SSD-Assisted Ransomware Detection and Data Recovery Techniques

Sungha Baek, Youngdon Jung, David Mohaisen, *Senior Member, IEEE*, Sungjin Lee, and DaeHun Nyang, *Member, IEEE*

Abstract—As ransomware attacks have been prevalent, it becomes crucial to make anti-ransomware solutions that defend against ransomwares. In this article, we propose a new ransomware defense system, called *SSD-Insider++*, which prevents users' files from being damaged by ransomware attacks. SSD-Insider++ is embedded into an SSD controller as a form of firmware. By being separated from a host machine, it not only provides more robust data protection than software-based ones which are vulnerable to evasion attacks, but also offers interoperability with various platforms. SSD-Insider++ is composed of two novel features, ransomware detection and perfect data recovery, which are tightly integrated with each other. The detection algorithm observes I/O patterns of a host system and decides whether the host is being attacked by ransomwares in an early stage. Once an encryption attack is detected, the recovery algorithm is triggered to recover original files by leveraging a delayed deletion feature of an SSD at a low cost. Our experimental results show that SSD-Insider++ achieves high accuracy of detecting ransomwares with 0 percent FRR/FAR in most cases and provides an instant data recovery with 0 percent data loss. The overhead of running SSD-Insider++ is negligible – only 80 ns and 226 ns are spent more for handling 4-KB reads and writes, respectively.

Index Terms—Ransomware, malware detection, data recovery, flash-based SSDs

1 Introduction

Ransomware is a type of malicious software that perpetually blocks access to the victim's files. To extort a ransom from data owners, a ransomware encrypts users' files using encryption algorithms and releases decryption keys only after a ransom is paid. To avoid existing malware defense systems, a ransomware uses complex command and control networks like Tor, and ransom payments are made through cryptocurrency which is actually impossible to track. The high financial gains and the difficulty of defending against a ransomware make it a "profitable business" to cybercriminals [1].

There have been several attempts to defend against ransomware attacks. The most common solution widely used today is installing vaccine software in computers which identifies ransomwares using signature-based detection [2], [3]. However, owing to its inability of detecting new variants with unknown signatures, a backup-based solution, which stores old copies of files in backup storage, recently receives serious attention [4]. The main drawback of the backup-based solution is high I/O overhead for data backup. To mitigate this, a hardware- or SSD-assisted backup solution has recently been introduced [5], [6]. By offloading backup operations to an SSD, it minimizes extra I/Os involved with data backup. This offloading, however, leads to the accumulation of large amounts of old data in an SSD, which results in serious garbage collection (GC) overheads.

In this paper, we propose a novel SSD-assisted ransomware defense system, called *SSD-Insider++*. Similar to other SSD-assisted solutions, SSD-Insider++ is designed to be part of storage firmware executing in an SSD controller. However, by putting intelligent features, (1) online ransomware detection, (2) perfect data recovery, and (3) lazy detection, into the storage side, SSD-Insider++ overcomes the limitations of the previously proposed techniques.

The online detection algorithm of SSD-Insider++ is fundamentally different from signature-based ones. It monitors and analyzes I/O patterns of a host machine and decides whether or not the host is being attacked by ransomware programs. This decision is made at run time by observing invariant features that characterize unique I/O behaviors of ransomware-infected host machines. This makes it possible to detect a ransomware attack at its early stage.

For the ransomware detection, SSD-Insider++ inevitably allows ransomwares to encrypt files. Even worse, there is a possibility that it would fail to detect ransomware activity. To compensate this, SSD-Insider++ employs instant backup/recovery and lazy detection algorithms. For quick data backup/recovery, SSD-Insider++ does not create any copies of data; instead, it leverages the operational characteristics of an SSD that keeps old versions of data to hide the out-of-place update nature of NAND flash. This enables us to back up original files without

- *Sungha Baek is with Inha University, Incheon 22212, Republic of Korea. E-mail: sungha@seclab.inha.ac.kr.*
- *Youngdon Jung and Sungjin Lee are with the Department of Information & Communication Engineering, Daegu Institue of Science and Technology, Daegu 42988, Republic of Korea. E-mail: {yeavov, sungjin.lee}@dgist.ac.kr.*
- *David Mohaisen is with the University of Central Florida, 4328 Scorpius Street, Orlando, FL 32816-2450 USA. E-mail: mohaisen@ucf.edu.*
- *DaeHun Nyang is with Cyber Security Department, Ewha Womans University 52, Ewhayeodae-gil, Seodaemun-gu, Seoul 03760, Republic of Korea. E-mail: nyang@ewha.ac.kr.*

Manuscript received 17 May 2019; revised 21 June 2020; accepted 13 July 2020. Date of publication 22 July 2020; date of current version 8 Sept. 2021. (Corresponding authors: Sungjin Lee and DaeHun Nyang.) Recommended for acceptance by J. Shu. Digital Object Identifier no. 10.1109/TC.2020.3011214

any extra copies and to instantly roll back infected files if necessary. Since backup data occupy SSD space, SSD-Insider++ have to regularly delete them to reclaim free space. Before persistently removing them, SSD-Insider++ runs the lazy detection algorithm that evaluates changes of entropy values between old and new data. If a noticeable difference is observed, SSD-Insider++ informs a user that ransomwares may infect files and asks her to recover original ones.

Finally, a tight integration of the ransomware detection and recovery algorithms gives us another benefit. Thanks to the fast and accurate ransomware detection, SSD-Insider++ does not require us to maintain large amounts of backup data inside an SSD, enabling us to throw away data early once they turn out to be useless for the recovery. This eliminates a burden of keeping obsolete data on the SSD side, in contrast to existing SSD-assisted solutions [5].

In order to assess the feasibility of SSD-Insider++, we have implemented both the ransomware detection and recovery algorithms in our in-house open-channel SSD. We have evaluated SSD-Insider++ using real-world and in-house ransomware programs, including WannaCry and Mole, while various background applications are running. Our implementation of SSD-Insider++ has 100 percent detection accuracy with almost 0 percent FRR/FAR in most cases with shorter than 10 seconds of detection latency. We also have confirmed that SSD-Insider++ recovers encrypted files within 1 second without any data loss.

The rest of this paper is organized as follow. In Section 2, we review prior studies and Section 3 gives a design of SSD-Insider++. After describing our detection algorithm in Section 4, we explain how files can be recovered by SSD-Insider++ in Section 5. Section 6 evaluates SSD-Insider++ and, in Section 7, we discuss remaining issues. Section 8 concludes with future directions.

2 Existing Ransomware Defense Systems

We review previously proposed detection and recovery techniques for protecting systems from ransomware attacks.

2.1 Ransomware Detection Techniques

Ransomware detection techniques aim at detecting and removing ransomware programs before they infect user files. Depending on detection methods, they can be classified into two types: signature-based and content-based detections.

Signature-Based Detection. It may be classical yet the most general technique to detect ransomware programs [2], [3]. It analyzes an application's code prior to its execution and stops the launching if it is suspected to be capable of any malicious activity. This detection is done by extracting a code string pattern (a signature) and comparing it to a repository of known malicious code patterns. Signature-based detection, however, often fails to detect ransomware programs if their binaries change. The mutable nature of ransomwares makes it hard to detect ransomwares only by signatures and pushes the detection paradigm further into the realm of content-based defenses.

Content-Based Detection. It detects ransomware activity by monitoring dynamic behaviors of applications and generated data [7], [8], [9]. Scaife *et al.* reported key features that could be a strong indicator of ransomware activity [7]; 1) a file type change which is necessarily followed by ransomware attacks; 2) a low similarity between an original file and its modified version; and 3) a high entropy of a modified file. While performing better than the signature-based one (e.g., detecting new variants), the content-based detection has fundamental limitations. First, since it has to monitor a large volume of data in real time, high CPU and memory overheads cannot be avoided. Second, it involves tweaks in OS, which requires privilege escalation for the monitoring software. Finally, since the content-based detection should allow the execution of ransomwares for a while for monitoring, the loss of original files are unavoidable, unless it performs expensive backup process constantly.

2.2 Data Recovery Techniques

The data recovery techniques take a different approach from the signature-based one in that they focus on recovering infected files.

Decryption-Based Recovery. A 'No More Ransomware Project' may be the most recognized effort to provide decryption-based recovery services [10]. This project helps infected users to regain access to their encrypted files, and, to this end, it creates and maintains a repository of keys that can decrypt data locked by ransomwares. Unfortunately, this approach only works for a previously reported ransomware whose description key is already known.

Backup-Based Recovery. It monitors every change happening in a file system and creates copies of original files in preallocated backup storage which is exclusively managed by vaccine software [4]. Once a ransomware displays a window with instructions for payment or users notice that their files are encrypted, it enables users to recover earlier versions of infected files by using backup files kept in the backup storage. Its key drawback is high overheads incurred when making backup files. It also has to preallocate additional storage to keep original files, which reduces the usable storage capacity. Finally, it runs as user-level applications and requires privilege escalation for monitoring activity inside a file system. Thus, it is vulnerable to ransomwares that are able to evade defense systems [5].

SSD-Assisted Recovery. To lessen overheads caused by online backups and to avoid the vulnerability of the software-based solutions, SSD-assisted recovery techniques are proposed [5], [6], [11], [12]. FlashGuard puts a backup algorithm to an SSD [5]. It leverages the out-of-place update nature of NAND flash to remove extra I/Os for data backup. FlashGuard maintains all the data that have been read by a host system at least for 2$\sim$4 weeks. This is based on the assumption that once data are read, they are potentially encrypted and written by a ransomware. This strong assumption, however, creates serious overheads. As time goes by, a large amount of data are accumulated inside an SSD, which results in excessive SSD GC and leads to waste storage space to keep (probably) uninfected data.

SSD-Insider, which is our prior work [6], showed that the problems of FlashGuard could be overcome by employing ransomware detection inside an SSD. Since ransomware activity is detected in the early stage of infection, it is unnecessary to maintain large amounts of old data. SSD-Insider, however, has fundamental limitations. First, it assumes that ransomwares always overwrite victim files in place, but

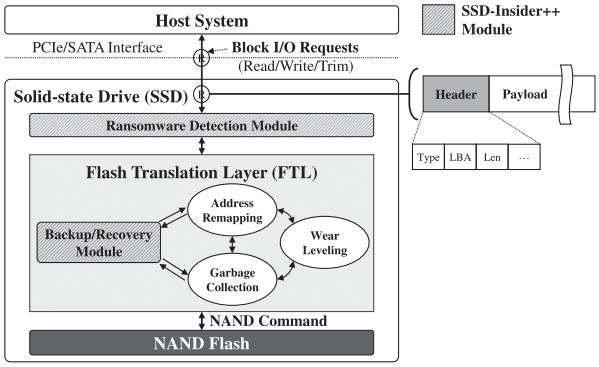


Fig. 1. Overall architecture of SSD-Insider++.

there are many variants that perform out-of-place update attacks [7]. Second, there is no way of recovering original files in the worst case where it fails to detect new ransomware behaviors.

3 Design of SSD-Insider++

SSD-Insider++ addresses the limitations of the existing software- and hardware-based ones by putting the ransomware detection and data recovery algorithms into an SSD. Fig. 1 shows the design of SSD-Insider++ which is composed of two main modules, detection and recovery.

Ransomware Detection Module. The key consideration in designing the ransomware detection algorithm is achieving high detection accuracy with limited information. SSD-Insider++ is only able to access (1) a header of a block I/O packet and (2) its payload. A payload could have useful information, such data entropy, which can be used as a meaningful indicator reflecting infected data [7]. Owing to limited computing power, however, examining the content of a payload exhaustedly at runtime is infeasible.

This practical limitation leads us to develop the detection algorithm which makes a decision by referring to a header of a block I/O packet. The block I/O header contains essential information regarding an I/O request, such as the type of a request (i.e., read, write, or trim), the location of data to be read or written (i.e., a logical block address (LBA)), and its length. By monitoring a stream of packets arriving from the host, SSD-Insider++ can (1) detect unique I/O patterns that are observed when a ransomware runs and (2) know which data are being modified by a ransomware.

Fig. 2a shows what happens when an in-place update ransomware is infecting files. The content of a victim file is read, encrypted, and overwritten in place. A ransomware attempts to infect as many files as possible in a quick manner, trying not to be noticed by a user. Thus, if similar 'update-after-read' I/O patterns are heavily observed, it could be regarded as a sign of ransomware attacks.

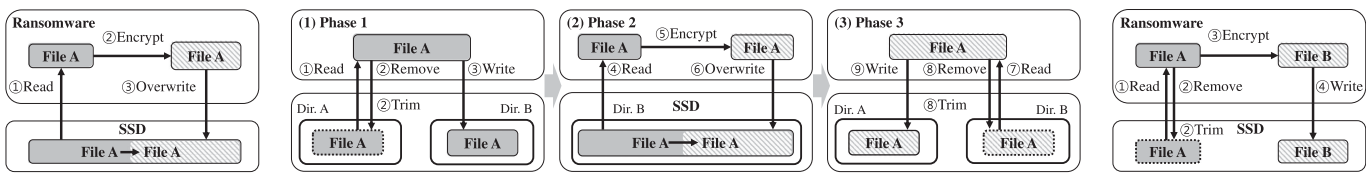
However, some ransomwares behave differently, never overwriting victim files. Fig. 2b and 2c show how out-of-place update ransomwares attack user files. Those two types of attacks are classified by 'Class B' and 'Class C', respectively, in N. Scaife *et al.*'s study [7] Note that the in-place update ransomware is referred to by 'Class A' in [7]. Even though common 'update-after-read' patterns are not observed here, the Class B/C ransomware has to explicitly delete an original file to prevent it from being found and recovered by a user later. Fortunately, whenever a file is removed, a file system sends a trim command to an SSD immediately. This could be used as an useful hint to identify ransomware activity. That is, if heavy 'trim-after-read' operations are detected (e.g., ①∼② in Fig. 2b and 2c), we can suspect that there would be ransomware attacks, and trimmed data could be a victim.

We have seen that ransomware activity can be identified at the storage level just by seeing headers of block I/O commands. Unfortunately, while a ransomware itself has unique patterns which are not typical observed in normal applications, I/O patterns actually seen by SSD-Insider++ are the mixture of I/O requests from a ransomware and normal applications. Thus, detecting the unique patterns of a ransomware from incoming I/O traffic is a key issue. This will be discussed in Section 4 in detail.

Backup/Recovery Module. As shown in Fig. 1, the recovery algorithm is implemented as part of a flash translation layer (FTL). An FTL maintains a remapping table in DRAM that maps LBAs to physical page addresses. Old data are kept in the flash until a garbage collector of an FTL cleans them up. If a ransomware is detected, it is possible to roll back the entire storage status to a safe point. This rollback operation can be done instantly because it only requires updates of mapping entries in DRAM.

This backup/recovery policy of SSD-Insider++ gives us another opportunity for more robust ransomware detection. Since old and new versions of data coexist in the flash for a relatively long time (unless GC removes old ones), we could have a chance to compare entropy values of old and new data. The increase of entropy compared with its original data could be a good indicator of ransomware infection. The high overheads for computing entropy values can be hidden over idle times or can be mitigated by sampling input data.

While it looks like straightforward, several technical issues must be taken into account. The first issue is how to



(a) Ransomware Class A (b) Ransomware Class B (c) Ransomware Class C

Fig. 2. Detailed behaviors of three different types of ransomwares. 'Class A' is an in-place update ransomware that reads (① in (a)), encrypts (②), and overwrites a victim (③). 'Class B' and 'Class C' are an out-of-place ransomware. 'Class B' copies a victim file to a different directory (from Dir. A to Dir. B in the phase 1 in (b)), removing the file on the original directory, and performs encryption on the new directory (the phase 2). It finally copies back the victim to the original directory (the phase 3). The encrypted file is written as a new file in different LBAs because the original file was removed by the ransomware. 'Class C' removes a victim (①∼②) and writes encrypted data with a different filename (③∼④).

TABLE 1
A Summary of Normal Applications

Category	Application	Description
Heavy data erasure	DataWiping	Overwriting disk data satisfying the DoD 5220.22-M standard
	MySQL	Heavy database workloads
	CloudStorage	File synchronization w/ Dropbox
	WindowUpdate	Updating Windows 10 OS
	Fragments	Defragmenting a file system
	Compilation	Compiling the Linux kernel
	DataCopy	Copying a large number of files
IO-intensive	IOStress	Running IOMeter, I/O stress tool
CPU-intensive	Compression	Compressing files w/ Bandizip
Normal	P2PDown	Downloading files w/ BitTorrent

keep track of all the old versions of data, while preventing them from being erased by GC until their safety is confirmed. Second, even though the rollback operation can perfectly recover infected files to original ones, it must be done without affecting data consistency. Finally, all those features should be implemented in a lightweight manner. In Section 5, we discuss those issues in detail.

4 In-Storage Ransomware Detection

SSD-Insider++ uses 'update-after-read' and 'trim-after-read' behaviors to detect suspicious activity. The common outcome of the two operations is erasure of old data. Throughout the rest of this paper, we use the term '*erasure*' to represent a situation where data are updated or trimmed after being read. There would be a time interval between a read and an update/trim destined for the same data. A *time window* T is used as a criterion that decides whether or not a pair of two operations constitutes an erasure operation. If the time interval between two is shorter than T, they make one erasure operation; otherwise, they are regarded as independent events. Currently, T is set 10 seconds.

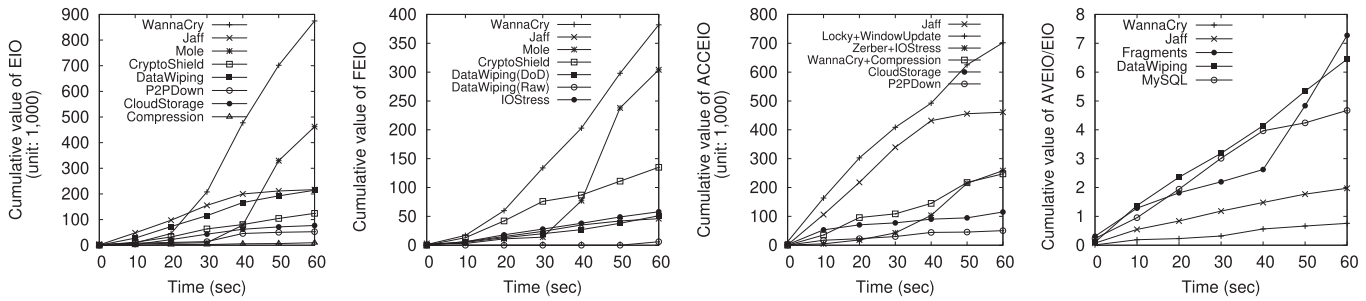
4.1 Invariant Features of Ransomware

To capture ransomware's behavioral traits, and to find features capable of distinguishing ransomwares, we have analyzed the I/O footprints of ransomwares and ordinary applications. We have mainly evaluated four real-world ransomwares, `WannaCry`, `Jaff`, `Mole`, and `CryptoShield`, along with two additional ones, `Locky` and `Zerber`. All of them are in-place update ransomwares.

Ordinary applications we have selected are summarized in Table 1. We categorize them into four types. The 'Heavy data erasure' type includes applications which issue many overwrites and/or trims. The 'IO-Intensive' type represents applications that issue a large number of I/Os for a short period of time. The 'CPU-intensive' type is I/O intensive as well, but requires considerable CPU cycles. The 'Normal' type issues many I/Os, but is not so intensive compared to the I/O intensive type and does not overwrite or trim existing data. Those applications enable us to understand how differently ransomwares leave I/O footprints.

Through experiments, we have found the four major and two minor features uniquely observed with ransomwares.

- *EIO* is the number of Erasure IOs for a short time slice. A *time slice* is defined to be a unit time period for which the number of I/O events are profiled, and it is set 1 second by default. *EIO* is the most significant feature indicating the property of reading, encrypting and erasing the same block of a file for a short period of time. As illustrated in Fig. 3a, `WannaCry` and `Mole` show high cumulative values of *EIO*. This supports our hypothesis that heavy erasures follow reads within a short duration when ransomwares are active. This feature, however, is also observed during the execution of normal applications. The accumulated number of `DataWiping` is as high as that of `Jaff` and that of `CryptoShield` is as low as those of `CloudStorage` and `P2PDown`. Therefore, more features distinguishing ransomwares from these applications are necessary.
- *FEIO* is the Fraction of Erasure IOs over the total number of writes for a time slice (i.e., 1 second). As we have seen in Fig. 3a, one of the hard-to-distinguish applications is `DataWiping`. This is actually not surprising, considering typical behaviors of data wiping workloads that intensively overwrite disk blocks. More specifically, wiping tools perform overwriting over a single block multiple times to persistently destroy data. For example, the DoD 5220.22-M [13] requires 7 updates per one read operation over the same block. A similar trend is also observed with `IOStress`, which is the workload of an I/O stress tool, IOMeter. *FEIO* is able to exclude such noises created by applications like `DataWiping` by taking into account the fraction of EIO over the total writes. Fig. 3b confirms that *FEIO* captures the high rate of updates in the total writes that occur during



(a) Cumulative values of EIO (b) Cumulative values of $FEIO$ (c) Cumulative values of $ACCEIO$ (d) Cumulative values of $AVEIO$

Fig. 3. Four major features that reflect I/O behaviors of in-place update ransomware.

ransomware operations. Except for Jaff, all the ransomwares show high *EIO* and *FEIO* compared with normal applications.

- *ACCEIO* is the ACCumulated number of *E*rasure IOs for a time window T (i.e., 10 time slices). Sometimes, CPU-intensive or IO-intensive jobs might be running while a ransomware is in operation. In this case, the speed of the ransomware slows down, and thus IO requests are dispersed over a rather long time span. Some ransomware like Jaff intentionally delays updates to evade defense systems, and this is the reason why Jaff is not detected by *EIO* and *FEIO* which only consider *EIO*s happened for a short time period (i.e., 1 second). With a longer time distance (i.e., 10 seconds), however, *ACCEIO* captures such behaviors very well as shown in Fig. 3c. In order to emulate CPU- and IO-intensive scenarios, we run three combinations of ransomwares and normal applications, Locky +WindowUpdate, Zerber+IOStress, and WannaCry+Compression, respectively. As illustrated in Fig. 3c, *ACCEIO* can detect suspicious activity, including Jaff that intentionally slows down overwrite operations.
- *AVEIO* is the AVErage length of continuously *E*rased *I*Os in a time window T. *AVEIO* captures the run-length characteristics of ransomware's target files. Ransomwares mainly target documents and images, so it usually does not involve erase operations over a large number of continuous blocks (e.g., several KB to MB), unlike data wiping, file-system defragmentation, and DB updates. Thus, the length of continuously erased blocks is relatively short compared with those applications. Fig. 3d shows that the values of *AVEIO* of WannaCry and Jaff are noticeably lower than those of Fragments, DataWiping, and MySQL.
- *SHORTSLOPE and LONGSLOPE*: The four aforementioned features are the principal features, but, for more accurate detection, SSD-Insider++ employs two auxiliary features: *SHORTSLOPE* and *LONGSLOPE*. Two features are devised to capture ransomware's behavior of abrupt increase of erasure volume from short-term and long-term views, respectively. This is reasonable because the activation of ransomwares causes a sharp increase of erasure operations over previous time periods. *SHORTSLOPE* is the fraction of the number of erasures during a current time slice over the average number of erasures during the previous time slice. *LONGSLOPE* is the fraction of the number of erasures during a current time slice over the average number of erasures during the previous time window.

Fig. 3 shows that the six features (two minor ones are omitted) capture well the ransomware's behavioral characteristics and distinguish it from applications having similar behavior, but using only one feature might miss the active ransomware. Combining all of the six features to maximize detection accuracy is thus important. In this study, we take a machine learning approach to combine them. Using IO data trace collected during the ransomware's active period, a machine learning algorithm is trained with the above six features. Owing to the resource limitation and the tight time-bound characteristics of an SSD, we make use of 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 [14].

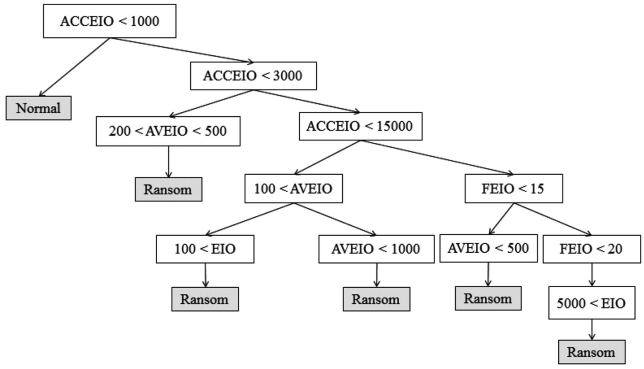


Fig. 4. An illustration of the decision tree.

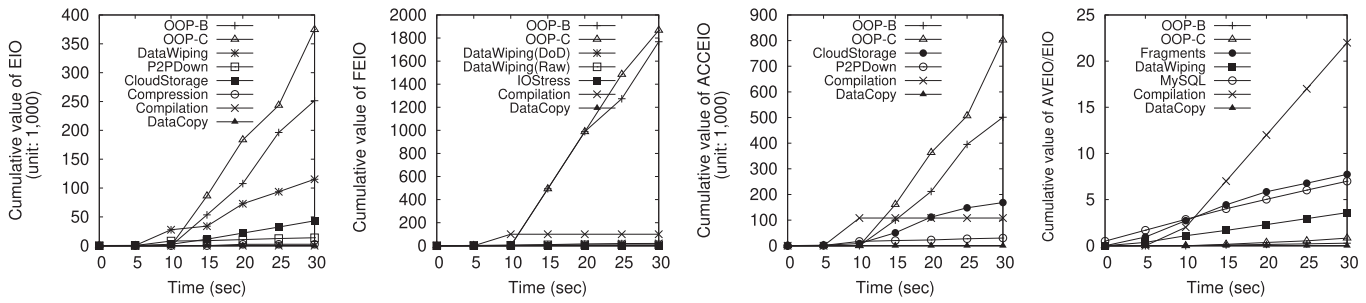
Each vector of input data used in learning is six dimensions, and each dimension is the six attributes defined in the paper. The tree created with this input data is a binary classifier that distinguishes whether or not ransomware is present and has a maximum depth of 5. The criterion for determining ransomware in the top layer of the tree is *ACCEIO*. And, the behavior of ransomware is determined primarily by *ACCEIO* at the higher layer, and by *AVEIO* and *EIO* at the lower layer.

Fig. 4 shows an example of how the decision tree is built. If *ACCEIO* is very low (1,000 or less), it is determined that it is not ransomware. If *ACCEIO* is slightly higher (less than 3,000), *AVEIO*, which means the length of the document, becomes an important criterion for judgment. Finally, if *ACCEIO* is very high (15,000 or higher), it is distinguished according to *FEIO* which separates permanent deletion.

4.2 Applicability to Out-of-Place Update Ransomware

To confirm the feasibility of using the six features to detect out-of-place ransomwares, we have carried out additional experiments with two out-of-place update ransomwares, OOP-B and OOP-C, which are manually implemented based on the observations reported by [7]. OOP-B and OOP-C represent 'Class B' and 'Class C' ransomwares, respectively. Similar to our analysis in the in-place update ransomwares, we compare them with normal applications, including DataWiping, P2PDown, IOStress, and so on. Additionally, we include two normal applications, Compilation and DataCopy, which incur many trim requests to an SSD. Compilation compiles a large number of source codes to build Linux's kernel image. The compilation process creates and removes many temporary files (e.g., object and assembly files), so it frequently sends trim requests. DataCopy is a scenario where user files in one directory are copied to another. Once a file has been copied, the file is removed from the original directory.

Fig. 5 shows that the four invariant features of in-place update ransomwares are consistently observed in out-of-place update ransomwares. OOP-B and OOP-C show noticeably different values from the normal applications; OOP-B



(a) Cumulative values of EIO (b) Cumulative values of $FEIO$ (c) Cumulative values of $ACCEIO$ (d) Cumulative values of $AVEIO$

Fig. 5. Four major features that reflect I/O behaviors of out-of-place update ransomware.

and `OOP-C` have high values for *EIO*, *FEIO*, and *ACCEIO*, and a low value for *AVEIO*, which implies that the four features can be directly used for identifying out-of-place ransomwares. In particular, we found that `Compilation` and `DataCopy` behave differently. `Compilation` creates many temporary files which are eventually trimmed out, but these are not mostly read before being trimmed since they are already cached in the page cache. As a result, 'trim-after-read' operations are not frequently observed. `DataCopy` explicitly reads and trims files after copying them. A time interval between a read and a trim is longer than a time window T. This is because the file system intentionally delays sending trims to lessen I/O traffic. To make it difficult for a user to recover the deleted files, the out-of-place ransomware explicitly sends trim commands by invoking `fsync()` just after the removal of victim files. Thus, the time interval is short compared to `DataCopy`.

4.3 Ransomware Detection Algorithm

Extracting the parameters of the six features in a lightweight manner at run time is an important design issue. SSD-Insider++ realizes this by maintaining a simple in-memory data structure, a *counting table*. Fig. 6 shows the counting table, each entry of which is composed of *Time*, *LBA*, *RL*, and *WL*. *Time* denotes the time slice number at which the entry is created or updated. *LBA* is a starting LBA where data begins to be read or written. *RL* and *WL* represent the length of LBA blocks that have been consecutively read or written, starting from the *LBA*, respectively. The counting table covers entries created or updated for a time window comprised of N time slices. Entries with time slices out of this time range are removed.

Fig. 6 also illustrates how the counting table works with some example cases. For the sake of simplicity, the length of every request is assumed to be 1. When a new request comes, SSD-Insider++ extracts the required information from an I/O request header, along with the current timestamp. It then decides whether to create or update an entry by looking up the table. While it is not displayed in Fig. 6, SSD-Insider++ maintains a hash table to quickly locate a desired entry in the counting table. It gets an LBA and returns a pointer that locates an entry in the counting table. Currently, the hash table contains 250K slots. The first request is a read to LBA 1 (i.e., `(1,R)`). Since there is no matched entry in the table, SSD-Insider++ creates a new entry with a starting LBA, 1, and the timestamp, 0 second. It is also registered in the hash table. Since one LBA is read, *RL* becomes 1 initially. The second and third requests are both reads to LBAs 2 and 3, respectively (i.e., `(2,R)` and `(3,R)`). Since those LBAs are read consequently from LBA 1, *RL* is updated to 3.

If there are duplicate reads, it is just ignored. For example, at the time slice 1, `(1,R)` is ignored. Write requests are dealt with differently from reads. In Fig. 6, the write request to LBA 4 (i.e., `(4,W)`) is ignore because it was not read before. However, the next write to LBA 3 (i.e., `(3,W)`) must be reflected on the table because it was read before. Instead of simply updating *WL* to 1, SSD-Insider++ splits the entry into two: the one starting with LBA 1 and the other starting with LBA 3. This is because the write starts from LBA 3, not from LBA 1, and the run-length of erased LBA blocks should be 1. Conversely, more than two entries can be merged into one. For example, after serving `(2,W)` at the time slice 3, the table entry starting with LBA 1 now has *WL* of 2, so it can be merged with the next entry.

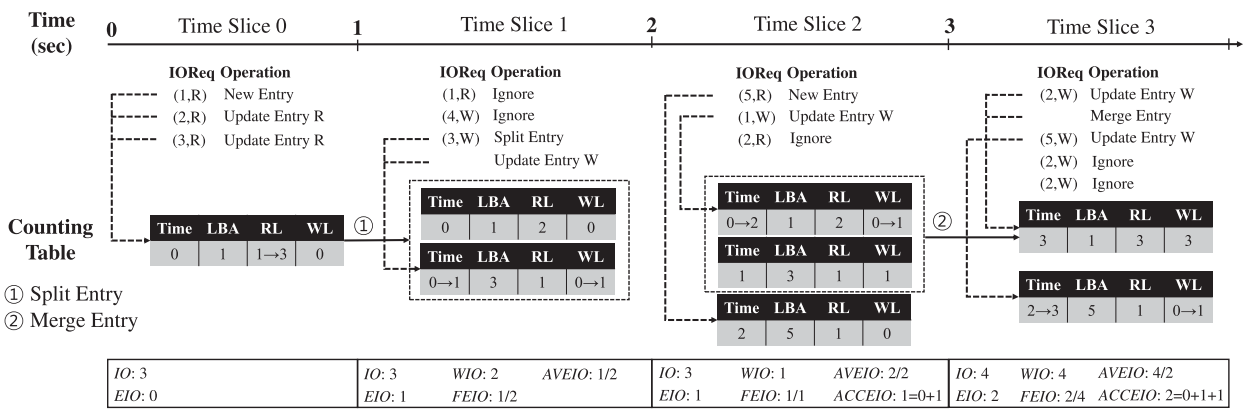


Fig. 6. A data structure of the counting table and working examples.

Using the information collected in the table, SSD-Insider ++ extracts the values of the six features. Fig. 6 shows how those values are obtained. While *IO* counts the total number of I/Os (including reads and writes) which happen during each slice, *WIO* includes write I/Os only. Similarly, *EIO* is measured by counting the number of erasure operations that happen for each time slice. For example, *EIO*s for the 0th, 1st, 2nd, and 3rd slices are 0, 1, 1, and 2, respectively. *FEIO* is calculated by dividing *EIO* by *WIO*. *ACCEIO* is the sum of *EIO* over a time window T, excluding the current slice. For the 0th, 1st, 2nd, and 3rd slices, the values of *ACCEIO* are 0, 0, 1, and 2, respectively. *AVEIO* is the average length of *EIO* over a time window, so it can be obtained by dividing the sum of *WL* values by the number of entries in the table. For instance, *AVEIO* from the 0th to 3th slide is 4/2. Note that, for the sake of simplicity, we do not include *SHORTSLOPE* and *LONGSLOPE* in Fig. 6.

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**
/* for req_i, update the counting table depending on the type as illustrated in Fig. 6 */
9: **end for**

Algorithm 1 shows the detection algorithm with the counting table at a high level. When a request comes, it first sees if a time slice expires or not. If not, the detection algorithm updates table entries as mentioned before. Otherwise, the detection algorithm drops the obsolete entries in the counting table by sliding the window (Line 6), and adjusts *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 is highly likely to be under attack of ransomwares, and 0 means otherwise. For a time window (i.e., 10 time slices), the outputs of the tree are all added up to give a score ranging from 0 to 10 (Line 5). To determine whether a ransomware is active, we use a threshold value 3 which is empirically chosen based on experiments (see Section 6.2).

5 In-Storage Data Recovery

In this section, we explain how the backup/recovery module of SSD-Insider++ is incorporated into an existing FTL. After that, the lazy detection algorithm is presented which examines the actual contents of data to obtain entropy changes between original files and modified ones and uses this information to detect ransomware attacks.

5.1 Online Backup and Instant Recovery

Online Backup Process. We first explain how a conventional FTL handles update and trim commands from the host. In

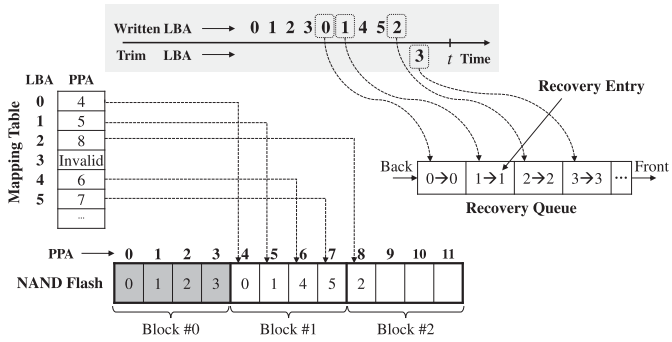


Fig. 7. Handling of overwrites and trims in SSD-Insider++. In the SSD-Insider++'s FTL, the information of the old entries (i.e., LBA 0 → PPA 0, LBA 1 → PPA 1, and LBA 2 → PPA 2) are put into the queue for data recovery.

Fig. 7, the host writes nine LBAs (i.e., 0, 1, 2, 3, 0, 1, 4, 5, and 2), and three of them (i.e., LBAs 0, 1, and 2) are updated. Additionally, LBA 3 is trimmed by the host. Using a mapping table, an FTL can append all the data to flash, avoiding in-place updates. The mapping table indexed by LBA numbers points to physical page addresses (PPA) that holds the latest version of data. For example, the mapping entry for LBA 0 points to PPA 4 in Block #1 (i.e., LBA 0 → PPA 4). The old versions of LBAs 0, 1, and 2 still exist in the flash, but they are not pointed to by the mapping table and the space occupied by them are reclaimed later by a garbage collector. The data for LBA 3 is trimmed out, so its mapping entry points to nothing.

In order to keep track of old versions of data in the flash, SSD-Insider++ introduces an additional data structure, a *recovery queue*, which maintains pairs of LBAs and old PPAs. Whenever an update to a certain LBA comes, SSD-Insider++ FTL puts a pair of the LBA and the corresponding old PPA into the queue. Using old pairs of LBAs/PPAs in the queue, SSD-Insider++ is able to know where old version of data are stored and use it for the recovery later. Trimmed LBAs are also dealt with in the same manner. For example, when LBA 3 is discarded, SSD-Insider++ puts a pair of LBA 3/PPA 3 into the queue. Thus, even if out-of-place ransomwares delete original files after writing down encrypted ones elsewhere with different LBAs, SSD-Insider++ is able to revive the deleted ones since all the required information is maintained the queue.

Let's see how SSD-Insider++ recovers original data using Fig. 7. Suppose that the ransomware infection is detected at t. The new data for LBAs 0, 1, and 2 are encrypted and the data for LBA 3 is removed by the ransomware. The FTL recovers the original data just by updating the mapping table so that the up-to-date mapping pairs (i.e., LBA 0 → PPA 4, LBA 1 → PPA 5, LBA 2 → PPA 8, and LBA 3 → '-') are replaced by the old ones (i.e., LBA 0 → PPA 0, LBA 1 → PPA 1, LBA 2 → PPA 2, and LBA 3 → PPA 3).

The recovery queue grows as an SSD receives update and trim commands. Since DRAM inside an SSD is a precious resource shared by other modules, it is infeasible to maintain all the LBA/PPA pairs in DRAM. Thus, SSD-Insider++ has to limit its maximum size and flush out its contents into the flash when the queue reaches the maximum. The in-memory recovery queue is set to 8-KB so that it fits into a 8-KB physical page size. Each queue entry is consisted of

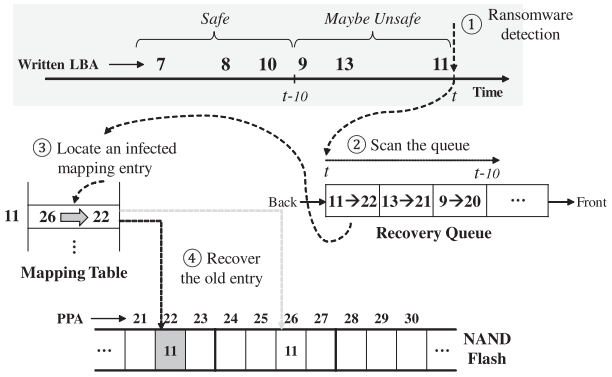


Fig. 8. Data recovery process in SSD-Insider++. A ransomware attack is detected (①) at time t, and a roll-back time, Δt, is chosen as 10 seconds by a user. The SSD-Insider++ FTL scans the recovery queue (②), examining LBAs, old PBAs, and timestamps. For LBAs 9, 13, and 11 that were recently overwritten (i.e., $> t - 10$), 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., $\leq t - 10$), a user presumes that their data are safe, and thus their mapping entries are not updated.

one LBA (4-byte), one PPA (4-byte), and a timestamp (4-byte), so the total 682 entries are maintained in the 8-KB in-memory queue. It means that, whenever 682 update and trim commands come, only one flush operation involving one page write is required. Extra overheads introduced by adding the recovery queue is thus negligible.

Data Recovery Process. Once ransomware activity is detected, SSD-Insider notifies a user of suspicious behaviors being observed. This notification can be made by using a customizable I/O interface 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 [15]. 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 an user.

If a ransomware attack is suspected, SSD-Insider++ gets a *roll-back time*, Δt, from a user which indicates how much we have to go back to the past. SSD-Insider++ makes an SSD read-only, ignoring all the writes arriving from the host. And then, as depicted in Fig. 8, SSD-Insider++ scans backup entries in the two queues from the back to the front, replacing FTL's mapping entries with corresponding backup entries. After the recovery process finishes, the status of the mapping table is rolled back to the time just before Δt seconds. This roll-back process is done in one second because it does not involve any data copies, but just updating the mapping table. After the SSD is recovered, a user is able to copy recovered files to a safe storage device.

The recovery algorithm restores the status of the SSD to Δt 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. This consistency problem can be resolved by using 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 the recovery is similar to one after sudden power loss or a system failure. Only the differences are: 1) it is intentionally caused by SSD firmware and 2) it looks like that a power failure happened Δt seconds before. A series of experiments on EXT4 show that file-system is successfully recovered to a consistent state with `fsck` (see Section 6.2).

Permanent Removal of Backup Data. Garbage collection is only the way of permanently deleting data in the flash. In an attempt to safely keep original data, obsolete data which were updated by new data or trimmed by the host are not immediately deleted by GC in SSD-Insider++; instead, the removal of obsolete data are delayed until they exceed a predefined *backup time* T_{back}.

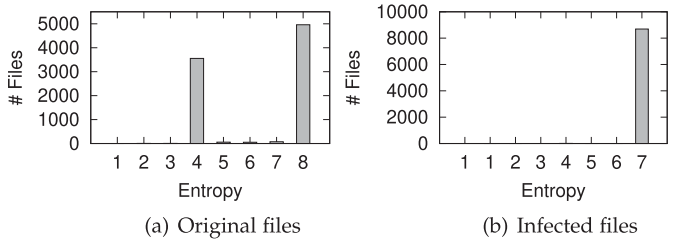
This delayed removal could be a serious burden on the FTL side. Once a victim flash block is chosen, a typical FTL garbage collector just needs to copy valid pages only to a free block, excluding invalid ones. In our case, however, some of those obsolete pages have to be copied if their backup times are not expired yet, which results in the increase of copy costs during GC. The backup time T_{back} decides the overall GC overheads; the longer T_{back}, the more data are accumulated. However, if T_{back} is set too short, it is more likely that a user loses their files by ransomware attacks. For this reason, the previous study sets T_{back} sufficiently long (e.g., two to four weeks), sacrificing user-perceived performance.

Fortunately, thank to the detection algorithm having high accuracy, SSD-Insider++ is relatively free from the trade off between performance and recoverability. Our experimental results say that, for about 99 percent of ransomware attacks, SSD-Insider++ detects suspicious activity in few seconds (e.g., within 10 seconds). This implies that, if the detection algorithm says nothing for a relatively short period of time (e.g., several ten minutes), SSD-Insider++ can throw away obsolete data, believing that there have been no attacks from ransomwares. In this study, T_{back} is configured to 30 minutes. Considering that the detection time of 10 seconds, it is set very conservatively.

With T_{back} of 30 minutes, the recovery queue size in the flash could be huge. Let's consider the worst-case scenario where an SSD offers the write throughput of 512 MB/s and an application updates existing data at the maximum write throughput for a long time. In this case, the recovery queue grows up to 1.3 GB. (= 512 MB / 8 KB $\times$ 12 bytes $\times$ 30 $\times$ 60). Considering that the capacity of modern SSDs is several terabytes, the recovery queue of 1.3 GB is small. However, since the recovery queue should be scanned to detect original data whenever GC is invoked, it would cause non-trivial overheads in the worst case. According to experiments with normal use cases, the recovery queue in the flash is kept small enough, and thus serious overheads for scanning the queue do not occur. But, under heavy overwriting workloads, noticeably high overheads are observed. Such overheads can be reduced by shortening the backup time T_{back} or by employing the idea of [16].

5.2 Lazy Detection

In the worst case, SSD-Insider++ couldn't detect ransomware activity for some reasons. This has not happened or not been observed in our experiments with various ransomwares.



(a) Original files (b) Infected files

Fig. 9. Entropy values of original and infected files.

However, as part of efforts to defend data against potential evasion attacks by future or unknown ransomwares, SSD-Insider++ adopts a lazy detection approach.

An idea of the lazy detection is based on the well-known fact that the resulting files of ransomware attacks have high entropy values, compared to those of original ones. This is the immutable property all of the ransomware programs have. Therefore, if we can measure the changes of entropy values between new and old files, a more accurate detection is possible even when the I/O pattern-based algorithm fails to detect suspicious activity. Fig. 9 shows the entropy values of files that are often selected as victims for ransomware attacks (e.g., `doc`, `ppt`, `txt`, `pdf`, and `gif`) and the entropy values of when they are encrypted by a ransomware. While the original files have diverged entropy values, the infected ones have high entropy all the time.

SSD-Insider++ internally maintains the locations of original data and corresponding modified one. Given a certain LBA number, a physical page containing original data can be obtained by looking at the recovery queue and a page holding the latest data can be found by referring to the mapping table. For example, in Fig. 7, the original and the modified data for LBA 2 are stored in PPA 2 and PPA 8, respectively. However, SSD-Insider++ cannot figure out a pair of old and new data for an LBA that was trimmed since the corresponding mapping entry points to nothing. In the example of Fig. 7, SSD-Insider++ is able to know that PPA 3 contains old data for LBA 3, but the corresponding new data cannot be identified due to the lack of information. The applicability of the lazy detection is thus limited to in-place update ransomwares only.

The most challenging issue with the lazy detection is hiding computation overheads associated with entropy computation. SSD-Insider++ may exploit a backup time that leaves obsolete data undeleted for a relatively long time. Thus, before backup data are persistently removed, SSD-Insider++ computes entropy values of old and new data in background. As another option, we can perform entropy computation for randomly sampled data. In our experience (see Fig. 16), a low sampling rate does not badly affect detection accuracy. Thus, using a sufficiently low sampling rate (e.g., 10 percent) is a reasonable choice. A final option is using a hardware accelerator. Y. Lai *et al*. showed that entropy of data can be computed at throughput of 30 Gbit/s [17] with hardware, which can remove almost all of the associated overheads. This work selects the first two approaches.

SSD-Insider++ reads both original and new LBAs in chronological order and calculates their entropy values. Then, SSD-Insider++ considers the follows three cases:

1) *Both original and new LBAs have high entropy*: It is hard to distinguish whether the original one was encrypted by ransomwares or it had high entropy data by nature. Thus, we exclude them to make a decision.
2) *Both original and new LBAs have low entropy; or original LBAs have high entropy but new ones have low*: New data might be overwritten by normal applications (e.g., a vector editor) with up-to-date data having similar value patterns. This is an indication of which our system might not be under attack by ransomwares.
3) *Original LBAs have low entropy, but new ones have high*: This is a strong sign indicating that user files may be encrypted by ransomwares.

SSD-Insider++ calculates the numbers of LBAs that belong to the second case or the third case, respectively, and computes an *encryption ratio* which represents a ratio of the number of third-case LBAs to the sum of the second- and third-case LBAs. If the ratio is kept higher than 0.2 for a short period of time (i.e., 1 second), it decides that some of files were encrypted by ransomwares. The ratio, 0.2, is empirically chosen while running both ransomwares and I/O intensive applications simultaneously; that is, when nose I/Os are heavily issued. Please note that creating a compression file (e.g., `tar.gz`) and writing a bunch of images files to a disk do not cause a ransomware alarm because those files are usually written to new ones. However, if a user intentionally encrypts files in place, a false alarm might occur. Currently, it is user's responsibility to decide whether or not there has been a ransomware attack.

6 Experimental Results

We implemented SSD-Insider++ in an in-house open-channel SSD prototype. Unlike off-the-shelf SSDs, our SSD platform enabled us to implement and test various FTL algorithms because all those algorithms were run on the block device layer of the host system. Our implementation could be adopted to off-the-shelf SSDs because of no differences in the design principle. 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 was used as a host OS.

6.1 Ransomware Data Sets

We used various well-known ransomwares such as Locky.bdf, Locky.bbs, Zerber.ufb, WannaCry, Jaff, Mole, GlobeImposter, and CryptoShield [18]. As well as those, we implemented three new in-house ransomwares that were not reported before using open source ransomwares [19], [20]; one of them performed an 'in-place update' encryption attack (denoted by INP-A) while the others were based on an 'out-of-place update' (denoted by OOP-B and OOP-C).

We set up the environment where a ransomware ran with various normal applications, which were divided into four types, as in Section 4.1. (1) a 'Heavy erasure' type included a data wiping tool (`WMP`) satisfying DoD 5220.22-M, heavy database updates (`MySQL`), cloud storage synchronization (`Dropbox`), Windows updates (`WindowUpdate`), Linux kernel compilation (`Compilation`), and copy files (`Data-Copy`). (2) an 'IO-intensive' type included IO stress tools such as `DiskMark`, `IOMeter`, and `HDTunePro`, and (3) a 'Normal app' type included software installation such as Viusal Studio (`VS`) and AutoCAD (`AutoCAD`), web-surfing

TABLE 2
Various Combinations of Ransomwares and Applications for Training and Testing of SSD-Insider++

For Training		
Category	Normal Application	Ransomware
Ransom only	None	Locky.bbs
Heavy data erasure (overwrite/trim)	WPM (DataWiping)	None
	MySQL (Database)	None
	Dropbox (CloudStorage)	None
	WindowUpdate	Locky.bdf
IO-intensive	DiskMark (IOStress)	Zerber.ufb
	IOMeter (IOStress)	Zerber.ufb
	HDTunePro (IOStress)	Zerber.ufb
Normal App	AutoCAD/VS (Install)	Locky.bdf
	Chrome (WebSurfing)	Locky.bbs
	Outlook	Locky.bdf
	BitTorrent (P2PDown)	None
	SQLite	None
For Testing		
Category	Normal Application	Ransomware
Ransom only	None	WannaCry
Heavy data erasure (overwrite/trim)	WPM (DataWiping)	GlobeImposter
	MySQL (Database)	INP-A
	DataCopy	OOP-B
	Compilation (Linux kernel)	OOP-C
IO-intensive	IOMeter (IOStress)	CryptoShield
CPU-intensive	Bandizip (Compression)	Mole
	PotEncoder (VideoEncode)	Jaff
Normal App	AutoCAD/VS (Install)	GlobeImposter
	PotPlayer (VideoDecode)	WannaCry
	Outlook	Mole
	Chrome (WebSurfing)	GlobeImposter

(`Chrome`), email synchronization (`Outlook`), P2P download (`BitTorrent`), `SQLite` activities, and video playback (`PotPlayer`). (4) a 'CPU-intensive' type included compression (`Bandizip`) and video encoding (`PotEncoder`) that consumed lots of CPU cycles.

Various combinations of ransomwares and normal applications were used for training a decision tree with ID3. Using the trained tree, we carried out experiments with a different set of ransomwares and applications. The whole dataset and evaluation scenarios for the experiments are summarized in Table 2. Experiments with each combination were conducted 20 times, and the average was taken.

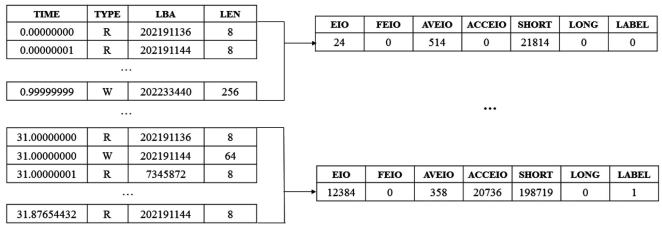


Fig. 10. The process of generating input data for learning.

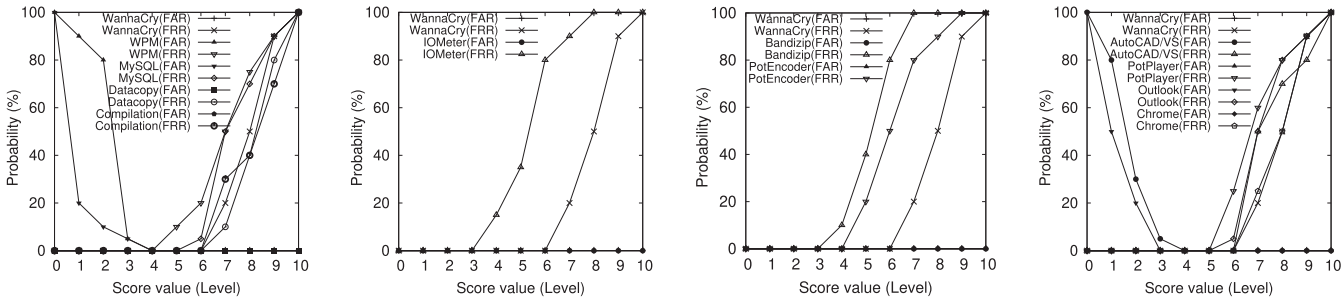
Fig. 10 illustrates how our detection algorithm is trained. We annotate each time slice of a training trace with six properties and a label. The values of six properties are obtained as mentioned in Section 4.1. A label indicates whether ransomware is activated at a specific time slice. The 0 label reports that ransomware is operating, while the label 1 indicates that no ransomware is running.

It is noteworthy that, as shown in the table, we have run different combinations of ransomwares and applications simultaneously for the training and for the testing, respectively. This is to show how accurately and promptly SSD-Insider++ detects unknown or new ransomwares under mixed I/O patterns. For example, for the training phase, we included only three ransomwares (i.e., Locky.bbs, Zerber.ufb, and Locky.bdf) and didn't include CPU-intensive applications. On the other hand, for the testing phase, we used five real-world ransomwares (i.e., WannaCry, GlobeImposter, CryptoShield, Mole, and Jaff) as well as three in-house ransomwares (i.e., INP-A, OOP-B, and OOP-C) which were not used for the training.

6.2 Evaluation of Detection Algorithm

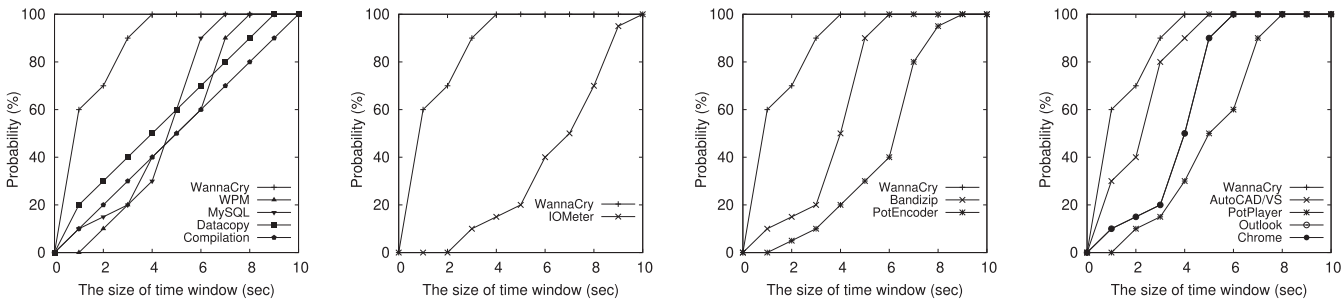
*FAR and FRR Analysis.*To evaluate the accuracy of SSD-Insider ++ detection algorithms, we measured False Acceptance Rate (FAR) and False Rejection Rate (FRR) of the detection algorithm varying the threshold of the score value to detect a ransomware. The FAR is a ratio of which SSD-Insider++ indicates ransomware attacks even though it is not. Conversely, the FRR is a ratio of which SSD-Insider++ does not send any alarms even when the system is under ransomware attacks. For accurate detection, both FAR and FRR should be low enough.

Fig. 11 summarizes our experimental results on accuracy in terms of FAR/FRR. We evaluated the combinations of



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

Fig. 11. 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 percent and FAR is at most 5 percent only when heavy overwriting such as data wiping.



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

Fig. 12. SSD-Insider++'s detection latency: all of the ransomware in various scenarios were detected 100 percent in 10 seconds.

normal applications and ransomwares listed in 'For Testing' in Table 2. We denoted application names only in the graphs (e.g., `WPM` instead of WPM+GlobeImposter). For comparison purposes, the FAR/FRR values of when WannaCry solely (i.e., None+WannaCry) ran were included in all the graphs. Even though there was some variation depending on the scenario, the detection algorithm was able to detect ransomware's activity accurately under all types of background applications when the score threshold was 3. In terms of FRR, the worst background noise came from IO-intensive and CPU-intensive jobs as shown in Figs. 11b and 11c. This is because they interfered with ransomwares to slow down the speed of data erasure by heavy usage of CPU and IO. In case of FAR, on the other hand, the worst scenario was heavy overwriting applications (see Fig. 11a) whose behaviors were similar to those of ransomwares. However, even in such heavy CPU, IO usage and heavy overwriting scenarios, SSD-Insider++ detected accurately ransomware activity with the threshold value 3.

Overall, FRRs in all the scenarios were 0 percent, and FARs were close to at most 5 percent with the threshold value 3. This 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. A false alarm might interrupt users, but it would rarely happen only with heavy DB update (`MySQL`), data wiping (`WPM`), and software install (`AutoCAD/VS`) as shown in Fig. 11. Here we note that 5 percent of FAR is not average for all applications, but only with those uncommon applications, so false alarm to users occurs 5 percent $\times$ the uncommon applications' running probability.

The out-of-place ransomwares, OOP-B and OOP-C, exhibited low FAR and FRR, so they were accurately detected by SSD-Insider++ even when they ran with the applications that issued many trim and I/Os requests. As mentioned in Section 4.2, this was due to the fact that out-of-place ransomwares behaved quite differently, generating unique I/O footprints, from normal applications.

Detection Latency. The time window size, T, has a high impact on SSD-Insider++'s detection accuracy and latency because it affects deciding values of three invariant features, *ACCEIO*, *AVEIO*, and *LONGSLOPE*. There is a trade-off between accuracy and latency; if we increase T, the accuracy gets better because we monitor incoming I/O traffic longer, but the detection latency increases. For a small T, we can detect faster, but with lower accuracy due to lack of information. In Fig. 12, we measured detection accuracy with threshold score 3 varying the time window length. If T is 10s, ransomwares in all the scenarios were detected 100 percent. The detection algorithm using $T < 10$ could not detect perfectly ransomwares in some scenarios having CPU-intensive and IO-intensive background applications such as `IOMeter` and `Bandizip` as shown in Fig. 12b and 12c. Otherwise, most of them were detected successfully with a time window of $T = 7, 8, 9$. For example, $T = 8$ was enough for `WPM` and `MySQL` with similar data erasure behavior as shown in Fig. 12a.

6.3 Evaluation of Recovery Algorithm

Recovery Time. We assess the time taken to recover infected SSDs. For the 13 scenarios, we measured the time between the infection and SSDs recovery. Fig. 13 summarizes our experimental results. Even though there are differences among the scenarios, we notice that the elapsed time to fully recover SSDs was less than 1 second. As expected, this is because SSD-Insider++ only modified the mapping table for data recovery, avoiding physical data copies in flash.

For traces like `Bandizip` and `PotEncoder`, SSD-Insider ++ showed a relatively long recovery time. Those traces wrote a large amount of data before the ransomware was detected. When the recovery module was activated, it found many items in the recovery queue to be examined and to roll back infected mapping entries into safe ones. However, even with such workloads with long queue lengths, recovery times were short. It must be noted that, even though some parts of the recovery queue were stored in flash (see Section 5.1), reading in-flash queue entries didn't take so long because their sizes were small enough.

Data Consistency. To evaluate data consistency, we intentionally exposed the host to ransomware attacks repeatedly 100 times using a custom ransomware we developed. It

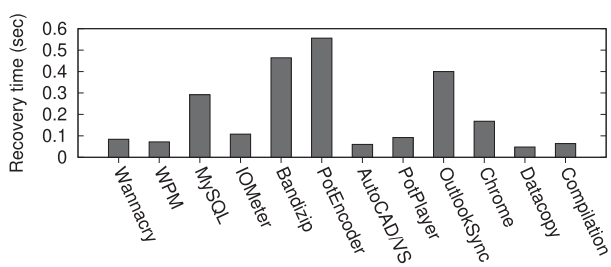


Fig. 13. Total elapsed times taken to recover ransomware-infected SSDs with various background applications.

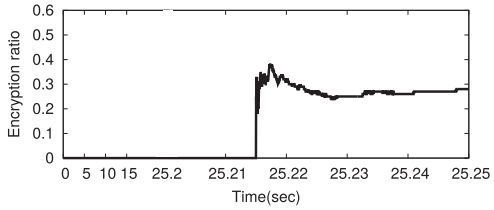


Fig. 14. Change of entropy ratios over time.

infected larger than 1 GB files at an arbitrary point of time. Once SSD-Insider++ detected the activity of the ransomware, it stopped servicing writes from the host, asked the user to recover infected files and 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. Our results confirmed that infected SSDs successfully returned to a consistent status with no encrypted files left. See Table 2 in [6] for more detailed results.

Lazy Detection. We evaluated the effectiveness of the lazy detection algorithm. To this end, we intentionally disabled the I/O pattern-based detection algorithm, and then we ran the `INP-A` ransomware while updating source codes (i.e., `git pull`) and compiling them. This background job was I/O intensive and overwrote many files. Fig. 14 plots how the encryption ratio has changed over time. While the background task solely ran, the encryption ratio was maintained low because of small differences of entropy values between old and new files. `INP-A` was triggered at 10 seconds, and by analyzing the entropy values of overwritten LBAs later, SSD-Insider++ detected a sharp increase of the entropy ratio around 25 seconds.

We evaluated how FAR and FRR values changed depending on a target encryption ratio which was used as a criterion to decide whether ransomware attacks happened or not. We conducted experiments with three ransomwares, `INP-A`, `CryptSky` [21], and `GonnaCry` [22], which ran with three usual scenarios, `git clone`, `Compilation`, and `git pull` each. Fig. 15 shows our experimental results. When the target encryption ratio was set too high (e.g., 0.5∼0.9), FRR greatly increased, lowing detection accuracy. On the other hand, regardless of the encryption ratio, FAR was kept low. As expected, this was because in normal applications there were no significant changes on entropy values between old and new data. This implies that, even if a relatively small increase in an encryption ratio is observed, it can be regarded as a sign of ransomware attacks. Consequently, we found that when the ratio was 0.2, both FAR and FRR became almost zero.

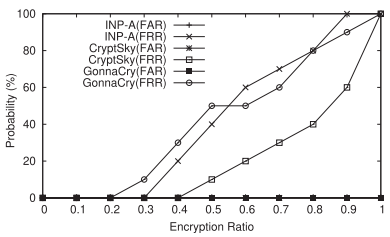
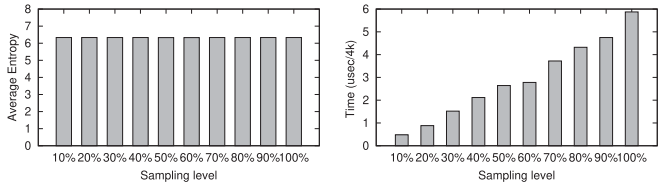


Fig. 15. FRR and FAR with different encryption ratios.



(a) Accuracy (b) Computation time

Fig. 16. Impact of sampling-based entropy computation.

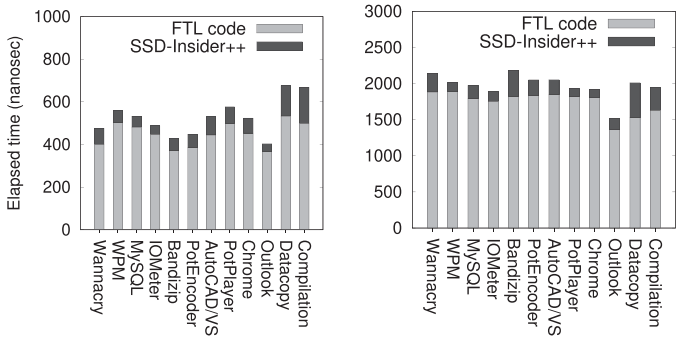
Finally, we evaluated the impact of sampling-based entropy computation on accuracy and overhead. Fig. 16 shows measured average entropy values and computation times for a same dataset with different sampling levels ranging from 100 to 10 percent. Here, 100 percent sampling means that we measured an entropy value for all the target data, while 10 percent means that we measured entropy for data of 10 percent randomly selected. As illustrated, the similar level of accuracy was maintained even with 10 percent sampling. This indicates that we can reduce entropy computation time by a fraction of 10, while maintaining its accuracy.

6.4 Overhead Evaluation

I/O Elapsed Time. For the 13 testing traces in Table 2, we measured read/write elapsed times increased by SSD-Insider++. Fig. 17 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.

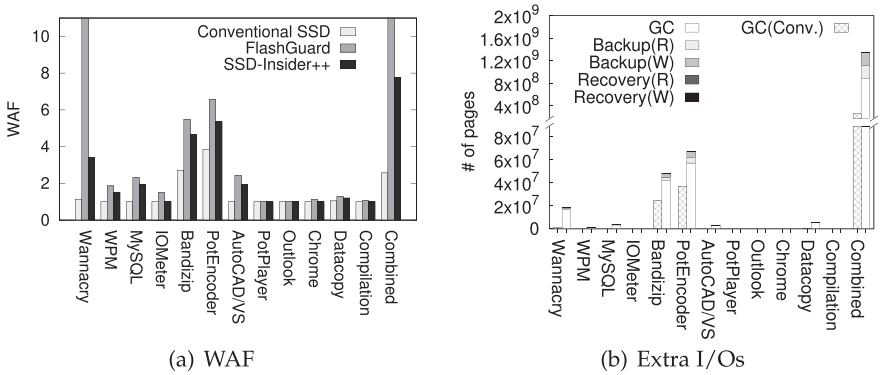
Compared with the conventional FTL, read and write latencies with SSD-Insider++ increased by 17.3 and 12.8 percent, on average. More specifically, the elapsed times taken for the FTL (without SSD-Insider++) to handle 4-KB block reads and writes were 462 ns and 1,765 ns, respectively, on average. The extra overheads added by the SSD-Insider++ detection/recovery algorithms were just 80 ns and 226 ns, respectively. Considering that NAND page-read and page-write latencies are 50 μs and 500 μs [23], these extra delays accounted for a negligible portion of the total I/O latency, not affecting the user-perceived response times.

Garbage Collection Cost. SSD-Insider++ recovery algorithm has to copy invalid pages to keep old versions of data



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

Fig. 17. Impact of SSD-Insider++ on I/O elapsed times.



(a) WAF (b) Extra I/Os

Fig. 18. Analysis of GC I/Os.

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. However, compared with FlashGuard that keeps all the obsolete data for a very long period of time (e.g., at least two or four weeks), SSD-Insider++ incurs less GC overheads. This is because SSD-Insider++ maintained backup data for only 30 minutes, as explained in Section 2.2.

We compared FlashGuard and SSD-Insider++ under the worst-case scenario where GC costs were high. We filled up 90 percent of the SSD with files and ran traces on it. In addition, we created an I/O intensive workload, `Combined`, by combining three heavy workloads, `Bandizip`, `PatEncoder`, and `DataCopy`. Fig. 18a shows that SSD-Insider++ had 4.6, 5.3, 3.3, and 7.7 WAFs for `Bandizip`, `PotEncoder`, `WannaCry`, and `Combined`, respectively, which were higher than the conventional FTL. However, it was much lower than FlashGuard that had 5.5, 6.6, 52.7, and 60.7 WAFs for the four benchmarks. Other workloads (e.g., `PotPlayer` and `Outlook`) had low WAFs which were close to 1.0 because they did not require many page copies for GC.

We also analyzed extra I/Os that occurred during GC in SSD-Insider++. In Fig. 18b, 'GC' is the number of pages copied during GC, 'Backup(R)' and 'Backup(W)' represent the number of invalid pages read and written to back up original data, and 'Recovery(R)' and 'Recovery(W)' are the number of pages read to scan and to update the recovery queue in the flash during GC. 'Backup(R/W)' and 'Recovery(R/W)' accounted for a trivial proportion of the total I/Os during GC. This confirms that those extra I/Os do not cause serious overheads.

For `Combined`, however, we have observed a noticeably high WAF with a relatively large number of extra I/Os for backup and recovery pages. Under a write-heavy workload, the recovery queue gets huge (see Fig. 19) and the number of backup pages accumulated in the flash increases, which results in non-trivial I/O overheads during GC. As mentioned in Section 5.1, this problem can be mitigated by using a better backup algorithm [16] or by reducing the length of the backup time.

Finally, some readers might notice that, even considering the number of extra I/Os for backup and recovery pages, the WAFs of SSD-Insider++ were higher than those of the conventional SSD. For example, in the case of Bandizip (Fig. 18b), extra I/Os for backup/recovery pages accounted for a negligible proportion of the total I/Os, but SSD-Insider++ suffered from a much higher WAF. This is owing to its victim selection policy. To prevent backup pages from being removed by GC, SSD-Insider++ chooses victims, assuming that backup pages contain latest data. This leads SSD-Insider++ to select blocks with more valid pages, which results in the increase of WAFs. Fig. 18b also displays the number of page copies for GC in the conventional SSD (labeled by GC(Conv.)). As expected, it requires a smaller number of page copies for some workloads.

Recovery Queue Overhead. To understand the impact of flushing out the recovery queue on performance, we have measured the write throughput of SSD-Insider++ when a write intensive workload, FIO, ran. Fig. 20 shows our experimental result. SSD-Insider++ showed 8 percent lower write throughput than the conventional SSD. We observed that the number of extra writes to flush out recovery-queue entries to the flash was small as explained in Section 5.1. Instead, SSD-Insider++ software running in ARM CPUs incurred non-trivial overheads when it handled large number of I/Os simultaneously. We believe that this overhead can be lowered through software optimization.

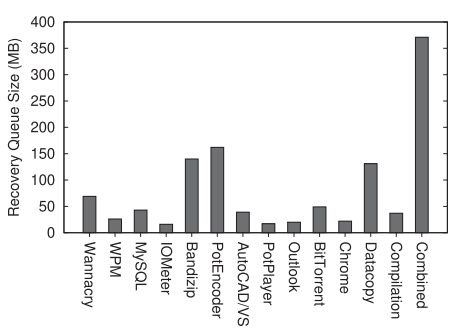


Fig. 19. Recovery queue size (MB).

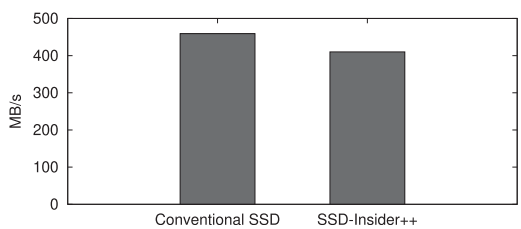


Fig. 20. Write throughput with FIO.

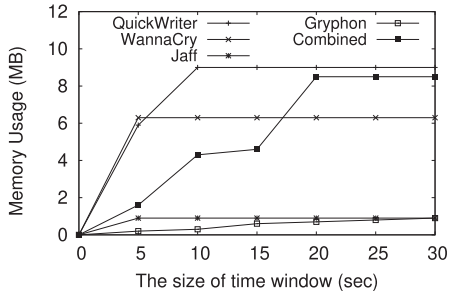


Fig. 21. Amount of memory used by the counting table.

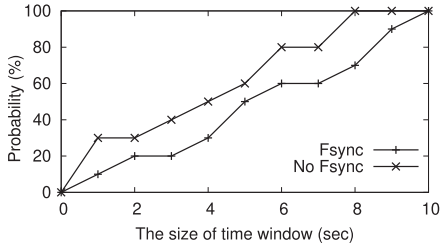


Fig. 22. Detection latency of OOP ransomwares.

DRAM Requirement. SSD-Insider++ has to maintain additional data structures, the hash table, the recovery queue, and the counting table. The sizes of the hash table with 250K slots and the recovery queue are fixed to 10 MB and 8 KB, respectively. However, the counting table size varies depending on input workloads. To understand how much memory SSD-Insider++ consumed, we measured the memory consumption of the counting table for five workloads. Fig. 21 shows our experimental results. QuickWriter is our in-house ransomware to overwrite heavily to consume as much memory as possible. Gryphon is a newly-found ransomware with a strategy similar to the delayed overwriting. Heavy overwriters such as QuickWriter and WannaCry consumed the largest amount of memory, but even with 30s time window, 9 MB was enough for our SSD-Insider++ operation. Finally, we measured the memory usage of the Combined workload. Similar to other heavy workloads, its memory consumption was about 8.5 MB.

7 Discussion

Delayed TRIM. We have assumed that all the out-of-place update ransomwares explicitly invoke `fsync()` to immediately send trim commands to deleted files. This is a reasonable strategy because it makes it impossible for users to recover original files. However, there is a possibility that some OOP ransomwares do not explicitly call `fsync()` after deleting victim files. Even in such a case, SSD-Insider++ is able to detect out-of-place update ransomware.

Fig. 22 shows our experimental results using two different types of OOP ransomwares: one with explicit `fsync()` and the other with no `fsync()`. For no `fsync()`, the detection latency slightly increased owing to delayed trims. However, SSD-Insider++ could detect ransomware activities in 10 seconds. The commit time of EXT4 for synchronizing all its data and metadata was 5 seconds. We guess that this forced all the pending trims for deleted files to be eventually sent to the SSD in 10 seconds.

When or how frequently trim commands are sent to an SSD is different depending on file-system implementation. Some file systems may delay sending trims as long as possible. SSD-Insider++ cannot detect ransomware activities, but since deleted files are not permanently erased and still exist in the file system, users can recover original files.

File System Dependency. SSD-Insider++ is designed for journaling file systems (e.g., NTFS and EXT2/3/4). Unlike journaling file systems, log-structured file systems (LFS) and copy-on-write file systems (COW) append almost all of the data to storage media sequentially, avoiding overwrites of user data. Since our detection algorithm identifies attacks based on frequent in-place update behaviors of ransomwares, it would not work properly under LFS and COW. One way to address the above problem is to enhance a host-to-storage interface so that more detailed file-system level information, such as inode numbers and offsets, is delivered to the FTL. This allows us to keep track of I/O access patterns of ransomwares on a file basis, so SSD-Insider++ can identify ransomware behaviors more accurately without relying on LBA-based overwriting patterns.

8 Conclusion

In this paper, we have proposed an SSD-assisted ransomware detection and data recovery technique, called SSD-Insider++. Based on a new set of behavioral features of ransomware programs, SSD-Insider++ was able to detect ransomware attacks early. By carrying out various experimental results with real-world ransomwares, the discovered features have been proven to be effective as strong indicators of ransomware activity. SSD-Insider++ also supported quick and perfect data recovery by leveraging the out-of-place update nature of NAND flash. To address the worst case scenario where SSD-Insider++ cannot identify ransomware attacks, SSD-Insider++ supported a lazy detection algorithm. To show the feasibility of SSD-Insider++, the detection/recovery algorithms were implemented in an open-channel SSD. Our evaluation results showed that SSD-Insider++ was accurate and fast for detection, and it could perfectly recover an infected SSD without any data loss.

Acknowledgments

An earlier version of this article was presented at the IEEE International Conference on Distributed Computing Systems, July 2-5, 2018 [6]. This work was supported in part by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT) under Grant NRF-2017R1E1A1A01077410, in part by Global Research Laboratory (GRL) Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Science and ICT under Grant NRF-2016K1A1A2912757, and in part by Institute for Information & Communications Technology Promotion (IITP) grant funded by the Korea government (MSIT) under Grant 20180003910012003, Behavior-Based Ransomware Detection Technology Using I/O Distribution. Sungha Baek and Youngdon Jung contributed equally to this work.

References

[1] M.-H. Cho, "Korean web host hands over 1 Billion won to ransomware crooks," 2017. [Online]. Available: https://www.zdnet.com/article/korean-web-host-hands-over-1-billion-won- to-ransomware-crooks

[2] S. Kumar and E. H. Spafford, "A generic virus scanner for C++," in *Proc. Comput. Secur. Appl. Conf.*, 1992, pp. 210–219.
[3] P. Traynor, M. Chien, S. Weaver, B. Hicks, and P. McDaniel, "Noninvasive methods for host certification," *ACM Trans. Inf. Syst. Secur.*, vol. 11, no. 3, 2008, Art. no. 16.
[4] A. Continella *et al.*, "ShieldFS: A self-healing, ransomware-aware filesystem," in *Proc. Annu. Conf. Comput. Secur. Appl.*, 2016, pp. 336–347.
[5] J. Huang, J. Xu, X. Xing, P. Liu, and M. K. Qureshi, "FlashGuard: Leveraging intrinsic flash properties to defend against encryption ransomware," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2017, pp. 2231–2244.
[6] S. Baek, Y. Jung, A. Mohaisen, S. Lee, and D. Nyang, "SSD-insider: Internal defense of solid-state drive against ransomware with perfect data recovery," in *Proc. IEEE Int. Conf. Distrib. Comput. Syst.*, 2018, pp. 875–884.
[7] N. Scaife, H. Carter, P. Traynor, and K. Butler, "CryptoLock (and Drop It): Stopping ransomware attacks on user data," in *Proc. IEEE Int. Conf. Distrib. Comput. Syst.*, 2016, pp. 303–312.
[8] G. Canfora, F. Mercaldo, and C. A. Visaggio, "An HMM and structural entropy based detector for android malware: An empirical study," *Comput. Secur.*, vol. 61, pp. 1–18, 2016.
[9] T. Yang, Y. Yang, K. Qian, D. C. T. Lo, Y. Qian, and L. Tao, "Automated detection and analysis for Android ransomware," in *Proc. IEEE Int. Conf. High Perform. Comput. Commun.*, 2015, pp. 1338–1343.
[10] No more ransomware project, 2018. [Online]. Available: https://www.nomoreransom.org
[11] D. Min *et al.*, "Amoeba: An autonomous backup and recovery SSD for ransomware attack defense," *IEEE Comput. Archit. Lett.*, vol. 17, no. 2, pp. 245–248, Jul.–Dec. 2018.
[12] J. Park, Y. Jung, J. Won, M. Kang, S. Lee, and J. Kim, "RansomBlocker: A low-overhead ransomware-proof SSD," in *Proc. 56th Annu. Des. Autom. Conf.*, 2019, pp. 1–6.
[13] National Industrial Security Program, "Operating Manual," 2006. [Online]. Available: https://www.esd.whs.mil/portals/54/documents/dd/issuances/dodm/522022m.pdf
[14] J. R. Quinlan, "Induction of decision trees," *Mach. Learn.*, vol. 1, no. 1, pp. 81–106, 1986.
[15] Seagate Technology, "SMART Attribute Specification," 2011. [Online]. Available: https://t1.daumcdn.net/brunch/service/user/axm/file/zRYOdwPu3OMoKYmBOby1fEEQEbU.pdf
[16] X. Wang, Y. Yuan, Y. Zhou, C. C. Coats, and J. Huang, "Project almanac: A time-traveling solid-state drive," in *Proc. 14th EuroSys Conf.*, 2019, pp. 1–16.
[17] T. W. Lai, Yu-Kuenand H.-P. You, "Hardware-assisted estimation of entropy norm for high-speed network traffic," *Electron. Lett.*, vol. 50, no. 24, pp. 1845–1847, 2014.
[18] Virus total. 2020. [Online]. Available: https://www.virustotal.com
[19] Virtual gangster. 2020. [Online]. Available: https://github.com/roothaxor/Ransom
[20] A POC windows crypto-ransomware (academic), 2019. [Online]. Available: https://github.com/mauri870/ransomware
[21] A simple, fully python ransomware PoC, 2019. [Online]. Available: https://github.com/deadPix3l/CryptSky
[22] GonnaCry rasomware, 2019. [Online]. Available: https://github.com/tarcisio-marinho/GonnaCry
[23] Micron Technology, Inc., "MT29F8G08 Data Sheets," 2020. [Online]. Available: https://www.micron.com/products/nand-flash/slc-nand/part-catalog/mt29f8g08ababawp-itx

Sungha Baek received the BS degree in computer science and engineering and mathematics from Inha University, South Korea, in 2005, the MS degree in computer science and engineering from Inha University, South Korea, in 2007, and the PhD degree in computer science and engineering from Inha University, South Korea, in 2020. His current research interests include big data and artificial intelligent.

Youngdon Jung received the BS degree in computer science and engineering and Japanese language and culture from Inha University, South Korea, in 2017 and the MS degree in information and communication engineering from Daegu Institue of Science and Technology, South Korea, in 2019. He is currently working as a technical staff with Lotte Data Communication in South Korea. His current research interests include storage systems, operating systems, system architecture, and system software.

David Mohaisen (Senior Member, IEEE) received the MSc and PhD degrees from the University of Minnesota, Minneapolis, Minnesota, in 2012. He is currently an associate professor with the University of Central Florida, Orlando, Florida, where he only directs the Security and Analytics Lab (SEAL). Before joining University of Central Florida, Orlando, Florida, in 2017, he was an assistant professor at SUNY Buffalo (2015–2017) and a senior research scientist at Verisign Labs (2012–2015). His research interests are in the areas of networked systems and their security, online privacy, and measurements. He is an associate editor of the *IEEE Transactions on Mobile Computing*, and is a senior member of ACM (2018).

Sungjin Lee received the BE degree in electrical engineering from Korea University, South Korea, in 2005 and the MS and PhD degrees in computer science and engineering from Seoul National University, South Korea, in 2007 and 2013, respectively. He is an assistant professor at Daegu Institue of Science and Technology, in South Korea. Before joining Daegu Institue of Science and Technology, South Korea, he was an assistant professor at Inha University, South Korea (2016–2017) and a postdoctoral associate with the Computer Science and Artificial Intelligence Laboratory, Massachusetts Institute of Technology, Cambridge, Massachusetts (2013–2016). His current research interests include storage systems, operating systems, and system software.

DaeHun Nyang (Member, IEEE) received the BEng degree in electronic engineering from Korea Advanced Institute of Science and Technology (KAIST), South Korea, the MS and PhD degrees in computer science from Yonsei University, Korea, in 1994, 1996, and 2000 respectively. He had been a senior researcher at Electronics and Telecommunications Research Institute, Korea (ETRI), from 2000 to 2003. Since 2020, he has been a professor at Cyber Security Department of Software and Engineering Division of Ewha Womans University, South Korea. From 2003 to 2020, he had been a professor at Computer Science and Engineering Department of Inha University, Korea where he was also the founding director of the Information Security Research Laboratory. He is a member of the board of directors and editorial board of Korean Institute of Information Security and Cryptology. He also serves ETRI Journal as a section editor (Information security section). He's a member of KIISC. His research interests include all algorithmic aspects of AI security, AI attack, network security, privacy, usable security, biometrics and their applications to authentication, network measurement, load balancing algorithm, and INS (In-Network Security) powered by programmable routers.