MimosaFTL: Adding Secure and Practical Ransomware Defense Strategy to Flash Translation Layer[*]

Peiying Wang[†‡§] wangpeiying@iie.ac.cn

Shijie Jia[†‡] jiashijie@is.ac.cn

Bo Chen[¶] bchen@mtu.edu

Luning Xia[†§] halk@is.ac.cn

Peng Liu[||] pliu@ist.psu.edu

Abstract

Ransomware attacks have become prevalent nowadays due to sudden flourish of cryptocurrencies. Most existing defense strategies for ransomware, however, are vulnerable to privileged ransomware who can compromise the operating system and hence any backup data stored locally. The out-of-place-update and the isolation nature of flash memory storage, for the first time, makes it possible to design a defense strategy which is secure against the privileged ransomware.

In this work, we propose MimosaFTL, a secure and practical ransomware defense strategy for mobile computing devices equipped with flash memory as external storage. MimosaFTL is secure against the privileged malware by taking advantage of unique characteristics of flash storage. In addition, it is more practical (compared to prior work) for real-world deployments by: 1) incorporating a fine-grained detection scheme which can detect presence of ransomware accurately; and 2) allowing the victim to efficiently restore the infected external storage to the exact point when the malware starts to perform corruption. Experimental evaluation shows that, MimosaFTL can mitigate ransomware attacks effectively with a small negative impact on both I/O performance and lifetime of flash storage.

1 Introduction

In recent years, a special type of malware named ransomware has become very popular among cybercriminals. According to a report by Symantec [39], the number of ransomware attacks increased over 36% in 2017, and more than 4,000 ransomware attacks occur daily [42]. The latest notable ransomware instance, WannaCry [26, 40], has spread across 150 countries and infected more than 250,000 machines in a short period.

[*]A technical report of Computer Science Department, Michigan Technological University. A preliminary version of the paper appears in ACM CODASPY '19.

[†]Institute of Information Engineering, Chinese Academy of Sciences, Beijing, China

[‡]Data Assurance and Communication Security Research Center, Chinese Academy of Sciences, Beijing, China

[§]School of Cyber Security, University of Chinese Academy of Sciences, Beijing, China

[¶]Department of Computer Science, Michigan Technological University, Houghton, USA

[||]College of Information Sciences and Technology, The Pennsylvania State University, University Park, USA

Different from regular malware, ransomware extorts ransom money from victims by either locking the victim systems (i.e., *locker ransomware*) or encrypting the data (i.e., *crypto-ransomware*). Locker ransomware can be easily defended since data are still there and we can simply unplug storage medium from the infected system and plug it to a clean system to copy out the data. On the contrary, crypto-ransomware is more difficult to be defended since the data have been encrypted by strong encryption with secret key only known to the ransomware attacker. Therefore, the paper focuses on defending against crypto-ransomware.

In the literature, many approaches have been proposed to defend crypto-ransomware. They can be roughly categorized into two families: 1) ransomware detection; 2) data recovery from ransomware attacks. The idea of ransomware detection is to detect ransomware and block it as fast as possible before it can cause significant damage to the valuable user data. The detection usually relies on monitoring file system activities [27, 11, 20, 21, 34] or analyzing cryptographic operations [21, 11, 23]. A pure detection-based solution is unfortunately not sufficient due to the following reasons: First, regardless how fast the detection can be, the ransomware still runs before being blocked and encrypts some data. Second, *if the ransomware can compromise the operating system (OS) and obtain root privilege (i.e., privileged ransomware), it can simply disable the detection capability.* The other category of ransomware defense relies on backing up valuable data, and restoring them after ransomware attacks. The data can be backed up in either local storage [11, 38] or third-party cloud [45]. However, those approaches are all problematic. Backing up data in the third-party cloud results in extra storage and communication cost which may not be acceptable by users [38]. Backing up data locally can eliminate the need of third-party cloud. However, *the backups are vulnerable to privileged ransomware which can compromise the OS and obtain root privilege.*

Mobile devices like smart phones and tablets have been used extensively nowadays. According to statista [37], there are approximately 4.92 billion mobile devices in 2018. Unlike desktop computers, mobile computing devices usually use flash memory as external storage media (e.g., eMMC cards, MicroSD cards and SSD drives). Compared to traditional mechanical drivers broadly used in desktop computers, flash memory has significantly different characteristics: First, flash memory cannot be over-written before an erase operation is performed, which can only be performed on the basis of a large block (usually a few hundred KBs). However, the write operation is usually performed on the basis of a small page (usually a few KBs). Therefore, directly over-writing a page requires first erasing the entire encompassing block which further requires backing up valid data stored in other pages of this block, causing significant write amplification. Second, flash memory is vulnerable to wear. In other words, frequently writing/erasing the same flash block will eventually deteriorate the integrity of the storage. To accommodate the special nature of flash memory, an *out-of-place-update* strategy is usually used in flash storage, in which the newly updated data will be written to a new empty page, rather than the page being occupied by the old data.

Flash memory's special nature makes it possible to design a defense strategy secure against privileged ransomware. Mobile computing devices, which are equipped with flash media as external storage, suffer significantly from ransomware attacks recently [4]. Due to the out-of-place-update feature of flash media, the user data being over-written by ransomware[1] will be temporarily preserved in flash memory, which can be utilized later for data recovery. Additionally, the flash memory is usually used in the form of an isolated device being attached to a host system, with independent processor, memory, and software component (i.e., flash translation layer). This

[1]Deletion of user data can be viewed as a special over-write operation, being achieved by over-writing the target data with garbage information.

prevents the ransomware from having direct access to the raw flash using the limited read/write interface being offered, *even if the ransomware can compromise the entire host OS*. In other words, the ransomware will not be able to corrupt those old data being preserved in flash memory. Taking advantage of the aforementioned properties, it is possible to design more secure strategies specifically for mobile devices to enable data recovery from ransomware attacks.

Limitations of existing defense strategies utilizing flash properties. FlashGuard [15] and SSD-Insider [5] were designed to allow data recovery from ransomware by utilizing special nature of flash memory. They, however, both suffer from significant limitations.

The basic idea of FlashGuard is: Having observed that any data being encrypted by ransomware need to be read first and then deleted, it modifies garbage collection strategy in flash translation layer (FTL) such that the invalid data having been read will not be reclaimed. In this way, it ensures that, for those data encrypted by ransomware, their plaintexts are always preserved in flash memory and used for data recovery. Their design, however, is not practical: First, FlashGuard does not employ any detection algorithm or IDS, and hence does not have any knowledge on when ransomware comes. Therefore, it needs to preserve all the historical versions of the "possibly" attacked data for a long period (e.g., preserve the data which have been read for 20 days [15]) to maximize probability of successful recovery. This unfortunately may be overkill and impractical for mobile devices which usually have limited storage space. Second, FlashGuard does not provide a concrete recovery component which can take care of data recovery transparently to end users. Therefore, the users need to manually recover data encrypted by ransomware. Specifically, the user needs to unplug the flash storage device, plug it into another clean and isolated computing device [15], manually identify the corrupted data and metadata, and recover them using data preserved by FlashGuard. This is very impractical considering that the users usually do not have necessary computer skills.

SSD-Insider [5] tries to improve FlashGuard by introducing a ransomware detection and a data recovery component. However, it also suffers from a few significant issues. First, their ransomware detection mechanism is not effective. This is because, they detect ransomware by collecting "run-length" of overwritten blocks within a small fixed time window (e.g., 10 seconds). This relies on their observation that ransomware "conducts overwriting immediately after reading and encrypting the victim's files" [5]. This observation, however, is not necessarily true according to our independent study (Sec. 3). Specifically, we observed a special type of ransomware (Sec. 3), which will not overwrite the original LBAs with random data until attacking a large number of files. In other words, the overwriting pattern of this type of ransomware is difficult to be observed during this small fixed time window. In addition, CPU/IO-intensive applications may slow down activities of ransomware, making it difficult to observe the overwrite pattern of ransomware in this small fixed time window as well. Second, their recovery component is coarsely designed and can only recover data before 10 seconds. In other words, they strongly rely on an implied assumption that the ransomware starts to corrupt data and will be caught within 10 seconds. This assumption usually cannot hold in practice, since real-world ransomware does not necessarily perform attacks within 10 seconds (Sec. 3). In addition, the ransomware may actively play against the victim, after they know the design of SSD-Insider. A key challenge they cannot resolve is to allow the victim to locate and restore the exact point when the ransomware starts to corrupt external storage. Our design can successfully tackle this challenge. Third, they implemented their solution into an open-channel SSD platform, which actually runs on the block device layer [6] and is accessible to the ransomware. Such an implementation unfortunately contradicts the overall design rationale, since security of ransomware defense strategies for flash storage [15] strongly relies on the close nature of flash memory (which prevents the ransomware from corrupting data preserved in flash memory).

In this work, we aim to eliminate the aforementioned limitations and design a secure yet more practical (compared to FlashGuard and SSD-Insider) ransomware defense strategy specifically for mobile devices equipped with flash storage. The resulting design, MimosaFTL, is secure against privileged ransomware by taking advantage of the special nature of flash storage. In addition, it incorporates a detection component which monitors access behaviors caused by the ransomware in the FTL and, once the ransomware is detected, an efficient data recovery process will be invoked to restore the infected external storage to a good previous state. Our design relies on two key insights:

First, we introduce a fine-grained detection scheme which can detect presence of ransomware accurately with low overhead. It is advantageous to have a detection component, because: if no malware is detected, we can remove all the preserved old data to release storage space. Unlike SSD-Insider which simply relies on counting the number of overwrites for detection, we rely on more fine-grained characteristics, including I/O access type, I/O location as well as I/O length, which could result in a more accurate detection. In addition, our detection is less expensive than SSD-Insider, since we introduce a one-time preprocessing step which uses K-means clustering to distill a few access patterns in the beginning, and during detection, the observed behavior can be compared to the known patterns efficiently. Second, we allow efficiently restoring the infected external storage to the exact point when the ransomware starts to perform corruption. Thanks to the out-of-place-update and the isolation nature of flash storage, by simply manipulating garbage collection, old data can be preserved and the only concern remaining is the metadata. To efficiently restore the metadata to the exact point before ransomware corrupts user data (we call this point "corruption point"), MimosaFTL does the following (Sec. 4.4 and 4.5): 1) It backs up the latest metadata when the detection does not detect ransomware (we call this point "latest good point"); 2) Each change of the metadata from the latest good point is kept in the flash memory OOB area; 3) The metadata in the corruption point is efficiently reconstructed by applying a binary search between the latest good point and the detection point (i.e., when the ransomware is detected), together with a small number of user involvements.

Contributions. Our contributions are summarized as follows:

- We have collected more than five hundred real-world ransomware samples, and analyze their access behaviors in the FTL. By applying K-mean clustering, we have successfully identified a few unique access patterns of ransomware on the flash memory.

- We design a fine-grained detection scheme in the FTL, which can effectively detect presence of ransomware by monitoring access behaviors on the flash memory caused by ransomware. In addition, we design a scheme which can efficiently recover external storage to the exact point when ransomware starts to corrupt data.

- We implement a prototype of MimosaFTL using real-world flash firmware, which was ported to an electronic development board. Experimental evaluation shows that MimosaFTL can effectively mitigate ransomware attacks with a small negative impact on both I/O performance and lifetime of flash devices.

2 Background

2.1 Ransomware

Ransomware is a special form of malware that restricts access to a victim computer in order to extort the victim for financial gain [24]. Traditional malware typically aims to collect sensitive information stealthily without raising suspicion. On the contrary, ransomware will notify the victim, after having infected his/her valuable files.

Ransomware can be classified into locker ransomware and crypto-ransomware based on whether cryptographic algorithms are used to restrict data from victims. Locker ransomware is designed to restrict interaction with the system by weak techniques, such as simply locking the screen [44] or modifying the master boot record (MBR) and/or partition table [2], which can be easily restored without paying the ransom. Crypto-ransomware uses cryptographic algorithms to encrypt the victim's valuable files (e.g., documents and images) silently. Once the encryption is completed, the victim will be asked for a ransom to obtain the decryption keys needed for recovering plaintext files.

Crypto-ransomware can be divided into three categories based on types of encryption schemes being used [29]: symmetric key crypto-ransomware, asymmetric key crypto-ransomware, and hybrid key crypto-ransomware. Symmetric key crypto-ransomware simply uses symmetric encryption to encrypt files. The symmetric key may be reverse engineered or even brute forced [3]. Asymmetric key crypto-ransomware uses asymmetric encryption for file encryption. A drawback is that, encrypting large files using asymmetric encryption is usually time consuming. Hybrid key crypto-ransomware mitigates the drawback of asymmetric key crypto-ransomware, by using symmetric encryption to encrypt files, and using asymmetric encryption to encrypt the corresponding encryption key. Hybrid key crypto-ransomware is the mainstream of ransomware [23].

2.2 NAND Flash Memory

NAND Flash stores information in an array of memory cells, which are grouped into blocks of a few hundred Kilobytes. Each block is further divided into a certain number (e.g., 32, 64, or 128) of pages. Typical page size is 512 bytes, 2KB, and 4KB [8]. A page usually contains a small spare out-of-band (OOB) area which is used for storing various metadata (e.g., error correction code) [22].

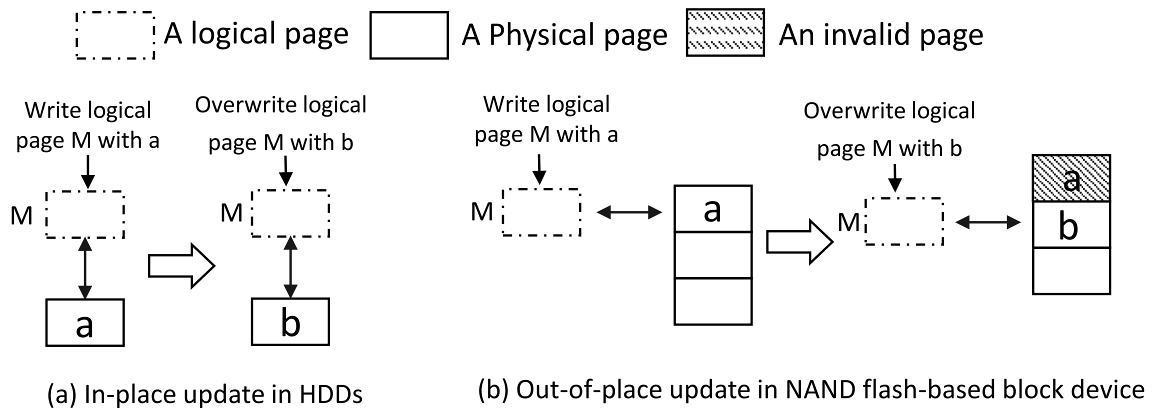

A logical page
A Physical page
An invalid page
Write logical page M with a
Overwrite logical page M with b
M
a
b
(a) In-place update in HDDs
Write logical page M with a
Overwrite logical page M with b
M
a
b
(b) Out-of-place update in NAND flash-based block device

Figure 1: Overwrite operation in HDDs and NAND flash-based block devices.

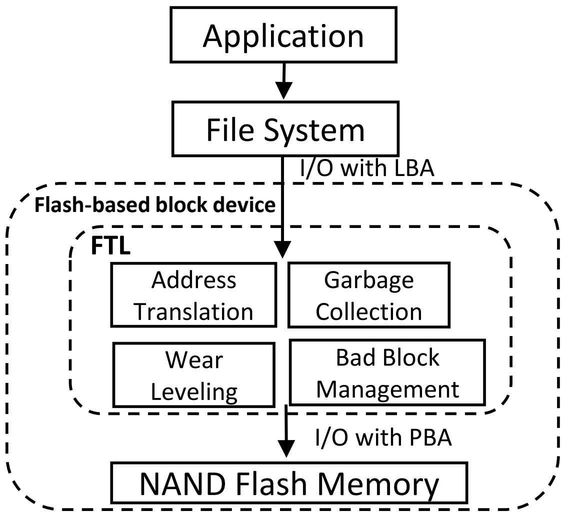


Figure 2: The architecture of a flash-based block device.

Different from traditional mechanical hard drives, NAND flash has a few unique characteristics, as described in the following. First, NAND flash has an erase-before-write design, i.e., overwriting a flash cell is not feasible before an erasure is performed over it. Second, the unit for reading/programming flash is usually a page, but the unit for erasing flash is usually a block (i.e., block erasure). Therefore, overwriting a page requires first erasing the entire encompassing block. This may cause significant write amplification, since data stored in the other pages of this block needs to be backed up before block erasure and written back after block erasure. Third, each block can be programmed/erased for a limited number of times. Therefore, a block will be worn out if the number of programs/erasures performed over it exceeds a certain threshold. To accommodate this special nature of flash memory, an out-of-place update rather than an in-place update strategy, is used in flash storage (Figure 1), in which when a page is overwritten, the new data will be simply written to a new empty page, while the old data will be temporarily preserved in the old page before garbage collection is invoked.

To be compatible with traditional block-based file systems (e.g., NTFS, EXT4, FAT32), a flash device is usually used as a block device by exposing a block-based access interface (i.e., a *flash-based block device* like SSD drive, USB stick, eMMC card, and SD card). The architecture of a flash-based storage system is shown in Figure 2, in which we can observe that to transparently handle the special nature of flash memory, a piece of special firmware, Flash Translation Layer (FTL), is introduced between the file system and the raw flash. The FTL usually implements four key functions: address translation, garbage collection, wear leveling, and bad block management.

Address translation. Flash-based block devices usually implement an out-of-place-update strategy, and therefore location of valid data may change over time. Thus, the FTL needs to keep track of mappings between addresses from upper layer and actual physical addresses in flash memory. Utilizing these mappings, the FTL can translate addresses from the upper layer (we call them Logical Block Addresses, or LBAs) to physical flash memory addresses (we call them Physical Block Addresses, or PBAs), providing a unique block-based access interface.

Garbage collection. Garbage collection is necessary to periodically reclaim those pages which store invalid stale data (we call these pages invalid pages). The garbage collection runs as follows: 1) It selects those blocks which satisfy a certain reclaim threshold as victim blocks. For example, the victim blocks can be those with the largest number of invalid pages. 2) It copies valid data stored in the victim blocks to free blocks, and meanwhile, updates the corresponding mappings; 3) It finally erases the victim blocks.
Wear leveling. Each flash block has a limited number of program/erase (P/E) cycles. Therefore, to prolong lifetime of flash memory, programmings/erasures need to be distributed evenly across the entire flash memory. This can be achieved by wear leveling.
Bad block management. Due to their limited P/E cycles, flash blocks will eventually turn "bad" and cannot be used to reliably store data any more. Bad block management is thus required to carefully manage those bad blocks.

2.3 K-means Clustering

K-means clustering [19] is a widely used unsupervised learning algorithm which is used for unlabeled data. It can be viewed as an optimization problem: given a set of n data points (each is from a d-dimension space), it places k centroids that can minimize the overall squared distance between each data point and its closest centroid.

Silhouette coefficient [33] is usually used to evaluate effectiveness of clustering, which is defined as:

$$s(i) = \frac{b(i)-a(i)}{\max\{a(i),b(i)\}}$$

where, for each data point i in the dataset, $a(i)$ is the average distance between i and all the other data points within the same cluster; $b(i)$ is the smallest average distance of i to all data points in any other cluster (i.e., i is not a member of this cluster); the average $s(i)$ over all data points of a cluster measures how tightly grouped all the data points in the cluster. In general, the silhouette coefficient ranges from -1 to $+1$, and a higher value usually indicates a more ideal clustering model.

3 Studying Access Activities on Flash Memory Caused by Real-world Ransomware

Ransomware behaves differently from benign software and other types of malware. For example, the ransomware usually needs to read the victim data, encrypt them, and 1) over-write the victim data with the ciphertext; or 2) write back the ciphertext to a different location and delete the victim data. Our intuition is, the special behaviors of ransomware in the upper layers (e.g, file system) will eventually cause repeated special access behaviors (i.e., access patterns) on the underlying flash memory. Since all the access requests issued by the file system will be handled by the underlying FTL (Sec. 2.2), by hacking into the real-world FTL, we may be able to detect abnormal access behaviors on the flash memory caused by ransomware, which is running in the upper layers.

In the FTL, there is no semantic information from the upper layers. To monitor the access activities of ransomware on flash memory, we can only utilize limited information as follows: access type (i.e., read/write), destination LBA, and size of each I/O request. Note that, the delete operation is usually implemented by writing the target location with NULL data.

Collecting access activities of ransomware on flash memory. To study specific access behaviors on flash memory caused by ransomware, we randomly selected and ran 518 prevalent crypto-ransomware samples collected from VirusTotal [1] and Github [13], including 11 different ransomware families. Column 1 and 2 in Table 1 summarize information about the samples being used. To collect access activities of ransomware on flash memory, we followed a few steps: 1) We ported an open-source FTL framework, OpenNFM [10], to an electronic development board LPC-H3131 [25]. 2) We attached LPC-H3131 as a USB mass storage device of a computer running a virtual machine with Windows 7 and FAT32. 3) We ran each ransomware sample in the virtual machine and used Tera Term Pro [36], a serial debugging tool, to output access activities. After each run, the virtual machine was restored to a clean state.
Extracting access patterns on flash memory caused by ransomware. To distill patterns of access behaviors on flash memory caused by ransomware, we utilize K-means clustering algorithm (Sec. 2.3), a widely used unsupervised learning algorithm. Compared to supervised learning [5], the unsupervised learning algorithm is more suitable for ransomware scenarios, because: the ransomware samples are created by diversified hackers or organizations, resulting in various ransomware categories with different behavior patterns, and such prior knowledge is unknown. We consider a *3*-dimensional space. These 3 dimensions include the access type (i.e., read or write), the starting destination LBA, and the size of each I/O request. In other words, each data point for K-means is a 3-dimension vector (access type, starting destination LBA, size of I/O request).

Training. We used half ransomware samples from each ransomware family (Table 1) for training purpose. For each sample, we collected 150 successive access. Note that each access contains the access type (i.e., read or write), the starting destination LBA, and the size of I/O request, and is viewed as a data point (i.e., a 3-dimension vector) for K-means clustering. During training, we changed value of k incrementally from 2 to 8, and calculated the silhouette coefficient for each k value. We found that K-means clustering can obtain the best clustering result[2] when k=4. After the training, we obtain the following results for our K-means clustering model: 1) Centroids of 4 clusters, which can be used to label new data; 2) Labels for the training data (each data point is assigned to a single cluster).

Extracting access patterns. After having fixed the K-means clustering model (i.e., k=4), we extracted access patterns using remaining ransomware samples from each ransomware family. For each ransomware sample, we selected 150 successive access. Each access corresponds to a 3-dimension data point, which was used as an input to the fixed K-means clustering model.

Column 3 to 6 in Table 1 show the classification results corresponding to the four clusters A, B, C, and D. The corresponding four types of access patters are summarized as follows (see Figure 3):

- **Type A**: the ransomware reads from successive LBAs, and writes back ciphertext (after encryption) to these LBAs with the same starting LBA. The size of the ciphertext being written back is almost[3] the same as the size of the content being read.
- **Type B**: the ransomware reads from successive LBAs, and writes back ciphertext to these LBAs with the same starting LBA. The size of the ciphertext being written back is smaller than the size of the content being read. This is because, the victim file may be compressed before being encrypted or only a portion of a file is encrypted (e.g., CryptoLocker).

[2]During training of K-means clustering, we found the silhouette coefficient is both high when k is either 2 or 4. However, $k = 2$ means there are only 2 clusters, which may easily lead to a situation that mistakenly categorizes normal software behaviors as ransomware behaviors, causing higher false positives. Therefore, we choose k as 4.

[3]Some ransomware samples like Ransom32 append RSA or AES key information at the end of each ciphertext, and hence the size of the data being written back is slightly increased.

Table 1: Real-world ransomware samples and K-means clustering results.

Family	#Samples	A	B	C	D	Time
TeslaCrypt	26(5.02%)	145	5	0	0	12
Locky	131(25.29%)	137	1	9	3	12
Cerber	23(4.44%)	143	4	3	0	2
Ransom32	28(5.40%)	131	8	11	0	8
CTB-locker	71(13.71%)	2	141	5	2	2
CryptoLocker	49(9.46%)	1	146	2	1	2
HydraCrypt	38(7.34%)	0	1	121	28	12
Samas	30(5.79%)	0	0	31	119	19
Bart	6(1.16%)	2	4	10	134	2
CryptoWall	102(19.69%)	1	7	15	127	13
Maktub	14(2.70%)	0	3	15	132	4
Total	518	-	-	-	-	-

- **Type C**: the ransomware reads from successive LBAs, and writes the ciphertext to new free LBAs. The size of the ciphertext being written is almost the same as the size of the content being read.

- **Type D**: the ransomware reads from successive LBAs, and writes the ciphertext to new free LBAs. The size of the content being read/written exhibits certain periodic characteristics, e.g., the length of each successive read is always 32 or 64 (in LBAs), and the length of each successive write is always 8 (in LBAs). Potential reasons for this type of pattern are: To attack victim files quickly, ransomware usually uses symmetric encryption to encrypt files [23]. As plaintext input and ciphertext output of symmetric encryption are commonly divided into groups with fixed size (e.g., 16 bytes in AES), both the successive read/write will exhibit certain periodic characteristics in length. For this type of ransomware, we also observed that it will not overwrite the original LBAs with random data until having attacked a large number of data (i.e., the delay of overwriting the original LBAs with randomness is more than 10 seconds [5]).

For each ransomware family, we also measured their average time of attacking the entire external storage on LPC-H3131 (see column 7 of Table 1, in minutes). The time varies from 2 to 19 minutes, which indicates that different ransomware variants may have a completely different attack time span. Therefore, detecting ransomware by observing overwrites in a small fixed time window (e.g., 10 seconds [5]) or simply restoring the victim system to a fixed historical check point (e.g., before 10 seconds [5]) may not be proper.

4 MimosaFTL Design

4.1 Model and Assumptions

System model. We consider mobile computing devices which are equipped with flash-based block devices (e.g., eMMC cards, SD cards, MiniSD cards, SSD drives) as external storage. This is the most common form of mobile computing devices nowadays. It can be also applied to desktop/laptop systems which use flash-based block devices (e.g., SSD drives).

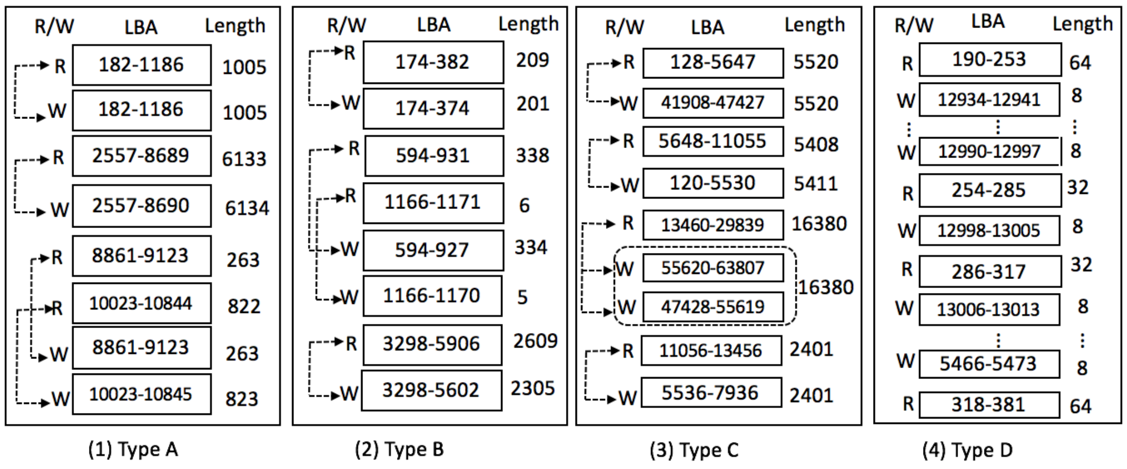


Figure 3: Four types of ransomware access patterns observed in FTL. R:read, W:write, Length: the LBA length of a successive read or write operation.

Threat model. We consider crypto-ransomware which encrypts the victim's data and asks for ransom money. We do not consider locker-ransomware which simply locks the victim system, since it can be easily defended by copying out the data from the victim system. In addition, the ransomware can compromise the entire host OS. However, it is not able to compromise the FTL. This is because, a flash-based block device usually only exposes a block access interface to the OS, with independent processor, memory, firmware (i.e., FTL), all of which are inside the flash device (Sec. 2.2) and invisible to the OS. Also, once the ransomware has successfully propagated to the victim system, it will run and encrypt the victim data to gain profit.
Assumptions. We assume ransomware will not imitate regular non-malicious software when performing encryption (e.g, slowly encrypt victim data in a long period). Such highly intelligent ransomware is rarely found in practice, since most ransomware tends to encrypt victims' data and ask for ransom in a short period. We also assume the mobile device always has spare storage space in flash memory.

4.2 Design Overview of MimosaFTL

MimosaFTL aims to design a practical ransomware defense strategy, which is secure against privileged ransomware. To achieve this goal, MimosaFTL adds three components to flash translation layer (FTL): ransomware detection, data backup and data recovery. The **ransomware detection** component is running in the FTL and monitoring access behaviours on flash memory, aiming to detect presence of ransomware in a timely manner. Since the privileged ransomware is not able to compromise the FTL (Sec. 4.1), the detection component can always remain secure. The **data backup** component backs up essential data for later recovery of external storage hacked by ransomware. The external storage includes both data and metadata. By utilizing the out-of-place-update feature of the flash-based block device, MimosaFTL simply modifies the garbage collection to preserve the data corrupted by the ransomware. In addition, MimosaFTL periodically keeps the latest version of the "good" metadata after relying on the detection component to identify a good state. The **data recovery** component restores the external storage corrupted by ransomware once ransomware has been detected. Both the data and the metadata will be recovered.

4.3 Ransomware Detection

Based on the access patterns extracted in Sec. 3, our detection scheme can simply monitor access behaviors on the flash memory, and compare them with the known ransomware patterns. We introduce a new data structure, namely, recently requested access (RRA) list, to keep track of the recent I/O requests from the block device. When an access request is received by the FTL, its abstract information will be inserted as an entry into the RRA list. Each entry of the list consists of three components: access type (i.e., *read* or *write*), starting destination LBA, and length of this access in LBAs (e.g., how many LBAs are read/written by this access). Note that the delete (e.g., trim [41]) operation can be treated as an overwrite. The user can determine the length of the RRA list. A larger length implies a less frequent detection process with less incurred overhead, while a smaller length implies a more frequent detection process with more overhead. When the list is filled, the detection process will be triggered. After the detection process, MimosaFTL will clear the RRA list and prepare for the next detection process.

We elaborate details of the detection process in Algorithm 1, in which access requests in the RRA list are analyzed to detect presence of ransomware as follows:

1) For ransomware which writes ciphertext to the original LBAs of the victim files (i.e., type A or B): we first check the RRA list, and find out whether there are a read and a write, that start with the same LBA address. If we can find such a pair of read and write, we compare their corresponding length. If the difference is smaller than a threshold δ, we find a type-A ransomware pattern. Otherwise, we find a type B ransomware pattern. We accumulate the number of type A patterns, and compute a ratio between the number of type A patterns and the total number of access requests in the RRA. If the ratio is larger than a threshold (i.e., $Threshold\_1$, which will be discussed in Sec. 6), we detect a type A ransomware attack. Similarly, we accumulate the number of type B patterns, and find out whether there is a type B ransomware attack.

2) For ransomware which writes ciphertext of the victim files to new free LBAs (i.e., type C or D): we first compare the length of a read request with the accumulating length of continuous write requests before the next read request is invoked. If the two lengths are close to each other (e.g., less than the threshold δ), we find a type C ransomware pattern. We accumulate the number of type C patterns, and compute a ratio between the number of type C patterns and the total number of access requests in the RRA. If the ratio is larger than $Threshold\_1$, we detect a type C ransomware attack.

For type D ransomware, we analyze the size of each successive read/write. We identify top-three sizes which appear most frequently, and summarize their total number of appearances. We compute a ratio between this sum and the total number of access requests in the RRA, and check whether or not the ratio exceeds another threshold, $Threshold\_2$. We call this "condition 1". In addition, we need to distinguish type D ransomware from benign applications like file encryption and compression applications, as they may generate similar I/O access patterns. The main difference is that benign applications are not designed to deny access to the original files, and usually the original files will not be deleted or overwritten, but ransomware will delete/overwrite the original files, and hence the mappings of the destination LBAs of the continuous read will turn invalid gradually. Therefore, we further check whether or not the mappings of the destination LBAs of the continuous read turn invalid gradually. we call this "condition 2". Only if both the condition 1 and 2 are true, we

Algorithm 1 Ransomware detection in MimosaFTL.

```
Require:
    The $RRA$ list, and each access request in the $RRA$ list includes the access type ($read/write$),
    the starting Destination LBA ($DLBA$) and the size ($Len$).
 1: $n_A = 0; n_B = 0; n_C = 0;$
 2: $Initialize\ a\ map\ <key,\ value>,\ where\ key = $ "$Len$" $and\ value = $
    "$\#\ of\ appearances\ of\ this\ Len$";
 3: for $each\ access\ request\ in\ the\ RRA\ list$ do
 4:     if $read's\ DLBA == write's\ DLBA\ \&\&\ |read's\ Len - write's\ Len| <= \delta$ then
 5:         $n_A + +;$
 6:     end if
 7:     if $read's\ DLBA == write's\ DLBA\ \&\&\ write's\ Len < read's\ Len$ then
 8:         $n_B + +;$
 9:     end if
10:     if $|the\ length\ of\ a\ read\ request - the\ total\ length\ of\ the\ continuous\ write\ requests\ before$
        $the\ next\ read\ request| <= \delta$ then
11:         $n_C + +;$
12:     end if
13:     $Update\ the\ map:\ if\ key\ (i.e.,\ Len)\ is\ not\ in\ the\ map,\ insert\ the\ key\ and\ value\ is$
        $initialized\ as\ 1;\ otherwise,\ increase\ the\ corresponding\ value\ by\ 1$
14: end for
15: $Obtain\ the\ top\ 3\ largest\ values\ from\ the\ map:\ v_1,\ v_2,\ v_3;$
16: $n_S = v_1 + v_2 + v_3;$
17: $n \leftarrow length\ of\ the\ RRA\ list;$
18: if $(n_A/n > Threshold\_1)\ ||\ (n_B/n > Threshold\_1)\ ||\ (n_C/n > Threshold\_1)$ then
19:     $Ransomware\ detected;$
20: else if $(n_S/n > Threshold\_2)\ \&\&\ Mappings\ of\ the\ destination\ LBAs\ of\ the\ continuous$
    $read\ become\ invalid\ gradually$ then
21:     $Ransomware\ detected;$
22: end if
```

conclude that a type D ransomware attack is detected.

4.4 Data Backup

To allow restoring external storage after ransomware attacks, MimosaFTL needs to back up both data and metadata (e.g., the mapping table, which records mappings between LBAs and PBAs).
Back up metadata. For creating metadata backup, we take advantage of the fact that MimosaFTL has a detection component. Periodically, if no ransomware is detected, we will create a redundant copy of the current essential metadata and discard old versions of metadata. Note that: 1) The size of metadata is usually much smaller than data, and hence only a few flash blocks are required to be used to store metadata. In addition, the blocks storing metadata backup are usually invisible to the OS, and hence cannot be corrupted by ransomware. 2) The metadata will be correctly backed up if the ransomware is correctly detected, which is of high probability[4] according

[4]Rarely when false negative happens, the metadata may not be properly backed up, and they can only be recovered

to our evaluation in Sec. 6.

Back up data. Thanks to the out-of-place-update feature of flash-based block devices, the data will be temporarily preserved and can be used for data recovery from ransomware attack. However, garbage collection will eventually reclaim those data which have been corrupted by ransomware since they have become invalid. Simply disabling garbage collection [14] can prevent those data from being removed. This, however, is problematic, since data will fill the entire flash memory shortly without a garbage collection. Taking advantage of the detection component, we use a phased garbage collection strategy. The idea is to perform garbage collection periodically only when the system is sure that there was no ransomware present. Here we define the "phase" as a time period between two sub-sequential detection processes. When a detection process is invoked and does not detect presence of ransomware (i.e., the beginning of a phase), the system will perform the following steps: 1) conducting a garbage collection on blocks storing invalid data[5]; 2) backing up the current metadata. In addition, any data being overwritten (note that deletion is a special overwrite operation) during this phase will be frozen and will not be removed by garbage collection. When the detection process detects presence of ransomware, the data recovery component will be invoked (Sec. 4.5).

4.5 Data Recovery

Once ransomware is detected, MimosaFTL will immediately inform the user and make the storage as read-only. Then, the ransomware will be blocked and removed by the user (blocking and removing malware is not our focus in this work). Once the attack is confirmed by the user, a recovery component will be triggered to interact[6] with the victim user to restore the external storage being corrupted by the ransomware. Since the data are preserved due to the out-of-place-update feature of flash memory, the major problem in the recovery component will be restoring the metadata at the point of time right before the ransomware starts to damage the external storage (we call this point of time "corruption point"). Such a recovery design is advantageous, because: 1) The expensive direct data recovery can be avoided since the "good" data are preserved in flash memory, and by restoring the metadata which point to the "good" data, the external storage can be recovered. 2) The recovery of metadata can be done efficiently, considering their small size.

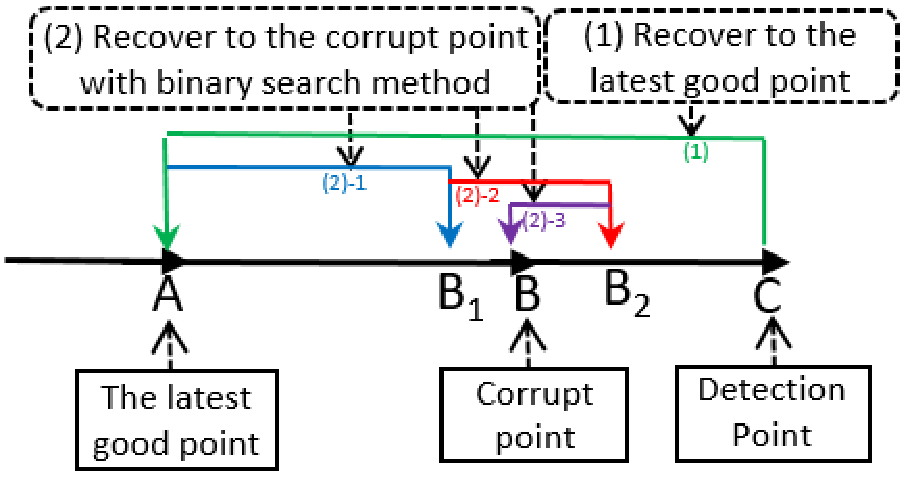
However, restoring the metadata at the corruption point is challenging, because: The detection component of MimosaFTL relies on observing access behaviors of ransomware for detection, and a few user files will be corrupted unavoidably before the point of time when ransomware is detected (we call this point of time "detection point"). Since MimosaFTL periodically backs up the latest "good" metadata (we call this point of time "latest good point"), we can first restore the external storage to this latest good point, and then approach the corruption point from the latest good point.

To restore the metadata at the corruption point, MimosaFTL needs to rely on information stored in the OOB areas. MimosaFTL will store additional information in the OOB of each flash page storing user data: backup version number (which records the version number of the periodical metadata backup), writing sequence number (which records the sequence number of writing in a

by performing "brute force".

[5]Rarely when false negative happens, the invalid data which were corrupted by ransomware will be removed by garbage collection at the beginning of each phase. In this case, MimosaFTL can only restore data within the phase. This is a limitation of MimosaFTL and will be investigated in our future work.

[6]User involvement (via customized I/O commands, e.g., SCSI commands) is usually necessary, since only the user knows the latest version of the data.



(2) Recover to the corrupt point with binary search method
(1) Recover to the latest good point
(1)
(2)-1
(2)-2
(2)-3
A
B1
B
B2
C
The latest good point
Corrupt point
Detection Point

Figure 4: Recovery timeline of MimosaFTL.

specific periodical metadata backup) and destination LBA. Note that we only use a small portion of the OOB area, which will not significantly affect its regular use. For example, in our experimental evaluation, we only need to use 8 bytes of the OOB, which is 12.5% of its entire capacity.

When MimosaFTL detects the ransomware and informs the victim user, the user will trigger the recovery component to perform the following steps (see Figure 4): First, it will recover the system to the latest good point (i.e., from point C to A). This step is straightforward, since MimosaFTL has backed up metadata of point A and all the valid data of point A has been preserved (Sec. 4.4). Second, to recover the system to the corruption point (i.e., from point A to B), MimosaFTL will read the OOB areas of the entire physical flash pages, and find out the ones with the largest backup version number (i.e., the latest backup version), and then create a list of destination LBA and PBA mappings in the order of their writing sequence number. Next, MimosaFTL will recover the mapping table at point B using an efficient binary search approach as described below:

MimosaFTL first applies half of the entire changes to the metadata at point A, restoring the metadata at point B_1. The user then checks the data (at the OS level) recovered at point B_1. If the user does not find corruption at B_1, the corruption point should be located somewhere between B_1 and C (otherwise, the corruption point should be located somewhere between A and B_1), and the next point being examined should be B_2. The metadata at B_2 can be restored by applying half of entire changes between B_1 and C to the metadata at point B_1. Recursively, the search will reach the corruption point B. Note that interacting with the user for data recovery is necessary since only the user knows the latest good state of the data. However, due to the efficient binary search, the user involvement can be kept small (i.e., log(n), where n is the number of write operations between the latest good point and the detection point).

The recovery process of MimosaFTL requires interacting with the victim user, which can be achieved by taking advantage of the reserved operation codes (i.e., *0x60H* to *0x7FH*) of SCSI commands [16, 46] if the flash storage medium supports SCSI interface.

5 Security Analysis and Discussion

5.1 Security Analysis

A pattern-based detection solution usually cannot guarantee 100% accuracy. In the following, we analyze the security of MimosaFTL under two cases: 1) the ransomware is correctly detected; 2) the ransomware is not correctly detected.
Case 1: the detection component correctly detects the ransomware. Clearly in MimosaFTL, both data and metadata will be correctly backed up and the ransomware is not supposed to bypass the FTL to corrupt those backups even if it can compromise the host operating system (Sec. 4.1). Once the ransomware is correctly detected, the recovery component will be correctly triggered by the user to recover the external storage to the corruption point.
Case 2: the detection component does not correctly detect the ransomware. This usually includes *false positives* and *false negatives*, which will be discussed respectively as follows.

- Rarely, benign applications and regular user operations may exhibit similar access patterns as ransomware, causing false positives. MimosaFTL can handle false positives because, the data recovery component is triggered by the user, and the user can simply not trigger the data recovery component if that is a false positive.
- Some ransomware variants may escape from being detected (e.g., new ransomware variants), causing false negatives. When the false negative happens, MimosaFTL is at least as good as FlashGuard [15], which does not employ ransomware detection and can at most allow restoring data from last check point. Similarly, MimosaFTL can also allow restoring the data from the last detection if a false negative happens.

5.2 Discussion

Working on the FTL layer rather than the upper layers. MimosaFTL is a solution which requires being incorporated into the flash translation layer (FTL). This seems unavoidable since its security strongly relies on the close nature of flash memory. Such an incorporation is not unique, since a lot of existing security defenses [43, 9, 30, 18, 17, 8, 15, 14] for flash memory have a similar requirement of incorporating security strategies into the FTL. The incorporation could be achieved by either collaborating with flash memory vendors or turning the defenses into industry standards.
Handling ransomware-like benign applications. Some special benign applications like encryption, compression, and deletion applications may exhibit ransomware-like access behaviors. MimosaFTL is designed to not mistakenly detect them as ransomware: 1) For benign encryption and compression applications, they commonly treat the original file content carefully, since their ultimate goal is to generate an encrypted/compressed version of the original file, rather than to restrict access to the file [20]. In other words, the original files usually remain intact when they are running, though automatic deletion may be deliberately activated by the user after the encryption or compression is done. 2) For secure deletion applications, they usually open a file and overwrite its content with new, meaningless data [31], e.g., all zeros. However, they usually will not read the files during deletion. 3) If the user changes the file content via a benign application (e.g., updating the data of a Microsoft Word file), it may also generate similar access patterns as ransomware. However, there are some key differences. For example, benign applications usually read a single file at a time and modify different parts of the file continuously, but ransomware usually performs reading and rewriting in an interleaved manner.

Protecting SCSI interface. MimosaFTL supports user interaction via SCSI commands (note that this requires the flash storage media to support SCSI interface). To prevent privileged malware from abusing this new interface to disturb our design, we can introduce a simple authentication using secrets only known by the user. Specifically, every time when the SCSI interface is used, the user needs to provide a secret password which needs to be verified by the FTL (the secret password is stored in the metadata area of the flash which is invisible to the upper layer and hence the ransomware). Since the ransomware does not have this secret password and is not able to pass the authentication in order to use this SCSI interface.

6 Implementation and Evaluation

6.1 Implementation

We have implemented a prototype of MimosaFTL using OpenNFM [10], an open source NAND flash controller framework. We ported MimosaFTL to LPC-H3131 [25], a development board equipped with 180 MHz ARM processor, 512 MB NAND flash, and 32 MB SDRAM. The block size of the NAND flash is 128 KB and the page size is 2 KB. The entire NAND flash has 4,096 erase blocks, and each block is composed of 64 pages. Moreover, the size of the OOB area in each page is 64 bytes. Each mapping entry can be represented by 3 bytes, and the mapping table occupies approximately 6 blocks.

6.2 Evaluation

6.2.1 Effectiveness of MimosaFTL in Detection

We evaluate effectiveness of the detection component of MimosaFTL by looking into its false positives and false negatives.

Dataset construction. To provide a comprehensive evaluation of the detection component, as shown in Table 2, we collected 346 new prevalent ransomware samples belonging to 11 different families from VirusTotal (73.4%) and Github (26.6%). Our dataset covers a majority of existing ransomware families that appear from 2001 to 2018. In addition, we built a dataset containing 95 benign application samples (a portion of them are shown in Table 2), including: 1) software that has ransomware-like behaviors such as file encryption, compression and data deletion; 2) multimedia tools and applications (e.g., media player, audio/video transcoding applications, and games); 3) developer tools; 4) office tools. Besides these benign application samples, we also collected I/O access requests from installing/upgrading these benign software and web server (e.g., search engine service, web mail server).

Ransomware detection accuracy. In order to evaluate detection accuracy of MimosaFTL, we measured both false negatives and false positives, varying the thresholds to determine the best detection effectiveness. We choose δ as 5 (in LBAs).

First, we evaluated whether MimosaFTL can successfully distinguish ransomware from benign applications, following a few steps: 1) We ported MimosaFTL to LPC-H3131, and used the board as an external storage; 2) We ran all the ransomware samples and benign application samples; 3) We varied the thresholds in the detection algorithm (i.e., $Threshold\_1$ and $Threshold\_2$) and kept track of those thresholds by which the detection algorithm successfully detects ransomware samples or mistakenly detects benign application samples as ransomware in Figure 5. The left half of the

Table 2: Ransomware samples and portion of the selected benign applications.

Ransomware	#Samples	Benign application	Main capability
Samas	36	7-zip	Compression
Cerber	47	Winzip	Compression
Ransom32	53	WinRAR	Compression
Maktub	14	TrueCrypt	Encryption
Jigsaw	56	DiskCryptor	Encryption
Radamant	36	SDelete	Deletion
CryptoFortress	37	Eraser	Deletion
HydraCrypt	29	Davinci Resolve	Multimedia tools
WannaCry	12	Eclipse	Developers tools
Critroni	20	Anaconda	Developers tools
CryptoDefense	6	SQLite	Developers tools

figure is for ransomware samples (with thresholds vary between 0.8 and 0.97), and the right half of the figure is for benign application samples (with threshold vary between 0.02 and 0.5). A clear difference is observed between the ransomware samples (minimum 0.8) and the benign application samples (maximum 0.5).

Second, to determine thresholds described in Algorithm 1, we measured false positives and false negatives by varying $Threshold\_1$ for type A/B/C ransomware and $Threshold\_2$ for type D ransomware. As shown in Figure 6, MimosaFTL does not have any false positives/false negatives when $Threshold\_1 \in (0.55, 0.78)$ and $Threshold\_2 \in (0.63, 0.87)$.

6.2.2 Efficiency of MimosaFTL in Recovery

We evaluate how efficiently the MimosaFTL can recover the external storage corrupted by ransomware. Table 3 shows each separate time spent on recovering all the 512MB data in LPC-H3131. We observe that most time is spent on building the LBA and PBA mappings, which requires reading OOB areas of all those flash pages storing user data.

Table 3: Individual time components for recovery (in seconds).

Restoration to point A	Build LBA and PBA mappings	Rebuild a mapping table	System restart
0.65	2.83	0.43 to 1.26	1.32

6.2.3 Impact of MimosaFTL on Flash-based Block Devices

We evaluate impact of MimosaFTL on regular flash-based block devices, in terms of I/O throughput and lifetime of flash memory.

Impact on flash storage I/O throughput. To access impact of MimosaFTL on the storage I/O throughput, we benchmarked the original OpenNFM and MimosaFTL using fio [12] with non-buffered I/O option. We ran fio in a host computer with Intel i5 CPU (3.30GHz, 4GB RAM) and Windows 10 Pro 64-bit. We set the array length of the RRA list as 150 (i.e., the detection

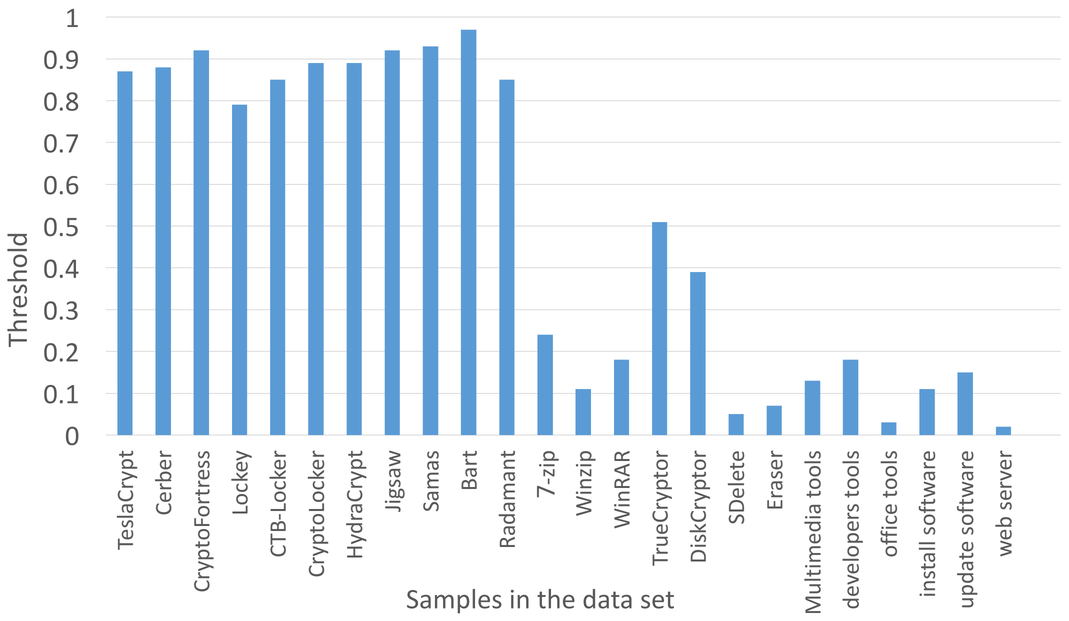


Figure 5: The threshold that the detection component successfully detects the ransomware or mistakenly detects benign applications as ransomware.

component is triggered upon every 150 continuous access requests). We created backup for the essential FTL metadata (e.g., addresses mapping table) once a day if ransomware is not detected.

The benchmark results for I/O throughput of the two systems are shown in Figure 7. We observe that MimosaFTL decreases the read (i.e., sequential and random read) throughput by up to 6.8%, and decreases the write (including sequential and random write) throughput by up to 7.2%. We analyze the additional overhead of MimosaFTL in the following: 1) MimosaFTL running the detection algorithm needs to update the RRA list, and the detection process is triggered when the RRA list is filled. Since the RRA list is maintained in RAM, updating it does not incur too much overhead. In addition, the detection process needs to analyze all the entries in the RRA list which will incur overhead. However, this only happens periodically. 2) MimosaFTL needs to back up metadata daily in our implementation, which takes as less as 0.1 seconds for each back up operation. 3) MimosaFTL adopts phased garbage collection, which delays execution of garbage collection on invalid blocks. MimosaFTL tries to perform garbage collection during idle time, reducing its impact on the entire performance. 4) To allow restoring mapping tables during recovery, MimosaFTL needs to write relevant information to the OOB area of each page. Compared to OpenNFM, this does not bring extra overhead, as OpenNFM also needs to keep similar information in the OOB area to enable recovery from power failure.

Impact on the flash device's lifetime. To prolong lifetime of flash memory, MimosaFTL utilizes a global wear leveling strategy. In MimosaFTL, when allocating blocks, blocks with smaller P/E cycles will be allocated first. In addition, the blocks having larger P/E cycles will be swapped with blocks having smaller P/E cycles. To evaluate wear leveling effectiveness, we use hoover economic

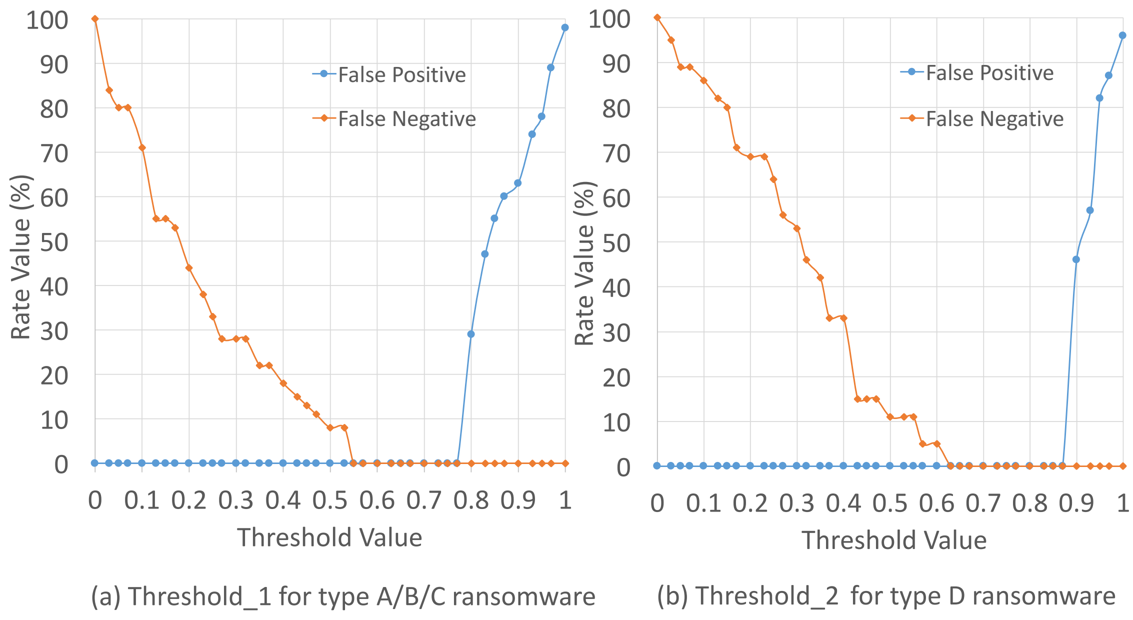


Figure 6: The false positive rate and false negative rate vary with adjustment of the threshold.

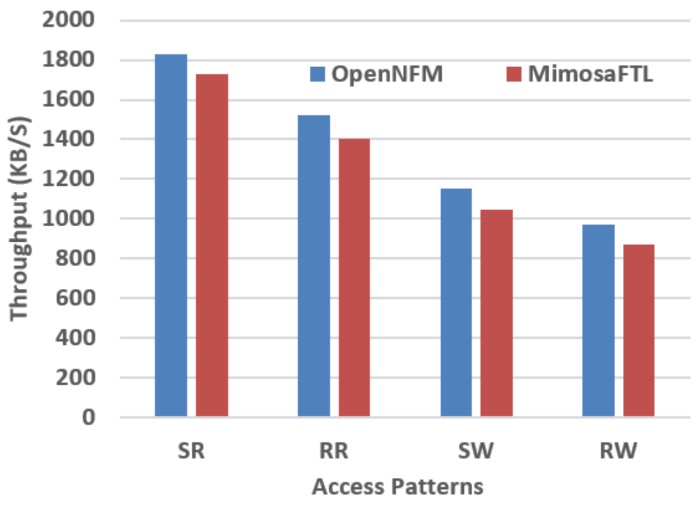


Figure 7: Comparisons of read/write throughput (KB/s) between OpenNFM and MimosaFTL. SR - sequential read, RR - random read, SW - sequential write, RW - random write.

wealth inequality indicator [32, 8, 18], which calculates an appropriately normalized sum of the difference between each measurement and their mean. Assuming erasure counts of all the n erase blocks are e_1, e_2,..., e_n, and $E = \sum_{i=1}^{n} e_i$, a wear leveling inequality (WLI) can be computed as:

$WLI = \frac{1}{2}\sum_{i=1}^{n}\|\frac{e_i}{E} - \frac{1}{n}\|$. This indicates the fraction of erasures that must be re-assigned to other blocks in order to achieve completely even wear.

We repeatedly wrote data to the board, completely filled the 480 MB flash, and then erased the data. After having written 480 GB data, we calculated the number of erasures performed on each flash block. We then computed the WLI, obtaining 9.2%. This small value indicates a small impact of MimosaFTL on the lifetime of flash memory.

7 Related Work

The existing work of ransomware defense can be categorized into detection and recovery.
Ransomware detection. Existing ransomware detection approaches mainly monitor typical ransomware file system activities [20, 21, 34, 11] or analyze cryptographic primitives [21, 11, 23]. Kharraz et al. [21] were the first to analyze a large number of ransomware samples. They suggest some potential defenses in term of file system interactions, encryption mechanisms and financial incentives. Unveil [20] generates an artificial user environment and monitors desktop lockers, file access patterns and I/O data entropy. CryptoDrop [34] observes file type changes and measures file modifications using similarity-preserving hash functions and shannon entropy to detect ransomware. ShieldFS [11] monitors file system access activities and collects features like folder listing, file read/write/rename, file type and write entropy.

Additionally, other work utilizes honeypot techniques [28], software-defined networking (SDN) [7] or machine learning approaches [35] for ransomware detection. The detection approaches discussed so far are workable under the assumption that the OS is trusted and the malware cannot compromise it. However, advanced ransomware may run with root privileges, and is able to disable or bypass the detection mechanisms. The detection component of MimosaFTL is secure against this type of privileged ransomware.
Data recovery from ransomware attacks. Inspired by copy-on-write file systems, ShieldFS [11] automatically shadows a file whenever the original one is modified. PayBreak [23] leverages the fact that in a hybrid cryptosystem the session key must be used during the symmetric encryption. It observes the use of these keys, holds them in escrow, and is able to decrypt files that would otherwise only be recoverable by paying the ransom. The aforementioned approaches are vulnerable to the privileged ransomware which can compromise the OS and disturb the data (or key) recovery.

FlashGuard [15] can defend against the privileged ransomware by exploiting features present in the lower layer flash memory. However, due to lack of a detection algorithm as well as a user-friendly recovery component, FlashGuard is far from being practical. SSD-Insider [5] incorporates a ransomware detection component based on the overwriting patterns in a small fixed time window (e.g., 10 seconds). However, SSD-Insider is not practical either because: First, it relies on an observation that ransomware "conducts overwriting immediately after reading and encrypting the victim's file", which is not necessarily true according to our study (Sec 3). Second, its recovery component is coarsely designed and can only recover data before 10 seconds.

8 Conclusion

In this work, we propose MimosaFTL, the first secure yet more practical ransomware defense strategy for mobile computing devices that are equipped with flash memory as external storage.

Security analysis and experimental evaluation show that MimosaFTL can defend against privileged ransomware with a small negative impact on storage performance and device's lifetime.

References

[1] Virustotal. https://www.virustotal.com/en/, 2018.

[2] Mohammad Mehdi Ahmadian, Hamid Reza Shahriari, and Seyed Mohammad Ghaffarian. Connection-monitor & connection-breaker: A novel approach for prevention and detection of high survivable ransomwares. In *Information Security and Cryptology (ISCISC), 2015 12th International Iranian Society of Cryptology Conference on*, pages 79–84. IEEE, 2015.

[3] Mohammad Mehdi Ahmadian, Hamid Reza Shahriari, and Seyed Mohammad Ghaffarian. Connection-monitor & connection-breaker: A novel approach for prevention and detection of high survivable ransomwares. pages 79–84, 2015.

[4] Nicolo Andronio, Stefano Zanero, and Federico Maggi. Heldroid: Dissecting and detecting mobile ransomware. pages 382–404, 2015.

[5] 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 *38th IEEE International Conference on Distributed Computing Systems,ICDCS 2018, Vienna, Austria, July 2-6, 2018*, pages 875–884, 2018.

[6] Matias Bjørling, Javier González, and Philippe Bonnet. Lightnvm: The linux open-channel ssd subsystem. In *FAST*, pages 359–374, 2017.

[7] Krzysztof Cabaj, Marcin Gregorczyk, and Wojciech Mazurczyk. Software-defined networking-based crypto ransomware detection using http traffic characteristics. *Computers & Electrical Engineering*, 2017.

[8] Bo Chen, Shijie Jia, Luning Xia, and Peng Liu. Sanitizing data is not enough!: towards sanitizing structural artifacts in flash media. In *Proceedings of the 32nd Annual Conference on Computer Security Applications*, pages 496–507. ACM, 2016.

[9] Bo Chen and Radu Sion. Hiflash: A history independent flash device. *arXiv preprint arXiv:1511.05180*, 2015.

[10] Google Code. Opennfm. `https://code.google.com/p/opennfm/`, 2011.

[11] Andrea Continella, Alessandro Guagnelli, Giovanni Zingaro, Giulio De Pasquale, Alessandro Barenghi, Stefano Zanero, and Federico Maggi. Shieldfs: a self-healing, ransomware-aware filesystem. In *Proceedings of the 32nd Annual Conference on Computer Security Applications*, pages 336–347. ACM, 2016.

[12] Freecode. fio. http://freecode.com/projects/fio, 2014.

[13] Github. A repository of live malwares for your own joy and pleasure. https://github.com/ytisf/theZoo, 2018.

[14] 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. ACM, 2017.

[15] 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.

[16] INCITS. Scsi command operation codes, 2015. `http://www.t10.org/lists/op-num.htm`.

[17] Shijie Jia, Luning Xia, Bo Chen, and Peng Liu. Nfps: Adding undetectable secure deletion to flash translation layer. In *Proceedings of the 11th ACM on Asia Conference on Computer and Communications Security*, pages 305–315. ACM, 2016.

[18] Shijie Jia, Luning Xia, Bo Chen, and Peng Liu. Deftl: Implementing plausibly deniable encryption in flash translation layer. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 2217–2229. ACM, 2017.

[19] Tapas Kanungo, David M. Mount, Nathan S. Netanyahu, Christine D. Piatko, Ruth Silverman, and Angela Y. Wu. An efficient k-means clustering algorithm: Analysis and implementation. *IEEE Trans. Pattern Anal. Mach. Intell.*, 24(7):881–892, 2002.

[20] Amin Kharraz, Sajjad Arshad, Collin Mulliner, William K Robertson, and Engin Kirda. Unveil: A large-scale, automated approach to detecting ransomware. In *USENIX Security Symposium*, pages 757–772, 2016.

[21] Amin Kharraz, William Robertson, Davide Balzarotti, Leyla Bilge, and Engin Kirda. Cutting the gordian knot: A look under the hood of ransomware attacks. In *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*, pages 3–24. Springer, 2015.

[22] Youngjae Kim, Brendan Tauras, Aayush Gupta, and Bhuvan Urgaonkar. Flashsim: A simulator for nand flash-based solid-state drives. In *Advances in System Simulation, 2009. SIMUL'09. First International Conference on*, pages 125–131. IEEE, 2009.

[23] Eugene Kolodenker, William Koch, Gianluca Stringhini, and Manuel Egele. Paybreak: defense against cryptographic ransomware. In *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security*, pages 599–611. ACM, 2017.

[24] Xin Luo and Qinyu Liao. Awareness education as the key to ransomware prevention. *Information Systems Security*, 16(4):195–202, 2007.

[25] Mantech. Lpc-h3131. http://www.mantech.co.za/, 2017.

[26] Mcafee. Wannacry: Ransomware spreads like wildfire, attacks over 150 countries. `https://securingtomorrow.mcafee.com/consumer/consumer-threat-notices/wannacry-ransomware-attacks/`.

[27] Shagufta Mehnaz, Anand Mudgerikar, and Elisa Bertino. Rwguard: A real-time detection system against cryptographic ransomware. In *International Symposium on Research in Attacks, Intrusions, and Defenses*, pages 114–136. Springer, 2018.

[28] Chris Moore. Detecting ransomware with honeypot techniques. In *Cybersecurity and Cyberforensics Conference (CCC), 2016*, pages 77–81. IEEE, 2016.

[29] Bharti Nagpal and Vinayak Wadhwa. Cryptoviral extortion: Evolution, scenarios, and analysis. In *Proceedings of the International Conference on Signal, Networks, Computing, and Systems*, pages 309–316. Springer, 2016.

[30] Joon-Young Paik, Keuntae Shin, and Eun-Sun Cho. Poster: Self-defensible storage devices based on flash memory against ransomware. In *Proceedings of IEEE Symposium on Security and Privacy*, 2016.

[31] Joel Reardon, David A Basin, and Srdjan Capkun. Sok: Secure data deletion. *ieee symposium on security and privacy*, 12(3):301–315, 2013.

[32] Joel Reardon, Srdjan Capkun, and David Basin. Data node encrypted file system: Efficient secure deletion for flash memory. pages 17–17, 2012.

[33] PJ Rousseeuw. Silhouettes: A graphical aid to the interpretation and validation of cluster analysis. *Journal of Computational & Applied Mathematics.*, 20(20):53–65, 1999.

[34] Nolen Scaife, Henry Carter, Patrick Traynor, and Kevin RB Butler. Cryptolock (and drop it): stopping ransomware attacks on user data. In *Distributed Computing Systems (ICDCS), 2016 IEEE 36th International Conference on*, pages 303–312. IEEE, 2016.

[35] Daniele Sgandurra, Luis Muñoz-González, Rabih Mohsen, and Emil C Lupu. Automated dynamic analysis of ransomware: Benefits, limitations and use for detection. *arXiv preprint arXiv:1609.03020*, 2016.

[36] SOFTPEDIA. Tera term pro web. http://www.softpedia.com/get/Network-Tools/Telnet-SSH-Clients/Tera-Term-Web.shtml, 2018.

[37] Statista. Number of mobile phone users worldwide from 2013 to 2019 (in billions), 2018. `https://www.statista.com/statistics/274774/forecast-of-mobile-phone-users-worldwide/`.

[38] Kul Prasad Subedi, Daya Ram Budhathoki, Bo Chen, and Dipankar Dasgupta. Rds3: Ransomware defense strategy by using stealthily spare space. In *Computational Intelligence (SSCI), 2017 IEEE Symposium Series on*, pages 1–8. IEEE, 2017.

[39] Symantec. 2017 internet security threat report. `https://www.symantec.com/security-center/threat-report`.

[40] Symantec. A new breed of threat: Wannacry and petya. `https://www.symantec.com/content/dam/symantec/docs/security-center/white-papers/istr-ransomware-2017-en.pdf`.

[41] Cactus Technologies. Solid state drive primer-controller functions-trim command. `https://www.cactus-tech.com/resources/blog/details/solid-state-drive-primer-12-controller-functions-trim-command`.

[42] Thebestvpn. Cyber security statistics. `https://thebestvpn.com/cyber-security-statistics-2018/`.

[43] Michael Yung Chung Wei, Laura M Grupp, Frederick E Spada, and Steven Swanson. Reliably erasing data from flash-based solid state drives. In *Fast*, volume 11, pages 8–8, 2011.

[44] Wikipedia. Trojan.winlock. `https://ru.wikipedia.org/wiki/Trojan.Winlock`.

[45] Joobeom Yun, Junbeom Hur, Youngjoo Shin, and Dongyoung Koo. Cldsafe: An efficient file backup system in cloud storage against ransomware. *IEICE TRANSACTIONS on Information and Systems*, 100(9):2228–2231, 2017.

[46] Qionglu Zhang, Shijie Jia, Bing Chang, and Bo Chen. Ensuring data confidentiality via plausibly deniable encryption and secure deletion–a survey. *Cybersecurity*, 1(1), 2018.