Poster: Your Access Control List Is Recoverable Even if Your OS Is Compromised

Caleb Rother, Bo Chen
Department of Computer Science, Michigan Technological University
{crother, bchen}@mtu.edu

***Abstract*—In the history of access control, nearly every system designed has relied on the operating system for enforcement of its protocols. If the operating system (and specifically root access) is compromised, there are few if any solutions that can get users back into their system efficiently. In this work, we have proposed a method by which file permissions can be efficiently rolled back after a catastrophic failure of permission enforcement. Our key idea is to leverage the out-of-place-update feature of flash memory in order to collaborate with the flash translation layer to efficiently return those permissions to a state pre-dating the failure.**

***Index Terms*—access control, ACL, flash translation layer, recovery**

I. Introduction

Access control lists (ACL) are commonly implemented by operating systems to specify permissions of users/groups to access system resources including files, directories, etc. To ensure the ACL can work correctly for controlling access of system resources, an implied assumption is that the OS should not be compromised. This assumption however, does not always hold. Major commodity operating systems use large monolithic kernels which are prone to attacks [1]. Once the OS is compromised, the adversary may arbitrarily change the file permissions by modifying the ACL. After the adversary is evicted, efficiently restoring file permissions is a crucial task in timely returning the system to its normal state. However, the malicious changes to the ACL could be wide-scale and efficiently restoring the entire ACL is a non-trivial problem.

Thus, we aim to tackle this problem. To ensure efficiency, we should not rely on remote backups and the restoration should purely happen in the local device. Our design relies on two insights: 1) We ensure restoration of the ACL entries by utilizing raw data in the underlying flash memory. To change file permissions, the adversary typically needs to overwrite them at the OS level. However, a flash storage device performs out-of-place updates internally and, therefore, an old version of the ACL entries may still be preserved on the raw NAND flash. Therefore, it is possible to roll back the ACL entries by extracting the corresponding raw data being preserved. 2) To enable an efficient rollback of the ACL entries, we should avoid expensive data copying. Instead, we transparently restore them by carefully reconstructing the corresponding mappings in the PMT table, which maintains all mappings between the block addresses (accessible to the file system) and the flash memory addresses. Note that data recovery from malware attacks using flash memory has been explored in prior works [3], [5], [6], which focus on restoring the entire storage. However, this work specifically focuses on restoring ACL entries which requires a fine-grained analysis over the raw flash memory data.

II. Background

Access control list (ACL). An ACL is a list of permissions associated with a system object. It has been implemented in a variety of file systems to control access of files and directories, including but not limited to NTFS (used in Windows), ext2/ext3/ext4 (used in Linux and other Unix-like OSes), APFS (used in macOS and iOS), etc. This work mainly focuses on the ACL used in the ext file systems, in which the ACL entries are split up on a directory-by-directory basis and stored in inodes which are created when the file system is initialized and stored in groups on the drive.

Flash memory and flash translation layer. NAND flash has been used broadly as the external storage in both standard computers and mobile devices today, as it consumes much less power and is capable of much faster read/write speeds. Flash memory is typically organized into blocks, each of which consists of pages. Each page contains a small out-of-bound (OOB) area. The flash memory works in a unique way, requiring an erase before writing, as well as erasing in terms of blocks.

To remain compatible with traditional block file systems such as ext4, a flash storage device (e.g., an SSD) usually exposes a block access interface. This is achieved by introducing a new flash translation layer (FTL), a piece of special firmware which stays between the file system and the NAND flash, transparently managing the special characteristics of NAND flash. The FTL implements the out-of-place update strategy [5], in which upon updating the data stored on a flash page, it places the new data to a new page, and simply invalidates the old page without sanitizing it immediately. Therefore, an old version of the data is temporarily preserved on its original page. This could enable rollbacks of this data for nearly zero storage overhead. *Our work therefore utilizes this phenomenon to restore the ACL entries compromised by the adversaries*. The FTL also maintains a page mapping table (PMT) keeping track of the physical location of the data on the flash memory. In other words, given the logical address on the block device, by searching the PMT table, we can identify the physical flash page where the corresponding data is stored.

III. Threat Model and Assumptions

We consider a computer using flash memory as external storage. The adversary is able to compromise the OS of the computer, obtaining root access. The adversary tries to disrupt the access control by means of modifying files' access permissions. We make a few assumptions: 1) The structure of the file system is either intact or has not been compromised for the entire duration that the computer has been on, as it would require the adversary with very high levels of knowledge about the operating system to compromise the file system's structure without completely destroying the computer. 2) The target operating system stays in the first partition of the disk, which is common for most users. 3) The FTL is secure, as it is isolated by the storage hardware from the OS.

IV. Preliminary Design

The design consists of three phases: setup, monitoring, and restoration.

Setup. Upon initialization, the FTL will 1) identify the number of inode groups stored on the filesystem, 2) find the logical page number for the first inode of each inode group and store the logical page number in an array in ascending order (further referred to as "Array 1"), and 3) store the size of each inode group, measured in pages. In the event of a shutdown of the drive, the aforementioned data can either be committed to disk and read on restart, or the setup process can be run again.

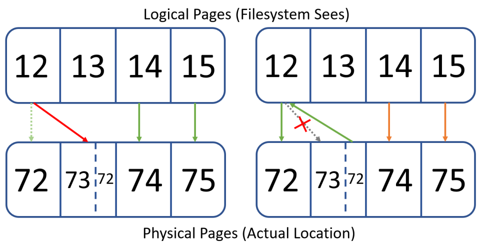
Monitoring. Following setup, any overwrite to a logical page in the FTL will cause the following procedure to be run:

1) Check if this operation is overwriting a logical page containing inodes. This is achieved via a binary search of Array 1, checking whether the page being written is within the size of an inode group from any entry.
2) If the logical page contains inodes, we will find out where the current physical copy of the page is stored by looking it up in the PMT. Append the resulting physical page number to the OOB area of the new page being written. Then, store this logical page number to an array (further referred to as "Array 2"). Also whitelist this physical page from being removed by garbage collection.

Restoration. When the user has identified that the ACL is compromised and wants to roll-back the latest changes to it, the user will send a restoration command to the FTL. Upon receiving the restoration command, the FTL will perform the following steps:

1) For each logical page number in Array 2, do:
   a) Read the current entry by looking it up the PMT, read the corresponding physical page, and check the OOB area.
   b) Update the PMT by changing the current page's entry in the PMT to point back to its old physical page number stored in the OOB from step (a).

This reverts the data stored in the inodes to their contents before last written, effectively undoing the undesired changes. Fig. 1 gives a visual demonstration. **Future work**. There are some remaining issues we plan to address in the future work including: 1) the timely detection of unauthorized changes to the ACL entries, and 2) the seamless integration of the detection and the recovery components for a full-fledged framework.



Logical Pages (Filesystem Sees)
12
13
14
15
12
13
14
15
72
73
72
74
75
72
73
72
74
75
Physical Pages (Actual Location)

Fig. 1. A visual demonstration of rolling back changes to a page. Note that we store ID 72 in page 73's OOB area for easy retrieval.

V. Preliminary Implementation and Evaluation

Our preliminary design is made on ext2 which uses a traditional ACL. The design can be easily ported to ext3 or 4 by modifying the setup. We have implemented our design by modifying the open-source FTL firmware OpenNFM [2] and porting it to an electronic development board LPC-H3131 (ARM9 32-bit, 32MB RAM) for testing. The LPC-H3131 was attached to a host computer via a USB 2.0 interface. The test was performed in Kali Linux running in a virtual machine equipped with 2 4.9 GHz CPU cores (coming from an Intel i9-10900KF chip) and 4 GB of RAM. To simulate the attack on compromising the ACL entries, we used the chmod command to set entries to new permissions values. To simulate the recovery of the ACL entries after the attack, we issued a command to the FTL via a write to a specific page in order to activate the restoration [4].

Our experiment showed that a rollback of the ACL entries is feasible. We also measured the time needed for rolling back an AVL entry which is around 11 milliseconds on average. We observed that the time needed for rolling back more ACL entries (e.g., 30) does not increase a lot, as the restoration of the ACL entries can be efficiently achieved by rolling back the corresponding mapping entries in the PMT table, without copying any actual data.

Acknowledgment. This work was supported by US National Science Foundation under grant number 2225424-CNS, 1928349-CNS, and 2043022-DGE.

References

[1] Critical RCE Vulnerability in Linux Kernel Let Hackers Compromise The Entire Systems Remotely. https://cybersecuritynews.com/linux-kernel-bug-3/.

[2] Opennfm. https://code.google.com/p/opennfm/.

[3] N. Chen and B. Chen. Defending against os-level malware in mobile devices via real-time malware detection and storage restoration. *Journal of Cybersecurity and Privacy*, 2(2):311–328, 2022.

[4] N. Chen, B. Chen, and W. Shi. A cross-layer plausibly deniable encryption system for mobile devices. In *Proc. of SecureComm '22*.

[5] L. Guan, S. Jia, B. Chen, F. Zhang, B. Luo, J. Lin, P. Liu, X. Xing, and L. Xia. Supporting transparent snapshot for bare-metal malware analysis on mobile devices. In *Prof. of ACSAC '17*.

[6] W. Xie, N. Chen, and B. Chen. Enabling accurate data recovery for mobile devices against malware attacks. In *Proc. of SecureComm '22*.