Journal of Information Security and Applications 93 (2025) 104078

Contents lists available at ScienceDirect

Journal of Information Security and Applications

journal homepage: www.elsevier.com/locate/jisa

GuardFS: A file system for integrated detection and mitigation of Linux-based ransomware

Jan von der Assen [a],*, Chao Feng [a], Alberto Huertas Celdrán [a], Róbert Oleš [a], Gérôme Bovet [b], Burkhard Stiller [a]

[a] Communication Systems Group CSG, Department of Informatics IfI, University of Zurich UZH, CH—8050 Zürich, Switzerland
[b] Cyber-Defence Campus within armasuisse Science & Technology, CH—3602 Thun, Switzerland

ARTICLE INFO

Keywords:
Cybersecurity
Ransomware
Malware
Fingerprinting
Machine learning

ABSTRACT

Although ransomware has received broad attention in media and research, this evolving threat vector still poses a systematic threat. Related literature has explored their detection using various approaches leveraging Machine and Deep Learning. While these approaches are effective in detecting malware, they do not answer how to use this intelligence to protect against threats, raising concerns about their applicability in a hostile environment. Solutions that focus on mitigation rarely explore how to prevent and not just alert or halt its execution, especially when considering Linux-based samples. This paper presents *GuardFS*, a file system-based approach to investigate the integration of detection and mitigation of ransomware. Using a bespoke overlay file system, data is extracted before files are accessed. Models trained on this data are used by three novel defense configurations that obfuscate, delay, or track access to the file system. The experiments on *GuardFS* test the configurations in a reactive setting. The results demonstrate that although data loss cannot be completely prevented, it can be significantly reduced. Usability and performance analysis demonstrate that the defense effectiveness of the configurations relates to their impact on resource consumption and usability.

1. Introduction

Digitalization has become pervasive in today's business landscape. The technological advancements that uphold today's enterprises have brought many advantages, such as reduced operational costs, improved time-to-market strategies, and global access to suppliers and customers. However, this also made organizations vulnerable to cybersecurity breaches. As past instances have shown, the impact of successful breaches can take many forms, ranging from economic damage to the endangerment of personal health [1,2].

Within the lively threat landscape, many attack vectors are employed. One threat that has been evolving over multiple decades of proliferation is ransomware. Ransomware distributed over floppy drives has been mentioned as early as the 1980s [3]. Nevertheless, ransomware was not a widely discussed attack until the emergence of digital payment solutions in the 2010s. Nowadays, even the greater public is aware of this threat vector. In terms of numbers, ransomware has been accounted to attribute for up to 20% of all cybercrime [4]. Economically speaking, the average cost of a ransomware attack has been estimated in the range of 1 to 8 Million USD [5]. While ransomware can be seen as a business-endangering attack, there are even specific breaches where the unavailability of ransomware has led to even more severe damages. For example, in the case of the Brno University Hospital in the Czech Republic, the deployment of ransomware led to the redirection of patients and the inability to conduct urgent surgical interventions [6,7].

The constant evolution of ransomware attacks, in terms of attack behavior and sophistication, has stimulated research on how to defend against it. Nowadays, thousands of papers address different areas of the cyber kill chain of ransomware. For example, industry leaders such as IBM [8] name threat detection as one of the main pillars of defense. Here, AI-driven solutions are vital to detect the constantly changing attack behavior [8]. Indeed, numerous papers have demonstrated high accuracy in detecting ransomware, many of them achieving 99% accuracy [9–12].

Looking at the broad coverage of ransomware detection in current research, one could reason that the ransomware threat can be disregarded in light of intelligent intrusion detection systems. However, from an incident response perspective, the current state of the art presents multiple challenges. First, even though ransomware is an important threat, on a system level, ransomware is rare to be encountered since the large majority of runtime would be spent in a benign state.

* Corresponding author.
E-mail address: vonderassen@ifi.uzh.ch (J. von der Assen).

https://doi.org/10.1016/j.jisa.2025.104078

Available online 24 June 2025
2214-2126/© 2025 The Authors. Published by Elsevier Ltd. This is an open access article under the CC BY license (http://creativecommons.org/licenses/by/4.0/).

Due to this low base rate, even a high accuracy in detecting ransomware may lead to many false positives [13], which must be resolved through incident response. This is costly since a single false alert requires up to 30 min of active human investigation [14].

Secondly, ransomware defense is mainly centered around recovery [8]. As such, most systems are not resilient enough to prevent ransomware but instead focus on recovering data. In general, research in intrusion detection does not always investigate or discuss the actual implications of the defense that it enables [13]. Therefore, there is a clear opportunity to investigate the usefulness of AI-driven detection in conjunction with an integrated, reactive defense that can mitigate ransomware in a resilient way while demonstrating the portability of the models and their application in realistic settings, including novel AI-based evasion methods [15].

To address the previously mentioned challenges, the work at hand presents the following contributions:

- The design and implementation of an integrated mitigation framework for ransomware in Linux-based systems, which has seen a strong rise (*e.g.,* an increase of 75% the first half of 2022) of ransomware attacks. The emergence of these Linux-based ransomware samples targeting popular hypervisors such as ESXi may indicate a relevant attack path [16]. The proposed framework uses the file system abstraction present in the operating system as a target level to implement this functionality. Thus, a file system composed of an overlay and underlay file system is presented. The detection uses file system-related system calls to extract data on a process level. By processing a variety of features extracted during a pre-defined duration, individual processes can be classified as malicious or benign. For mitigation, several novel and baseline defense strategies, including deceptive and defensive techniques, are designed for various workloads that are differentiated by data sensitivity and latency criticality.
- The instantiation and deployment of the framework on a Raspberry Pi acting as an FTP server affected by three ransomware samples. Here, the detection system's delay and accuracy performance are assessed when presented with known and unknown malware while running a benign workload in parallel.
- To measure the effectiveness (*i.e.,* security guarantees) and efficiency (*i.e.,* the resource consumption), a pool of experiments have been performed. Here, a virtualized testbed using a fast storage underlay system is used to measure the usefulness of reactive defense from a mitigation perspective. Specifically, eight ransomware samples were retrieved and tested against seven different defense scenarios. The resulting 48 experiments assess the number of bytes lost and the resource consumption when deploying each ransomware. Several experiments are conducted to assess the usefulness of the different defense strategies when running benign workloads, which are oriented towards specific use cases (*e.g.,* server administration, sensor persistence). Based on this analysis, recommendations are made regarding selecting a defense strategy given the data criticality and latency requirements of the benign workload.

The remainder of this work is structured as follows. Section 2 reviews related work dealing with dynamic ransomware detection systems. While Section 4 presents the design of the proposed framework, Section 5 shows its implementation details. Then, Section 6 introduces the testbeds and ransomware samples used to validate the framework. Section 7 evaluates the framework detection performance and consumption of resources in the previous scenario. Furthermore, the applicability of the defense methods against different sets of requirements is analyzed. Finally, Section 8 presents conclusions of this work and outlines future areas of research in ransomware mitigation.

2. Related work

This section reviews the literature combining ransomware detection and mitigation. Neglecting the adversarial context of detection systems is a common pitfall [13]. Thus, ransomware detection systems are analyzed from a defense perspective and discussed based on their applicability. Furthermore, the review analyzes the design and scenario of related studies; since the results are established on concrete scenarios and malware samples, a direct quantitative comparison is not presented, since the underlying factors are not comparable.

Cryptolock [17] safeguards against Windows-based Ransomware by warning users when there is a possibility of encryption occurring on the system. The system assesses each process through a reputation score, which indicates the level of suspicion associated with that process over time. Additionally, the system employs a similarity hash function called *sdhash* [18] to gauge the likeness between the original version of a file and its updated version after a write operation. Since the score is computed after the file is written, data loss prevention is only realistic after the detection duration.

Another approach that leverages the notion of entropy is presented by [12]. In their approach, the ransomware detection system computes the entropy based on the file name suffix. For comparison, a backup system is used to compute and compare the entropy of user data. In an optional step, multiple Machine Learning (ML) models are available if the system load allows their application in terms of system resources. The authors stress that this computation has to be applied by categorizing data based on the types of files that are accessed. While the authors present highly promising results with respect to the detection of malware, they do not directly assess the effectiveness or efficiency in a real-world defense scenario. Furthermore, the authors only outline that file recovery is a conceptually compatible strategy.

Since many ransomware instances not only encrypt data but also communicate over the network (*e.g.,* for key exchange, operational control), *RansomSpector* [19] monitors both the file system and network traffic to provide improved accuracy. The proposed monitoring component captures the system calls via a hypervisor and checks whether the system call is related to the file system (*e.g.,* OPEN, LINK, WRITE) or network activity (*e.g.,* CONNECT, BIND). If it is, the system call is sent to the Detector, which then performs pattern matching. If both file system and network operations match the malicious pattern, the process is identified as malicious. RansomSpector has been evaluated on Windows 7 to establish the detection rate and performance consumption, not the defense effectiveness. Due to the alert-based mitigation strategy, files may not be preserved. Thus, substantial management efforts may need to be dedicated to resolving false and true positives (*i.e.,* file recovery).

Redemption [21] stands out as a newly proposed system capable of detecting and preventing ransomware. It achieves this by intercepting write operations, redirecting them to mirrored files, and preserving the original files. The implemented kernel intercepts file system operations calculates entropy ratios, utilizes File Content Overwrite, and considers access frequency to identify suspicious activity. To evaluate their system, Windows was chosen as the only target. Using this testbed, they assessed the detection performance and the resource overhead incurred by the system. Furthermore, they conducted usability experiments. While the results are promising, it is not clear whether the defense metrics were measured or whether they were inferred from the mitigation performance. Furthermore, [15] demonstrated that explicit feedback to ransomware can be used by an adaptive attacker since the solution is not deceptive.

ShieldFS [20] is a self-healing file system that adds ransomware protection capabilities to the Windows operating system. For the data collection, an I/O file system sniffer has been developed that incorporates additional information, namely entropy, process identifier (PID), and timestamp. When trained classifiers report malicious behavior, the file system discards written copies and terminates the process. To assess the effectiveness of this approach, virtual machines were used

Table 1
Categorization of surveyed related work.

Work	Mitigation	Prevent	Detect	Platform	Evaluation metrics	Data collection
[17] 2016	Alerts, Process termination	✗	✓	Windows	D | M	User data
[20] 2016	Shadow drive Buffering, Discard writes	✓	✓	Windows	D | M | P | U	IRPLogger
[21] 2017	Buffers, Termination	✓	✓	Windows	D | P | U	I/O System
[12] 2019	Backup recovery	✗	✓	Backup systems	D	✗
[19] 2020	User notification	✗	✓	KVM/Windows	D	System calls
[9] 2023	Trap directories, File renaming	✗	✓	Linux	D| P	Performance metrics
[22] 2023	MTD: Trapping, Delays	✗	✗	Linux	M | P	Performance metrics
[23] 2024	Isolation, Process termination	✗	✓	Windows sandbox	D	Windows API calls
[24] 2024	Explainable alerts	✗	✓	Windows Sandbox	D | P	Windows API calls
This	Process termination, Delaying Deceptive Modifications, Tracking	✓	✓	Linux	D | M | P | U	File system calls

D = Detection evaluation, M = Mitigation evaluation, P = Performance analysis, U = Usability impact analysis.

for experimental deployment. Aside from assessing the classification, it was assessed how well the system can recover the files. Aside from promising detection and prevention results, several limitations are present. First, the system was only assessed for the Windows platform. Furthermore, DoS attacks could be executed on the buffers, and the termination signal could be used for adaptation [15].

[9] presented an ML-based detection system using performance metrics gathered from the Linux kernel. Among other malware, ransomware was identified with high accuracy. Different Moving Target Defense (MTD) techniques can be deployed after successful detection. As shown during experiments with real malware, these techniques allow the ransomware to be terminated at some point. Nevertheless, reactive deployments lead to data loss, while proactive ones were considered wasteful in terms of resources.

Novel approaches in ransomware detection acknowledge the evolutionary nature of ransomware, hence the necessity for a detection system to keep up with the changing nature of ransomware. *FeSAD*, a recently proposed framework, was demonstrated to apply binary classification for this context successfully. While the work confirms the effectiveness of using Windows API calls and reasons about the applicability of potential mechanisms such as system isolation or task termination, no preventative measures are proposed [23].

[22] provides a virtualization of the MTD-based defense from [9], leading to multiple defense methods. First, the recursive (*i.e.,* infinite) directory tree defense is implemented in the file system, where a directory listing is extended by a specific directory. If this directory is listed, the directory itself is returned, leading to an infinite trap if one were to apply a depth-first directory traversal. Furthermore, operations can be cheaply delayed, and it is possible to obfuscate the file suffixes and the identifying magic bytes. The effectiveness is analyzed in a reactive setting, showing that multiple samples can be prevented. Importantly, this relies on the traversal strategy of the samples employed. Moreover, no assessment of the impact on benign workloads has been performed.

Arguing that existing models for ransomware detection lack explainability, [24] propose *XRan*, a Convolutional Neural Network-based model. The authors leverage multiple dynamic and static sources (*e.g.,* Windows API calls, linked libraries) to achieve local and global explainability. While the authors add an effective and explainable detection method to the growing body of literature, the effectiveness of the approach in a defense scenario is not tested, and no discussion on portability to other platforms is present.

Table 1 summarizes the previously analyzed contributions in terms of their mitigation, prevention, and detection strategies, the platform they target, how data is being collected, and the approach to evaluating the work. In conclusion, despite the advances in OS-level ransomware detection and mitigation, the following five challenges are still open:

(ch1) Only limited experience is drawn from platforms targeting Linux-based operating systems. It is not only unclear how well approaches can be ported to Linux-based systems due to the lack of experiments; assumptions of these approaches may not be aligned with common workloads (*e.g.,* the presence of a user to warn, the ability to use shadow drives).

(ch2) Ransomware mitigation approaches informed by a reactive deployment scenario terminate the encryption behavior. However, the understanding of how to stop and prevent ransomware during execution is sparsely covered.

(ch3) The usefulness of detection systems is not discussed in the context of an actual defense scenario. Thus, not only the development of novel detection methods, but their usefulness in reactive defense is to be investigated.

(ch4) As a fourth challenge, all ransomware mitigation approaches provide feedback to the attacker (*e.g.,* file recovery, process termination). As showcased in [15], this threat model may not be realistic anymore.

(ch5) Evaluating these defense approaches is based on simulation results without covering real-world ransomware samples.

In summary, there is no work applying a reactive ransomware defense system that deceptively prevents ransomware on Linux-based systems, especially when considering a broad set of real-world samples in realistic execution settings.

3. Ransomware threat model

Although the term ransomware is often used unanimously during threat modeling, there exist various flavors. The most aggressive ransomware variant is *crypto-ransomware*, which traverses the file system of the target and potentially mapped file systems (*e.g.,* NFS, Samba shares). For each of the files of interest (usually a subset of all files), an encrypted copy is produced in lieu of the original file. Thus, the availability of the original data is threatened, for at least the time it takes to restore the data from a backup. In the worst case, all data can be lost if victims are not able to retrieve the encryption key [25].

When the key cannot be retrieved by means of law enforcement, victims consider the payment of the ransom that is demanded by the attackers, although it is not clear if the key is actually released. Without actually encrypting data, *scareware* tries to trick users into believing that their data has been compromised, although it may not have actually been breached by the attackers. In that sense, one could reason that this threat vector is less impactful [26].

More similar to crypto-ransomware is *locker malware*, which tries to make data or functional assets unavailable by other means than file encryption. For example, a specific lock screen may be installed by the malware to lock the user out of the device [27].

Ransomware-as-a-service cannot necessarily be differentiated by its damage or breach function but rather describes the business model and degree of sophistication of the threat actors. Here, a ransomware strain can be bought from the attackers, which may also offer to take care of other aspects such as payment processing [28].

Finally, *extortion*-based ransomware aims to increase the chances of paying the ransom. Here, one or more of the previously described attack vectors (*e.g.,* file encryption or deletion) are combined with the leakage of sensitive data so that victims can be pressured into paying the ransom to avoid publication of the data [29].

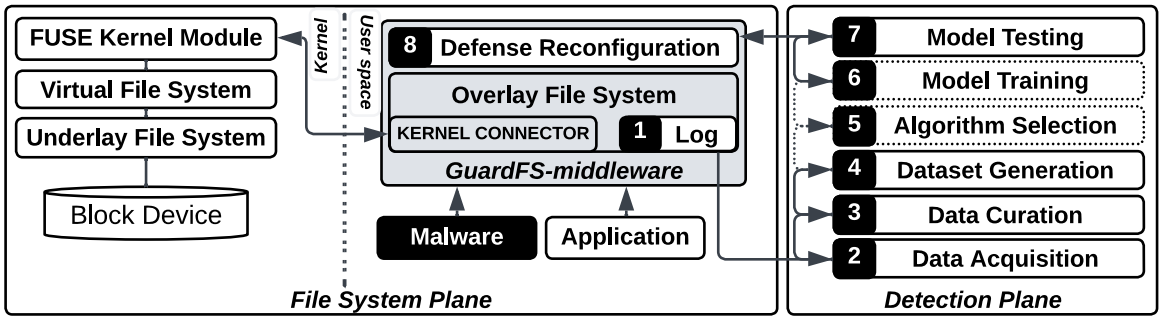


Fig. 1. File system-based framework architecture.

With all the previously defined damage methods, a wide array of infection methods are possible. For example, spear phishing is a common approach the gain initial access to the victim's infrastructure [29]. For the threat model considered in this work, the infection method is not ignored. Thus, it is assumed that the ransomware has user-space access to a Linux system and that it has sufficiently elevated privileges to be able to read, write, and delete files. Furthermore, it is assumed that the attacker can execute binaries and scripts on the host. With respect to the damage function, crypto-ransomware is considered due to its aggressive nature and its prevalence in conjunction with other behaviors.

4. Framework design

This section presents a framework for integrated detection and mitigation of ransomware in autonomous Linux-based devices by applying both aspects from a file system abstraction level. More in detail, it shows the details of the framework architecture and that aims to overcome the challenges *ch1* to *ch2* outlined in Section 2.

4.1. Architecture

To defend against the previously described ransomware attack model on Linux-based devices (*ch1*), the framework shown in Fig. 1 is proposed. The primary motivation behind the design is the evidence provided in [22], attesting to a file system-based defense. Due to its proactive defense, it could hypothetically act earlier but with less insight on the malicious behavior (*e.g.,* working directory of the process, process name). Thus, several files were lost in [22], motivating the need for a reactive framework, where additional information would be available. The framework is designed in a distributed manner, with two planes separating concerns and responsibilities. This allows computationally weaker devices (*e.g.,* resource-constrained devices, mobile devices) to implement the defense and data creation components, with an external node running all components related to the detection. However, a computationally capable device could run all components in the same execution environment. As presented in the architecture, the following two planes provide two functions.

1. *File System Plane*. It is a fully functional overlay file system in charge of servicing any file system-related system call. The behavior of the call handler depends on the defense configuration. Secondly, the file system acts as a data collector and transmitter for the detection system.
2. *Detection Plane*. It provides intelligence in the device for reactive mitigation. Input data can be received and collected from the file system plane. Due to its positioning in user space, additional data sources can be integrated.

4.1.1. File system plane

The most important task of the file system plane is to act, as the name implies, as a file system. Since it is conceptualized as an *overlay file system*, this means that it will receive system calls from an abstract representation of a file system (*i.e.,* the *virtual file system* running in the kernel). The overlay file system must handle the system call and return the appropriate response to the calling process. For example, when a hypothetical process writes to a file, the following system call is invoked in the overlay file system.

```
ssize_t write(int fd, const void buf[.count], size_t count
    );
```

Now, the overlay file system can invoke the same system call on the *underlay file system*, which can be any file system mounted on the Linux machine that operates on the underlying storage. To do so, it sends the same system call, passing the file handle `fd`, buffer `buf`, and a number of bytes written `count`. Finally, it returns the `count` back.

With this behavior, the overlay file system forwards calls to the file system, which is simply the existing file system implementation in a Linux operating system. However, the actual execution of the call would depend on the *defense reconfiguration*. This configuration describes how to intercept system calls. For example, one configuration could involve looking up the calling process by process identifier (PID) and ignoring the system call. To design a defense configuration, it is important to consider the ransomware threat model. For example, for ransomware that overwrites the target file, the `write()` system call behavior can be modified. However, other crypto-ransomware will simply create a new file and delete the target file — thus, the `unlink()` system call is considered. The actual behavior of the defense can be made more granular, depending on the output of the detection system. In summary, this enables stealthy defense approaches (*ch3*) beyond simply terminating the process (*ch2*).

The second task of the file system is to provide data to the *Detection Plane*. This is especially critical when mitigation and detection should be aligned in a way that enables not only the detection of damage done but also the prevention thereof. For example, a detection system could easily detect encrypted files based on resource consumption (*i.e.,* ongoing encryption) [9] or based on activity on the files in the normal file system. However, this would detect damages already done to the system. Thus, the (overlay) file system presents a unique opportunity, as the system calls received and the data passed in them is "the last mile" before the damage is written to disk. Nevertheless, data collection is optional, and proactive defenses could be implemented, too.

There are numerous system calls from which data can be captured. Thus, the first dimension is the system call type. Similarly, many system calls are parametrized by flags, which can be captured. Next are the file paths and file descriptors. Here, the file suffix can be an important dimension since ransomware differs by the subset of file types they target [22]. It is important to consider that only calls such as `open()` give access to the file path, whereas others use the internal

representation of a descriptor. Another important dimension is buffers, such as the ones passed by `write()` and `read()`. Processing buffers can lead to high resource consumption. One approach is to transform it into the entropy, measuring the randomness. For example, Eq. (1) computes the Shannon Entropy.

$$H = -K \sum_{i=1}^{m} p_i log(p_i) \quad (1)$$

4.1.2. Detection plane

The *Detection Plane* provides all necessary capabilities to implement ML/DL-driven reactivity to reconfigure defenses. Thus, this plane houses all data analysis-related aspects, starting from *data acquisition* to the actual *model evaluation*. As presented in Fig. 1, the model training and algorithm selection steps are only considered during the implementation (or its continuous refinement) of the proposed platform.

The main consideration in the *Data Acquisition* step is how data can be pulled from the file system. Thus, the business logic is informed by the behavior of the data producer. For example, if logs are continuously provided by the file system plane in a stream, they should be consumed in a stream-based manner. If logs are only written periodically, regular checks are performed. Logs from a file system, especially based on system calls, can contain many elements per slice. Furthermore, the size of the logs can vary greatly, with some logs potentially being empty when no process accesses the file system during the monitoring cycle.

This synchronization and preliminary buffering must be considered in the *Data Curation* step, along with the selection of features and the processing of computed features. At this stage, data from the file system can be correlated with other data. For example, for each process that was involved with file system activity, data about its resource consumption can be extracted from the process monitor. For example, [30] presents an overview of kernel events and performance metrics that can be extracted. Here, many pitfalls must be considered, as outlined by [13]. For example, spurious correlation between the features and the class it represents must be avoided. To finalize the creation of a dataset, the *Dataset Generation* step persists the data and splits it into a training and evaluation dataset.

During the instantiation of the framework, the next step is the *Algorithm Selection*, where different algorithms for the classification of the data are evaluated. The most appropriate one is selected for the *Model Training* step, where the data is fed to the algorithm, producing one or more models based on the hyperparameters defined. Finally, the *Model Testing* stage applies to the creation of the framework and to its continuous operation. For the latter, new vectors are continuously evaluated to produce a classification of the file system activity. Importantly, the output must be published to inform a closed loop between the two planes. After all, successful detection of malicious behavior is only effective if the right action is taken in a time-effective manner (*ch3*). Thus, the classification results are published, at least for results that evaluate to malicious behavior. For example, the identifiers of malicious processes can be written to a file log, queue, or socket so that the file system plane can reconfigure the defense behavior.

5. Framework implementation

With the conceptual elements in the framework introduced, this section presents a specific instantiation of the framework, consisting of an overlay file system with multiple defense configurations and an ML-based binary classifier for reactive deployment of the configurations. The purpose of the prototype is to illustrate the viability of the framework and the usefulness of reactive mitigation by using real-world samples during testing (*ch5* and *ch3*). Here, [22] stands as a motivating counter-example that provided a *proactive* mitigation approach using file system semantics.

5.1. File system plane

The file system plane is implemented as an overlay file system by leveraging bindings to the *FUSE* [31] library from the `Go` programming language [32]. Essentially, each system call destined to the subtree of the file system under the mount point can be hooked into, effectively overwriting the normal behavior. Thus, the overlay system is achieved by receiving system calls, adapting the parameters passed, and issuing a new system call. For example, in an `open()` system call, the file path of the file to open or create is passed. Here, the path is changed to reflect the path in the underlay, a new *open()* system call is executed, and the results are passed to the caller.

To gather data about malicious and benign processes, a separate thread collects data for a number of seconds before writing it to an output stream that also persists in the underlay file system. Three different intervals (*i.e.,* 1, 5, and 10 s) were evaluated for buffering. In theory, longer buffers should present richer behavioral data [33] but at the cost of delayed detection and higher memory constraints. For example, in [22], it was demonstrated that fast samples were able to encrypt at a rate of ≈14 MB/s. Hence, ten seconds is used as an upper bound since already a proactive approach could potentially lead to lower losses. The lower bound of one second had been established once the framework was fully implemented, highlighting that lower delays led to a high computational load. Based on a preliminary analysis of three ransomware samples, only the `read()` and `write()` operations and their parameters were considered for persistence. Due to the high number of system calls produced by the aggressive ransomware behavior, the following dimensions are recorded: *(i)* the process identifier (PID) is recorded to distinguish between processes. For `write()` calls, *(ii)* the Shannon Entropy of the buffers passed is calculated. Furthermore, *(iii)* a timestamp is recorded so that additional metrics can be computed. Furthermore, *(iv)* the file path, including the file name and suffix, are recorded. Detection systems, such as the one experimentally used in Section 5.6, can use the path information as a feature or, as in this case, to adapt the data pre-processing.

Besides collecting data and routing system calls, the file system plane implements four different defense configurations that can be reactively and selectively applied. This means that two processes could access (*e.g.,* write to or read from) the same file. However, only the benign one will be forwarded to the underlay, while the malicious one will be deceived or mitigated.

5.2. Defense 1: Killing processes (PKILL)

The first technique is the conceptually simplest one, which has already been explored in the literature [17,23]. As such, it should also serve as a comparison baseline for other defense configurations. Furthermore, it is implemented in the overlay file system to add the novel element of delaying any modifying system calls until a time T has expired. This duration shall allow enough time to gather data about the process, classify its behavior, and decide how to react. As presented in Fig. 2, `PKILL` invokes a process termination through the operating system based on the PID if the file system receives the signal of it being malicious. Any explicitly benign behavior is forwarded to the underlying file system while returning this response to the calling process of the system call.

5.3. Defense 2: Obfuscating responses (OBF)

While the previous defense may be successful in deterring certain types of ransomware, it inadvertently shows weaknesses. First, it does not protect against upcoming damage, and a controller could redeploy the malware. Secondly, as shown in a recent research work [15], a limitation of this type of defense is that it presents an explicit trigger to the attacker. Thus, an attacker will be able to learn precisely under which circumstances the ransomware was detected, potentially leading

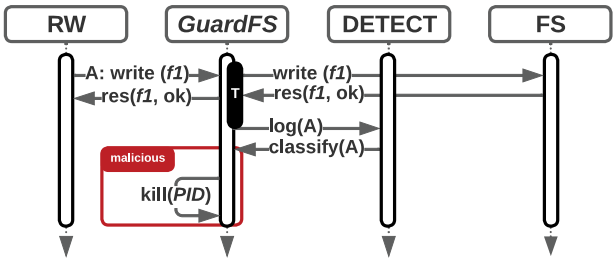


Fig. 2. PKILL defense.

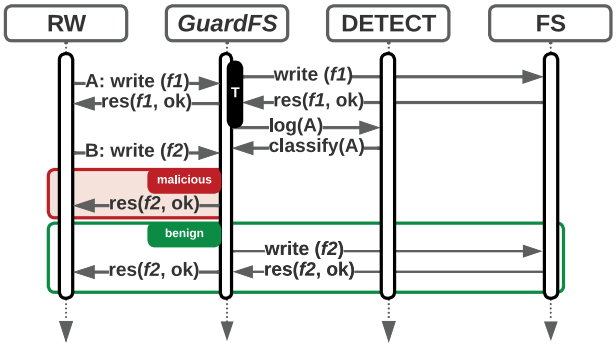


Fig. 3. OBF defense.

to an adaptation of the behavior. For example, as demonstrated in [34], an RL-based agent could learn an optimal attack policy based on connection loss (*i.e.,* being detected) and current action (*e.g.,* encryption rate, algorithm). Thus, OBF presents a defense that does not give an attacker explicit feedback while protecting against loss of data.

As shown in Fig. 3 any operations of unknown processes are immediately forwarded to the underlay systems. During each time period T, data is collected about all processes. After evaluating the current buffer of logs, benign processes are continuously given access to the underlay file system. However, for malicious processes, the PID is cached in the file system, and any damage-inflicting system calls are decepted. Thus, a `read()` system call is still granted; however, `write()`, `rename()`, and `unlink()` (*i.e.,* deleting a file or directory) are ignored in a special way. No actual system call to the underlay is executed. However, a falsified response is crafted to let the caller believe it was executed. For example, a *write()* system call passes the file descriptor and a buffer. The caller expects a single number that indicates how many of the bytes in the buffer were written. Thus, this defense configuration heuristically waits a small amount of time and then responds with the length of the input buffer, leading the caller to believe that all data was successfully written. In future iteration cycles, each behavior is newly evaluated.

5.4. Defense 3: Delaying and obfuscating responses (DEL+OBF)

The previously described defense (*i.e.,* OBF) applies response obfuscation to all file system operations that would lead to changes in the underlay system if they were issued from a process that was previously classified as malicious. DEL+OBF is designed for use cases where data is even more sensitive, but latency is not critical. For example, a data collector for a long-term heart monitor may constitute highly sensitive data. However, the latency might not be critical, especially if the sensor already transmits collected data in batches (*e.g.,* once per hour). In such a case, it only matters that the data is eventually persisted, but since there are no immediate read operations, a short delay can be tolerated. From an implementation perspective, the effect of this delay would need to consider various OS-related aspect, such as page cache, which influence the buffering between user-space and VFS.

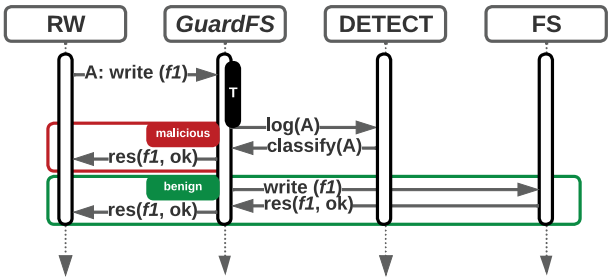


Fig. 4. DEL+OBF defense.

Thus, this technique implements the same defense for maliciously classified processes (*i.e.,* obfuscate system call responses). However, as shown in Fig. 4, all system calls of all processes are blocked for the duration of the timer T. By adhering to this timer, the situation where a system call is responded to without knowing if its payload is malicious or not does not arise. In theory, this is done at the expense of the buffering requirements represented by the following equation, which have to be accommodated in the memory of the device. Later, a process' behavior is evaluated without considering the previous classification.

$$(min(\delta, \epsilon) + min(\delta, \beta)) \times T \tag{2}$$

Intuitively, the memory requirements are calculated for the duration of the buffering. Then, the duration is multiplied by the effective load caused by untracked processes, which consists of malicious behavior (*i.e.,* encryption) and benign behavior (*i.e.,* any other write operations). The load from encryption ϵ is controlled by the ransomware and bound by the disk or file system throughput δ. Although the ransomware encryption rate may not seem useful, it is an important construct, as will be presented in the experiments, since highly sequential ransomware will be limited in terms of encryption rate due to the buffering itself since they do not implement an asynchronous (*i.e.,* non-blocking) traversal and encryption. Analogously, the throughput for all benign untracked processes is considered by the factor of β, since the buffering has to be performed for these processes, too, as indicated in Eq. (2). Thus, it is important to consider that buffering and blocking are applied to all processes since the trustworthiness is not known beforehand, and it is not considered for the full process lifecycle.

5.5. Defense 4: Tracking processes (TRACK)

In terms of design, DEL+OBF should present the highest security guarantee, while OBF presents the lowest latency of benign processes. The final configuration TRACK tries to maintain state information about long-running processes and discriminate between malicious and benign processes by looking at the calling PID.

As presented in Fig. 5, unknown processes receive the same response as all processes do in DEL+OBF – they are responded to in a blocking manner. Thus, for duration T, their activity is collected and classified by the detection system. Only then is a response created. The response behavior follows the same obfuscating behavior for malicious processes while it forwards system calls for benign processes. This state information is then tracked for subsequent calls to the file system by adding the PID to a hash table.

For known benign processes (*i.e.,* for hits in the hash table), the system calls are immediately forwarded to the underlay file system. However, that does not mean that their behavior is not monitored. In an opposing way, their behavior is tracked by a different `monitoring` thread for a time duration T, after which the classification is evaluated, and the hash tables are updated. In the case of a benign process that at some point exhibits malicious behavior, the average data loss would be represented by Eq. (3) (assuming that malicious behavior would be

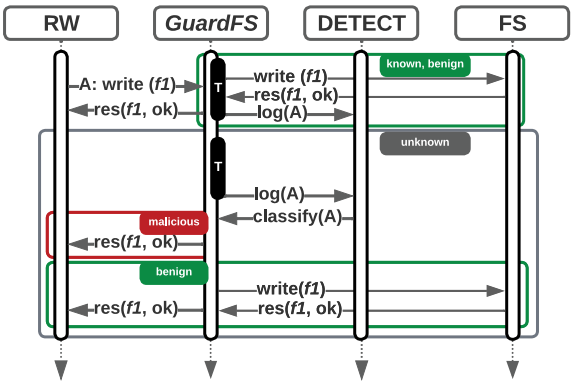


Fig. 5. TRACK defense.

detected). The factor of 0.5 is introduced to indicate an average loss, assuming a uniform distribution of data loss materialization.

$$min(\delta, \epsilon) \times T \times 0.5 \tag{3}$$

For processes that are known to be malicious, the file system will immediately invoke the obfuscating responses. This is an important factor to consider since, in this configuration, ransomware may only be blocked for one monitoring iteration. Afterward, responses are obfuscated and sent back at high throughput without blocking the calling process. In parallel, data is still continuously collected, even though the process was classified as malicious.

5.6. Detection plane

Due to the pure user-space implementation of the file system plane described in 5.1, several approaches for detecting ransomware are feasible. For example, kernel metrics, hardware counters, system calls, or performance measurements could be used. However, to implement a pure file system approach, only related system calls are used. Thus, in the monitoring thread that implements the *Detection Plane*, a dataset is curated from continuously reading the system calls and aggregating them to buckets. The bucketing is applied so that for each time slice, the metrics in Table 2 are aggregated.

Three ransomware samples were executed to create malicious and benign behavior. These represent only a minority of all obtained samples used for the subsequent experiments, since it is assumed that a defender cannot train on all existing samples, since ransomware is highly polymorphic in practice. More specifically, *RansomwarePoC*, *DarkRadiation*, and *roar* (see Section 6.3) were deployed by directly giving them shell access. *RansomwarePoC* was selected since it represents an open-source Proof-of-Concept sample, *DarkRadiation* was selected since it represents a full-fledged sample which incorporates Command-and-Control interaction, and *roar* represents a stealthy sample. To the best of the authors knowledge, *roar* is the only available sample developed for this specific purpose. For benign behavior, a workload consisting of many read-and-write operations with various file contents is needed. Otherwise, deciding between benign behavior and ransomware would be trivial. For example, simulating a desktop scenario with only a few infrequent write operations to a plain text file could be easily distinguished against an attack scenario. Thus, an FTP server was deployed on the same Raspberry Pi device (see Section 6.1), and the load was generated using the Apache JMeter [35] stress testing suite. The choice of this application scenario may appear arbitrary. However, it has been chosen since to the best of the authors' knowledge, there is no *dynamic* execution payload that yields a representative dataset. Thus, the following rationale was followed: from the file system detection plane, the application protocol does not influence the activity since there is only visibility into the file operations. However, to provide a non-trivial detection problem, various files with different levels of entropy should be used with a high amount of traffic. Hence, the choice of the application protocol does not necessarily matter as long as it enables one to easily generate a lot of activity in the file system. By configuring multiple client threads in the stress testing tool, a high load was placed on the server, consisting of reading and traversing the directory structure. The files deployed on the device used a broad set of file types provided by [36]. Another subset from the corpus was used to upload it to the device so that write operations with high entropy (*e.g.,* ZIP archives, JPG images) are also present in benign behavior. Under these conditions, normal data was collected for 258 min (≈ four hours). Data collection involving encryption by each ransomware sample and benign behavior spanned several hours. *roar* did not achieve full encryption in that time and was, therefore, terminated after roughly two hours. Due to the low encryption activity of *roar*, only 1.7% of the collected system calls are labeled as malicious in the resulting dataset. For *DarkRadiation*, 13.9% of all system calls were labeled as malicious, and for *RansomwarePoC*, 9.2% were malicious calls. For labeling, a set of benign processes are pre-defined allowing unrelated PIDs to be labeled as malicious and ones related to those benign processes to be labeled as benign [37].

Table 2
Shape of aggregated dataset.

Time	Writes	Reads	pid	e_min	e_mean	e_max
20	131	102	232	7.86	7.86	7.86
20	73	2	533	7.94	7.95	7.96

Once data was collected, the datasets that included the presence of different behaviors (*i.e.,* benign or one of the three ransomware samples) were aggregated into buckets of 2, 5, and 10 s. Furthermore, time-sensitive or leaking features (*e.g.,* file extensions, paths, time, PID) were removed. Ultimately, each row in the dataset holds for each process the number of operations per system call type (*e.g.,* read, write, rename) and the minimum, maximum, and mean entropy of the buffers while using the file path to identify the suffix. Finally, each dataset was split, where 80% was to be used for training and 20% for testing.

Two approaches were followed to implement the model training component using ML. First, an aggregated dataset was used to create one global model, where all ransomware data was labeled as malicious and the remaining data as benign. In the second approach, three models were trained for each bucket configuration. Here, only two out of three ransomware samples were included in the training data to analyze whether unknown ransomware behavior can be detected based on the behavior of other samples. Of course, only benign data was used for anomaly detection.

Thus, for both approaches and all three bucketing configurations, a model was created using three algorithms: Random Forest Classifier, Logistic Regression, and Isolation Forest [38]. These algorithms do not exhaustively demonstrate the effectiveness of ML-based detection; however, they represent popular choices and may fulfill the goal of obtaining a simple classifier to test the reactive defense system. As shown in the subsequent experiments, all classifiers achieved high accuracy in classifying the malicious behavior using the default parameters provided by *scikit-learn*, so no parameter tuning involving a validation split was used.

6. Validation scenarios

To assess the effectiveness and efficiency of the described prototype, it was deployed in multiple scenarios, ranging from single-board computing to a container-based testbed. All scenarios are inspired by a non-interactive server scenario. This section describes the configurations and related artifacts that were used in the experiments. Sources, samples, and data are available in [37].

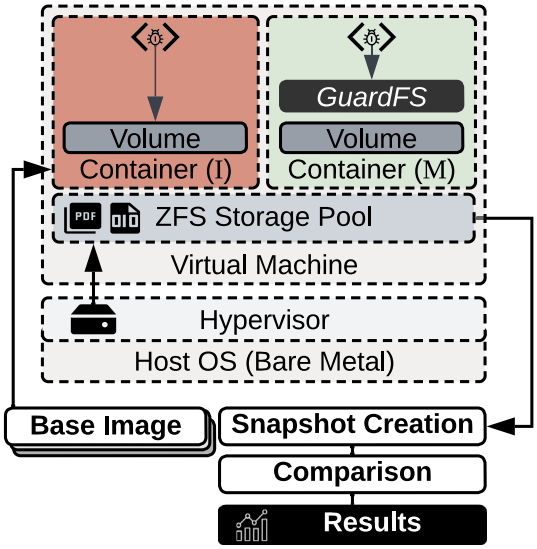


Fig. 6. Virtualized ransomware testbed architecture.

6.1. Single-board computing

Raspberry Pi devices are implemented as a system on a chip, making them a low-cost computing platform that can be used in a variety of use cases. Due to them being exposed to ransomware in the past, they fit the threat model described in this work. Furthermore, since they are considered resource-constrained devices (either in terms of computational resources or management capabilities), they present an excellent test bed that allows one to experiment with the effectiveness of the platform. Specifically, a Raspberry Pi 4B with 2 GB of memory was used to run an FTP server at high load. On this device, experiments are performed to assess the capabilities of the detection system both in an online and offline experiment (*i.e.,* with data gathered on the device, but evaluated locally).

6.2. Virtualized testbed

While the deployment of real-world malware in a real device allows experimentation close to reality, it is tedious to gather data in a reproducible and scalable way so that numerous samples can be tested. Thus, the testbed shown in Fig. 6 is developed, where the hardware presents high-performance access to storage, by means of two NVMe storage devices and an AMD Ryzen 5700G processor running at 4.7 GHz with 64 GB of DDR4 memory. Experiments are executed in containers where a ZFS dataset is mounted. Optionally, a configuration of *GuardFS* can be mounted, too. Since the underlay resides on the ZFS dataset, snapshots can be easily created and compared to understand the effects.

6.3. Malware samples

Aside from *RansomwarePoC*, *DarkRadiation*, and *roar*, which have already been used during the implementation of the framework (*i.e.,* for the model creation step), a broad set of malware was obtained from malware databases and integrated into the testbed. As of the author's knowledge, there is no Linux-based testbed with a higher number of samples to perform dynamic analysis (*i.e.,* non-simulated experiments).

- *Babuk* is sophisticated malware, whose source code was leaked. The ransomware is written in `golang`, targeting different platforms. Thus, the encryption module was extracted and compiled for the x86 platform [39].
- *Blackbasta* is a renowned ransomware-as-a-service enterprise. A leaked binary was obtained as an ELF file. After reverse-engineering it was discovered that it targets specific folder paths used by *VMWare*. Thus, *GuardFS* is mounted on `/vmfs/volumes` [40].
- *Cl0p* is a strain of a famous group of attackers. A binary was fetched from MalwareBazaar [40].
- *Conti* operates as a service since 2020. A binary is available in MalwareBazaar [40].
- *DarkRadiation* is a ransomware that targets Linux-based systems. This sophisticated ransomware is implemented entirely in bash using Telegram for communication instead of a dedicated C&C server [41].
- *GoCry* is an educational, open-source ransomware written in `golang` [42].
- *javaRansomware* is an educational, portable, open-source ransomware developed in `Java` [43].
- *lockbit* is another strain of the real-world ransomware that was operational in 2023 [40].
- *lollocker* is an open-source ransomware strain using `bash` to orchestrate the encryption using `OpenSSL` [44].
- *Monti* is a modified strain of existing real-world ransomware [45], which can be retrieved as an ELF-file on MalwareBazaar [40].
- *Ransomware-PoC* is a proof of concept open-source Python ransomware payload [46].
- *roar* appears the most technically elaborate ransomware. Technically, it is an open-source adaption of *RansomwarePoC*, aiming at diversifying the encryption behavior (*e.g.,* encryption algorithm, encryption speed) and optimizing the most stealthy operation by using Reinforcement Learning [15].

Thus, the selection of samples includes both educational, open-source samples (*i.e., javaRansomware*, *Ransomware-PoC*, *roar*), with the remaining attributed to real-world incidents, where samples are obtained from a malware databse [40].

7. Experiments

This section presents a pool of experiments that evaluates the performance of the proposed framework while detecting and classifying the ransomware families introduced in Section 6. Although the detection methods do not constitute a key contribution to the field, understanding their performance is relevant to contextualizing the results of the reactive defense. Thus, the detection plane is evaluated against the testing datasets to understand how well it detects ransomware samples in two cases. First, when the given ransomware sample is present in the training data, and second, when it is considered an unseen behavior. Next, the experiments move to a production scenario, considering the Single-board computing scenario, where the detection delay is contextualized in terms of files encrypted until detection. In the end-to-end experiments, the detection plane is integrated with the defense techniques. Here, the previously described virtualized testbed is employed so that the seven different configurations of the defense strategies are confronted with eight ransomware samples to compute the amount of data that is lost. Finally, the overhead of the detection and defense components on benign workloads is established, leading to a comparison with related approaches. As such, it is assumed that an operational model is available; in-depth discussions on the effect of misclassifications (*i.e.,* false negatives on malicious behavior and false positives on benign behavior) within varied scenarios represent a limitation of the work, as outlined in Section 7.4.1.

7.1. Evaluating test datasets

To assess the performance of the detection plane in isolation, data obtained in the Single-Board computing scenario is leveraged, considering multiple samples. The actual evaluation is carried out on the remote device that was used for training the models.

Table 3
Accuracy for unseen Ransomware — random forest classifier.

Time window	RansomwarePoC	DarkRadiation	Roar
2 s	99.92%	94.71%	99.43%
5 s	99.91%	95.55%	99.65%
10 s	99.90%	96.92%	100%

7.1.1. Classifying unseen ransomware

To understand how well the file system behavior classification performs for unseen malware samples, the datasets were combined into nine different combinations to train one model based on the Ransom Forest Classifier. Thus, one model was trained for each combination of the three different time windows (*i.e.,* 2 s, 5 s, and 10 s) and combining two out of three ransomware samples (*i.e.,* leaving out either RansomwarePoC, DarkRadiation, and roar). Two second time windows have been defined since they present the lower bound at which the monitoring and evaluation cycle operates stable. The upper bound has been set at 10 s, since larger windows would likely lead to large data losses (*e.g.,* 20 s of uninterrupted encryption). The three samples (see Section 5.6) were chosen as their approach, and technical implementation provide a diverse sample set.

Table 3 shows the accuracy for the different combinations of ransomware data and time window sizes. The random forest classifier performed best when aggregating the behavioral data into 5- or 10-s slices. In the 2-s time window, the accuracy is the lowest, although just a slight difference compared to other others (*e.g.,* 2.21% difference in the worst case). This can be explained by the fact that, when aggregating into the 2-s window, not enough file system operations may be gathered at all times since ransomware samples cannot constantly encrypt at full speed, since they, like every other userspace process, may be interrupted by another process or blocked by I/O operations.

Even in the 5-s time period, the accuracy of detecting roar is lower compared to the 10-s time window; evaluating the 5-s time window multiple times over 10 s would likely provide comparable accuracy to the model with a longer window size.

For both RansomwarePoC and DarkRadiation, the accuracy in the 10-s window size is close to 100%. Thus, using these models, it is possible to detect the two strains, even when data was collected from other samples. For DarkRadiation, the accuracy increases with increasing window size, with the highest accuracy observed at 96.92%. The fact that DarkRadiation is detected with a lower accuracy is surprising since it encrypts without trying to be stealthy and in fact, encrypts at the highest speed. The result can be explained by the fact that DarkRadiation is the only sample that uses pools of subprocesses to parallelize the encryption process. Thus, grouping data based on PID leads to multiple vectors for each time window for DarkRadiation, so certain features may not reflect the malicious behavior as robust as others, as it leads to different patterns in the dataset, leading to difficulties during the classification stage. Another contributing factor is that the average entropy of RansomwarePoC and Roar is 6, whereas, in the case of DarkRadiation, it is around 8 due to differences in encryption behavior and implementation. Nevertheless, the random forest classifier is relatively robust to these differences, as the accuracy is above 94% for all time windows considered.

7.1.2. Classifying known ransomware

Even in the previously described setup with partial training data, most vectors can be accurately classified for all window sizes. However, more data is available in practice, given the breadth of available ransomware samples. Thus, assuming that the behavior can be generalized over multiple samples, a dataset is created using training data from all samples. Again, models are trained by aggregating the data into three window sizes. Furthermore, we compare two algorithms (*i.e.,* Random Forest and Logistic Regression) for classification. After concatenating all Ransomware and benign datasets together and splitting the data into 80% train and 20% test datasets, the respective models were trained. Both logistic regression and random forest classifier have shown high accuracy — close to 100%, as shown in Table 4.

Table 4
Accuracy of models trained on three samples.

Algorithm	2 s	5 s	10 s
Logistic regression	99.53%	99.76%	99.87%
Random forest classifier	99.93%	99.97%	99.98%

Table 5
Performance of RF model (5 s).

F_1	TPR	FNR	FPR	TNR
99.9967%	99.887%	0.113%	0.8313%	99.1687%

Table 6
Detection delay (5 s model).

Sample	Min.	Avg.	Max.	Data loss
RansomwarePoC	4 s	6 s	10 s	18 files — 12.3 kB
DarkRadiation	4 s	8 s	10 s	35 files — 23.9 kB
Roar	8 s	19 s	44 s	3 files – 2.1 kB

In summary, the Random Forest (RF) Classifier for a window size of 10 s presents the best results. For practical reasons, the 5-s variant may provide comparable performance while providing faster detection. This model is used for the subsequent experiments involving the execution of ransomware. To contextualize the model's performance, where false positives and false negatives both influence data loss, the confusion matrix is illustrated in Table 5. Specifically, the true positive rate (TPR), false negative rate (FNR), false positive rate (FPR), and true negative rate (TNR) are presented and the F_1 score is computed.

7.2. Evaluating after deployment

To understand how effective and efficient the detection plane can be when running in a real device, the Single-Board Computing (see Section 6) scenario was instantiated. The detection system and the FTP workload run in parallel for each sample. Then, the delay between malware deployment and detection is measured. Furthermore, it is computed how many files were successfully encrypted by the sample in that time. This contrast is especially important in light of stealthy malware samples, such as roar, that decrease the encryption speed in favor of appearing less aggressive. The results, shown in Table 6, reflect that intelligent ransomware such as roar is, in fact, able to evade detection better than other strains. Overall, the maximum detection delay observed across ten iterations of the experiment was 44 s, while the minimum was 4 s. However, putting this into the perspective of the encryption speed, stealthy ransomware such as roar cannot encrypt as many files as the other samples since it uses periodic phases of hibernation, which explains the variations of detection delay.

Most samples can be detected based on a few seconds of *active* encryption. While the number of files lost appears daunting, it has to be emphasized that in this scenario, no defense mechanism is present. As will be shown by the subsequent experiments, the file system defense can save some of the data that the ransomware is encrypting until the classification is positive.

7.3. End-to-end experiment

So far, the detection plane was developed using a small number of ransomware samples, leveraging strains that differ in implementation and purpose. As such, the previous experiments demonstrated detection performance in offline and online settings. However, measuring the usefulness of an AI-based detection system for mitigating cyberattacks must include the complexities of the defense behavior. For example, active mitigation advertently changes the device's behavior and, ideally,

Table 7
Data lost (in kilobytes) per configuration and Ransomware.

Scenario	Delay	*Babuk*	*Blackbasta*	*Cl0p*	conti	*GoCry*	*javaRansomware*	*lollocker*	*Monti*	**Average**
NO DEFENSE	0	10,655,744	10,655,744	10,655,744	561,152	10,655,744	2,634,752	2,926,592	10,655,744	7,425,152
PKILL	0	107,752	9,168	668,159	49,255	0	39,762	620,125	28,889	190,388
OBF	0	77,172	9168	70,644	96,371	0	48,500	137,304	88,259	65,931
DEL+OBF	1 s	1,286	8,266	19,598	69	0	0	1,692	69	3873
DEL+OBF	5 s	1,289	4,313	0	0	0	0	19	0	703
DEL+OBF	10 s	1,286	4,313	387	19	0	0	19	10	754
TRACK+OBF	<5 s	27,025	8,781	0	42,427	0	12,149	29,777	19,489	17,456

even the malicious behavior, as the goal is to interrupt, diminish, or prevent the behavior. Thus, if such a system considers only detection without mitigation, the actual performance can only be approximated.

Thus, a series of experiments are executed using the virtualized testbed presented in Section 6.2, spanning all malware samples from Section 6.3. Concerning the detection plane, the same Random Forest-based model from the online test is deployed in the testbed. Then, in each round, one workload is considered to assess *(i)* the defense effectiveness as established by the number of bytes lost when ransomware is deployed, *(ii)* the resources consumed by the ransomware and the defense platform, and *(iii)* the impact of the defense platform on benign workloads.

7.3.1. Defense effectiveness

One experiment per defense configuration and ransomware sample have been performed. To compare the performance of the malware, each sample is also deployed against a baseline strategy, where no defense is active. At the beginning of each experiment, the detection plane is executed, which monitors access to the file system in the background. If a process is classified as ransomware, the defense strategy, which is the subject of the experiment, is deployed. The ransomware is given enough privileges to directly execute any operations on the files in the home directory. All files are available through the overlay file system to account for any encrypted files in the experiments.

Algorithm 1 Pessimistic Computation of Data Loss

Require: BASELINE_FILES $\neq nil$
Require: SNAPSHOT_FILES $\neq nil$
  SNAPSHOT_CHECKSUMS $\leftarrow$ []
  FILES_MODIFIED $\leftarrow$ []
  BYTES_LOST $\leftarrow 0$
  PTR $\leftarrow 0$
  while PTR $<$ SNAPSHOT_FILES.length() **do**
    F $\leftarrow$ SNAPSHOT_FILES[PTR]
    SNAPSHOT_CHECKSUM $\overset{+}{\leftarrow$ sha256sum(F)
    PTR $\leftarrow$ PTR+1
  end while
  while PTR $\leq$ BASELINE_FILES.length() **do**
    F $\leftarrow$ BASELINE_FILES[PTR]
    CHECKSUM $\leftarrow$ sha256sum(F)
    if CHECKSUM is not in SNAPSHOT_CHECKSUMS **then**
      FILES_MODIFIED $\overset{+}{\leftarrow$ F
      BYTES_LOST $\leftarrow$ BYTES_LOST + lookup_size(F)
    end if
    PTR $\leftarrow$ PTR+1
  end while

After a maximum of five minutes after the sample has entered the encryption phase, the experiment is concluded, and the snapshot of the underlay is created. To assess the damage done by the sample, Algorithm 1 computes the number of bytes lost. First, a list of checksums is computed based on the file contents in the snapshot after the experiment. Then, the same is done for the files in the initial dataset. Finally, for each file in the initial dataset, it is checked whether the checksum is contained in the post-experiment checksums. If not, the file size of the original file is assessed and added to the final result number.

In that sense, the amount of data loss is computed on a pessimistic approach. For example, if only a single bit of the file is modified, the whole file is considered lost since no assumptions on the type of data are made. This also presents the danger that some malware samples may appear stronger than they realistically are. For example, some samples could delete many files without encrypting them, which would be computationally cheap. Nevertheless, it is assumed that this computation of the file modifications presents a fair approximation of data loss. It is key to highlight that only *data loss* is quantified — other impacts, such as loss of confidentiality from data access, are out of scope.

As evident from the first row in Table 7 where no security mechanisms are present, most samples achieve full encryption of the ≈10 GB of data in the system. Still, there are differences in terms of data loss since certain samples focus on a dedicated set of file types. On average, 7.43 GB of data is encrypted or lost when the samples are not interrupted. The second baseline measurement (*i.e.,* a defense strategy that has already been explored in research and thus implemented for comparative purposes) is the PKILL defense, which enacts process termination upon detection. The first observation for this defense strategy is that even without a novel defense mechanism, reactive detection can lead to the large majority of data being protected, as 97% of data remains unmodified compared to the uninterrupted case. Nevertheless, some malware samples can still destroy multiple hundred Megabytes of data until the process is positively classified and terminated. For example, *Blackbasta* encrypts roughly 9 KB of data until mitigation. In these cases, it is likely that this defense approach can be truly autonomous, and some degree of administrative intervention (*e.g.,* decommissioning the device, restoring backups) is needed. This shows that by itself, the delay for detection should be further optimized to save more data. Furthermore, there is a clear termination signal that the ransomware could leverage for self-adaptation, motivating the need for additional measures.

Next, OBF presents a different mitigation approach, which also operates in a non-blocking manner (*i.e.,* the behavior until the first monitoring cycle does not face interference). Nevertheless, this defense configuration can reduce the data loss by ≈65% compared to PKILL. This may indicate that the obfuscating defense is more suppressive against the malicious sample. Furthermore, looking into the modification times in the snapshot, it is revealed that after the detection of the ransomware, the ransomware continues to execute without any data loss. This indicates that the defense is indeed stealthy (*i.e.,* the attacker does not receive an immediate signal that it is being mitigated), which could prevent sophisticated ransomware such as roar from improving.

To improve the damage dealt until the detection system raises the alarm and deploys the defense, the three variants of DEL+OBF (*i.e.,* ones blocking for 1, 5, and 10 s) all show another strong improvement compared to either killing the process or just obfuscating file system responses. First, based on a 1-s timer, only 3.87 MB of data is lost on average for the whole experiment – ≈98% less than in the PKILL defense. The main reason this residual data is lost is that the initial delay is not long enough for the detection system. This also explains why the 5 and 10-s timers can save an additional ≈81.85% of data, comparing the average data loss to the 1-s timer. Naturally, some data is lost, as the detection system does not perform perfectly for all samples. Furthermore, the encryption windows may not be perfectly aligned

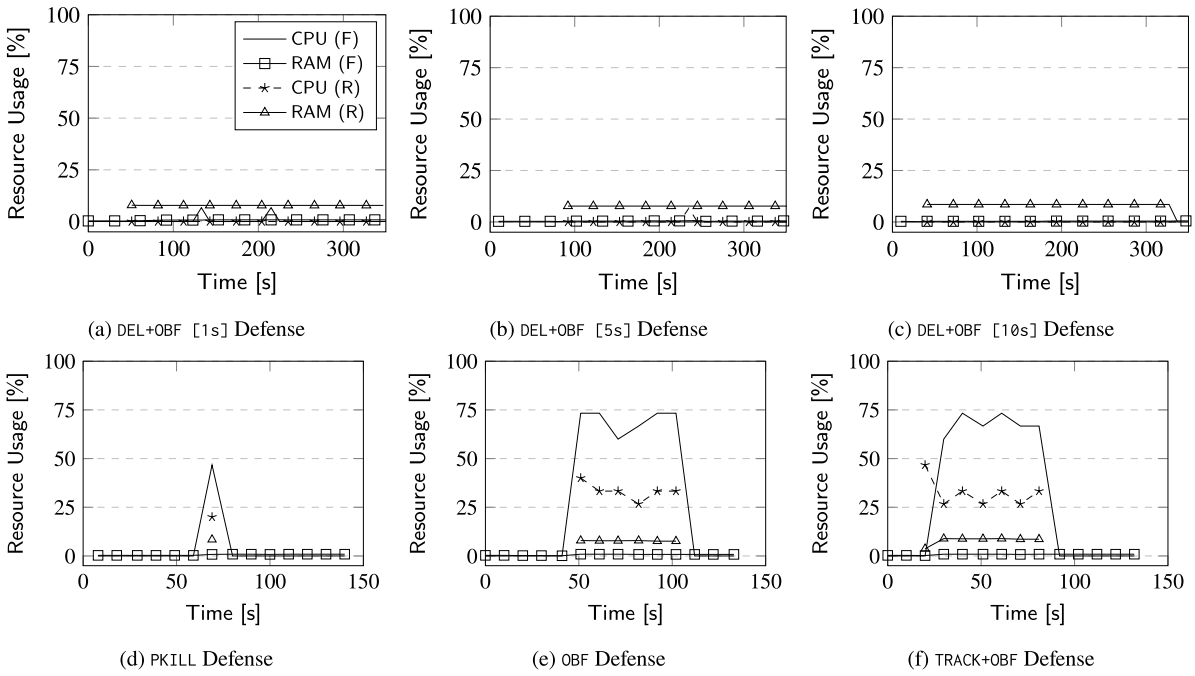


Fig. 7. Performance analysis of different defense methods and configurations against *javaRansomware*.

with the monitoring windows (*e.g.,* the first 5-s window may contain the first 100 ms of encryption towards the end). Interestingly, using a larger time window does not improve the defense.

Although the strategies that incorporate both obfuscation after an initial delay clearly present the most robust defense guarantees, they do so at the cost of an increased delay. Thus, application scenarios where persistence delay is critical would suffer from this strategy. To present a hybrid solution, TRACK+OBF applies only an initial delay for new processes. Thus, delayed critical applications would only suffer from a single performance hit, and subsequent operations can be services like benign applications. Of course, this comes at the cost that the defense is weakened. If ransomware first exhibits a benign behavior and then turns to encryption, this behavior is only detected with the delay of the monitoring and detection cycle. As shown in the last row, this is the case, as ≈17.46 MB of data are lost on average. Thus, it outperforms the remaining non-blocking defense mechanisms while underperforming against the ones delaying the execution for improved detection. Furthermore, it does so at the cost of increased complexity since the state must be maintained longer than the monitoring cycle.

In summary, the proposed defense methods provide an improved defense system. For workloads that require high-security guarantees while being able to sacrifice delay requirements, the DEL+OBF defense for a 5-s timer is the best choice. If the application should still perform with low delay, the TRACK+OBF and OBF strategies could be considered, as indicated in Table 7. The former presents stronger security guarantees but at increased complexity for managing the state and thus increased resource requirements.

7.3.2. Resource efficiency

Although all defense mechanisms present a certain improvement in resilience against ransomware attacks, they lead to different behaviors from the offensive (*i.e.,* blocking or non-blocking encryption) and defensive (*i.e.,* stateless or stateful) perspective. In Fig. 7, the analysis of executing the *javaRansomware* sample against the different defense configurations regarding resource consumption is visualized. For each experiment, the relative CPU consumption and the reserved system memory are visualized. These two metrics are assessed for the process (and subprocesses) spawned by the ransomware and for the file system, which includes the defense and detection planes running in the background.

The first observation is that, due to the delaying aspect of the DEL+OBF defense family, the overall resource consumption is smoothed, resulting in less bursty resource consumption. However, it is important not to confuse “less bursty” with “fewer resources consumed” – overall, the same resources are consumed since the same workload is performed. Still, the ransomware iterates through all the buffers representing the protected files, encrypts them, and attempts to write them to disk. In that sense, delaying and obfuscating the modifying operations (*i.e.,* rename, delete, write), not only mitigates the modifications but also slows down the ransomware.

Since the OBF and TRACK+OBF defenses do not extensively delay the ransomware execution but instead try to mimic the file system in its normal state, there is substantially bursty resource consumption. Here, it can be seen that keeping up with computationally intensive ransomware leads to significant resource consumption by the file system, too. However, assuming that ransomware attacks are still rare, it could be argued that this resource consumption can be afforded. Interestingly, the execution of the ransomware does not change once the file system mitigation goes into action, emphasizing that the ransomware may not detect the mitigation. This is magnified by the fact that the ransomware stops the encryption process by itself once the target data is consumed, encrypted, and disk persistence is attempted.

If the DEL+OBF strategies are compared to the PKILL defense, it is clear that these novel defense mechanisms are not more resource-effective. Indeed, PKILL is the most resource-effective approach since only a few monitoring cycles exhibit ransomware behavior. However, this comes at a cost, where it is assumed that *(i)* the ransomware does not alter its behavior based on the KILL signal and *(ii)* that it is actually possible to prevent further execution of the sample. Here, the defense methods introduced in this paper present the advantage of being stealthy while avoiding the damage function of the ransomware, even if the sample may not be removable.

In summary, the PKILL defense is the most resource-effective one, although at the expense of weak security (*i.e.,* assuming killing the process mitigates the malware). The other strategies require roughly the same amount of resources, although the ones involving a delay component smooth the consumption over time while providing a slightly stronger defense.

Table 8
Overhead on Benign workloads.

Scenario	Time [s]	CPU [%]		RAM [MB]	
		System	GuardFS	System	GuardFS
WL1	3	19.7	–	2.6	–
WL1+TRACK+OBF	8	13.4	56.5	1.8	10.8
WL2	4	1	–	4.8	–
WL2+TRACK+OBF	6	1	1	4.9	14.1
WL3	16	51.1	–	13.43	–
WL3+TRACK+OBF	20	33.33	1	1.9	12.1
WL4	23	9.8	–	10.5	–
WL4+TRACK+OBF	24	2.24	50.1	2.25	7.28

7.3.3. Usability in benign workloads

Although efficient resource usage when mitigating ransomware is important, efficient operation in benign settings is also critical since devices likely spend much more time in a non-infected state. Thus, efficiency optimizations when running benign workloads are an important pillar to ensure the overall efficiency of the solution. Previous experiments demonstrated that the TRACK+OBF defense presents strong defense effectiveness. Due to its design optimizations concerning efficiency, this defense mechanism is deployed in this scenario while confronting it against several additional workloads. The following workloads are then measured concerning their execution time and the resources required by the workload and the defense system. For each workload, these values are established when running them in the overlay system without any monitoring, detection, or mitigation system running and when deploying them with the aforementioned defense strategy (see Table 8):

- In the first workload WL1, the effect of the system on system administration is considered. Thus, the packages for *apache2*, *BIND9*, and *MariaDB* servers are downloaded and subsequently installed into the overlay file system. This task involves many (concurrent) read and write operations involving high entropy data.
- For WL2, a reading from a real air quality sensor is retrieved and stored on the disk. Thus, this workload shows low read operations, a single longrunning write operation with low delay requirements.
- The third workload WL3, involves the creation of a backup — here, an archive of 977 files, protected by the file system, is created. The resulting archive is stored as a GZIP-compressed tarball on the same file system.
- In WL4, a long-running, write-intensive task is carried out, which involves continuous download of the *libreoffice* suite into the file system. Here, it is also investigated how the system performs when there are no other bottlenecks, leading to highly asynchronous operation. Thus, the data is downloaded over a local network link provisioned at 1 Gbit/s. In that sense, writing network data at such a high throughput also resembles a stress test for the file system.

Looking at the *Time* column, it can be seen that the defense and monitoring system does lead to a delay in execution time. This comes as expected, as this defense configuration involves an initial delay for each process (and any spawned child processes) to classify the first interaction with the file system. This efficiency is least optimal for short-running processes that involve the creation of many files by numerous parallel processes, as represented by WL1 in the first row. This effect becomes less prominent for workloads like WL4 that are long-running and involve continuous write operations. Furthermore, for workloads such as WL2, that are not delay-critical and operating in a highly asynchronous manner (*i.e.,* spending a lot of time waiting for other I/O tasks such as network requests), this constraint may likely not play a difference. In such cases, the added defense effectiveness could provide a credible tradeoff between usability and security. Another observation is that even for creating high entropy data, such as in WL3, no false alerts (and thus mitigation) were raised during regular operation. Regarding resource consumption, it can be seen that for short-running, bursty workloads involving many system calls, the file system requires substantial compute resources and static memory consumption in the magnitude of roughly 10 MB.

7.4. Comparison with related work

In the proactive approach proposed in [22], a file-system-based ransomware mitigation is described, enabling a comparison of reactive and proactive paradigms in autonomous ransomware defense. As highlighted by the experiments that assess the overhead in benign settings, the reactive approach shown in this paper does present one issue. To be able to react to attacks, data must be monitored continuously. In this paper, file system-related system call parameters were considered. This makes detection robust. However, processing system calls becomes more expensive as more system calls are created. In that sense, the higher the load on a system, the more resources are required to assess the system calls. This monitoring cost is avoided in a proactive defense since the defense techniques are deployed beforehand.

To make proactive defense viable, deploying such a defense strategy imposes constraints on how the defense can be designed. After all, the defense strategy is constantly executing, which leads to overhead created by the mitigation. In [9], the proactive defense was found too expensive, especially for ransomware, since deception was achieved by creating a set of actual files to trap the ransomware encryption. Thus, only a lightweight defense mechanism was considered suitable for proactive deployment. In that sense, [22] presents a lightweight defense mechanism that can be deployed without intelligence. However, implementing it relies on specific assumptions of adversarial behavior, thus weakening the defense's effectiveness. For example, one of the techniques relies on the ransomware performing a full traversal of the target file system before attempting any encryption. Although this was demonstrated to be effective, it could potentially be circumvented by changing the traversal strategy or by performing traversal in parallel to the encryption. Indeed, the defense mechanisms presented here also rely on the attacker performing encryption (and thus creating a specific pattern of read and write operation, as well as creating high entropy data as output). However, as discussed in the history of ransomware, crypto-ransomware is still a highly relevant threat vector and has been so for a long time. This is backed by the results on the defense effectiveness, which show that the proactive approach led to ≈300 MB being lost, while the experiments on this reactive deployment showed losses between ≈66 MB for the worst case and ≈0.7 MB for the best strategy. In summary, reactive mitigation can provide optimized defense, although resources for continuous monitoring, processing, and classification are needed.

7.4.1. Limitations

Based on the results obtained using the methodology of this study, the following limitations of the work are explicitly discussed. First, it must be acknowledged that the assumption of the threat model is that an attack only has black-box access to the system. It is assumed that the attack can only learn from the actions taken on-device, *i.e.,* interpret the obfuscated answers. Nevertheless, other channels could be leveraged by malware to understand the success of their actions.

Secondly, a productive implementation would need to continuously recreate datasets to account for the influence of the defense systems. As part of this, special considerations would need to be given to the case where a long-running malicious process is misclassified, leading to successful attack execution. Similarly, a misclassified benign process would be blocked from writing benign data. In the current manuscript,

these cases are only investigated experimentally (*i.e.,* assessing overhead on benign workloads and measuring data loss from ransomware execution). Further studies could address this by collecting additional data (*e.g.,* combining configurations of benign and malicious executions and evaluating model performance regarding misclassification).

8. Conclusion

This work presented the design and prototypical implementation of an integrated defense platform that leverages the file system to provide autonomous and fully automated mitigation against ransomware, assuming that a detection system is in place. The detection plane relies on system call data that can be intercepted on the file system level of the operating system. With this data, an ML-based binary classifier can deploy different strategies in the defense plane. Then, multiple novel mechanisms involving stealthy defense have been proposed. Finally, a set of experiments has shown the performance of the detection plane in offline and online tests, while the detection and mitigation capabilities were assessed against several ransomware samples in a real scenario and using a virtualized testbed. Here, the defense effectiveness, resource consumption, and side effects on benign workloads were studied, leading to a comparison of proactive defense solutions with a data-driven, reactive defense.

In conclusion, this work demonstrated that ML-based reactivity can optimize the defense capabilities of a defense system. Depending on the security requirements, the defense strategy with the highest robustness could provide an almost completely automated defense system, with no manual intervention required after successful detection. It is critical to select the appropriate defense configuration depending on the type of workload to be performed in the benign setting. For highly delay-critical, and short-lived processes, the most complex defense method `TRACK+OBF` is suitable. In contrast, ones that can sacrifice delay for improved security benefit from the `DEL+OBF` mechanisms. These two also show different effects on resource consumption, with the second one being less bursty. Finally, from the detection plane, it can be concluded that the ML-based classifier presents robust detection for a myriad of different ransomware samples, even when behavioral data was gathered for different (and thus unseen) samples. Still, monitoring system-call data comes at a cost, especially at high load, where the number of system calls to be processed increases.

Based on the experiences drawn, multiple avenues for further research are identified. First, it will be investigated how the detection plane could be made more lightweight for scenarios where data processing cannot be offloaded. Here, different data sources, such as performance metrics will be analyzed. Furthermore, the platform will be tested using other benign workloads (*e.g.,* office usage, IoT scenarios) and additional ransomware samples, especially ones aiming to be stealthy. Here, the portability of the platform to other operating systems will be investigated.

CRediT authorship contribution statement

Jan von der Assen: Writing – review & editing, Writing – original draft, Visualization, Validation, Supervision, Methodology, Conceptualization. **Chao Feng:** Writing – review & editing, Conceptualization. **Alberto Huertas Celdrán:** Writing – review & editing, Project administration. **Róbert Oleš:** Writing – review & editing, Visualization, Software, Methodology, Data curation. **Gérôme Bovet:** Writing – review & editing, Supervision, Resources, Project administration. **Burkhard Stiller:** Writing – review & editing, Supervision, Resources, Project administration, Funding acquisition.

Declaration of competing interest

The authors declare the following financial interests/personal relationships which may be considered as potential competing interests: Jan von der Assen, Chao Feng, Alberto Huertas, Burkhard Stiller reports financial support was provided by Cyber-Defence Campus. If there are other authors, they declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This work has been partially supported by *(a)* the Swiss Federal Office for Defense Procurement (armasuisse) with the CyberMind project (CYD-C-2020003) and *(b)* the University of Zürich UZH.

Data availability

Included in the manuscript.

References

[1] Cybersecurity Agency IS. Shifting the balance of cybersecurity risk: Principles and approaches for security-by-design and -default. 2023, https://www.cisa.gov/sites/default/files/2023-04/principles_approaches_for_security-by-design-default_508_0.pdf [Last Visit December 2023].

[2] Von Der Assen J, Franco MF, Killer C, Scheid EJ, Stiller B. CoReTM: An approach enabling cross-functional collaborative threat modeling. In: 2022 IEEE international conference on cyber security and resilience. CSR, 2022, p. 189–96.

[3] CrowdStrike Holdings Inc. A brief history of ransomware. 2022, https://www.crowdstrike.com/cybersecurity-101/ransomware/history-of-ransomware/, [Last Visit January 2024].

[4] Isabella H. 10 common types of malware attacks and how to prevent them. 2023, https://www.techtarget.com/searchsecurity/tip/10-common-types-of-malware-attacks-and-how-to-prevent-them, [Last Visit January 2024].

[5] IBM. Cost of a data breach 2022. 2023, https://www.ibm.com/reports/data-breach, [Last Visit January 2024].

[6] Botek A. Brno university hospital ransomware attack (2020). 2020, https://cyberlaw.ccdcoe.org/wiki/Brno_University_Hospital_ransomware_attack_(2020), [Last Visit January 2024].

[7] Jon R. Ransomware attacks on hospitals have changed. 2023, https://www.aha.org/center/cybersecurity-and-risk-advisory-services/ransomware-attacks-hospitals-have-changed, [Last Visit January 2024].

[8] IBM. Ransomware protection solutions - prevent ransomware attacks. 2023, https://www.ibm.com/ransomware, [Last Visit January 2024].

[9] Assen Jvon der, Celdrán AH, Sánchez PMS, Cedeño J, Bovet G, Pérez GM, Stiller B. A lightweight moving target defense framework for multi-purpose malware affecting IoT devices. In: IEEE international conference on communications. 2023, p. 1–6.

[10] Sánchez PMS, Celdrán AH, Bovet G, Pérez GM, Stiller B. SpecForce: A Framework to Secure IoT Spectrum Sensors in the Internet of Battlefield Things. IEEE Commun Mag 2023;61:174–80.

[11] Celdrán AH, Sánchez PMS, Castillo MA, Bovet G, Pérez GM, Stiller B. Intelligent and behavioral-based detection of malware in IoT spectrum sensors. Int J Inf Secur 2023;22:541–61.

[12] Kyungroul L, Sun-Young L, Kangbin Y. Machine learning based file entropy analysis for ransomware detection in backup systems. IEEE Access 2019;7:110205–15.

[13] Arp D, Quiring E, Pendlebury F, Warnecke A, Pierazzi F, Wressnegger C, Cavallaro L, Rieck K. Dos and don'ts of machine learning in computer security. In: 31st USeNIX security symposium. USENIX Security 22, 2022, p. 3971–88.

[14] Willis C. Reducing the dreaded false positives. 2021, https://www.zencos.com/blog/next-generation-aml-false-positives, [Last Visit January 2024].

[15] von der Assen J, Celdrán AH, Luechinger J, Sánchez PMS, Bovet G, Pérez GM, Stiller B. RansomAI: AI-powered ransomware for stealthy encryption. In: 2023 IEEE global communications conference. IEEE; 2023, p. 2578–83.

[16] Trend Micro Research. Defending the expanding attack surface: Trend micro 2022 midyear cybersecurity report | trend micro (US). 2025, https://www.trendmicro.com/vinfo/us/security/research-and-analysis/threat-reports/roundup/defending-the-expanding-attack-surface-trend-micro-2022-midyear-cybersecurity-report.

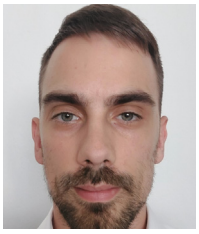
[17] Nolen S, Henry C, Patrick T, evin RB, K B. CryptoLock (and Drop It): Stopping Ransomware Attacks on User Data. IEEE 36th Int Conf Distrib Comput Syst 2016;30:3–312.

[18] Roussev V. Data fingerprinting with similarity digests. In: Advances in digital forensics VI: sixth IFIP WG 11.9 international conference on digital forensics, Hong kong, China, January 4-6 (2010), revised selected papers 6. 2010, p. 207–26.
[19] Fei T, Boyang M, Jinku L, Fengwei Z, Jipeng S, Jianfeng M. Ransomspector: An introspection-based approach to detect crypto ransomware. Comput Secur 2020;97:101997.
[20] Continella A, Guagnelli A, Zingaro G, Pasquale GDe, Barenghi A, Zanero S, Maggi F. Shieldfs: A self-healing, ransomware-aware filesystem. In: Proceedings of the 32nd annual conference on computer security applications. 2016, p. 336–47.
[21] Amin K, Engin K. Redemption: Real-time protection against ransomware at end-hosts. In: Research in attacks, intrusions, and defenses: 20th international symposium. 2017, p. 98–119.
[22] von der Assen J, Celdran AH, Sefa R, Stiller B, Bovet G. MTFS: a moving target defense-enabled file system for malware mitigation. In: 2024 IEEE 49th conference on local computer networks. LCN, Los Alamitos, CA, USA: IEEE Computer Society; 2024, p. 1–6.
[23] Fernando DW, Komninos N. FeSAD ransomware detection framework with machine learning using adaption to concept drift. Comput Secur 2024;137:103629.
[24] Gulmez S, Gorgulu Kakisim I. XRan: Explainable deep learning-based ransomware detection using dynamic analysis. Comput Secur 2024;139:103703.
[25] Almashhadani AO, Kaiiali M, Sezer S, O'Kane P. A multi-classifier network-based crypto ransomware detection system: A case study of locky ransomware. IEEE Access 2019;7:47053–67.
[26] Arslan RS. Identify type of android malware with machine learning based ensemble model. In: 2021 5th international symposium on multidisciplinary studies and innovative technologies. ISMSIT, Vol. 62, 2021, p. 8–632.
[27] Chesti IA, Humayun M, Sama NU, Jhanjhi N and. Evolution, mitigation, and prevention of ransomware. In: 2020 2nd international conference on computer and information sciences. ICCIS, 2020, p. 1–6.
[28] Alwashali AAMA, Abd Rahman NA, Ismail N. A survey of ransomware as a service (raas) and methods to mitigate the attack. In: 2021 14th international conference on developments in eSystems engineering. DeSE, 2021, p. 92–6.
[29] Anand CS, Shanker R. Advancing crypto ransomware with multi level extortion: A peril to critical infrastructure. In: 2023 2nd international conference for innovation in technology. INOCON, 2023, p. 1–5.
[30] Sánchez Sánchez PM, Huertas Celdrán A, Buendía Rubio JR, Bovet G, Martínez Pérez G. Robust federated learning for execution time-based device model identification under label-flipping attack. Clust Comput 2023;1–12.
[31] libfuse. Libfuse. 2024, https://github.com/libfuse/libfuse, [Last Visit January 2024].
[32] hanwen. Go-fuse. 2023, https://github.com/hanwen/go-fuse, [Last Visit January 2024].
[33] Cen M, Jiang F, Qin X, Jiang Q, Doss R. Ransomware early detection: A survey. Comput Netw 2024;239:110138.
[34] Celdrán AH, von der Assen J, Feng C, Padovan S, Bovet G, Stiller B. Next generation of ai-based ransomware. In: 2024 IEEE global communications conference: communication & information systems security. Cape Town, South Africa.: IEEE; 2024, p. 1221–6.
[35] The Apache Software Foundation. Apache jmeter. 2024, https://jmeter.apache.org/, [Last Visit January 2024].
[36] Garfinkel S, Farrell P, Roussev V, Dinolt G. Bringing Science to Digital Forensics with Standardized Forensic Corpora. Digit Investig 2009;6:2–11.
[37] Cyber-Tracer. Guardfs. 2025, https://github.com/Cyber-Tracer/guardfs, [Last Visit February 2025].
[38] Géron A. Hands-on machine learning with scikit-learn, keras, and tensorFlow: concepts, tools, and techniques to build intelligent systems. O'Reilly Media; 2019.
[39] Fraunhofer-Institut für Kommunikation, Informationsverarbeitung und Ergonomie FKIE. Babuk. 2021, https://malpedia.caad.fkie.fraunhofer.de/details/win.babuk, [Last Visit January 2024].
[40] abusech. MalwareBazaar. 2024, https://bazaar.abuse.ch/, [Last Visit January 2024].
[41] Huertas Celdrán A, Sánchez Sánchez PM, Assen Jvon der, Shushack D, Bovet GÁngel Luis Perales Gómez, Martínez Pérez G, Stiller B. Behavioral fingerprinting to detect ransomware in resource-constrained devices. Comput Secur 2023;135:103510.
[42] Lima JO. Gocry. 2017, https://github.com/jeffotoni/gocry, [Last Visit January 2024].
[43] Drakatos P. JavaRansomware. 2017, https://github.com/PanagiotisDrakatos/JavaRansomware, [Last Visit January 2024].
[44] Allen Z. Lollocker. 2015, https://github.com/zmallen/lollocker, [Last Visit January 2024].
[45] BlackBerry. The curious case of Monti ransomware: A real-world doppelganger. 2022, https://blogs.blackberry.com/en/2022/09/the-curious-case-of-monti-ransomware-a-real-world-doppelganger, [Last Visit January 2024].
[46] jimmy ly00. JavaRansomware. 2020, https://github.com/jimmy-ly00/Ransomware-PoC, [Last Visit January 2024].

Jan von der Assen received his MSc degree in Informatics from the University of Zurich. Currently, he is pursuing his Doctoral Degree under the supervision of Prof. Dr. Burkhard Stiller at the Communication Systems Group, University of Zurich. His research interest lies at the intersection between risk management and the mitigation of cyber threats. Contact him at vonderasssen@ifi.uzh.ch.

Chao Feng received the MSc degree in Informatics from the University of Zurich, Switzerland. He is currently pursuing his Ph.D. in computer science at the Communication Systems Group, Department of Informatics at the University of Zurich. His scientific interests include IoT, cybersecurity, data privacy, machine learning, and computer networks. Contact him at cfeng@ifi.uzh.ch.

Alberto Huertas Celdrán is a senior researcher at the Communication Systems Group CSG, Department of Informatics Ifl, University of Zurich UZH. He received his Ph.D. degree in Computer Science from the University of Murcia, Spain. His scientific interests include cybersecurity, machine and deep learning, continuous authentication, and computer networks. Contact him at huertas@ifi.uzh.ch.



Róbert Oleš holds a Bachelor's degree in Economics and a Master's in Computer Science. Previously, he worked as a data scientist with a focus on large-scale geospatial projects. Robert currently works as a software engineer. His interests lie in distributed systems, software engineering, and predictive data analysis. Contact him at olesrobert1@gmail.com.

Gérôme Bovet is the head of data science for the Swiss Department of Defense. He received his Ph.D. in networks and computer systems from Telecom ParisTech, France. His work focuses on Machine and Deep Learning, with an emphasis on anomaly detection, adversarial and collaborative learning in IoT sensors. Contact him at gerome.bovet@armasuisse.ch.

Burkhard Stiller chairs the Communication Systems Group CSG, Department of Informatics Ifl, University of Zürich UZH, as a Full Professor. He received the MSc and Ph.D. degrees from the University of Karlsruhe, Germany. His main research interests include fully decentralized systems, network and service management, IoT, and telecommunication economics. Contact him at stiller@ifi.uzh.ch.