RansomBlocker: a Low-Overhead Ransomware-Proof SSD

Jisung Park
Seoul National University
jspark@davinci.snu.ac.kr

Youngdon Jung
DGIST
yeavov@dgist.ac.kr

Jonghoon Won
Seoul National University
barber@snu.ac.kr

Minji Kang
Seoul National University
kyang@snu.ac.kr

Sungjin Lee
DGIST
sungjin.lee@dgist.ac.kr

Jihong Kim
Seoul National University
jihong@davinci.snu.ac.kr

ABSTRACT

We present a low-overhead ransomware-proof SSD, called RansomBlocker (RBlocker). RBlocker provides 100% full protections against all possible ransomware attacks by delaying every data deletion until no attack is guaranteed. To reduce storage overheads of the delayed deletion, RBlocker employs a time-out based backup policy. Based on the fact that ransomware must store encrypted version of target files, early deletions of obsolete data are allowed if no encrypted write was detected for a short interval. Otherwise, RBlocker keeps the data for an interval long enough to guarantee no attack condition. For an accurate in-line detection of encrypted writes, we leverages entropy- and CNN-based detectors in an integrated fashion. Our experimental results show that RBlocker can defend all types of ransomware attacks with negligible overheads.

1 INTRODUCTION

Ransomware stealthily encrypts user files and demands a ransom from a user for an encryption key to access the files. The high financial benefits of ransomware make it a profitable business, which motivates many cyber-criminals to develop and distribute various ransomware programs. In the first six months of 2018, 181.5 million ransomware attacks have been reported, which is a 229% increase over the same period in 2017 [11].

Typical anti-ransomware solutions attempt to protect user files by detecting ransomware before it runs and/or by backing up files on remote or local storage. However, since most anti-virus programs run in a host system as a user application, they are vulnerable to evasion attacks with root privileges. Backing up original files cannot be a 100% reliable solution either because a backup system itself can be infected [5]. Moreover, creating a backup for recovery can cause significant overheads, which may degrade user experience.

To address the potential vulnerability and backup overheads of host-level defense schemes, storage-level solutions have been proposed recently [3, 5]. They put ransomware detection/recovery logics in a NAND flash-based solid state drive (SSD) which is *physically decoupled* from a host, and leverage the *out-of-place update* property of NAND flash to lessen backup overheads. NAND flash memory does not support in-place updates, so SSDs handle host writes in an append-only manner, leaving obsolete file data unmodified. Only SSD firmware, a flash translation layer (FTL), can explicitly remove them later through garbage collection (GC). Therefore, by preventing GC from erasing possibly attacked data, SSD-level solutions can back them up at no I/O cost. Furthermore, since this backup and removal processes are done inside SSDs, they can protect data from evasion attacks at the host level.

The existing SSD-level solutions, however, are insufficient either to be widely used in practice. FlashGuard [5], which provides 100% full protections from ransomware attacks, suffers from a very high space overhead of maintaining backup data owing to its overly conservative backup policy. Since every ransomware has to first read a victim file to encrypt it, FlashGuard assumes that once data are read by a host, they are possibly attacked by ransomware, thus keeping them until no ransomware attack is guaranteed. Although this simple scheme guarantees the highest level of protection, normal data (not attacked by ransomware) are also unnecessarily stored in an SSD. This high space overhead makes FlashGuard almost impossible to be used. Even worse, it greatly increases the overall GC cost.

SSD-Insider [3] addresses the limitations of FlashGuard by leveraging a ransomware detection algorithm on the storage side. By monitoring unique I/O footprints of ransomware, it is able to detect whether a host is under a ransomware attack or not within 10 seconds. This early detection enables SSD-Insider not to maintain backup data for a long time, which mitigates the high space overhead problem. SSD-Insider, however, could be easily compromised by ransomware variants that intentionally issue modified I/O patterns that evade the SSD-Insider's detection algorithm.

In this paper, we propose a novel SSD-level protection technique against ransomware attacks, called *RansomBlocker* (*RBlocker*). Unlike the existing SSD-level solutions, RBlocker examines the content of incoming data for detecting ransomware attacks. Specifically, at run time, RBlocker determines whether incoming data are *encrypted*. Since encrypting victim files is an invariant step of all ransomware attacks, if an incoming write is not encrypted, we can guarantee that the data are not infected by ransomware. If encrypted data are identified, RBlocker delays the deletion of its original data by excluding them from GC process. Such a selective backup can significantly mitigate the space overhead, and is not affected by new variants that change I/O footprints because the backup decision is based on the actual content of data.

Achieving high accuracy in detecting encrypted writes is a key challenge in designing RBlocker. RBlocker uses *entropy* of input data as a key metric to determine encrypted writes, based on the fact that encrypted data have a high entropy value. However, normal files like images and videos also exhibit high entropy, which

DAC '19, June 2–6, 2019, Las Vegas, NV, USA
© 2019 Association for Computing Machinery.
ACM ISBN 978-1-4503-6725-7/19/06…$15.00
https://doi.org/10.1145/3316781.3317889

makes entropy less effective. To address this, we leverage convolutional neural networks (CNN) to accurately identify only the data encrypted by cryptographic algorithms, filtering out normal data with high entropy. Unlike entropy that can be quickly computed with hardware logics [7], a CNN-based classifier takes a relatively long time. To lessen its overhead, RBlocker takes a *two-phase approach* which first examines an entropy value of input data and then applies the CNN-based classifier only to data with high entropy. To further minimize the detection cost without sacrificing the accuracy, RBlocker also employs a *file-based detection policy* that performs per-file basis detection, rather than small blocks.

Another challenge in designing RBlocker is that RBlocker cannot logically link encrypted data with its original data. If we can assume that all ransomware attacks simply overwrite original data with encrypted data, it is straightforward to figure out this linkage between the encrypted data and the original data. However, more complex ransomware variants, such as CryptoLocker, can execute out-of-place attacks which write encrypted data to a new file and delete original one [10]. In this case, RBlocker cannot tell which deleted data are the original data of the encrypted data. RBlocker overcomes this problem by employing a *time-out based backup policy*. When a file is deleted, RBlocker delays a deletion request by a given time-out threshold τ. After τ time units, RBlocker checks if there were any encrypted writes in the past τ units. If no encrypted writes were present, RBlocker concludes that it is safe to process the deletion request. If there were encrypted writes, RBlocker assumes that there was a ransomware attack and does not process the deletion request until no attack is guaranteed.

To evaluate the effectiveness of RBlocker, we have implemented it on an open SSD development platform [8]. To support the in-line detection of encrypted writes, we have implemented a hardware accelerator for the CNN-based detection. Our experimental results show that RBlocker defends all types of ransomware attacks, and incurs negligible overheads for data backup and detection.

The rest of the paper is organized as follows. In Section 2, we review NAND flash-based SSDs and present a key motivation of our work. Section 3 describes the protection mechanism of RBlocker. We present the design and implementation of RBlocker in Section 4, followed by its evaluation in Section 5. We conclude in Section 6.

2 BACKGROUND AND MOTIVATION

2.1 SSD-Level Ransomware Defence

Figure 1 depicts an architecture of a NAND flash-based SSD. Modern SSDs run an FTL, storage firmware, which is in charge of providing backwards compatibility to the block I/O interface. NAND devices do not allow in-place updates, so a page (a unit of reads and writes) cannot be overwritten before it is erased. The erasure operation is done in a unit of a block, which is a group of multiple pages. To hide such properties, an FTL always takes an *out-of-place update* strategy that writes incoming data to free pages, and for serving future reads, it maintains logical-to-physical (L2P) mappings to keep track of the locations of physical pages that are mapped to logical pages which are seen by the host.

In the example of Figure 1, when a logical page 0 (LP#0) is overwritten with new data A′, an FTL writes it to a free physical page 3 (PP#3), and updates the L2P mapping. After that, an FTL changes the status of the physical page 0 (PP#0) as invalid which

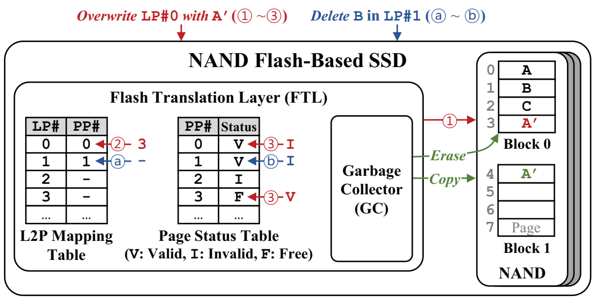


Figure 1: An overview of a NAND flash-based SSD.

was mapped to LP#0. FTL's GC module reclaims PP#0 later to create free space for future writes. Valid pages in a block selected for GC (e.g., PP#3 in block 0) must be copied to free pages in other blocks. To minimize GC costs, an FTL typically selects a block with the largest number of invalid pages as a victim. To further lessen GC overheads, a new command, called a trim, is recently added [6]. It allows SSDs to know which data are removed from file systems. For example, if data B is deleted, LP#1 is trimmed by the host and the FTL invalidates the corresponding PP#1. Without a trim support, PP#1 would be copied during GC since it is still considered valid.

The existing SSD-level approaches take advantage of the out-of-place update nature of an SSD. Invalid pages are permanently removed from NAND devices only when GC erase them. Thus, even when ransomware overwrites or deletes a file, an SSD is able to preserve its data by not allowing GC to erase corresponding pages. For example, in Figure 1, suppose that an FTL marks PP#0 and PP#1 as suspicious pages after they are (logically) overwritten with new data and trimmed. As long as it is not clear whether the pages are actually attacked by ransomware, GC copies them along with valid pages to a free block so as to use them for recovery later. Those suspicious pages will be permanently removed once no ransomware attack is (somehow) confirmed.

How long suspicious pages should be kept as valid in an SSD is an important factor in designing SSD-level techniques. If they are erased too shortly (e.g., after few minutes), users may lose a chance to recover original files after ransomware attacks. Keeping suspicious pages for a very long time (e.g., several weeks) is also not a reasonable choice because of its high space and GC overheads, in particular, when they are not actually attacked by ransomware. As shown in Figure 1, with an SSD-level protection, GC has to copy two more pages (i.e., PP#0 and PP#1), which negatively affects both the performance and lifetime of the SSD.

2.2 Limitations of Existing Approaches

Two existing approaches, FlashGuard [5] and SSD-Insider [3], choose one of the above two extremes, respectively. FlashGuard keeps all invalid pages for 2-4 weeks, if the pages have been read before invalidation. This is based on the fact that ransomware has to read and encrypt victim files. While it may be the most thorough way to protect user files against ransomware attacks, the space and GC overheads cannot be avoided. By analyzing block I/O traces collected from enterprise systems, FlashGuard's authors conclude that only few files are removed after being read. However, many counterexamples can be found in various scenarios. For a simple example, suppose that a user downloads, plays, and removes a video file. With FlashGuard, this file will not be removed from an SSD for a very long time, unnecessarily occupying huge space.

SSD-Insider [3] maintains suspicious data for a short period of time. It puts a ransomware detection algorithm to an FTL which is able to detect ransomware running in a host within 10 seconds by analyzing headers of block I/O requests. Suspicious pages which are invalidated 10 seconds ago can be garbage collected, because they are confirmed safe from ransomware attacks by the detector. Thus, SSD-Insider does not waste lots of free space to keep suspicious data, and does not seriously suffers from GC overheads. The detection algorithm of SSD-Insider, however, can be easily compromised by ransomware variants. For example, SSD-Insider guesses that there may be ransomware attacks if a sharp increase of overwrite requests from a host is observed for a short time interval. If ransomware intentionally slows down its encryption speed, SSD-Insider may fail to detect it, allowing GC to permanently erase victim data.

3 ENCRYPTION-AWARE PROTECTION

The proposed RBlocker overcomes the limitations of the existing solutions by leveraging another invariant feature of ransomware attacks: *encrypted writes*. Here, encrypted writes are defined to be write requests that carry data encrypted by cryptographic algorithms. All ransomware programs have to encrypt user files to block user access to them. Thus, if we can (1) detect encrypted writes from ransomware and (2) identify the locations of their victims, a more accurate selection of backup candidates is possible. In contrast to FlashGuard that has to back up all overwritten and deleted data, it may require a small amount of free space to keep suspicious data. Moreover, since encrypted writes are involved in all ransomware attacks, the selection accuracy of backup candidates would be maintained high, regardless of the types of ransomware.

3.1 Two-Phase Detection of Encrypted Writes

For an encryption-based protection to work, the detection of encrypted writes should be accurate enough to satisfy the following two conditions: *zero* false-negative-rate (FNR) and *near-zero* false-positive-rate (FPR). The FNR represents a rate of which RBlocker misjudges an encrypted write as non-encrypted one. If FNR > 0, we are not able to guarantee 100% full protections because GC may erase data manipulated by ransomware. On the other hand, the FPR indicates a rate of which RBlocker misjudges a non-encrypted write as an encrypted one. The higher FPR is, the more data RBlocker unnecessarily keeps for recovery, thus wasting precious free space.

To tackle this technical challenge, we propose two detection schemes: one based on *entropy* of incoming data and the other based on *convolutional neural networks* (CNN). The entropy-based detection is effective in achieving a zero FNR. To make users impossible to decrypt victim files, ransomware uses cryptographic algorithms, such as AES-128, whose resultants have extremely high entropy. Hence, just by setting a sufficiently low entropy value as a threshold, all data encrypted with such algorithms can be effectively detected. For example, based on the Shannon's entropy estimation whose resulting value ranges from 0 to 8 [12], encrypted data with cryptographic algorithms exhibit 7.99+ entropy values. If a threshold value is set to 7.9, we can achieve an effectively-zero FNR in detecting encrypted writes. Another benefit of the entropy-based detection is its high performance. By using hardware accelerators,

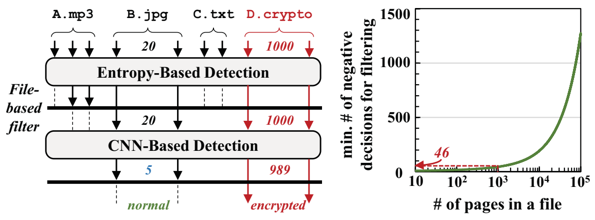


Figure 2: A two-phase detection with file-based filtering.

a throughput of 30 Gbit/s (≈ 3.75GB/s) is achievable [7], which is higher than the write bandwidth of recent high-speed SSDs [9].

The entropy-based detection is able to achieve an effectively-zero FNR, but it provides a high FPR ranging from 0.1 to 0.45 depending on target data. This is because of data that are not encrypted but have high entropy values, like media files and compressed files. Fortunately, those files are stored according to standard formats, and pieces of data belonging to the same file do not have high entropy all the time (owing to headers, footers, and metadata embedded in the middle of a file). This implies that if their unique signatures and patterns can be detected, we can distinguish their data from encrypted data with cryptographic algorithm. To this end, we devise a simple CNN-based detector with five convolution layers and two fully connected layers, which can provide a much lower FPR close to 0.05 through an intensive training with a large file set.

What we should consider next is the efficient integration of the entropy- and CNN-based detectors. It is infeasible to use the CNN-based detector as a primary detector, since its maximum throughput is limited to 16 MB/s even with an FPGA implementation. For this reason, we employ a two-phase approach. The entropy-based detector first filters out low entropy data with zero FNR. Then, it sends the rest to the CNN-based detector, so that normal files with high entropy can be excluded. In normal PC usage scenarios, only a small fraction of data (less than 10%) have higher entropy than 7.9, so its impact on performance would not be so severe.

3.2 File-Based Detection Policy

The two-phase detection has serious drawbacks. First, even though lots of data traffic is reduced by the entropy-based detector, CNN-based detector still could be a performance bottleneck, particularly when many media files are burstly written. We alleviate this problem by adopting a file-based detection policy, which is depicted in Figure 2 (left). It is motivated by an observation that ransomware encrypts victims on a file basis, and almost all data in a resultant file have high entropy values. A clustering of high entropy data within a single file, therefore, is a strong indicator that the file might be generated in ransomware attacks. If file data along with its inode number can be delivered to an SSD, the entropy-based detector is able to check whether the incoming file contains low entropy pages. If so, the entire writes on the file are simply ignored. In Figure 2, among four files, only two files, `B.jpg` and `D.crypto`, are passed to the CNN-based detector since low entropy writes (dashed lines) are found in `A.mp3` and `C.txt`.

Second, there is a possibility that the CNN-based detection makes a mistake by judging an encrypted file (e.g., `D.crypto`) as a normal file. This cannot be avoided because it is the nature of CNN-based algorithms. The probability is low (about 1.0+%) for 8-16 KB page, but it could be a serious security hole. We address this by extending the file-based detection policy. Under the extended, entire n writes on a file are considered non-encrypted, if more than k writes out of them are classified as non-encrypted ones. This probabilistic classification works well in practice. For example, with values of n = 1,000 and k = 46, we can decrease the FNR to less than 10^{-16}, the bit error rate of the standard SSD requirements [4]. In Figure 2 (left), `B.jpg` turns out to be non-encrypted since 15 normal pages are detected. More generally, given n, the minimum number of k that makes the FNR lower than 10^{-16} is plotted in Figure 2 (right) [1]. Note that when n is too small to make a probabilistic decision on a file, the CNN-based detector assumes that all the writes are encrypted.

To deliver inode numbers to an SSD, system layers, including a file system, a device driver, and a storage protocol, must be properly modified. However, it does not involve the changes of their principle designs. Moreover, considering the high flexibility of the recent NVMe layers, adding such a feature to an SSD would not be a serious obstacle in realizing the idea of the file-based detection.

3.3 Time-Out Based Backup Policy

Identifying victim data that are being replaced with encrypted writes is challenging due to various patterns of ransomware attacks. Most ransomware attacks remove victim files by overwriting them with encrypted ones (i.e., *in-place update* attack). In this case, we easily find the location of victim data because victim and encrypted data stored in the same logical address. However, some ransomware variants delete (trim) victim files before or after writing encrypted ones in a new file (i.e., *out-of-place* attacks). In such a case, there is no way to associate encrypted writes with victim data since they are located in different logical addresses.

Let's consider a following example. Suppose that ransomware performs out-of-place attacks on `LP#0`, and at the same time, `LP#1` is deleted by a user. Two trims for `LP#0` and `LP#1` are sent to an SSD, and the encrypted data of `LP#0` are written to a new logical page `LP#2` slightly later. The SSD receives two trims and one encrypted write, but cannot know which one, `LP#0` or `LP#1`, is associated with `LP#2`. Note that the arriving order of trim and write requests is exchangeable, depending on ransomware polices; the encrypted write at `LP#2` may arrive before the trim for `LP#0`.

We address this challenge by employing a time-out based backup policy. It temporarily backs up all the pages trimmed by the host. Then, if no encrypted writes are observed for a certain period of time (e.g., 1 hour), it allows an FTL to erase them. The rationale behind this policy is that ransomware tries to complete attacks as soon as possible (e.g., several minutes), in order not to be noticed by users [5]. Suppose that `PP#X` is trimmed at time t. If there are no encrypted writes between $(t - \Delta t_M)$ and $(t + \Delta t_M)$, it is safe to erase `PP#X`. Here, Δt_M is the maximum interval between a trim request and encrypted writes from the out-of-place attacks. If there are encrypted writes between $(t - \Delta t_M)$ and $(t + \Delta t_M)$, `PP#X` has to be kept in an SSD longer. Since Δt_M is different depending on variants, it should be sufficiently long. Currently, Δt_M is set to 1 hour. Considering that ransomware attacks finish in several minutes, it is chosen in a conservative manner.

[1]This graph is derived from the $P(X > k) = 1 - \sum_{i=0}^{k} \binom{n}{i} \times (0.01)^i \times (0.99)^{(n-i)} < 10^{-16}$, where X is the number of times that CNN-based detection misjudges an actual encrypted write as normal one in n tries.

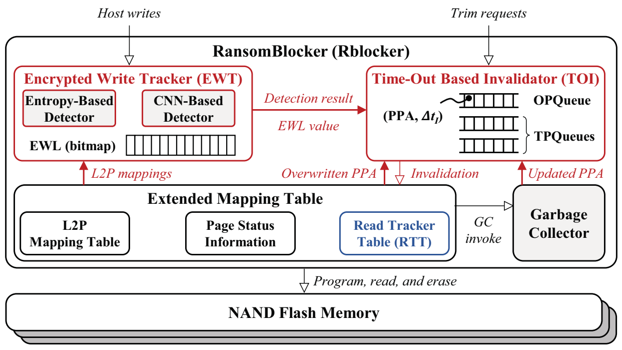


Figure 3: An organizational overview of RBlocker.

4 DESIGN OF RANSOMBLOCKER

Based on the detection and backup polices explained in Section 3, we have designed a low-overhead ransomware-proof SSD, called RBlocker. Figure 3 depicts an overall organization of RBlocker, which is based on an existing page-level FTL. The main additions for RBlocker are an Encrypted Write Tracker (EWT), a Time-Out based Invalidator (TOI), and a Read Tracker Table (RTT). The EWT is in charge of identifying and tracking encrypted writes using the hardware accelerators. The page backup for recovery is managed by the TOI. It invalidates all trimmed or overwritten pages only when they turn out to be safe. Same as FlashGuard, RTT is used to keep track of which physical pages have been read by the host, in order to exclude unread pages from backup candidates.

4.1 Encrypted Write Tracker

It is straightforward to detect potential in-place attacks of ransomware. With the detection technique explained in Section 3, the EWT can identify whether an incoming overwrite is encrypted. If so, the information required for recovery (e.g., locations of victim to be overwritten) is delivered to TOI, so that the invalidation is delayed until its safety is confirmed. Detecting potential out-place attacks, however, is rather complicated, because of its time-out based backup policy. For all the pages trimmed by the host, the EWT sends their information to the TOI. Moreover, for every new write located to free page, it is necessary to keep track of its status (encrypted or not), along with a timestamp, for at least Δt_M period. Obviously, maintaining all those information could be a serious burden, so the EWT takes an approximate approach.

Figure 4 illustrates how the EWT deals with the history of encrypted writes. The EWT divides the monitoring period, Δt_M, into multiple time frames. Then, it maintains a linear bitmap table, where each bit entry indicates whether encrypted writes occur for a corresponding time frame. If Δt_M is 1 hour and a time frame is 1 minute,

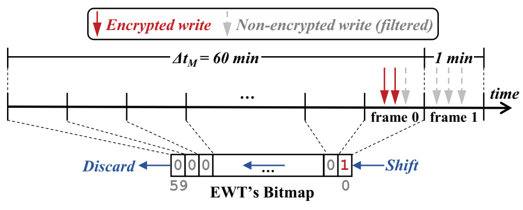


Figure 4: An encrypted write tracking in the EWT.

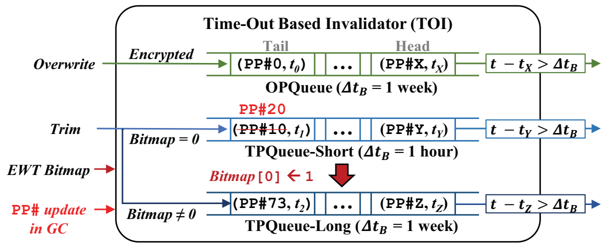


Figure 5: An operational overview of the TOI module.

the EWT maintains a 60-bit long bitmap. Suppose that, at the time frame 0, more than one encrypted writes are observed. If it is, the LSB entry in the bitmap is set to 1. After the time frame 0 is expired, the EWT shifts the bitmap by one bit left. If no encrypted writes are observed during the time frame 1, the new LSB entry is set to 0. An overflowed bit is simply discarded because we are not interested in the information of time frames older than Δt_M.

4.2 Time-Out Based Invalidator

The TOI is responsible for deciding how long data invalidation should be delayed. Figure 5 depicts how the TOI works. The TOI receives the information of potential victim pages from the EWT, which includes their physical locations. Upon receiving, the TOI first sees if victims were read by the host before. If not, it ignores them because they are safe from ransomware attacks. Otherwise, the TOI puts a pair of victim's physical location and a current timestamp (e.g., (`PP#0`, t_0) in Figure 5) into one of the three queues according to their types (explained below). Note that data that are put into a queue are still valid in an SSD. Each queue has a different backup time, Δt_B. The TOI monitors items in the queues regularly. If one that stays longer than Δt_B is found (e.g., $t - t_X > \Delta t_B$ in Figure 5, where t is current time), the TOI removes it from the queue, invalidating its page in an SSD.

Overwritten victim pages are pushed to OPQueue (Overwritten Page Queue). Δt_B for OPQueue is set long enough (e.g., 1 week). On the other hand, trimmed pages are put into either TPQueue-Short or TPQueue-Long (note: TPQueue is an abbreviation for Trimmed Page Queue). If the EWT bitmap is 0, it means that there have no encrypted writes for the past Δt_M period. Since trimmed pages are yet safe from ransomware attacks, they are put into TPQueue-Short with a short Δt_B (e.g., 1 hour). If the bitmap is larger than 0, it implies that encrypted data were written before pages are trimmed. Thus, they are pushed to TPQueue-Long with longer Δt_B, says a week. Items staying in TPQueue-Short should be moved to TPQueue-Long if encrypted writes occur. The TOI merges two TPQueues by moving items in TPQueue-Short to the tail of TPQueue-Long.

4.3 Garbage Collection

Garbage collection (GC) in RBlocker is performed in a very similar way as that of typical SSDs. The only additional task RBlocker's GC module should take care of is to update physical locations of victim pages in the TOI's queues when their locations have changed. This happens when GC copies victim pages in a NAND block to another one. For example, in Figure 5, if GC moves data of `PP#10` to `PP#20`, the new location should be provided to the TOI so as to update the relevant information in the queue (i.e., (`PP#10`, t_1) to (`PP#20`, t_1)). Knowing which physical page is in the queues is easy. During GC, RBlocker can retrieve the logical page number of a physical page to move by reading its spare (out of band) area. If this page is marked valid (in the page status table) but the mapping entry points to a different physical page, it means that the page is backed up for recovery purpose.

4.4 Discussion

Recovery Issue: If files are infected by ransomware, they can be recovered by using a recovery tool for RBlocker. There are two key issues. The first is how to perfectly reconstruct original files from backup data kept in RBlocker. It is straightforward when files are destroyed by the in-place attacks, since logical page numbers of victim files can be used to find matched physical pages in RBlocker. However, in case of the out-of-place attacks, more efforts are required because infected files were written in different locations. For recovery, it may be necessary to use techniques adopted in existing forensic tools. Fortunately, RBlocker is able to group victim pages belonging to the same file by using an inode number obtained from the file system, which makes it easier for us to reconstruct original files. The second is that a recovery tool, which runs a host system, can be abused by virus software. This problem can be avoided by running a tool on a clean system that is not compromised by virus.

HW Accelerator: Adding HW accelerators inevitably increases the price of an SSD. In our experience, the CNN-based detector implemented using Xilinx's CHaiDNN utilizes about 80% of ZCU102's FPGA fabrics, requiring considerable hardware resources. However, considering the recent advance of embedded AI accelerators (e.g., Edege TPU and Jetson), running the CNN-based detector inside an SSD will be a feasible design choice in the near future.

5 EXPERIMENTAL RESULTS

5.1 Experimental Settings

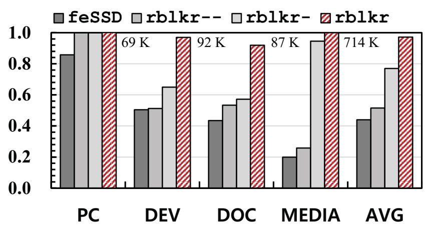
We have implemented RBlocker using an open flash development platform [8]. It supports 512 GB device capacity in maximum, but we limited the SSD capacity to 32 GB for fast evaluations. Our SSD platform consisted of 8 channels, each of which had 8 NAND chips. Each chip had 512 NAND blocks with 256 4-KB NAND pages.

For experiments, we have developed three in-house ransomware variants by extending open ransomware [1, 2]. Each ransomware behaved differently in terms of attack patterns; the first performs in-place attacks; the second performs out-of-place attacks with a 'delete-before-write' order; and the last performs out-of-place attacks as well, but with a 'delete-after-write' order. They used different cryptographic algorithms, AES, DES, and RC, respectively, which were often used by well-known ransomware [3, 5, 10].

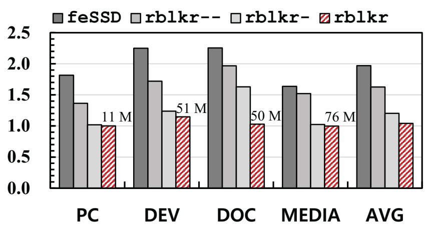
Four distinct real-world workloads were used for our evaluation. Table 1 summarizes their important characteristics (I/O intensiveness and read/write ratio). PC and DEV were collected from general PC usages (e.g., documenting, installing programs, etc.) and from development servers, respectively. DOC and MEDIA were generated by Filebench [13] that emulated file servers that store documents (e.g., `.pdf`, `.docx`, etc.) and media files (e.g., `jpg`, `png`, `mp3`, and

Table 1: I/O characteristics of four benchmark workloads.

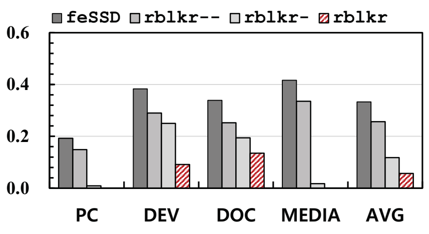
workload	PC	DEV	DOC	MEDIA
read:write	1:4	1:2	1:3	4:1
I/O intensiveness	low	moderate	moderate	high



(a) Normalized IOPS.



(b) Normalized block erasure counts.



(c) Proportion of backed up pages.

Figure 6: Comparisons of the impact of different techniqeus on the SSD performance and lifetime.

mp4), respectively. We extended Filebench to issue I/Os with actual contents of 40-GB documents and 20-GB media files.

We compared RBlocker with a FlashGuard-enabled SSD (feSSD), which was configured to keep backup data for 1 week. We also compared three versions of RBlocker; one only with the entropy-based detection (rblkr--), another with the entropy- and file-based detection (rblkr-), and the other with every detection approach (rblkr). The throughputs of hardware accelerators in RBlocker were emulated by software based on the numbers reported by [7] (for encryption-based one) and our FPGA implementation (for CNN-based one). Finally, we set Δt_M and Δt_B of all the RBlocker variants to 1 hour and 1 week, respectively.

5.2 Evaluation Results

To compare the impact of RBlocker on the SSD performance and lifetime, we measured IOPS values and block erasure counts as shown in Figure 6(a) and (b), respectively. IOPS values and block erasure counts were normalized by those of an unmodified SSD with no ransomware protection policy. All the three versions of rblkr outperformed feSSD. FeSSD provided comparable performance only under PC which exhibited a low I/O intensiveness and thus was less affected from GC. As shown in Figure 6(b), the block erase count of feSSD was significantly increased by up to 124% over the unmodified SSD, because of a large amount of data unnecessarily backed up. While rblkr achieved almost the same performance and lifetime over unmodified SSD, rblkr- and rblkr-- incurred non-trivial the performance and lifetime loss for some workloads. In particular, IOPS values of rblkr-- was almost the same as those of feSSD under most workloads.

To better understand how effectively RBlocker reduced backup overheads in detail, we measured the proportion of backup pages as shown in Figure 6 (c). Under every workload, the space overhead of RBlocker was less than 13% of the total SSD capacity, while that of feSSD was up to 41%. This implies that our two-phase detection was very effective in achieving a low FPR, even guaranteeing an effective-zero FNR. In particular, RBlocker effectively filtered out multimedia files that have high entropy but were not encrypted. As shown in results from MEDIA, almost all of writes to multimedia files were filtered by the entropy-based detector. Considering a huge difference between rblkr- and rblkr--, we found that the file-based detection was effective. The CNN-based detector showed consistently good performance across all the workloads.

Finally, we evaluated the backup accuracy of RBlocker. To this end, we ran our in-house ransomwares under all the workloads. Once ransomware attacks were finished, we read all the pages kept in the TOI queues, and checked whether all the victim data were successfully backed up. For a thorough assessment, we repeated the above steps 100 times, and confirmed that, for all the cases, RBlocker was able to successfully keep all of the original pages.

6 CONCLUSIONS

We have presented RBlocker which provided a 100% full protection against ransomware attacks. By examining host writes on the SSD side with HW accelerated detectors, RBlocker can selectively back up only the data which are highly likely to be attacked by ransomware. As a results, RBlocker showed almost the same performance and lifetime over an SSD without ransomware protection.

We plan to extend RBlocker so that it can be efficiently integrated with AI acceleration chips (e.g., Edge TPU) for more quick and accurate ransomware detection. Developing an efficient and secure recovery tool is also one of our future plans.

ACKNOWLEDGEMENTS

This research was supported by the National Research Foundation of Korea (NRF) grant funded by the Ministry of Science and ICT (MSIT) (NRF-2018R1A2B6006878 and NRF-2018R1A5A1060031). The ICT at Seoul National University provided research facilities for this study. (*Co-corresponding Authors: Sungjin Lee and Jihong Kim*)

REFERENCES

[1] A POC Windows Crypto-Ransomware, https://github.com/mauri870/ransomware, 2018.
[2] Virtual Gangster, https://github.com/roothaxor/Ransom, 2018.
[3] S. Baek et al. SSD-Insider: Internal Defense of Solid State Drive Against Ransomware with Perfect Data Recovery. In *Proceedings of the IEEE International Conference on Distributed Computing Systems*, 2018.
[4] A. Cox. JEDEC SSD Endurance Workloads. In *Proceedings of the Flash Memory Summit*, 2011.
[5] J. Huang et al. FlashGuard: Leveraging Intrinsic Flash Properties to Defend Against Encryption Ransomware. In *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security*, 2017.
[6] G. Kim et al. Performance Analysis of SSD Write Using TRIM in NTFS and EXT4. In *Proceedings of the IEEE Conference on Computer Sciences and Convergence Information Technology*, 2011.
[7] Y.-K. Lai et al. Hardware-Assisted Estimation of Entropy Norm for High-Speed Network Traffic. *Electronics Letters*, 50(24):1845–1847, 2014.
[8] S. Lee et al. Application-Managed Flash. In *Proceedings of the USENIX Conference on File and Storage Technologies*, 2016.
[9] G. Santos. SSD Ranking: The Fasted Solid State Drives, 2018.
[10] N. Scaife et al. CryptoLock (and Drop It): Stopping Ransomware Attacks on User Data. In *Proceedings of the IEEE International Conference on Distributed Computing Systems*, 2016.
[11] H. N. Security. Ransomware Back in Big Way, 181.5 Million Attacks since January, 2018.
[12] C. E. Shannon. A Mathematical Theory of Communication. *ACM SIGMOBILE Mobile Computing and Communications Review*, 5(1):3–55, 2001.
[13] A. Wilson. The New and Improved FileBench. In *Proceedings of the USENIX Conference on File and Storage Technologies*, 2008.