Ransom Access Memories: Achieving Practical Ransomware Protection in Cloud with DeftPunk

Zhongyu Wang, Yaheng Song, Erci Xu, Haonan Wu, Guangxun Tong, Shizhuo Sun, Haoran Li, Jincheng Liu, Lijun Ding, Rong Liu, Jiaji Zhu, and Jiesheng Wu, *Alibaba Group*

https://www.usenix.org/conference/osdi24/presentation/wang-zhongyu

This paper is included in the Proceedings of the 18th USENIX Symposium on Operating Systems Design and Implementation.

July 10–12, 2024 • Santa Clara, CA, USA

978-1-939133-40-3

Open access to the Proceedings of the 18th USENIX Symposium on Operating Systems Design and Implementation is sponsored by

Ransom Access Memories: Achieving Practical Ransomware Protection in Cloud with DeftPunk

Zhongyu Wang*, Yaheng Song*, Erci Xu†, Haonan Wu, Guangxun Tong, Shizhuo Sun, Haoran Li, Jincheng Liu, Lijun Ding, Rong Liu, Jiaji Zhu, and Jiesheng Wu
Alibaba Group

Abstract

In this paper, we focus on building a ransomware detection and recovery system for cloud block stores. We start by discussing the possibility of directly using existing methods or porting one to our scenario with modifications. These attempts, though failed, led us to identify the unique IO characteristics of ransomware, and further drove us to build DeftPunk, a block-level ransomware detection and recovery system. DeftPunk uses a two-layer classifier for fast and accurate detection, creates pre-/post-attack snapshots to avoid data loss, and leverages log-structured support for low overhead recovery. Our large-scale benchmark shows that DeftPunk can achieve nearly 100% recall across 13 types of ransomware and low runtime overhead.

1 Introduction

Ransomware has become increasingly prevalent in recent years, posing a major threat to data security. Recent reports show that ransomware attacks have caused billions of dollars in losses to individuals and organizations [27, 29]. Typically, attackers exploit system vulnerabilities to access and encrypt users' data for ransom.

Cloud services are also facing increasingly aggressive ransomware threats. In ALIBABA cloud, our cloud block store (a.k.a., Elastic Block Storage, or EBS) has been under constant attacks. In Q3 of 2022 alone, we have received nearly one thousand reports from our EBS Virtual Disk (VD) users, yielding a 118% increase over the entire year of 2021.

To combat ransomware, practitioners have proposed various detection and recovery methods. At the application layer, users can use antivirus software and firewalls [21, 24, 31] to identify suspicious behaviors by monitoring file access and checking malware signatures. At the OS layer, recent works have demonstrated their ability to detect ransomware activities via behavioral analysis [33, 41, 45, 57]. Third, at the hardware layer, prior works have shown that, with hardware-assistance from customizable devices, the log-structured design of SSDs can be leveraged to detect and roll back ransomware activities [34, 35, 43].

Unfortunately, these traditional methods cannot be directly applied for our EBS. First, the application/OS layer methods require strong user cooperation. As a cloud vendor, we are unable to enforce our users to use specific software and/or OS. In addition, field statistics suggest that tenants may not always keep the software up to date, leaving potential vulnerabilities. Second, cloud providers usually use commodity hardware for cost efficiency and portability. Thus, it is impractical for us to build protection that only works on specialized devices. Third, certain vendors, including us, have already provided (periodical) VD snapshot for data recovery. Simply relying on such mechanisms may not be favorable for users, as they may lose the data between two snapshots due the hour-level interval and high CapEx led by the extra storage space

Existing solutions, while failing to work directly, inspire us to use the IO characteristics of ransomware (e.g., excessive Write-After-Read access) for detection, and leverage the garbage collection (GC) of log-structured design for recovery. This is because the block store service, by default, can monitor and analyze the block-level IO traffic. Moreover, our EBS is built on an append-only distributed file system (DFS), and thus supports similar mechanism to roll back data that have not been reclaimed by GC (such as SSD). The benefits of such design are two-fold. First, it does not rely on certain software or hardware support. Second, it can be easily deployed with little extra overhead imposed.

Therefore, we started exploring the possibility of building this solution. However, real world data show that such a preliminary attempt fails to deliver satisfactory accuracy in detection. We assume the main reason is that the existing detection algorithms—bounded by their prerequisites (e.g., need to use SSD's weak SoC) and lack of real-world access/analysis—can not effectively and efficiently distinguish ransomware traffic from normal IOs. Additionally, we find that the existing recovery methods can not always guarantee lossless data recovery due to the limited time of multi-version support.

In this paper, we propose DeftPunk, a block-level ransomware detection and recovery system for the cloud. Based on extensive comparisons between ransomware and normal workloads, DeftPunk constructs a rich set of features and uses a two-layer classifier for fast and accurate ransomware detection. To avoid data loss, DeftPunk creates pre- and post-attack snapshots to "lock in" the effects of the attack, and persists all modifications in between. For recovery, DeftPunk follows a "undo-redo" strategy to roll back the data to the pre-attack state and only redo the users' writes.

*Equal contributions.
†Corresponding author.

Specifically, we first assemble a large-scale ransomware dataset by collecting more than 140 hours of block-level IO traces from 13 mainstream ransomware (e.g., Wannacry [30] and Mallox [18]) and 16 types of real-world workloads from our EBS. The comprehensive comparison shows that, apart from the well-known Write-After-Read (WAR) pattern, ransomware also exhibits other characteristics, such as read to write ratio, access offset distribution and frequent access on the system disk. These observations help us to extend the feature set of `DeftPunk` by including IO statistics, dependency, working set size, offset and certain LBA spatial access.

The above extended features help `DeftPunk` to achieve higher accuracy. But, for deployment efficiency under the sheer volume of VDs and their IOs, we also need to consider the runtime overhead. Therefore, `DeftPunk` adopts a two-layer model. The first layer, focusing on eliminating false positives (i.e., filtering normal IOs), uses a straightforward decision tree algorithm with simple features (only requiring O(1) computation). The positive cases classified by the first layer will be further sent to the second layer for a double check. The second layer emphasizes on exposing all ransomware cases, and thus uses a more sophisticated algorithm (i.e., XGBoost [38]) with the complete set of features.

Based on the output of the two-layer model, `DeftPunk` can create a pair of snapshots right before and after the attack. Moreover, with multi-version support by the EBS log-structured design and GC pausing, `DeftPunk` can persist all data modifications during the attack. For recovery, `DeftPunk` follows a "undo-redo" strategy to roll back the data to the pre-attack state, and, based on a rule-based model, only redo the writes made by the users.

Based on our assembled dataset, we benchmark `DeftPunk` with a set of state-of-the-art ransomware detection methods. The results show that `DeftPunk` can always achieve nearly 100% recall with 95.8% precision, outperforming all other peers. Moreover, `DeftPunk` only uses 7 vCPU cores for processing 1 million IOPS for detection and can recover valid data at 4.62 GB per second. We have deployed `DeftPunk` in our EBS service for a few invited users, and it has successfully prevented 2 attacks with data fully recovered.

The contributions of this paper are summarized as follows:

- We assemble and release a large-scale ransomware benchmark with real-world traces[1].
- We build `DeftPunk`, a practical ransomware detection and recovery system for the cloud EBS.
- We extensively evaluate `DeftPunk` and the results show that `DeftPunk` outperforms peers by up to 95.77% in precision and nearly 100% in recall.

The rest of the paper is organized as follows. §2 gives a brief overview of our EBS and background of ransomware. §3 discusses the existing solutions and related work. §4 identifies the goals and challenges. §5 presents the design of `DeftPunk` and §6 shows the evaluation of `DeftPunk` in depth. We end this paper with a discussion on `DeftPunk`'s limitations in §7 and a short conclusion in §8.

[1]The dataset is at: https://tianchi.aliyun.com/dataset/177511?lang=en-us

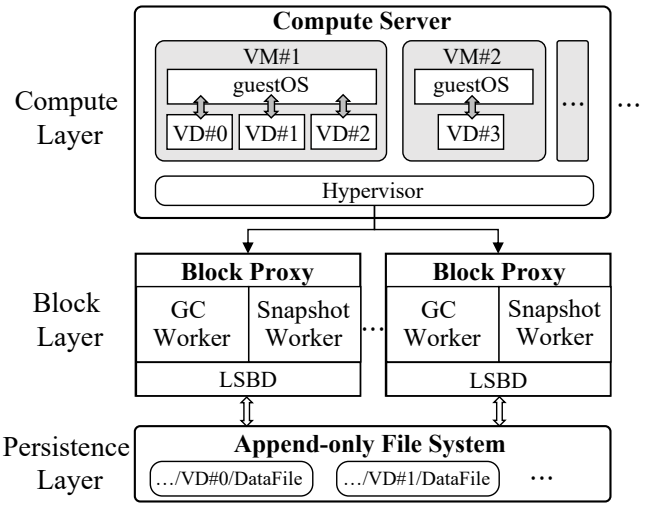


Figure 1: Overview of EBS Architecture. **VD**: *Virtual Disk*; **LSBD**: *Log-Structured Block Device*. **GC**: *Garbage Collection*.

2 Background

2.1 EBS in ALIBABA Cloud

Elastic Block Storage (EBS) serves as a cornerstone in today's cloud. To provide virtual block devices to users with high flexibility and availability, our EBS, similar to other vendors' architecture [9–11, 17], follows a "compute-to-store" philosophy. With this setup, the storage clusters are physically disaggregated—interconnected by data center network—from the compute servers (and subsequently the virtual machines running on them).

Figure 1 illustrates an overview of the EBS architecture. On the compute end, each server can host multiple Virtual Disks (VDs) and also embeds a client within the hypervisor to forward VD's requests to backend storage clusters. Once the *Block Proxies* receive the VDs' requests, they will then persist/fetch the data to/from the corresponding file in the distributed file system (DFS).

Like Google [49], Azure [37], and Alibaba Cloud [48], we too adopt a log-structured distributed file system (DFS) as the storage backend. The Block Proxy employs a Log-Structured Block Device (LSBD) to transform VDs' IO to an appending-only DataFile provided by the underlying DFS. One key functionality is to register the mapping (i.e., from VD LBA to location in DataFile) in a per-VD IndexMap to track latest location of data in the DataFile. Also, this requires a garbage collection (GC) for reclaiming space from stale data. In addition, we also set up a Snapshot Worker to allow users to create snapshots of their VDs and store them as binaries.

Layer	Mechanism	Schemes
App.	Anti-virus real-time scan	Fortinet [24], Kaspersky [21], Windows defender [31]
OS	File system behavioral analysis	CM&CB [33], UNVEIL [45], WaybackVisor [41], Towards. [57]
HyperV.	Scheduled snapshot	AWS [3], HUAWEI [4]
HW.	Detection with real-time rollback	FlashGuard [43], SSD-insider [34], SSD-insider++ [35], RSSD [53]

Table 1: Common protections against ransomware. **App.**: *Application*; **HyperV.**: *Hypervisor*; **HW.**: *Hardware*.

2.2 Ransomware

Ransomware, such as WannaCry [30], has been rampantly spreading across the globe, leading to billions of dollars financial losses. The typical procedure of a ransomware attack has three steps: (1) the attacker infects the victim's machine via weak password and/or system vulnerabilities; (2) the ransomware encrypts the victim's data; (3) the victim is instructed to pay a certain amount of ransom (e.g., cryptocurrency) for decryption.

Both industry and academia have been working on ransomware detection and protection. In Table 1, we summarize the commonly used methods by layers. First, at the Application layer, users can install antivirus software and firewalls [21,24,31] to monitor the suspicious behaviors (e.g., frequent access to unauthorized files and encryption) or match the signatures in the viruses database. Second, by intercepting system calls and file access patterns within the OS, recent studies have also proposed to detect ransomware activities via behavioral analysis [33,41,45,57]. Third, users or vendors could ask the hypervisor to take snapshots of the whole runtime to directly recover the data. This method is widely available by major cloud service providers [3,4]. Finally, many prior works have shown that the log-structured design of SSDs can be leveraged to detect and rollback ransomware activities [34,35,43]. The key idea is that valid data which are overwritten and encrypted by the ransomware will be marked as stale. But, the Flash Translation Layer (FTL) usually would not immediately reclaim their space, providing an opportunity to recover users' data before garbage collection.

3 Motivation and Related Work

The cloud is no stranger to ransomware attacks, especially when an increasing number of users are migrating their sensitive data to the public cloud [5–7,13,14,26–29]. A recent report by Zscaler cloud [7] states that ransom attacks have increased by 38% from April 2022 to April 2023, and they predict that attackers are likely to develop new types of ransomware and campaigns optimized for targeting cloud services and workflows. Sophos report [27] also indicates that the average ransom has increased from $812,380 in 2022 to $1,542,333 in 2023, not to mention the cost of the data recovery process and the losses due to downtime. In our cloud, we have also been witnessing a growing number of ransomware attacks on users. In just one quarter (2022 Q3), our cloud has recorded nearly one thousand ransom incidents, leading to an increase of 118% compared to the previous year.

One might wonder, with all the protection approaches available (see Table 1) and vendors' high emphasis on data security, why ransomware attacks are still so prevalent in the cloud. Here, we summarize the key reasons and challenges based on our observations and statistics from the field.

Human mistakes. User awareness is the first, and often the weakest, line of defense against ransomware attacks. To be human is to err, so it is not uncommon for users to fall victims to phishing or malicious emails and other malware. A recent survey by Fortinet [28] reports that phishing remains the top ransom tactic (56%). Other ransomware reports, such as those from Sophos [27] and SpyCloud [29], have likewise emphasized the vulnerability of humans in ransomware defense. Similarly, in our cloud, we discover that nearly all ransomware attacks start with a negligence being exploited (e.g., outdated software and weak password).

Lack of protection in VM. Normally, with antivirus software properly installed and security patches regularly updated, VM should be able to operate safely even under the threats of ransomware. However, in the cloud, VMs, can often run in an under-protected environment due to the following reasons.

First, only a small fraction of VMs are under sufficient antivirus software protection. For instance, Zscaler [2] reports that 17% of organizations are running workloads on unprotected virtual machines which is consistent with our observations from the field. Even for ones who have, they can still be at risks of latest attacks as 28% of VMs attacked by ransomware are running with vulnerable outdated OS and software. Third, while we provide OS images with built-in security support and automatic updates, only 19.4% of users opt in. One main reason, after discussion with multiple users, is that they tend to reuse their own OS images for consistency and compatibility after migrating to cloud.

Snapshot protection is expensive and coarse-grained. Our EBS allows users to persist the Virtual Disk (VD) as a snapshot and later use it to restore to a certain point in time. By taking periodical snapshots, users can conveniently recover from a ransomware attack by simply restoring the VD to the most recent checkpoint before the attack. However, in practice, this is not the case. First, the snapshot-based protection can be expensive. Even with our latest incremental snapshot service, the cost incurred from snapshots can easily 2.5 times more than the monthly rental of the VD, assuming VD is under normal traffic and snapshots are taken on hourly basis. Moreover, even if users are willing to pay the price, recovery by periodical checkpoint is still coarse-grained in the cloud EBS. For example, the highest specification VD in our cloud can achieve a throughput of 4000 MB/s and 100M IOPS. In this case, even if we use minute-level periodic snapshots, it would still result in a significant amount of data loss.

Hardware-based protection is impractical. Academia have proposed multiple approaches for defending against ransomware attacks via hardware assistance [34, 35, 43, 53]. However, these attempts would fall short for large-scale cloud deployment, such as EBS. First, previous works are based on specialized and/or prototype hardware, such as Open-Channel SSDs [34] or FPGAs, yielding a small chance for large-scale deployment. In addition, even if manufacturers manage to produce such hardware, it is still impractical as ransomware evolves rapidly. At the same time, frequently updating the firmware or providing backward compatibility for legacy devices would be a huge challenge.

4 Goals, Opportunities and Challenges

To this end, we have shown that the existing ransomware protection mechanisms can not be shoehorned to cloud services such as EBS. In this section, we first list the design goals of an ideal ransomware protection mechanism for EBS. Then, we discuss the potential opportunities for a practical detection approach based on intrinsic properties of EBS and the ransomware attack patterns. Finally, we present the insights we gained and challenges we faced from building and exploring this preliminary design.

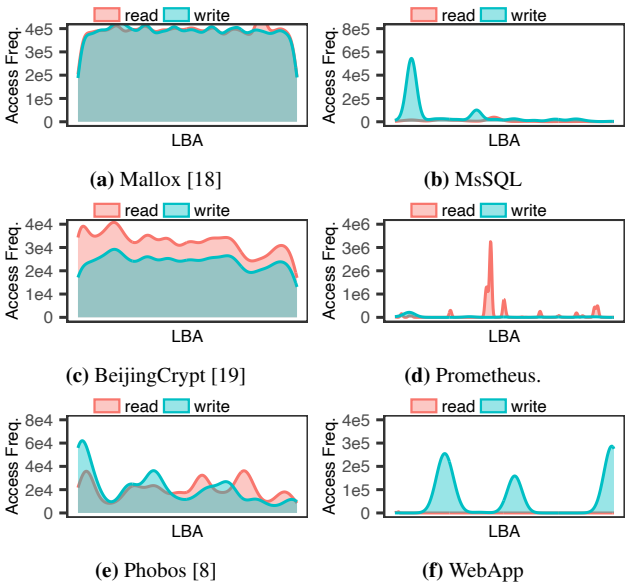
4.1 Design Goals

The role of a cloud vendor profoundly limits our ability and choices in applying existing techniques. Therefore, to develop a practical ransomware detection for EBS, we start by identifying the key requirements.

- *No data lost/tainted after recovery.* The most fundamental requirement for ransomware protection is to ensure that, once detected, no ransomed data is lost during recovery. In other words, all user issued IOs—before, during and after the attack—should be preserved and no tainted data left. On top of that, we need recover data in a timely manner with minimal effort from users.
- *Transparent to users.* The monitoring and detection should not rely on users' awareness or cooperation with the vendor, such as installing certain softwares, using specific OS, or updating patches under certain schedules. In addition, to obey privacy and security protocols, we also cannot directly control the VMs and/or insert (kernel) modules to proactively defend ransomware attacks.
- *Hardware independent to vendors.* As a cloud vendor, our solution should be based on commodity products, instead of depending on features only provided by specialized hardware (e.g., customizable Open-Channel SSDs) or prototype devices (e.g., FPGAs). Moreover, even for the commodity products, the ideal solution should provide backward compatibility, meaning that it can support legacy devices such as early models of SSDs or even HDDs.
- *Low runtime overhead.* The proposed solution should run with low resource consumption (e.g., CPU, memory, space, and network bandwidth). This is because, given the destructive impacts and the increasing prevalence of ransomware, the occurrence—compared to the massive volumes of VDs—is still rare in the wild. For example, on average, less than 0.001% of the VDs in our cloud are subjected to ransomware attacks on a daily basis. Hence, the high CapEx led by high resource consumption would not be acceptable to vendors or tenants.

4.2 Opportunities

While previous ransomware prevention techniques fail to work directly in EBS, we discover that the log-structured block device (LSBD) design bears great similarities with SSD internals. This motivates us to explore the possibility of borrowing ideas from the existing hardware-based methods. Next, we discuss the two similar opportunities our EBS shares with the hardware-based ransomware protection.



(a) Mallox [18] **(b)** MsSQL

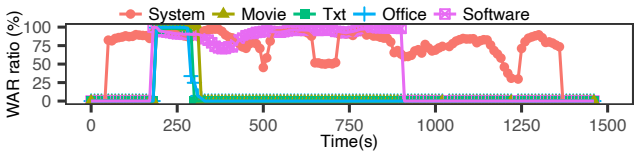
(c) BeijingCrypt [19] **(d)** Prometheus.

(e) Phobos [8] **(f)** WebApp

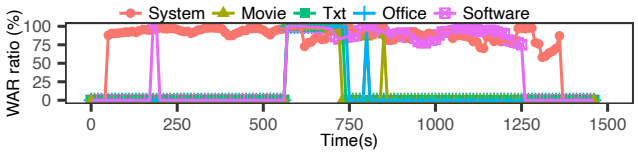
Figure 2: Comparation of LBA access frequency between ransomware and regular applications

Block-level ransomware access pattern. Ransomware usually follows a read-encrypt-write procedure on users' data. For example, a recent survey indicates that up to 76% of ransomware employed encryption-based attack mode [27]. Previous studies have shown that this access pattern holds across different ransomware families and can be caught at file system level (e.g., directory traversal, file type change, access frequency, etc [46, 54]) and device (i.e., SSD) level (e.g., statistics of erasure IO in [34, 35]). As Block Proxies handle VD requests in the format of Logical Block Address (LBA), we further explore the possibility of detecting ransomware attacks at the block level.

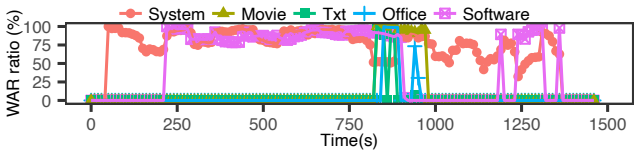
We start with the spatial and temporal patterns of typical ransomware. For the spatial pattern, Figure 2 (a)(c)(e) and



(a) BeijingCrypt [19].



(b) Globeimposter [16].



(c) Babuk [12].

Figure 3: Write-After-Read IO ratio by time on ransom attack

(b)(d)(f) respectively illustrate the LBA access patterns of a VD under ransom attacks and normal workloads. The x-axis represents the normalized LBA, and the y-axis shows the frequency of access within 10 minutes. By comparing the two, we can see an outstanding difference. The ransomware tends to have a similar amount of reads and writes to the same LBA. This is consistent with observations from previous studies [34, 35]. Namely, the "read, encrypt, and write back" pattern of ransomware is manifested as erasure IO (i.e., EIO) or Write-After-Read IO (i.e., WAR IO) at the block-level.

We then examine the temporal pattern of ransomware attack. Figure 3 shows the variations in the WAR ratio (i.e., the ratio of WAR IO to the total write requests) of three VMs (each with five VDs) under various ransomware attack. In each VM, the five VDs consist of one system drive and four data drives loaded with different types of data.

For example, Figure 3(a) illustrates the variations in the WAR ratio of the VM when it is under a BeijingCrypt attack. The WAR ratio of the system drive rapidly rises from 0 to 100% around 60s and maintains a relative high level until the attack ceases at 1370s, at which point the WAR ratio falls back to 0 rapidly. A similar pattern of "climb-maintain-drop" in the WAR ratio can be observed on other data drives, albeit with variations in the start or end time. Figure 3 (b) and (c) illustrate the WAR ratio dynamics for the VM during attacks by Globeimposter and Babuk, respectively, revealing patterns similar to those depicted in Figure 3(a). Hence, we can conclude that cloud-based ransomware also exhibit a distinct pattern of WAR IO at the block level, which is consistent with insights from ransomware detection on hardware level.

To sum up, we can conclude that ransomware attacks at the block level exhibit distinct patterns that can be detected by analyzing the LBA access records.

Method	Precision	Recall	F1-score
SSD-insider++	63.05%	87.53%	73.26%
RanSAP	84.83%	94.37%	89.23%
WaybackVisor	71.36%	93.06%	80.66%
Combine Model	90.74%	92.10%	91.42%

Table 2: Comparison of four ransomware detection algorithms.

Multi-version nature of LSBD. Recall that one key feature of EBS is adopting the log-structured block device (LSBD) in the Block Proxy (See §2.1). In this setup, all writes from the front-end VDs are appended to the end of the log, and the Block Proxy maintains a mapping table—called IndexMap—to track the latest metadata of data. Periodically, Block Proxy reclaims the space with garbage collection (GC).

This design is similar to the SSD's append-only internal architecture, which serves as one of the prerequisites for hardware-based ransomware protection. The knack here is that, in both EBS and SSD, the stale data are usually *not immediately* reclaimed by GC and overwritten with new data. Instead, it can survive for a certain period of time which essentially enables multiple versions of data to co-exist. As a result, in the face of ransomware attacks, we can leverage this multi-version nature to conveniently roll back to early versions by altering the IndexMap.

4.3 A Preliminary Exploration

Based on the above two opportunities, a potential design for ransomware protection in EBS arises. In short, we can actively monitor all incoming IO from each VD and check whether the LBA access matches the ransomware patterns. If detected, we can conveniently roll back the data to a previous clean version. The benefits of this design are that: (1) it only relies on block-level IO records, thereby being transparent to users and hardware independent; (2) it can leverage the multi-version nature of LSBD, hence no extra storage space is required.

In this prototype, there are mainly three components. (1) *IO Trace Collector*: Collect the block-level IO records from Block Proxy including the operation code (read, write or trim), LBA, and length; (2) *Ransomware Monitor*: Periodically analyzing the IO records of all VDs to determine if any are targeted by ransomware attacks. The monitor essentially functions as a feature extractor combined with a classifier. The former extracts handcrafted features from IO records, while the latter is a machine learning-based classifier; (3) *Data Recovery*: Upon detection of an attack, the data recovery mechanism is activated, restoring the data to a pre-attack state by changing the IndexMap.

To validate the effectiveness, we test this prototype on our benchmark, which includes around 150 hours of Block IO traces from 13 common ransomware and 16 types of real-world workloads (see §6.1). In addition, we have implemented four variations by adopting three state-of-the-

art detection methods from previous work including SSD-insider++ [35], RanSAP [40] and WayBackVisor [42], and a combined one (i.e., all features included). Table 2 demonstrates the precision, recall, and F1-score results of the four methods. From the results, we can observe that all four left room for improvement and combining features certainly help (i.e., highest F1-score).

Such solutions still do not meet our goals as they incur data loss or leave tainted data unhandled. First, false negatives is unacceptable as the valid copies of data may be lost during later GC. Second, having a less satisfied precision (i.e., identifying normal activity as ransomware) is also consequential as users' normal IO can be wrongly interrupted and rolled back.

4.4 Challenges

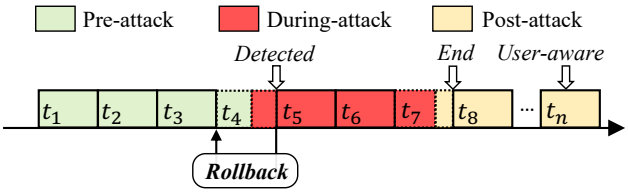


Figure 4: A typical process of ransomware attack, detection, and recovery.

Insufficient features. The straightforward design is based on the SSD and thus bounded by its limited on-chip resources (e.g., computational power and memory space). As a result, the feature extraction and classification algorithms become inherently straightforward (e.g., simple statistics of erasure IO in [35]), thereby yielding low accuracy. This also applies to the solutions based on kernel modules as they, too, need to be lightweight to avoid interfering with normal IO activities.

In EBS, the Block Proxies can have much more resources at hand. In addition, we can also set a standalone machine to run the detection module. Therefore, it is possible for our solution to include more features and adopt complex models for improvement. However, introducing features is helpful only when they reflect the unique characteristics of ransomware attacks, which requires careful analysis on the traces. Moreover, though EBS may have more computational power, spending too many resources on ransomware detection is still not economically acceptable due to the relatively low occurrence rate of ransomware attacks. In summary, we need to strike a balance between accuracy and overhead by identifying the unique characteristics and including them for feature construction.

Impermanent multi-version support vs. data loss. Even if we can achieve a high-accuracy model through the above endeavors, it is unlikely such a solution can always detect the exact timing of the ransomware attacks with no false positives/negatives. Hence, we may still run into data loss when rolling back is wrongly invoked.

Here, we use Figure 4 to illustrate a rundown on typical cases that might lead to data loss. In the figure, each rectangle represents a sample, and the background color indicates the three phases of the attack: pre-, during-, and post-attack. Assume that ransomware initiates an attack at the t_4 window (as indicated by the dashed line) and continues until t_7 window (also indicated by the dashed line). The detection model identifies the attack at t_5 and rolls back the data to t_4.

First, if the proposed solution fails to detect the attack at t_5 (i.e., false negatives), the user's data may become irretrievable if garbage collection (GC) has already kicked in. Note that data recovery can only issued by users but users may not be aware of the attack in time. Second, if the detection model wrongly labels normal IO as ransomware attack (i.e., false positives), it might cause user panic and even lead to incorrect data rollback. Third, even if the ransomware attack is alarmed, the user fails to stop normal IO until t_n which might be several hours after attack, the rollback-based recovery would result in greater data loss that all the user's normal IO during- and post-attack would be reverted.

To solve above issues, the easiest way is to provide the multi-version support for all data modifications permanently. However, this is not feasible as it would quickly deplete the user's purchased disk space within months or even days due to the constant accumulation of data. The same reason also applies to vendors. Therefore, we need to find a solution that can efficiently persist all IO records from right before a ransomware attack until the end of it.

5 `DeftPunk` Design

We now introduce the design of `DeftPunk`, a practical ransomware detection and data recovery framework for cloud EBS. `DeftPunk` does not require users' cooperation, designated software support or customization on hardware in detection, and can guarantee no data loss during the recovery. The key to `DeftPunk` success is employing a two-layer machine learning model to efficiently detect attacks and create snapshots. Then, `DeftPunk` can notify the user and perform the subsequent data recovery. In this section, §5.1 presents an overview of the framework and workflow of `DeftPunk`, followed by a detailed discussion on the three main components: the feature engineering (§5.2), two-layer classifier (§5.3) and data recovery (§5.4).

5.1 Overview

Data preprocessing. Figure 5 demonstrates the high-level procedures and interactions between components in `DeftPunk`. First, the IO Tracer, embedded in the Block Proxies (BP), constantly monitors all incoming IO record (i.e., formatted as `<timestamp, offset, length, operation>`) and group them as IO records by VDs. Then, IO tracer generates samples with a sliding window of 10 seconds. For example, assume a BP that serves three VDs. For a minute

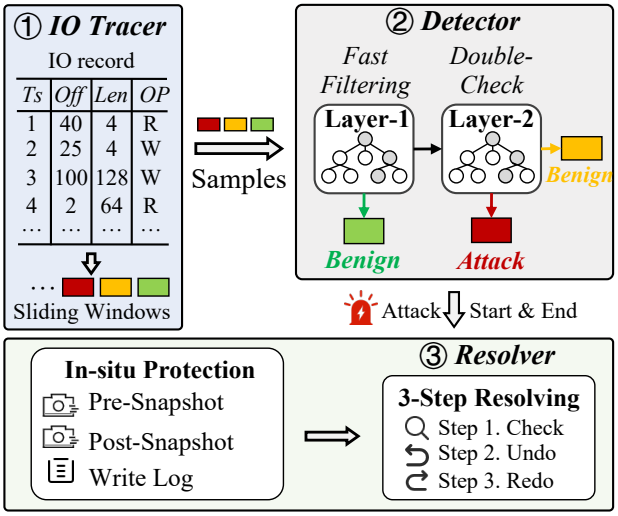


Figure 5: Overview of DeftPunk.

of monitoring, IO Tracer would generate 6 samples of IO records for each VD.

Ransomware detection. Upon receiving the IO record samples, DeftPunk first uses the *Layer-1 Classifier*, based on simple features (e.g., IO count) and a decision tree, to perform fast scanning with an emphasis on high recall and low computational cost. If the sample fails to pass the first layer (i.e., suspicious of a ransom attack), it will be further checked by the *Layer-2 Classifier*. The second model uses a superset of Layer-1's features by including more complex ones, (e.g., statistics of entropy, Working Set Size, and IO offset), and employs a more sophisticated model (i.e., XGBoost), aiming at high precision. If the sample is again labeled as positive, DeftPunk would trigger snapshot generations.

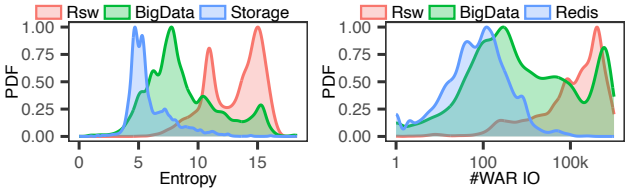
Creating snapshots. When a positive sample arrives, the snapshot worker would first create a Pre-attack Snapshot and a Post-attack Snapshot at the beginning and the end of it, respectively. As a ransom attack may span across multiple 10-second samples, the snapshot worker continues to create new Post-attack Snapshot to replace the previous one until the incoming sample becomes negative. Meanwhile, during the attack, DeftPunk would also pause the EBS garbage collection for this VD to prevent the accidental deletion of the valid data until the post-attack snapshot is finalized.

Data recovery. When the user has been notified of the attack (e.g., via our alert message or discovering certain data become inaccessible), one needs to issue a recovery request. DeftPunk would follow a three-step recovery process. (i) *checking*: DeftPunk first checks and lists all tainted LBAs (i.e., 4KB long each) during the attack. If a LBA is further modified (by the user) after the post-attack snapshot, then data in that LBA would not be recovered. (ii) *undo*: DeftPunk would revert all data in tainted LBAs to the version of the pre-attack snapshot. *redo*: DeftPunk would run another rule-based model to identify IOs made by the users and redo them.

5.2 Feature Engineering

In §4.4, we mentioned that one key reason for suboptimal performance of existing solutions is the insufficient feature engineering. To construct features for DeftPunk, we first conduct an extensive study on real-world VDs' IO traces to identify the fundamental differences between ransomware and normal user behaviors (§5.2.1). Based on the insights, we then devise the extended features for the two-layer model.

5.2.1 Characterizing IO Behavior



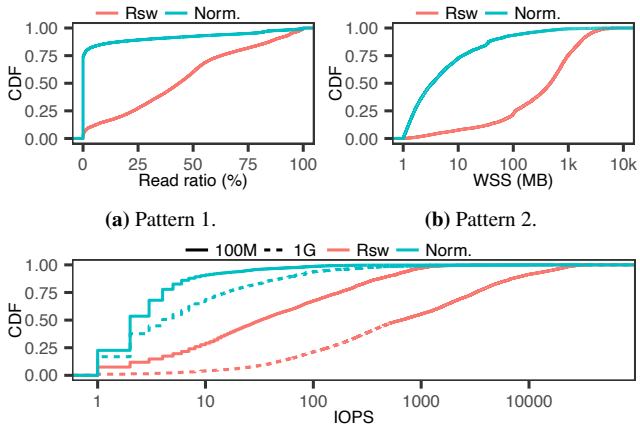
(a) Offset entropy **(b)** WAR IO count

Figure 6: Opportunities of detecting ransomware at block-level.

We first reason why typical features used in existing works, such as *offset entropy* in [40,42] and *WAR IO count* in [34,35], can be ineffective. In Figure 6(a), we present the probability density function (PDF) of the offset entropy for samples in our dataset. "Rsw" indicates the PDF of all ransomware samples. "BigData" and "Storage" represent two different types of common workloads. We can observe a distinct difference in the PDF of offset entropy between Rsw and Storage, while there exists large PDF overlap between BigData and Rsw. Figure 6 (b) exhibits a similar pattern on Bigdata and Rsw.

This comparison clearly shows that normal workloads can be easily mislabeled (or the other way around) if we solely rely on a selected few simple features. We believe that one major root cause is that previous work mostly focus on positive samples (i.e., ransom attack IOs) without noticing or studying the similarities they share with normal traffic. Hence, we study both the patterns of ransomware and normal I/O behaviors and propose three patterns for better feature engineering.

- **Pattern 1.** Ransomware typically exhibits nearly equal amounts of read and write in bytes, whereas the read-to-write ratio of normal IOs is often skewed. Figure 7(a) presents the cumulative probability density (CDF) of the read-ratio (i.e., the proportion of read requests to total requests). We can see a notable gap between the CDF of normal workloads (Norm.) and ransomware (Rsw), with the average value for Norm. being X and the average value for Rsw being Y.
- **Pattern 2.** The IO offsets of ransomware are distributed more broadly across the LBA space, and each offset is typically accessed only once. In contrast, the IO offsets under



(a) Pattern 1. **(b)** Pattern 2.

(c) Pattern 3.

Figure 7: Three unique patterns that can distinguish between ransomware and normal workloads.

normal workloads tend to be concentrated. Figure 7(b) shows the CDF of the working set size (WSS) for the two categories of samples, and we can similarly observe a significant difference between Norm. and Rsw.

- **Pattern 3.** Ransomware displays obvious WAR on the system disk, particularly in the Master Boot Record (MBR) region at the beginning of the LBA, whereas such operations are rarely performed by normal users. Figure 7(c) displays the IOPS (Input/Output Operations Per Second) for the two categories of samples within the first 100MB and the first 1GB of the LBA.

5.2.2 `DeftPunk` Feature Engineering

Based on the findings above, we construct the following as `DeftPunk`'s enhanced feature engineering for ransomware detection. Table 3 presents the 43 features we employed along with their corresponding meanings. For ease of understanding, we categorize these features into 5 classes, which include:

- **IO Dependency**. This set of features characterizes the behavior of adjacent I/O operations in a temporal context. In addition to including Write-After-Read (WAR), which has been adopted by other works [34, 35], we also incorporate three other types of read-write sequences, such as Read-After-Write (RAW), Read-After-Read (RAW), and Write-After-Write (WAW).
- **IO Statistics.** This part constitutes the basic statistics of VD IO, including the IO count, total bytes, and IO size. Here, features are made separately for read (R), write (W), and the sum of read and write (RW). The design of this feature set is inspired by **Pattern 1.**
- **Working Set Size (WSS)**. Inspired by **Pattern 2**, we track the working set size (WSS) for 6 types of I/O. Here, WSS refers to the proportion of the LBA space that is accessed by a specific type of IO.
- **Offset Statistics**. Compared to WSS, this category of features characterizes the distribution of I/O across the LBA at a finer spatial granularity, such as the mean, variance, standard deviation, coefficient of variation (CoV) [32], and entropy [56] of the IO offset. The design of this type of feature is also inspired by **Pattern 2**.
- **Access on LBA Head Region**. As mentioned in **Pattern 3**, ransomware may tamper with data in the head region of the LBA. Bearing this in mind, we conduct statistics on IO that fall within the first 100MB and 1GB of the LBA, including the I/O count and total bytes.

In Table 3, we also list the computational complexity (in Big O notation) for each class. Note that the traditional definition of Write-After-Read (WAR) is to check whether there are sequential read and write requests accessing the same LBA offset. However, we extend the definition of WAR by relying on both the offset and the length to decide whether the LBA accessed by the sequential read and write requests are overlapped. Therefore, our check for WAR requires a time complexity of O(logn) rather than O(1). The reason to do this is that we find that ransomware can read a large volume of data and then writes it in smaller chunks, exhibiting the behavior of "read-write-...-write". According to the traditional definition of WAR, this behavior would be characterized as 1 WAR and 2 WAWs, which can be similar to normal user IO. However, our definition of WAR would describe this behavior as 3 WARs and 2 WAWs, thereby distinguishing it from the I/O behavior of normal users.

5.3 Two-layer Model

The enhanced feature engineering can improve the precision/recall of the detection. However, simply building a classifier based on the entire set of features would not be practical for production deployment. This is because certain features such as IO dependency, offset entropy, and WSS would have high computational complexity and thus yield a high overhead in detection. For example, given a VD with 1 million IO records, the processing time for IO dependency can be 14 seconds for one thread. In this case, blindly applying the entire set of features would consume nearly 56 virtual CPU cores for 1 million IOPS online only for feature calculation.

Therefore we propose a two-layer model to balance the trade-off between overhead and detection accuracy. First, we use a simple feature set in the first-layer model for fast filtering the majority of non-ransom activities. We apply the complex feature set with second classifier to scrutinize the suspected cases labeled by the first model. Specifically, the two-layer classifier is implemented as follows.

Layer-1: Fast and Broad Filtering. This layer aims to achieve a high recall and rapid initial scanning of ransomware using features with O(1) complexity. Hence, we evaluate the detection performance of three commonly-used simple classifiers, including k-Nearest Neighbors (kNN) [39], Logistic Regression (LR) [50], and Decision Trees (DT) [51], with

Type	Description	Complexity	#Features
IO Dependency on Block	**IO count / Bytes** of (RAW / WAR / RAR / WAW) IO	O(logn)	8
IO Statistics	**IO count / Bytes / Size / Bps** of (R / W / RW) IO	O(1)	11
Working Set Size (WSS)	**WSS** of (R / W / RAW / WAR / RAR / WAW) IO	O(logn)	6
IO Offset Statistics	**Var / CoV** of (RW) IO	O(1)	2
	Entropy of (R / W / RAW / WAR / RAR / WAW) IO	O(logn)	6
Access on LBA Head Region	**IO count / Bytes** on first (100M / 1G) of (R / W) IO	O(1)	8

Table 3: Feature engineering of `DeftPunk`. *In the second column "Description", the bold text separated by slashes ("/") represents different metrics, while the parts within parentheses separated by slashes represent different IO types. For example, the meaning of the first row is to calculate the IO count and Bytes for each of the four IO types: RAW (Read-After-Write), WAR, RAR, and WAW. Therefore, there are a total of* $2 \times 4 = 8$ *features. R, W and RW IO refers to read, write, and read-write, respectively.*. **Var**: *variance*; **WSS**: *working set size*; **CoV**: *coefficient of variance*.

Layer	Model	Precsion	Recall	F1-score
Layer-1	KNN	73.1	82.5	77.5
	LR	32.7	91.1	48.1
	DT	**87.5**	**95.9**	**91.5**
Layer-2	RF	96.2	97.6	96.9
	lightGBM	95.1	97.1	96.1
	CatBoost	95.4	98.5	96.9
	XGBoost	**95.8**	**98.6**	**97.1**

Table 4: Two-tier model selection. **KNN**: *k-nearest neighbors*; **LR**: *logistic regression*; **DT**: *decision tree*; **RF**: *random forest*.

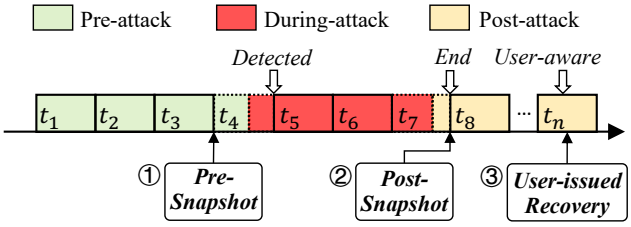


Figure 8: `DeftPunk`'s timeline for lossless data recovery.

the results depicted in Table 4. We choose the Decision Tree (DT) as the classifier for layer-1 as it achieves the highest recall (95.9%).

Layer-2: Accurate Double Check. Suspected cases from layer-1 are passed to the layer-2, which employs computationally expensive features that can better distinguish between ransomware and normal behavior. Here, we add all the features mentioned in Table 3 to the model, and similar to layer-1, we test the performance of four commonly-used complex classifiers, including Random Forest (RF) [36], LightGBM [44], CatBoost [52], and XGBoost [38], with the results presented in Table 4. The results indicate that XGBoost achieves the highest F1-score (97.1%) and thus is chosen as our layer-2 classifier. Our further experiments in §6.6 demonstrate that the two-layer model maintains the same detection performance as single-layer model but with much less computation needed.

5.4 Creating Snapshots

Once an attack is detected, `DeftPunk` generates a pair of snapshots, called pre-/post-attack snapshots. The goal is to "lock in" the effects of the ransomware to make sure all data modification during the period are recorded and recoverable. Then, inspired by the "undo-redo" mechanism in database, `DeftPunk` reverts all the LBAs modified by the ransomware. We also employ another rule-based model to identify the LBAs modified by the user and redo them.

Creating pre-attack snapshot. Once notified by the two-layer classifier, the snapshot worker checkpoints the IndexMap and data of the VD at the time of beginning of the labeled batch window. This is enabled by the multi-version nature of LSBD, which allows the snapshot worker work backwards as long as the previous changes have not been garbage collected. This is guaranteed by setting up the length of a sample to be 1 minute and configuring the GC to only collect stale data that are at least older than 30 minutes. Meanwhile, we also pause the GC on this VD.

Creating post-attack snapshot. Generating the post-attack snapshot is different. As a ransomware attack might be longer than a single sample, `DeftPunk` waits until the two-layer classifier labels an incoming as negative. Then, the snapshot worker creates a post-attack snapshot at the end of last labeled sample and resume GC. In addition, we also record all the writes, including data and LBA, between two snapshots as a write log. Note that the writes between pre-snapshot and detection time can be obtained due to the multi-version support. Plus, the modifications after the detection time can be recorded as we have paused the GC. Finally, we compare the IndexMaps between the two snapshots to only keep data pre-snapshot that are modified during the attack and drop others for space efficiency.

5.5 Data Recovery

To this end, we have acquired three pieces of data, including a pre-attack snapshot (i.e., data, LBA and versions at t_3 in Figure 8), a post-attack snapshot (i.e., LBA and versions at t_8), and a write log (i.e., data and LBA between t_1 and t_5). Now, `DeftPunk` follows three steps to recover the data.

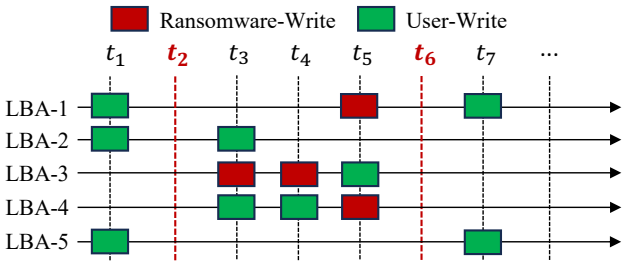


Figure 9: Five cases of LBA data recovery by `DeftPunk`.

Identifying the tainted LBAs. Once the user has been notified of the ransomware attack, they can initiate the request for data recovery. `DeftPunk` would first check if the LBAs listed in the post-attack snapshot are modified by the users after the attack. `DeftPunk` would not try to recover data from such LBAs as they are deemed valid by the users (i.e., LBA-1 and LBA-5 in Figure 9).

Undo. Executing undo is straightforward. `DeftPunk` would revert all involved LBAs (i.e., only modified during the attack but not after) to the version of the pre-attack snapshot.

Redo. Ransomware usually skips files that are already opened (e.g., LBA-2) for writes or kills the user's writing process (e.g., LBA-1) to ensure data integrity of the encrypted files. Still, it is possible that the user's IO exists in the tainted LBAs such as LBA-3 (i.e., user overwrites encrypted LBA block) and LBA-4 (i.e., user's writing is not immediately stopped).

To avoid data loss, `DeftPunk` needs to redo these users' IOs. First, `DeftPunk` employs a rule-based model, which checks the in-place write-after-read (WAR) patterns, to determine whether an IO is indeed modified by the ransomware. Note that, in this case, simply checking the WAR pattern can effectively single out the ransomware IOs because these LBAs have already been filtered by the two-layer model. In other words, the chance of a user's IO coincidentally matching the ransomware's IO pattern is extremely low. We further discuss how to handle such a corner case in §5.6. Then, `DeftPunk` would drop the tainted IOs and apply the normal ones in order.

5.6 Corner cases.

Missing the start/end. The ransomware might start or end right around the 10-second sliding window splitting. Hence, a few IOs made by the ransomware might evade the two-layer model checking and taint the users data. In this case, we further include IOs from one more sample before and after the labeled sample to the write log to avoid data loss.

Mislabeled writes. During the attack, the users might coincidentally write to the LBAs with the similar pattern as the ransomware, e.g., encrypting or rewrite the files. Consequently, the rule-based model may incorrectly mark the behaviors as ransomware and revert them. Note that experiencing this type of mislabeling is rather unlikely in practice. This is because all known ransomware would choose to terminate the user's writing process targeting the same LBAs or simply avoid these addresses (i.e., files) to ensure the integrity of ransomed data. So far, we have not observed any such cases in the field.

Nevertheless, we employ a manual checking process if the user is unsatisfied with the automatic undo and redo. We provide the users with a tool to associate files to LBAs. Then, for the files (and their corresponding LBAs) that are mistakenly reverted, the users can choose to drop/apply the writes on an individual basis. Note that simply using this tool and asking the users to manually check all the LBAs would be impractical given the sheer volume of IOs during runtime.

Data inconsistency. `DeftPunk` is a block-level solution and hence does not provide file/application-level consistency guarantees. It is possible that recovery process can lead to data inconsistency. For example, applying a user's overwrites on the encrypted data may result in a corrupted file (i.e., LBA-3). To resolve this, restoring to the pre-attack snapshot guarantees a clean start. In addition, users can again use our manual checking tool for a finer control of what IOs to apply or drop on tainted LBAs.

Multiple attacks. As the data recovery is only triggered by the user after the notification, it is rare but still possible that the same LBAs have been attacked multiple times. Now, we further discuss the case of two consecutive attacks. The analysis and solution shall apply to the case of multiple attacks (e.g., three or more). Depending on the distance between the two attacks, we can have the following cases:

- *One pair of snapshots.* When the the second attack closely follows the first one (i.e., within 2 samples), `DeftPunk` would treat the two as a single attack as the post-attack snapshot would include an extra sample (see corner case 1). Then, `DeftPunk` performs data recovery as usual. If there is a user's IO between the two attacks, `DeftPunk` can use the rule-base model to check and only redo user's IOs.
- *Two pairs of snapshots.* If the two attacks are far apart (e.g., one day away), `DeftPunk` creates a pair of snapshots for each attack. Then, `DeftPunk` performs data recovery on each attack separately in time order. In this case, the user's IOs between the two attacks would not be affected as the two are treated as independent attacks.

5.7 Runtime Modes

To avoid mislabeling, `DeftPunk` runs in a per-batch mode in the field. Specifically, a batch is 10-minute long and thus includes 60 10-second records. `DeftPunk` labels a batch as positive as long as one sample has been flagged.

Further, users can enable/disable Deftpunk on a Viritual Disk (VD) basis. For example, if a user has mounted multiple VDs, the user can choose to only protect important VDs (e.g., data drives) with Deftpunk but not ones for caching. In addition, we initially allowed system administrators to use only a part of the feature set or just Layer-1 of the two-

App.	#Batch	#IO($1e5$)	App.	#Batch	#IO($1e5$)
Prometheus	980	893.5	MongoDB	128	197.2
Storage	938	277.6	Postgress	100	48.4
MsSQL	921	459.2	Oracle	77	112.7
MySQL	900	949.6	RabbitMQ	55	82.1
WebAPP	739	159.5	ElasticSearch	40	32.2
Redis	435	70.1	etcd	29	7.4
MessageBus	370	90.6	influxDB	6	0.9
BigData	220	749.0	Kafka	5	18.1

Table 5: Negative samples in `DeftPunk` benchmark.

Config	Content	#Types
Ransom family	loki [22], BeijingCrypt [19], makop [20] Sodinokibi [25], babuk [12], VoidCrypt [15] phobos [8], GlobeImposter [16] wannacry [30], mallox [18]	13
OS version	WinServer 64-bit 2016, 2019, 2022 (w/ and w/o container), CentOS	6
App.	Copying, Massive Write, Massive Read, ZIP CRYPT, MySQL	5

Table 6: Configurations of simulated ransomware attack.

layer model to achieve lightweight detection. However, we later found that such a trade-off is not efficient as it leads to a significant drop in detection performance with minor reduction in overhead. We present an experiment on this in §6.6. We recommend that users enable the full feature set and adopt the complete two-layer model.

6 Evaluation

Our evaluations intend to answer the following questions:

- What is the composition of the benchmark? (§6.1)
- How does `DeftPunk` perform against other methods on a per-sample basis? (§6.2)
- How does `DeftPunk` perform in zero-shot scenario? (§6.3)
- How does `DeftPunk` perform in per-batch setup? (§6.4)
- Does the feature engineering work? (§6.5)
- What is `DeftPunk`'s runtime overhead? (§6.6)
- How is `DeftPunk` in deployment? (§6.7)

6.1 Ransomware Benchmark

The dataset is composed of two parts:

Negative samples (normal IO). Benign samples consist of I/O records from EBS virtual disks (VDs) running real-world workloads from the field. Table 5 shows the types of workloads, the number of batches, and the volume of IO records. Note that each batch refers a group of IO records of a VD within a 10-minute span. Each IO record is quadruple, formatted as `<timestamp, offset, length, operation>`. To generate negative samples, we reuse the sliding window mechanism. Therefore, a 10-minute batch can generate 55 1-minute long negative samples. In total, we have gathered nearly 2 million samples.

Postive samples (ransomed IO). We generate positive samples by simulating ransomware attacks atop the normal IOs. Specifically, we follow a "ransomware-OS-application" configuration to generate batches. For each configuration, as shown in Table 6 we mix and match the ransomware families (1 in 13), operating systems (1 in 6), and background applications (1 in 5). Each generated batch is around 10 minutes long and we use the same sliding window methodology to generate positive samples. Note that a ransomware attack can be shorter than 10 minutes. Therefore, we also discard samples that do not have any ransom activities (i.e., samples before/after the ransomware in the generated batch). In total, we have 52 thousands positive samples.

Metric. Ransomware detection is a typical binary classifier, determining whether a sample is positive (i.e., ransomed) or not. Therefore, we use *precision* (i.e., the proportion of positive identifications that are actually correct), *recall* (i.e., the proportion of actual positives that are correctly identified), and the *F1-score* (i.e., the harmonic mean of precision and recall) for measurement.

6.2 Per-Sample Test

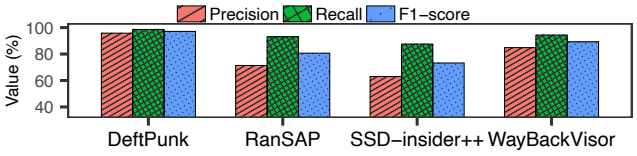


Figure 10: Overall performance of `DeftPunk`.

We compare `DeftPunk` with three other ransomware detection methods including SSD-insider++ [35], WaybackVisor [42] and RanSAP [40]. We split the dataset as 90% for training and 10% for testing. We use 10-fold cross-validation which means we train and test the models 10 times and each time with a different 10% as test set.

We calculate the average recall, precision, and F1-score of each candidate from the 10 tests. In Figure 10, we can observe that `DeftPunk` clearly outperforms all others with a 98.6% in recall, 95.8% in precision, and 97.1% in F1-score. This validates the effectiveness of `DeftPunk`'s feature engineering and two-layer model.

We further analyze the false positive/negative cases. First, it is possible `DeftPunk` incorrectly flags normal IOs as ransomware activities (i.e., precision is not 100%). Normally this is because users' IOs exhibit the same write-after-read patterns as ransomware activities, such as data encryption, in-place compression, and format conversion (e.g., changing a mp4 video file to a mkv one). `DeftPunk` alleviates this issue by leveraging multiple patterns instead of just write-and-read for detection. Still such coincidences may occur and lead to the mislabeling. Note that having a few false positives

is acceptable. Recall that DeftPunk follows a detect-notify-rollback process. Therefore, if DeftPunk mislabels normal activities, the notified users can just ignore the alerts, and inform us to delete these snapshots. No rollback would be executed unless the users confirm an attack has occurred.

Second, it is possible DeftPunk misses some ransomware activities (i.e., recall is not 100%). In the per-sample test, the main reason is that the ransomware attack, which usually lasts several minutes, can span multiple samples. Certain samples (e.g., the very first or last one) may not contain enough ransomware activities to be successfully identified, thereby lowering the recall. Note that, in practice, DeftPunk is deployed in a per-batch manner (i.e., 10-minute window), where the recall is near 100% (see §6.4).

6.3 Zero-shot Detection Performance

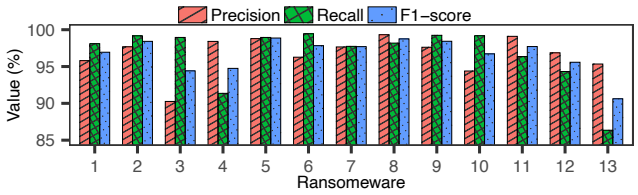


Figure 11: Zero-shot performance of DeftPunk.

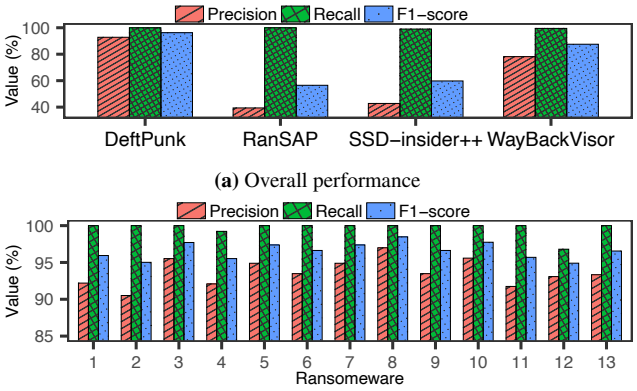
Given that ransomware can be fast evolving, we further measure DeftPunk's detection performance on "unseen" ransomware. Each time, we remove one type of the 13 ransomware families from the training dataset but still keep samples from that family in the testing set (i.e., zero-shot).

Figure 11 displays the results where each group of three bars show the performance of a specific ransomware family were removed from the training set. We can see that DeftPunk is capable of effectively detecting most unseen ransomware, maintaining at least 95% recall and 90% precision. This indicates that our feature engineering captures the common characteristics of ransomware.

The only exception is Wannacry. Further analysis suggests that Wannacry is intentionally slowing down its operation (e.g., CPU utilization is below 25%, and disk throughput is below 10%) to evade detection. Similar to the aforementioned discussion on false negatives, the lower recall is caused by the scant amount of ransomware activities in some samples due to the intentionally *diluted* IOs. Note that such an evasion strategy—while useful in the per-sample test—is not effective in our deployment scenario (i.e., per-batch experiment, see §6.4) for more details.

6.4 Per-Batch Experiment

Note that in the previous two experiments we calculate the performance on a per-sample basis. While they validate the overall designs of DeftPunk in a microbenchmark fashion, DeftPunk in practice is deployed on a per-batch checking



(a) Overall performance

(b) Zero-shot performance

Figure 12: Per-batch performance of DeftPunk.

basis (see §5.4). In other words, as long as one sample within a 10-minute batch is marked as positive, the entire batch is marked as positive. Note that per-batch testing is closer to real-world setup due to the streaming of IOs.

In Figure 12, we rerun the experiments in §6.2 and §6.3 but with a per-batch basis. We can see that DeftPunk can always achieve near 100% recall across all situations with a minor (around 2% on average) decrease in precision. Surprisingly, with per-batch detection, DeftPunk can even successfully identify Wannacry ransomware in zero-shot experiment which intentionally dilutes its IO for evasion. On stark contrast, other methods while also have an increasing recall, can experience considerable precision loss, yielding low practicality (i.e., frequently issuing false alarms).

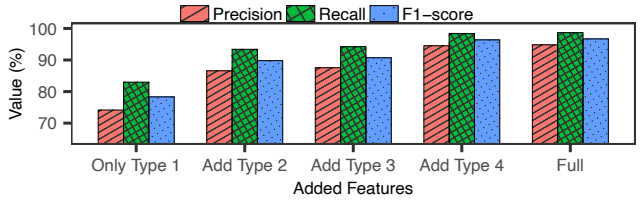
Obviously, compared to experiments in §6.2 and §6.3, the precision and recall in per-batch experiments are much higher. This is because the per-batch test reduces the chances of mislabeling by taking multiple samples (i.e., 1 batch = 55 samples) into consideration. Nevertheless, DeftPunk may still fail to identify certain ransomware activities. One representative case is the Babuk ransomware (i.e., the 12th in Figure 12(b) being undetected in a VD with mostly text files. Further analysis reveals that Babuk is designed to not encrypt text files, thus being latent (i.e., not encrypting files) and avoiding the detection.)

6.5 Effectiveness of Features

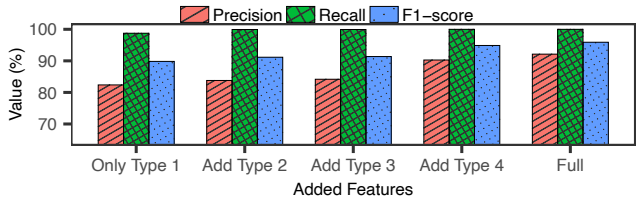
Now, we measure the effectiveness of DeftPunk's feature engineering. Specifically, from only one type of features, we add on another set of features and evaluate the overall performance of DeftPunk on both per-sample and per-batch basis. In Figure 13, we can see that DeftPunk's performance steadily increases with the addition of more features. This validates the effectiveness of DeftPunk's feature engineering.

6.6 Runtime Overhead

Monitoring and preprocessing in DeftPunk incur negligible overhead as the IO Tracer only checks and packs the quadru-



(a) Per-sample performance.



(b) Per-batch performance.

Figure 13: Ablation study of DeftPunk's feature engineering. *In the figure, the Type 1 features corresponds to all features associated with the 1st row (IO Dependency on Block) in Table 3, with Type 2 through Type 4 following in a similar fashion.*

Model	Precision	Recall	F1-score	Time (s)
Layer-1 Only	87.3%	95.9%	91.5%	**3.72**
Layer-2 Only	94.9%	98.6%	96.7%	**56**
DeftPunk	95.8%	98.6%	97.1%	**4.85**

Table 7: Comparison of DeftPunk v.s. Layer-1/2 only.

ple metadata in an asynchronous fashion (i.e., non-blocking).

For detection, we evaluate the speedup made by the two-layer model by comparing it against the Layer-2 only model. Table 7 shows that, on processing 1 million IOs with one thread, the two-layer model only takes 4.85 seconds, yielding a 11.5× speedup over the Layer-2 only model. Including the overhead for detection, as well as data processing and transfer, we are able to process data at 140,000 IOPS using a 2.7GHz vCPU, which means that on processing 1 million IOs per second we need around 7 vCPUs.

We can enforce DeftPunk to run with only Layer-1 enabled for lower overhead. In Table 7, we can see that, with only first layer, DeftPunk can speed up 24.3% (i.e., from 4.85 to 3.72 seconds). However, both precision and recall decrease significantly (e.g., from 95.8% to 87.3% in precision). By weighing the trade-off between performance and speedup, we believe the two-layer model is the best choice for DeftPunk.

For recovery, DeftPunk introduces additional storage space because of the snapshots and write logs. For a typical 100GB VD under ransom attack, our experiments show that the storage overhead is around 150MB on average, much less than the periodical snapshots.

6.7 Deployment

Currently, we have deployed DeftPunk in our EBS service for limited internal users. For each internal EBS cluster (hosting more than 30K VDs), we deploy one machine (8 virtual CPU(2.7 GHz), 32 GiB Memory) to run DeftPunk's detection and recovery components. DeftPunk has already successfully prevented two attacks with data fully recovered within 240 minutes after internal users issued reports. We expect to release DeftPunk for public review in the near future.

7 Potential Limitations

7.1 EBS-specific Solution

The success to DeftPunk is based on two properties of ALIBABA EBS, the log-structured design and block-level IO support. However, this does not mean DeftPunk can only be applied to ALIBABA EBS. First, the log-structured design is widely adopted in many cloud storage systems (e.g., HDFS [55]) and file systems (e.g., F2FS [47]). Practitioners can also leverage the multi-version support, snapshots and GC pausing to secure a full-copy of data modifications during attack. Second, block-level IO monitoring and analysis can be achievable for cloud vendors and system administrators. Moreover, with our open dataset and users' own traces, they can also train their own models to detect ransomware activities following our feature engineering practice.

7.2 Effectiveness on Unknown Ransomware

In this paper, DeftPunk shows its effectiveness on 13 types of well-known ransomware. There can be others that are not covered or even still under development. However, we believe this does not pose a great threat to DeftPunk's validity. First, the zero-shot experiments show that the features of DeftPunk can generalize well to unseen types of ransomware. Second, DeftPunk can be quickly adapted to new ransomware by introducing new features and deployed to production systems as it is a pure software solution.

7.3 Threats of Mimicry Attacks

The features of DeftPunk arise from the three ransomware IO characteristics (i.e., Pattern 1-3 in §5.2.1). First, to encrypt, ransomware exhibit unique in-place write-after-read LBA patterns. Second, to quickly attack multiple files, ransomware would touch many files and encrypt a small proportion in each of them, yielding a wide range of sporadic LBA accessing. Third, to increase the impacts, ransomware would especially favor important areas (e.g., head region).

It is possible that future attackers may choose to evade DeftPunk detection by avoiding the above behaviors in their ransomware. However, we believe such efforts can be ineffective or even against the interests of ransomware (i.e., becoming less stealthy or unprofitable). First, using different access patterns (e.g., Wannacry slowdown) can still be caught by DeftPunk even under zero-shot setup (see §6.4). Moreover, further slowing down the IOs can backfire as it takes much longer time to finish file encryptions and can easily alert the users. Additionally, countermeasures for this slowdown attack in DeftPunk is straightforward—condensing the

workloads (e.g., include a set of features to treat 10-minute workloads as 1-minute ones).

Second, we have seen "pseudo-ransomware" (e.g., wiperware [1]) choose not to encrypt but to directly destroy data (e.g., filling zeroes), thereby not showing write-after-read patterns (only writes, no read). But, they are no longer active as few victims would pay the ransom and current ransomware usually allows victims to decrypt a small proportion of data to show validity. In addition, we notice certain early ransomware (e.g., Gpcode.ak) do not follow in-place write-after-read but to delete original files and create new ones with encryption. However, they, too, soon have died out due to showing unusual patterns (i.e., frequent file deletion) and excessive IO traffic (creating a mass amount of files), which can be easily singled out and restored (e.g., PhotoRec for Gpcode.ak [23]).

Third, we can see that even if ransomware avoid Pattern 2 and 3 (i.e., "Only Type 1" in Figure 13), `DeftPunk` still shows high recall (98.8%). But, by doing this, ransomware would impact less files and/or target less important ones, thus being unattractive to the attackers.

In other words, Patterns 1-3 persist across all 13 families of ransomware are as a result of such patterns reflecting the fundamental nature of ransomware, especially after generations of evolutions. Admittedly, future attacks may evade detection in a different fashion or combine multiple strategies together. For example, ransomware can encrypt a selected set of important files with a more “diluted” pattern to minimize footprint.

To sum up, two lessons we have drawn from the above analysis are: (1) Closely monitoring the latest development. Propagation of new ransomware, while fast, still takes time to spread, which renders a window for us to analyze. (2) Building adaptive solutions. The security "arms race" is often inevitable and never-ending. One important advantage of `DeftPunk` is software-based and thus offers high flexibility, which further enables us to quickly adapt to emerging threats.

8 Conclusion

In this paper, we revisit a pressing problem in data security: defending against ransomware attacks. With a large-scale study on the IO characteristics of ransomware, we identify a rich set of features and leverages the log-structured multi-version properties to build `DeftPunk`, a block-level ransomware detection and recovery system. Our extensive evaluation shows that `DeftPunk` can achieve nearly 100% recall with 95.8% precision with minor overhead.

Acknowledgments

The authors would like to thank our shepherd Prof. George Amvrosiadis and anonymous reviewers for their meticulous reviews. We are also grateful for the support from the EBS team, and feedbacks by Amber Bi, Keely Xu, and Qingke X.F. on early drafts of this paper. This research was partly supported by Alibaba ARF/AIR program and NSFC(62102424).

References

[1] NotPetya Technical Analysis by LogRhythm Labs. https://gallery.logrhythm.com/threat-intelligence-reports/notpetya-technical-analysis-logrhythm-labs-threat-intelligence-report.pdf, 2017.

[2] 2022 Cloud (In)Security Report. https://www.zscaler.com/blogs/security-research/2022-cloud-security-report, 2022.

[3] AWS Backup Anomaly Detection for Amazon EBS Volumes. https://aws.amazon.com/cn/blogs/storage/aws-backup-anomaly-detection-for-amazon-ebs-volumes/, 2022.

[4] Storage Anti-Ransom Solution. https://e.huawei.com/cn/solutions/storage/oceanprotect/ransomware, 2022.

[5] 2023 Ransomware Trends Report. https://www.veeam.com/ransomware-trends-report-2023, 2023.

[6] 2023 State of the Cloud Report. https://info.flexera.com/CM-REPORT-State-of-the-Cloud?lead_source=Website%20Visitor&id=Flexera.com-PR, 2023.

[7] 2023 ThreatLabz State of Ransomware. https://info.zscaler.com/resources-industry-reports-2023-threatlabz-ransomware-report-old, 2023.

[8] A deep dive into Phobos ransomware. https://www.malwarebytes.com/blog/news/2019/07/a-deep-dive-into-phobos-ransomware, 2023.

[9] AliCloud - Elastic Block Storage. https://www.alibabacloud.com/zh/product/disk, 2023.

[10] Amazon Elastic Block Store. https://aws.amazon.com/cn/ebs, 2023.

[11] Azure Disk Storage. https://azure.microsoft.com/zh-cn/products/storage/disks, 2023.

[12] Babuk Ransomware: In-Depth Analysis, Detection, Mitigation, and Removal. https://www.sentinelone.com/anthology/babuk/, 2023.

[13] Batched, Fileless, Highly Adversarial | Annual Report on Cloud Ransomware Attacks in 2022. https://developer.aliyun.com/article/1150967, 2023.

[14] Cloud Ransomware | Understanding And Combating This Evolving Threat. https://www.sentinelone.com/cybersecurity-101/cloud-ransomware-understanding-and-combating-this-evolving-threat, 2023.

[15] Dark - VoidCrypt (.dark) ransomware virus removal and decryption options. https://www.pcrisk.com/removal-guides/24606-dark-voidcrypt-ransomware, 2023.

[16] GlobeImposter. https://malpedia.caad.fkie.fraunhofer.de/details/win.globeimposter, 2023.

[17] Google Cloud - Persistent Disk. https://cloud.google.com/persistent-disk?hl=zh-CN, 2023.

[18] How to eliminate the Mallox ransomware from a computer? https://www.malwarebytes.com/blog/news/2019/07/a-deep-dive-into-phobos-ransomware, 2023.

[19] How to remove Beijing ransomware. https://www.pcrisk.com/removal-guides/19222-beijing-ransomware, 2023.

[20] How to remove Makop ransomware and prevent further file encryption? https://www.pcrisk.com/removal-guides/16848-makop-ransomware, 2023.

[21] Kaspersky Anti-Ransomware Tool. https://www.kaspersky.com.cn/, 2023.

[22] Loki Locker (.Loki or .Rainman) ransomware virus - removal and decryption options. https://www.pcrisk.com/removal-guides/21572-loki-locker-ransomware, 2023.

[23] PhotoRec. https://www.cgsecurity.org/wiki/PhotoRec, 2023.

[24] Ransomware Protection Solutions. https://www.fortinet.com/solutions/enterprise-midsize-business/ransomware-protection, 2023.

[25] REvil / Sodinokibi: The Crown Prince of Ransomware. https://www.cybereason.com/blog/research/the-sodinokibi-ransomware-attack, 2023.

[26] Securing Your Amazon Web Services Cloud Environment Against Ransomware. https://aws.amazon.com/cn/campaigns/disaster-recovery-form/, 2023.

[27] The state of ransomware 2023. https://www.sophos.com/en-us/content/state-of-ransomware, 2023.

[28] The 2023 Global Ransomware Report. https://www.fortinet.com/content/dam/fortinet/assets/reports/report-2023-ransomware-global-research.pdf, 2023.

[29] The 2023 SpyCloud Ransomware Defense Report. https://spycloud.com/resource/2023-ransomware-defense-report/, 2023.

[30] WannaCry ransomware attack. https://en.wikipedia.org/wiki/WannaCry_ransomware_attack, 2023.

[31] Windows Defender. https://www.microsoft.com/en-us/windows/comprehensive-security, 2023.

[32] H. Abdi. Coefficient of variation. Encyclopedia of research design, 1(5), 2010.

[33] M. M. Ahmadian, H. R. Shahriari, and S. M. Ghaffarian. Connection-monitor & connection-breaker: A novel approach for prevention and detection of high survivable ransomwares. In 2015 12th International iranian society of cryptology conference on information security and cryptology (ISCISC), pages 79–84. IEEE, 2015.

[34] 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 2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS), pages 875–884, 2018.

[35] S. Baek, Y. Jung, D. Mohaisen, S. Lee, and D. Nyang. SSD-assisted ransomware detection and data recovery techniques. IEEE Transactions on Computers (ToC), 70(10):1762–1776, 2020.

[36] M. Belgiu and L. Drăguţ. Random forest in remote sensing: A review of applications and future directions. ISPRS journal of photogrammetry and remote sensing, 114:24–31, 2016.

[37] B. Calder, J. Wang, A. Ogus, N. Nilakantan, A. Skjolsvold, S. McKelvie, Y. Xu, S. Srivastav, J. Wu, H. Simitci, et al. Windows Azure Storage: A Highly Available Cloud Storage Service with Strong Consistency. In Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles (SOSP), pages 143–157, 2011.

[38] T. Chen and C. Guestrin. Xgboost: A scalable tree boosting system. In Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining (KDD), pages 785–794, 2016.

[39] G. Guo, H. Wang, D. Bell, Y. Bi, and K. Greer. Knn model-based approach in classification. In On The Move to Meaningful Internet Systems (CoopIS), pages 986–996, 2003.

[40] M. Hirano, R. Hodota, and R. Kobayashi. RanSAP: An open dataset of ransomware storage access patterns for training machine learning models. Forensic Science International: Digital Investigation, 40:301314, 2022.

[41] M. Hirano and R. Kobayashi. Machine learning based ransomware detection using storage access patterns obtained from live-forensic hypervisor. In 2019 sixth international conference on internet of things: Systems, Management and security (IOTSMS), pages 1–6. IEEE, 2019.

[42] M. Hirano, T. Tsuzuki, S. Ikeda, N. Taka, K. Fujiwara, and R. Kobayashi. WaybackVisor: Hypervisor-based scalable live forensic architecture for timeline analysis. In Security, Privacy, and Anonymity in Computation,

Communication, and Storage (SpaCCS), pages 219–230, 2017.

[43] J. Huang, J. Xu, X. Xing, P. Liu, and M. K. Qureshi. FlashGuard: Leveraging intrinsic flash properties to defend against encryption ransomware. In Proceedings of the 2017 ACM SIGSAC conference on computer and communications security (CCS), pages 2231–2244, 2017.

[44] G. Ke, Q. Meng, T. Finley, T. Wang, W. Chen, W. Ma, Q. Ye, and T.-Y. Liu. Lightgbm: A highly efficient gradient boosting decision tree. Advances in neural information processing systems (NIPS), 30, 2017.

[45] A. Kharaz, 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), pages 757–772, 2016.

[46] A. Kharraz and E. Kirda. Redemption: Real-time protection against ransomware at end-hosts. In Research in Attacks, Intrusions, and Defenses (RAID 2017), pages 98–119, 2017.

[47] C. Lee, D. Sim, J. Hwang, and S. Cho. F2FS: A new file system for flash storage. In 13th USENIX Conference on File and Storage Technologies (FAST), pages 273–286, 2015.

[48] Q. Li, Q. Xiang, Y. Wang, H. Song, R. Wen, W. Yao, Y. Dong, S. Zhao, S. Huang, Z. Zhu, et al. More Than Capacity: Performance-oriented Evolution of Pangu in Alibaba. In 21st USENIX Conference on File and Storage Technologies (FAST), pages 331–346, 2023.

[49] M. K. McKusick and S. Quinlan. GFS: Evolution on Fast-forward: A discussion between Kirk McKusick and Sean Quinlan about the origin and evolution of the Google File System. Queue, 7(7):10–20, 2009.

[50] S. Menard. Applied logistic regression analysis. Number 106. Sage, 2002.

[51] A. J. Myles, R. N. Feudale, Y. Liu, N. A. Woody, and S. D. Brown. An introduction to decision tree modeling. Journal of Chemometrics: A Journal of the Chemometrics Society, 18(6):275–285, 2004.

[52] L. Prokhorenkova, G. Gusev, A. Vorobev, A. V. Dorogush, and A. Gulin. CatBoost: unbiased boosting with categorical features. Advances in neural information processing systems (NIPS), 31, 2018.

[53] 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), pages 726–739, 2022.

[54] N. Scaife, H. Carter, P. Traynor, and K. R. Butler. Cryptolock (and drop it): stopping ransomware attacks on user data. In 2016 IEEE 36th international conference on distributed computing systems (ICDCS), pages 303–312, 2016.

[55] K. Shvachko, H. Kuang, S. Radia, and R. Chansler. The Hadoop Distributed File System. In 2010 IEEE 26th symposium on mass storage systems and technologies (MSST), pages 1–10, 2010.

[56] A. Wehrl. General properties of entropy. Reviews of Modern Physics, 50(2):221, 1978.

[57] C.-Y. Yang and R. Sahita. Towards a Resilient Machine Learning Classifier–a Case Study of Ransomware Detection. arXiv preprint arXiv:2003.06428, 2020.