Minding the Semantic Gap for Effective Storage-Based Ransomware Defense

Weidong Zhu*, Grant Hernandez*, Washington Garcia†, Dave (Jing) Tian‡, Sara Rampazzi*, and Kevin Butler*
*University of Florida, †University of Dayton Research Institute, ‡Purdue University
*{weidong.zhu, grant.hernandez, srampazzi, butler}@ufl.edu
†washington.garcia@udri.udayton.edu
‡daveti@purdue.edu

***Abstract*—Ransomware attacks have become frequent and high-profile, leading to billions of dollars worth of data and operational losses every year. Many state-of-the-art defenses created to combat ransomware assume a trusted operating system, but they are susceptible to privileged adversaries that can compromise the OS. Deploying defense schemes inside storage can solve this problem, but they suffer from low accuracy as they lack access to higher-level semantics of data, such as filesystem metadata. To address these challenges, we develop *SrFTL*, a ransomware defense framework that bridges heuristics and semantic knowledge with SSD's flash translation layer (FTL) to enhance the detection of encryption ransomware, even in the presence of privilege escalation. SrFTL enforces policy within the SSD, while our improved ransomware classification combines content and behavior-based heuristics for detection. Classification occurs within an enclave, allowing designers to customize the detection policy and leverage multiple semantic information and I/O access patterns that are visible to the host's filesystem but not only at the storage level, to identify ransomware activity.**

Our evaluation of a proof-of-concept prototype of SrFTL shows that ransomware classification reaches zero false positives and true negatives in detecting real-world ransomware from eight families, outperforming state-of-art FTL-based defensive solutions (e.g., MimosaFTL) while incurring an average performance overhead of 2.2% across different I/O-intensive workloads. With SrFTL, we aim to help designers to better protect their systems against privileged ransomware by effectively conveying semantic information between the storage and the OS filesystem.

***Index Terms*—Ransomware, detection, solid-state drive, TEE**

I. Introduction

Ransomware is a high-profile cybersecurity threat that has significantly impacted large companies, organizations, governments, and individuals in recent years. Some of the more notorious ransomware incidents include an attack on ICBC [1], which is the biggest lender in China, disrupting the trade of the US Treasury market in 2023, and an attack against a German hospital [2] in 2020 that led to a disruption of emergency care and the death of a patient. In 2023, human-operated ransomware attacks increased over the last year by 60% [3]. Rather than simply contaminating the computer system, ransomware locks victim data and systems to extort victims for financial reward. While multiple variants of ransomware exist, the most wide-ranging threat comes from encryption ransomware, which encrypts victim's data until a ransom is paid. Thus, encryption ransomware has received extensive attention from academia and industry [4], [5], [6], [7], and this paper focuses on its defense.

Previous defensive approaches have relied on analyzing ransomware workflow and features and fall into two primary categories. **Software-level** methods leverage properties of the Operating System (OS) and other software such as cryptographic libraries [8], [9] and filesystem information [4], [6], and may deploy a recovery strategy by enabling data backup [10], [11]. However, ransomware that is able to gain root privilege (i.e., *privileged*) on a machine can disable such defenses. By contrast, **firmware-level** defenses [12], [13], [14] run *inside* a storage device. Such defenses monitor the I/O requests at the storage level to differentiate malicious traffic generated by encryption ransomware. Solid-state drives (SSDs) present a particularly compelling platform for deploying firmware-level defenses, given their widespread adoption and performance compared to magnetic storage. SSDs contain an independent processor that runs its firmware, including a flash translation layer (FTL) that services I/O requests from a host system and operates on flash memory. Not only is the interface to the drive limited – such that if an attacker compromises the OS, they still cannot compromise the storage – but because of the nature of SSD operations, when data is deleted, copies on the drive continue to exist until the FTL explicitly erases them. As such, these drives are typically more resilient to ransomware attacks.

Despite such advantages, we observe that storage devices lack semantic information on incoming I/O requests, which have no inherent meaning to the drive. In other words, there is a *semantic gap* between the filesystem on the host and the storage device. This makes it difficult for firmware-level solutions to classify the I/O traffic generated by ransomware for two main reasons: *(1)* Ransomware compromises data within the OS, which contains semantic information at file-level granularity about the stored data. Firmware-level ransomware detection deployed in the storage device can only observe those ransomware operations indirectly by reasoning about data page accesses. Such information used by existing works is often obscure – e.g., access addresses [12] and offset [13], [14] – and cannot be correlated to specific file operations, causing potential misdetection. *(2)* Since firmware-level methods do not have access to semantic information to correlate ransomware data operations to actual file compromise – since are mixed with benign I/O traffic – they also present reduced capability to detect ransomware when benign applications run

in parallel, as in multicore systems such as servers, desktops, and laptop computers [15].

To address this semantic gap, one potential approach is to integrate ransomware detection into semantic-aware disk that implements filesystem knowledge within the storage device [16], [17]. However, the OS is limited to deploying the filesystem only supported by the semantic disk, limiting flexibility and usability. Moreover, existing semantic disks adopt a "grep-box" method [18] in which they only integrate partial filesystem functionalities into the storage device without encoding the full filesystem, which makes the disk less maintainable [16]. Therefore, the filesystem information provided by existing semantic disks might be insufficient for ensuring ransomware detection.

In this paper, we show how semantic information can enhance ransomware detection with our framework *SrFTL* (Semantic-reinforced Flash Translation Layer). Our solution tailored for SSDs leverages a Trusted Execution Environment (TEE) and deploys enforcement mechanisms within the storage while allowing semantic information from the OS to inform the classification of potential ransomware. Unlike simply incorporating ransomware detection into a semantic-aware disk, our approach only requires a change to the FTL without the need for additional hardware support – such as a more powerful processor and more DRAM space – of the drive itself, facilitating the deployment, adaptability, and scalability. SrFTL uses the FTL within the SSD to collect I/O requests and send them to a secure hardware enclave where they can be securely and accurately classified with additional filesystem-level information. The classification result can be sent to the SSD, which can suspend the deletion of victim data if a ransomware attack is detected. To ensure a secure data transfer between the enclave and SSD, we design a secure channel approach that bridges the SSD and enclave with assurances of confidentiality and integrity, while the authenticity of the enclave is provided by a remote attestation. Filesystem metadata is transferred through the secure channel from the SSD to the enclave and then parsed to extract semantic information for classification. To demonstrate the effectiveness of our approach we leverage Intel's Software Guard Extensions (SGX) as an exemplar of a secure enclave. Note that other solutions that provide a TEE can also be used. In our proof-of-concept implementation, we show how SrFTL can leverage I/O access patterns, data content, and filesystem metadata as potential heuristics reaching 100% accuracy in detecting 8 real-world ransomware families while staying secure against privileged attacks that can compromise the host's OS. We thus make the following contributions:

- We design SrFTL, a privilege-resistant ransomware detection framework which allows to establish ransomware classification combined with semantically-informed heuristics based on access behavior and data content.
- We devise a novel ransomware classification method on SrFTL that can parse filesystem metadata for our detection while leveraging semantic heuristics, such as data content.
- We implement and evaluate a prototype of SrFTL on an SSD emulator that includes Intel SGX hardware. Our analysis illustrates that SrFTL can achieve zero false negatives and zero false positives detection, outperforming state-of-the-art FTL-based solutions such as MimosaFTL [13], while incurring minimal performance overhead of 2.2% over a regular SSD.

II. Background

This section illustrates the basics of flash-based SSD, ransomware, and trusted execution environments.

A. Flash-based SSDs

Flash memory is a popular non-volatile storage medium for SSDs. It consists of data pages that are typically 4KB. Read and write operate at page-level granularity, while multiple pages are grouped into a flash block, the granularity at which erasures occur. However, flash memory has a limited number of program/erase (P/E) cycles. For example, multiple-level cell (MLC) flash memory can only be erased 10K times [19].

SSDs leverage a flash translation layer (FTL) to maintain flash memory and manage data operations. FTL operates independently from the host OS with the following functionalities. **Address translation** translates the logical block addresses (LBAs) to physical page addresses (PPAs) to locate the physical address of data. **Garbage collection (GC)** periodically reclaims invalid data pages to release free space. To mitigate the limited P/E cycles of SSDs, **wear-leveling** (i.e., evenly distributing the writing on blocks) and **bad block management** (i.e., identifying broken flash blocks) serve to improve the lifetime and reliability.

Once these functions are employed in the SSD, physical actions of data overwrite or erasure does not result in immediate data removal until FTL processes garbage collection to release storage space. This offers a cost-free data backup that benefits ransomware defense. Ransomware attackers are compelled to trigger GC, an action that can potentially raise suspicion [12], to physically remove victim data. Thus, flash-based SSDs raise the bar for the success of a ransomware attack.

B. Ransomware on Flash-based SSDs

Ransomware can be classified into two types: locker and encryption. Locker ransomware [20] typically makes the entire computer system inaccessible, such as preventing the OS booting by modifying the Master Boot Record (MBR) of the computer, whereas encryption ransomware compromises users' data directly by encrypting it. Given that locker leaves data accessible and users can recover them by plugging the storage device into another computer, encryption ransomware is a more severe threat, and this paper focuses on it.

We classify encryption ransomware functionality as *overwriting* data and writing data *out-of-place*. Overwriting victim data involves replacing it with ciphertext, invalidating the victim flash pages. In contrast, out-of-place ransomware writes the encrypted data into new LBAs; the original data is then deleted (e.g., using Trim command [21]). *Advanced ransomware* has further capabilities to circumvent ransomware defenses, such as manipulating data randomness. We discuss advanced ransomware in Section V.

C. Trusted Execution Environment

A trusted execution environment (TEE) runs independently from the host system, including from the operating system (OS), as a security component of the processor, and it guarantees the confidentiality and integrity of the data and code.

Intel Software Guard Extensions (SGX) is a widely deployed TEE that strengthens the security of applications [22], [23]. SGX creates memory regions in the DRAM, called enclaves, to isolate code and data from the untrusted OS. An application outside the enclave can call trusted routines within the enclave, which executes specific code and returns a result to the application. Enclave code and data are placed in the Enclave Page Cache (EPC), which is encrypted and decrypted by a dedicated Memory Encryption Engine (MEE) in the processor. *Attestation* is the process of demonstrating whether the enclave was established correctly on a secure platform. When two applications run in different enclaves, a secure communication channel can be established between them using a *local attestation* or *remote attestation*. *Local attestation* allows enclaves in the same device to verify their trustworthiness through exchanging cryptographic messages between the enclave and a local verifier. With *remote attestation*, a quote is generated and sent to a remote challenger, which can be used to establish trust in the enclave.

III. Motivation

This section discusses the limitations of existing defenses to reason the motivation of SrFTL and gives the threat model.

A. Limitations Of Software-level Solutions

Privilege-escalated ransomware. The OS possesses a large trusted computing base (TCB), meaning that any compromise of hardware or software within the TCB can potentially put the entire system at risk, exposing a large attack surface. This makes OS vulnerable to privilege escalation. For instance, in 2022, 1066 assigned CVEs were related to privilege escalation [24], indicating that such threat is persistent. Thus, ransomware can escalate its privilege to disable defenses within the OS – we call them with these abilities *privileged*.

To illustrate the capabilities of privilege ransomware, we implement a vulnerability CVE-2019-13272 [25] in an Ubuntu Desktop 18.04 with Linux kernel 4.15.0 to steal root privilege for a non-root user and explore three potential attacks to compromise software-level solutions. (1) *Replacement attack*. State-of-the-art software-level methods modify OS kernels – e.g., system services [26] and filesystem filter driver [4], [6], [27] – in the I/O path to sniff I/O requests and operations for ransomware detection. A privilege-escalated user can replace those customized OS modules with non-modified ones to circumvent the defense. (2) *Pass-through attack*. With root privilege, adversaries can directly access storage devices without interacting with I/O sniffers in the OS. For example, privileged ransomware can directly use $pread/pwrite$ system calls to read and overwrite with encrypted data on the storage device in Linux. (3) *Termination attack.* Software-level defense methods typically launch an ad-hoc program [6], [27] as an analysis engine to receive the sniffed I/Os and operations for classifying malicious traffic. Thus, a termination attack can terminate the analysis thread to avoid being detected.

B. Limitations Of Firmware-level Solutions

Limited semantic information. It has been explored in previous work on how semantic information, such as filesystem (FS) metadata and data randomness, describes data features that can facilitate ransomware classification [5], [6]. However, current firmware-level defenses [13], [15] cannot use this semantic information as heuristics because they lack access to such information at the storage level. Therefore, distinguishing malicious I/Os from benign I/O traffic is challenging as benign applications can be running simultaneously and obscure the I/O access pattern (e.g., I/O sequence) of ransomware.

Integrating the filesystem (e.g., IPFS [28]) into a TEE can potentially solve the privileged compromise. However, adversaries can deploy a pass-through attack to bypass the protected filesystem. Although some works [29], [30] employ in-storage enforcement to prevent unauthorized data modification, they do not consider semantic information for achieving accurate ransomware classification. Therefore, we design our SrFTL framework to overcome these limitations and improve detection even in the presence of OS compromise.

Challenges of semantic-aware detection in the SSD. A potential solution to leverage semantic information might be incorporating the filesystem directly into the storage. However, such architectures bring limited usability and compatibility issues because they require the OS to only use the filesystems supported by the storage device. For example, state-of-the-art semantic disks [16], [17] only implement partial filesystem functionalities (e.g., file creation) into the storage device. Such limited information is typically insufficient for ransomware detection because designed for performance purposes and not security. Finally, although integrating the entire filesystem into the SSD can apparently mitigate the compromise, it significantly makes the storage device less maintainable [16] without preventing future exploitation. Finally, such solution requires the SSD to have sufficient computing and memory resources, which is not realistic for SSDs equipped with computing-bounded processors and small DRAM, further limiting its deployment and scalability.

C. Threat Model

We assume that the OS can be entirely and arbitrarily compromised. Thus, adversaries can achieve root privileges and consequently disable any software-level ransomware defenses.

Flash-based SSDs are isolated from the OS by having independent CPUs, memories, and firmware. Moreover, SSDs have a smaller trusted computing base (TCB), and OS can only communicate with them by limited I/O interfaces [31]. Thus, we assume that privileged adversaries cannot tamper with the SSD and cannot steal secrets (e.g., keys) from it. We also assume that the firmware can be securely retrieved and updated into the SSD by a digital signature and secure boot [32], [33]. Thus, we trust the SSDs.

We assume the SSD can generate cryptographically-secure random numbers (e.g., using an accelerometer [34]). We also

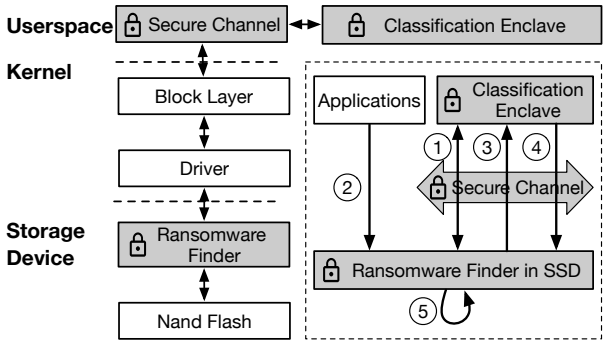

Fig. 1: The architectural overview and workflow of SrFTL. SrFTL modules are shown in the gray area.

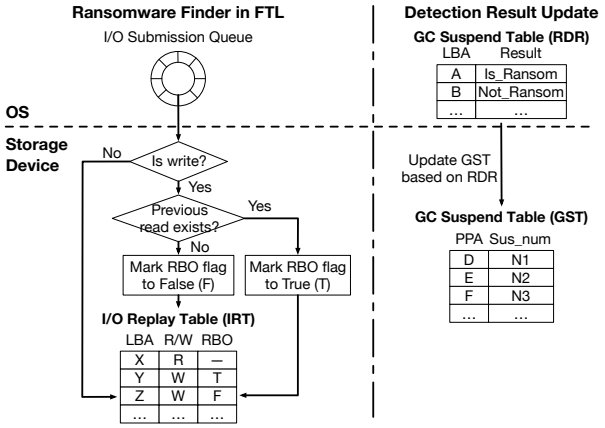

Fig. 2: The ransomware detection flow of Ransomware Finder.

consider the manufacturer of the SSD to be trusted. Our design assumes that the manufacturer can securely embed a credential into the SSD for future upper-layer user authentication [29], [33], and distribute the credential to authenticated users securely – e.g., using two-factor authentication – through existing key protection technologies such as USB security key [35]. Finally, we assume the presence of a trusted remote administrator who can securely hold and manage a credential distributed by the manufacturer, and we trust the computing platform of the remote administrator [36]. Details on this assumptions are illustrated in Section IV-C.

We assume the CPU is trusted and we do not consider physical attacks in this setting. Privileged adversaries cannot change the code and data inside enclaves, since are protected within the CPU package boundary. We do not consider side-channel attacks [37], [38] such as timing and cache collision. Mitigation of side channels in trusted hardware is orthogonal to this work as they are actively researched [39].

IV. SrFTL Design

In Figure 1, we show the overview of SrFTL in its main components. Ransomware Finder in the SSD records I/O request metadata (e.g., address and operation type). This information is transferred to a TEE, where we deploy a Classification Enclave. The Classification Enclave analyzes the received I/O requests to detect ransomware in the enclave, returning the result to Ransomware Finder, which suspends the erasure of victim data in the SSD. In addition, we design SrFTL to establish a Secure Channel to ensure the confidentiality, integrity, and authenticity of the communication between Ransomware Finder and Classification Enclave.

Figure 1 also illustrates the workflow of SrFTL. A secure channel is established (as a one-time process) before starting the detection ①. Then, SrFTL SSD starts serving normal or malicious I/O ②. Ransomware Finder collects metadata of I/Os into predefined tables, which are fetched by the Classification Enclave for ransomware judgment through the Secure Channel ③. Once the classification is concluded, the detection result is returned to the Ransomware Finder ④, and the victim data pages are suspended for garbage collection (GC) ⑤ to prevent deletion.

A. Ransomware Finder

Our Ransomware Finder operates within the SSD as a part of the FTL. All incoming I/O requests are collected by the FTL, starting with logical block addresses (LBAs), which can be used to reason the physical page address (PPA) of data by searching a logical-to-physical (L2P) address mapping table. Figure 2 shows the workflow of Ransomware Finder, which records the metadata of the incoming I/O requests in *I/O replay table* (IRT), allowing the upper classification enclave to pull and analyze the state of storage for ransomware judgment. The metadata of each entry (i.e., I/O request) in IRT consists of LBA, operation type (e.g., read/write), and an indicator flag of read-before-overwrite (RBO) operation.

LBA and operation type are essential for SrFTL to determine the location of data generated by potential ransomware. Moreover, encryption ransomware typically read the victim data before overwriting or deleting it, generating an inevitable RBO access pattern [12] to the victim data. We thus create an RBO indicator in IRT to record such an access pattern. Specifically, SrFTL reserves a "read bit" in the out-of-band (OOB) area, which is used as a metadata store [12] for each flash page to indicate whether the page has been read. For an overwrite or deletion (i.e., TRIM [21]) request, if the accessed LBA has a prior read (i.e., read bit is 1), the current request triggers a RBO access pattern, and the RBO flag of the accessed LBA in the IRT is set as $True$ (T); otherwise, the RBO flag is set as $False$ (F). Once an IRT is full, SrFTL sends it to the classification enclave over a secure channel. We set the size of the IRT to 1000 entries, which achieves the best trade-off between the table size and detection granularity. To ensure all I/O requests are mediated, if the IRT is not changed for a long time, SrFTL submits it to the enclave.

When the detection is executed the ransomware detection result (RDR) is sent to the SSD. Details on the classification enclave are illustrated in Section IV-B. In our implementation, our SrFTL classifies the potential ransomware I/O requests as *Not_Ransom*, *Low*, and *Is_Ransom*, where *Not_Ransom* means the request is benign, *Low* indicates the request has a low probability of being malicious, and *Is_Ransom* indicates compromised data. SrFTL uses the number of *Is_Ransom* requests to reason about the occurrence of ransomware, and it sends a

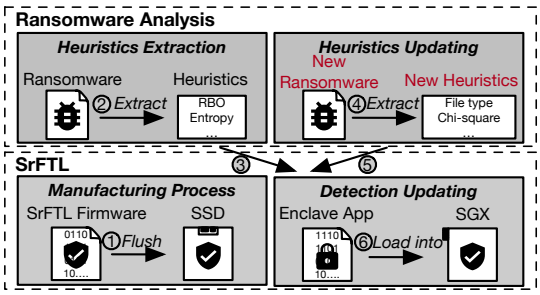

Fig. 3: The workflow of establishing and updating ransomware classification in the enclave.

warning to the user when the ratio of *Is_Ransom* requests in the IRT is over a pre-defined threshold, which is reasoned by training the classification with real-world ransomware samples and benign applications (see Section V-E). In addition, SrFTL ensures fail-safe data protection even if false negatives occur during the detection. Since RBO behavior is generated by encryption ransomware, SrFTL gives all victim data with such access pattern (i.e., overwritten or deleted) at least the *Low* threat level, which cannot be downgraded to *Not_Ransom*.

To prevent victim pages from being erased, we modify the GC mechanism such that suspicious victim pages do not receive invalid status – waived from GC – until the SSD has gone through a specific number of GC routines. SrFTL creates a GC Suspend Table (GST) to record the suspended number of GC operations for each flash page. This number increases with the threat level. Moreover, SrFTL prevents the erasure of any data collected in the IRT until the detection result has been returned from the classification enclave. Based on the result, it updates the GST. Since threat levels of data pages in an RDR can be lower than previously processed RDRs containing the same data pages, SrFTL prevents the downgrade of threat levels for existing GC-suspended flash pages.

B. Classification Enclave

Our proof-of-concept SrFTL prototype employs a hardware-backed SGX enclave as TEE to incorporate the ransomware classification. We design the classification enclave to leverage a secure channel (discussed in Section IV-C) to securely fetch data from the SSD for ransomware classification and send classification results back to the SSD.

SrFTL allows users to integrate their detection methods into the classification enclave. Moreover, a filesystem metadata parser can be devised in the classification enclave to extract file-based heuristics from storage device. Specifically, it takes the following steps in our SrFTL: (1) determine the layout of the FS to understand its metadata on the storage device, (2) read the raw FS metadata from the storage device to the enclave through a secure channel, and (3) parse the raw FS metadata to extract the target file's attributes in the enclave.

Figure 3 shows the workflow of deploying a defense on our SrFTL. The SrFTL firmware (i.e., Ransomware Finder) should be first loaded into the SSD ①. Then, developers should collect real-world ransomware and analyze them to select relevant ransomware heuristics ② based on their objectives. The heuristics are then used to generate a new classification enclave application ③, which should be loaded ⑥ into the enclave and verified through our secure channel (see Section IV-C). Developers can also update their detection directly in the enclave. New ransomware samples can be analyzed to extract new heuristics ④ for building a new enclave applications ⑤, which can be flushed to the enclave ⑥ without updating the SSD firmware. In Section V, we show how to leverage SrFTL to devise a practical example of ransomware detection.

C. Secure Channel

Our design incorporates a secure channel that ensures the *authenticity*, *integrity*, and *confidentiality* of data transfer between the SSD and enclave. Figure 4 shows the workflow of the secure channel. Since the SSD cannot directly communicate with the enclave, we design a relay application – *Data Transfer App* – in the host OS to provide minimal OS services (i.e., `pread/pwrite`) through OCALLs (i.e., calls pre-defined functions in the OS) for the enclave to establish the secure channel with the SSD. Note that our design only supports one secure channel at a time.

Authenticity. Authenticating the classification enclave and SSD is challenging because their communication must pass through the untrusted OS. To establish a trustworthy secure channel, SrFTL has been designed to leverage a remote administrator to verify the enclave (step 1) using the TEE (e.g., SGX) remote attestation. This is a common way [40], [41] to verify that the enclave runs correctly on the target machine before provisioning secrets. The remote component of SrFTL can be deployed as a trusted computer managed by users or an online service provided by the manufacturer that allows the users to manage their SSD securely remotely. As discussed in Section III-C, we assume the manufacturer is trusted and the remote administrator securely retrieves the credential. Thus, the enclave, verified by the remote administrator using remote attestation, can be used for the following authentication process. Once the enclave is verified, the remote administrator and the enclave derive a shared key to build a secure (encrypted) communication between them (step 2).

SrFTL leverages a public/private key pair to verify the authenticity of the SSD. We assume that during the manufacture of the SSD, the private key K^{-}_{SSD} can be embedded into the firmware as performed in other secure storage [42], [33]. The public key K^{+}_{SSD} instead might be distributed to the users (e.g., certificate) for developing the classification enclaves. Thus, the enclave can use the public key to verify the authenticity of the SSD because only the SSD holds the private key. Specifically, in our design the enclave and SSD first generate and exchange random numbers – R_{ENCL} (step 3) and R_{SSD} (step 4) – for the following key establishment. Then, the enclave creates a secret S (e.g., a random number) and uses it to derive a key K by hashing with R_{ENCL} and R_{SSD}, and the enclave encrypts the secret S with public keyK^{+}_{SSD} and send it to the SSD, along with its HMAC value $HMAC(K,S)$ (step 5). Thus, the SSD uses its private key K^{-}_{SSD} to decrypt secret S and derive the key K with S using the same hash method as the enclave. Once the SSD verifies

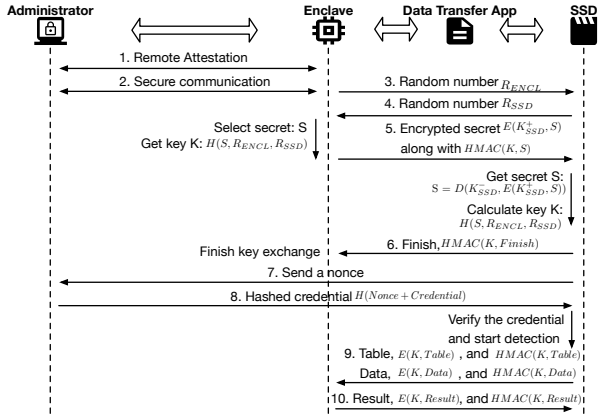

Fig. 4: The workflow of establishing a secure channel and subsequently the process of ransomware classification.

that the generated key K using the HMAC, it sends a *Finish* message $HMAC(K,Finish)$ to the enclave, ending the key exchange (step 6). Thus, an encrypted communication channel (using K) can be established between the enclave and SSD.

To prevent replay attacks, the SSD sends a randomly generated nonce to the remote administrator (step 7) through the established channels. The remote administrator then can compute a hash value $H(Nonce + Credential)$ using the credential and nonce and sends it to the SSD (step 8). Finally, the SSD can compare against the received hashed credential to verify the authenticity of the enclave.

Integrity and confidentiality. After the establishment of a secure channel, the classification enclave fetches the I/O replay tables (i.e., IRT) and data from the SSD periodically (i.e., every second) for ransomware classification (step 9), and the enclave returns the detection result (RDR) to the SSD after the classification process (step 10). SrFTL uses the shared key K to encrypt the transferred data – including IRT, data, and RDR – to provide confidentiality. Thus, adversaries cannot observe the data transferred on the secure channel to infer the classification algorithms in the enclave. Our SrFTL design further leverages a keyed HMAC to ensure the integrity of transferred data on the secure channel.

Privileged attack resistance. An adversary can resend the previous results to the SSD (replay attack) or prevent the SSD from receiving detection results (denial-of-service attack). To handle these attacks, we design SrFTL to assign a unique table ID for each suspicious table, and the table IDs are attached to the transferred suspicious tables and detection results (RDRs). Any RDR attached with previous table IDs is deemed as suspicious, thus the SSD skips their processing and notifies users of such suspicious behavior. Furthermore, the table ID is used as a nonce when encrypting the transferred data. Thus, adversaries cannot predict the data transfer pattern to infer the content of transferred tables and results. Finally, if the number of suspicious tables is increasing for a long time without any new RDRs received in the SSD, the adversary might have terminated the secure channel, and SrFTL disables block erasure in the SSD to avoid potential data loss.

V. Ransomware Classification

This section illustrates how to devise an accurate ransomware detection while bridging the semantic gap on SrFTL.

A. File-based Heuristic Extraction

We use the ext4 in our classification implementation for filesystem metadata extraction. Through the documentation of ext4 [43], we determine its metadata layout on the storage device, and it is as follows: superblock describes the enclosing FS, group descriptor contains the location of the inode table, and the inode table maintains the specific metadata of each file. Thus, the enclave fetches data in the address of the superblock from the storage through a secure channel and parses it to extract the storage address of the group descriptor and the number of inodes. Then, the enclave reads the storage with the address of the group descriptor. Based on the content of the group descriptor, the enclave can locate the inode table on the storage and then fetch the inode table from the storage to extract the final FS metadata.

Through the filesystem parser, a filesystem metadata table (FMT) is created, and each entry of it includes a key-value pair, where the index is the first LBA (LBA_{First}) of the file followed by its metadata[1], including file length ($File_{Length}$), file type, and file type change flag[2] ($Type_{Flag}$). For each reloading of the filesystem metadata in the enclave, the $Type_{Flag}$ is adjusted accordingly by comparing the current fetched file flag with the previously stored one. If the file type changes, the $Type_{Flag}$ is set to 1. Otherwise, is set to 0. Since frequent FS metadata loading can burden the I/O bandwidth of normal requests, our implementation reloads FS metadata in the enclave every second to track FS operations timely without significant performance overhead.

Then, our classification selects two file-based heuristics through the parsed metadata.

File type changes. Given that files tend to maintain their types even after the content modification, our methodology includes monitoring the modification of file types, with any alteration deemed as suspicious. Therefore, our classification tracks the file type change for each read request, which can be processed by ramsomware for victim files, in an IRT. For the LBA of each IRT read entry, if it falls within the range [LBA_{First}, LBA_{First}+$File_{Length}$] of an FMT entry, the filesystem metadata of the LBA can be located and the $Type_{Flag}$ is used to indicate the file type change.

File Deletion. The deletion operation of files can be suspicious because ransomware typically deletes files after it encrypts victim data. Moreover, the victim files need to be read before their deletion. Thus, we employ file deletion as a heuristic for ransomware detection. The process of extracting this heuristic is similar to the file type changes. Our classification searches the FMT through the LBA of the read requests in the IRT. If its metadata does not exist in the FMT, the file deletion occurs.

[1]Other metadata can be included. This is determined by the used heuristics in the detection method.

[2]The initial flag value is 0.

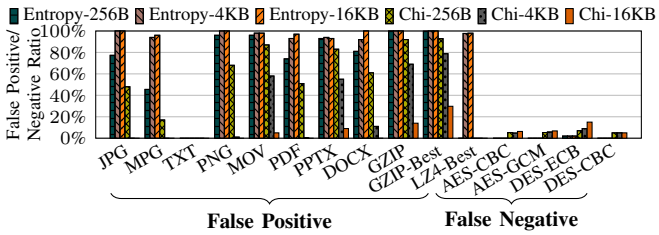

Fig. 5: **The false positive and false negative ratio when solely detecting data using entropy or chi-square testing at different data processing granularities.**

B. Incorporating Content-based Heuristics

Encryption ransomware alters data content to an encrypted format, which increases data randomness. However, benign applications, such as compression, can also generate data with relatively high randomness, leading to false positives. Thus, our classification introduces multiple randomness evaluators (i.e., entropy and chi-square testing) as content semantics to avoid such misclassification.

Entropy of writing data. Encryption induced by ransomware can increase data entropy due to the high randomness of encrypted data. Thus, our classification method monitors the entropy of writing operations. To achieve that we leverage the entropy calculation explored in previous work [6]. Based on our entropy evaluation on the ransomware dataset, we deem the trigger of entropy heuristic when the entropy exceeds 7 as encrypted data has typically a high entropy value approaching 8. Figure 5 shows this implementation can detect data with high randomness, especially those generated by encryption algorithms.

Chi-square testing for writing data. While high entropy serves as a strong indicator for data encryption, it introduces a high false positive as shown in Figure 5 when handling applications that generate high randomness data, such as compressed data. In contrast, we observe that chi-square testing incurs low false positives and false negatives when detecting data with a coarse granularity (e.g., 16KB). Thus, we introduce chi-square testing as a heuristic. In our implementation, we use 293.25 as the threshold by referring to the chi-square table with 255 degrees and 5% significance level [44]. If the chi-square value of data is smaller than 293.25, it might be generated by encryption ransomware.

Heuristic Extraction. Through the secure channel, the classification enclave sends the LBAs extracted from IRT to the SSD to fetch data for computing entropy and chi-square testing values. Since the chi-square testing (i.e., 16 KB) has a coarser granularity than a flash page (i.e., 4 KB) in our classification method, we group four data pages – each LBA represents an address of a 4 KB data page – in IRT for the calculation of chi-square testing. For every four consecutive write requests in IRT, their corresponding data can be used to calculate a chi-square testing value, which is assigned to each page in the four consecutive data pages. Since entropy has a small granularity, in our implementation each data page is divided into segments with 256 B granularity, and the entropy of each data page is represented by the highest entropy value of those segments.

TABLE I: The applications used for training.

Dataset	Description
FIO-randrw [47]	Test with randrw workload
LZ4-Default [48]	Compress with default mode
g++ [49]	RocksDB compilation
RocksDB [50] (randrw)	*readrandomwriterandom* workload of db_bench
curl [51]	Network download
IP-Ransom	In-place ransomware
OP-Ransom	Out-of-place ransomware
Bitman	Real-world ransomware
WannaCry	Real-world ransomware
Linux.Cryptor and grep [52]	Mixing real-world ransomware and data searching
Globeimposter and cp [53]	Mixing real-world ransomware and data copy

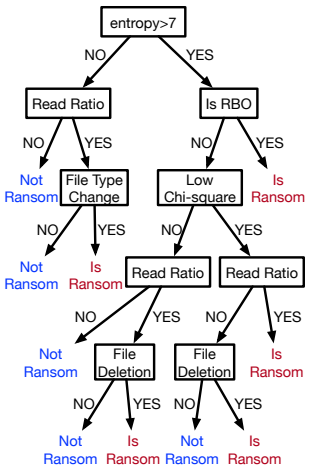

Fig. 6: The trained decision tree after training.

C. Incorporating Behavior-based Heuristics

SrFTL uses semantic information for ransomware detection but also enables monitoring the block-level I/O access pattern [6], [12], [5] that firmware-level solutions can provide. Our classification leverages the read-before-overwritten access pattern and the read ratio as detection heuristics because they are persistent ransomware behavior based on our observations of real-world ransomware.

Read-before-overwritten (RBO) access. As discussed in the previous sections, ransomware reads data from a storage drive for encryption and then removes the victim data. Our approach monitors the overwriting or deletion of previously read data to aid ransomware classification.

Read ratio. Since encryption yields an equivalent volume of encrypted data compared to plaintext, the number of read operations incurred by ransomware approximates the number of writes. Thus, our classification approach incorporates the read ratio as a heuristic. We evaluate the dynamic read ratio of workloads and ransomware listed in Table III and Table IV, and we observed that the read ratio exhibited by real-world ransomware consistently falls within the range of 25% to 75%. Thus, we deem that a read ratio located in this interval implies a higher risk of ransomware occurrence.

Heuristic extraction. The RBO heuristic can be easily extracted from the RBO flag in the IRT. In addition, our classification enclave calculates the number of read requests in the IRT and uses it to get the read percentage.

D. Advanced Attack Defense

Adversaries can develop complex attacks to circumvent existing defenses [45], [46] thus we incorporate countermeasures in SrFTL against existing advanced ransomware.

Padding attack defense. Padding 0s to the end of files is a technique used to decrease data randomness and thus avoid detection [45], [46]. We prevent this attack by calculating data entropy and chi-square testing at fine granularity (256B and 16KB, respectively) instead of at file level.

Encoding attack defense. Adversaries can encode the encrypted data to reduce the source alphabet and decrease

Algorithm 1 Ransomware Detection.

Input: *SuspiciousTable* = IRT fetched from the SSD
Data = the data of each LBA
Ransom_number = 0
for Read request i in SuspiciousTable **do**
  if The FS metadata of i not exists **then**
    Mark file deletion heuristic as YES
    break
  end if
end for
for Write request i in SuspiciousTable **do**
  if Data is encoded **then**
    Decode the Data
  end if
  Calculate 6 heuristics
  $Decision_i = DecisionTree(6\ heuristics)$
  if $Decision_i == Is\_Ransom$ **then**
    Ransom_number++
  end if
end for
if Ransom_number / table_size > Ransom_Threshold **then**
  Report the occurrence of ransomware
  All the read data of IRT are victim data and set them to *Is_Ransom*
else
  Set the results of read LBAs to *Not_Ransom*
end if

randomness [45]. Nevertheless, encoding algorithms (e.g., Base64 [54]) transfer the original data to specific characters (e.g., A-Z, a-z, 0-9, +, and / in Base64) that may facilitate the detection of encoding. Therefore, SrFTL monitors the byte value of the data to spot those consecutive encoded bytes (256B at least) and deploys the decoding method to those bytes. Then, SrFTL performs entropy and chi-square methods to evaluate the randomness of potential encoded data. Other encoding algorithms that reduce the alphabet set can be considered for our encoding identifier. For example, ASCII85 [55] maintains a pre-determined alphabet set, including 0–9, A–Z, a–z, and 23 special characters (e.g., ? and !), to encode data. Thus, our method can be adapted to other encoding algorithms by monitoring those special characters and employing a decoder. We leave extending our encoding identifier in future work.

E. Combine Them Together: Ransomware Detection

We combine the six heuristics described in Section V-B and Section V-C for robust ransomware classification using a decision tree. Specifically, we employ the ID3 algorithm [56] to train our decision tree, given its prevalence in classification tasks [15]. Each heuristic yields a binary input (i.e., YES or NO) as an indicator. We collect these heuristics of I/O requests for training when running SrFTL with different benign applications and ransomware samples, as shown in Table I. Figure 6 shows the established decision tree.

Algorithm 1 illustrates the ransomware detection process of our approach. For the fetched I/O replay table (IRT), we evaluate each I/O request by extracting the aforementioned six heuristics and feeding them into the decision tree for ransomware classification. As discussed in Section IV-A, SrFTL marks requests with RBO behavior to *Low* at least in the FTL to prevent potential data loss. Thus, victim data accessed with RBO behavior are maintained even if its returned result is *Not_Ransom*. Given that victim data must be read before being encrypted and sent back to the storage, our approach assesses write requests for ransomware decisions and suspends GCs for the data corresponding to read requests. Finally, our method records the number of requests classified as *Is_Ransom*. If the ratio of *Ransom_number* exceeds the *Ransom_Threshold*, ransomware occurrence is ascertained. The *Ransom_number* is set to 0.3 based on decision tree training. Subsequently, users are notified and can take remedial action to mitigate the ransomware threat, such as disabling the internet and terminating suspicious processes within the OS.

VI. Security Analysis

We examine all potential attack surfaces of our design and demonstrate how SrFTL could mitigate these threats.

A. Secure Channels.

Adversaries could attempt to impersonate the SSD to communicate with the enclave. However, the private key cannot be extracted from the SSD for shared key establishment as it is embedded in the firmware by the manufacturer as discussed in Section IV-C. Thus, adversaries cannot authenticate themselves to communicate with the enclave.

The communication between the remote administrator and the enclave is facilitated by remote attestation. This safeguard assures that adversaries cannot impersonate either the enclave or remote administrator, thereby preventing unauthorized access to either entity. Since the SSD in SrFTL only supports one secure channel at a time, a privileged adversary might terminate the benign enclave and build a malicious one to establish a connection (step 3 to step 6 in Figure 4) with the SSD. However, ransomware classification remains secure without being compromised. The malicious enclave must intercept the credential and resend it to the SSD to prove its authenticity. Nevertheless, it cannot steal the credential from the channel between the remote administrator and benign enclave without their provisioned keys generated during remote attestation. Moreover, adversaries cannot infer the shared key K from the benign enclave because they cannot unveil the secret S for calculating the shared key. Thus, the encrypted hashed credential sent from the benign enclave to the SSD cannot be decrypted and used by the misrepresented enclave. Additionally, since the random nonce makes the hashed credential different at each authentication cycle, replay attacks cannot compromise SrFTL. Even if adversaries terminate the classification enclave, this does not lead to data loss because SrFTL prevents the erasure of deleted data if no detection result is returned to the SSD, and it can be easily detected by a remote administrator if it has a liveness check strategy to the enclave.

B. Attack on Filesystem Metadata.

Privileged adversaries cannot tamper with the file system metadata in the enclave due to its strong isolation. However, adversaries can modify the raw metadata on the storage or the filesystem in the OS. Fortunately, existing filesystems prevent this raw metadata manipulation due to metadata caching in the memory. For example, a journaling filesystem (e.g., BFS [57]) first creates a log on storage to record all filesystem operations

TABLE II: Parameters of the SSD used for evaluation.

SSD Parameter	Value	SSD Parameter	Value
Capacity	256GB	Chips / Channel	8
Page Size	4KB	Channels	8
Pages / Block	256	Page Read	0.04ms
Blocks / Plane	4096	Page Write	0.2ms
Planes / Chip	1	Block Erase	2ms

in a sequential and atomic manner. Then, the metadata can be updated in memory, which is periodically flushed to the storage to minimize storage overhead. Thus, the filesystem metadata in the main memory can overwrite the falsified raw metadata on the storage, and the enclave can still retrieve the correct metadata. Additionally, modifying the filesystem metadata in memory is an attack pattern of ransomware, which should be considered when designing the detection method. SrFTL affords the capability to establish a detector for potential filesystem metadata tampering. For example, adversaries with elevated privileges may retain the original "magic numbers" (the first four bytes of a file) within encrypted files to hide the changing of file type. However, this can be easily detected by monitoring the data randomness (entropy and chi-square value) within the file. If the file type is not changed while data randomness is high, this can be deemed as suspicious behavior and be reported to the remote administrator. Moreover, privileged adversaries may circumvent file system writes by directly flushing data to the raw device. To counteract this attack, SrFTL can monitor the incoming data to identify those data that lack filesystem metadata, which could indicate this filesystem falsification. To conclude, SrFTL allows the detection of metadata falsification compared to traditional software-level methods that can be disabled and manipulated.

VII. Implementation

Since Linux systems serve as the backend for numerous critical infrastructures, an increasing number of ransomware attacks are now targeting Linux. In 2022, there was a 75% surge in ransomware incidents relevant to Linux systems compared to the previous year [58]. Therefore, we build SrFTL on Linux. To support a full-stack hardware and software research with SSD and SGX, we ported the FEMU SSD emulator [59] to QEMU-SGX [60], allowing us to simultaneously prototype changes at the FTL/SSD layer and the guest OS with hardware-backed SGX. FEMU [59] is a QEMU-based NVMe SSD virtual module that emulates a physical platform. We modify FEMU's FTL to perform ransomware detection and the result is processed in the SSD. We set the suspended GC cycles of *Low* and *Is_Ransom* pages to 5 and 50, respectively. Moreover, we modify the FTL to allow the enclave to use direct disk read and write at specific "magic" LBAs to fetch pending tables and data and send detection results to the SSD. We run the untrusted part of our enclave application in a standard guest OS process, which launches an enclave that communicates with FEMU-based FTL. SrFTL leverages this platform to add its ransomware-specific application code.

TABLE III: The detail of our evaluation workloads.

Trace Type	Trace ID	Trace Name	Read Ratio
Filebench	B1	webserver	90.9%
	B2	fileserver	33.3%
	B3	randomrw	31.1%
	B4	webproxy	83.3%
MSR	M1	hm_0	35.5%
	M2	prxy_0	3.1%
	M3	rsrch_0	9.3%
	M4	wdev_0	20.1%
FIU	F1	mail	8.6%
	F2	web	21.4%
	F3	homes	0.9%

TABLE IV: The characteristics of ransomware samples.

Name	Source	Purpose
IP-ransom	In-house	Training
OP-ransom	In-house	Training
Bitman	VirusTotal	Training
WannaCry	TheZoo	Training
Linux.Cryptor	TheZoo	Training
Globimposter	VirusTotal	Training
Base64	In-house	Testing
Padding	In-house	Testing
Ulise	VirusTotal	Testing
Mikey	VirusTotal	Testing
Encoder	TheZoo	Testing
Cryp	TheZoo	Testing
Crypnux	TheZoo	Testing
Vipasana	TheZoo	Testing

VIII. Evaluation

Goals. Our evaluation answers the following research questions: **(RQ0)** How SrFTL defend against potential privileged attacks? **(RQ1)** How efficient is our approach in detecting ransomware and real-world applications? **(RQ2)** How many false positives and false negatives are generated? **(RQ3)** How much performance overhead is introduced by SrFTL? **(RQ4)** How does SrFTL affect the SSD's lifetime? For **RQ0**, we evaluate how SrFTL behaves in the presence of potential rootkit attacks in Section VIII-A. We also test the detection efficiency of SrFTL while comparing SrFTL against an existing in-storage detection method [13] in Section VIII-B for **RQ1** and **RQ2**. Finally, we evaluate the performance and lifetime of SrFTL and compare it against normal SSDs in Section VIII-C to answer **RQ3** and **RQ4**.

Experimental Setup. All experiments are hosted on a machine with an Intel Xeon E3-1245 v5 3.50GHz processor with 64GB memory. We use SGXv1, supported by our X11-SSH motherboard and BIOS combination, and Intel SGX SDK [61] v2.6. The host machine has a 128MB maximum EPC. We pre-allocate 80MB of SGX memory to the guest machine (in practice, our enclave only uses 4MB). The host operating system is Ubuntu 18.04, running a KVM-SGX kernel [62] for QEMU-SGX passthrough. The guest is running an Ubuntu 18.04 with kernel 4.15.0 on a 50GB QCOW2 disk image. We attach a FEMU-backed 256GB NVMe SSD to the guest machine as shown in Table II. Finally, we allocate 4GB of memory for the guest and 4 vCPUs. Note that all experiments are run within the same guest system and SSD.

Workloads. We evaluate the performance and lifetime of SrFTL using macro-benchmarks [63] and real-world workloads as shown in Table III. Moreover, we evaluate the false positives by examining commonly used applications as shown in Table V. MSR traces [64] were collected with bursty and idle periods, while FIU traces [65] were collected from virtual server systems at FIU. Since MSR and FIU traces provide no real data, we copy a private dataset (1 GB) – including multiple types of files, such as mp4, pdf, and gzip – to the emulated SSD. Thus, the traces would be arbitrarily associated with data and files from our private dataset.

Comparisons. MimosaFTL [13] is an FTL-level ransomware defense that monitors the access pattern of I/Os to indicate

TABLE V: Real-world benign applications case study of false positive and performance degradation.

Benign Applications	Application Description	False Positive	Degraded Performance Ratio
Shred [68]	Data wipper	No	4.3%
GCC [69]	Compile linux-sgx [61]	No	1.3%
RocksDB [50]	*readwhilewriting* workload of db_bench	No	2.3%
Git [70]	Repository clone	No	5.5%
Wget [71]	Linux kernel download	No	4.2%
Gzip-Best [72]	Compress with best ratio	No	0.2%
LZ4-Best [48]	Compress with best ratio	No	0.7%
Dropbox [73]	Cloud data syncing	No	4.2%

• The performance degradation is calculated by comparing the execution time of applications when SrFTL is running or not.

ransomware traffic. We re-implement MimosaFTL[3] in an unmodified FEMU SSD, which is the same as the implementation of SrFTL, to evaluate the efficiency of the ransomware detection of our SrFTL. For the recent requests access (RRA) list in MimosaFTL, we match its length with our IST (1,000).

Real-World Ransomware. Table IV shows the ransomware samples collected from real-world sources, including VirusTotal [66] and TheZoo [67]. Additionally, we implement two typical encryption ransomware – in-place (IP-ransom) and out-of-place (OP-ransom) – for training purposes that are not reported to the public, similar to [14]. IP-ransom writes encrypted data to the original victim read address, while OP-ransom writes to a new address space before removing the original files. Moreover, we build two advanced ransomware using Base64 encoding algorithm (Base64) and padding technology (Padding) as illustrated in Section V-D.

A. Rootkit Attacks Testing

We deploy three privileged attacks as described in Section III-A to test the ability of SrFTL to resist privileged compromise. For a *replacement attack*, privileged adversaries must fabricate a malicious enclave application to replace the benign one. However, they cannot get authenticated by the remote administrator through the remote attestation of SGX, indicating the failure of such a replacement. For ransomware with *pass-through attack*, they can circumvent file-level heuristics. However, SrFTL deploys the detection enforcement at the firmware level without being bypassed while providing content-based heuristics and I/O access patterns – i.e., content-based heuristics and I/O access patterns – for the decision tree to identify ransomware behaviors. Finally, we evaluate the *termination attack* by killing the thread of the enclave application. SSD cannot receive any classification results (RDRs) after the termination. However, SrFTL can detect such attack because it monitors the received RDRs; if no new RDR arrives for a long time while the number of the suspicious table is increasing, SrFTL deems the occurrence of the attack and terminates the erasure operations in the SSD (see Section IV-C).

B. Detection Accuracy Testing

Our collected ransomware samples include both Linux and Windows variants. To support the running of Windows-based ransomware samples, we run them using Wine [74], which is a widely adopted and robust tool [75], [76] for running Windows applications in Linux. As illustrated in Section V-E, our classification method leverages a detection threshold (*Ransom_Threshold*) to indicate the occurrence of ransomware. We evaluate the effectiveness of our detection method within SrFTL framework by quantifying the ratio of false negatives and false positives when the *Ransom_Threshold* value is adjusted. Figure 7 and Figure 8 indicate that SrFTL achieves zero false negatives and positives when the threshold value falls between 0.2 and 0.5. Thus, our classification method achieves an accurate ransomware detection with zero false negatives as the threshold is 0.3 (see Section V-E).

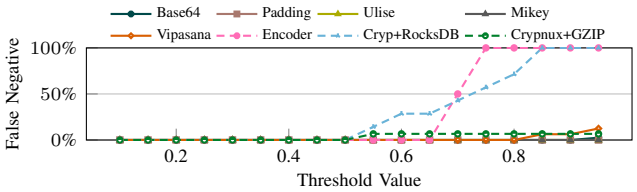

Fig. 7: The false negative of SrFTL when the ransom threshold changes.

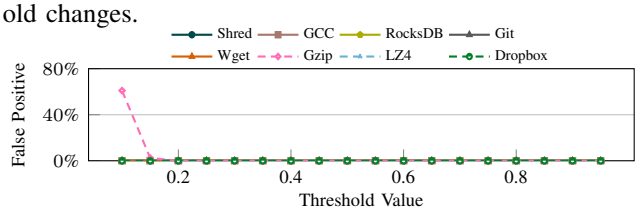

Fig. 8: The false positive of SrFTL when the ransom threshold changes.

To quantify the importance of selected heuristics, we deconstruct the composition of heuristics that were identified as positives. Figure 9 and Figure 10 show that SrFTL exhibits robust capabilities in detecting the established heuristics for the decision tree. Vipasana triggers the fewest heuristics because they only encrypt a small amount of victim data and generate non-encrypted data (e.g., HTML, txt, and executable files) to notify the success of the attack. However, it cannot circumvent our detection method because the identified heuristics can still provide sufficient information for our decision tree to make the correct ransomware classification. Benign applications rarely trigger false positives because they only activate a limited set of heuristics for our classification method that cannot incur false positives in notifying ransomware occurrence.

We further compare the detection accuracy of SrFTL against MimosaFTL to illustrate the benefit of bridging the semantic gap. Figure 11 and Figure 12 show the ratio of detected data to ransomware writes in real-world ransomware samples and benign applications. SrFTL detects 91% of writes generated by ransomware, whereas MimosaFTL only identifies 29.8% of ransomware writes on average. For benign applications, SrFTL achieves low false positives with an average of 0.7% misclassified requests; in contrast, MimosaFTL yields 8.3% false positives on average. MimosaFTL provides a worse detection accuracy than SrFTL for the following reasons. First, SrFTL can provide more heuristics, including semantic information of data content and FS metadata, with an intelligent classification (i.e., decision tree) to reason ransomware behaviors. In con-

[3]No source code was made available by the authors.

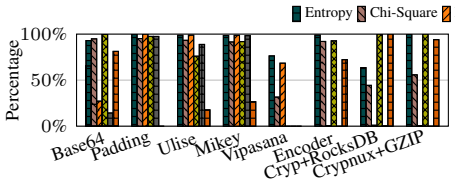

Fig. 9: The percentage of positive heuristics in ransomware testing.

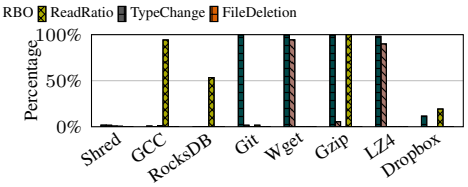

Fig. 10: The percentage of positive heuristics in benign application testing.

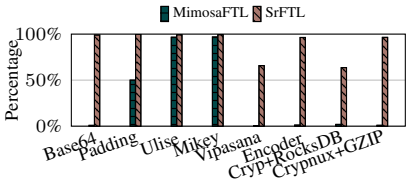

Fig. 11: The percentage of identified data pages in ransomware testing.

Fig. 12: The ratio of detected data pages in benign application testing.

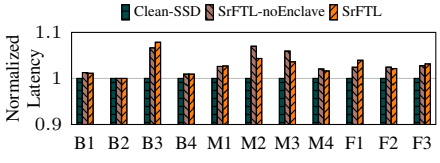

Fig. 13: The normalized average response time.

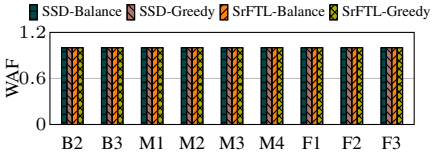

Fig. 14: The write amplification factor (WAF).

trast, MimosaFTL only relies on I/O access pattern, which is insufficient to detect ransomware and eliminate false positives. Benign applications can present similar I/O access patterns to ransomware, leading to false positives; for example, GCC reads source codes to compile executable or library files, generating a read-before-write behavior similar to ransomware. In addition, storage protocols (e.g., NVMe) can break the I/O access patterns, paralyzing FTL-based detection. Since FS always combines multiple writes together and submits them in a batch [77] to the storage device, MimosaFTL fails to detect out-of-place ransomware as it monitors the length of consecutive read and write requests, and the batched write incurs a much longer data length over read requests. Thus, SrFTL provides a more robust capability in detecting encryption ransomware over state-of-the-art FTL-based solutions.

C. Performance And Lifetime Testing

We compare the performance of an unmodified SSD (Clean-SSD), SrFTL that deploys the classification into the OS instead of the enclave (SrFTL-noEnclave), and SrFTL. In Figure 13, SrFTL-noEnclave and SrFTL increase the average latency over Clean-SSD by 1.7% and 2.2%, respectively. Although SGX can bring non-negligible overhead [78], it brings trivial effects to SrFTL because the detection module of SrFTL is not in the critical path of I/O operations. Additionally, SrFTL leads to extra reads for entropy and chi-square computing. However, modern SSDs have a high read performance that can avoid significant performance overhead [79]. Moreover, only write requests in IRT need extra reads for content-based heuristics, and data in an IRT are read synchronously by the classification enclave. Thus, the maximum bandwidth incurred by an IRT is 97.7 MB/s, which is calculated by the following equation: $IRT\_BW = 1,000 * PG\_Size/PG\_Read$. However, our implemented SSD provides 1.25 GB/s maximum write bandwidth, and SrFTL does not need to load data of read requests in an IRT. Thus, this overhead is trivial for modern SSDs that can achieve high bandwidth with gigabytes per second. Table V shows false positives and the performance degradation incurred by SrFTL when running real-world applications. Since SGX uses a dedicated chip (i.e., memory encryption engine) to achieve encryption that does not consume the resource of normal CPU cores, SrFTL introduces trivial performance overhead by 2.8% on average.

Since webserver (B1) and webproxy (B4) cannot generate enough writes for GC, we only evaluate fileserver (B2) and randomrw(B3) in Filebench for lifetime testing. We evaluate the lifetime through the write amplification factor (WAF), which is determined by the ratio of data written to flash memory to the data generated by the host. A large WAF value indicates a worse SSD lifetime. Then, we employ two GC methods in this evaluation. The "balance" method is the default GC algorithm of FEMU that selects flash blocks parallelly for erasure, and the "greedy" algorithm prioritizes the blocks with the most invalid pages. Figure 14 shows that SrFTL introduces trivial lifetime degradation over balance and greedy SSDs by 0.01% and 0.003%, respectively. We also test the ratio of data pages with RBO access pattern that creates Low pages, and it only takes 3.5% of data writes on average. Thus, SrFTL incurs trivial SSD lifetime degradation because of accurate ransomware classification and few suspended victim pages led by RBO access pattern (see Section IV-A).

IX. Discussion

This section demonstrates the limitations of the proposed approach and details that inform a practical implementation and deployment of SrFTL.

Real-time Detection. Ransomware identification needs to be fast to terminate the attack rapidly. Based on our proof-of-concept implementation, SrFTL could detect ransomware in no more than one second if ransomware runs continuously. If ransomware runs for a short time and generates a small amount of I/Os, it can be detected in less than six seconds. Since attackers might develop new variants to circumvent existing defenses, false negatives might happen. However, our approach leverages the inevitable access pattern – i.e., read-before-overwrite or read-before-delete – of encryption ransomware to suspend the GC of victim data even if the benign classification result is returned. Thus, a false negative does not lead to the risk of data loss.

Upgradable Ransomware Defense. Ransomware detection is an arms race where adversaries can learn system behavior to develop more effective ransomware variants, indicating

the necessity of more effective defenses to address new variants. However, updating firmware-level defenses requires developing new firmware and flashing it into the storage device, which can interrupt I/O services and can be challenging with the risk of data loss [80]. Thus, updating ransomware defense in the firmware could jeopardize the use of storage devices by degrading storage availability, increasing data loss probability, and introducing extra time and financial (i.e., extra storage space) expenses for data backup. SrFTL overcomes this limitation by enabling ransomware classification in the secure enclave. Designers can easily upgrade their ransomware detection by making a new enclave application and loading it into the enclave without interacting with the SSD firmware.

SGX Enclave. The EPC memory available to SGX is limited (e.g., 128MB), and we need to consider the memory consumption due to the filesystem metadata in the enclave. In the real world, the scale of the filesystem rarely exceeds such dimension because most users are unlikely to have a large quantity of valuable data that needs to be protected. For example, we extract the necessary metadata of the Linux kernel source code, which includes 52885 files, and it only occupies 2.3MB. If the available memory size of SGX is 128MB[4], over 2.9 million files' metadata could be stored. To mitigate the potential of metadata overload in the enclave, users can implement policies to selectively preserve critical files' metadata in the enclave. For example, we can preserve the metadata of files that have been most frequently accessed recently.

Full-disk Encryption. For hardware-based full-disk encryption (e.g., Samsung SSDs that use the TCG's Opal standard), SrFTL is compatible. Encryption of hardware-based SSDs is transparent to the host, and the enclave can still retrieve data from the storage for ransomware classification. For software-based disk encryption, a potential solution might exploit existing SGX-based encryption methodologies. For example, SGX-LKL [82] implements SGX-based full disk encryption in dm-crypt. Our design can be combined with the detection method in the enclave to make a full-disk encryption environment compatible with our solution.

Data Recovery. When remote administrator notices ransomware infection, typically they need to terminate the attack by turning off the host. Our design can help to retain the arrival time to the SSD of each flash memory page in the out-of-bound (OOB) area [12], [13]. To recover compromised data, the remote administrator can unplug the SSD and insert it into a secure computer. Then, an enclave application can be launched into the secure computer using the secure channel policy, allowing the remote administrator to retrieve all recently read and invalidated pages and their timestamps from the SSD. Through the read data pages, the remote administrator can pinpoint the infected files and recover them to a past state using invalidated data timestamps.

Ransomware-like Applications. Encryption applications can have access patterns similar to ransomware, leading to false positives. However, monitoring encryption in a compromised OS is challenging because privileged adversaries can pretend to behave as normal applications. Therefore, the usage of benign encryption should be strictly protected and monitored. Migrating crypto functionalities into TEEs is a realistic solution against privileged attacks that was proposed and implemented by existing works [83], [30]. Our design can incorporate training on benign encryption operations in the classification enclave to avoid false positives. Moreover, ransomware-like applications (i.e., compression and encryption) present unique characteristics over ransomware. For example, such applications have restricted access to the original files, indicating that the intactness of processed data can be used for the detection [13], [5]. Through analyzing the behavior of ransomware-like applications, a more sensitive detection can be applied to our SrFTL framework, empowering the capability of detecting these applications.

Deployment to Other Platforms. Our proof-of-concept prototype is implemented on a Linux system. However, ransomware also targets Windows machines. Windows-based systems rely on different system interfaces for raw disk read and write, block-level OCALLs, functions for interacting with the file system for metadata, and procedures for setting up Intel SGX environments. We leave the challenges of implementing SrFTL on Windows machines as future work.

Apart from SGX, SrFTL can also be deployed atop other Trusted Execution Environments (TEEs). For example, SrFTL can be ported to mobile platforms using flash-based storage (e.g., eMMC cards). Such devices typically implement ARM64 CPUs instead of Intel, and ARM TrustZone [84], which is a slight equivalent to SGX when acting as a TEE. Thus, SrFTL could be potentially implemented into its Trusted Application (TA).

Extensibility of SrFTL. SrFTL can facilitate the deployment of a more comprehensive detection with heuristics based on the metadata. For example, SrFTL enables the detection of directory traversal by monitoring the read requests to directory metadata [26], tracking access frequency by observing the number of accesses to the LBAs of files [26], and identifying suspicious renaming operations performed on files [4]. In addition, SrFTL exhibits promising potential for enhanced management of false positives and negatives compared to existing firmware-level solutions. Previous work, like MimosaFTL [13], proposes a passworded SCSI command that can be vulnerable in a compromised OS. Our approach allows the remote administrator to connect the enclave and SSD, thereby, the falsely classified data and files can be altered to be benign. We envision the future expansion of our framework to encompass more sophisticated detection mechanisms and improved victim data management.

X. Related Work

Previous ransomware defenses have primarily focused on detection [8], [27], [6], [26] and data recovery[4], [85], [86]. UNVEIL [5] proposed an artificial user environment that can efficiently detect ransomware. Similarly, ShieldFS [4] ana-

[4]SGX2 can be expanded to 1TB [81].

lyzed billions of low-level file system I/Os to understand the features of benign applications, which enabled the recognition of typical ransomware behavior. Redemption [26] designed a transparent buffer in the kernel to monitor the I/O request pattern for detecting ransomware. These defenses can effectively detect encryption-based ransomware. However, they allow the victim's data to be encrypted before detection. In contrast, data recovery allows users to restore their data without paying a ransom. PayBreak [9] supervises the encryption functions in the standard crypto libraries and holds all the encryption keys in a third party for potential data recovery. RDS3 [87] backs up data in the spare space of a computing device to restore the data encrypted by ransomware. However, an advanced attacker may get root privileges and thus disable or bypass the detection and recovery strategies in the OS.

Some new mechanisms leverage flash memory's special characteristics to defend against ransomware attacks [14], [13], [15]. FlashGuard is the first scheme that can defend against ransomware attacks, even if the attacker obtains root privilege, by retaining victim data as long as possible [12]. Similar to FlashGuard, RSSD [31] leverages NVMe over fabric to back up the data to prevent ad hoc attacks on firmware-level methods. However, they cannot detect ransomware when it presents. MimosaFTL [13] and SSD-Insider++ [15] provide fine-grained detection of ransomware-like I/O requests and data recovery at the SSD FTL. However, they only leverage the characteristics of I/O access patterns and cannot defend against advanced ransomware [45]. Moreover, they are challenging to upgrade defense as we discussed in Section III.

Tamper-resistant storage [29], [30], [33] is also used to defend against ransomware. Krahn et al. propose an SGX-based external storage device, called Pesos [29], that combines a host-side TEE with storage to protect data. Inuksuk [30] creates a secure zone on the storage device for holding sensitive user data. DiskShield [33] implements a file system in the SSD firmware, allowing secure communication between host TEE and storage. However, they need additional storage space and can only protect the data in the secure zone. Thus, the data outside the secure area is still vulnerable to ransomware. Moreover, users cannot find and terminate ransomware in time because they have no detection deployment. Therefore, tamper-resistant storage is still not the perfect answer for encryption ransomware.

XI. Conclusion

This paper presents SrFTL, a fail-secure ransomware defense platform that leverages the OS-level filesystem metadata and raw FTL I/O patterns. SrFTL operates securely in a hostile environment due to its use of Intel SGX to ensure the integrity and authenticity of its ransomware classification decisions. SrFTL achieves this while maintaining low overhead and high detection accuracy while avoiding reloading the SSD firmware for updating the detection method.

Acknowledgement

We would like to thank the anonymous reviewers and paper readers for their valuable and insightful comments and feedback. This work was partially supported by NSF CNS-2055014 and CNS-2145744.

References

[1] "Reuters." https://www.reuters.com/world/china/chinas-largest-bank-icbc-hit-by-ransomware-software-ft-2023-11-09/, 2023.

[2] "A patient has died after ransomware hackers hit a german hospital.." https://www.technologyreview.com/2020/09/18/1008582/a-patient-has-died-after-ransomware-hackers-hit-a-german-hospital/, 2020.

[3] "10 essential insights from the Microsoft Digital Defense Report 2023." https://www.microsoft.com/en-us/security/business/security-insider/reports/microsoft-digital-defense-reports/10-essential-insights-from-the-microsoft-digital-defense-report-2023/, 2023.

[4] A. Continella, A. Guagneli, G. Zingaro, G. D. Pasquale, A. Barenghi, S. Zanero, and F. Maggi., "ShieldFS: A Self-healing, Ransomware-aware Filesystem," in *32th Annual Computer Security Applications Conference (ACSAC)*, 2016.

[5] A. Kharraz, S. Arshad, C. Mulliner, W. Robertson, and E. Kirda., "UNVEIL: A Large-Scale, Automated Approach to Detecting Ransomware," in *25th USENIX Security Symposium (USENIX Security)*, 2016.

[6] N. Scaife, H. Carter, P. Traynor, and K. R. Butler, "CryptoLock (and Drop It): Stopping Ransomware Attacks on User Data," in *IEEE 36th International Conference on Distributed Computing Systems (ICDCS)*, 2016.

[7] C. Zhou, L. Guo, Y. Hou, Z. Ma, Q. Zhang, M. Wang, Z. Liu, and Y. Jiang, "Limits of i/o based ransomware detection: An imitation based attack," in *2023 IEEE Symposium on Security and Privacy (SP)*, 2023.

[8] D. Xu, J. Ming, and D. Wu., "Cryptographic Function Detection in Obfuscated Binaries via Bit-precise Symbolic Loop Mapping," in *38th IEEE Symposium on Security and Privacy (S&P)*, 2017.

[9] E. Kolodenker, W. Koch, G. Stringhini, and M. Egele., "PayBreak: Defense Against Cryptographic Ransomware," in *15th ACM Asia Conference on Computer and Communications Security (ASIACCS)*, 2017.

[10] V. Prabhakaran, A. C. Arpaci-Dusseau, and R. H. Arpaci-Dusseau., "Analysis and Evolution of Journaling File Systems," in *2005 USENIX Annual Technical Conference (ATC)*, 2005.

[11] M. Virable, S. Savage, and G. M. Voelker., "BlueSky: A Cloud-Backed File System for the Enterprise," in *10th USENIX conference on File and Storage Technologies (FAST)*, 2012.

[12] J. Huang, J. Xu, X. Xing, P. Liu, and M. K. Qureshi., "FlashGuard: Leveraging Intrinsic Flash Properties to Defend Against Encryption Ransomware," in *24th ACM Conference on Computer and Communications Security (CCS)*, 2017.

[13] P. Wang, S. Jia, B. Chen, L. Xia, and P. Liu., "MimosaFTL: Adding Secure and Practical Ransomware Defense Strategy to Flash Translation Layer," in *10th ACM Conference on Data and Application Security and Privacy (CODASPY)*, 2019.

[14] S. Baek, Y. Jung, A. Mohaisen, S. Lee, and D. Nyang., "SSD-Insider: Internal Defense of Solid-State Drive against Ransomware with Perfect Data Recovery," in *38th International Conference on Distributed Computing Systems (ICDCS)*, 2018.

[15] S. Baek, Y. Jung, D. Mohaisen, S. Lee, and D. Nyang, "Ssd-assisted ransomware detection and data recovery techniques," *IEEE Transactions on Computers (TOC)*, 2021.

[16] M. Sivathanu, V. Prabhakaran, F. I. Popovici, T. E. Denehy, A. C. Arpaci-Dusseau, and R. H. Arpaci-Dusseau, "Semantically-Smart disk systems," in *2nd USENIX Conference on File and Storage Technologies (FAST)*, 2003.

[17] J. M. Terry, N. A. Clarkson, and G. S. Barrall, "Filesystem-aware block storage system, apparatus, and method," 2011.

[18] A. C. Arpaci-Dusseau and R. H. Arpaci-Dusseau, "Information and control in gray-box systems," in *Proceedings of the Eighteenth ACM Symposium on Operating Systems Principles (SOSP)*, 2001.

[19] J. Kim, J. Lee, J. Choi, D. Lee, and S. H. Noh, "Improving SSD reliability with RAID via Elastic Striping and Anywhere Parity," in *43rd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, 2013.

[20] A. Neville, "Trojan.Winlock." https://www.symantec.com/security-center/writeup/2012-110617-3328-99, Nov. 2012.

[21] "SSD TRIM." https://searchstorage.techtarget.com/definition/TRIM, 2019.

[22] M. Bailleu, J. Thalheim, and P. Bhatotia., "Speicher: Securing LSM-based Key-Value Stores using Shielded Execution," in *17th USENIX Conference on File and Storage Technologies (FAST)*, 2019.

[23] A. Baumann, M. Peinado, and G. Hunt., "Shielding Applications from an Untrusted Cloud with Haven," in *11th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2014.

[24] "CVE - CVE." https://cve.mitre.org/index.html, 2023.

[25] "Base64 Encoding Algorithm." https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2019-13272.

[26] A. Kharraz and E. Kirda., "Redemption: Real-Time Protection Against Ransomware at End-Hosts," in *International Symposium on Research in Attacks, Intrusions, and Defenses, (RAID)*, 2017.

[27] S. Mehnaz, A. Mudgerikar, and E. Bertino., "RWGuard: A Real-Time Detection System Against Cryptographic Ransomware," in *22nd International Symposium on Research in Attacks, Intrusions, and Defenses (RAID)*, 2017.

[28] "Understanding SGX Protected File System." https://www.tatetian.io/2017/01/15/understanding-sgx-protected-file-system/, 2017.

[29] R. Krahn, B. Trach, A. Vahldiek-Oberwagner, T. Knauth, P. Bhatotia, and C. Fetzer, "Pesos: Policy enhanced secure object store," in *Proceedings of the Thirteenth EuroSys Conference (EuroSys)*, 2018.

[30] L. Zhao and M. Mannan., "TEE-aided Write Protection Against Privileged Data Tampering," in *The Network and Distributed System Security Symposium (NDSS)*, 2019.

[31] B. Reidys, P. Liu, and J. Huang, "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 (ASPLOS)*, 2022.

[32] "Ssd security firmwares features tech brief.." https://www.datasheetarchive.com/whats_new/1c1a884377ab1954f2efc54b614636ec.html.

[33] J. Ahn, J. Lee, Y. Ko, D. Min, J. Park, S. Park, and Y. Kim., "DiskShield: A Data Tamper-Resistant Storage for Intel SGX," in *18th ACM Asia Conference on Computer and Communications Security (ASIACCS)*, 2020.

[34] J. Voris, N. Saxena, and T. Halevi, "Accelerometers and randomness: Perfect together," in *Proceedings of the Fourth ACM Conference on Wireless Network Security*, 2011.

[35] "What is a USB security key, and how do you use it?." https://www.tomsguide.com/news/usb-security-key.

[36] A. Dhar, I. Puddu, K. Kostianen, and S. Capkun, "Proximitee: Hardened sgx attestation and trusted path through proximity verification," 2018.

[37] S. van Schaik, A. Kwong, D. Genkin, and Y. Yarom, "SGAxe: How SGX fails in practice." https://sgaxeattack.com/, 2020.

[38] P. Kocher, J. Horn, A. Fogh, D. Genkin, D. Gruss, W. Haas, M. Hamburg, M. Lipp, S. Mangard, T. Prescher, M. Schwarz, and Y. Yarom., "Spectre Attacks: Exploiting Speculative Execution," in *40th IEEE Symposium on Security and Privacy (S&P)*, 2019.

[39] T. Yavuz, F. Fowze, G. Hernandez, K. Y. Bai, K. R. B. Butler, and D. J. Tian, "Encider: Detecting timing and cache side channels in sgx enclaves and cryptographic apis," *IEEE Transactions on Dependable and Secure Computing*, 2023.

[40] V. Scarlata, S. Johnson, J. Beaney, and P. Żmijewski, "Supporting third party attestation for intel® sgx with intel® data center attestation primitives," 2018.

[41] "Attestation Services for Intel Software Guard Extensions." https://www.intel.com/content/www/us/en/developer/tools/software-guard-extensions/attestation-services.html.

[42] C. Meijer and B. van Gastel, "Self-encrypting deception: Weaknesses in the encryption of solid state drives," in *2019 IEEE Symposium on Security and Privacy (SP)*, 2019.

[43] "Ext4 Disk Layout." https://ext4.wiki.kernel.org/index.php/Ext4_Disk_Layout#Directory_Entries, 2019.

[44] J. Pont, B. Arief, and J. Hernandez-Castro, "Why current statistical approaches to ransomware detection fail," in *International Conference on Information Security (ISC)*, 2020.

[45] J. Han, Z. Lin, and D. E. Porter, "On the effectiveness of behavior-based ransomware detection," in *16th EAI International Conference on Security and Privacy in Communication Networks (SecureComm)*, 2020.

[46] X. Ugarte-Pedrero, I. Santos, B. Sanz, C. Laorden, and P. Bringas, "Countering entropy measure attacks on packed software detection," in *9th IEEE Consumer Communications and Networking Conference (CCNC)*, 2012.

[47] "Flexible I/O Tester." https://github.com/axboe/fio, 2023.

[48] "LZ4." http://lz4.github.io/lz4/.

[49] "The GNU G++ Compiler." https://faculty.cs.niu.edu/~hutchins/csci241/compiler.htm, 2023.

[50] "RocksDB: A Persistent Key-Value Store for Flash and RAM Storage." https://www.kernel.org/.

[51] "curl - Linux manual page." https://man7.org/linux/man-pages/man1/curl.1.html, 2023.

[52] "grep - Linux manual page." https://man7.org/linux/man-pages/man1/grep.1.html, 2023.

[53] "cp - Linux manual page." https://man7.org/linux/man-pages/man1/cp.1.html, 2023.

[54] "Base64 Encoding Algorithm." https://medium.com/swlh/base64-encoding-algorithm-42abb929087d.

[55] "ascii85(1) - Linux man page." https://linux.die.net/man/1/ascii85, 2023.

[56] "Decision Trees: ID3 Algorithm Explained." https://towardsdatascience.com/decision-trees-for-classification-id3-algorithm-explained-89df76e72df1, 2023.

[57] "BFS Filesystem for Linux." https://docs.kernel.org/filesystems/bfs.html.

[58] "Linux Ransomware Poses Significant Threat to Critical Infrastructure." https://www.darkreading.com/vulnerabilities-threats/linux-ransomware-poses-significant-threat-to-critical-infrastructure, 2023.

[59] H. Li, M. Hao, M. H. Tong, S. Sundararaman, M. Bjørling, and H. S. Gunawi., "The CASE of FEMU: Cheap, Accurate, Scalable and Extensible Flash Emulator," in *16th USENIX Conference on File and Storage Technologies (FAST)*, 2018.

[60] J. M., "Virtualizing Intel Software Guard Extensions with KVM and QEMU." https://software.intel.com/en-us/articles/virtualizing-intel-software-guard-extensions-with-kvm-and-qemu, May 2019.

[61] Intel, "Intel Software Guard Extensions for Linux* OS." https://github.com/intel/linux-sgx, 2019.

[62] Intel, "KVM-SGX." https://github.com/intel/kvm-sgx, 2019.

[63] "Filebench - A Model Based File System Workload Generator." https://github.com/filebench/filebench, 2019.

[64] D. Narayanan, A. Donnelly, and A. Rowstron., "Write Off-Loading: Practical Power Management for Enterprise Storage," in *6th USENIX Conference on File and Storage Technologies (FAST)*, 2008.

[65] R. Koller and R. Rangaswami, "I/o deduplication: Utilizing content similarity to improve i/o performance," *ACM Transactions on Storage*, 2010.

[66] "VirusTotal." https://www.virustotal.com/gui/home/upload, 2019.

[67] "TheZoo - A Live Malware Repository." https://github.com/ytisf/theZoo, 2019.

[68] "How to Securely Erase a Disk and File using the Linux shred Command." https://www.freecodecamp.org/news/securely-erasing-a-disk-and-file-using-linux-command-shred/, 2023.

[69] "Open Whisper Systems.." https://signal.org/blog/private-contact-discovery/.

[70] "Git." https://git-scm.com/, 2023.

[71] "GNU Wget." https://www.gnu.org/software/wget/, 2023.

[72] "GZIP." https://www.gzip.org/.

[73] "Dropbox." https://www.dropbox.com/, 2023.

[74] "Wine." https://www.winehq.org/, 2019.

[75] "The year of WINE on Linux." https://www.datamation.com/open-source/the-year-of-wine-on-linux/, 2023.

[76] "Desktop Linux Market survey." https://archive.ph/20120524145331/http://www.desktoplinux.com/cgi-bin/survey/survey.cgi?view=archive&id=0813200712407#selection-247.5-247.32/, 2023.

[77] G. Haas and V. Leis, "What modern nvme storage can do, and how to exploit it: High-performance i/o for high-performance storage engines," *Proceedings of the VLDB Endowment*, 2023.

[78] M. Taassori, A. Shafiee, and R. Balasubramonian., "VAULT: Reducing Paging Overheads in SGX with Efficient Integrity Verification Structures," in *23th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2018.

[79] S. Wu, J. Zhou, W. Zhu, H. Jiang, Z. Huang, Z. Shen, and B. Mao, "Ead: a collision-free and high performance deduplication scheme for flash storage systems," in *2020 IEEE 38th International Conference on Computer Design (ICCD)*, 2020.

[80] "How To Upgrade SSD Firmware." https://www.storagereview.com/how-to-upgrade-ssd-firmware.

[81] "How will Intel's "Ice Lake" redefine the scope of data security?." https://fortanix.com/blog/2021/04/how-will-intels-ice-lake-redefine-the-scope-of-data-security/, 2019.

[82] C. Priebe, D. Muthukumaran, J. Lind, H. Zhu, S. Cui, V. A. Sartakov, and P. R. Pietzuch, "Sgx-lkl: Securing the host os interface for trusted execution," *ArXiv*, 2019.
[83] "TresorSGX." https://github.com/ayeks/TresorSGX, 2016.
[84] Arm, "Arm TrustZone Technology." https://developer.arm.com/ip-products/security-ip/trustzone, 2019.
[85] J. Yun, J. Hur, Y. Shin, and D. Koo., "CLDSafe: An Efficient File Backup System in Cloud Storage against Ransomware," in *IEICE Transactions on Information and Systems*, 2017.
[86] J. Strunk, G. Goodson, M. Scheinholtz, C. Soules, and G. Ganger, "Self-securing storage: protecting data in compromised systems," in *Foundations of Intrusion Tolerant Systems, 2003 [Organically Assured and Survivable Information Systems]*, 2003.
[87] K. P. Subedi, D. R. Budhathoki, B. Chen, and D. Dasgupta., "RDS3: Ransomware Defense Strategy by Using Stealthily Spare Space," in *IEEE Symposium Series on Computational Intelligence (SSCI)*, 2017.