result in permanent loss of data.

Redirection-based storage protection technology provides a virtual environment that user's operations will not be truly reflected in the storage device. All storage I/O operations are redirected to a special area, and this special area will be cleared after reboot. There're mature applications based on the redirection technology on PC platform, including hard disk recovery card and some sandbox programs [1]. These programs are all implemented for Windows operating system. Due to the different storage subsystem, the realization in Windows is not applicable to Linux, let alone the Linux-based Android operating system [2].

Android realizes sandbox mechanism to restrict the behavior of applications [3]. However, the sandbox mechanism used in Android is in application level and only restricts the operations requested by the corresponding application [4]. There’s no system-wide storage protection technology to restrict the high-privilege application’s operations in Android. To meet the demand of storage device protection in Android, the protection technology based on operation redirection is described in detail in this paper.

Android Storage Subsystem

Since Android is based on Linux kernel, the storage system hierarchy of Android and Linux are similar. It can be divided into two parts, file system and storage device system. These two parts are connected by the storage device driver.

File System Hierarchy. Android file system hierarchy is layered, including user space components and kernel space components [5]. The file system layer is above the device driver layer and under the system call interface layer. The file system layer consists of I/O cache, particular file system, virtual file system (VFS), index node (inode) cache and directory cache. User’s programs can request file operations by calling the GNU library functions or by directly calling the system interfaces. Through virtual file system, file operations will be dispatched to the real file system [6]. For the real file system, it interacts with the storage device under the help of interfaces provided by Linux kernel. All these layers in kernel space are transparent in user’s view.

Block Device. Smart devices, especially Android device often adopts NAND-based storage as main storage because of its advantages, high performance, small size, low heat, and silence compared to HDD [7]. NAND-based storage device that is often used in Android device is embedded Multi-Media Card (eMMC). In block device driver, the request queue is introduced to complete the I/O operations. The kernel will automatically insert the I/O request into the queue and remove the request after completion. Block device driver consists of three parts: block layer, core layer and host controller layer.

- Block layer is responsible for managing the request queue and translating I/O request into MMC request.
- Core layer encapsulates the operation related commands, such as reading, writing, cancellation, high-priority interrupt, and so on, then waits for the schedule
- Host controller layer is responsible for controlling eMMC device through the bus and finally finishes the MMC request.

Implementations

Shadow Region. Shadow region is composed of sectors that are occupied by a special file that is created when entering the redirection mode. Shadow region is used as a temporary area to store the redirected data. This file is created in eMMC card with its size set to 128MB. Currently in Android OS, the file system on most of the eMMC storage device is Ext4 file system. File system parameters can be obtained in file system super block. In the shadow file's inode structure, the sectors occupied by this file can be obtained. So the whole storage space is divided into two parts, normal storage region and shadow region. For these two regions, different redirection strategy will be applied, which will be discussed later in this paper.

Monitor and Modify I/O Request. In eMMC block device driver, a structure named mmc_request is used to store and forward eMMC card operation request. When the read/write operation requests from system block layer reach abstract device layer, the mmc_request structure will be initialized with read/write request parameters. After being decomposed by MMC protocol, the request is then forwarded to device to be finished. The structure mmc_request describes the information of read/write operation, including command, data and the callback function after the completion of request.

All eMMC operation requests are issued in function mmc_blk_issue_rq. The source code of this function is modified to monitor eMMC I/O requests. The structure variable mmc_queue_req contains the read or write command. Meanwhile, the structure variable mmc_blk_request contains the information of data size and sector position where the data will be written to or read from. In order to realize the operation redirection, original sector address value will be modified to the sector occupied by shadow region.

Sector Mapping Tree. When redirecting operation, no matter read or write, the mapping relationship between the original sector and the shadow region sector needs to be found out. Due to the large numbers of I/O operations during the runtime of operating system, the redirection mapping table is huge if it's organized in an array or a linear list and the time spent in searching is too long. To improve the efficiency of inserting, deleting and searching, a splay tree table is organized in kernel space.

The redirection table structure is briefly shown in Fig. 1. One megabyte is defined as a unit block, and the whole storage address space is divided into several blocks. Each node in splay tree represents the mapping relationship between one original sector and one shadow region sector. When searching a sector address in the table, table index is calculated first to locate the corresponding splay tree. Then, the program searches in the splay tree to obtain the mapping relationship. Finally, the sector node is adjusted closer to the root node in the tree so that the next access will be faster.

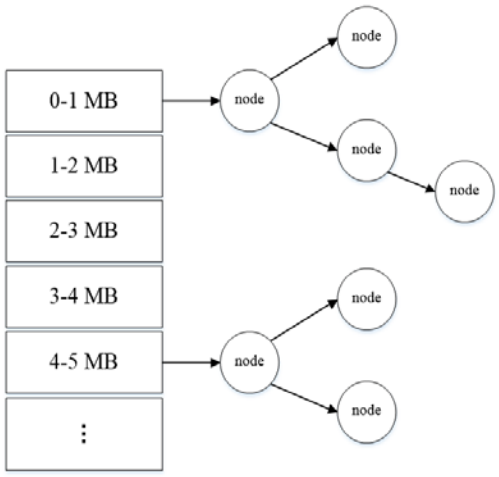


Figure 1. Splay tree for searching the mapped address

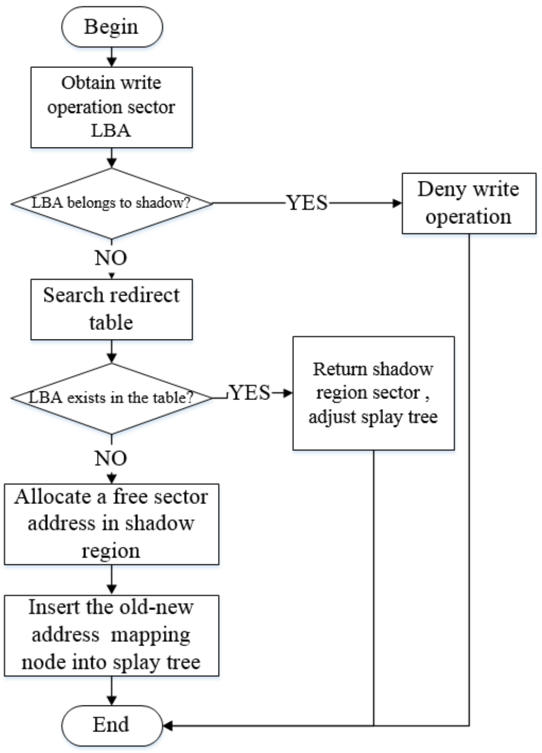

Begin
Obtain write operation sector LBA
LBA belongs to shadow?
YES
Deny write operation
NO
Search redirect table
LBA exists in the table?
YES
Return shadow region sector , adjust splay tree
NO
Allocate a free sector address in shadow region
Insert the old-new address mapping node into splay tree
End

Figure 2. Flow diagram of write operation redirection

Read/Write Operation Redirection.

Write Operation. When user space applications write some blocks of data to storage device, for every sector in these blocks, do the following steps. The steps are also described in Fig. 2.

1) Obtain write operation destination sector (LBA).
2) Check whether LBA belongs to shadow region. If LBA belongs to shadow region, deny write operation and return error code.
3) Search for LBA in the redirect table. If LBA exists in the table, return the mapped shadow region sector.
4) If LBA doesn’t exist in the redirect table, allocate a free sector in shadow region, and then insert the newly created mapping node into the splay tree.
5) If shadow region has no free storage space, return error code without redirection.

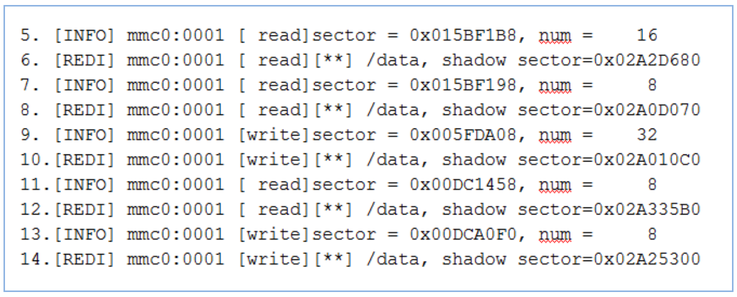
Read Operation. For read operation request, do the following steps.

1) Obtain read operation source sector (LBA).
2) Check whether LBA belongs to shadow region. If LBA belongs to shadow region, read data from shadow region sector.
3) If LBA doesn’t exist in the redirect table, read data from original sector.

Experiment Results

Because Samsung opens the Android kernel source code of their own devices to public, we experiment the redirection-based protection technology in a real mobile device instead of Android emulator. Google provides a toolset called Native Development Kit (NDK) for users to develop projects with the native code languages such as C and C++. The testing device is Samsung Galaxy Note 3, with Android OS version 4.4.2.

Operation Redirection. In order to observe the operation redirection behavior, using Android debug bridge tool, kernel messages can be demonstrated in command line. The kernel log is shown in Fig. 3. When user data partition is accessed, block device driver will trigger the redirection.
For example, as shown in Fig. 3, shadow region was allocated at sector 0x02A00000. In the seventh and eighth line of kernel log, read operation for sector 0x015BF198 was redirected to shadow region sector 0x02A0D070. In the ninth and tenth line, write operation for sector 0x005FDA08 was redirected to shadow region sector 0x02A010C0. All operations to data partition are redirected to shadow region, keeping original data unchanged.

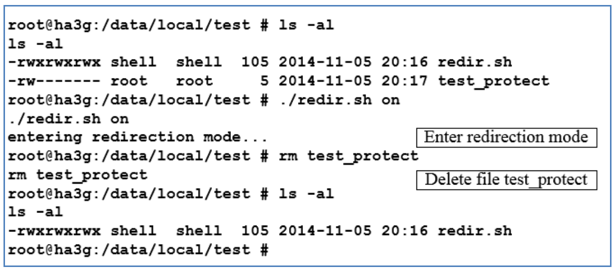


```
[INFO] mmc0:0001 [ read]sector = 0x015BF1B8, num =    16
[REDI] mmc0:0001 [ read][**] /data, shadow sector=0x02A2D680
[INFO] mmc0:0001 [ read]sector = 0x015BF198, num =     8
[REDI] mmc0:0001 [ read][**] /data, shadow sector=0x02A0D070
[INFO] mmc0:0001 [write]sector = 0x005FDA08, num =    32
[REDI] mmc0:0001 [write][**] /data, shadow sector=0x02A010C0
[INFO] mmc0:0001 [ read]sector = 0x00DC1458, num =     8
[REDI] mmc0:0001 [ read][**] /data, shadow sector=0x02A335B0
[INFO] mmc0:0001 [write]sector = 0x00DCA0F0, num =     8
[REDI] mmc0:0001 [write][**] /data, shadow sector=0x02A25300
```

Figure 3. Android kernel log of operation redirection

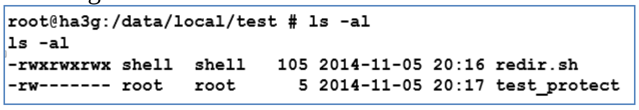
Data Protection. Data protection result is shown in Fig. 4 and Fig. 5. In Fig. 4, the device entered redirection mode at first. Then, we tried to delete the file “test_protect” with command “rm test_protect”. Finally, command “ls -al” showed that file “test_protect” has been deleted.

To check the original file data, we rebooted the device and listed the directory again. Fig. 5 depicts that the file “test_protect” showed up again. It proves that data protection is successful.



```
root@ha3g:/data/local/test # ls -al
ls -al
-rwxrwxrwx shell  shell  105 2014-11-05 20:16 redir.sh
-rw------- root   root     5 2014-11-05 20:17 test_protect
root@ha3g:/data/local/test # ./redir.sh on
./redir.sh on
entering redirection mode...
root@ha3g:/data/local/test # rm test_protect
rm test_protect
root@ha3g:/data/local/test # ls -al
ls -al
-rwxrwxrwx shell  shell  105 2014-11-05 20:16 redir.sh
root@ha3g:/data/local/test #
```

Figure 4. Enter redirection mode and delete file



```
root@ha3g:/data/local/test # ls -al
ls -al
-rwxrwxrwx shell  shell   105 2014-11-05 20:16 redir.sh
-rw------- root   root      5 2014-11-05 20:17 test_protect
```

Figure 5. Original file remains unchanged after reboot

Table 1. I/O performance in normal mode and redirection mode

	Normal mode	Redirection mode
Seq.Read [KB/s]	10753	9217
Seq.Write [KB/s]	10766	9006
Rand.Read [IOPS](4KB)	2373	2097
Rand.Write [IOPS](4KB)	3475	3051

Performance. Due to the operation redirection, storage I/O performance is affected. To compare the performance in redirection mode with the performance in normal mode, a benchmark application named Mobibench is introduced.

Mobibench is capable of measuring the I/O performance of Android file system [8]. Set partition to /data and keep other benchmark settings default. Sequential read/write and random read/write were measured respectively in normal mode and redirection mode. Testing results are listed in Table 1. The unit IOPS stands for I/O operations per second.
After comparing the I/O performance data in Table 1, redirection mode is proved to result in a slight performance degradation. Sequential read/write speed dropped by about 16% and random read/write speed dropped by about 12%. The performance degradation is mainly caused by searching and inserting mapping sectors in the splay tree. However, the performance lost is almost imperceptible with common operations on the device. The sacrifice of performance is acceptable for users to ensure the data security.

Conclusion

In this paper, we propose a redirection-based protection technology for the storage device in Android. By modifying the block device driver, eMMC I/O requests are monitored and data operations are redirected to a shadow region. The experiment results show that this protection technology is feasible in Android. I/O operations redirection is successful and data protection is realized.
For development of this technology, some further work needs to be done. First, the stability must be improved. Second, splay tree is not fast enough to search for the node when the load of I/O request becomes heavy. A more efficient data structure that combines hash algorithm and splay tree should be taken into consideration [9]. Last but not the least, the efficiency can be improved by discarding some redundant and useless I/O requests [10].

Acknowledgements

This work is supported by National Natural Science Foundation of China (No. 61370063), 863 Program (No. 2013AA01A212), Students' Innovative Plan of BIT (201410007046), China Aerospace Science and Technology Fund (CAST).

References

[1] Yee, Bennet, et al. "Native client: A sandbox for portable, untrusted x86 native code." Security and Privacy, 2009 30th IEEE Symposium on. IEEE, 2009.

[2] Singh, Jasmeet, Khalid Hussain, and Akshat Aggrawal. "An Application Sandbox Model based on Partial Virtualization of Hard-Disk and a Possible Windows Implementation." International Journal of Computer Applications 57 (2012).

[3] Blasing, Thomas. "Android Application Sandbox." 4. GI FG SIDAR Graduierten-Workshop über Reaktive Sicherheit. 2010.

[4] Blasing, Thomas, et al. "An android application sandbox system for suspicious software detection." Malicious and unwanted software (MALWARE), 2010 5th international conference on. IEEE, 2010.

[5] Bovet, Daniel P., and Marco Cesati. Understanding the Linux kernel. O'Reilly Media, Inc., 2005

[6] Yan Ding, Hongyi Fu, and Yuzhuo Li, "Research on VFS Layer Rootkit Technique in Linux." Computer Engineering 36.84 (2010): 161-164. (In Chinese).

[7] Kim, Hyukjoong, and Dongkun Shin. "Cross-layered view on android storage IO system." Computing and Convergence Technology (ICCCT), 2012 7th International Conference on. IEEE, 2012.

[8] Jeong, Sooman, et al. "AndroStep: Android Storage Performance Analysis Tool." Software Engineering (Workshops). 2013.

[9] Hui Zhang, Junwen Ji and Xiaosu Chen, "Hash-splay search algorithm in data-stream reassembling," Journal of Southeast University (Natural Science Edition) (2008): S1 (In Chinese).

[10] Lee, Kisung, and Youjip Won. "Smart layers and dumb result: IO characterization of an android-based smartphone." Proceedings of the tenth ACM international conference on Embedded software. ACM, 2012.