Travelling the Hypervisor and SSD: A Tag-Based Approach Against Crypto Ransomware with Fine-Grained Data Recovery

Boyang Ma
State Key Lab of ISN
School of Cyber Engineering
Xidian University
Xi'an, China

Yilin Yang
State Key Lab of ISN
School of Cyber Engineering
Xidian University
Xi'an, China

Jinku Li[*]
State Key Lab of ISN
School of Cyber Engineering
Xidian University
Xi'an, China

Fengwei Zhang
Southern University of Science and Technology
Shenzhen, China

Wenbo Shen
Zhejiang University
Hangzhou, China

Yajin Zhou
Zhejiang University
Hangzhou, China

Jianfeng Ma
State Key Lab of ISN
School of Cyber Engineering
Xidian University
Xi'an, China

ABSTRACT

Ransomware has evolved from an economic nuisance to a national security threat nowadays, which poses a significant risk to users. To address this problem, we propose RansomTag, a tag-based approach against crypto ransomware with fine-grained data recovery. Compared to state-of-the-art SSD-based solutions, RansomTag makes progress in three aspects. *First*, it decouples the ransomware detection functionality from the firmware of the SSD and integrates it into a lightweight hypervisor of Type I. Thus, it can leverage the powerful computing capability of the host system and the rich context information, which is introspected from the operating system, to achieve accurate detection of ransomware attacks and defense against potential targeted attacks on SSD characteristics. Further, RansomTag is readily deployed onto desktop personal computers due to its parapass-through architecture. *Second*, RansomTag bridges the semantic gap between the hypervisor and the SSD through the tag-based approach proposed by us. *Third*, RansomTag is able to keep 100% of the user data overwritten or deleted by ransomware, and restore any single or multiple user files to any versions based on timestamps. To validate our approach, we implement a prototype of RansomTag and collect 3, 123 recent ransomware samples to evaluate it. The evaluation results show that our prototype effectively protects user data with minimal scale data backup and acceptable performance overhead. In addition, all the attacked files can be completely restored in fine-grained.

[*]Corresponding author.

CCS CONCEPTS

• **Security and privacy → Malware and its mitigation**.

KEYWORDS

Ransomware; Hypervisor; Fine-Grained; Data Recovery

ACM Reference Format:
Boyang Ma, Yilin Yang, Jinku Li, Fengwei Zhang, Wenbo Shen, Yajin Zhou, and Jianfeng Ma. 2023. Travelling the Hypervisor and SSD: A Tag-Based Approach Against Crypto Ransomware with Fine-Grained Data Recovery. In *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security (CCS '23), November 26–30, 2023, Copenhagen, Denmark.* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3576915.3616665

1 INTRODUCTION

Over the last few years, ransomware [15] attacks have been growing more widespread due to a high and increasing yield potential. Moreover, ransomware has evolved from an economic nuisance to a national security threat [56]. For example, in June 2022, Costa Rica's government declared a "national emergency" due to ransomware after it had been reeling from unprecedented attacks for two months, which sparks a new ransomware era [45]. It is no doubt that ransomware poses a significant risk to user security, and it is time to begin treating it as we would any other serious threat [9].

Different from other types of malware (e.g., viruses, worms, trojans, botnets, adware, spyware), the main purpose of ransomware is to lock the victim systems (a.k.a locker ransomware) or encrypt the user files (a.k.a, crypto ransomware), and then it demands a ransom for unlocking the system or releasing a decryption key to access the files. Further, it is hard to track the transactions as ransomware widely deploys anonymous payment mechanisms (e.g., Bitcoin) to accomplish the process [20]. Compared to locker ransomware, crypto ransomware is more difficult to tackle as it usually encrypts user files with strong cryptographic algorithms and enough-long cryptographic keys. That means the data can hardly be decrypted without the keys released from the attackers. Accordingly, we target crypto ransomware in this paper.

The ultimate goal of defending against crypto ransomware is to protect user data. To this end, researchers have proposed a number of solutions. Some of them identify attacks and protect user files in the host [10, 27–29, 31, 35, 52] or hypervisor layer [54]. However, such solutions suffer from data loss [27, 29, 52, 54], incomplete functionality [31, 35], or privilege escalation attacks [10, 27–29, 31, 35, 52]. To address these limitations, several SSD-based approaches against crypto ransomware have been designed in recent years [4, 5, 21, 43, 47, 50, 60]. These approaches leverage the intrinsic characteristic of SSD, i.e., out-of-place update [21], to provide hardware-level data protection without additional overhead.

However, there are three main limitations to the existing SSD-based approaches. *First*, it is hard to implement accurate and comprehensive detection algorithms into the SSD firmware due to insufficient context information and limited CPU power and memory. Thus, it causes the backup data to be excessively redundant [21, 47, 50] or incomplete [4, 5, 43, 60]. *Second*, they suffer from potential targeted attacks on SSD characteristics [4, 5, 21, 43, 47, 60], i.e., exhausting the available storage space (GC attack), hiding I/O patterns (timing attack), and forcing data erasing (TRIM attack), or introduce additional attack surface for network threats [50]. *Third*, they cannot recover a specific (attacked) file to a certain version wanted by the user, since there is no semantic information of "file" at the SSD level. In other words, they cannot provide fine-grained data recovery.

In this paper, we propose RansomTag, a tag-based approach for file activity-based detection outside of the operating system (OS) and fine-grained data recovery for SSDs, to address the above limitations. Similar to the existing SSD-based solutions, RansomTag leverages the intrinsic characteristic of SSD, i.e., out-of-place update, to back up user data without additional overhead. Nevertheless, compared to previous systems, RansomTag makes progress mainly in three aspects.

First, RansomTag decouples the ransomware detection functionality from the firmware of the SSD and integrates it into a hypervisor of Type I. Thus, we can leverage the powerful computing capability of the host system and the rich context information from the OS, e.g., process and file activities, to achieve accurate and comprehensive detection of ransomware attacks. We believe this is necessary and important as it can resist potential targeted attacks on SSD characteristics, i.e., GC, timing, and TRIM attacks [50] (see details in § 5.4.2). In addition, it will significantly reduce the backup storage overhead. Further, unlike traditional Type I hypervisors, the hypervisor adopted by us (called thin hypervisor) is based on a parapass-through architecture, which is designed to allow most of the I/O accesses from the OS to pass-through the hypervisor, while the hypervisor only intercepts the minimum file-related accesses necessary to implement the ransomware detection functionality. Such a thin hypervisor is readily deployed onto desktop personal computers.

Second, RansomTag bridges the semantic gap between the hypervisor and the SSD through the tag-based approach proposed by us, which is simple but effective with minor changes to standard SCSI protocol commands. Specifically, RansomTag precisely labels each I/O request with a tag in the hypervisor to inform the SSD firmware how to handle it. To achieve that, it leverages several bits of the reserved field in the storage command transformed from the I/O request to store various types of tags.

Third, RansomTag is able to provide complete protection and fine-grained recovery of user data. In other words, RansomTag will keep 100% of the user data overwritten or deleted by an identified crypto ransomware sample. Besides, RansomTag monitors file system status in the hypervisor to back up the changes of file metadata. By doing so, RansomTag is able to provide fine-grained recovery of user data, which can restore any single or multiple user files to any versions based on timestamps. Further, it can precisely distinguish the malicious writes of ransomware from concurrent writes issued by one or more benign applications to the user files, and restore the files even the malicious and benign writes intersect.

Overall, to achieve RansomTag, we need to solve three main challenges:

C1: How to precisely identify the ransomware's I/O requests in the hypervisor without losing any files? To address this issue, we develop a lightweight introspector that only obtains I/O-related context information (e.g., process and file activities) of the OS in the hypervisor layer, and implement a high-precision detection algorithm with this information. Further, we reserve all the user data at the very beginning of the ransomware detection algorithm to ensure that no files are lost (see details in § 3.3).

C2: How to bridge the semantic gap between the hypervisor and the SSD to transmit the detection results for fine-grained backup? To address it, we propose a simple but effective tagging approach, which travels the hypervisor and SSD, to mark each I/O operation from ransomware. Thus, the SSD can precisely retain the original data to be encrypted (see details in § 3.4).

C3: How to monitor the file system status in the hypervisor for file-level fine-grained recovery? We monitor (in the hypervisor) and back up (in the SSD) all the changes in files' metadata which are maintained by the file system in addition to the files' raw data. With the metadata, specified files and versions can be recovered successfully (see details in § 3.5).

To validate our approach, we develop a prototype of RansomTag with a dedicated thin hypervisor and a development board of SSD [46]. We collect 3,123 recent ransomware samples, including 3,061 Windows samples and 62 Linux samples, and run them in Microsoft Windows 10 and Ubuntu 22.04 respectively to evaluate our prototype. The evaluation results show that our system is able to successfully identify all the ransomware attacks and restore all the attacked files in fine-grained. In addition, it can resist potential targeted attacks on SSD characteristics. We further evaluate the impact of RansomTag on the I/O performance and lifespan of the SSD. The results show that the impact introduced by our prototype is acceptable. To engage the community, we plan to open-source RansomTag and release the dataset of ransomware samples in our evaluation at *https://github.com/XDU-SysSec/RansomTag*.

In summary, this paper makes the following contributions:

- We propose a novel ransomware defense approach called RansomTag, which is able to precisely detect and mark each I/O request issued by crypto ransomware in the hypervisor, and provide fine-grained user data recovery in the SSD.
- We develop a prototype of RansomTag, and implement the ransomware detection and data recovery to a thin hypervisor and a development board with modified SSD firmware, respectively.
- We perform a comprehensive evaluation to validate our approach. The evaluation results show that our prototype effectively protects user data from crypto ransomware attacks with minimal scale data backup and acceptable performance overhead, and the potential targeted attacks on SSD characteristics can be prevented. Meanwhile, all the attacked files can be completely restored in fine-grained.

2 BACKGROUND

2.1 SSD Intrinsic Characteristic

With the speed and reliability benefits, SSDs (Solid-State Drives) are gaining popularity for laptops, desktop personal computers, and servers. Specifically, SSDs divide the whole storage space into blocks, which are further divided into pages. Data is read and written at the page level, however, it has to be erased at the block level. Meanwhile, unlike HDDs (Hard-Disk Drives), SSDs must erase unneeded data blocks before new data can be written, which significantly impacts the I/O performance. To address this problem, modern SSDs always issue write operations to free pages that have been erased in advance (i.e., out-of-place update) rather than waiting for the expensive erase operation for each write. In other words, SSDs can only perform writes in an appending manner, leaving the obsolete data unaltered, which will be later erased (and recycled) by the GC (garbage collection) module [21]. In addition, the flash translation layer (FTL), which resides in the SSD controller, maintains an address mapping table and remaps overwritten logical block addresses (LBAs) to free physical space and marks invalid pages for GC. This intrinsic characteristic of SSD brings inspiration for backing up original data, which might be achieved by retaining those invalid pages where old data is located by delaying GC. As crypto ransomware always requires overwriting or deleting the original user files after being encrypted, if the GC process is modified to prevent erasing obsolete data, the attacked file data is naturally protected without additional overhead.

2.2 File Systems in Windows/Linux

To back up and recover the attacked files in fine-grained, we need to understand how a file system stores, organizes, and manages files and directories on a storage device. We focus on NTFS and ext4, which are the primary file systems in Windows and Linux, respectively.

New Technology File System (NTFS). NTFS is a proprietary journaling file system developed by Microsoft, and it is the primary file system for recent versions of Windows. In NTFS, the fundamental unit of disk usage is a cluster. To manage the files' information and their data content, NTFS maintains several files (called metadata files or metafiles) that define and organize the file system, and it uses names starting with '$' for these internal files. For example, the metafile $MFT represents Master File Table (MFT), a database containing information about every file and directory on an NTFS volume. There is at least one entry in the MFT for every file on an NTFS file system volume, including the MFT itself. All the information about a file, including its name, size, time and date stamps, permissions, and data content, is stored either in MFT entries or in the space outside the MFT indicated by MFT entries. An MFT entry is typically 1KiB in size, and every file or directory on a disk requires at least one MFT entry. In addition, the metafile $Bitmap keeps track of all used and unused clusters on an NTFS volume. Each bit in the $Bitmap represents one cluster. If that bit is set to 1, then the cluster is in use; otherwise, the cluster is free. When a file takes up the space on the NTFS volume, the corresponding bits in the $Bitmap are set to 1. To recover an attacked file in NTFS, besides its data content, its meta information stored in metafiles is also required to be backed up and recovered. Thus, the file can be reconstructed with its meta-information.

Fourth Extended File System (ext4). Ext4 is getting included in all the modern distributions of Linux. Particularly, ext4 allocates the storage space in units of the block, and each block is typically 4KiB in size. A block is a group of sectors between 1KiB and 64KiB, and a number of blocks are, in turn, grouped into larger units called block groups. The data block bitmap tracks the usage of data blocks within the block group. In ext4, each file and directory is associated with an inode, which is a data structure that keeps track of all the files and directories within ext4. Each inode is assigned an integer known as an "inode number." The inode bitmap records which entries in the inode table are in use. Note that in ext4, the filename is not directly stored in its inode. Instead, each filename is stored in the directory file as an entry, which has the filename (a string) and its corresponding inode number. Therefore, to recover an attacked file in ext4, we need to back up the file's metadata in the corresponding inode, the path information (including the filename) stored in each level of directory files, and the data content.

3 SYSTEM DESIGN

3.1 Assumptions and Threat Model

In this paper, we first assume that crypto ransomware leverages all the techniques that other types of malware might use to launch an attack. For example, ransomware can use zero-day vulnerabilities to compromise user systems and propagate the malware. Also, some crypto ransomware variants only encrypt a specific part of a file instead of aggressively encrypting the entire file to evade detection [27]. Moreover, certain advanced ransomware variants attempt to bypass a defense system by various methods. For example, they are able to escalate their privileges to the administrator by manipulating the access token [12, 37, 54], and then bypass the defense system in the kernel or attack the backup data in the storage device [21]. In the meantime, by attacking the virtual machine introspection (VMI) technique [17], they could prevent a VMI-based system from obtaining correct OS context information for identifying malicious behaviors.

On the other hand, the hypervisor and the SSD firmware are trusted. Thus, the defense system in the hypervisor and SSD would be protected and cannot be tampered with. Further, as a VMI-based approach, we need to introspect OS context information (e.g., process and file activities) from the guest OS for accurate attack detection. However, there is a strong semantic gap for the VMI technique [23], which is an open security problem that leads to getting incorrect information from an untrusted OS. Since we use the OS context information for attack detection, we need to get reliable data from the guest OS to identify attacks correctly. Note that to address this problem, several promising approaches have been proposed [23]. Accordingly, we assume that one such approach is integrated into our system. We will detailly analyze this security issue in § 5.4.1 and § 6.

3.2 Approach Overview

In this work, our goal is to detect crypto ransomware accurately and recover the user data in fine-grained. Traditional solutions work at the user or kernel level, which will be easily bypassed by advanced ransomware variants after gaining the system's administrator privileges [21, 54]. Instead, existing SSD-based systems put the ransomware detection functionality into the SSD firmware to avoid being evaded, but they suffer from inaccurate detection results due to the limited computing power and less context information, which leads to excessive [21, 47] or incomplete [4, 5, 43, 60] backup data. Moreover, these SSD-based systems [4, 5, 21, 43, 47, 60] hardly resist potential targeted attacks on SSD characteristics or introduce additional attacking surface for network threats [50].

To address the above limitations, we propose RansomTag, a hypervisor/SSD co-design defense system against crypto ransomware. Specifically, RansomTag introduces a hypervisor between the user OS and the SSD, and puts the ransomware detection functionality into the hypervisor. Thus, it is able to resist kernel-based attacks launched by privileged ransomware. In addition, it is able to achieve accurate detection results with powerful computing capability and rich context information of the host system. Note that to reduce the performance overhead, the hypervisor introduced by us is a lightweight one, which is based on a parapass-through architecture that only intercepts the interested file-related I/O operations. At the same time, we expand the functionality of the hypervisor and the SSD firmware to provide fine-grained data backup and recovery. We believe this is important since users often prefer to recover certain version files required by them quickly, rather than spending a lot of time recovering the entire mass storage device.

The overall design of RansomTag is shown in Figure 1.

Regularly, when a user process in the OS makes an operation to a file (e.g., open, create, read, write, close), it invokes the corresponding system call and switches from the user mode to the kernel mode. Accordingly, the OS kernel creates an I/O request to satisfy the file operation, which will then be sent to the file system. After receiving the I/O request, the file system issues its own more explicit request to the storage device driver [61]. The storage device driver eventually translates the request into a standard storage protocol command (like SCSI) and accesses the hardware registers, which are memory-mapped into system memory, to inform the storage device (i.e., SSD) to start direct memory access (DMA) transfer. According to the received I/O commands, the SSD completes the data access.

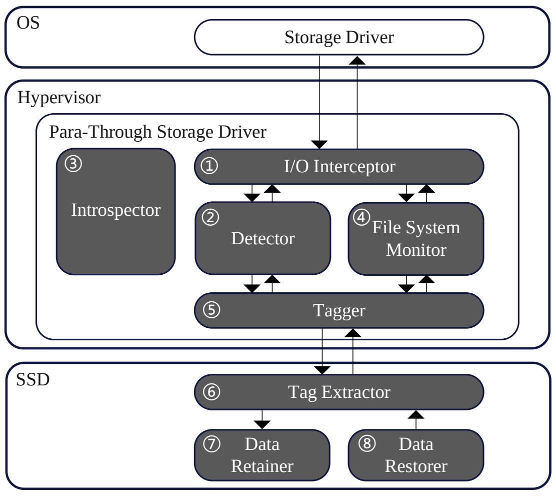


Figure 1: The overall design of RansomTag.

To achieve ransomware detection, we introduce two modules into the hypervisor, i.e., ① *I/O Interceptor* and ② *Detector*, as shown in Figure 1. Specifically, the *I/O Interceptor* module intercepts all the storage I/O commands before they are issued to the SSD by capturing the OS's accesses to the SSD's registers. The *Detector* module then identifies crypto ransomware with all the information from the intercepted commands through a ransomware detection algorithm. Particularly, to reduce the false positives and false negatives of detection, the *Detector* module requires as much information about process and file activities within the OS as possible.

However, there is a big challenge to address here, since the *Detector* module works in the hypervisor layer while the ransomware process is running in the OS kernel. Namely, there is a semantic gap between the hypervisor and the OS [14], which means the difference between the high-level OS abstractions from the internal OS and the hardware-level abstractions from the hypervisor. For instance, in the OS we can see semantic-level objects such as processes and files, while in the hypervisor, we only see memory pages and storage protocol commands of the OS. Then, how can we precisely identify the I/O request issued by ransomware process in the hypervisor without losing any files (i.e., $C1$ as mentioned in § 1)? To address this challenge, we introduce a new module in the hypervisor, i.e., ③ *Introspector* as shown in Figure 1, to obtain the context information of file operations from the CPU and memory space in real-time, which will be sent to the *Detector* module for high-precision ransomware detection.

To achieve data backup and recovery in fine-grained, we introduce two modules into the SSD firmware, i.e., ⑦ *Data Retainer* and ⑧ *Data Restorer*, as shown in Figure 1. Specifically, the *Data Retainer* module leverages the SSD's intrinsic characteristic (i.e., out-of-place update) to retain the original data without extra overhead, and the *Data Restorer* module reads out the retained data as needed to recover the user files.

However, due to the lack of high-level semantic information in the SSD, we have two challenges for data backup and fine-grained data recovery. *First*, it is a challenge to inform the SSD of detection results since there is a semantic gap between the hypervisor and the SSD, i.e., $C2$ as mentioned in § 1. Specifically, the hypervisor follows the storage protocol when transferring data to the SSD, and the SSD cannot receive any data other than the storage commands. Therefore, the backup/recovery module in the SSD firmware could hardly precisely determine whether the incoming storage commands need to be handled to retain the original data. To address this challenge, we propose a simple but effective tagging approach. Specifically, we introduce two modules to interconnect the hypervisor and the SSD, i.e., ⑤ *Tagger* and ⑥ *Tag Extractor*, as shown in Figure 1. The *Tagger* module resides in the hypervisor and marks the related I/O commands with a tag according to the ransomware detection results, which will be sent to the SSD for further processing. Then, the SSD can accurately identify whether each incoming command is from ransomware or not after the *Tag Extractor* module in the SSD extracts the tag. Thus, the *Data Retainer* module is able to back up the data accordingly.

Second, to achieve the goal of fine-grained recovery, we need to monitor and record the changes in the file system in the hypervisor. However, it is another challenge to monitor the file system status in the hypervisor, i.e., *C*3 as mentioned in § 1, because the concept of "file" only exists in the OS, while we can only extract the states of the file system from the captured I/Os in the hypervisor. Specifically, the file's metadata is indispensable when recovering a file since all the attributes of the file (e.g., file name, date modified, the data location) are recorded as a metadata entry stored in the file system's metafile. However, the file's metadata will be updated at the file system level when the file is attacked by ransomware, and the changes of metadata can not be captured directly in the SSD for backing up. To solve this problem, we introduce a new module in the hypervisor, i.e., ④ *File System Monitor*, as shown in Figure 1, to monitor file metadata changes by capturing the I/O commands targeting the file system's metafiles. Specifically, the *File System Monitor* module examines the read commands among these I/O commands for possessing the current state of the file system in the SSD. In addition, it filters their write commands to let the *Tagger* module stamp them with a tag for backing up the previous metadata.

Next, we present the design of ransomware detection, data backup, and data recovery detailedly in § 3.3, § 3.4, and § 3.5, respectively.

3.3 OS-Isolated and High-Precision Detection

The ransomware detection functionality of RansomTag resides in a Type I hypervisor, which is isolated from the attack-prone operating system [2, 32, 53]. However, most of the well-known Type I hypervisors, such as VMware ESXi, Microsoft Hyper-V, and Xen, are not suitable for our system since they are pretty large and complex. For example, these hypervisors include numerous device drivers and resource manager to support various devices as well as multiple virtual machines (VMs), which cause significant performance overhead. Therefore, we introduce a lightweight Type I hypervisor, which gets rid of the functionalities of multiple VMs support and lets most accesses pass through directly except for storage I/O or memory-related operations.

3.3.1 Context Information Obtaining. We introduce an *Introspector* module in the hypervisor to obtain the context information of the OS. The *Introspector* module utilizes the VMI technique to retrieve information about the current state of the OS. Compared to other VMI systems [14, 16, 48], the *Introspector* module is a lightweight (or thin) one, which only obtains I/O-related context information when the I/O command is intercepted. Specifically, it examines the physical memory of the OS and leverages detailed knowledge of the OS's algorithms and data structures to rebuild higher-level I/O-related information, such as the list of the running processes, and the files operations.

3.3.2 Ransomware Detection Algorithm. Previous studies [27, 29, 52] have demonstrated that significant changes occur and exhibit distinctive, repetitive patterns in file system activities when the OS suffers a ransomware attack. Therefore, RansomTag takes full advantage of the host's computing power and rich context information to implement the state-of-the-art ransomware detection algorithm [27, 52, 54], which combines the file activities (such as read, write or overwrite, rename or delete files) and the entropy of the file's data as the patterns of crypto ransomware.

Specifically, the detection algorithm detects crypto ransomware from two aspects. *First*, it monitors whether the process matches the file behavior patterns. *Second*, it regards as a suspicious process if the entropy of the data to be written is high. It is worth pointing out that compressed data (e.g., zip files, video stream data, and pictures) also tends to exhibit high entropy. To address that, we utilize a state-of-the-art approach [13] to reliably distinguish the compressed and encrypted data. If the above two conditions are met, the process is immediately regarded as crypto ransomware. Unlike the existing solutions that block the ransomware process directly, RansomTag does not interfere with all the read and write operations of the ransomware process. On the contrary, RansomTag transparently marks ransomware's write commands with a special tag. Note that a small number of files may be encrypted (and lost) before the ransomware attack is detected. To avoid losing data, we tag all the write requests with high entropy data as well until the process is determined as benign. It is worth pointing out that the ransomware detection algorithms and the framework of our system are orthogonal. Namely, to improve the detection accuracy, all the existing and further advanced detection algorithms (e.g., machine learning [47], decoy technology [35], and others [28, 52]) can be readily deployed in our system.

3.4 Fine-Grained Data Backup

By utilizing the SSD's intrinsic characteristic (i.e., out-of-place update), RansomTag is able to back up the original data by excluding the retained blocks from GC. Further, in RansomTag, the SSD can accurately retain the data manipulated by crypto ransomware due to the high-precision detection algorithm adopted by us, which reduces the scale of data backup significantly. The details of the data backup are shown in Figure 2. In particular, in the flash controller of an SSD, the FTL maps the host side or file system logical block addresses (LBAs) to the physical address of the flash memory (logical-to-physical mapping). In general, a free page is allocated to the new data from a write command, and the physical address of the page is updated in the FTL to map to the LBA carried in the command (❶). If the LBA has already been mapped to a physical page (i.e., for an overwrite operation), the previous page would be marked as invalid and erased later by GC (❺). Similarly, if a read command arrives, the FTL searches the mapping table through the

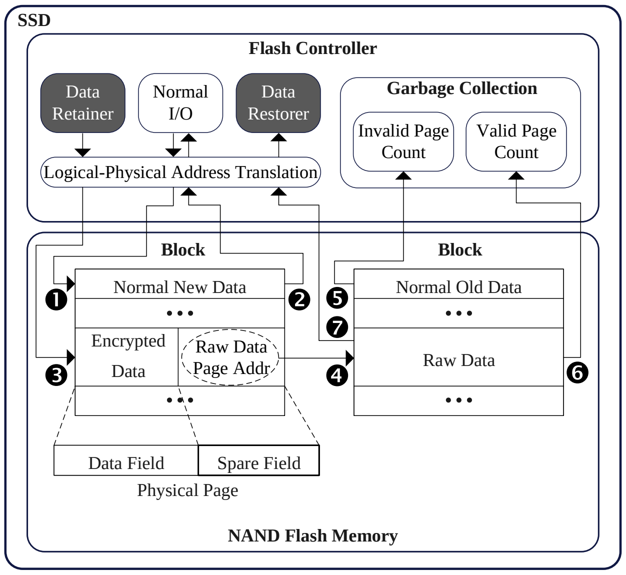


Figure 2: The details of workflows in the SSD.

LBA in the command and reads the data from the corresponding physical page (❷). As mentioned in § 3.3, the write commands from ransomware are stamped with a special tag in the hypervisor. The *Tag Extractor* module in the SSD forwards the tagged write commands to the *Data Retainer* module. Same as the regular writes, a series of free pages are allocated to store the new incoming (encrypted) data (❸). However, the difference is that the physical addresses of the previous pages that mapped to the same LBA are written into the spare fields of these new pages synchronously with the encrypted data. Note that the commodity SSDs typically reserve 16-64 bytes out-of-band (OOB) metadata for each physical page [21], which is also called spare field [51]. Therefore, the spare field of a page is enough to contain the physical address of a page. In other words, a pointer is stored in the spare field of the encrypted data page, which points to the page of raw data before being encrypted (❹). Regularly, these raw data pages will be marked as invalid and erased later by GC. However, to retain these pages, we modify the GC module to treat these pages as valid (❻). Thus, all the original data that needs to be protected is retained in the SSD.

It is worth pointing out that with the introspected context information, RansomTag can precisely distinguish the malicious writes of ransomware from concurrent writes issued by other benign processes. Thus, by tagging each write command in the hypervisor, RansomTag can clearly inform the SSD whether each targeted logic block data needs to be retained, which achieves fine-grained data backup. Later, the user files are able to be successfully restored even the malicious and benign writes intersect, and all the benign data can be reserved.

3.5 Fine-Grained Data Recovery

3.5.1 File System Monitoring. To reconstruct a file from the backup data in the SSD, the meta information of the file is indispensable apart from the data content. As mentioned in § 2.2, in NTFS, all the information about a file is stored in an entry of the $MFT metafile. Thus, we only need to monitor and record the changes of the $MFT metafile. Besides, to ensure the logic units (i.e., the clusters) containing the protected data are not used again, we also need to monitor the status of the $Bitmap metafile. While for ext4, the inode table, which stores all inode entries, needs to be monitored and backed up. Similarly, to protect the blocks that keep the backup data, the status of the data block bitmap must also be monitored.

To achieve the above procedures, we introduce a *File System Monitor* module to recognize the write commands targeting the file system metafiles, and inform the *Tagger* module to stamp a special tag on these write commands. By doing so, the metadata of the original files will later be retained in the SSD for fine-grained data recovery.

3.5.2 Fine-Grained Data Recovery. As mentioned in § 2.2, two key factors are indispensable for rebuilding a file to any version, i.e., the raw data of the file and its metadata. In RansomTag, both of them are retained in the SSD. To recover the files, the recovery program designed by us reads out all the retained versions of the metafiles (i.e., $MFT for NTFS, and inode table for ext4) from the SSD and parses them. Then it lists all the file records, and each of them represents a version of a file. Users can manually choose specific versions of the files they want to retrieve according to the listed records. In addition, users can also specify multiple versions or files in bulk for efficiency. The recovery program eventually reads out the data of the specified files from the SSD and rebuilds the files.

Reading File's Metadata. The recovery program initiates a read request targeting the metafiles. The read request is translated into a read command by the OS and intercepted by the *I/O Interceptor* module in the hypervisor. With the context information, the read command is recognized by the *Detector* module and then stamped with a special tag by the *Tagger* module. In the FTL of the SSD, the command is forwarded to the *Data Restorer* module according to the tag. The LBA in the read command has been mapped to the newest data page (the latest version of metafiles). However, the overwrites to the metafiles are captured, and the previous page has been retained. Therefore, the physical page corresponding to the LBA in the read command has a pointer in its spare field, which points to the retained page (❹ in Figure 2). The *Data Restorer* module eventually reads the previous version of the metafiles from the retained page (❼ in Figure 2). The recovery program iterates the above process until all versions of the metafiles have been read.

Reading File's Data. For NTFS, the small files ($<$ 1KiB) are stored entirely within their MFT records, while the files that do not fit within a record are allocated clusters outside the MFT. Similarly, for ext4, the data of small files ($<$ 60 bytes) is directly stored in inodes [26]. Therefore, the small files can be recovered immediately as soon as all the versions of the metafiles are read.

For large files, our recovery program first parses all the versions of file records (MFTs or inodes) and extracts the attributes of each file, such as filename, date modified, and logic addresses of the file data. Particularly, since the filename is stored in a directory inode in ext4 instead of contained in its inode, our recovery program must traverse all directory inodes to get the files' name for ext4 files. According to these attributes, the user chooses what files and versions need to be restored instead of recovering the entire SSD. After the user confirms the files and versions that they want to restore, the recovery program initiates a series of read requests to require the data located in the logic units (i.e., the clusters for NTFS and the blocks for ext4) of the files that the user chooses. In the SSD, the mapped physical pages of the LBAs in these read commands store the pointers of the raw data pages (retained pages) in their spare fields. Finally, the *Data Restorer* module reads the raw data from the retained pages for recovery (❼ in Figure 2).

4 IMPLEMENTATION

To validate our approach, we develop a proof-of-concept prototype of RansomTag. We modify and integrate an open-source Type I hypervisor called BitVisor [53] to implement the hypervisor-side functionalities of RansomTag. Specifically, BitVisor only supports a single VM simultaneously running on it, which is allowed to access the real hardware directly. The storage driver in the BitVisor, which is called a parapass-through driver, is used to handle intercepted I/Os for detecting ransomware. We add the lightweight *Introspector* module (as shown in Figure 1) into the storage driver for obtaining the OS context information in the course of I/Os being intercepted and issued. Moreover, we integrate the *Detector*, the *File System Monitor*, and the *Tagger* modules into the storage driver of BitVisor to achieve crypto ransomware detection, file system monitoring, and I/Os tagging, respectively. Furthermore, we expand BitVisor to support floating-point arithmetic for computing the entropy of the data to be written.

Additionally, we integrate the *Tag Extractor*, *Data Retainer*, and *Data Restorer* modules into an open-source FTL project called OpenNFM [19]. Then, we port the modified FTL to a development board, i.e., LPC-H3131 [46], equipped with 180MHz ARM microcontroller, 512MiB NAND flash, and 32MiB SDRAM as the evaluation SSD. Besides, we develop a Python program tool to recover the attacked files in fine-grained. Overall, the prototype of RansomTag consists of about 4, 200 lines of C code and 1, 690 lines of Python code.

4.1 Key Techniques in the Hypervisor

4.1.1 Context Information Obtaining. The *Introspector* module of RansomTag is used to obtain the context information of the OS for precise detection, including file-related operations, process structure, file object, etc. The paths to obtain the process and file information of 64-bit Windows 10 and 64-bit Linux are shown in Figure 3.

Particularly, in 64-bit Windows 10, it obtains the process context by directly accessing the thread pointer stored in the CPU as long as the I/O commands are intercepted. It is because a thread is a basic unit to which Windows allocates processor time [39], and it is the entity within a process that Windows schedules for execution [61]. Thus, the process context of Windows can be obtained from a thread since one or more threads run in the context of the process. As shown in Figure 3(a), in 64-bit Windows 10, each process represented by an executive process (EPROCESS) structure has one or more threads, and each thread is represented by an executive thread (ETHREAD) structure [61]. Meanwhile, an I/O request packet (IRP), which is represented by an IRP structure, is where the I/O system stores the information that it needs to process an I/O request [61]. The IRP structure contains a FILE_OBJECT structure used by the system to represent a file object, and a file object represents an open instance of a file [41] and contains the file-related information we need. In addition to an EPROCESS structure, the

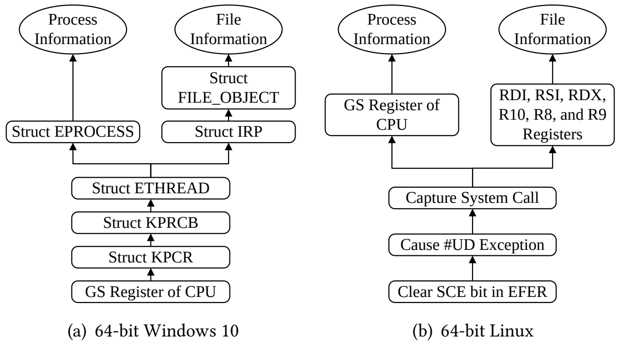


Figure 3: The path to obtain the OS context information.

ETHREAD structure also includes an IRP list to represent the IRP sequence of the process. When an I/O command is intercepted, the processor has not yet switched and still stays in the context of the process that issued the I/O. Therefore, we can obtain the context information (e.g., process, file object) associated with the incoming I/O commands, as long as the pointer of the current ETHREAD structure is obtained. The pointer of the current ETHREAD structure can be obtained in the kernel processor control block (KPRCB), a member of the kernel processor control region (KPCR) structure. The address of the KPCR structure can be read from the GS register in the kernel mode (ring 0), since in the kernel mode the value in the GS register is swapped from the KernelGSBase model specific registers (MSRs) by the SWAPGS instruction [11].

In 64-bit Linux, however, the above method is not applicable since the processes and threads in Linux are implemented differently from Windows. Specifically, in Linux kernel space, processes and threads are implemented through a universal structure, called a task (hence the struct task_struct) [6, 25]. Thus, we can only obtain a task structure pointer from the CPU register, which could be either a user process or a kernel thread. However, since the Linux kernel uses a set of kernel threads (called flusher threads) to sync dirty pages to the disk [33], the current CPU context when an incoming I/O command is intercepted may be a kernel task of the flusher thread instead of the user process that initiates I/O requests. To address this problem, RansomTag captures file-related system calls (e.g., open, read, write, close, mmap, rename) and corresponding process structure (i.e., the struct task_struct) to identify the ransomware-like behaviors, and marks the I/O commands issued by the detected process. The details are shown in Figure 3(b). To capture system calls of the guest OS in the hypervisor, we need to make those system calls trap to the hypervisor. Note that on the x64 platform, system calls are implemented by SYSCALL/SYSRET instructions [11]. This feature is controlled by the SCE bit of the Extended Feature Enable Register (EFER). If we clear this bit, when a system call is invoked, the OS will cause an Undefined Opcode Exception (#UD) and trap to the hypervisor for handling the exception. Thus, we can capture the system call and obtain the system call number from the RAX register. If the system call is file-related, we obtain the file operation details from the system call's parameters stored in RDI, RSI, RDX, R10, R8, and R9 registers [57], including file descriptor, filename, read/write length, data buffer, etc. Meanwhile, the struct task_struct pointer can be obtained from the GS register.

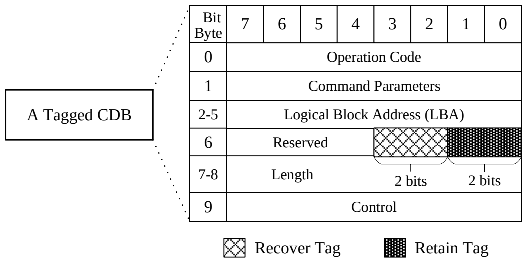


Figure 4: The location of the tags in a CDB.

After introspection, the hypervisor emulates the SYSCALL/SYSRET operations and returns to the OS.

4.1.2 Tagging Approach. To bridge the semantic gap between the hypervisor and the SSD, we propose the tagging approach (see § 3.2). Specifically, the *Tagger* module in the hypervisor sets the reserved field of an incoming storage command as a meaningful tag, while the *Tag Extractor* module in the SSD extracts the tag from the received command and issues it to the corresponding module. In the prototype, we implement two types of tags, i.e., the "retain" tag and the "recover" tag. In particular, the write commands with a "retain" tag indicate that the related data must be retained, and the "recover" tag is dedicated to the read commands issued by our recovery process for data restoration.

In RansomTag, the tags are labeled on the SCSI protocol commands. In the SCSI standards, the commands are sent in a command descriptor block (CDB) which is 10 bytes long generally. The details of the CDB structure are shown in Figure 4. The byte 0 of a CDB contains an *Operation Code* identifying the operation type (e.g., read or write) being requested by the CDB. We choose the reserved byte 6 of CDB to store the tags. Specifically, the "retain" tag is represented by bits 0 and 1 of byte 6, and the "recover" tag is represented by bits 2 and 3. Note that since the tagging is performed transparently in the hypervisor layer, it is difficult for the ransomware process in the OS to tamper with the tags or bypass this step.

4.1.3 Entropy Calculation of Data Transferred via DMA. As mentioned in § 3.3.2, RansomTag requires to calculate the entropy of the written data to each file for ransomware detection. Particurlaly, the Shannon entropy is computed as: $H = \sum_i -p_i \log_2 p_i$, where p_i is the frequency of the value of the i-th byte in the array. However, it is not easy to determine that the written data belongs to a certain file since the data is transferred through direct memory access (DMA), which is a mechanism to transfer the data between the devices and the memory without processor intervention. In other words, a process switch may occur in the CPU at the stage of data transfer, and the ransomware process may be replaced by another process, which means it is hard to determine which file the data transferred via DMA came from due to the uncertainty of context information. To address this issue, RansomTag parses the intercepted write command, extracts the starting LBA stored in bytes 2-5 of the CDB, and calculates the ending LBA based on the transfer length stored in bytes 7-8 of the CDB (as shown in Figure 4). With the exact context information obtained at the arrival time of the write command, RansomTag records the file object and the LBA range of the written data. At the data transfer stage, RansomTag compares the targeting LBA of the data to be transferred. If the targeting LBA is within the starting and ending range of the recorded LBA, the frequency of the byte value in the data is accumulated. In contrast, if the targeting LBA is out of range, RansomTag considers the data transfer of the previous file completes, and it calculates the final entropy value of that file.

4.1.4 File System Monitoring. As mentioned in § 3.5.1, the file's metadata is indispensable for rebuilding the file. Specifically, the changes of the $MFT and $Bitmap metafiles for NTFS, as well as the inode table and block bitmap for ext4, must be backed up. To back up the changes of $MFT or inode table, the *Tagger* module directly tags the write commands targeting the $MFT metafile or inode table. Thus, the data of the $MFT or inode table to be overwritten can be retained in the SSD. However, it is complex to monitor the operations of the $Bitmap metafile of NTFS and the block bitmap of ext4. Specifically, the *File System Monitor* module maintains a bitmap to record the clusters for NTFS or the blocks for ext4 that need to be protected. *First*, the initial usage status of the $Bitmap metafile or block bitmap must be recorded when the SSD is loaded at the boot time. To achieve it, the *File System Monitor* module intercepts the OS's read commands targeting them when the SSD is loaded. Thus, RansomTag can obtain the initial version of the two bitmaps. Next, the *File System Monitor* continuously monitors the changes of them and records all changes in the bitmap it maintains. *Second*, the *File System Monitor* module double-checks the write commands identified as normal by the *Detector* module. If the write commands aim at the protected clusters/blocks recorded in the bitmap, they are also labeled with the "retain" tag by the *Tagger* module.

4.2 Key Techniques in the SSD

4.2.1 Traveling Back to the Raw Pages. In RansomTag, for data recovery, the SSD needs to map the raw pages to be overwritten by the encrypted data and the pages storing the encrypted data. As illustrated in § 3.4, the *Data Retainer* module leverages the spare field of a page to store the physical page address (PPA), which needs to be retained.

For instance, as shown in Figure 5, a part of the original data of a user file is stored in the LBA 05, which has been mapped to the PPA P0. Crypto ransomware attempts to overwrite the file with the encrypted data. The write command, which carries the encrypted data to LBA 05 is detected by the *Detector* module in the hypervisor and is tagged with the "retain" tag by the *Tagger* module. In the SSD, the FTL allocates a new physical page (P8) for the incoming write command, and the mapping of the LBA 05 in the L2P mapping table is set to P8. When the encrypted data is written to the data field of P8, the address of page 0 (P0) is synchronously written to the spare field of P8.

4.2.2 Delaying Garbage Collection. To prevent the retained pages from being erased by GC, we modify the GC module to exclude these pages. Specifically, in OpenNFM, the GC module maintains two counts, i.e., the invalid page count and the valid page count, to record the validity of pages in each block. When the GC is triggered, it scans the counts to find all the blocks with invalid pages as the erasing candidates. Since the unit for erasing is the block, the valid

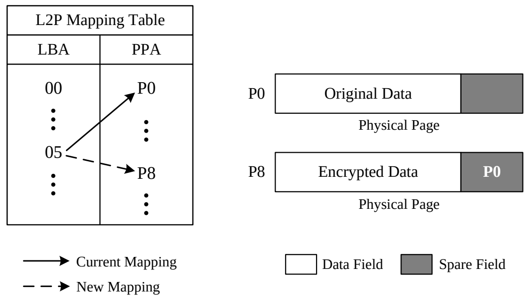


Figure 5: Travelling back to the raw pages.

pages in the candidate blocks to be erased are copied to new blocks. In RansomTag, the blocks which have pages to be retained by us are excluded from GC.

5 EVALUATION

5.1 Experimental Setup

In our experiments, we run the crypto ransomware samples in 64-bit Microsoft Windows 10 and 64-bit Ubuntu 22.04 (with Linux kernel 5.15), respectively. It is because Windows has been the most widely used and attacked operating system in the world [18], while Linux is becoming an increasingly popular target for ransomware authors due to the high value of the devices it powers [36]. We utilize two desktop personal computers (PCs) with Intel 10700k CPU running at 4.0GHz and 32GB DRAM. One is used to run the samples, called the experimental PC. The other is dedicated to data recovery, called the recovery PC. The development board [46] with the modified FTL is plugged into the experimental PC as the evaluation SSD.

Since crypto ransomware encrypts user files for extortion, we build a user document directory consisting of various types of files, including documents, media files, archive files, etc. It is worth pointing out that the files in the directory are collected from real-world users' computers and do not contain any sensitive personal data. Further, to run the malware samples successfully, we turn off the anti-virus software, firewall, and any other security policies in the OS. At the same time, the OS can access the Internet freely so that the samples are able to communicate with their remote servers.

To evaluate that RansomTag can detect crypto ransomware from malware samples in the real world, we collect 3,123 recent ransomware samples (including 3,061 Windows samples and 62 Linux samples) from VirusTotal [59] and VirusShare [58] by using ransomware-related search terms (e.g., ransomware, ransom, etc.) or known variant names. For each sample, we first copy the user directory to the experimental PC, and then we run the sample for at least 30 minutes. After a round of the experiment, we reset the experimental PC and the SSD. Then, we recopy the user directory to the SSD for the next round.

To verify the effectiveness of file recovery, we perform four types of recovery experiments (see § 5.3). In addition, we analyze the attack surface of RansomTag and evaluate its effectiveness of resisting potential attacks (see § 5.4). Further, to evaluate the impact of RansomTag on I/O performance, we perform several micro-benchmark and macro-benchmark experiments, respectively (see § 5.5). Finally, we calculate the wear leveling inequality to evaluate the impact of RansomTag on the SSD's lifespan (see § 5.6).

5.2 Effectiveness of Detection

The results of the experiment in 64-bit Windows 10 show that RansomTag successfully identifies 1,332 crypto ransomware samples from 24 families in our 3,061 malware dataset for Windows. Meanwhile, the results of the experiment in 64-bit Ubuntu 22.04 show that RansomTag successfully identifies 19 crypto ransomware samples from 6 families in our 62 malware dataset for Linux. In addition, in the experiments in both OSes, all the write commands initiated by these samples for encrypted data are tagged. The detected crypto ransomware families and the detected numbers in these families are shown in Table 1. Note that even if a sample is labeled as ransomware by one or more anti-virus software, it does not mean that the sample is active crypto ransomware and will encrypt user files. For instance, the sample may be screen locker ransomware that extorts the user by locking the OS, or it cannot encrypt user files due to the failure to receive the symmetric key from the remote server.

Table 1: The list of detected crypto ransomware families and numbers.

Windows Samples			Linux Samples		
Family	**Number**	**Rate**	**Family**	**Number**	**Rate**
REvil	478	35.89%	REvil	6	31.58%
Cerber	316	23.72%	AvosLocker	6	31.58%
Djvu	312	23.42%	HelloKitty	4	21.06%
Conti	68	5.11%	Cylance	1	5.26%
Razy	63	4.73%	Buhti	1	5.26%
FileCoder	27	2.03%	IceFire	1	5.26%
CryptoLocker	13	0.98%			
Kryptik	10	0.75%			
Mikey	9	0.68%			
GenericKD	7	0.53%			
Kazy	7	0.53%			
AgentWDCR	3	0.23%			
Ranapama	3	0.23%			
Zusy	3	0.23%			
WannaCryptor	3	0.23%			
VirLock	2	0.15%			
Others (8 families)	8	0.60%			
Total (24 families)	1,332	-	**Total (6 families)**	19	-

False Positives. As our sample dataset is not a ground truth set of crypto ransomware, we need to confirm whether each detected sample is a true or false positive. However, re-executing each sample is not feasible to evaluate false positives since a sample may fail to encrypt files again because of the network or remote server problem [27]. To address this issue, we check the user files in the SSD after each sample runs. If the sample is detected as ransomware, and at least one file is encrypted, we consider the sample to be true positive. On the contrary, if the detected sample does not encrypt any user file, it is considered as a false positive. We leverage a Python script to automatically check whether a user file in the SSD is encrypted. Specifically, the script first calculates the entropy of the file. If the entropy is high, it further distinguishes the compressed and encrypted data by utilizing a state-of-the-art approach [13] since the compressed data (including zip files, video stream data, pictures, etc.) also exhibits high entropy. By doing so, the script can find out the encrypted files in the SSD. Eventually, we find all the detected samples encrypt at least one file. It indicates that there is no false positive in our results. In addition, for each true positive sample, which has encrypted the user files, we perform full drive recovery to evaluate the recovery capability of RansomTag. The results show that all the attacked files can be recovered completely, which means the success rate of file recovery is 100% (see details in § 5.3).

False Negatives. The experiment results show that the rest 1,772 samples from our sample set are treated as non-crypto ransomware by RansomTag. To verify if these samples have false negatives, we calculate the hash value of every file in the user document directory after each sample runs, and then compare the calculated results with the hash values of their original files. The results show that all the files' hash values are not changed, which means all these files are not tampered with. Therefore, the false negative rate is approximated to zero. Note that it is an approximation because the samples treated as non-crypto ransomware by RansomTag may be crypto ransomware, while they do not perform any behaviors of crypto ransomware in our experiment. As described early, there are many reasons for this. For example, the sample may be screen locker ransomware, or it cannot encrypt user files due to the failure of the network. However, as a dynamic detecting system, RansomTag can identify attacks immediately as long as the sample exhibits crypto ransomware behaviors.

5.3 Effectiveness of File Recovery

To evaluate the effectiveness of fine-grained file recovery, we perform four types of recovery experiments, i.e., files recovery of full drive, specified files recovery, specified file versions recovery, and file recovery under concurrent benign writes, after each detected sample has successfully encrypted user files. After each attack, we plug the SSD into the recovery PC and run the recovery program to perform the experiments. To confirm the integrity of the recovered files, we use a python script to calculate the hash values for each recovered file and its original one, and compare the two calculated hash values. If the two values are equal, then we think that the recovered file is the same as the original one.

Files Recovery of Full Drive. This experiment is designed to evaluate whether RansomTag can fully recover all the attacked files on the drive. For each true positive ransomware sample, we perform a full drive recovery after it successfully encrypts the files. The recovery program reads and parses all the retained metadata entries (i.e., MFT records for NTFS or inodes for ext4) of the files in the list from the SSD. For small files ($<$ 1KiB in NTFS or $<$ 60 bytes in ext4), as mentioned in § 3.5.2, their data is directly stored in their metadata entries. Hence, these files are rebuilt as long as the retained metadata entries are read out. While for large files whose data is stored in the data area (the clusters for NTFS or the blocks for ext4), their data is read out after the data addresses are parsed. Finally, the recovery program calculates the hash value of each recovered file and compares the result with its original one. The results show that the hash values of all recovered files are the same as their original ones. Thus, it indicates that RansomTag successfully recovers all attacked files in the SSD.

Specified Files Recovery. In this experiment, we run the recovery program after the attacks, and randomly specify 100 files among all the files in the SSD to be recovered. As a result, the 100 files we specified can be integrally recovered. It indicates that RansomTag can recover specified files instead of the entire drive.

Specified File Versions Recovery. In this experiment, we choose 100 detected crypto ransomware samples that repeatedly overwrite the original file multiple times, and run the samples again in turn. After each sample attack, we run the recovery program and rebuild each tampered version of a file. By confirming manually, the part that has not yet been encrypted in each file version is completely recovered. It indicates that RansomTag can recover not only the entire file, but also a portion of the file. This feature allows RansomTag to cope with more complex scenarios.

File Recovery under Concurrent Benign Writes. As mentioned in § 3.4, RansomTag achieves fine-grained data backup, and it can restore the user files even when malicious and benign data writes intersect. To evaluate it, we leverage a Python script to invoke three benign applications (i.e., 7zip, Notepad, MySQL) and let them issue concurrent write operations during the ransomware sample attack. After the attack, we recover the attacked files on the drive. The experiment results show that all the attacked files are completely restored, including the original data and the new written data by the benign applications. It indicates that our system is able to recover user files in fine-grained, even under concurrent benign write circumstances.

Regarding the recovery time, as our recovery program can recover one or more single files, compared to those SSD-based approaches that need to scan all the retained data pages, the recovery time of RansomTag is flexible. In other words, it depends on the number of files the user wants to recover and the read performance of the storage device.

5.4 Security Analysis and Evaluation

In this subsection, we analyze the attack surface of RansomTag and evaluate its effectiveness of resisting potential attacks.

5.4.1 Attack Surface. First, it is worth pointing out that since RansomTag resides in the hypervisor and the firmware of the SSD, our system is naturally immune to the widely spread privilege escalation attacks inside the OS, which attempt to bypass the defense by obtaining administrator privileges of the OS [21, 54]. Second, for the existing SSD-based solutions, an attacker may evade their detection algorithms which only monitor the I/O stream patterns [4, 5, 60] or compute the entropy of data content [43, 47]. For example, a ransomware variant may intentionally issue modified I/O patterns to evade the I/O pattern-based detection algorithm [47]. Similarly, an advanced attacker could also designedly diminish the entropy value of the data by encrypting only a part of the file or inserting low-entropy data at regular intervals. However, RansomTag can defeat these evading attacks since it adopts a comprehensive detection algorithm that monitors multiple indicators simultaneously, including file-related operations, data entropy, etc. In addition, relying on the powerful computing capability of the host, RansomTag can introduce new advanced detection algorithms if needed.

Theoretically, based on the functionality and components of our system, RansomTag faces three potential attack vectors, i.e., targeted attacks on the detection functionality, targeted attacks on the tagging mechanism, and target attacks on the backup and recovery module. Next, we analyze these potential attacks respectively.

Targeted Attacks on the Detection Functionality. The first type of potential attack on the detection functionality is that privileged ransomware may compromise the OS by targeted attacks on

VMI to prevent the *Introspector* module of RansomTag from getting the correct context information of the OS. Such attacks include kernel object hooking attacks, dynamic kernel object manipulation attacks, and direct kernel structure manipulation attacks [23]. Since bridging the semantic gap between the hypervisor and an adversarial, untrusted OS is still an open security problem, the *Detector* module of RansomTag may produce false detection results under such attacks. To address this problem, we need to integrate one mitigation approach into our system. We will detailly explore the promising mitigations for this issue in § 6.

The second type of potential attack on the detection functionality is the time-of-check-to-time-of-use (TOCTOU) attack. However, our system is immune to such an attack. This is because the data flow is constrained in the trusted hypervisor from the data being introspected to being used by the *Detector* module.

Targeted Attacks on the Tagging Mechanism. Since RansomTag marks the I/O commands in the hypervisor to send the retaining or restoring operation instructions to the SSD, an adaptive attacker may attempt to bypass or distort our system by attacking the tagging mechanism. For example, they may tamper with the tagged commands before the SSD extracts the tags from the incoming commands. However, such attacks are virtually impossible to implement in our system. The reason is that RansomTag marks the I/O commands within the trusted hypervisor and extracts the tags in the trusted SSD.

Targeted Attacks on the Backup/Recovery Module. Since RansomTag retains and recovers the attacked data by leveraging the SSD intrinsic characteristic (i.e., out-of-place update), this intrinsic characteristic is also a point that can be used by ransomware to perform targeted attacks. There are three potential attacks, namely the GC attack, the timing attack, and the TRIM attack [50].

For the GC attack, an attacker can exhaust the SDD's free space and force it to wipe the retained data by GC. However, such an attack will not succeed in our system since RansomTag blocks the I/O requests from the detected ransomware process if the SSD's available space is insufficient. Thus, the ransomware has no chance of triggering GC, and the retained data will not be released.

For the timing attack, certain advanced ransomware can deliberately slow down encryption in order to hide its I/O patterns to evade detection. Such an attack will bypass the SSD-based defenses that detect ransomware by only monitoring the I/O patterns. However, RansomTag makes this evading strategy in vain because it implements a comprehensive detection algorithm based on high-level context information (e.g., process, file activities, and file data). Previous studies [27, 54] have demonstrated that this dynamic behavior-based detection algorithm is able to identify ransomware as long as it performs crypto ransomware behaviors. In addition, with the powerful computing capability of the host system, RansomTag is able to integrate advanced algorithms to improve detection efficiency.

For the TRIM attack, ransomware attackers can utilize the SSD TRIM command to destroy the retained data. Specifically, a TRIM command, which is known as TRIM in the ATA command set and UNMAP in the SCSI command set, allows an operating system to inform an SSD that blocks of data are no longer considered to be in use and can be erased internally. Unlike HDDs (Hard Disk Drives), an SSD cannot overwrite a page of flash storage before erasing the entire block of storage in which the page is located. This behavior introduces a performance issue for I/O writes to previously used blocks of data when compared with I/O writes to unused or erased blocks [22]. To improve the performance, the file system can issue a TRIM command to the SSD to notify which blocks of data are no longer in use and can therefore be erased. By issuing the TRIM command after encryption, ransomware can speed up the reclamation of the original data blocks to prevent the original data from being retained. However, RansomTag can defend against this attack. It is because, in RansomTag, a ransomware process will be quickly detected as long as it encrypts user files. Thus, if the ransomware process issues a TRIM command, RansomTag will block it.

5.4.2 *Effectiveness of Resisting Targeted Attacks on SSD Characteristics.*

To evaluate the effectiveness of resisting targeted attacks on SSD intrinsic characteristics, we develop three ransomware prototypes for each attack by modifying an open-source project [62]. Specifically, these prototypes not only encrypt the user files like other crypto ransomware, and each of them performs one of the three attacks.

In the first experiment, we leverage the ransomware prototype to launch a GC attack by writing amounts of non-encrypted data after encryption. Nevertheless, when the storage space is running out, all of the write commands issued by the ransomware prototype process are blocked by RansomTag, since the process has already been detected as ransomware. Therefore, the retained data is intact under the GC attack.

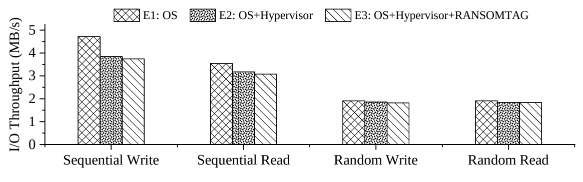
To evaluate the effectiveness of resisting timing attack by RansomTag, we utilize the second prototype to deliberately slow down encryption. Specifically, it encrypts one file every five minutes to avoid I/O pattern matching. However, the experiment result shows that the prototype process has been identified as crypto ransomware by RansomTag as long as it encrypts the first few files. Therefore, no matter how it subsequently adjusts the encryption speed, the original data it intends to overwrite is retained.

In the third experiment, the last ransomware prototype performs a TRIM attack by calling the related API after file encryption. It sends the TRIM command directly to the SSD drive to speed up the reclamation of the original data blocks. However, the TRIM command issued by the prototype process is blocked as the process has already been detected by RansomTag when it encrypts files. Thus, it indicates that our system is able to defend against the TRIM attack.

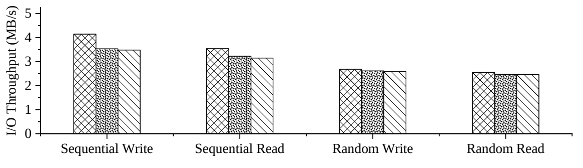
5.5 Impact on I/O Performance

To evaluate the impact of RansomTag on I/O performance, we perform micro-benchmark and macro-benchmark in three environments: (1) in a bare OS environment (Windows or Ubuntu) (*E*1); (2) in an OS running on the hypervisor with RansomTag disabled (*E*2); (3) in an OS running on the hypervisor with RansomTag enabled (*E*3).

Micro-Benchmark. We utilize DISKSPD, a storage performance tool from Microsoft [38], to test the I/O throughput (i.e., micro-benchmark) in Windows. Similarly, we leverage FIO [3], a popular tool to benchmark storage in Linux, to perform the micro-benchmark in Ubuntu. By setting the different parameters of FIO or DISKSPD, we test four categories of I/O throughput (i.e., sequential



(a) 64-bit Windows 10

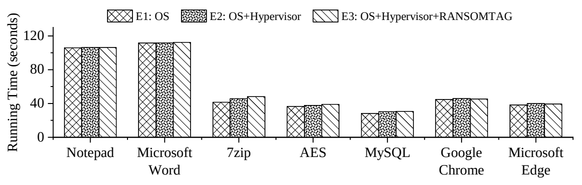


(b) 64-bit Ubuntu 22.04

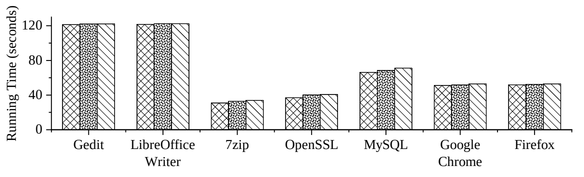
Figure 6: The I/O performance of Micro-benchmark.

write, sequential read, random write, and random read) in the above three environments. The results in Windows and Ubuntu are shown in Figure 6(a) and Figure 6(b), respectively. The three bars from left to right represent $E1$, $E2$, and $E3$, respectively. For random write and read in Windows and Ubuntu, the I/O throughput is not much different in $E1$, $E2$, and $E3$. It indicates that the I/O performance overhead on random write and read introduced by RansomTag is acceptable. However, the I/O throughput of sequential write and read in $E2$ and $E3$ is slightly lower than that in $E1$, but there is almost no difference between the throughput in $E2$ and $E3$. It indicates that the overheads on sequential write and read are mainly introduced by the hypervisor instead of RansomTag. The main reason is that the hypervisor intercepts Memory-Mapped I/Os (MMIOs) using shadow paging, which may cause performance degradation by intercepting numerous I/Os.

Macro-Benchmark. We further perform a macro-benchmark experiment to evaluate the impact of RansomTag on common real-world Windows and Linux applications, respectively. We choose a number of widely used Windows applications (including Notepad, Microsoft Word, 7zip, AES, MySQL, Google Chrome, and Microsoft Edge) and Linux applications (including Gedit, LibreOffice Writer, 7zip, OpenSSL, MySQL, Google Chrome, and Firefox), and execute the same workflows in the above three environments by using the AutoIt [55] tool in Windows and a bash script in Ubuntu, respectively. By recording the running time and CPU utilization of each application in every environment, we get the impact of RansomTag on these applications. The results are shown in Figure 7 and Figure 8. In the figures, the running time and CPU utilization of Windows and Ubuntu applications' workflows are almost flat in the three environments. It indicates that the impact of RansomTag on real-world applications is acceptable. Particularly, to precisely evaluate the time consumption of OS introspection, we add several lines of code into RansomTag to get high-resolution time before and after the introspection by reading the current value of the processor's time-stamp counter (TSC) [11]. Thus, we can get the time overhead of the introspection. The results show that the average time consumption of the introspection is 78 nanoseconds (about 4% of the overall overhead) in Windows and 57 nanoseconds (about 6% of the overall overhead) in Ubuntu, respectively.

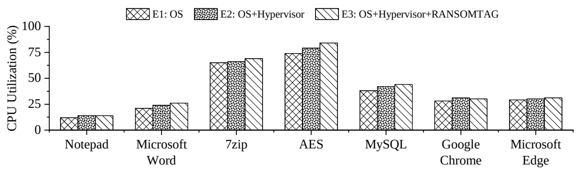


(a) 64-bit Windows 10

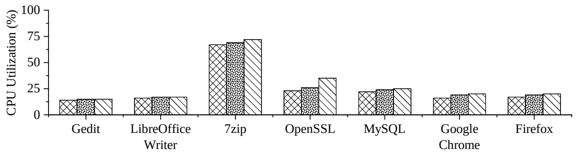


(b) 64-bit Ubuntu 22.04

Figure 7: The running time of real-world applications.



(a) 64-bit Windows 10



(b) 64-bit Ubuntu 22.04

Figure 8: The CPU utilization of real-world applications.

5.6 Impact on SSD's Lifespan

We utilize wear leveling effectiveness to evaluate the impact of RansomTag on the SSD's lifespan. The wear leveling effectiveness can be measured by Hoover economic wealth inequality indicator [7, 24, 49]. It calculates an appropriately normalized sum of the difference of each measurement to the mean [60]. The wear leveling inequality (WLI) is computed as: $WLI = \frac{1}{2}\sum_{i=1}^{n}\left\|\frac{e_i}{E} - \frac{1}{n}\right\|$. In the equation, $e_1, e_2, ..., e_n$ are erasure counts of all the n erase blocks, and $E = \sum_{i=1}^{n} e_i$.

We erase and write the blocks of the SSD $500,000$ times with RansomTag disabled and enabled, respectively. The values of WLI are respectively 3.088% and 3.104%. In other words, the value of WLI only increased by 0.016% with RansomTag enabled, which means the impact of RansomTag on SSD's lifespan is negligible.

6 DISCUSSION

First, RansomTag detects ransomware in the hypervisor and provides hardware-level protection for user data in the SSD. Thus, with SSDs widely used in cloud computing platforms, it is naturally suitable for cloud environments. In addition, it is also readily deployed on desktop PCs. Recently, the virtualization-assisted technique has been widely used in system security. For example, virtualization-based security (VBS) [42] is a security feature starting with Windows 10 and Windows Server 2016. It uses the Windows hypervisor to create a virtual secure mode. Hypervisor-Enforced Code Integrity (HVCI), which is commonly referred to as memory integrity, is an application example of using VBS to significantly strengthen code integrity policy enforcement [40]. Therefore, we believe a virtualization-based ransomware defense solution is practical and feasible on desktop PCs.

Second, we are not committed to developing a new ransomware-detecting algorithm in this paper. Instead, we leverage the state-of-the-art approaches that have proven effective to achieve a high-precision detection rate. As mentioned in § 3.3.2, the ransomware detection algorithms and the framework of our system are orthogonal. Moreover, compared to the existing SSD-based approaches, RansomTag has a more powerful computing capability. Therefore, through several APIs designed by us, all the existing and further advanced detection algorithms (e.g., machine learning [47], decoy technology [35], and others [28, 52]) can be readily deployed in our system for continually improving the detection accuracy.

Third, RansomTag is an architecture-free system since it neither depends on any SSD vendor-specific technology nor needs to modify the OS or add any modules to the OS. Instead, RansomTag leverages tagging on the standard storage protocol to solve the issue of limited semantic information in the SSD. Besides the Windows/NTFS and Linux/ext4 implementation in our prototype, we believe it is not difficult to migrate the approach to other OSes and file systems. To achieve the migration, it requires two aspects of information, i.e., the OS abstractions for precise detection and the knowledge of the target file system for fine-grained data backup and recovery.

Fourth, our current prototype leverages the OS context information obtained by the VMI technique to precisely detect ransomware. As analyzed in § 5.4.1, it is possible to get incorrect data when introspecting from a compromised OS. Fortunately, this limitation can be mitigated in two ways. On the one hand, several promising approaches have been proposed to address the limitation of the semantic gap for VMI [23], e.g., (1) paraverification, (2) building trust area by fine-grained memory protection and monitoring hardware, or (3) detecting OS inconsistencies over the lifetime. Thus, we can integrate such an approach into our system. We leave it as one future work. On the other hand, besides the context-based detection algorithms, RansomTag supports to introduce various detection algorithms due to the powerful computing capability of the host system. Therefore, we can adopt VMI-independent approaches (e.g., AI-based algorithms) for attack detection outside of the OS when they are mature.

Fifth, as for the period of data retaining, we set the period as long as possible in our current prototype depending on the storage space of the SSD. Since our ultimate goal is to completely protect user data against crypto ransomware, even if the SSD is fully occupied, RansomTag refuses to purge any retained data. Considering that the capacity of modern SSDs is several terabytes, we think it is worthwhile to trade a small portion of storage capacity for data security. In addition, we can add security level settings to RansomTag, and users are responsible for setting the period of data retaining by themselves.

Sixth, users might leverages hardware-based or software-based encryption techniques to secure important data in the storage drives. For example, self-encrypting drives (SEDs) [30] use an onboard AES encryption chip that encrypts data before it is written and decrypts data before it is read directly from the NAND media. In such hardware-based encryption drives, the encryption and decryption are transparent to the upper layers, e.g., the OS or hypervisor, and it does not interfere with the RansomTag's attack detection. Therefore, our system will not introduce extra storage overhead. As for software-based encryption programs, their encryption behaviors are generally different from crypto ransomware. For example, they do not overwrite, delete, or rename the original files by default. Thus, RansomTag is able to easily identify them as benign processes. However, if users choose the option to delete the original files after encryption, the behaviors of the encryption programs would match one pattern of crypto ransomware. In this case, the detection may cause a false positive, which introduces an extra storage overhead. To mitigate this issue, one potential solution is to improve the detection algorithm by monitoring extortion messages. If any message about ransom extorting is found after encryption, RansomTag considers it as a crypto ransomware attack.

7 RELATED WORK

With the increasingly severe impact, a number of approaches of ransomware detection and data recovery have been proposed in recent years.

To avoid user data being encrypted, some solutions try to identify attacks as early as possible by monitoring the file (and network) I/O activities in the host [27, 29, 52] or hypervisor layer [54]. Unfortunately, some user files have already been encrypted when ransomware is detected by these defense systems. Due to the lack of data backup and recovery functionalities on these systems, the victim still has to pay a ransom to access those files. To address this problem, PayBreak [31] and RWGuard [35] strive to decrypt the attacked files with captured symmetric keys by hooking specific cryptographic APIs, e.g., Microsoft's CryptoAPI library, when encryption operations are performed by ransomware. However, they cannot recover encrypted files using ransomware's custom-written cryptographic library. Other solutions (such as ShieldFS [10], Redemption [28]) protect the original file from being encrypted by file-shadowing or redirecting access requests when a process accesses it. Such solutions avoid the original files being attacked. However, in addition to the introduction of large performance overhead, they are easily disabled or bypassed by kernel-level attacks launched by ransomware with administrator privileges [21, 54].

To tackle the aforementioned limitations, a number of SSD-based approaches against crypto ransomware have been proposed [4, 5, 21, 43, 47, 50, 60]. These approaches leverage the intrinsic characteristic of SSD, i.e., out-of-place update, to back up user data efficiently. However, due to the lack of high-level context information, these SSD-based approaches can only detect ransomware by identifying encrypted data [43, 47], modeling the I/O stream patterns of ransomware [4, 5, 60], or examining the validity of SSD's physical pages [21, 50], thus they cannot achieve accurate attack detection. For example, FlashGuard [21] adopts a conservative policy that identifies attacks if a physical page is marked invalid after

Table 2: Comparison results of RansomTag with representative related approaches.

Name	Location	Self-security	Computing Power	Detection Indicators	Data Loss	Recovery Capability	File-level Recovery	Backup Security
UNVEIL [27]	OS	Weak	Strong	File Activity + Data Entropy	Yes	No	N/A	N/A
CryptoDrop [52]	OS	Weak	Strong	File Activity + Data Entropy	Yes	No	N/A	N/A
PayBreak [31]	OS	Weak	Strong	Encryption Key	Yes	Partial	Yes	N/A
RWGuard [35]	OS	Weak	Strong	Decoy Files + Encryption Key	Yes	Partial	Yes	N/A
ShieldFS [10]	OS	Weak	Strong	File Activity + Data Entropy	No	Yes	Yes	Weak
Redemption [28]	OS	Weak	Strong	File Activity + Data Entropy	No	Yes	Yes	Weak
RansomSpector [54]	Hypervisor	Strong	Strong	File/Network Activity + Data Entropy	Yes	No	N/A	N/A
FlashGuard [21]	SSD	Strong	Weak	SSD Pages' Validity	Yes	Yes	No	Weak
SSD-Insider [4]	SSD	Strong	Weak	I/O Pattern	Yes	Yes	No	Strong
SSD-Insider++ [5]	SSD	Strong	Weak	I/O Pattern	Yes	Yes	No	Strong
RansomBlocker [47]	SSD	Strong	Weak	Data Entropy	No	Yes	No	Strong
MimosaFTL [60]	SSD	Strong	Weak	I/O Pattern	No	Yes	No	Strong
AMOEBA [43]	SSD	Strong	Weak	Data Entropy	Yes	Yes	No	Strong
RSSD [50]	SSD	Strong	Weak	SSD Pages' Validity	No	Yes	No	Strong
RansomTag	**Hypervisor + SSD**	**Strong**	**Strong**	**File Activity + Data Entropy**	**No**	**Yes**	**Yes**	**Strong**

being read by the host. It keeps all such data until no ransomware attack is guaranteed. On the contrary, SSD-Insider [4] leverages a jacobinical strategy that only monitors the I/O request headers to detect whether a host is under attack or not within a small fixed time window, i.e., the first 10 seconds. Moreover, according to the experiments by AMOEBA [43], these systems [4, 21, 43] fail to fully recover all infected pages due to a certain ratio of false negatives. Even worse, some of them might break data integrity during the restoring process [4, 5], or disturb the system's regular deployment by setting up the SSD to read-only immediately after the attack is detected [60].

To better understand our system, we compare RansomTag with 14 representative related approaches, and the results are shown in Table 2. Particularly, we make the comparison from eight aspects, i.e., the location where the system works (Location), the security of the system itself (Self-security), the computing power for detection algorithms (Computing Power), the indicators to detect attacks (Detection Indicators), the possibility of data loss (Data Loss), the data recovery capability (Recovery Capability), the file-level fine-grained recovery capability (File-level Recovery), and the security of backup data (Backup Security). Overall, compared to OS-based solutions, RansomTag has higher self-security and backup security, as it locates in the hypervisor and backs up data in the SSD instead of the OS. Compared to the hypervisor-based solution, RansomTag has no data loss and achieves file-level fine-grained data recovery. While compared to SSD-based solutions, RansomTag is able to obtain much more context information (i.e., detection indicators) for precise detection with powerful computing power, and it provides file-level fine-grained data recovery.

In addition, a number of studies on ransomware from other perspectives have been proposed. Huang et al. [20] develop a set of methodologies that enable an end-to-end analysis of the ransomware ecosystem and perform a large-scale, two-year measurement study of ransomware payments, victims, and operators. Cicala et al. [8] analyze the encryption model and the encryption key generation process of ransomware samples from different families, and discuss algorithms and functions used in modern crypto ransomware. Moussaileb et al. [44] provide a systematic review of ransomware countermeasures starting from its deployment on the victim machine until the ransom payment via cryptocurrency. McIntosh et al. [34] propose a set of unified metrics to evaluate published studies on ransomware mitigation and forecast future trends of ransomware evolution and future research directions. Alqahtani et al. [1] conduct a survey devoted to exploring and analyzing the state-of-the-art in ransomware attack detection, and critically and comprehensively analyze these solutions with the focus on the methods, means, and techniques used at every phase of the detection model. Compared to these studies, RansomTag has a different goal, which aims to achieve accurate crypto ransomware detection and fine-grained data backup and recovery.

8 CONCLUSION

In this paper, we propose RansomTag, a tag-based approach against crypto ransomware with fine-grained data recovery. In particular, it decouples the ransomware detection functionality from the firmware of the SSD and integrates it into a lightweight hypervisor of Type I, which is readily deployed onto desktop personal computers. Thus, it can leverage the powerful computing capability of the host system and the rich context information, which is introspected from the OS, to achieve accurate detection of ransomware attacks and defense against potential targeted attacks on SSD characteristics. In addition, we propose a tag-based approach to bridge the semantic gap between the hypervisor and the SSD. Thus, RansomTag can precisely back up user data without additional overhead by leveraging the intrinsic characteristic of SSD, i.e., out-of-place update. Further, RansomTag can provide complete protection and fine-grained recovery of user data, which can restore any single or multiple user files to any versions based on timestamps. The experimental results show that RansomTag can effectively detect crypto ransomware and recover all the attacked files with an acceptable performance overhead.

ACKNOWLEDGMENTS

The authors would like to thank all reviewers for the insightful comments. Those comments helped to re-shape this paper. This work is partially supported by the National Natural Science Foundation of China (Key Program Grant No. 62232013, Grant No. 62002151, U21A20464, and 62372218), by the Foundation for Innovative Research Groups of the National Natural Science Foundation of China (Grant No. 62121001), by the Key R&D Program of Shaanxi Province of China (Grant No. 2019ZDLGY12-06), by the Shenzhen Science and Technology Program (Grant No. SGDX20201103095408029).

REFERENCES

[1] Abdullah Alqahtani and Frederick T Sheldon. 2022. A Survey of Crypto Ransomware Attack Detection Methodologies: An Evolving Outlook. *Sensors* 22, 5 (2022), 1837.

[2] Kurniadi Asrigo, Lionel Litty, and David Lie. 2006. Using VMM-based sensors to monitor honeypots. In *Proceedings of the 2nd international conference on Virtual execution environments*. 13–23.

[3] Jens Axboe. 2023. FIO. https://github.com/axboe/fio.

[4] SungHa Baek, Youngdon Jung, Aziz Mohaisen, Sungjin Lee, and DaeHun Nyang. 2018. SSD-insider: Internal defense of solid-state drive against ransomware with perfect data recovery. In *2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS)*. 875–884.

[5] SungHa Baek, Youngdon Jung, David Mohaisen, Sungjin Lee, and DaeHun Nyang. 2021. SSD-Assisted Ransomware Detection and Data Recovery Techniques. *IEEE Trans. Computers* 70, 10 (2021), 1762–1776.

[6] Daniel P Bovet and Marco Cesati. 2005. *Understanding the Linux Kernel: from I/O ports to process management.* " O'Reilly Media, Inc.".

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

[8] Fabrizio Cicala and Elisa Bertino. 2022. Analysis of Encryption Key Generation in Modern Crypto Ransomware. *IEEE Trans. Dependable Secur. Comput.* 19, 2 (2022), 1239–1253.

[9] CNN. 2021. Ransomware is a national security risk. https://tinyurl.com/4he7utk9.

[10] Andrea Continella, Alessandro Guagnelli, Giovanni Zingaro, Giulio De Pasquale, Alessandro Barenghi, Stefano Zanero, and Federico Maggi. 2016. ShieldFS: A Self-Healing, Ransomware-Aware Filesystem. In *Proceedings of the 32nd Annual Conference on Computer Security Applications*. 336–347.

[11] Intel Corporation. 2022. Intel® 64 and IA-32 Architectures Software Developer's Manual (2022). https://tinyurl.com/mt58w3a9.

[12] The MITRE Corporation. 2022. Access Token Manipulation. https://attack.mitre.org/techniques/T1134/.

[13] Fabio De Gaspari, Dorjan Hitaj, Giulio Pagnotta, Lorenzo De Carli, and Luigi V Mancini. 2020. Encod: Distinguishing compressed and encrypted file fragments. In *Network and System Security: 14th International Conference, NSS 2020*. 42–62.

[14] Brendan Dolan-Gavitt, Tim Leek, Michael Zhivich, Jonathon Giffin, and Wenke Lee. 2011. Virtuoso: Narrowing the semantic gap in virtual machine introspection. In *2011 IEEE symposium on security and privacy*. IEEE, 297–312.

[15] FBI. 2022. Ransomware. https://tinyurl.com/2rnmxrzn.

[16] Yangchun Fu and Zhiqiang Lin. 2013. Space Traveling across VM: Automatically Bridging the semantic gap in virtual machine introspection via online kernel data redirection. *ACM Transactions on Information and System Security* 16, 2 (2013), 586–600.

[17] Tal Garfinkel, Mendel Rosenblum, et al. 2003. A virtual machine introspection based architecture for intrusion detection. In *NDSS*, Vol. 3. 191–206.

[18] GlobalStats. 2021. Desktop Windows Version Market Share Worldwide - June 2021. https://tinyurl.com/4zrfxp9j.

[19] GoogleCode. 2011. OpenNFM. https://code.google.com/p/opennfm/.

[20] Danny Yuxing Huang, Maxwell Matthaios Aliapoulios, Vector Guo Li, Luca Invernizzi, Elie Bursztein, Kylie McRoberts, Jonathan Levin, Kirill Levchenko, Alex C Snoeren, and Damon McCoy. 2018. Tracking ransomware end-to-end. In *2018 IEEE Symposium on Security and Privacy (SP)*. 618–631.

[21] Jian Huang, Jun Xu, Xinyu Xing, Peng Liu, and Moinuddin K Qureshi. 2017. FlashGuard: Leveraging intrinsic flash properties to defend against encryption ransomware. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. 2231–2244.

[22] IBM. 2022. IBM Spectrum Scale with TRIM-supporting NVMe SSDs. https://shorturl.at/hzCHS.

[23] Bhushan Jain, Mirza Basim Baig, Dongli Zhang, Donald E Porter, and Radu Sion. 2014. Sok: Introspections on trust and the semantic gap. In *2014 IEEE symposium on security and privacy*. IEEE, 605–620.

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

[25] The kernel development community. 2021. Processes and threads. https://linux-kernel-labs.github.io/refs/heads/master/lectures/processes.html.

[26] kernel.org. 2022. Inline Data. https://tinyurl.com/ynrd68ju.

[27] 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)*. 757–772.

[28] Amin Kharraz and Engin Kirda. 2017. Redemption: Real-time protection against ransomware at end-hosts. In *Research in Attacks, Intrusions, and Defenses: 20th International Symposium, RAID 2017*. 98–119.

[29] 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 *Detection of Intrusions and Malware, and Vulnerability Assessment: 12th International Conference, DIMVA 2015*. 3–24.

[30] Kingston. 2021. What is SSD encryption and how does it work? https://www.kingston.com/en/blog/data-security/how-ssd-encryption-works.

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

[32] Kenichi Kourai and Shigeru Chiba. 2005. Hyperspector: Virtual distributed monitoring environments for secure intrusion detection. In *Proceedings of the 1st ACM/USENIX international conference on Virtual execution environments*. 197–207.

[33] Robert Love. 2013. *Linux system programming: talking directly to the kernel and C library.* " O'Reilly Media, Inc.".

[34] Timothy McIntosh, ASM Kayes, Yi-Ping Phoebe Chen, Alex Ng, and Paul Watters. 2021. Ransomware mitigation in the modern era: A comprehensive review, research challenges, and future directions. *ACM Computing Surveys (CSUR)* 54, 9 (2021), 1–36.

[35] Shagufta Mehnaz, Anand Mudgerikar, and Elisa Bertino. 2018. Rwguard: A real-time detection system against cryptographic ransomware. In *Research in Attacks, Intrusions, and Defenses: 21st International Symposium, RAID 2018*. 114–136.

[36] Trend Micro. 2017. Erebus Linux Ransomware: Impact to Servers and Countermeasures. https://tinyurl.com/3tjtcjw6.

[37] Microsoft. 2021. Access Tokens. https://tinyurl.com/5vnyhhh7.

[38] Microsoft. 2021. DISKSPD. https://github.com/microsoft/diskspd.

[39] Microsoft. 2021. Processes and Threads. https://tinyurl.com/3sa395yy.

[40] Microsoft. 2022. Enable virtualization-based protection of code integrity. https://tinyurl.com/3dx9u2r4.

[41] Microsoft. 2022. FILE_OBJECT structure. https://tinyurl.com/5356jbuy.

[42] Microsoft. 2022. Virtualization-based Security. https://tinyurl.com/4eeh6fhd.

[43] Donghyun Min, Yungwoo Ko, Ryan Walker, Junghee Lee, and Youngjae Kim. 2022. A Content-Based Ransomware Detection and Backup Solid-State Drive for Ransomware Defense. *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.* 41, 7 (2022), 2038–2051.

[44] Routa Moussaileb, Nora Cuppens, Jean-Louis Lanet, and Hélène Le Bouder. 2021. A survey on windows-based ransomware taxonomy and detection mechanisms. *ACM Computing Surveys (CSUR)* 54, 6 (2021), 1–36.

[45] NBC News. 2022. Costa Rica, 'under assault' is a troubling test case on ransomware attacks. https://tinyurl.com/5n9338ye.

[46] Olimex. 2019. LPC-H3131. https://tinyurl.com/38fwkekd.

[47] Jisung Park, Youngdon Jung, Jonghoon Won, Minji Kang, Sungjin Lee, and Jihong Kim. 2019. RansomBlocker: A low-overhead ransomware-proof SSD. In *Proceedings of the 56th Annual Design Automation Conference 2019*. 1–6.

[48] Jonas Pfoh, Christian Schneider, and Claudia Eckert. 2011. Nitro: Hardware-based system call tracing for virtual machines. In *Proceedings of the 2011 International Conference on Advances in Information and Computer Security*. 96–112.

[49] Joel Reardon, Srdjan Capkun, and David A Basin. 2012. Data node encrypted file system: Efficient secure deletion for flash memory. In *USENIX Security Symposium*. 333–348.

[50] Benjamin Reidys, Peng Liu, and Jian Huang. 2022. RSSD: defend against ransomware with hardware-isolated network-storage codesign and post-attack analysis. In *Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*. 726–739.

[51] SAMSUNG. 2019. K9K8G08U1M datasheet. https://tinyurl.com/jcm3uswa.

[52] Nolen Scaife, Henry Carter, Patrick Traynor, and Kevin R. B. Butler. 2016. CryptoLock (and Drop It): Stopping Ransomware Attacks on User Data. In *2016 IEEE 36th International Conference on Distributed Computing Systems (ICDCS)*. 303–312.

[53] Takahiro Shinagawa, Hideki Eiraku, Kouichi Tanimoto, Kazumasa Omote, Shoichi Hasegawa, Takashi Horie, Manabu Hirano, Kenichi Kourai, Yoshihiro Oyama, Eiji Kawai, et al. 2009. Bitvisor: a thin hypervisor for enforcing i/o device security. In *Proceedings of the 2009 ACM SIGPLAN/SIGOPS international conference on Virtual execution environments*. 121–130.

[54] Fei Tang, Boyang Ma, Jinku Li, Fengwei Zhang, Jipeng Su, and Jianfeng Ma. 2020. RansomSpector: An introspection-based approach to detect crypto ransomware. *Computers & Security* 97 (2020), 101997.

[55] AutoIt Team. 2018. AutoIt v3 is a freeware BASIC-like scripting language designed for automating the Windows GUI and general scripting (2018). https://www.autoitscript.com/site/autoit/.

[56] TheWashingtonPost. 2021. Ransomware is a national security threat and a big business — and it's wreaking havoc. https://tinyurl.com/357vxevm.

[57] Linus Torvalds. 2022. syscall_wrapper.h. https://github.com/torvalds/linux/blob/master/arch/x86/include/asm/syscall_wrapper.h.

[58] VirusShare. 2021. VirusShare.com - Because Sharing is Caring. https://virusshare.com/.

[59] VirusTotal. 2021. Analyze suspicious files and URLs to detect types of malware. https://www.virustotal.com.

[60] Peiying Wang, Shijie Jia, Bo Chen, Luning Xia, and Peng Liu. 2019. Mimosaftl: adding secure and practical ransomware defense strategy to flash translation layer. In *Proceedings of the Ninth ACM Conference on Data and Application Security and Privacy*. 327–338.

[61] P. Yosifovich, D.A. Solomon, and A. Ionescu. 2017. *Windows Internals, Part 1.*

[62] zaqoQLF. 2022. ransomware-python. https://github.com/zaqoQLF/ransomware-python.