FlashGuard: Leveraging Intrinsic Flash Properties to Defend Against Encryption Ransomware

Jian Huang
Georgia Institute of Technology
Atlanta, GA
jian.huang@gatech.edu

Jun Xu
Pennsylvania State University
University Park, PA
jxx13@ist.psu.edu

Xinyu Xing
Pennsylvania State University
University Park, PA
xxing@ist.psu.edu

Peng Liu
Pennsylvania State University
University Park, PA
pliu@ist.psu.edu

Moinuddin K. Qureshi
Georgia Institute of Technology
Atlanta, GA
moin@ece.gatech.edu

ABSTRACT

Encryption ransomware is a malicious software that stealthily encrypts user files and demands a ransom to provide access to these files. Several prior studies have developed systems to detect ransomware by monitoring the activities that typically occur during a ransomware attack. Unfortunately, by the time the ransomware is detected, some files already undergo encryption and the user is still required to pay a ransom to access those files. Furthermore, ransomware variants can obtain kernel privilege, which allows them to terminate software-based defense systems, such as anti-virus. While periodic backups have been explored as a means to mitigate ransomware, such backups incur storage overheads and are still vulnerable as ransomware can obtain kernel privilege to stop or destroy backups. Ideally, we would like to defend against ransomware without relying on software-based solutions and without incurring the storage overheads of backups.

To that end, this paper proposes FlashGuard, a ransomware-tolerant Solid State Drive (SSD) which has a firmware-level recovery system that allows quick and effective recovery from encryption ransomware without relying on explicit backups. FlashGuard leverages the observation that the existing SSD already performs out-of-place writes in order to mitigate the long erase latency of flash memories. Therefore, when a page is updated or deleted, the older copy of that page is anyway present in the SSD. FlashGuard slightly modifies the garbage collection mechanism of the SSD to retain the copies of the data encrypted by ransomware and ensure effective data recovery. Our experiments with 1,447 manually labeled ransomware samples show that FlashGuard can efficiently restore files encrypted by ransomware. In addition, we demonstrate that FlashGuard has a negligible impact on the performance and lifetime of the SSD.

CCS '17, October 30-November 3, 2017, Dallas, TX, USA

ACM ISBN 978-1-4503-4946-8/17/10…$15.00
https://doi.org/10.1145/3133956.3134035

CCS CONCEPTS

• **Security and privacy** → systems security; hardware security;

KEYWORDS

Encryption ransomware; solid-state drive; data recovery

1 INTRODUCTION

Recently, criminals are unleashing brash attacks on users' machines through a new type of malicious software called encryption ransomware [34, 36, 53]. For example, the WannaCry ransomware [53] launched on May 12, 2017 (one week before the CCS submission deadline) has infected more than 230,000 computers across 150 countries. Among the victims are government agencies, schools, hospitals, and police departments.

Different from traditional malware which typically disrupts computer operations and gathers sensitive information, encryption ransomware stealthily encrypts the files on user's machine and demands users pay a ransom to restore the files. Since the operations performed by ransomware are indistinguishable from benign software, ransomware can easily bypass various antivirus, making it increasingly prevalent in cyber criminals [30, 46]. According to a study from IBM Security [17], the number of users who came across encryption ransomware in 2016 increased by more than 6,000% over the previous year. The ransomware attacks cost their victims about a billion dollars in 2016 which is a 41x increase compared to the cost in all of 2015 [13].

To counteract ransomware, researchers have proposed several detection systems that use file access patterns [19, 44] or features of cryptographic algorithms [9] to identify ransomware. However, these detection mechanisms still cannot prevent ransomware from locking up user data. First, existing ransomware detection occurs only after observing the actual damage. Given that the encrypted data may contain the files considered to be valuable, victims still have to shoulder the burden of paying the ransom. Second, some ransomware can run with administrator privileges, which allow them to load kernel code and carry out kernel-level attacks. Given that the existing defense systems typically run within the kernel, ransomware can easily disable or work around the aforementioned detection mechanisms.

To address these issues, one instinctive solution would be to enable file backup on local persistent storage (e.g., journaling and log-structured file systems [32, 40]) or remote machines (e.g., NFS [31] and cloud-based storage [12, 51]). However, this is insufficient at guarding against ransomware. First, any file backup mechanisms inevitably impose storage overhead. Second, ransomware may find and jump to the backup and encrypt it regardless of whether it is on shared network drives, local hard disk drives, external storage devices, or plugged-in USB sticks [55]. Third, ransomware with the kernel privilege can also terminate backup processes, making them futile against ransomware defense.

As the replacement to conventional persistent storage devices – hard disk drives (HDDs), Solid-State Drives (SSDs) have been widely used on many kinds of computing platforms, because they provide orders of magnitude better performance than HDDs while their cost is fast approaching to that of HDDs [14, 16, 47, 54]. A unique property of SSDs is that a physical page cannot be written until it is erased, however the erase operation incurs significantly longer latency. To overcome such a shortcoming, modern SSD performs out-of-place write for every write. Therefore, SSDs intrinsically support the logging functionality without requiring an explicit backup. Such a feature will naturally preserve the old copies of overwritten or deleted files for a period of time before they are reclaimed by the process of garbage collection. Moreover, the firmware-level logging could isolate the data protection and recovery from operating system (OS) kernels and upper-layer software.

Unlike existing ransomware detection systems [44, 45] and explicit file backups [33, 41], we take advantage of the intrinsic flash properties and build a ransomware-tolerant SSD named FlashGuard, which has a *lightweight hardware-assisted* data recovery system. It allows users to reinstate the data held in captivity by ransomware.

While the proposed system is based on the out-of-place write characteristic of an SSD, it is challenging to leverage such a feature for data recovery for two major reasons. First, once data is deleted or overwritten but gets left behind on the drive, SSD controller may perform garbage collection (GC) to erase the blocks taken up by such stale data for free space. Given that stale data may contain the original data copies "deleted" or "overwritten" by ransomware, FlashGuard needs to hold stale data and prevent GC from discarding them. Since holding too much stale data could increase the GC overhead, which further affects the performance of regular storage operations significantly [3] and even jeopardizes the SSD lifetime [1], an efficient GC mechanism is desirable. Second, we must guarantee that the change to the GC is resistant to the potential attacks against SSDs from the ransomware running with the kernel or administrator privilege.

To tackle these challenges, we implemented FlashGuard's data recovery system in SSD firmware by augmenting GC mechanism with the ability to only hold the data potentially deleted or overwritten by ransomware. We prototyped FlashGuard on a 1 TB programmable SSD with minimal modifications to the exiting SSD design. Using a real world set of 1,477 distinct ransomware samples covering 13 families, we show FlashGuard can quickly recover the files held by ransomware. For example, we demonstrate FlashGuard can restore 4 GB of encrypted data in 30 seconds. Using a set of publicly available storage traces, we extensively evaluated the impact of FlashGuard upon the storage performance. Our experimental results show that FlashGuard incurs negligible performance overhead (up to 6%) and has trivial impact (less than 4%) on SSD lifetime.

To the best of our knowledge, FlashGuard is the first defense scheme that can efficiently offset the damage of ransomware to user data even if ransomware run with administrator privileges to load kernel code or exploits a kernel vulnerability. Overall, this paper makes the following contributions:

- We conduct a study of more than a thousand ransomware samples and find meaningful insights on their characteristics of the encryption time and backup spoliation behavior.
- We propose a ransomware-tolerant SSD, FlashGuard, which has a firmware-level recovery system to defend against encryption ransomware by leveraging the inherent out-of-place write property in existing SSDs.
- We design and implement FlashGuard in a real programmable SSD and demonstrate that FlashGuard can rapidly restore encrypted data with a large set of ransomware samples.
- We perform extensive evaluations with acknowledged storage traces collected from different real-world applications and show that FlashGuard has negligible negative impact on storage performance and SSD lifetime.

The rest of this paper is organized as follows. § 2 characterizes encryption ransomware. § 3 discusses the threat model. We introduce the background in § 4. § 5 presents the design and implementation of FlashGuard, followed by its evaluation in § 6. § 7 discusses the possible attacks against FlashGuard and their solutions. § 8 summarizes the related work. We conclude our work in § 9.

2 RANSOMWARE STUDY

Among various strains of ransomware, encryption ransomware is the most common type that encrypts user data and demands money in exchange for decrypting them. The objective of this work is to design and develop a ransomware-tolerant SSD which has the data-recovery capability to offset the damage to user data resulting from encryption ransomware. To achieve this, we first analyze the behaviors of encryption ransomware and understand how they interact with user data by conducting a study on a large number of ransomware samples. Different from prior studies on ransomware [19, 20, 44], our study focuses on two aspects – *encryption time* and *backup spoliation*.

2.1 Study Methodology

We gathered 1,477 encryption ransomware samples from VirusTotal [52] and classified them into 13 distinct ransomware families based on the ransom notes they present to victims. Table 1 illustrates these families, their encryption strategies and the number of samples in each ransomware family.

Following the common scientific guidelines [10], we executed each ransomware sample within a virtual machine (VM) running 64-bit Windows 7 SP1 with 2 CPU cores and 4 GB main memory on a host machine (configured with 2.67 GHz Intel quad-core Xeon processor and 8 GB DRAM). We removed the barriers of ransomware

[1] we do not deem `Petya` ransomware that it deletes backups because `Petya` demolishes and replaces Windows file system.

Table 1: **Ransomware families, their encryption time, and behaviors of deleting backup files (backup spoliation).**

Family	Samples		Encryption		Backup spoliation
	Num	%	Target	T (*mins*)	
Petya[1]	14	0.95	MFT	2	✯
CTB-Locker	119	8.05	Files	14	✗
JigSaw	5	0.34	Files	16	✗
Mobef	7	0.47	Files	16	✗
Maktub	10	0.68	Files	22	✓
Stampado	42	2.84	Files	27	✗
cerber	29	1.96	Files	37	✓
Locky	344	23.29	Files	43	✓
7ev3n	16	1.08	Files	44	✓
TeslaCrypt	75	5.08	Files	44	✓
HydraCrypt	13	0.88	Files	70	✓
CryptoFortress	4	0.27	Files	75	✓
CryptoWall	799	54.10	Files	75	✓
Total	1477	100	–	–	–

execution by disabling protection services such as firewall, Microsoft security protection, and user account control. Moreover, we granted all ransomware samples the administrator privilege. Since ransomware might perform key-exchange with the control server and establish those encryption keys used for locking up user data, we enabled the access to the Internet. However, considering ransomware may attempt to propagate themselves, we used a filtered host-only adapter to control their traffic and minimize their impact upon the host. After executing each ransomware, we revert the VM to a clean snapshot.

We conduct two experiments to measure ransomware's encryption time and examine whether ransomware attacks backup files (e.g., Volume Shadow Copies [42]) respectively. We describe their experimental setups as follows:

Encryption time. We placed a set of files (9,876 files in total) following the file-type distribution in a normal user's computer [11] in each VM. Table 2 shows the distribution of these files covering more than 18 unique file types. We run each ransomware sample and use the screenshot method described in [19] to examine their execution time. Specifically, we detect the changes to the screen of the virtual machine, screenshot the ransom notifications, and calculate the time it took for a ransomware to encrypt files and display a message on the screen to notify victim. To avoid false positives, we disabled Windows notification and manually examined each screenshotted notification.

Backup spoliation. To determine whether a ransomware also attacks file backups such as the volume shadow copies, we created and enclosed several volume shadow copies on VMs. We deem a ransomware sample targets at backups if we observe the disappearance of these shadow copies.

2.2 Our Findings

Table 1 describes how fast ransomware encrypts data and notifies victim with a ransom screen (the 5th column), and whether ransomware attacks file backups (the last column). According to our study, ransomware typically displays ransom screen immediately after the encryption (sometimes even before the encryption has been completed). The notification procedure takes little time and most of the execution time of encryption ransomware is spent on the encryption part.

Table 2: **File distribution in a normal user's computer.**

Type	Number		Size		
	Num	%	Avg (KB)	Total (MB)	%
pdf	2378	24.08	565.27	1312.70	30.28
html	2117	21.43	59.15	122.29	2.82
jpg	1073	10.86	335.08	351.12	8.10
doc	797	8.07	361.92	281.69	6.50
txt	788	7.98	553.89	426.23	9.83
xls	584	5.91	587.68	335.16	7.73
ppt	501	5.07	2110.94	1032.80	23.82
xml	353	3.57	132.59	45.71	1.10
gif	349	3.53	81.64	27.83	0.64
ps	208	2.11	764.85	155.36	3.58
csv	188	1.90	202.77	37.23	0.86
gz	128	1.30	628.64	78.58	1.81
log	99	1.00	170.80	16.51	3.81
unk	59	0.60	358.53	20.66	4.77
eps	40	0.41	516.59	20.18	4.66
png	39	0.39	312.85	11.92	2.75
others	141	1.77	343.62	58.72	1.35
Total	9876	100	449.44	4334.67	100

We observed that ten families complete the file encryption in less than an hour. For ransomware `CTB-Locker`, `JigSaw`, `Mobef` and `Petya`, their encryption takes even less than 20 minutes. Moreover, we discovered that some ransomware encrypt only small files or files with certain extensions. For example, `JigSaw` encrypts only files smaller than 10 MB, `CTB-Locker` only locks up files with certain extensions and `Petya` only encrypts a system's Master File Table (MFT) [27].

> **Observation 1:** Ransomware typically locks up data rapidly and the size of the data encrypted is relatively small.
> **Implication:** Ransomware would like to minimize the chances of being terminated and caught, or ransomware authors may want to collect ransom quickly.

Table 1 also shows that eight ransomware families attempt to delete backup files. Recall that we assigned ransomware samples the administrator privilege, which grants the ransomware the permission to destroy backups. We observed that some ransomware families attempt to bypass User Access Control if the privilege of deleting the backup files is not given. For instance, `cerber` [4] firstly escalates its privilege and then deletes Shadow Copies using the WMIC utility [56].

> **Observation 2:** Ransomware variants proactively try to remove any means that victims could have to recover from the attack without paying the ransom.
> **Implication:** Ransomware can obtain kernel privilege to terminate or destroy software-based defense systems such as explicit data backups.

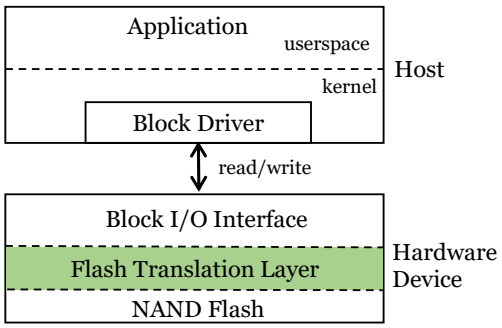


Figure 1: **The system architecture of using an SSD. The main idea of FlashGuard is proposed and implemented in the Flash Translation Layer (FTL) in SSD firmware.**

3 THREAT MODEL

As this work focuses on defending against encryption ransomware, we exclude the damage caused by non-encryption ransomware because they typically lock a computer system in a way which is not difficult for a knowledgeable person to reverse. For example, the ransomware `Trojan WinLock` trivially restricts access to the computer system and asks users to pay ransom to receive a code for unlocking their machines. In addition, we assume that encryption ransomware must be capable of restoring user data because inaccessibility and non-recoverability after paying ransom can significantly influence the rewards of ransomware attacks.

In this work, we only consider the situation where data on persistent storage are overwritten or deleted by ransomware. The targets not only include the files created by user-level applications (e.g., `.docx` and `.zip`) but also the metadata files that are required for file systems (e.g., Master File Table).

As discussed in § 2, some ransomware (e.g., `cerber`) will try to elevate its privileges to run as administrator. Once the privilege is given, the ransomware can disable or terminate any kernel-level defense mechanisms. As such, we do not assume the OS is trustworthy. Rather, we trust the SSD firmware. We believe this is a realistic assumption because (1) firmware is located within a storage controller, making it hardware-isolated to ransomware processes; (2) in comparison with the OS kernel, firmware has a small Trusted Computing Base (TCB) typically less vulnerable to malware attacks.

Overall, we believe this is a realistic threat model. First, it considers all types of ransomware attacks that aim to encrypt user data. Second, this threat model covers the cases in which the OS kernel is compromised such as WannaCry [53]. With the advance in ransomware defense, we believe ransomware authors will also actively exploit the vulnerabilities in the OS kernel. To the best of knowledge, this is the first work that explores malware defense solutions at the firmware level.

4 APPROACH OVERVIEW

In this section, we briefly describe the technical background on SSDs and then discuss how the intrinsic properties of SSDs can be leveraged for building a lightweight data recovery system to protect against encryption ransomware.

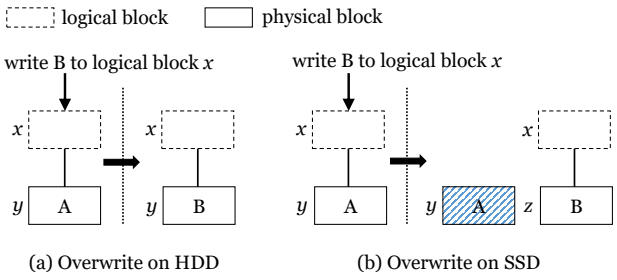


Figure 2: **The fundamental difference between HDD and SSD for an overwrite operation. When a logical block *x* is overwritten, HDD will update the mapped physical block *y* with the new data *B*, while SSD will place the new data *B* on a free block *z* and garbage collect the block *y* later.**

4.1 Technical Background in SSDs

Same as conventional HDDs, a commodity SSD employs a block interface to encapsulate the idiosyncrasies of flash devices (see Figure 1). As such, it gives upper-level software systems (such as file systems) an impression that both SSD and HDD perform storage operations in the same manner. At the hardware level, however, an SSD is fundamentally different from HDD which physically overwrites data on disks when a logical overwrite occurs (as shown in Figure 2-a).

Given an SSD, each physical page can be written only after it is erased. Unfortunately, erase operation can be performed only at block (which has multiple pages) granularity and such operations are time-consuming. Therefore, SSDs issue the writes to free pages which have been erased in advance (i.e., out-of-place write) rather than waiting for the expensive erase operation for every write, and GC will be executed later to clean the stale data on SSDs. Moreover, each flash block has limited endurance: it is rated only for a few thousands erase operations, therefore it is important for the blocks to age uniformly. SSDs employ both out-of-place write and GC to overcome the shortcomings of SSDs and maintain indirections in the Flash Translation Layer (FTL) for indexing the virtual-to-physical address mapping.

For a logical overwrite on SSD, the data is written to a free block which has already been erased, the old copies become invalid and are garbage collected later (see Figure 2-b). Details about the FTL logic will be discussed in § 5.

4.2 Rationale

To demand ransom, ransomware typically overwrites user files with encrypted contents. As described in § 4.1, SSDs naturally hold the old copies of the data overwritten by upper-level programs. As such, SSDs can be devised as a recovery system that holds data potentially manipulated by ransomware. Moreover, SSDs have an indirection layer at the firmware level to manage data. Building a recovery system on top of it, we can naturally isolate our recovery system from the OS, making it resistant to the attacks typically launched by malware to evade anti-virus. Taking advantage of the intrinsic characteristics of SSDs, we can also minimize the code space of our recovery system. As a result, SSDs naturally reduce the attack surface of our recovery system.

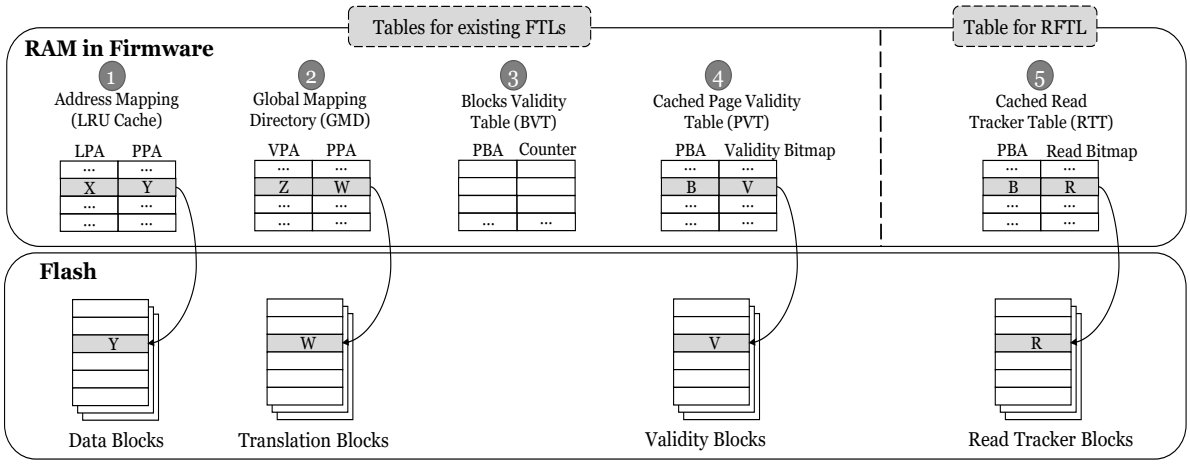


Figure 3: **Overview of RFTL in FlashGuard. RFTL slightly modifies the existing FTLs by adding a read tracker table (RTT) to track whether a page has been read. Cooperating with other tables, RTT helps RFTL track the pages that could be encrypted by ransomware. LPA*: logical page address,* PPA*: physical page address,* VPA*: virtual page address,* PBA*: physical block address.***

5 DESIGN AND IMPLEMENTATION

FlashGuard consists of two major components: a Ransomware-aware Flash Translation Layer (RFTL) and a tool for data recovery. The RFTL is designed for holding data potentially overwritten by encryption ransomware. The recovery tool is for victims to offset the damage to their files when they are aware of ransomware infection. In this section, we present the design and implementation of FlashGuard in details.

5.1 Ransomware-Aware FTL

The FTL in modern SSDs maintains four data structures (see (1)(2)(3)(4) in Figure 3) to support out-of-place write and GC functionalities in practice. For each I/O access, the address mapping table (1) is checked to translate the logical page address (LPA) to physical page address (PPA)[2]. For performance reason, the recently accessed mapping table entries (1) are stored in a cache (using LRU policy in RFTL) located in a small and fast SRAM. If a mapping entry is not cached, FTL will check the Global Mapping Directory (GMD) (2) to locate the corresponding translation page, and place the mapping entry in the address-mapping cache.

After certain storage operations, some pages in flash blocks may become invalid. To assist the GC operation, FTL usually uses the Block Validity Table (BVT) (3) to track the number of the valid pages in each block and to determine whether the block should be garbage collected or not. Since BVT is indexed in block-level granularity, it is small and can be fully stored in SRAM. Once a block is selected as the GC candidate, the Page Validity Table (PVT) (4) will be accessed to check which pages are valid and should be moved to a new flash block. The PVT could be a conventional page validity bitmap (PVB) or a recent optimized version which uses a log-structured merge-tree to reduce the space requirement of indexing the bitmap for each physical block [8]. In this work, we adopt the latter design for obtaining better performance. We will use examples (see § 5.2 and § 5.3) to illustrate how these data structures work collaboratively with other components in FlashGuard.

To augment an SSD with the capability of counteracting ransomware attacks, a straightforward solution is to keep all the invalid pages in the physical device until ransomware is detected. This is infeasible for two major reasons. First, an SSD would quickly fill up with stale data, making the SSD unusable and causing unacceptable resource inefficiency. Second, the GC operations will be executed much more frequently to compact and collect free blocks, which affects the storage performance significantly.

Therefore, it is desirable that SSDs only hold the invalid pages having the old versions of the data manipulated by encrypted ransomware. According to our study (§ 2) and CryptoDrop [44], the size of the data encrypted by ransomware is typically less than a gigabyte. Holding such a small dataset will have negligible impact on a commodity SSD which usually has TBs of storage capacity.

However, it is challenging to track the pages manipulated by encryption ransomware since the underlying FTL does not have any semantic information of the received storage commands. To overcome this, we propose the Ransomware-aware FTL to track the invalid pages that could result from ransomware. RFTL augments the conventional FTLs with only one additional data structure: the **Read Tracker Table (RTT)** (5), which requires minimal modification to the existing firmware implementation.

We propose RTT based on the insights that ransomware typically read user data from disk, encrypt it and then overwrite or delete the original copy [19, 44]. Therefore, if a page has been read and then become invalid later, it could be a victim page encrypted by ransomware. We use the RTT (5) to track the page that has been read and leverage the PVT (4) to check whether it is valid or not. RTT and PVT provide us the hints to decide whether the page should be reained or not.

[2]The mapping table can be managed in page-level, block-level or hybrid block/page granularity. FlashGuard uses fine-granular and fully-associative page-level mapping. We believe it also works for other two mapping schemes.

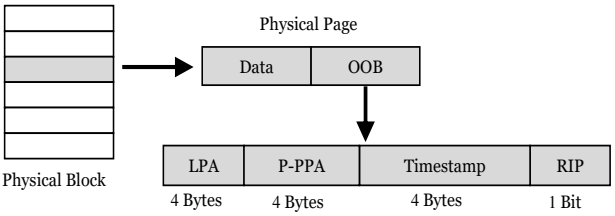


Figure 4: **The structure of the out-of-band (OOB) metadata in each physical page. It includes the LPA mapped to this physical page, the previous physical page address (P-PPA) mapped to the current LPA, the timestamp when the page is written, the retained invalid page (RIP) bit indicating whether this page should be retained if it becomes invalid.**

The RTT organizes entries in the way of the PVT (4), except that each entry in the RTT is a read bitmap[3] indexed by a block address. With the same optimization used in PVT, RTT enables RFTL to access and update the bitmap in an efficient manner. We use a buffer (4 KB in RFTL) to cache the frequently accessed RTT entries, which introduces only a small storage overhead in SRAM.

5.2 Read and Write Operations in RFTL

In this section, we describe how RFTL performs I/O requests in cooperation with the data structures discussed in § 5.1.

Read operation: When a read request to page X is received, RFTL first looks up the LPA in the cached address mapping table (1). If it is a cache miss, it searches the corresponding translation page in the GMD (2) to locate the mapping entry for X in the translation page. During this process, the RFTL also places the mapping entry in the LRU cache for the address mapping (1). If it is a cache hit when accessing the cached address mapping table (1), the read operation will be issued directly. After locating the PPA of page X for serving the read operation, the RFTL updates the read bitmap in RTT (5) and sets the corresponding bit to 1 to indicate that the corresponding physical page has been read.

Write operation: When receiving a write request, RFTL performs the same address lookup procedure as for read in the cached address mapping table (1). If the mapping entry exists in the LRU cache, the data is written to a new free page, the address mapping entry is updated with the new PPA. Otherwise, a new mapping entry is created. The updated or newly created mapping entries are propagated to the translation pages and GMD (2) when they are evicted from the cached address mapping table (1).

To enable the reverse mapping from the physical page in SSDs to logical page in file systems for data recovery, RFTL stores the metadata information of a page in its out-of-band (OOB) metadata. The commodity SSDs typically reserve 16-64 bytes OOB metadata for each physical page. FlashGuard leverages this space to store the metadata information about a page as shown in Figure 4.

The OOB metadata includes (1) the LPA mapped to this physical page, (2) the previous PPA (P-PPA) mapped to the current LPA (it is used when a page is overwritten and it enables FlashGuard to identify all the old pages mapped to the same LPA), (3) the timestamp when the page is written, and (4) a **Retained Invalid Page (RIP)** bit to indicate whether this page is invalid and also potentially manipulated by encryption ransomware. The OOB metadata is hardware-isolated and its content is filled within flash controller, therefore, it is naturally protected against the adversary that has the administrator privilege. We will discuss how these metadata can be leveraged for data recovery in § 5.4.

[3]In FlashGuard, we use a bitmap carries 64 bits because each block contains 64 pages.

5.3 Garbage Collection in RFTL

Garbage collection is an essential component in SSDs to provide free blocks for future use by compacting the used flash blocks and also guarantee all the flash blocks age uniformly to extend SSD lifetime. It also plays a critical role in preserving the old copies (invalid pages) of the data manipulated by ransomware. When GC executes, it first selects the candidate blocks, move the valid pages in those blocks to new free blocks, and then erases these candidate blocks for future use.

Key idea: To make an SSD capable of holding data for recovery, we propose a new GC scheme in RFTL. In particular, RFTL examines whether an invalid page in a GC candidate block has been read. The GC will retain those pages. The invalid pages that have never been read will be discarded/erased. The intuition behind this is that ransomware needs to read data from an SSD before performing encryption, therefore the pages that have never been read could not be a piece of damaged data caused by ransomware.

We describe the new GC scheme in Algorithm 1 and discuss its procedure as follows.

GC procedure: When the number of free blocks in an SSD is below a threshold (10% - 40% of all the flash blocks in commodity SSDs), GC will be triggered to free space. The existing GC typically employs a greedy algorithm for selecting the GC candidate blocks. More specifically, it chooses the block with the least number of valid pages. This selection procedure can be quickly completed by looking up the BVT (3) that tracks the number of valid pages for each block.

Different from the current block selection scheme for GC candidate, RFTL takes those retained invalid pages (RIP has set to be *Reserved* in Algorithm 1) as valid pages. Therefore, the GC in RFTL selects the block with the least number of both valid pages and retained invalid pages. Such a GC scheme implies that a block with multiple invalid pages retained for recovery may delay its collection (see Figure 5), which could reduce the additional GC overhead caused by copying retained invalid pages to new free blocks.

Once a candidate flash block is selected, RFTL checks the PVT (4) and searches the valid pages in that block. Since lazy policies are usually adopted to update the PVT, the information in PVT might be outdated. To address this issue, RFTL double checks each valid page indicated by PVT by looking at its OOB metadata. It retrieves the LPA from the OOB metadata and looks up the corresponding PPA through the address mapping table (1). If the PPA retrieved is the same as the PPA of the page, RFTL deems it valid.

Given a candidate flash block, RFTL migrates its valid pages and retained invalid pages to new free blocks. For those valid pages, their corresponding mapping entries in (1) are updated and pointed to the new PPAs. The retained invalid pages will be kept in the

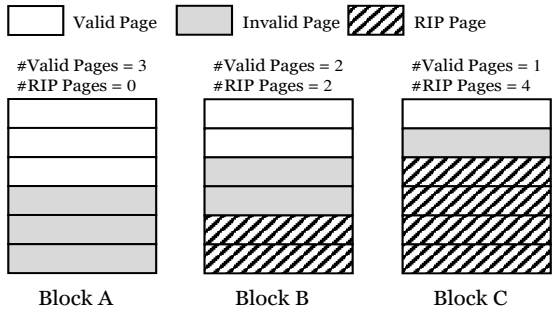


Figure 5: **An example of candidate block selection in state-of-the-art GC vs. RFTL's GC. Traditionally, block *C* is selected, as the number of the valid pages is the least. In RFTL, block *A* is selected, since RFTL counts the retained invalid pages (RIP) as valid pages.**

flash device for a certain time (a configurable threshold, 20 days in FlashGuard by default).

RFTL uses the timestamp stored in the page's OOB metadata to calculate how long this page has been retained. Once the interval between the timestamp in the OOB metadata and the current time is larger than the configured threshold, the page will be erased and reclaimed. Otherwise, both the page and its OOB metadata are copied to a free page, so that RFTL will keep retaining this invalid page in the SSD until it is expired when it is selected by GC next time (see line 13-21 in Algorithm 1).

For an invalid page X whose RIP bit is not set and has been read as indicated in RTT (5), it is treated as a page to be retained and will be copied to a free page Y. RFTL runs the GC procedure for this type of invalid pages as follows:

First, RFTL prepares the OOB metadata for the new page Y: the RIP bit is set to be *Reserved*, the timestamp is set to the current time (so that the content of this page will be conservatively retained for a certain period of time), the LPA and P-PPA are kept the same as in page X's OOB metadata. Second, RFTL copies the page X and its OOB metadata into the free page Y in a new free block. Third, page X's read bit in RTT is cleared and page Y's read bit in RTT is set to 1 (indicating the content of this page has been read). Finally, the page X is garbage collected (see line 23-26 in Algorithm 1). After this procedure, RFTL moves the retained invalid page to a new location and keeps holding it in the flash device.

For an invalid page which has never been read, RFTL will discard and garbage collect it (see line 28 in Algorithm 1), which is handled in the same way as in traditional SSDs.

Impact on SSD performance: The GC scheme in RFTL keeps the basic and essential procedures in the state-of-the-art FTLs, including candidate block selection and valid page movement. In our design, the overhead is introduced by copying retained invalid pages. The RFTL takes retained invalid pages as valid pages and the GC on the blocks carrying these pages will be delayed (see Figure 5). Meantime, RFTL also needs to ensure all the blocks age at the same rate (i.e., wear leveling) to extend the lifetime of the SSD. The blocks that have retained invalid pages, would still be selected as candidate blocks for GC, thus additional overhead would be introduced.

Algorithm 1 Garbage Collection in RFTL

Input: $ReserveTime$ = the time threshold for retaining invalid pages
$Reserved$ = the bit flag indicating a page is invalid but retained

```
1:  Select the candidate block for GC                                   ▷
    the candidate block has the least number of valid pages and
    retained invalid pages.
2:  Check PVT to find valid pages in candidate block
3:  for each valid page do
4:      Check page's OOB metadata
5:      Verify page's validity
6:      if page is valid then
7:          Copy page to a new free page
8:          Update address mapping entry
9:  for each invalid page do
10:     Check read tracker table (RTT)
11:     if page has been read then
12:         Check page's RIP bit
13:         if RIP == Reserved then
14:             page_timestamp← timestamp in page's OOB metadata
15:             if current_time - page_timestamp < ReserveTime then
16:                 Clear this page's read bit in the bitmap of RTT
17:                 Copy page and its OOB metadata to a new free page
18:                 Set the new page's read bit in RTT to 1 (Read)
19:             else
20:                 Discard and reclaim this page
21:                 Clear this page's read bit in the bitmaps of RTT
22:         else
23:             Set metadata (timestamp←current_time, RIP←Reserved)
24:             Clear this page's read bit in the bitmaps of RTT
25:             Copy page and its OOB metadata to a new free page
26:             Set the new page's read bit in RTT to 1 (Read)
27:     else
28:         Discard and reclaim this page
```

However, these blocks will not be frequently garbage collected because of the throttling and swapping mechanisms in the existing GC design: cold data (i.e., not frequently accessed data) is migrated to old blocks (i.e., blocks that experience more wear). The blocks which have many retained invalid pages will be accessed less frequently, and the chance that they will be collected shortly is small. In addition, if all the pages in a GC candidate block are invalid and will be retained, RFTL does not garbage collect them.

Impact on SSD lifetime: The SSD lifetime is determined by the wear-leveling and write traffic to the device. The GC in existing FTLs uses a greedy policy for candidate block selection, which always selects the block having the least number of valid pages. Such a GC policy provides maximal GC efficiency (i.e., the least page migration), and the throttling and swapping mechanisms are used to balance the wear between blocks. RFTL employs these techniques. Moreover, recent research [14, 50] on SSDs discloses that a relaxed wear-leveling can provide guaranteed SSD lifetime. Experiments with a variety of real-world workloads demonstrate that RFTL has minimal impact on SSD lifetime in § 6.

5.4 Data Recovery

To restore the invalid pages retained in an SSD when victims are aware of the ransomware infection, users can remove the SSD

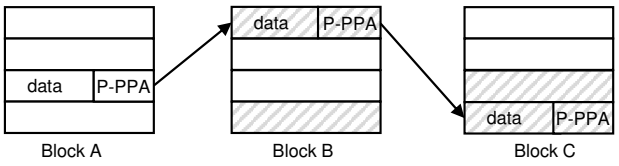


Figure 6: **FlashGuard can restore all the overwritten pages by travelling back to their previous versions with the previous physical page address stored in each page's OOB metadata.**

device and plug it into another clean and isolated computer for data recovery in case ransomware would attack the data recovery procedure. FlashGuard first checks the RTT (5) to locate all the pages that have been read recently. These pages are the candidate pages that may contain the user's stale data. As the RTT is cached in firmware RAM, this checking procedure is fast. To read the retained invalid page, FlashGuard checks the RIP bit in OOB metadata of each candidate page. If the RIP bit flag is set, the page is read from flash. Otherwise, RFTL will check the address mapping table (1) to figure out whether this page is valid or not. If it is invalid, the page is read from flash as well, since it is possible that this page is also a victim page.

We leverage the internal parallelism in an SSD to accelerate the procedure of reading the reatined invalid pages. Specifically, FlashGuard can simultaneously read the pages from multiple chips of an SSD to the host machine, therefore the recovery procedure will not take too much time (see the evaluation in § 6).

Once these retained invalid pages are read from a flash drive, the LPAs, P-PPAs, and timestamps stored in these pages' OOB metadata will be used to reconstruct the user files. FlashGuard can use the previous physical page address (P-PPA) stored in each page's OOB metadata to reverse an invalid page to its previous versions as shown in Figure 6. In order to maintain data locality for performance reasons, modern file systems usually manage the logical address space in a contiguous manner, and also flash controllers buffer storage operations to exploit temporal and spatial locality [22]. With these insights, the recovery tool in FlashGuard sorts the retained invalid pages with their LBAs and timestamps to reconstruct the original file. As a page could have been overwritten several times by either ransomware or trusted users, the recovery tool can reverse it to any older version and allow users to verify the content.

Since FlashGuard retains all the versions of the invalid pages in flash device, many other existing data recovery tools can also be leveraged to reconstruct user files (if there is no information available for data locality). For example, some recovery tools can read the first few bytes in each page to figure out the file type (e.g., `.ppt` or `.doc` file), and then use the defined layout for the file type to recover the data [7].

5.5 Metadata Recovery

As all the data structures (see Figure 3) are cached in firmware RAM, the cached data could be lost if a power failure happens. FlashGuard maintains their durability by leveraging the metadata recovery and check-pointing techniques that have been adopted in the state-of-the-art FTLs [8, 16]. RFTL identifies the recent written flash block by checking its OOB metadata (which includes timestamp as shown in Figure 4) and use the metadata information to recover the cached entries such as the address mapping table (1). For the data structure RTT (5) that tracks the recent reads, RFTL recovers it to the latest checkpointed states. For the blocks that have been written after the checkpoint, RFTL identifies their older versions (with P-PPA in OOB metadata) and conservatively marks them as 'read' in RTT.

An alternative solution is to use a battery or large capacitor to preserve the cached entries and persist them before power turns off, which simplifies the metadata recovery procedure significantly. We wish to take this solution as the future work.

5.6 FlashGuard Implementation

We implement FlashGuard on a programmable SSD with a state-of-the-art page-level FTL. The size of the SSD is 1 TB. Each block in the SSD has 64 pages and each page is 4 KB with 16 bytes of OOB metdata. The programmable SSD provides basic I/O control commands to issue read, write and erase operations against the physical flash device. The RFTL for FlashGuard is implemented based on the page-level FTL. FlashGuard is implemented with 5,718 lines of `C` code on top of the flash device. The SSD is over-provisioned with 15% of its full capacity by default, and the garbage collection is running in background. As we develop FlashGuard as a firmware solution, once the firmware is flushed into the device controller, commodity SSDs no longer allow firmware modifications. This characteristic ensures the integrity of FlashGuard.

We also implement a recovery tool that can read all the retained invalid pages from flash device and organize them in the manner as discussed in 5.4. The recovered data will be written back to SSD after having verified by users.

6 EVALUATION

Our evaluation demonstrates the efficiency of FlashGuard in two major aspects. First, we measure the effectiveness of deploying FlashGuard against encryption ransomware. We verify that FlashGuard can efficiently recover data locked by various types of encryption ransomware. Second, we evaluate the extra cost that would be introduced by FlashGuard. Specifically, we show that: (1) FlashGuard introduces negligible overhead to the storage operations from a variety of popular application workloads; (2) FlashGuard has minimal impact on the SSD lifetime.

6.1 Experimental Setup

To evaluate the capability of FlashGuard to recover data encrypted by ransomware, we use the 1,477 ransomware samples from 13 families as shown in Table 1. These samples are executed with the same experimental setup as described in § 2.1. Once a ransom screen appears, we start to run the recovery tool to recover encrypted data.

To evaluate the impact of FlashGuard on storage performance and SSD lifetime, we reply five sets of I/O traces collected from a variety of real-world applications (see Table 3): (1) the storage traces collected from enterprise servers running different applications (e.g., media server, research project management systems, and print server) in Microsoft Research at Cambridge for six days [28]; (2) the storage traces collected from machines running in a department at

Table 3: **A variety of real-world application workloads used for evaluating FlashGuard. R: *Read*, W: *Write*.**

	Workload	Description	IO Pattern
FIU IO Trace	online-course	course management system of a department using Moodle	R:22.3%, W:77.7%
	webmail	web interface to the mail server	R:18.0%, W:82.0%
	home	research group activities: developing, testing, experiments, etc.	R:0.9%, W:99.1%
	mailserver	department mail server traces	R:8.6%, W:91.4%
	web-research	research projects management using Apache web server	R:0.001%, W:99.999%
	web-users	web server hosting faculty, staff and graduate student web sites	R:10.0%, W:90.0%
Microsoft Servers	hm	hardware monitoring	R:35.5%, W:64.5%
	mds	media server	R:11.9%, W:88.1%
	prn	print server	R:10.8%, W:89.2%
	prxy	firewall/web proxy	R:3.1%, W:96.9%
	rsrch	research projects	R:9.3%, W:90.7%
	src	source control	R:56.4%, W:43.6%
	stg	web staging	R:15.2%, W:84.8%
	ts	terminal server	R:17.6%, W:82.4%
	usr	user home directories	R:40.4%, W:59.6%
	wdev	test web server	R:20.1%, W:79.9%
	web	web/SQL server	R:29.9%, W:70.1%
Misc	postmark	mail servers	R:83.2%, W:16.8%
	IOZone	filesystem benchmark	R:0.0%, W:100.0%
	TPC-C	online transaction processing	R:75.1%, W:24.9%
	TPC-E	OLTP of a brokerage firm	R:91.8%, W:8.2%

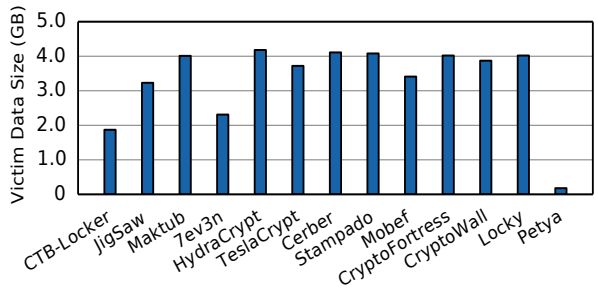


Figure 7: **The total size of the data encrypted by each ransomware familiy.**

FIU for twenty days [49]; (3) the database workload traces of running TPC-C benchmark and TPC-E benchmark for eight days [48]; (4) the storage traces of running IOZone benchmark [18] for ten days; (5) the storage traces of running the Postmark benchmark [29] for ten days. For each experiment, we first run 50 million mixed read and write operations to warm up the system and then replay each trace to collect the performance results.

6.2 Efficiency on Data Recovery

FlashGuard performs the procedure of data recovery following the approaches discussed in § 5.4. Once the recovery procedure is finished, we manually verify the pages that have been read from flash device. All the original versions of the encrypted data can be found in the flash pages recovered by FlashGuard. Figure 7 displays the average size of the data recovered from infection by different families, which ranges from 0.2 GB to 4.1 GB.

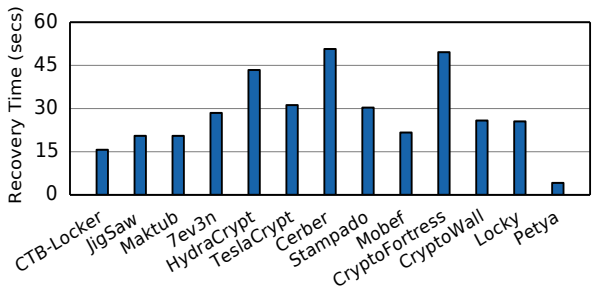


Figure 8: **The time of restoring the data that have been encrypted by ransomware.**

The execution time of restoring the encrypted data ranges from 4.2 seconds to 49.6 seconds as shown in Figure 8. FlashGuard leverages the internal parallelism in flash device to access the retained invalid pages in parallels. It is noted that the recovery time is not proportional to the victim data size, as the retained invalid pages are not evenly distributed across the parallel elements (i.e., chip-level packages) in flash device. However, the current recovery approach used in FlashGuard is much faster than the naive approach that scans the whole flash device (which takes 707.7 seconds).

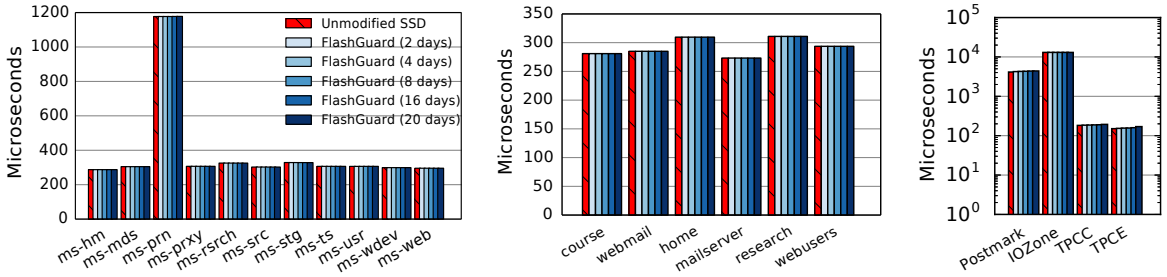
As most of the ransomware samples do not read and overwrite user data many times, it takes little time for FlashGuard to reconstruct the original files. Although encryption ransomware would attack user data with the knowledge of SSD properties, such as keeping reading and overwriting user data to an SSD (more cases will be discussed in § 7), FlashGuard can still restore the encrypted data since it retains all their older versions.

6.3 Impact on Storage Performance

To understand the impact of FlashGuard on storage performance, we begin with the default over-provisioning (15% of the SSD's full capacity), and run the acknowledged storage traces collected from real-world applications (see Table 3). We assume all the writes are encrypted, which means all the invalid pages that have been read will be retained in SSD. The time of holding these invalid pages ranges from 2 days to 20 days, the storage latency and throughput are reported in Figure 9 and Figure 10.

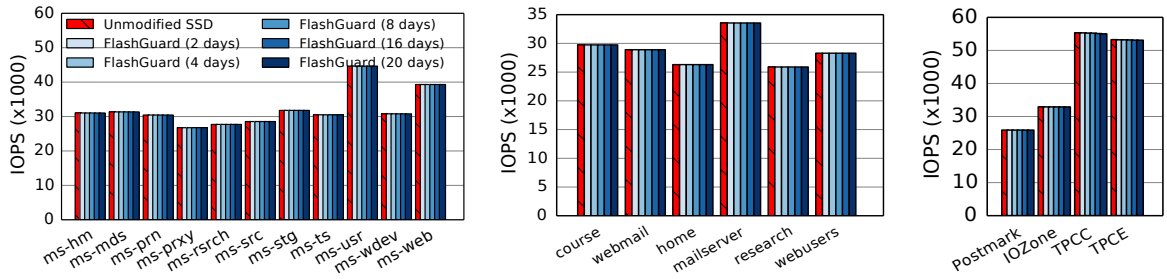
For most of the workloads, the average latency of running them on FlashGuard is almost the same as that of running them on the unmodified SSD as shown in Figure 9. For I/O-intensive workloads including Postmark, TPCC and TPCE, FlashGuard increases the average latency by up to 6.1%. As the time of holding retained invalid pages is increased, the average latency is slightly increased. In terms of I/O throughput, FlashGuard has trivial impact as shown in Figure 10. For I/O-intensive workloads, the average throughput drops by up to 0.6%. FlashGuard does not introduce much performance overhead for several reasons:

First, according to our statistical study on a variety of real-world storage traces collected over six to twenty days (see Figure 11), only a small portion (4.1% on average) of the storage operations have the similar I/O patterns (i.e., read-overwrite operations) as encryption ransomware. Therefore, FlashGuard retains only a small amount of invalid pages for regular applications. Second, the RFTL in FlashGuard delays the GC execution on the flash blocks with



(a) Server Storage in Enterprise (b) Server Storage in University (c) Misc I/O Workloads

Figure 9: **The average latency of running real-world workloads with FlashGuard vs. Unmodified SSD. The time of holding retained invalid pages in FlashGuard ranges from 2 days to 20 days. FlashGuard's average latency is almost the same as that of the unmodified SSD for a variety of workloads.**



(a) Server Storage in Enterprise (b) Server Storage in University (c) Misc I/O Workloads

Figure 10: **The average throughput of running real-world workloads with FlashGuard vs. Unmodified SSD. FlashGuard has negligible impact on the I/O throughput for most of these workloads.**

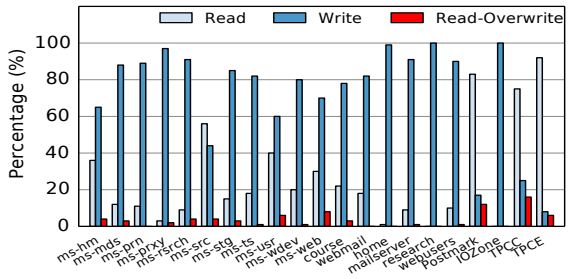


Figure 11: **The analytics on the I/O patterns of the real-world application workloads.**

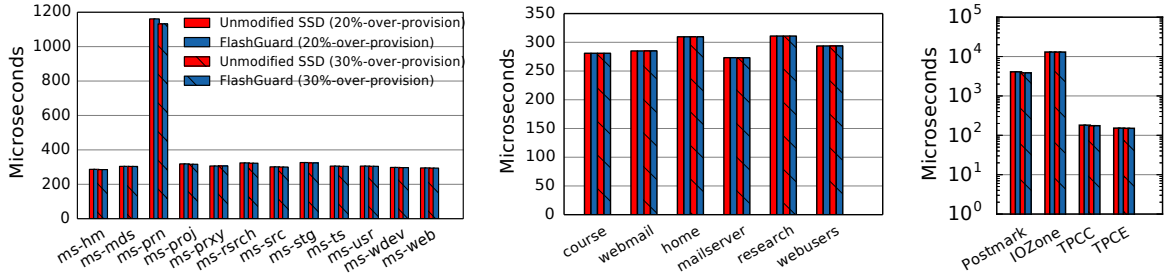
retained invalid pages by counting them as valid pages, which reduces the chances of moving retained invalid pages. Third, the GC is executed in background, which allows FTLs schedule GC during the idle time of flash controller, further reducing the performance interference caused by GC. Finally, the existing I/O schedulers and FTLs provide decent GC efficiency (i.e., the valid page movements during GC procedure) for many workloads. When all the pages on a flash block are invalid, the flash block will be erased without incurring any page movement. In FlashGuard, no additional page movement is required for a flash block whose pages are all retained invalid pages.

To further understand the performance overhead of FlashGuard, we profile the GC events and collect statistics on the number of additional page movements. As shown in Table 4, all the FIU workloads incur no additional page movements, although the time of holding the retained invalid pages is set to be 20 days. For the workloads running in enterprise servers, up to 0.8% of the page movements are contributed by retaining invalid pages. For those I/O intensive workloads such as Postmark, TPCC and TPCE, more page movements are introduced. Since the IOZone traces are write-only, no pages are required to be retained in FlashGuard.

We also investigate how the over-provisioning (i.e., reserve more free blocks in SSD) affects FlashGuard's performance. We increase the over-provisioning ratio from 15% (default setting) to 20% and 30% respectively, and do the performance comparison with the unmodified SSD. As demonstrated in Figure 12, the average I/O latency of running a variety of real-world workloads on FlashGuard is almost the same as that of running these workloads on unmodified SSD, indicating that FlashGuard has negligible negative impact on regular storage operations. As we increase the ratio of over-provisioning, the average latency is slightly decreased for both unmodified SSD and FlashGuard because the storage capacity is traded for performance. In terms of the storage throughput with different over-provisioning ratio (not shown in the paper), we reach the similar conclusion that FlashGuard introduces trivial overhead.

Table 4: **The additional page movements (%) for retaining invalid pages in FlashGuard over the time period from 2 to 10 days.**

	Server Storage in Enterprise											Server Storage in University						Misc I/O Workloads			
days	hm	mds	prn	prxy	rsrch	src	stg	ts	usr	wdev	web	course	webmail	home	mailserver	research	webusers	Postmark	IOZone	TPCC	TPCE
2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.1	0.0	0.0	0.0	0.0	0.0	0.0	8.5	0.0	8.1	5.3
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.1	0.0	0.1	0.0	0.0	0.0	0.0	0.0	0.0	9.1	0.0	8.3	5.5
8	0.5	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.1	0.0	0.4	0.0	0.0	0.0	0.0	0.0	0.0	9.7	0.0	8.8	5.7
16	0.5	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.3	0.0	0.6	0.0	0.0	0.0	0.0	0.0	0.0	9.7	0.0	8.8	5.7
20	0.5	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.3	0.0	0.8	0.0	0.0	0.0	0.0	0.0	0.0	9.7	0.0	8.8	5.7



(a) Server Storage in Enterprise (b) Server Storage in University (c) Misc I/O Workloads

Figure 12: **The average latency of running real-world workloads when the over-provisioning is changed from 20% to 30%. The time of holding retained invalid pages is set to be 20 days.**

6.4 Impact on SSD Lifetime

As each flash block has limited endurance, it is necessary to ensure FlashGuard can offer acceptable SSD lifetime. Two metrics are used to evaluate FlashGuard's impact on SSD lifetime. (1) The wear balance measures whether the flash blocks age evenly or not. We use the standard deviation of the remaining lifetime of all the flash blocks to evaluate SSD's wear balance. (2) The write amplification factor (WAF) [57] evaluates the actual amount of physical write traffic to the logical amount of write traffic. Larger WAF means that SSD suffers from more write traffic, indicating that the SSD would last for a shorter time.

FlashGuard aims to achieve the same wear balance as the unmodified SSD since their basic strategies for GC and block allocation are the same. During the GC procedure, the hot block (which is erased frequently) will be swapped with cold block to make sure the wear balance is well maintained.

The wear balance of FlashGuard is even better than unmodified SSD for some workloads such as those in enterprise servers (see Figure 13). This could be because FlashGuard delays GC execution on some flash blocks, which affects the wear distribution among the flash blocks. Overall, the experiments with a set of real-world workload traces demonstrate that FlashGuard could maintain the wear balance across all the flash blocks as well as that of the state-of-the-art SSD.

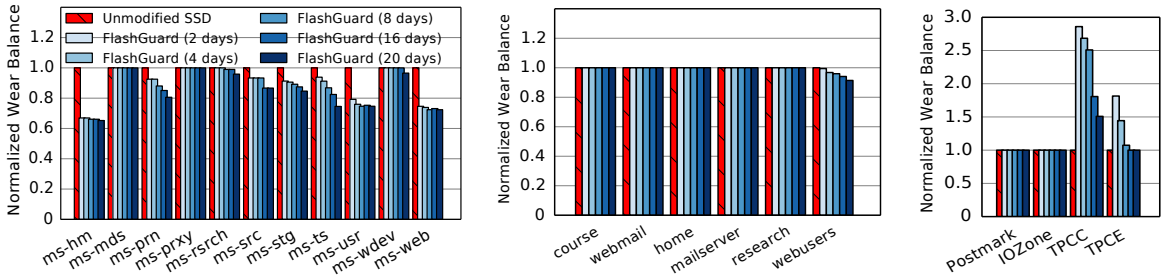
We use another metric WAF to evaluate how FlashGuard can affect SSD lifetime. As shown in Figure 14, for the storage workloads running in enterprise and university, the WAF of FlashGuard is the same as that of unmodified SSD. For IO-intensive workloads, the WAF is increased by up to 4%. This is because FlashGuard incurs additional page movements for retaining invalid pages. As the time of holding the retaining invalid pages in flash device is increased, the WAF is slightly increased. However this is less of a concern. For an SSD that usually has a lifetime of 160 - 250 weeks, the slightly increased WAF reduces its lifetime by only one or two weeks, which is acceptable in practice.

7 DISCUSSION AND FUTURE WORK

According to our study in § 2, few encryption ransomware was developed considering the SSD characteristics. In this section, we discuss the possible ransomware attacks against FlashGuard and potential research directions in the future.

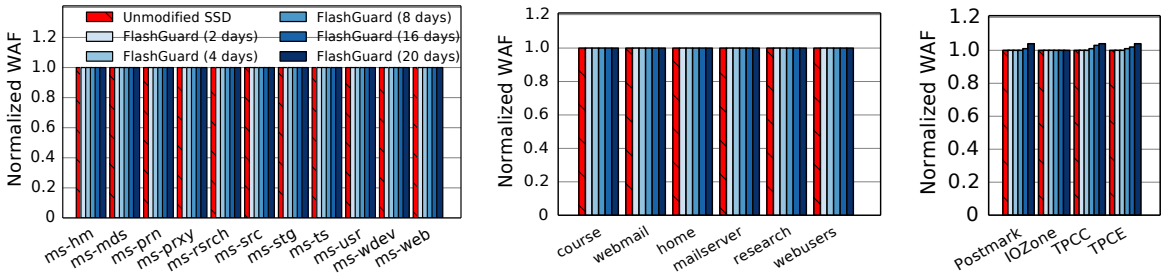
Exploiting storage capacity. To support data recovery, FlashGuard holds the data potentially encrypted by ransomware and prevents them from being discarded by garbage collection. Intuition suggests an attacker can exploit storage capacity and keep writing to occupy the available space in SSD, forcing FlashGuard to release its hold. Another potential attack is that a ransomware keeps reading and overwriting data to the SSD in order to cause FlashGuard to retain a large amount of garbage data. In practice, such attacks are in vain. FlashGuard refuses to release data hold if the lifespan of the holding data has not yet expired, even though the SSD is fully occupied. When such an incident happens, FlashGuard will stop issuing IO requests when the SSD is full, resulting in the failure of filesystem operations in OS. Therefore, although ransomware has the kernel privilege, it cannot prevent a user from noticing the abnormal events.

Timing attacks. Time is critical for both security and performance of FlashGuard. The longer FlashGuard holds stale data, the more



(a) Server Storage in Enterprise (b) Server Storage in University (c) Misc I/O Workloads

Figure 13: **The normalized wear balance (lower is better) across all the flash blocks after running real-world workloads.**



(a) Server Storage in Enterprise (b) Server Storage in University (c) Misc I/O Workloads

Figure 14: **The normalized write amplification factor (WAF) of FlashGuard compared to Unmodified SSD (lower is better).**

overhead it might impose to I/O operations. To obtain high storage performance, a user might set the lifespan of holding data relatively short. In this way, the user is exposed to the threat of ransomware attacks in that ransomware could slow down the pace of encrypting data and notifying victims.

As discussed in § 2, ransomware variants have been evolving to lock up user data and collect ransom rapidly to prevent from being caught. In § 6, we have already demonstrated that FlashGuard typically incurs only negligible overhead to regular I/O operations, even though we set the lifespan of holding data for 20 days. This implies FlashGuard is effective in defending against the aforementioned ransomware attacks. This is because it not only significantly increases the risk of ransomware of being caught but also thwarts ransomware authors from gaining rewards rapidly. We wish to explore new detection and defense mechanisms against the timing attacks in the future.

Secure deletion. FlashGuard retains overwritten contents for the sake of recovery. Intuitively, this design contradicts to the objective of secure deletion [24, 37–39, 54], which requires irrecoverable data deletion from a physical medium. However, we believe FlashGuard is compatible with secure deletion. In particular, FlashGuard can use a user-specified encryption key to encrypt the stale data potentially overwritten by ransomware. In this way, a user can still perform data recovery but not worrying about data leakage because adversaries cannot restore "securely deleted data" without the encryption key. As future work, we will develop this solution, making FlashGuard compatible with secure deletion.

Protecting against encryption ransomware on various platforms. FlashGuard leverages the intrinsic properties of Flash to protect against encryption ransomware, which means its approach can be applied to any kind of flash-based storage devices to protect different computing platforms against encryption ransomware. A typical example is the mobile device which has used Flash to store personal user data for decades. As the flash devices used on mobiles (e.g., eMMC) share the same intrinsic properties as that on personal computers and enterprise servers (e.g., SSDs) [15, 21, 25], our approach can be deployed on the mobile platform to enhance its storage system and protect users against the ever-increasing threat of mobile ransomware such as `Simplocker` [2, 26, 35, 43].

8 RELATED WORK

The lines of work most closely related to FlashGuard are research on ransomware detection and data recovery.

Ransomware detection. The prior research mainly focuses on demystifying ransomware attacks [20] and detecting their footprints [19, 44, 45]. Several defense mechanisms have been proposed and developed. Kharraz et al. proposed `UNVEIL` [19], a dynamic analysis system that characterizes encryption ransomware behavior and detects ransomware footprints by tracking how ransomware interacts with user data. `CryptoDrop` [44] is another ransomware detection system that alerts users when it observes a process that appears to tamper with a large amount of user data. In addition, recent research leverages machine learning techniques to perform ransomware analysis and classification. `EldeRan` [45] is such an example which models each program as a set of features and do the classification to determine if the program is a piece of ransomware.

The mechanisms discussed above offer effective detection of encryption ransomware. However, they do not provide sufficient,

proper cure for the damage that has been caused. As such, ransomware still locks up a few files. Given that encrypted files might be vitally important for business operations, the victims may still have to pay a hefty ransom request in order to minimize the damage. In this paper, we look beyond ransomware detection and concentrate on a solution to offset the damage to user files. To the best of our knowledge, FlashGuard is the first system designed to reinstate damage caused by encryption ransomware.

Data recovery. Data recovery techniques might allow users to restore their data to the copies prior to the encryption. A large number of backup systems have been proposed [5, 12, 33, 41]. The ones that have been commonly adopted on Unix systems are `dump` and `tar` utilities. They both support full and incremental backup strategies [33]. On Microsoft Windows system, the most popular backup system is Volume Shadow Copy Service that archives user data on local and external volumes in an incremental manner [41]. Another line of work capable of achieving data recovery are log-structured file systems [40] and journaling file systems [32]. They both maintain data updates in persistent logs. Once data loss or inconsistency occurs, they can recover the data back to previous states by rolling back the logs.

Apart from the backup systems integrated into modern OSes, other well-developed backup systems include the IBM Tivoli Storage Manager [5] that performs selective, incremental backup in conjunction with deduplication, and those cloud based storage systems [12] that synchronize file updates and creation with the backup storage running on the cloud.

As a defense mechanism, however none of them is sufficient and proper. To avoid loss of files newly updated or created, they have to perform backup frequently. From the perspective of efficiency, this is particularly time consuming. Since ransomware has already run with the kernel privilege, the backup systems proposed can be easily disabled or circumvented. For example, a backup process that synchronizes user files with a cloud storage can be terminated by ransomware. In this work, we design and develop firmware-level data recovery mechanism, making it naturally resistant to ransomware attacks launched at both user and kernel levels.

Looking beyond file backups, researchers proposed to integrate proactive defense mechanisms into the existing software systems recently. `ShieldFS` [6] monitors the low-level file access activities to detect ransomware and implements a protection layer with the copy-on-write mechanism to recover data. `PayBreak` [23] hooks crypto functions in the standard libraries to identify the invocations from ransomware and logs the encryption key for future data decryption. Similar to the attacks against file backups, ransomware can easily undermine these mechanisms by disabling them with kernel privilege or obfuscating the execution of its critical functions.

9 CONCLUSION

In this paper, we develop FlashGuard, a ransomware-tolerant SSD that retains the data potentially encrypted by ransomware in SSDs. With FlashGuard, we demonstrate that victims can efficiently reinstate the damage to their files caused by encryption ransomware. The design of FlashGuard takes advantage of the intrinsic flash properties. We show FlashGuard only introduces negligible overhead to regular storage operations and has trivial impact on SSD lifetime. In comparison with existing detection mechanisms against ransomware, FlashGuard is the first firmware-level defense system, it is naturally resistant to the ransomware that exploits kernel vulnerabilities or runs with the kernel privilege.

ACKNOWLEDGMENTS

We would like to thank the anonymous reviewers for their helpful comments and feedback. This research was supported in part by NSF grant 1526798, ARO W911NF-13-1-0421 (MURI), NSF CNS-1422594, and NSF CNS-1505664.

REFERENCES

[1] Nitin Agrawal, Vijayan Prabhakaran, Ted Wobber, John D Davis, Mark S Manasse, and Rina Panigrahy. 2008. Design Tradeoffs for SSD Performance.. In *USENIX Annual Technical Conference*. 57–70.
[2] Nicolo Andronio, Stefano Zanero, and Federico Maggi. 2015. HelDroid: Dissecting and Detecting Mobile Ransomware. In *Proc. International Symposium on Research in Attacks, Intrusion and Detection (RAID'15)*. Kyoto, Japan.
[3] Werner Bux and Ilias Iliadis. 2010. Performance of greedy garbage collection in flash-based solid-state drives. *Performance Evaluation* 67, 11 (2010), 1172–1186.
[4] Cerber Ransomware - New, But Mature. 2016. https://blog.malwarebytes.com/threat-analysis/2016/03/cerber-ransomware-new-but-mature/. (2016).
[5] IBM Comparing. 2002. Tivoli Storage Manager and VERITAS NetBackup in Real-World Environments. *A summary by IBM of the whitepaper and benchmark written by Progressive Strategies* (2002).
[6] Andrea Continella, Alessandro Guagneli, Giovanni Zingaro, Giulio De Pasquale, Alessandro Barenghi, Stefano Zanero, and Federico Maggi. 2016. ShieldFS: A Self-healing, Ransomware-aware Filesystem. In *Proc. the 32nd Annual Conference on Computer Security Applications (ACSAC'16)*. Los Angeles, CA.
[7] Data Recovery. 2014. https://support.microsoft.com/en-us/help/835840/data-recovery. (2014).
[8] Niv Dayan, Philippe Bonnet, and Stratos Idreos. 2016. GeckoFTL: Scalable Flash Translation Techniques For Very Large Flash Devices. In *Proc. SIGMOD'16*. San Francisco, CA.
[9] Jiang Ming Dongpeng Xu and Dinghao Wu. 2017. Cryptographic Function Detection in Obfuscated Binaries via Bit-precise Symbolic Loop Mapping. In *Proc. 38th IEEE Symposium on Security and Privacy (Oakland'17)*. San Jose, CA.
[10] Manuel Egele, Theodoor Scholte, Engin Kirda, and Christopher Kruegel. 2012. A survey on automated dynamic malware-analysis techniques and tools. *ACM Computing Surveys (CSUR)* 44, 2 (2012), 6.
[11] Simson Garfinkel, Paul Farrell, Vassil Roussev, and George Dinolt. 2009. Bringing science to digital forensics with standardized forensic corpora. *digital investigation* 6 (2009), S2–S11.
[12] James Gross. 2013. Cloud based storage: A brief look at dropbox. *Chronicles* 30, 4 (2013).
[13] How Ransomware Became a Billion-Dollar Nightmare for Business. 2016. https://www.theatlantic.com/business/archive/2016/09/ransomware-us/498602/. (2016).
[14] Jian Huang, Anirudh Badam, Laura Caulfield, Suman Nath, Sudipta Sengupta, Bikash Sharma, and Moinuddin K. Qureshi. 2017. FlashBlox: Achieving Both Performance Isolation and Uniform Lifetime for Virtualized SSDs. In *Proc. FAST'17*. Santa Clara, CA.
[15] Jian Huang, Anirudh Badam, Ranveer Chandra, and Edmund B. Nightingale. 2015. WearDrive: Fast and Energy-Efficient Storage for Wearables. In *Proc. 2015 USENIX Annual Technical Conference (USENIX ATC'15)*. Santa Clara, CA.
[16] Jian Huang, Anirudh Badam, Moinuddin K. Qureshi, and Karsten Schwan. 2015. Unified Address Translation for Memory-Mapped SSD with FlashMap. In *Proc. ISCA'15*. Portland, OR.
[17] IBM X-Force Research. 2016. Ransomware: How consumers and businesses value their data. *Technical Report* (2016).
[18] IOzone Lab. 2016. http://www.iozone.org/. (2016).
[19] Amin Kharaz, Sajjad Arshad, Collin Mulliner, William Robertson, and Engin Kirda. 2016. UNVEIL: A Large-Scale, Automated Approach to Detecting Ransomware. In *25th USENIX Security Symposium (USENIX Security 16)*. USENIX Association, Austin, TX, 757–772.
[20] Amin Kharraz, William Robertson, Davide Balzarotti, Leyla Bilge, and Engin Kirda. 2015. Cutting the Gordian Knot: A Look Under the Hood of Ransomware Attacks. In *Conference on Detection of Intrusions and Malware & Vulnerability Assessment (DIMVA)*. Milan, IT.
[21] Hyojun Kim, Nitin Agrawal, and Cristian Ungureanu. 2012. Revisiting Storage for Smartphones. In *Proc. 10th USENIX Conference on File and Storage Technologies*

(FAST'12). San Jose, CA.
[22] Hyojun Kim and Seongjun Ahn. 2008. BPLRU: A Buffer Management Scheme for Improving Random Writes in Flash Storage. In *FAST'08*. San Jose, CA.
[23] Eugene Kolodenker, William Koch, Gianluca Stringhini, and Manuel Egele. 2017. PayBreak: Defense Against Cryptographic Ransomware. In *Proc. the 2017 ACM on Asia Conference on Computer and Communications Security)*. Abu Dhabi, United Arab Emirates.
[24] Jaeheung Lee, Sangho Yi, Junyoung Heo, Hyungbae Park, Sung Y Shin, and Yookun Cho. 2010. An Efficient Secure Deletion Scheme for Flash File Systems. *J. Inf. Sci. Eng.* 26, 1 (2010), 27–38.
[25] Jing Li, Anirudh Badam, Ranveer Chandra, Steven Swanson, Bruce Worthington, and Qi Zhang. 2014. On the Energy Overhead of Mobile Storage Systems. In *Proc. 12th USENIX Conference on File and Storage Technologies (FAST'14)*. Santa Clara, CA.
[26] Robert Lipovsky, Lukas Stefanko, and Gabriel Branisa. 2016. The Rise of Android Ransomware. *White Paper* (2016).
[27] Master File Table (Windows). 2017. https://msdn.microsoft.com/en-us/library/windows/desktop/aa365230(v=vs.85).aspx. (2017).
[28] Dushyanth Narayanan, Austin Donnelly, and Antony Rowstron. 2008. Write Off-Loading: Practical Power Management for Enterprise Storage. In *Proc. 6th USENIX on File and Storage Technologies (FAST'08)*. San Jose, CA.
[29] NetApp. 1997. http://www.shub-internet.org/brad/FreeBSD/postmark.html. (1997).
[30] Gavin O'Gorman and Geoff McDonald. 2012. *Ransomware: a growing menace*. Symantec Corporation.
[31] John H Palevich and Martin Taillefer. 2008. Network file system. (Oct. 21 2008). US Patent 7,441,012.
[32] Vijayan Prabhakaran, Andrea C Arpaci-Dusseau, and Remzi H Arpaci-Dusseau. 2005. Analysis and Evolution of Journaling File Systems.. In *USENIX Annual Technical Conference, General Track*. 105–120.
[33] Curtis Preston. 2007. *Backup & recovery: inexpensive backup solutions for open systems.* " O'Reilly Media, Inc.".
[34] Ransomware Definition. 2016. http://www.trendmicro.com/vinfo/us/security/definition/ransomware. (2016).
[35] Ransomware on Mobile Devices. 2016. http://www.hipaajournal.com/ransomware-mobile-devices/. (2016).
[36] Ransomware: the Tool of Choice for Cyber Extortion. 2016. https://www.fireeye.com/current-threats/what-is-cyber-security/ransomware.html. (2016).
[37] Joel Reardon, David Basin, and Srdjan Capkun. 2013. Sok: Secure data deletion. In *Security and Privacy (SP), 2013 IEEE Symposium on*. IEEE, 301–315.
[38] Joel Reardon, Srdjan Capkun, and David Basin. 2012. Data Node Encrypted File System: Efficient Secure Deletion for Flash Memory. In *Proc. USENIX Security'12*. Bellevue, WA.
[39] Ronald L Rivest. 1997. All-or-nothing encryption and the package transform. In *International Workshop on Fast Software Encryption*. Springer, 210–218.
[40] Mendel Rosenblum and John K Ousterhout. 1992. The design and implementation of a log-structured file system. *ACM Transactions on Computer Systems (TOCS)* 10, 1 (1992), 26–52.
[41] Mark E Russinovich, David A Solomon, and Jim Allchin. 2005. *Microsoft Windows Internals: Microsoft Windows Server 2003, Windows XP, and Windows 2000*. Vol. 4. Microsoft Press Redmond.
[42] Ananda Sankaran, Kevin Guinn, and Dat Nguyen. 2004. Volume shadow copy service. *Power Solutions, March* (2004).
[43] Kevin Savage, Peter Coogan, and Hon Lau. 2015. The Evolution of Ransomware. *Symantec Technical Report* (2015).
[44] Nolen Scaife, Henry Carter, Patrick Traynor, and Kevin RB Butler. 2016. Cryptolock (and drop it): stopping ransomware attacks on user data. In *Distributed Computing Systems (ICDCS), 2016 IEEE 36th International Conference on*. IEEE, 303–312.
[45] D. Sgandurra, L. Muñoz-González, R. Mohsen, and E. C. Lupu. 2016. Automated Dynamic Analysis of Ransomware: Benefits, Limitations and use for Detection. *ArXiv e-prints* (Sept. 2016). arXiv:cs.CR/1609.03020
[46] Special Report: Ransomware and Businesses. 2016. http://www.symantec.com/content/en/us/enterprise/media/security_response/whitepapers/ISTR2016_Ransomware_and_Businesses.pdf. (2016).
[47] SSD prices plummet again, Close in on HDDs. 2016. http://www.pcworld.com/article/3040591/storage/ssd-prices-plummet-again-close-in-on-hdds.html. (2016).
[48] TPC-C: An On-line Transaction Processing Benchmark. 2001. http://www.tpc.org/tpcc/. (2001).
[49] Akshat Verma, Ricardo Koller, Luis Useche, and Raju Rangaswami. 2010. SRCMap: Energy Proportional Storage Using Dynamic Consolidation. In *Proc. 6th USENIX on File and Storage Technologies (FAST'10)*. San Jose, CA.
[50] Robin Verschoren and Benny Van Houdt. 2016. On the Impact of Garbage Collection on Flash-Based SSD Endurance. In *Proc. 4th Workshop on Interactions of NVM/Flash with Operating System and Workloads (INFLOW'16)*. Savannah, GA.
[51] Michael Virable, Stefan Savage, and Geoffrey M. Voelker. 2012. BlueSky: A Cloud-Backed File System for the Enterprise. In *Proc. 10th USENIX conference on File and Storage Technologies (FAST'12)*. San Jose, CA.
[52] VirusTotal - Free Online Virus, Malware and URL Scanner. 2016. https://www.virustotal.com/. (2016).
[53] WannaCry Ransomware Attack. 2017. https://en.wikipedia.org/wiki/WannaCry_ransomware_attack. (2017).
[54] Michael Yung Chung Wei, Laura M Grupp, Frederick E Spada, and Steven Swanson. 2011. Reliably Erasing Data from Flash-Based Solid State Drives.. In *Proc. 9th USENIX Conference on File and Storage Technologies (FAST'11)*.
[55] Will you backups protect you against ransomware. 2016. http://www.csoonline.com/article/3075385/backup-recovery/will-your-backups-protect-you-against-ransomware.html. (2016).
[56] WMIC - Take Command-line Control over WMI. 2002. (2002).
[57] Write Amplification Factor. 2017. https://en.wikipedia.org/wiki/Write_amplification. (2017).