SSD-Insider: Internal Defense of Solid-State Drive against Ransomware with Perfect Data Recovery

SungHa Baek, Inha University
Youngdon Jung, DGIST
Aziz Mohaisen, UCF
Sungjin Lee, DGIST
DaeHun Nyang, Inha University

***Abstract*—Ransomware is a malware that encrypts victim's data, where the decryption key is released after a ransom is paid by the data owner to the attacker. Many ransomware attacks were reported recently, making anti-ransomware a crucial need in security operation, and an issue for the security community to tackle. In this paper, we propose a new approach to defending against ransomware inside NAND flash-based SSDs. To realize the idea of defense-inside-SSDs, both a lightweight detection technique and a perfect recovery algorithm to be used as a part of SSDs firmware should be developed. To this end, we propose a new set of lightweight behavioral features on ransomware's overwriting pattern, which are invariant across various ransomwares. Our features rely on observing the block I/O request headers only, and not the payload. For perfect and instant recovery, we also propose using the delayed deletion feature of SSDs, which is intrinsic to NAND flash. To demonstrate their feasibility, we implement our algorithms atop an open-channel SSD as a working prototype called *SSD-Insider*. In experiments using eight real-world and two in-house ransomwares with various background applications running, SSD-Insider achieved a detection accuracy 0% FRR/FAR in most scenarios, and only 5% FAR when heavy overwriting resembling ransomware's data wiping occurs. SSD-Insider detects ransomware activity within 10s, and recovers instantly an infected SSD within 1s with 0% data loss. The additional software overheads incurred by the SSD-Insider is just 147 *n*s and 254 *n*s for 4-KB reads and writes, respectively, which is negligible considering NAND chip latency (50-1000 μs).**

I. Introduction

Ransomware, a type of malicious software (malware) that holds a user's data hostage to collect ransom, has been on the rise. To extort ransom from data owners, ransomware encrypts data using strong encryption algorithms, and the decryption keys are released only after a ransom is paid by the data owners. Ransomware tries to subvert standard malware defenses by using complex command and control (C&C) networks that utilize anonymous communication systems such as Tor, and by collecting ransom using cryptocurrency, such as Bitcoin, which is very difficult to track. The high potential financial gains to attackers and the difficulty of defending against ransomwares make them a "profitable business" to cybercriminals. For example, the Nayana web hosting firm in South Korea was attacked by a crypto ransomware called Erebus in 2017, where more than 3,400 web sites were affected according to the Korea Internet and Security Agency. As a result Nayana paid 1.14 million USD to the attackers in three installments to get the key, and even then could not totally recover the data [1]. Similar attacks were reported in the US. For example, in 2016, the Hollywood Presbyterian Medical Center was infected by a crypto ransomware, and had to pay 40 bitcoins (about $17,000 at that time) to the attacker after 10 days of not being able to operate because it could not access patients medical records [2].

Ransomwares are generally classified in two groups: locker and crypto ransomwares [3]. Locker ransomware prevents users from accessing an infected machine, while crypto ransomwares, such as WannaCry, CryptoWall, TeslaCrypt (a.k.a. AlphaCrypt) and Locky, lock users' data by encrypting the data to prevent users' data access. Most of the newly discovered ransomwares were crypto ransomware, compared to 80% of crypto ransomwares in 2015 [3]. As such, more attention should be paid to crypto ransomwares and their defenses, which is the case of this work.

In this work, we propose a new approach to ransomware detection and recovery by introducing the concept of NAND flash internal defense. Unlike the application layer detection systems, our approach is universal to every platform (operating system, filesystem, hardware configuration, etc.), and provides immunity for systems without requiring any vaccine software. By introducing six invariant features of ransomware for a machine learning algorithm and by taking advantage of SSD's unique nature, we develop a behavior-based detection and recovery system that runs inside SSDs as a form of storage firmware. With our ransomware detection and recovery algorithms built in, SSDs can effectively detect ransomwares' activity and perfectly recover user's data, even when 1) a user does not run any ransomware monitoring application, 2) the OS (or middleware) does not execute any ransomware monitoring task, 3) it works for an unknown OS, an unknown file system and unknown applications, and 4) it confronts previously unknown ransomwares. We realize the ransomware detection and recovery algorithms in a system, called *SSD-Insider*, which works as *a universal platform*. To this end, in this paper we deliver the following contributions:

- We design six invariant features to capture ransomwares' behavioral characteristics. We use the block address, size, and type of an IO request to an SSD, and analyze their correlation with ransomware activities using various real-world ransomwares.
- We build a machine learning technique for detecting ransomware using a binary decision tree using ID3 (Iterative Dichotomiser 3) [4].
- We design a new Flash Translation Layer (FTL) scheme that supports an instant recovery of infected files by leveraging the intrinsic delayed deletion feature of NAND flash.
- To show the feasibility of our system inside an SSD, we implement SSD-Insider using an open-channel SSD platform as a working prototype.

- We evaluate SSD-Insider using eight real-world ransomwares and in-house ransomwares while various background applications were running. Our implementation of SSD-Insider has 100% detection accuracy, with less than 10 seconds of detection latency. SSD-Insider also recovered encrypted files within 1 second, and without any data loss after recovery.

II. Motivation

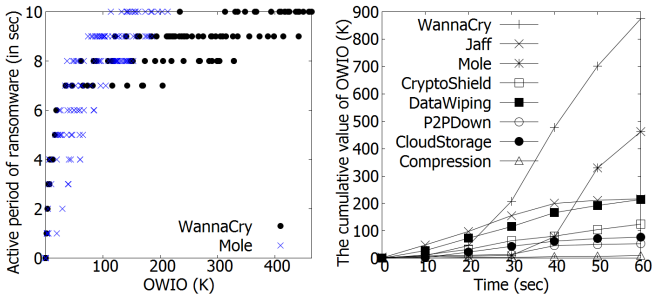
A. Limitations of legacy approaches

It is necessary to define effective ransomware-specific features so that a detector can recognize ransomware activity immediately and accurately. However, it is difficult to detect ransomware only by observing their signatures, because their binaries change frequently. For example, an interesting "ransomware-as-a-service" model called TOX was found in 2015, which helps inexperienced attackers to easily build a variant ransomware [5]. Ransomware also can easily evade anti-ransomware, firewall rule sets, and spam filters. The mutable nature of ransomware makes it also harder to detect only by signatures, and pushes the detection paradigm further into the realm of behavior-based defenses.

File type-based or content-based detection and data loss. The most recognizable change after/during a ransomware's activity is that a number of files are changed by their types or by contents. Scaife *et al.* have observed important features that could be strong indicators of ransomware [6]. Among them, the most important indicators include the following. 1) File type change which is necessarily followed by ransomware activities. 2) The similarity measure between an original file and its modified version is very low – since ransomware normally encrypts the victim file. 3) High entropy is also a strong evidence of encryption, but it is sometimes hard to be distinguished from compression that also has very high entropy. 4) A large number of file may be an indicator, although a weak one given that the deletion of many files happens usually regardless of ransomware action.

Since the content-based detection monitors a large volume of data in real time, it inevitably results in high CPU and memory overheads. Particularly, this approach involves tweaks in OS, which requires privilege escalation for the monitoring software. Considering that the detector can be infected as well, this approach is not desirable. Also, while the accuracy of content-based detection is very high owing to plenty of context information, this approach has to back up contents constantly for every writing requests to recover the already-encrypted files. This constant back up process results in a prohibitive overhead. As such, most works falling under this approach have focused on how to detect ransomware as early as possible to minimize the overhead for recovery. File type-based detection involves smaller overhead than content-based detection, but it can be easily evaded by ransomware to alter forcibly file extensions or the magic numbers for a given file type.

Application layer detection and its security. Most of the solutions against ransomware run in the application layer, thus they cannot secure the system in the worst case. First of all, in these solutions, users should be aware of the danger of ransomware, and they should install the anti-ransomware software



(a) Ransomware action vs. overwriting frequency.

(b) Ransomware and normal app's cumulative overwriting frequencies

Fig. 1: Ransomware's overwriting behavior (WannaCry, Jaff, Mole, and CryptoShield ransomwares): (a) There is a strong correlation between active period of ransomware and the number of overwritings per second (= $OWIO$), and (b) Cumulative values of overwritings show that ransomware does overwriting very frequently.

before being infected. However, a report in 2013 showed that 24% of personal computers (PCs) were unprotected by any anti-virus software [7]. Furthermore, more than 90% of devices that had anti-virus had not run a full system scan using their anti-virus within the last seven days. Even worse, anti-virus definitions were out of date in more than 15% of the PCs [8]. Also, cracked anti-virus software could get PC infected as well. Finally, killing a ransomware detector process would not be an issue to the creator of ransomware when characteristics of most commercial detector software are well known in advance.

B. Our approach: SSD-installed ransomware detector

Our approach is not to detect ransomware in the application layer or in the OS layer, but to put the detector inside an SSD. By doing so, we can solve the most crucial problems in ransomware detection: 1) Protection of users having no anti-ransomware software installed, and 2) high overhead for recovery of data loss caused by detection latency.

Challenges for detection and recovery. To realize our approach, we have to solve the following problems:

- SSD-Insider should work as a part of SSD firmware, which means that it should detect ransomware only with limited CPU power and memory. Monitoring content alteration incurs too much overhead to an SSD system as we discussed. Thus, SSD-Insider should be able to extract ransomware's behavioral characteristics by seeing only IO request headers instead of seeing the whole request data. The IO request header include only a logical block address, read/write type, and the size of data. This limited view of ransomware activity makes detection more difficult, considering that much more context and information are available to application layer detectors including file names, file sizes, file access time, process IDs, process names, and the amount of memory the process is using, etc.
- Detection latency is inevitable by any means, so some portion of disk blocks will have been already encrypted. A method to provide perfect data recovery even under detection latency should be provided.

Our approach to address the two challenging problems above is to use overwriting patterns as ransomware features for detection and to take advantage of SSD's delayed deletion, which is an intrinsic feature of NAND flash memory for recovery.

Overwriting patterns as lightweight and invariant features. To recognize ransomware activity by viewing only the distribution of IO request headers, we have paid attention to a ransomware's very unique behavior, *overwriting*. Overwriting in SSD is defined as the event of updating the same logical block address (LBA) to remove the data in the block after a block read. Fig. 1(a) shows how long WannaCry and Mole ransomwares were in action during 1-second time slices when overwriting frequency varies; it shows that the more frequent overwriting occurs, the longer WannaCry and Mole have been running. Fig. 1(b) further shows the cumulative graph of the number of overwriting requests for four ransomware (WannaCry, Jaff, Mole, and CryptoShield) and four normal applications (data wiping, P2P download, cloud storage synchronization, and compression). We notice that ransomwares have a relatively high growth rate compared to normal applications. While WannaCry and Mole stand out, Jaff and CryptoShield have a relatively low growth rate, alluding to a difficulty in distinguishing them from the normal applications. Even so, it is clear that ransomwares cannot but permanently delete victim files by overwriting them by either encrypted or other unrelated data. Also, ransomwares should overwrite as soon as possible to decrease the user's chance of recovery.

Using SSD's delayed deletion for perfect and instant recovery. To deal with the inability to perform in-place updates in NAND flash, the logical block address is separated from the physical block address in SSDs. Therefore, a storage firmware (called FTL) running inside SSDs maintains an address mapping table and remaps overwritten LBAs to free physical space. With FTL's address remapping, all the new data is appended to storage media, which enables us to hide the out-of-place update nature of NAND flash. However, since old versions of data are left in the flash, which wastes storage space, FTL's other module, the garbage collector (GC), reclaims the free space occupied by old versions. As observant readers might notice, SSDs always keep old versions of data that were overwritten by new data until they are permanently erased by GC. SSD-Insider takes advantage of the built-in backup capability of SSDs. SSD-Insider keeps track of old versions of data inside SSDs and never removes them until the ransomware detection algorithm confirms that the new versions are not affected by ransomwares. If a ransomware attack is detected, SSD-Insider can quickly recover the original data by rolling back a mapping table so that it points to the old versions. Since only an update of mapping table entries is required, the roll-back process can be completed rapidly without having to copy data physically.

III. Design of SSD-Insider

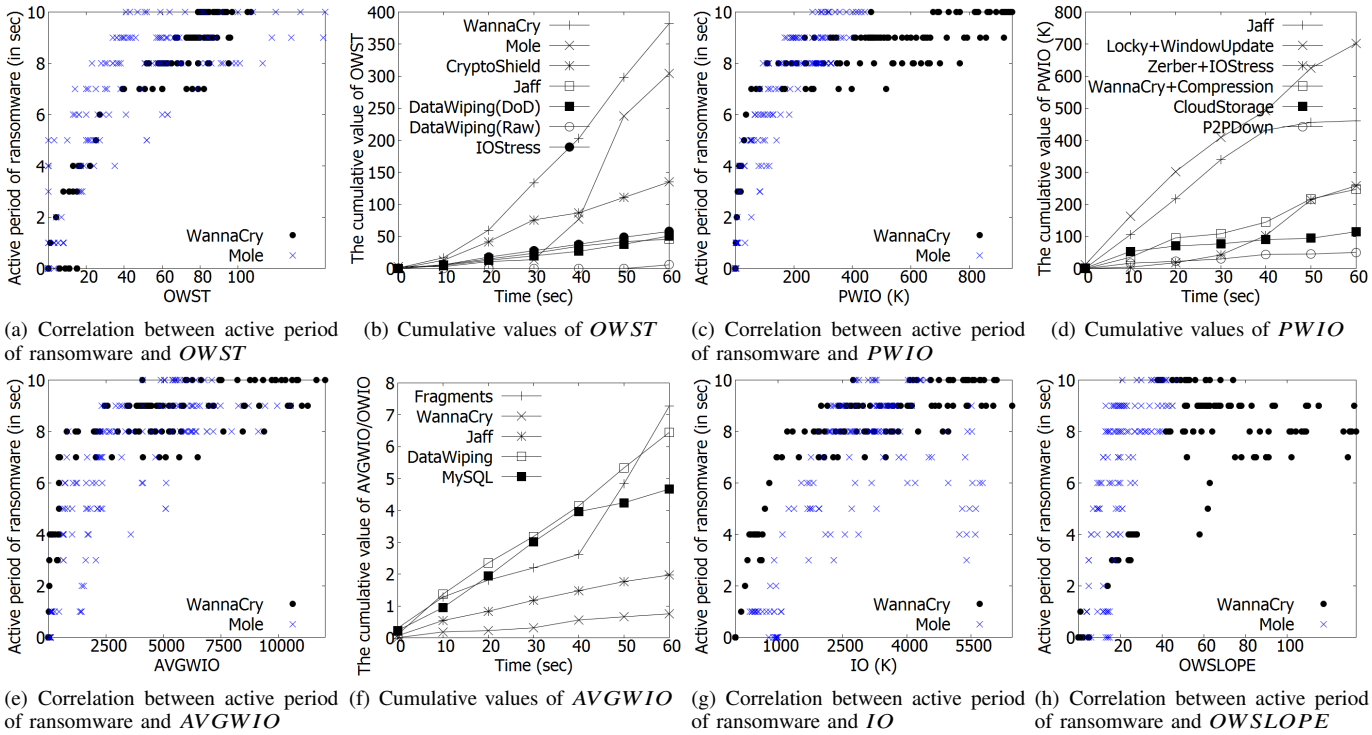
A. Invariant features of ransomware

Based on how they overwrite encrypted files, ransomware can be categorized into three classes: in-place overwriting (Class A), out-of-place overwriting (Class B), and deleting and overwriting the original file (Class C) [6]. Considering that attacker's chances to get ransom become higher when the original files cannot be recovered, it is necessary to "overwrite" the original files either by the encrypted contents or by random contents. Indeed, all of the ransomware samples we collected were found to conduct overwriting immediately after reading and encrypting the victim's files. We note that the overwriting is technically equivalent to "unrecoverable permanent erasure" conducted by a ransomware. To capture ransomware's behavioral traits, and to find features capable of distinguishing ransomware from normal applications with similar overwriting behavior, we conducted experiments that run several ransomwares and applications (data wiping, DB update, IO stress test, *etc*) in a sandbox and found the following six features; four principal features and two secondary ones[1]:

- $OWIO$ denotes the number of overwritings for a time slice (e.g., 1 sec).
- $OWST$ is the fraction of overwritten blocks over the total number of write requests during a time window.
- $PWIO$ is the number of overwritings for a time window consisting of N time slices.
- $AVGWIO$ is the average length of continuously overwritten blocks in a current time window.
- $OWSLOPE$ is the fraction of the number of overwritings during a current time slice over the average number of overwritings over the previous time window.
- IO is the fraction of the number of overwritings during a current time slice over the average number of writings over the previous time slice

OWIO. $OWIO$ is the most significant feature indicating the property of reading, encrypting and overwriting the same block of a file for a short period of time. Fig. 1(a) shows how long WannaCry and Mole ransomwares were running during 1-second windows when varying the value of $OWIO$. It shows that the more frequent overwriting occurs, the longer WannaCry and Mole have been in action. Also, the overwriting frequency of normal applications, as can be seen in Fig. 1(b), is not as high as that of ransomwares except for data wiping applications (*i.e.,* less than 100K). This supports our hypothesis that heavy overwriting follows reading operation within a short duration, when ransomwares are active. We utilize $OWIO$, the overwriting frequency as one of indicators of ransomware. This feature, however, can also be observed during normal program execution such as DB update after email synchronization (e.g., outlook), cloud storage synchronization (e.g., dropbox), OS update (MS Windows update), software installation, temporary file creation for web browsing, and during the operation of anti-virus software, data wiper, disk stress tools, etc. For example, Fig. 1(b) shows cumulative values of $OWIO$ for four ransomwares (e.g., WannaCry, Mole, CryptoShield, and Jaff) and for four normal applications (e.g., data wiping, cloud storage, compression, P2P download). The accumulated number of a data wiping program is as high as that of ransomwares as shown in Fig. 1(b), and that of CryptoShield is as low as that of cloud storage update and P2P download. Therefore, more

[1]Throughout this paper, overwriting is limited to overwritten data blocks that have been read within the last N seconds (time window), where N is a parameter adjusted based on the desired detection speed and accuracy.



(a) Correlation between active period of ransomware and $OWST$

(b) Cumulative values of $OWST$

(c) Correlation between active period of ransomware and $PWIO$

(d) Cumulative values of $PWIO$

(e) Correlation between active period of ransomware and $AVGWIO$

(f) Cumulative values of $AVGWIO$

(g) Correlation between active period of ransomware and IO

(h) Correlation between active period of ransomware and $OWSLOPE$

Fig. 2: Six ransomware features capture ransomware's behavioral characteristics (See also Fig. 1 for $OWIO$).

features distinguishing ransomwares from these applications are necessary to detection precisely.

OWST. One of the hard-to-distinguish applications is data wiping as we have seen in Fig. 1(b). The noticeable feature is how many overwritings there are among writing requests within a time window, where duplicate overwritings over a single block are counted only once. Typical data wiping applications require multiple overwritings over a single block to securely erase data, which incurs low $OWST$ value compared to that of a ransomware. For example, the DoD 5220.22-M [9] requires 7 overwritings per one read IO over the same block. Fig. 2(b) confirms that $OWST$ captures the high rate of overwriting in write IOs that occur during ransomware operation, where the strong correlation between $OWST$ and the active period of ransomware is clearly shown in Fig. 2(a).

PWIO. Sometimes, CPU-intensive or IO-intensive jobs might be running while a ransomware is in operation. In this case, the speed of ransomware slows down, thus the IO requests of ransomware are dispersed over rather a long time span. For example, the ransomware Jaff is too slow to be detected by $OWIO$ and $OWST$. However, $PWIO$, the accumulated number of overwritings during a longer-term window (10s) instead of a short-term slice (1s) of $OWIO$ captures it very well as shown in Fig. 2(d), whereas its correlation shown to be strong with the ransomware activity period as shown in Fig. 2(c).

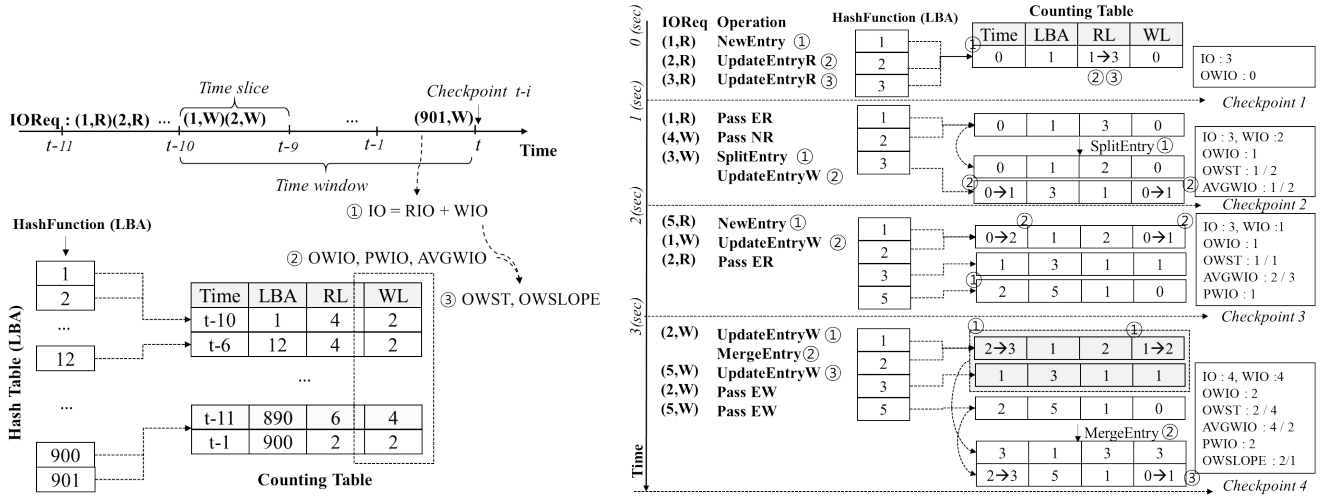
AVGWIO. $AVGWIO$ captures the run-length characteristics of ransomware's attack target. Because the ransomware mainly targets documents and images, it does not involve overwriting operations over many continuous blocks, unlike data wiping, defragmentation, and DB updates. Thus, the length of continuously overwritten blocks is relatively shorter than that of those applications, which is shown in Fig. 2(f), where its correlation with the active period is shown in Fig. 2(e).

Secondary features. The four aforementioned features are the principal features of SSD-Insider. The other auxiliary features are IO and $OWSLOPE$, where $OWSLOPE$ captures ransomware's behavior of abrupt increase of overwriting volume. Their correlations with ransomware's active period are shown in Fig. 2(g) and Fig. 2(h).

Owing to the resource limitation and the tight time-bound characteristics of the SSD system, we utilized a binary decision tree, instead of using more powerful machine learning algorithms, such as support vector machine or even deep learning. For the training algorithm of the tree, we used the ID3 algorithm [4].

B. Ransomware detecting algorithm

Data structure for detection. All I/O requests are monitored for ransomware detection, and each request consists of four items: Time, LBA, IOMode, and Length. Time denotes when the request was generated in the system. LBA is the starting address where the data is read or written, IOMode represents the request type (R/W), and Length is the number of blocks for the request. A request is denoted by IOReq, and Length is assumed to be 1. A *time window* is defined over I/O requests as a duration of monitoring to detects periodically suspicious behavior of ransomware, and is used as 10s in our experiments. It consists of N time slices, and it slides by a time slice (e.g., 1s) at every check point. To evaluate values of the six features, we build a counting table that basically stores IOReq's run-length of overwriting. During a time slice, SSD-Insider counts IOReq, and updates the counting table according to



(a) Counting table holds the run-length of overwritings for each time slice.

(b) Counting table update example using basic functions (NewEntry, UpdateEntryR, SplitEntry, UpdateEntryW, and MergeEntry)

Fig. 3: Design of SSD-Insider: data structure and working examples

Algorithm 1 RansomwareDetection

Require: N
1: **for** all req_i **do**
2: **if** the time slice expires **then**
3: Calculate 6 attributes for N time slices
4: $ransom_t$=DecisionTree$_{ID3}$(6 attributes)
5: $Score = Score + ransom_t$
6: Slide TimeWindow by one time slice
7: $Score = Score - ransom_{t-10}$
8: **end if**
9: **if** req_i is write-req **then**
10: Count overwritten blocks if req_i is already in the table
11: Merge or split the runs of overwritten blocks
12: **else**
13: Make a new entry and adjust the runs of overwritten blocks
14: **end if**
15: **end for**

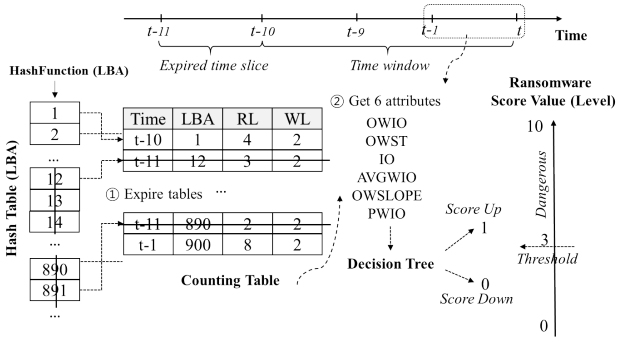


Fig. 4: Sliding time window and detecting ransomware activity

the counting value and LBA. The counting table consists of entries that store each consecutive overwriting. An entry is composed of Time, LBA, RL, and WL. Time denotes the time slice number at which the entry is created or updated. LBA is the starting address of a consecutive overwriting. RL is the total length of Read IO that occurs consecutively from the LBA. WL is the total length of write IO, i.e., consecutive overwriting, occurring after a read IO has occurred. A hash table consisting of LBAs for keys is defined for fast access of the counting table. Since an entry in the counting table stores a consecutive IOReq, multiple LBAs can be merged into an entry. Using the counting and the hash tables, we can calculate values of the six features. That is, IO is the sum of all read (RIO) and write IOs (WIO) during the current time slice (① in Fig. 3(a)). $OWIO$ is the sum of WLs for the current time slice. $PWIO$ is the sum of all WLs stored in the counting table from $t-11$ to $t-1$ (assuming $N = 10$, as in our experiments), when the current time is t. $AVGWIO$ is the average WL of all entries from $t-10$ to t in the counting table as in ② of Fig. 3(a). $OWST$ is the current time slice's $OWIO$ divided by WIO. Finally, $OWSLOPE$ is the value of $OWIO$ divided by $PWIO$ as shown in ③ of Fig. 3(a).

SSD-Insider's real-time detection. The detection algorithm is described in Algorithm 1. Using an example, how to keep track of run-length of overwriting during each time slice and window is described in Fig. 3(b). IOReqs used in this example are assumed to have (IOMode, LBA) and the Length is used as 1. Roughly speaking, the algorithm updates the tables according to the IO request type. When it is a write request, it checks whether it is an overwriting command or not and counts (Line 10). Also, because it is collecting the run-length of overwritten blocks, it manages the runs of overwritten blocks (Line 11). For a read request, it makes a new entry in the table or just update it to make a longer run (Line 13). When a time slice expires, it drops the obsolete entries in the counting table by sliding the window (Line 6), and adjust $Score$ by subtracting the dropped entry (Line 7). Six feature values are calculated using the counting table (Line 3), and the values are fed to the ID3 decision tree (DecisionTree$_{ID3}$) to obtain 0 or 1 results (Line 4), where 1 means that the system at the current checkpoint is highly likely to be under attack of a ransomware, and 0 means otherwise. For a time window (10s), the outputs of the decision tree are all added up to give a score ranging

from 0 to 10 as shown in Fig. 4 (Line 5). To determine whether ransomware is active or not, we used a threshold value 3 in our experiment (See section V-B).

C. Instant recovery algorithm

In order to support the recovery of damaged files, SSD-Insider's recovery algorithm should keep track of all the changes made to files, maintaining original versions. It also should retain data persistence/consistency and should not compromise I/O performance. In addition, SSD-Insider must be designed in a cost-effective manner so that it can be implemented in SSD environments. The proposed recovery algorithm is devised to satisfy all of the aforementioned requirements. In the following subsections, we explain how the SSD-Insider's FTL handles I/O requests while leaving change logs for recovery and detail its recovery process. We also discuss issues related to GC.

Data recovery process. Once the data recovery process is triggered by the detection algorithm, the SSD-Insider notifies users of which suspicious behaviors are detected [2]. If a user responds that ransomware attack is suspected, SSD-Insider's FTL first makes an SSD *read-only*, ignoring all the writes sent to it. As illustrated in Fig. 5, the SSD-Insider's FTL then scans backup entries in the queue from the back to the front, replacing individual mapping entries with the corresponding backup entries. Backup entries staying in the queue for longer than 10 seconds are ignored during the recovery process. After the recovery process finishes, the status of the mapping table is rolled back to the time just before 10 seconds. This roll-back process is accomplished within one second because it does not involve any data copies, but just updating the mapping table. After the storage is recovered, SSD-Insider asks users to reboot the system and to get rid of ransomwares using anti-virus programs. The recovery algorithm aims at restoring the status of the SSD to 10 seconds earlier, but this process is done without any awareness of data consistency between files and their metadata (e.g., inodes). Thus, a recovered SSD could have an inconsistent status, where on-disk file-system structures and files are partially updated and thus inconsistent. This consistency problem can be resolved by means of a file-system check/recovery tool (e.g., `fsck`). `fsck` is designed to restore a consistent file system after sudden power loss or a system failure. The SSD status after SSD-Insider's recovery is similar to one after sudden power loss or system failures. Only the differences are: 1) it is intentionally caused by SSD firmware and 2) it looks like that a power failure happened 10 seconds before ransomware attacks. A series of experiments on EXT4 show that file-system is successfully recovered to a consistent state with `fsck` (see Section V-B).

Garbage collection. Garbage collection (GC) is the only way of permanently erasing actual contents in flash. Old versions of data stored in flash pages should not be erased by GC until their new versions are confirmed safe. For this reason, the SSD-Insider's FTL has to copy even invalid pages during GC if it is unsure if they are encrypted by ransomware or not. Compared with typical FTLs, therefore, SSD-Insider would require more page copies for GC. However, since only a limited number of pages stay in the queue, extra page copies caused by SSD-Insider are trivial. According to our experiments, SSD-Insider exhibits almost 0% and 22% higher GC overheads than existing FTLs under the average- and the worst-case scenarios, respectively (see Section 5).

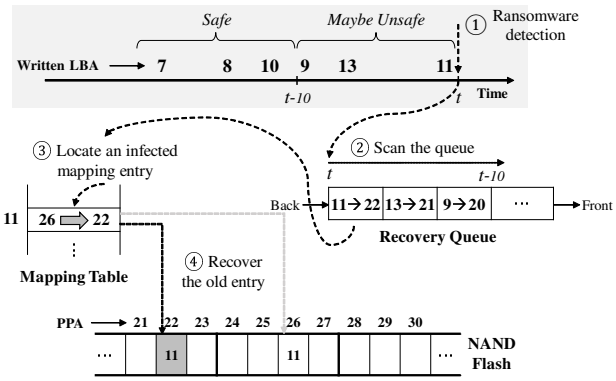


Fig. 5: Data recovery process in SSD-Insider. After ransomware is detected (①) at time t, the SSD-Insider FTL scans the recovery queue (②), examining LBAs, old PBAs, and timestamps. For LBAs 9, 13, and 11 that were recently overwritten (i.e., $> t - 10$), the SSD-Insider cannot guarantee their safety. Therefore, SSD-Insider locates a mapping entry in the mapping table (③) and updates the entry so that it points to the physical location of the original data (④). In case of LBA 11, its mapping entry is reverted from PPA 26 to PPA 22 that contains old but safe data. For LBAs 7, 8, and 10 that were written 10 seconds ago (i.e., $\leq t - 10$), SSD-Insider guarantees that their data are safe, and thus their mapping entries are not updated.

IV. Implementation of SSD-Insider

We implemented SSD-Insider's detection and recovery algorithms in an in-house open-channel SSD[3] prototype illustrated in Fig. 6. Our in-house SSD card was composed of 8 channels and 8 ways with a custom flash controller. It offered a capacity of 512 GB with 700 MB/s write and 1.2 GB/s read throughputs, which were comparable to commercial SSDs. To evaluate SSD-Insider in terms of performance and overhead, we used a x86 host with Intel's Xeon CPU running at 3.0 GHz and 4 GB DRAM. Our SSD card was connected to the x86 host through a PCIe interface. Ubuntu 16.04 with the Linux kernel 4.10 was used as a host OS.

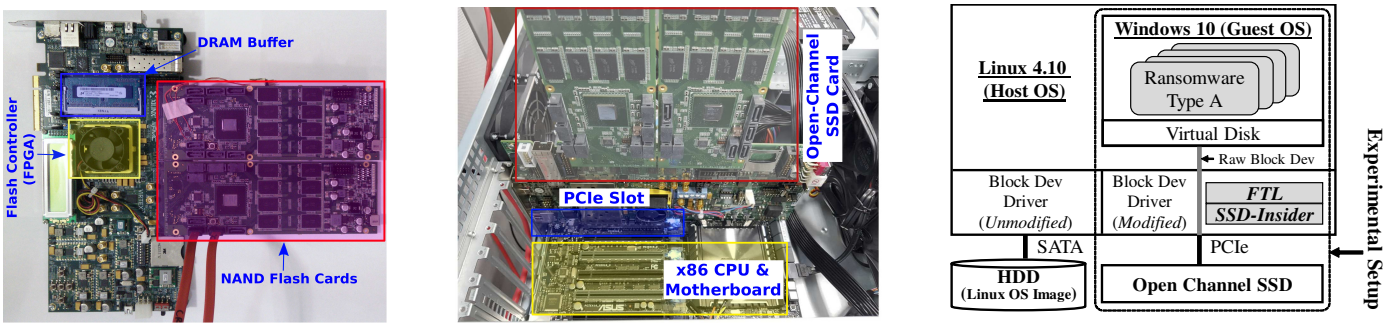
V. Experimental Results

A. Ransomware data sets

For our evaluation, we collected various well-known ransomwares such as Locky.bdf, Locky.bbs, Zerber.ufb, WannaCry, Jaff, Mole, GlobeImposter, and CryptoShield [12].

[2]This notification to end users can be made by using a customizable I/O interface facility with an integrated user application. The modern storage interface standards provide a way of adding user-defined commands so that the host and the storage device exchanges maintenance information [10], [11]. In our case, a 'ransomware attack alarm' can be added as a new command. Once the host receives an alarm from the SSD, it launches the application that displays a warning message and gets a response from users to decide whether a ransomware attack is suspected or not.

[3]Unlike off-the-shelf SSDs, the open-channel SSD platform enables us to implement and test various FTL algorithms because all those algorithms are run on the block device layer of the host system [11]. Even if the FTL algorithms, along with the detection/recovery algorithms, are implemented in the block layer, they could be directly adopted to the SSD firmware because of no differences in the design principle.



(a) Open-channel SSD card (b) The SSD card attached to a x86 host via PCIe (c) A diagram of our experimental setup

Fig. 6: SSD-Insider's working prototype and experimental setup

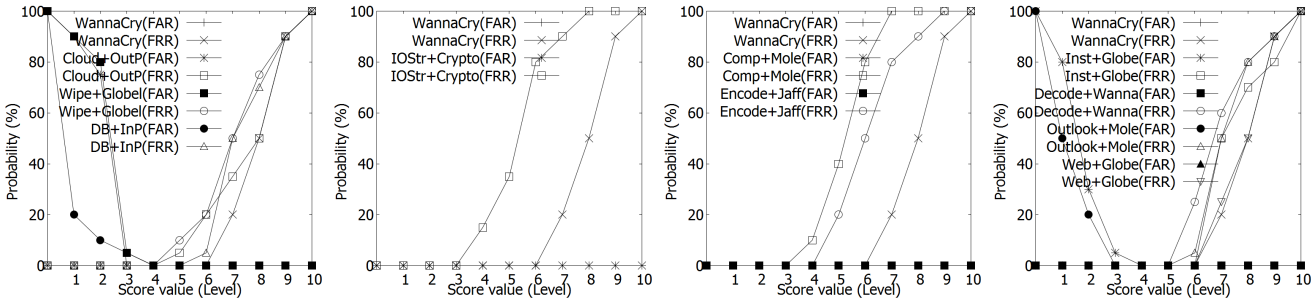
TABLE I: Data set with various combinations of ransomwares and applications for training and testing of SSD-Insider

Application Type	Application	Ransomware
For training		
Ransom only	none	Locky.bbs
Heavy overwriting	WPM (DataWiping)	none
Heavy overwriting	MySQL (Database)	none
Heavy overwriting	Dropbox (CloudStorage)	none
IO-intensive	DiskMark (IOStress)	Zerber.ufb
IO-intensive	IOMeter (IOStress)	Zerber.ufb
IO-intensive	hdtunepro (IOStress)	Zerber.ufb
Normal App	AutoCAD/VS (Install)	Locky.bdf
Normal App	Chrome (WebSurfing)	Locky.bbs
Normal App	OutlookSync	Locky.bdf
Normal App	WindowUpdate	Locky.bdf
Normal App	BitTorrent (P2PDown)	none
Normal App	Kakaotalk (SQLite)	none
For testing		
Ransom only	none	WannaCry
Heavy overwriting	Dropbox (CloudStorage)	In-house (outplace)
Heavy overwriting	WPM (DataWiping)	GlobeImposter
Heavy overwriting	MySQL (Database)	In-house (inplace)
IO-intensive	IOMeter (IOStress)	CryptoShield
CPU-intensive	Bandizip (Compression)	Mole
CPU-intensive	PotEncoder (VideoEncode)	Jaff
Normal App	AutoCAD/VS(Install)	GlobeImposter
Normal App	PotPlayer (VideoDecode)	WannaCry
Normal App	OutlookSync	Mole
Normal App	BitTorrent (P2PDown)	WannaCry
Normal App	Chrome (WebSurfing)	GlobeImposter

As well as those, we have implemented two new and in-house ransomwares that are not reported to the public using open source ransomwares from github [13], [14] – one of them has an 'in-place update' encryption attack while the other is based on an 'out-of-place update'. To evaluate SSD-Insider's performance, we set up the environment where ransomware ran with typical background applications, including three application types: applications with high rate of overwriting, CPU-intensive, and IO-intensive applications. While heavy overwriting applications confuse our detector, CPU/IO-intensive applications disturb the detector by slowing down ransomware's activity. Even the normal application class included applications with relatively high IO compared to user's casual usage pattern. The background applications were divided into four types: 1) Heavy overwriting type includes data wiping tool (WPM satisfying DoD 5220.22-M), cloud storage synchronization (Dropbox), and heavy database update (MySQL 5.5), 2) IO-intensive type includes IO stress tools such as IOMeter, HDtunepro, DiskMark, 3) CPU-intensive type includes compression (Bandizip), video encoding (Daum pot encoder), 4) Normal app type includes video playback (Daum pot player), email synchronization (Outlook), P2P download (BitTorrent), web-browsing (Chrome), SQLite activities (Kakaotalk), and software installation such as VS (Viusal Studio 2015) and AutoCAD 2016. Various combinations of a ransomware and a background application were used for learning decision trees with ID3, and also for performance evaluation. The whole dataset and evaluation scenarios for the experiment are summarized in Table I. For evaluation, experiment with each combination was conducted 20 times, and the average was taken. We also note that as shown in the table no ransomware for training is included for testing to show how accurately and promptly SSD-Insider detects unknown ransomwares.

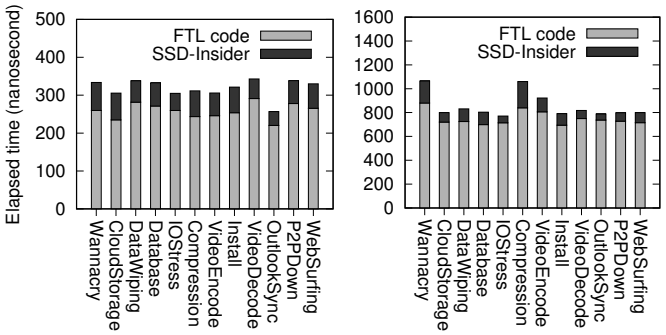
B. Performance evaluation

Ransomware detection accuracy. To evaluate the accuracy of SSD-Insider detection algorithms, we measured FAR (false acceptance rate) and FRR (false rejection rate) of the detection algorithm varying the threshold of the score value (or the frequency of which the decision tree reports ransomware activity for a time window) to detect ransomware. Fig. 7 summarizes our experimental results on accuracy in terms of FAR/FRR. Even though there is some variation, depending on the scenario, with the threshold with 3 the detection algorithm could detect ransomware's activity accurately under all types of background applications. We report that the worst background noise in terms of FRR came from IO-intensive and CPU-intensive jobs as shown in Fig. 7(b) and Fig. 7(c). That is, they interfered with ransomware to slow down the speed of overwriting by heavy usage of CPU and IO. In terms of FAR, the worst scenario came mostly from heavy overwriting type, such as DataWiping and Database applications as shown in Fig. 7(b). This is because heavy overwriting applications confused our detector by their similarity in behavior. However, as can be seen in Fig. 7, even in such heavy CPU, IO usage and heavy overwriting scenarios, SSD-Insider detected very accurately ransomware activity with the threshold value 3. FRRs in all scenarios were 0%, and FARs were close to 0% with the threshold value 3, which means that SSD-Insider did not miss any ransomware activity, but sometimes falsely recognized normal application's activity as that of ransomware. Before recovery process starts, SSD-Insider prompts users to confirm whether she will start the recovery process or not. False alarm might interrupt users, but it would rarely happen



(a) Heavy overwriting (b) IO-intensive (c) CPU-intensive (d) Normal App

Fig. 7: SSD-Insider's detection accuracy varying the score during a time window (10s) When the threshold score (or, the threshold frequency of the decision tree's reporting of ransomware activity for a time window) is set to 3, FRR is 0% and FAR is at most 5% only when heavy overwriting such as data wiping.



(a) Read elapsed time (b) Write elapsed time

Fig. 8: Impact of SSD-Insider on I/O elapsed times for ransomware-infected SSDs with various background applications running

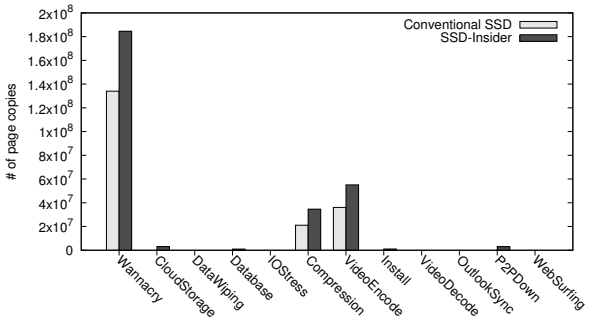


Fig. 9: A comparison of GC costs

only with heavy DB update, data wiping, and software install even at most FAR 5% as shown in Fig. 7.

Data consistency. To evaluate data consistency, we intentionally exposed the host to ransomware attacks repeatedly 100 times using a custom ransomware we developed. It mimicked the common behaviors of well-known ransomwares and infected larger than 1 GB files at an arbitrary point of time. Once SSD-Insider detected the activities of the ransomware, it stopped servicing writes from the host, asked the user if it would recover the files, recovered infected files if the answer was yes, and asked users to reboot the host. After the reboot, a `fsck` tool was triggered to find and resolve data inconsistency. After `fsck` finished, we saw if the file system was still in the consistent state. We also checked if all the infected files were rolled back to unencrypted versions. Table II summarizes the results, confirming that infected SSDs successfully returned to a consistent status with no encrypted files left.

TABLE II: A summary of file-system consistency checks

Type of corruption	# of occurrences	Corruption not resolved	Files left encrypted
No corruption	0	-	-
Wrong free-block count	100	×	×
Wrong inode-block count	100	×	×
Free-space bitmap	61	×	×

C. Overhead evaluation

I/O elapsed time. For the 12 testing traces in Table I, we measured read/write elapsed times increased by SSD-Insider. Fig. 8 shows the length of time spent by 1) the FTL codes and by 2) the SSD-Insider detection/recovery algorithms, excluding NAND device latency. To confirm the feasibility of SSD-Insider on embedded processors, we intentionally slowed down the host CPU clock from 3 GHz to 1.2 GHz on which SSD-Insider runs. Compared with the conventional FTL, read and write latencies with SSD-Insider increased by 23.5% and 15.6%, on average. More specifically, the elapsed times taken for the FTL (without SSD-Insider) to handle 4-KB block reads and writes were 477 *n*s and 1,372 *n*s, respectively. The extra overheads added by the SSD-Insider detection/recovery algorithms were just 147 *n*s and 254 *n*s, respectively. Considering that NAND chip's page-read and page-write latency is 50 μs and 500 μs [15], these extra delays account for a negligible portion of the total I/O latency – 0.3% for reads and 0.04% for writes – and do not affect the user-perceived response times.

Garbage collection cost. SSD-Insider recovery algorithm has to copy invalid pages to keep old versions of data during GC for data recovery later. Moreover, since those old pages occupy additional space, it would lead to more frequent GC invocations, which negatively affects both I/O performance and storage lifetime. In order to assess SSD-Insider under the worst-case scenario where GC costs are high, we filled up 90% of the SSD with user files and ran traces on it. As depicted in Fig. 9, even under the worst-case scenario, the SSD-Insider's FTL required 22% more page copies, on average [4]. For the traces except for Compression, VideoEncode, and WannaCry, only a few page copies were observed. This was because they

[4]The FTL algorithm used was based on page-level mapping with greedy victim selection, which was almost the same as an open-source FTL implementation in the Linux kernel [11]. Even though detailed architectures of commercial SSDs are unknown, literature reports that proprietary FTLs are based on similar schemes [16]. Hence, it may be safe that our experimental setup at least suffices to understand the impact of SSD-Insider GC costs.

TABLE III: DRAM requirements for SSD-Insider

Data structure	Unit size	# of entries	DRAM size (MB)
Hash table	42 Bytes	250,000	10 MB
Counting table	12 Bytes	1,000	0.03MB
Recovery queue	12 Bytes	2,621,440	30 MB

did not require many page copies for cleaning. We also carried out experiments with 70% SSD space utilization to understand GC overheads under the average case, and found that extra page copies caused by SSD-Insider were almost zero. This indicates that the effect of SSD-Insider on the increase of GC costs is negligible.

DRAM requirement. For the ransomware detection and recovery, SSD-Insider has to maintain additional data structures, such as a hash table, a counting table, and a recovery queue. TABLE III lists DRAM required by SSD-Insider. The total size of DRAM needed to run SSD-Insider is 40.03 MB, which is an affordable size for modern SSDs typically equipped with more than 1 GB DRAM [17], [18], [19].

VI. Related Work

For detection of ransomwares, signature matching is the most popular tool used by many anti-virus software. However, an attacker can easily create ransomware variants with different signatures to evade detection. Instead of checking ransomware signatures, a file integrity monitor can detect ransomware activities. Tripwire is a system file monitoring tool, doing hash comparison for checking illegal modification, but it is not appropriate for user files which might change frequently [20]. Drastic changes of a user file's content in a short time period may be a strong indicator for ransomware activities. Also, encrypted files have very high entropy, but ordinary files not. CryptoDrop and Unveil monitors in real-time user's data for changes [6], [21]. They use file type changes, drastic changes in file contents, and Shannon entropy as primary indicators to distinguish ransomware's activity. It also uses deletion and file type funneling of a large number of files as secondary indicators. CryptoDrop effectively detected ransomware's activity, but some of the encrypted files could not be recovered. To detect early before a ransomware encrypts any files, Moore proposed to use honeypot [22]. Orman described a monitoring technique, which observes in real-time repetitive loops that both symmetric and public key algorithms such as AES and RSA should have necessarily [23]. However, it should be noted that monitoring calls to encryption libraries is not enough, because encryption can be easily implemented using prevailed open source software. Canfora *et al.* developed detection schemes of ransomware's abnormal behavior using Hidden Markov Model (HMM) and structural entropy [24]. Cabaj *et al.* utilized Software-Defined Network (SDN) to detect and mitigate ransomware [25]. Continella *et al.* proposed a file-system level solution, called ShieldFS, for ransomware detection and recovery [26]. To detect ransomware, ShiledFS monitored various activities and information available at the file system level.

Numerous techniques have been studied to defense hacker attacks at the level of storage systems [27], [28], [29], [30]. Pennington *et al.* proposed a storage-based intrusion detection technique that monitored suspicious activities inside file servers or block store and prevented malware programs (e.g., backdoors and Trojan horses) from being inserted by attackers [30]. It used rule-based policies that triggered alarms when suspicious I/O behaviors were detected. Paul *et al.* extended an idea of the above work so that the behavior-based intrusion detection was done inside storage device [27]. Our approach was directly inspired by the ideas of the aforementioned works, but we focused on detecting ransomware behaviors rather than typical malware (e.g., virus programs). Moreover, we proposed the overhead-free recovery scheme that took advantage of the physical natures of NAND flash. Park *et al.* have presented a brief concept of self-defensible SSDs, claiming that the backup capability of SSDs would be effective to protect user data from ransomware attacks [31]. The basic idea of self-defensible SSDs is similar to that of SSD-Insider, but it presents only idea-sketch but did not explore the ransomware features. Huang *et al.* proposed FalshGuard that has a firmware-level recovery system, but it focused only on recovery process but not on detection algorithm [32]

VII. Conclusion

In this paper, we have proposed a new set of behavioral features of ransomware, which is invariant across various ransomwares and lightweight enough to be used in SSDs. By presenting various experimental results, the discovered features have been shown to be strong indicators of ransomware activity. We also have developed a perfect recovery algorithm of infected files using the delayed deletion feature of SSD. To show the feasibility of our techniques, the detection/recovery algorithms were implemented in an open-channel SSD as a working prototype called SSD-Insider.Our evaluation results showed that SSD-Insider was accurate and fast for detection, and it could perfectly recover an infected SSD in a second without any data loss. We believe that SSD-Insider opens a new direction of anti-ransomware research – better defense and recovery algorithms on other malicious software would be developed based on the similar concept of SSD-Insider, which are left for our future work.

Acknowledgement. This research was supported by Global Research Lab. (GRL) Program of the National Research Foundation (NRF) funded by Ministry of Science, ICT (Information and Communication Technologies) and Future Planning(NRF-2016K1A1A2912757). This work was supported in part by the Basic Science Research Program through the National Research Foundation of Korea funded by the Ministry of Education under Grant NRF-2016R1C1B2011415, and in part by NRF grants NRF-2016K1A1A2912757, NRF-2017R1E1A1A01077410, and NSF grant CNS-1809000. Dae-Hun Nyang and Sungjin Lee are the corresponding authors.

References

[1] "Korean web host hands over 1 billion won to ransomware crooks," 2017. [Online]. Available: http://www.zdnet.com/article/korean-web-host-hands-over-1-billion-won-to-\\ransomware-crooks/

[2] C. Everette, "Ransomware: to pay or not to pay?" *Computer Fraud & Security*, vol. 2016, no. 4, pp. 8–12, 2016.

[3] "An istr special report: Ransomware and businesses 2016," 2016. [Online]. Available: https://www.symantec.com/content/en/us/enterprise/media/security_response/whitepapers/ISTR2016_Ransomware_and_Businesses.pdf

[4] J. R. Quinlan, "Induction of decision trees," *Mach. Learn.*, vol. 1, no. 1, pp. 81–106, 1986.

[5] J. Walter, *Meet tox: Ransomware for the rest of us*. mcafee-labs/meet-tox-ransomware-for-the-rest-of-us/, 2015. [Online]. Available: https://blogs.mcafee.com/

[6] N. Scaife, H. Carter, P. Traynor, and K. Butler, "Cryptolock (and drop it): Stopping ransomware attacks on user data," in *Proceedings of Distributed Computing Systems (ICDCS)*. 2016 IEEE International Conference on, 2016, pp. 303–312.

[7] "Latest security intelligence report shows 24 percent of pcs are unprotected," 2013. [Online]. Available: https://blogs.microsoft.com/blog/2013/04/17/latest-security-intelligence-report-shows-24-\\percent-of-pcs-are-unprotected/

[8] "Opswat: Antivirus and compromised device report," 2015. [Online]. Available: https://www.opswat.com/resources/reports/antivirus-and-compromised-device-january-2015

[9] *National Industrial Security Program Operating Manual (NISPOM)*, Department of Defense, 2006.

[10] *Seagate SMART Attribute Specification*, Seagate, 2011.

[11] M. Bjørling, J. Gonzalez, and P. Bonnet, "Lightnvm: The linux open-channel SSD subsystem," in *USENIX Conference on File and Storage Technologies (FAST 17)*, 2017, pp. 359–374.

[12] "Virus total." [Online]. Available: https://www.virustotal.com/

[13] "Virtual gangster." [Online]. Available: https://github.com/roothaxor/Ransom

[14] "A poc windows crypt." [Online]. Available: https://github.com/mauri870/ransomware

[15] *MT29F8G08AAAWP NAND Flash Memory Specification*, Micron Technology, Inc., 2012.

[16] F. Chen, D. A. Koufaty, and X. Zhang, "Understanding intrinsic characteristics and system implications of flash memory based solid state drives," in *Proceedings of the International Joint Conference on Measurement and Modeling of Computer Systems*, 2009, pp. 181–192.

[17] *Hitachi accelerated flash*, Hitachi, 2015.

[18] *Samsung SSD 840 EVO data sheet, rev. 1.1*, Samsung, 2013.

[19] *PS3110 controller*, PHISON, 2014.

[20] G. H. Kim and E. H. Spafford, "The design and implementation of tripwire: A file system integrity checker," in *Proceedings of the ACM Conference on Computer and Communications Security*, 1994.

[21] A. Kharaz, S. Arshad, C. Mulliner, W. Robertson, and E. Kirda, "Unveil: A large-scale, automated approach to detecting ransomware," in *Proceedings of the 8th ACM International Conference on Embedded Software*, ser. USENIX Security '16. USENIX, 2016, pp. 757–772.

[22] C. Moore, "Detecting ransomware with honeypot techniques," in *Proceedings of Cybersecurity and Cyberforensics Conference*, 2016.

[23] H. Orman, "Evil offspring – ransomware and crypto technology," *IEEE Internet Computing*, vol. 20, no. 5, pp. 89–94, October 2016.

[24] G. Canfora, F. Mercaldo, and C. A. Visaggio, "An hmm and structural entropy based detector for android malware: An empirical study," *Computers & Security*, vol. 61, pp. 1–18, 2016.

[25] K. Cabaj and W. Mazurczyk, "Using software-defined networking for ransomware mitigation: The case of cryptowall," *IEEE Network*, vol. 30, no. 6, pp. 14–20, Dec 2016.

[26] A. Continella, A. Guagnelli, G. Zingaro, G. De Pasquale, A. Barenghi, S. Zanero, and F. Maggi, "Shieldfs: A self-healing, ransomware-aware filesystem," in *Proceedings of the Annual Conference on Computer Security Applications*, 2016, pp. 336–347.

[27] A. G. Pennington, J. D. Strunk, J. L. Griffin, C. A. N. Soules, G. R. Goodson, and G. R. Ganger, "Storage-based intrusion detection: Watching storage activity for suspicious behavior," in *Proceedings of the Conference on USENIX Security Symposium*, 2003, pp. 10–10.

[28] G. A. Gibson, D. F. Nagle, K. Amiri, F. W. Chang, E. Feinberg, H. Gobioff, C. Lee, B. Ozceri, E. Riedel, and D. Rochberg, "A case for network-attached secure disks," 1996.

[29] K. R. Butler, S. McLaughlin, and P. D. McDaniel, "Rootkit-resistant disks," in *Proceedings of the ACM Conference on Computer and Communications Security*, 2008, pp. 403–416.

[30] N. R. Paul, "Disk-level behavioral malware detection," Ph.D. dissertation, 2008.

[31] J.-Y. Paik, K. Shin, and E.-S. Cho, "Poster: Self-defensible storage devices based on flash memory against ransomware," in *Proceedings of IEEE Symposium on Security and Privacy*, 2016.

[32] J. Huang, J. Xu, and X. Xing, "Flashguard: Leveraging intrinsic flash properties to defend against encryption ransomware," in *Proceedings of ACM Conference on Computer and Communications Security (CCS)*, 2017.