Poster: Data Recovery from Ransomware Attacks via File System Forensics and Flash Translation Layer Data Extraction

Niusen Chen
Department of Computer Science,
Michigan Technological University
Houghton, MI, USA
niusenc@mtu.edu

Josh Dafoe
Department of Computer Science,
Michigan Technological University
Houghton, MI, USA
jwdafoe@mtu.edu

Bo Chen
Department of Computer Science,
Michigan Technological University
Houghton, MI, USA
bchen@mtu.edu

ABSTRACT

Ransomware is increasingly prevalent in recent years. To defend against ransomware in computing devices using flash memory as external storage, existing designs extract the entire raw flash memory data to restore the external storage to a good state. However, they cannot allow a fine-grained recovery in terms of user files as raw flash memory data do not have the semantics of "files".

In this work, we design FFRecovery, a new ransomware defense strategy that can support fine-grained data recovery after the attacks. Our key idea is, to recover a file corrupted by the ransomware, we can 1) restore its file system metadata via file system forensics, and 2) extract its file data via raw data extraction from the flash translation layer, and 3) assemble the corresponding file system metadata and the file data. A simple prototype of FFRecovery has been developed and some preliminary results are provided.

CCS CONCEPTS

• **Security and privacy → Malware and its mitigation**; **Hardware security implementation**.

KEYWORDS

Ransomware; Data Recovery; File System Forensics; Flash Translation Layer; Fine-grained

ACM Reference Format:
Niusen Chen, Josh Dafoe, and Bo Chen. 2022. Poster: Data Recovery from Ransomware Attacks via File System Forensics and Flash Translation Layer Data Extraction. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security (CCS '22), November 7–11, 2022, Los Angeles, CA, USA.* ACM, New York, NY, USA, 3 pages. https://doi.org/10.1145/3548606.3563538

1 INTRODUCTION

Computing devices including servers, personal computers, mobile devices increasingly store critical data nowadays and, a large portion of them use flash memory [3] as external storage. Ransomware is turning to a significant threat to today's computing devices. Two common types of ransomware are locker ransomware and crypto-ransomware. Locker ransomware locks the device to extort money. This type of ransomware can be easily mitigated by unplugging the storage medium from the victim device and plugging it to a healthy device. Crypto-ransomware however, is difficult to combat, as it encrypts files in the victim device using strong cryptography and, the decryption keys can only be obtained after the ransom is paid.

The existing ransomware defenses [4, 5, 7–12] for computing devices using flash storage purely rely on raw flash data for recovery, and suffer from a significant limitation that, they only allow restoring the entire raw flash memory data rather than the individual user files. However, ransomware typically corrupts user files stored in a victim device and the recovery after ransomware attacks should be file-driven. For example, the user should be able to determine what files need to be restored or which files should be restored first. Those existing approaches [4, 5, 7–12] unfortunately are not file-driven, as the raw flash memory data are located at the lower storage medium layer and do not have the semantics of "files".

To address the aforementioned limitation, we rely on both the file system forensics and the raw flash memory data extraction. Our key insights are: 1) Existing ransomware typically runs in the user space, and does not have the privilege to modify the file system metadata. Therefore, after ransomware attacks, it is possible to *restore the file system metadata of a corrupted file* via the file system forensics. 2) The ransomware either overwrites a victim file with its encrypted version or directly deletes[1] it after storing its encrypted version. However, due to the out-of-place updates performed in the underlying flash translation layer (FTL), the file data overwritten/deleted by the ransomware in the user space will be temporarily preserved in the flash memory [7, 8]. Therefore, it is possible to *extract the original file data of a corrupted file* from the FTL. By assembling both the file system metadata and the file data, we may restore a victim file attacked by the ransomware. The resulted design, FFRecovery, is a new ransomware defense strategy specific for computing devices using **F**lash storage, supporting a **F**ine-grained **Recovery** of victim user files.

2 BACKGROUND

Ransomware. A major threat nowadays comes from the crypto-ransomware which encrypts victim data and asks for a ransom. Typically, after hacking into a victim computing device, the ransomware will read a victim file, encrypt it, and 1) replace the victim file with the encrypted file, or 2) write the encrypted file to a new location and delete the victim file.

File system. File system is a method/data structure by which the operating system can control how the user data can be stored into/retrieved from the storage media. User data are typically viewed

[1]To really delete a file in the user space, the application always needs to overwrite it with randomness or garbled information.

CCS '22, November 7–11, 2022, Los Angeles, CA, USA

ACM ISBN 978-1-4503-9450-5/22/11.
https://doi.org/10.1145/3548606.3563538

as "files" and, every read/write operation on files will be translated transparently by the file system into the read/write operation on the external storage. To facilitate this translation, a file system usually divides the storage into blocks, and stores the data in the blocks. The file system will keep track of various bookkeeping information (or metadata) for a file, e.g., file name, locations of corresponding blocks in the disk, file creation time, etc. Based on whether a file system implements journaling or not, we can simply classify it into a journaling file system and a non-journaling file system.

The journaling file system usually keeps track of changes which are not yet committed to the main file system by recording them in a circular log called a "journal". The journal would be very useful for quickly restoring the system upon an unexpected power loss or system crash. Oftentimes, the journal file systems may only keep track of the stored metadata. They may also keep track of both the stored data and the metadata depending on the implementations. Popular journaling file systems include NTFS, EXT3, and EXT4. In EXT4, the journal typically can be referenced with inode number 8 [1]. Different from the journaling file system, the non-journaling file system does not have a journal. Popular non-journaling file systems include exFAT, FAT series (FAT12, FAT16, FAT32) and EXT2. A FAT file system consists of three main regions: boot region, file allocation table (FAT) region, and data region. The boot region contains basic file system information in the boot sector. The FAT table is stored after the boot region and indicates the clusters allocated to files. The data region contains all file data and metadata. The metadata for files/directories are stored in a tree-like structure contained in the data region, starting at the root directory. The nodes in this tree are known as directory entries, and contain metadata for the parent directory's files/directories. The location of the root directory is implied in the boot sector.

Flash memory. Flash memory especially NAND flash is turning to a mainstream storage medium today. NAND flash consists of blocks (typical block size is 16KB, 128KB, 256KB, or 512KB), and each block consists of pages (typical page size is 512B, 2KB or 4KB). Compared to traditional mechanical disks, NAND flash exhibits some unique characteristics. First, its unit of read/program operation is a page, but its unit of erase operation is a block. Second, it follows an erase-before-write design, which means, re-programming a page is not allowed before an erase operation is performed on the encompassing block. Third, each block only allows a limited number of program/erase (P/E) cycles, i.e., the block turns unreliable when its P/E cycles exceed the threshold and cannot be used to correctly store data. Due to the aforementioned special characteristics, an in-place update strategy is expensive as it requires first erasing the entire encompassing block. Therefore, NAND flash prefers an out-of-place update strategy in which an update is performed by writing the new data to a new location and meanwhile invalidating the old data. To allow the traditional block-based file systems (e.g., FAT32, EXT4, NTFS) to be used on flash storage, an extra flash translation layer (FTL) has been introduced between the file system and raw NAND flash. The FTL transparently manages the special nature of flash memory, exposing a block access interface externally. One key function implemented by the FTL is address translation which translates the block addresses of the file system to the physical flash memory addresses, by maintaining a mapping table between them. Another key function is garbage collection which reclaims invalid blocks, typically during the idle time.

3 SYSTEM AND ADVERSARIAL MODEL

System model. We consider a computing device which is equipped with a flash-based block device as external storage. This type of computing device is very common in real world. This can be a server or a personal computer which is equipped with an SSD drive. This can be also a mobile or an IoT device which is equipped with an SD/miniSD/microSD card, an eMMC card or a UFS card. A block-based file system (e.g., EXT4, FAT32, NTFS) is deployed to manage the external storage.

Adversarial model. We consider crypto-ransomware which encrypts user data and asks for ransom money. The ransomware can successfully run in the user space of a victim computing device. In addition, it can obtain the user-level privilege, and hence can freely open, read, write and delete user files. However, it cannot obtain the kernel-level privilege, and hence cannot manipulate the kernel as well as the system data belonging to the kernel. For example, it cannot modify or delete the file system metadata.

4 OUR DESIGN

After a ransomware attack, FFRecovery aims to allow a victim user to restore a file corrupted by the ransomware. In other words, the user can freely select what files to be recovered and use FFRecovery to restore them. To restore a file, we need to first restore its corresponding file system metadata, figuring out all the disk locations which store its file data. This can be achieved by file system forensics, considering that the ransomware does not have the privilege to corrupt the file system metadata. Next, we need to restore the original data stored in the aforementioned disk locations. This can be achieved by taking advantage of the out-of-place updates performed in the underlying flash translation layer, extracting the file data from the raw NAND flash.

Recovering the file system metadata by performing forensic analysis over the file system. To compromise a file, the ransomware will read it into the memory, encrypt it, and 1) overwrite the original file with the encrypted file, or 2) write the encrypted file to the disk and delete the original file. In both cases, the file system metadata of a given file may be restored by performing forensic analysis over the file system. For journaling file systems (e.g., EXT3 or EXT4), the original file system metadata can be obtained by analyzing the journals. For non-journaling file systems (e.g., FAT, exFAT), the original file system metadata can be obtained by analyzing the boot sector and the directory entries.

Recovering the file data by data extraction in the FTL. To reconstruct a file corrupted by the ransomware, merely recovering its file system metadata is insufficient, as the metadata only contains file information like the original locations of the file data at the block layer (e.g., original disk sectors holding the file data). However, as the ransomware has corrupted/deleted the file, the original block locations have stored something else. Fortunately, the FTL performs out-of-place updates. As shown in the example of Figure 1, the file data were originally stored at block 2 in view of the file system and the actual data were stored in flash memory location 1; after the file data are corrupted/deleted by the ransomware, the new data will be stored in flash memory location 4, but the old data are still

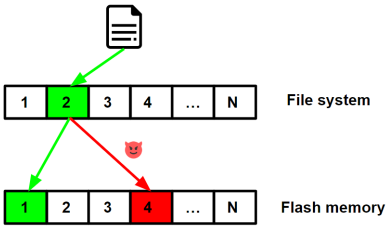

Figure 1: Recovering file content by extracting raw flash data.

preserved in flash memory location 1. To extract the original file data stored in flash memory location 1, there are two alternatives: 1) The OS can send the block location (here is 2) to the FTL and, the FTL will search the old mappings and find out that block 2 was mapped to location 1. The FTL will then extract the data from location 1 and send them back to the OS. 2) The FTL will restore the old mapping so that block 2 is mapped back to location 1. Then, the OS can simply read block 2 and obtain the original file data.

Operating FFRecovery **in practice**. After a victim computing device is attacked by the ransomware, the user will be noticed immediately as the ransomware usually displays an on-screen alert. As the user does not trust the victim device any more and, he/she will restore the files from the victim device and store them to another disk. The user should perform[2] the following operations as a root user: The user mounts a new disk, runs FFRecovery to recover target files, and writes the recovered files to the new disk.

Limitations. FFRecovery can work under the condition that the old data/mappings are preserved in the flash memory. As the old data and the old mappings have been invalidated by the ransomware, garbage collection in the FTL may reclaim them eventually. Therefore, FFRecovery needs to restore the files timely before the garbage collection reclaims the corresponding locations. This is doable as the garbage collection typically happens during idle time and, it only reclaims the flash blocks with a large number of invalid pages. A more reliable option would be periodically backing up the mapping table (typically small in size) in the FTL and temporarily disabled the garbage collection if the ransomware is present [5].

5 IMPLEMENTATION AND EVALUATION

Implementation. We used a Ubuntu virtual machine as a host computing device (Ubuntu 18.04, 4G memory, 40G SSD). We built a flash storage medium using LPC-H3131 [2] (with ARM9 32-bit ARM926EJ-S 180Mhz, 32MB SDRAM, and 512MB NAND flash) and porting to it an open-sourced flash controller OpenNFM [6]. The flash storage medium was attached to the hosting device via USB 2.0. FAT16 was used as the file system of the host computing device. OpenNFM was modified so that it can work with the OS to extract the raw flash memory data. A tool was also developed in python3 to perform file system forensics to restore the file system metadata and to restore a given file by working with the modified OpenNFM.

Evaluation. We first placed a file to the flash storage medium. To simulate the ransomware behaviors, we then manually overwrote the content of the file. Next, we used FFRecovery to restore the file (restoring file system metadata via file system forensics and extracting the corresponding file content from the FTL) and wrote the restored file to the local disk. We repeated the aforementioned process under different file sizes, namely, 1MB, 5MB, 10MB, and 50MB, and measured the time needed for different components. The experimental results are shown in Table 1. We can observe that: 1) File system forensics is fast and is not affected by the file size. This is because we only need to analyze the file system metadata of a specific file for recovery which is small and is not significantly affected by the file size. 2) The time for reading the file content from the FTL and writing the restored file to the local disk both grow linearly (approximately) with the file size. This is reasonable as the I/O overhead is determined by its size. In addition, reading the file content from the FTL is more expensive (dominating the time needed for recovery), because we read the file content from the LPC-H3131 via a slow USB 2.0 interface, but write the recovered file to the local disk via a fast SSD interface.

[2]Blocking the ransomware before the recovery is highly recommended to prevent it from disturbing the recovery process.

file size	1MB	5MB	10MB	50MB
File system forensics (s)	0.00046	0.00042	0.00042	0.00041
Read from the FTL (s)	0.3147	1.6141	3.1817	16.6811
Write to the local disk (s)	0.0025	0.0054	0.0081	0.0392

Table 1: The time needed for recovering a given file.

6 CONCLUSION

In this work, we have designed FFRecovery, a ransomware defense strategy which can support fine-grained data recovery. FFRecovery relies on file system forensic as well as flash translation layer to extract raw data. A simple prototype of FFRecovery has been developed and some preliminary results are provided.

ACKNOWLEDGMENTS

This work was supported by US National Science Foundation under grant number 1938130-CNS, 1928349-CNS, 2043022-DGE, and 2225424-CNS.

REFERENCES

[1] ext4 data structures and algorithms. https://docs.kernel.org/filesystems/ext4/globals.html.
[2] Lpc-h3131. https://www.olimex.com/Products/ARM/NXP/LPC-H3131/.
[3] Ssd market share. https://www.t4.ai/industry/ssd-market-share.
[4] SungHa Baek, Youngdon Jung, Aziz Mohaisen, Sungjin Lee, and DaeHun Nyang. Ssd-insider: Internal defense of solid-state drive against ransomware with perfect data recovery. In *2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS)*, pages 875–884. IEEE, 2018.
[5] Niusen Chen and Bo 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.
[6] Google Code. Opennfm. https://code.google.com/p/opennfm/.
[7] Le Guan, Shijie Jia, Bo Chen, Fengwei Zhang, Bo Luo, Jingqiang Lin, Peng Liu, Xinyu Xing, and Luning Xia. Supporting transparent snapshot for bare-metal malware analysis on mobile devices. In *Proceedings of the 33rd Annual Computer Security Applications Conference*, pages 339–349, 2017.
[8] Jian Huang, Jun Xu, Xinyu Xing, Peng Liu, and Moinuddin K Qureshi. Flashguard: Leveraging intrinsic flash properties to defend against encryption ransomware. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 2231–2244. ACM, 2017.
[9] Donghyun Min, Donggyu Park, Jinwoo Ahn, Ryan Walker, Junghee Lee, Sungyong Park, and Youngjae Kim. Amoeba: an autonomous backup and recovery ssd for ransomware attack defense. *IEEE Computer Architecture Letters*, 17(2):245–248, 2018.
[10] Peiying Wang, Shijie Jia, Bo Chen, Luning Xia, and Peng Liu. Mimosaftl: adding secure and practical ransomware defense strategy to flash translation layer. In *Proceedings of the Ninth ACM Conference on Data and Application Security and Privacy*, pages 327–338, 2019.
[11] Xiaohao Wang, Yifan Yuan, You Zhou, Chance C Coats, and Jian Huang. Project almanac: A time-traveling solid-state drive. In *Proceedings of the Fourteenth EuroSys Conference 2019*, pages 1–16, 2019.
[12] Wen Xie, Niusen Chen, and Bo Chen. Enabling accurate data recovery for mobile devices against malware attacks. In *18th EAI International Conference on Security and Privacy in Communication Networks*, 2022.