




Article

Adaptive Ransomware Detection Using Similarity-Preserving Hashing

Anas AlMajali ^{1,2,*} , Adham Elmosalamy ², Omar Safwat ²  and Hassan Abouelela ² ¹ Department of Computer Engineering, The Hashemite University, Zarqa 13115, Jordan² Department of Computer Science and Engineering, American University of Sharjah, Sharjah 26666, United Arab Emirates; b00092033@aus.edu (A.E.); b00093225@aus.edu (O.S.); b00092812@aus.edu (H.A.)

* Correspondence: almajali@hu.edu.jo or aalmajali@aus.edu

Abstract: Crypto-ransomware is a type of ransomware that encrypts the victim's files and demands a ransom to return the files. This type of attack has been on the rise in recent years, as it offers a lucrative business model for threat actors. Research into developing solutions for detecting and halting the spread of ransomware is vast, and it uses different approaches. Some approaches rely on analyzing system calls made via processes to detect malicious behavior, while other methods focus on the affected files by creating a file integrity monitor to detect rapid and abnormal changes in file hashes. In this paper, we present a novel approach that utilizes hashing and can accommodate large files and dynamically take into account the amount of change within each file. Mainly, our approach relies on dividing each file into partitions and then performing selective hashing on those partitions to rapidly detect encrypted partitions due to ransomware. Our new approach addresses the main weakness of a previous implementation that relies on hashing files, not file partitions. This new implementation strikes a balance between the detection time and false positives based on the partition size and the threshold of partition changes before issuing an alert.

Keywords: ransomware; Blake3; adaptive-integrity mesh hashing; ransomware detection; malware



Citation: AlMajali, A.; Elmosalamy, A.; Safwat, O.; Abouelela, H. Adaptive Ransomware Detection Using Similarity-Preserving Hashing. *Appl. Sci.* **2024**, *14*, 9548. <https://doi.org/10.3390/app14209548>

Academic Editors: Cuiyun Gao, Aiping Li, Yong Ding and Zhaoquan Gu

Received: 5 August 2024

Revised: 7 October 2024

Accepted: 16 October 2024

Published: 19 October 2024



Copyright: © 2024 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>).

1. Introduction

Ransomware is a type of malicious software (malware) that relies on demanding a payment from the victim (a ransom) for functionality, personal information, or data [1]. Ransomware comes in different shapes and forms; they differ in the way they enter the device and the damage they cause. However, one type of ransomware is prominent and has been on the rise recently, which is crypto-ransomware. This type of ransomware encrypts the files on the victim's device, rendering them useless without a decryption key. Following that, it prompts the victim with information regarding paying the ransom in order to obtain the decryption key and recover the files. A prime example of crypto-ransomware is the infamous WannaCry ransomware, which was launched in April 2017 [2]. The self-replicating crypto-ransomware worm capitalized on a Windows zero-day to propagate to other devices on a network [2]. The ransomware spread rapidly and resulted in billions of dollars in damages to critical infrastructures such as hospitals, all in a span of four days [2]. Thus, ransomware is a serious form of malware that must be studied extensively in order to not only detect it but also halt its malicious execution.

Despite significant advances in security solutions at both the enterprise and consumer-grade levels, ransomware can still be a challenge to detect both before and during the execution of a payload [3–5]. Let us elaborate; the new generation malware utilizes different types of encryption schemes (oligomorphic and polymorphic) in order to hide malicious payloads from antivirus software [6]. Furthermore, threat actors can deploy metamorphic methods that dynamically hide the payloads through opcode (i.e., machine code) modifications that result in varying signatures, making it hard to track the malware across systems [6].

Malware detection methods can be sectioned into three main groups. First, signature-based methods rely on uniquely identifying each instance of malware by creating a unique identifier (i.e., a signature) [6–9]. Commercial antivirus software comes with a database of such signatures and is updated regularly as new malware appears on the internet. The advantage of this method is the almost zero false positive rate, as each individual instance of malware is analyzed by experts before having its signature added to the database. However, one major disadvantage of this method is its inability to detect unknown malware until it has started spreading and causing damage.

Second, heuristically based methods work by matching specific code patterns or structures known to be used with malicious intent [6,7,10,11]. Similarly, they may perform a statistical analysis of the structure of code. While these methods are theoretically able to detect unknown malware, they rarely succeed, as they struggle against the evasion techniques mentioned above, and they also tend to generate false positives due to their nondeterministic nature [6].

Third, behavior-based methods rely on the real-time analysis of a program's execution and its activities [12,13]. For example, they may analyze system calls, network activity, or file changes. These methods have proven more resilient than the other methods when it comes to new-generation malware [14]. The proposed method is behavior-based, not signature-based. This means it can protect against zero-day, unlike signature-based detection. The behavior that the method detects is rapid file changes (encryption). One major challenge these methods face is malware that changes its behavior when its execution is being analyzed (e.g., in a sandbox environment or a virtual machine) [14]. Similar to heuristic-based methods, behavior-based ones also tend to flag benign software as malicious.

Our proposed method falls under the behavior-based malware detection techniques, as, at its core, it employs a file integrity monitor (FIM). FIM implementations try to detect any change in a file system and raise a flag if a change is captured. Typically, this is implemented by storing the hash for each file in the system and continuously comparing the stored hashes with the current hashes. If there is a mismatch, then it is logged, and an alert is issued after a certain threshold of changes indicating a malicious change in the file system. Using the same idea to detect changes in files due to ransomware is tempting, but it entails a major drawback. Ransomware changes the file, so it should be detected via a FIM; however, hashing the whole file is time-consuming. This means that a long time will pass before the FIM issues an alert, causing a lot of files to be lost due to the ransomware.

The authors of [15] proposed a crypto-ransomware detection method that relies on selective hashing. In this method, only small parts of the file are hashed to detect changes. By definition, encrypting a file changes the whole file and produces data that appear to be random. This idea reduces the hashing time substantially, making the behavioral detection of ransomware time-efficient. However, this method presents a drawback, as a threshold is set to issue an alert. The threshold is defined as the number of files that have changed. While this threshold can be efficient for small files, it fails for large files, as, even though a large amount of data are encrypted within a single file, it is still counted as one file. A false negative may occur due to counting the number of files. In this paper, we address the weakness in [15] by dividing each file into partitions, applying selective hashing for each partition, and setting the threshold as the number of partitions, instead of the number of files. The main contributions of this paper can be summarized in the following points:

1. An adaptive hashing methodology is tailored for efficient behavioral ransomware detection that overcomes the security weakness in [15] while still achieving the same performance. By adaptive, we mean that the user can configure two main parameters, partition size and threshold value, according to the security requirements of the environment used.
2. A software implementation of the newly proposed methodology is presented along with a performance analysis and a comparison with the previous implementation. The new adaptive hashing ransomware detection methods outperform the previous

implementation in two main metrics: the detection time and the percentage of saved files. For example, for 50 files of size ~ 100 MB each, 92% of the files are saved within 0.75s of scanning.

The rest of the paper is organized as follows: In Section 2, the literature is reviewed and discussed. In Section 3, the main methodology and implementation are presented. The results are presented and discussed in Section 4. Finally, the concluding remarks and future research directions are discussed in Section 5.

2. Literature Review

In this section, we present and discuss related work on ransomware detection with a special emphasis on methods that utilize hashing. Table 1 summarizes the literature review, which is discussed in the following subsections.

Table 1. Literature review summary.

Reference	Method	Environment	Remarks
Al-Muntaser et al. [16]	Full hashing of files	Windows industrial control systems	Detects any change, not specific to ransomware
Oujezsky et al. [17]	Hashing and machine learning	Backup files	N/A
Novak et al. [18]	Hashing and FAISS (vector quantization)	Backup files	N/A
Naik et al. [19]	Fuzzy hashing on the ransomware	N/A	Dunn index but does not detect new versions
Abbasi et al. [20]	Behavior-based (Windows API, machine learning)	Windows	Looks for feature optimization
Kang et al. [21]	Behavior-based (Windows API)	Windows	Feature selection analysis, peak accuracy = 98.7%
Kok et al. [22]	Signature-based combined with behavior-based	Windows API	Detection rate of 99%, but works for Windows only
Lee et al. [23]	Behavior-based	Machine learning from file entropy	A 99% accuracy rate, but can be bypassed with entropy-avoiding techniques

2.1. Hashing-Based Methods

Al-Muntaser et al. [16] introduced a real-time intrusion detection system for industrial control systems (ICSs) that incorporates hashing as an attribute to monitor. The system stores information such as the file name, permissions, last modification date, and file content hash, among other attributes. Furthermore, the system is event-based; it does not continuously run and instead only triggers checks in the case of file changes by utilizing Windows PowerShell cmdlets, which leads to a reduced performance overhead. However, their proposed system stores the full hash of the file, which may introduce scalability issues, especially for larger files, and which may have a great impact on ICSs, which are performance-critical systems.

Furthermore, Oujezsky et al. [17] proposed a system for data backup with ransomware protection. Initially, the system populates a database with hashes of system files from a trusted online application programming interface (API) or from the current system state,

which is considered the baseline for safety. Following that, the system continuously monitors critical system files and checks for signs of compromise (e.g., ransomware encryption); this is done by checking different file characteristics, including the file hash, and comparing them with the baseline. If a file is deemed safe, it is uploaded to the backup site; however, in cases where a file is detected as corrupted, then the file is restored by downloading the latest safe version of the file onto the backup site. The system utilizes machine learning (ML) techniques such as K-nearest neighbors (KNNs) to check whether the file's characteristics are similar enough to the safe, backed-up version.

Building on their previous work, a subsequent study by Novak et al. [18] improved upon the proposed data backup system by utilizing Facebook AI Similarity Search (FAISS). Instead of using traditional machine learning methods, the new approach uses FAISS for hash comparison. "Faiss is a library for efficient similarity search and clustering of dense vectors" [24]. Novak et al. use the library by generating vectors for the hashes, which allows for rapid hash matching. Additionally, they perform a quantization of the vectors using the product quantization (PQ) method, which further improves performance and reduces the time overhead.

Naik et al. [19] present a fuzzy hashing-based approach to ransomware detection with a focus on malware sample identification. The two-stage method uses hashing to identify ransomware samples. Initially, fuzzy hashing is used to capture similarities between ransomware samples and generate similarity scores. Following that, clustering is used to group together similar samples. Furthermore, the authors evaluated their method using WannaCry/WannaCryptor ransomware samples, comparing two fuzzy hashing methods (SSDEEP and SDHASH) and five clustering methods (K-Means, PAM, AGNES, DIANA, and CLARA). Overall, the method is presented as an efficient initial analysis tool for grouping ransomware samples, however, there is no claim of the method's ability to adapt to new, unseen malware families.

2.2. Windows API-Based Methods

Abbasi et al. [20] introduced an approach to ransomware detection based on a dynamic analysis of Windows API calls. To begin, the system captures a log of all Windows API calls made via processes. Following that, feature engineering using n-grams takes place to transform the data into a valid format for further processing. Before running the data through the ML models, relevant features are extracted to reduce complexity. While this can be done manually by qualified experts in the field, it is time-consuming and obstructs the automation of the procedure and its scalability. Thus, the authors proposed a particle swarm optimization (PSO)-based method for automatic feature selection. Next, the refined features are passed to an ML model to detect ransomware behavior. The authors tested the following ML models: regularized logistic regression (RLR), support vector machine (SVM), KNN, decision trees (DTs), and random forest (RF). The proposed method was able to achieve results close to and occasionally higher than those of manually selected features.

Moreover, Kang et al. [21] put forward a similar framework that analyzes Windows API calls. However, they proposed a novel approach to feature extraction using enhanced min-max relevance (EmRmR), which filters out irrelevant logs, reducing the workload in the subsequent steps. The new approach was tested on the following ML models: SVM, KNN, DT, and RF. RF consistently outperformed all other methods, reaching a peak accuracy of 98.7% using a 3-gram feature length.

Another paper by Kok et al. [22] combines static signature analysis with Windows API analysis, using ML models to develop a two-stage ransomware detection system. First, the system checks for known malware samples, using traditional signature matching with the SHA-256 hashing algorithm. In the second stage, the system uses ML models to analyze the Windows API calls made via the programs to provide an accurate prediction of whether the software is malicious or not. While the system achieves high detection accuracy (>99%) on the testing dataset, this method, like other Windows-API-based methods, cannot generalize to other operating systems.

2.3. Other Methods

Lee et al. [23] proposed an ML-based approach that relies on file entropy analysis to protect backup systems. Let us elaborate; the technique measures the file entropy of the files using different statistical methods, and it uses ML models to predict whether a file has been encrypted or not. Furthermore, their system establishes different baselines for different file types to accommodate the natural entropy levels of different formats, such as images and compressed files. The authors achieved very high detection rates (>99%) with low false positives and negatives on a dataset comprising 1200 files, both clean and encrypted, and across six file types. Additionally, the system adapts the baseline based on the user's usage patterns to further reduce false positives. Nevertheless, the sole reliance on entropy and the small dataset may not provide an accurate indication of real-world performance, as indicated by McIntosh et al. [25], who illustrated how entropy values can be spoofed using techniques such as Base64 encoding to pass encrypted content off as normal.

What distinguishes our method from the methods discussed in this section is the following. (1) Adaptive method: it can be configured to meet our security requirements, balancing false positives and false negatives and achieving fast detection that saves 98% of the time needed (Section 4.4). (2) Behavior-based method: It can be used to detect new ransomware versions based on their behavior. (3) It imposes a minimum overhead on the CPU and memory (Section 4.4).

3. Methodology

This section covers the experimentation setup comprehensively. It addresses the choice of hashing functions, how the files are hashed, and how the detection framework functions, and it presents a detailed description of the custom ransomware designed for testing the performance of the system, as well as the testbench setup.

3.1. Hashing Functions

In terms of hashing functions, Blake3 was chosen [26]. This decision was based on Blake3's architecture, which is designed to make use of the multicore and multithreaded capabilities of CPUs, leading to a superior speed and improved performance on lower-end consumer computers. The algorithm produces a 256-bit hash digest, which is robust against numerous attacks, including pre-image, collision, and differentiability attacks, providing it with 128-bit resistance [26].

3.2. File-Hashing Techniques

Three different, yet incremental, methods were developed to compare between them: full hashing, file-selective hashing (FSH), and similarity-preserving hashing (SPH). The methods vary in the number of hashes they generate, how many bytes they read from the file, and where in the files the bytes are read from (i.e., which snippets of the file are used to generate the hash).

First, full hashing is depicted in Figure 1. In this method, the entire file is hashed to produce a single hash value, regardless of the file size.

Second is FSH, as depicted in Figure 2 [15]. TOTAL_BYTES are chosen from the file randomly and spread across partitions of size SELECTION_SIZE. Following that, the randomly chosen bytes are concatenated and then hashed, resulting in a single hash but only reading TOTAL_BYTES bytes from any file, regardless of its size. If the file is smaller than TOTAL_SIZE, the entire file is hashed in a manner similar to the full hashing method.

Finally, SPH is shown in Figure 3, which is our proposed approach that aims to adapt to larger file sizes, quantify the size of file modification, and reduce performance footprint. Let us elaborate; the file is split into partitions of size PARTITION_SIZE, and within each partition, two blocks each of size BLOCK_SIZE bytes are chosen randomly, concatenated, and then hashed, resulting in a variable size of hashes for each file and allowing for the recognition of changed partitions in the file. Hashing partitions, instead of files, provides a finer resolution, which allows

for setting the threshold counter on partitions instead of files, as elaborated in the following subsection. Moreover, choosing two blocks randomly adds another layer of security, as it would be harder for the attacker to know which blocks were selected for hashing, which reduces the risk of keeping those two blocks unchanged by the attacker to trick the detection mechanism.

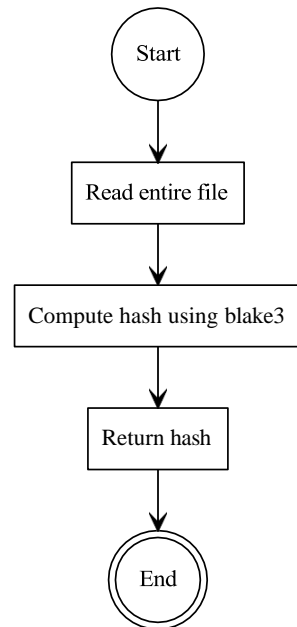


Figure 1. Full hashing method.

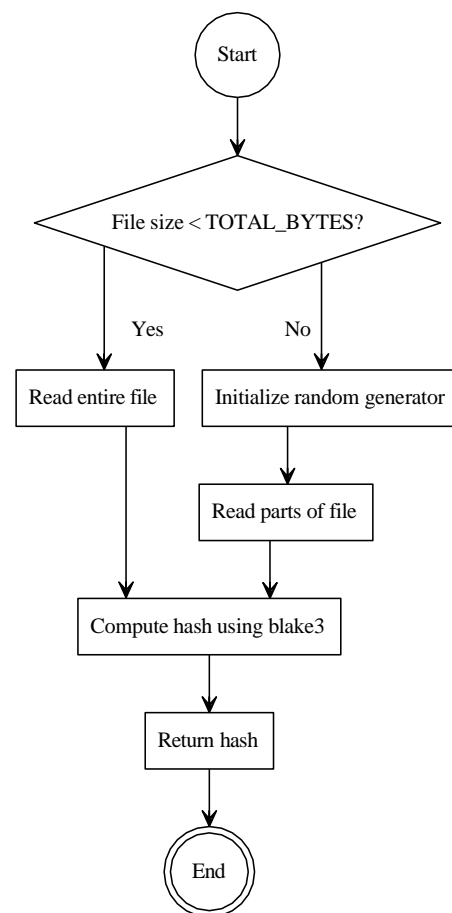


Figure 2. FSH method.

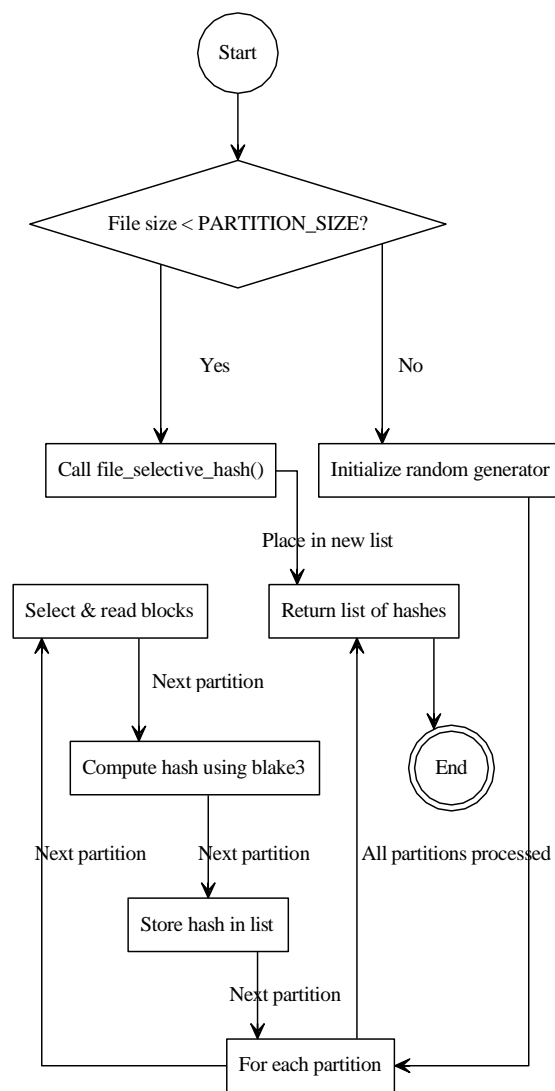


Figure 3. SPH method.

3.3. Detection Mechanism

In order to transform the hashing methods into a usable product, a comprehensive ransomware detection algorithm was developed around them. The algorithm is implemented in software and is run continuously to detect and provide alerts for potential ransomware attacks in a given directory. Let us elaborate; the algorithm is attached to a certain directory, and it proceeds to hash all files in all sub-directories recursively, comparing the hashes if previously hashed or storing them in cases of first discovery. If a change in hashes occurs, a counter is incremented, and if it exceeds a threshold, an alert is raised. However, if a full cycle through the tracked files takes place and the threshold is not exceeded, the changed files counter resets back to zero in order to reduce false positives and offer a more dynamic mechanism able to adapt to benign file changes. The full algorithm is described in Algorithm 1. Moreover, the detection algorithm, all three hashing techniques, and the custom ransomware were implemented as a Python module, which was written in a user-friendly and modular way to allow for extensions to any component, making it feasible to transform into a consumer product.

Algorithm 1 Monitor Directory for File Change**Require:** List of file partitions to track, Baseline hashes, Change threshold

```

1:  $total\_detected \leftarrow 0$ 
2: while True do
3:    $current\_hashes \leftarrow$  Hash all partitions
4:   for each  $partition$  in tracked partitions do
5:      $changes \leftarrow$  Compare  $baseline[partition]$  with  $current\_hashes[partition]$ 
6:      $total\_detected \leftarrow total\_detected + changes$ 
7:     if  $total\_detected \geq threshold$  then
8:       Raise alarm
9:       Break
10:    end if
11:  end for
12:   $total\_detected \leftarrow 0$ 
13: end while

```

3.4. Custom Ransomware

In order to test the performance of the detection algorithm, a custom ransomware implementation was created to emulate a real ransomware attack as closely as possible. Real-world ransomware usually relies on symmetric encryption techniques like the advanced encryption standard (AES) to encrypt user data. Other instances of real-world ransomware have used asymmetric keys for encryption or a combination of symmetric and asymmetric keys, like AES and RSA [27]. Nonetheless, encrypting the same files using symmetric encryption is faster than asymmetric encryption. This is why we used AES, which is a symmetric encryption algorithm and which is practically used in many real-world ransomware scenarios [27]. The ransomware is able to encrypt files in-place by processing them in chunks of a specified size. It uses AES in Counter (CTR) mode with a 256-bit key. First, a 16-byte initialization vector (IV) is randomly generated and prepended to the file. Following that, fixed-size chunks are read from the file, encrypted, and then placed back in the same location. This is repeated until all chunks of the file are encrypted. The in-place encryption allows for faster encryption, which closely mimics real ransomware. The Encrypt File method is described in Algorithm 2.

Algorithm 2 Encrypt File**Require:** File path $file_path$, Encryption key key , Chunk size $chunk_size$ **Ensure:** File at $file_path$ is encrypted in-place

```

1: Initialize AES cipher in CTR mode with  $key$ 
2: Generate random 16-byte initialization vector  $iv$ 
3: Open  $file\_path$  in read-write binary mode
4: Write  $iv$  to the beginning of the file
5: Set file size  $file\_size$  to current file size
6: for  $chunk\_start = 0$  to  $file\_size$  step  $chunk\_size$  do
7:   Seek to  $chunk\_start$  in file
8:   Read chunk of size  $chunk\_size$  from file
9:   if chunk is empty then
10:    break
11:  end if
12:  Encrypt the chunk using AES-CTR
13:  Seek back to  $chunk\_start$  in file
14:  Write encrypted chunk to file
15: end for
16: Close file

```


Furthermore, ransomware can encrypt a directory by iterating over all files and sub-directories recursively and calling the `encrypt_file` method described above. The specification of the `Encrypt Directory` algorithm is shown in Algorithm 3.

Algorithm 3 Encrypt Directory

Require: Directory path *dir*

Ensure: All files within *dir* and its subdirectories are encrypted

- 1: Traverse all files and subdirectories in *dir*
 - 2: **for** each file in directory **do**
 - 3: Call `Encrypt File` for each file
 - 4: **end for**
-

3.5. Test Bench Setup

All benchmarks were run on the same test bench. The specifications of the machine are summarized below:

- Operating system: Windows 11 Pro, Version 10.0.22631;
- Processor: Intel 13th-Gen i7-13700K, 5 GHz;
- RAM: 64 GB;
- Storage: 1 TB SSD.

4. Results

In this section, we present and discuss the results of the proposed ransomware detection methodology.

4.1. Hashing Performance

Our detection algorithm faces a major bottleneck: the hashing algorithm. The ideal hashing algorithm should be fast, incur minimal overhead, be parallelizable, and minimize false positives, as discussed in Algorithm 1.

For each of the three hashing techniques, we benchmarked performance across both total data size and the number of files. For each technique, we benchmarked eight data sizes, starting at 5 GB to 30 GB at 5 GB intervals split across 100, 400, 700, and 1000 files. We plotted the results over three trials to better understand the behavior of our hashing techniques. The results are summarized in Figures 4–6.

Figure 4 shows the execution time of the full hash algorithm. Quite expectedly, full hashing directly correlates to the size hashed, regardless of the number of files.

Figure 5 shows the performance of file-selective hashing. We can see that the execution time is now purely a function of the number of files, regardless of the total data size; as the number of files increases, the execution time increases, regardless of the data size. This behavior is explained in Figure 2, which shows how we only hash select portions of the file, leading to a super-efficient yet descriptive aggregate hash. Time could be further reduced if we minimized the file handling overhead or parallelized the file reads.

Figure 6 shows the SPH performance, which is comparable to that of FSH seen before with the added benefit of a better descriptive capability. SPH (as visualized in Figure 3) returns a list of hashes, with each describing the state of changes in different portions of the affected file. We think this technique has the highest potential, and with more research and optimization, it can become a highly performant litmus test to stop ransomware and maximize file recovery rates. As demonstrated in Figure 6, the maximum execution time was about 1.3 s for hashing 30 GB of 500 files.

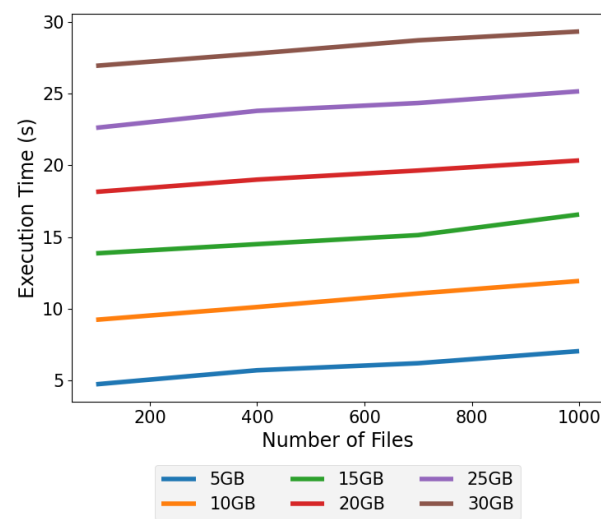


Figure 4. Performance of full-hash algorithm.

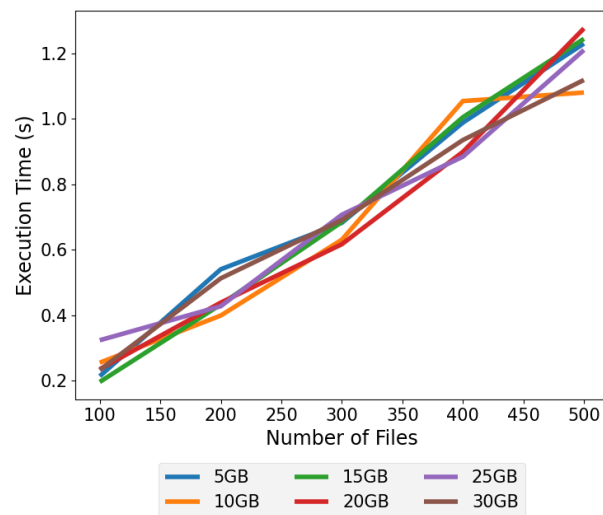


Figure 5. Performance of file-selective hash.

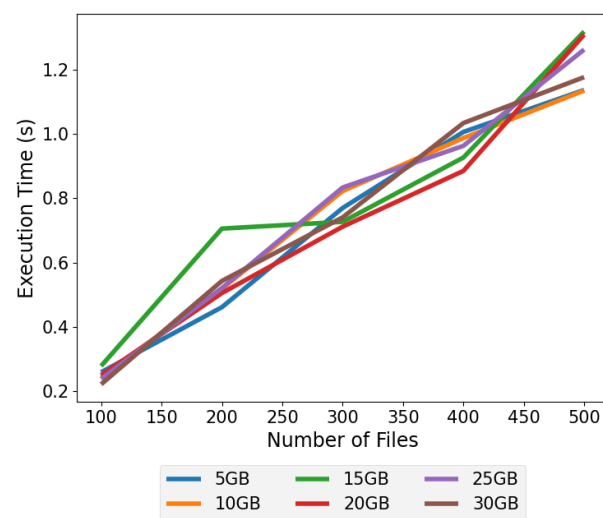


Figure 6. Performance of SPH.

4.2. SPH Performance

The system was put to the test by running the SPH alongside the custom ransomware implementation and seeing how well the SPH performed in terms of efficiency (the percentage of files saved) and speed (the time taken until detection). Furthermore, different threshold levels were chosen. The threshold represents the number of changed hashes (corrupted files). Both algorithms were tested. The test setup is summarized below:

- Fifty monitored files in total;
- Total size (distributed across the 50 files): 5 GB, 10 GB, 15 GB, and 20 GB;
- Algorithms: FSH and SPH;
- SPH partition sizes: 10 MB, 40 MB, 70 MB, and 100 MB;
- Threshold levels: 7, 21, 35, and 49;
- Each test was repeated three times for verification.

Figures 7 and 8 were generated, each with four sub-plots, one for each total file size (and a proportionally increasing partition size). Furthermore, since three trials for configuration were conducted, the median absolute deviation (MAD) was calculated, and it is depicted as a black line at the top of the bars in the plot. MAD refers to taking the median of the absolute differences of the data points from the mean. This technique offers robustness to outliers over methods for measuring the spread, such as the standard deviation, especially with such small sample sizes.

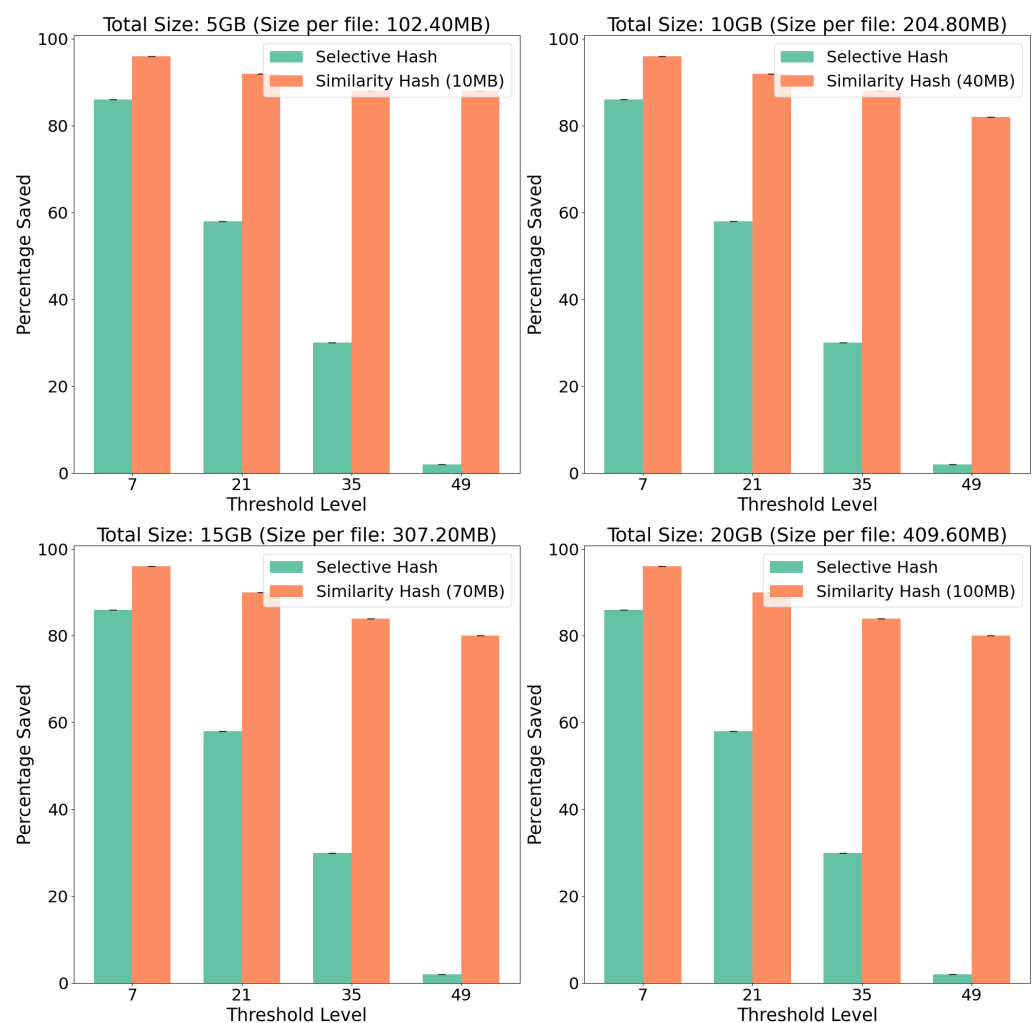


Figure 7. Comparing the algorithms in terms of the % of files saved.

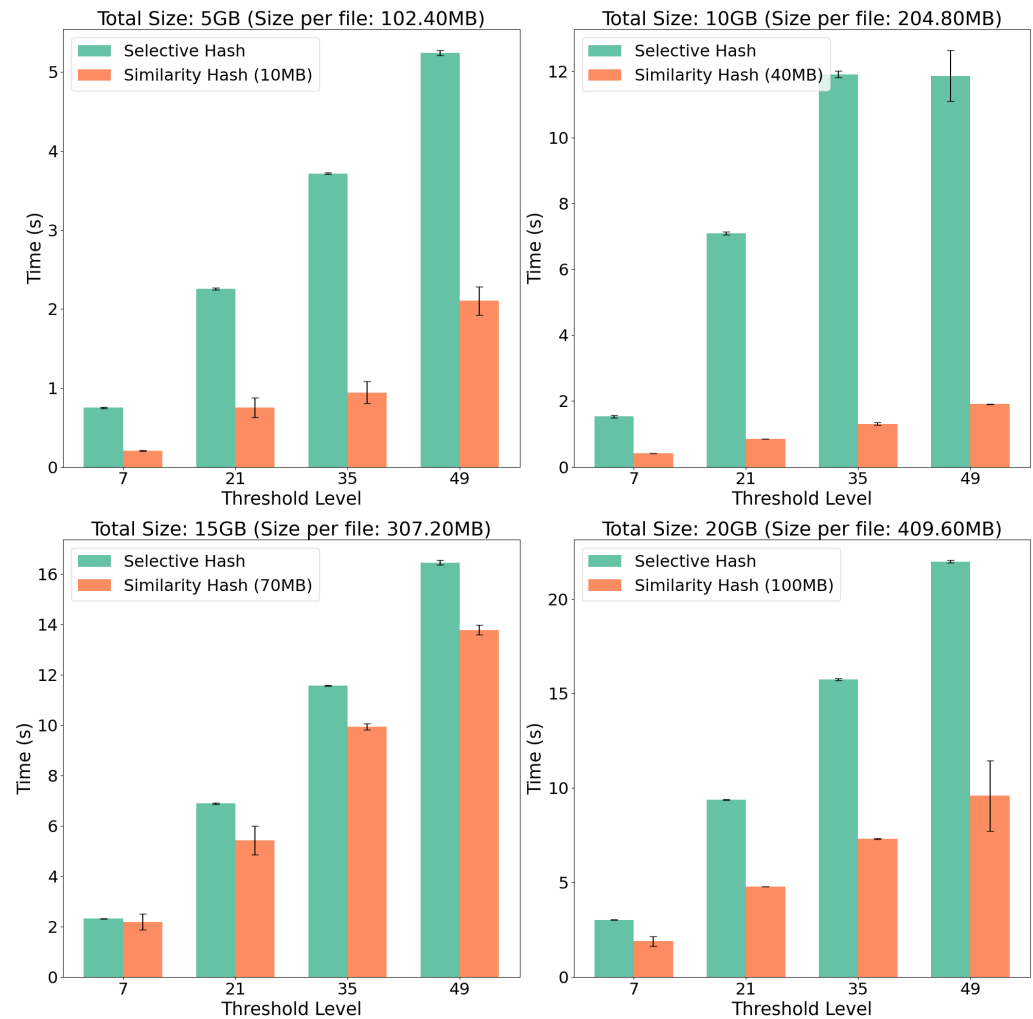


Figure 8. Comparing the algorithms in terms of detection speed since infection.

4.2.1. Comparing Algorithms

First, the SPH algorithm is compared against selective hashing. Figure 7 illustrates the comparison of the algorithms with regard to the percentage of files saved from ransomware. The main weakness in the FSH is obvious in this figure, as SPH performs better. This can be attributed to basing the detection on the count of partitions that changed, not the count of files that changed for SPH. If one file changes, it counts as one in FSH; however, in SPH, it counts more than one, depending on the partition size, which means that the threshold can be reached faster. It is also noticeable that, if we choose a less conservative approach that has a higher threshold (prone to false positives), SPH still performs much better than FSH.

Meanwhile, Figure 8 depicts the comparison in terms of time (in seconds) until detection, measured from the start of the ransomware execution (starting to encrypt files). This figure confirms that SPH performs better than FSH in terms of detection time. For example, for 50 files of size 100 MB each, 92% of the files (46 files) are saved within 0.75 s of scanning.

4.2.2. Comparing Partition Sizes and Threshold Values

Second, tests were conducted to compare the partition sizes of the SPH algorithm. Figure 9 shows the difference in the percentage of files saved across the four partition sizes. It is intuitive that smaller partition sizes have better (i.e., less) detection times. However, it should be mentioned that the partition size is correlated to the file size. If the partition size is greater than or equal to the file size, then SPH performs the same as FSH. In all test figures, the threshold value was tested. It is clear that smaller thresholds perform better in terms of the percentage of files saved and the detection speed. However, this means

that more partitions have to be hashed. The partition size and the threshold value are configurable parameters that can be set by the user. Those two parameters affect the false positive rate, the false negative rate, the detection time, the percentage of time saved, and the time required to scan through the complete file list. This is elaborated in Section 4.4.

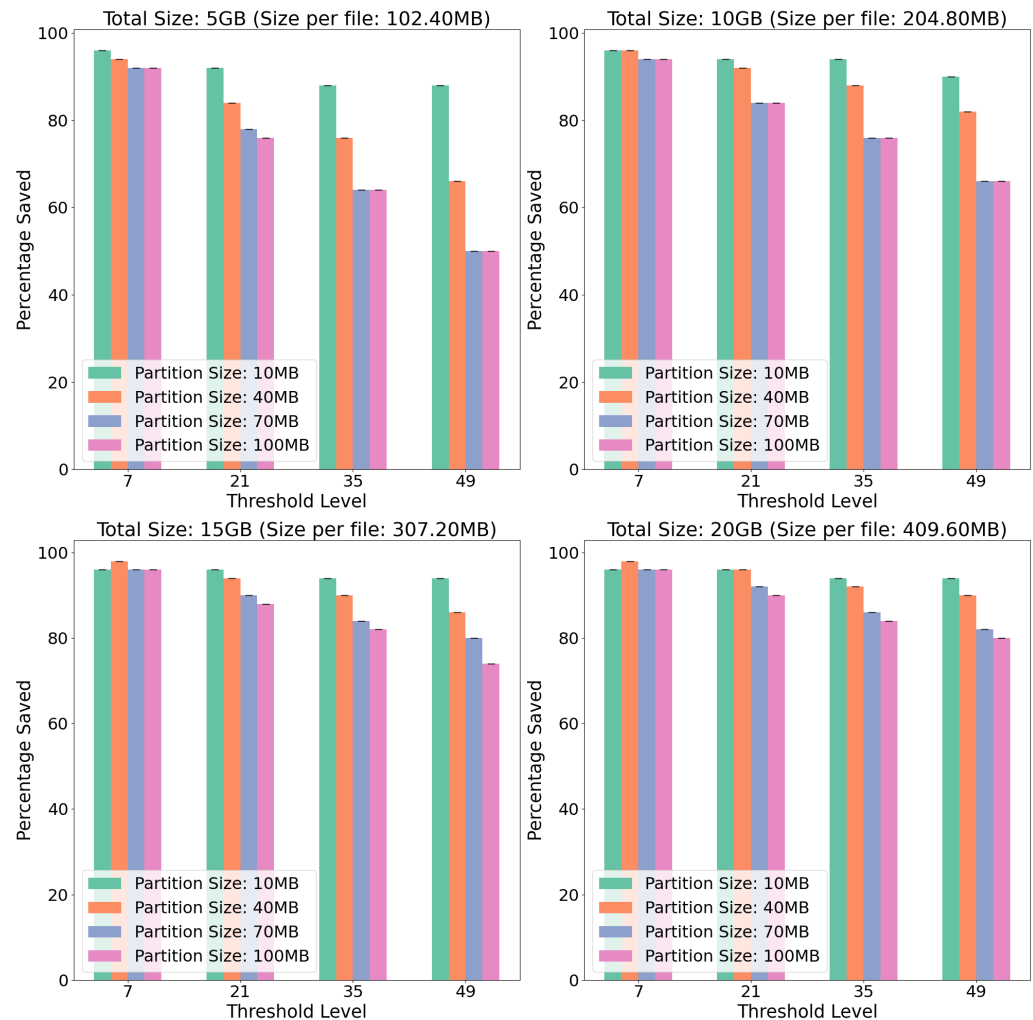


Figure 9. Comparing the partition sizes in terms of the % of files saved.

4.3. Computational Costs

Throughout the benchmarking tests, both the CPU and memory utilization for SPH were recorded periodically to assess the impact of the detection system on the computer. Figure 10 shows the CPU and memory utilization for SPH with 10MB throughout all file sizes. These configurations were chosen since they generate the largest number of hashes per file and, therefore, would provide a worst-case scenario for the system. Let us elaborate; the metrics were sampled at every iteration, which is a full loop through the tracked files. It can be seen that there is little to no impact on system resources, with CPU utilization being mostly constant across the run and peaking at 4.2%. As for memory utilization, the highest is 0.04% (26.17 MB). Finally, it should be noted that these metrics were recorded while the ransomware was running simultaneously to mimic a real use case for the system.

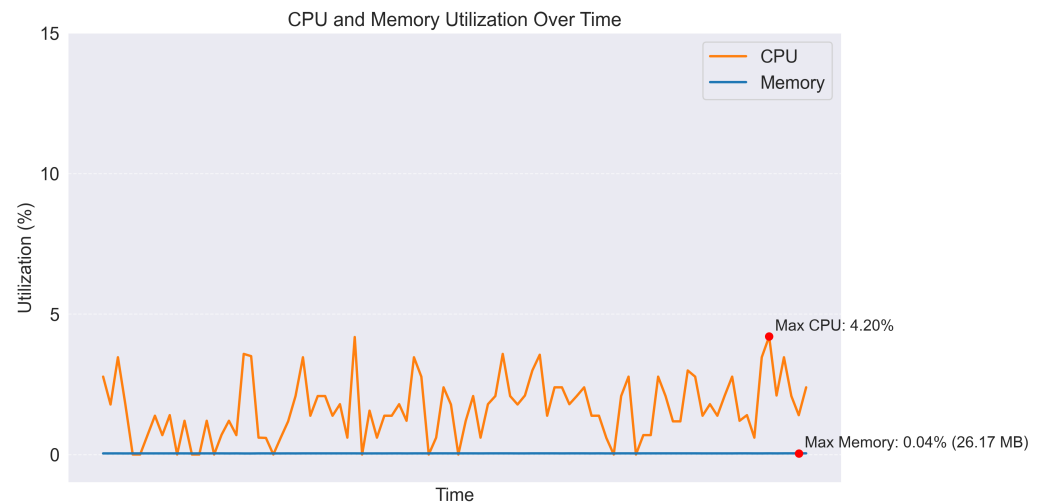


Figure 10. CPU and memory utilization for SPH with 10MB partition size.

4.4. Discussion

When comparing the efficiency of both algorithms, it can be seen that SPH outperforms traditional FSH at every level. The performance gap only increases as the threshold increases. Let us elaborate; in the conducted tests, each file has multiple hashes in the case of SPH. Meanwhile, for FSH, it is exactly one hash. As a result, a single file results in multiple detected changes under SPH, making it possible to detect the ransomware earlier. Similarly, the time till detection depicts a similar story. As the threshold increases, it takes SPH less time to detect the ransomware than FSH. SPH detects rapid changes in files, which can be benign changes resulting in false positives. However, the false positive rate can be controlled by setting the appropriate threshold value and setting a timer for resetting the threshold count to zero. One point to consider is that, due to the multi-threading nature of the code and natural variations in the test environment, there is increased variability in the results. However, it is still an accurate indicator of the performance, especially when taking the percentage of files saved into consideration.

SPH was able to identify the running ransomware within 1.74 s at a threshold level of 7 and within approximately 10 s for the more lenient threshold of 49 at a total file size of 20 GB. The reported average encryption time for modern ransomware is 42 min and 52 s, according to [28]. However, the same research notes that the fastest ransomware variant (LockBit 3.0) can encrypt 50 GB of data within 5 min and 50 s. By comparison, our algorithm can detect ransomware within a few seconds, depending on the file size, partition size, and threshold value (Figures 7 and 8).

To test the detection method in a realistic environment, we used a dataset called corpora [29]. Corpora is a dataset of 12.8K office files (including csv, pdf, docx, pptx, xlsx, txt, jpg, and more) of 11GBs in size. The simulated ransomware takes about 14.39 s to completely encrypt the dataset. SPH detects the ransomware within 24 milliseconds. This means that detection can be done in 0.17% of the time the ransomware takes to finish its work, saving 98% of files.

The results of the partition size comparison show a clear and predictable advantage to using smaller partition sizes (e.g., 10 MB and 40 MB). Let us elaborate; a smaller partition size leads to more hashes per file, and therefore, changes are detected faster, as a single file contributes multiple times to the total change counter. However, this increase comes at the cost of increased storage needed to save the extra hashes per file. This trade-off appears to be justified for smaller file sizes, especially at higher thresholds, as seen in the 5 GB sub-graph in Figure 8. Another observation is that larger file sizes tend to perform well (less corrupted files) compared to smaller ones due to the increased number of hashes, as previously mentioned.

Based on the observed results, it can be approximated that a partition size between 10 MB and 40 MB offers the most balance between detection efficiency and storage overhead. Similarly, a detection threshold between 21 and 35 would be suitable for the aforementioned reasons. Nevertheless, it must be noted that these parameters should be tuned for the given environment in which the FIM is operating, as they are impacted by many factors, including the average file size, update frequency, average change size, and risk tolerance, among other characteristics.

Finally, we want to discuss the adaptive part of SPH. By adaptive, we mean that the detection method can be configured according to user preferences. There are two main parameters that the user can configure: files' partition size and threshold value. Instead of hashing complete files to detect changes in them due to ransomware, in this method, file partitions are hashed. As Figure 9 shows, smaller partition sizes allow finer scanning and detection but increase the number of partitions that have to be hashed (i.e., they increase the overhead). To distinguish between normal changes (i.e., benign changes) and malicious changes (due to ransomware), a threshold of how many file partitions have changed is set. This threshold value is also adaptive, and it can be configured by the user. As Figures 7–9 show, setting a low threshold (e.g., seven) means having stricter security requirements. An alert is issued if seven file partitions are changed in this case. Setting a low threshold value might increase the rate of false positives, as a stricter security requirement is set. On the other hand, increasing the threshold value (e.g., 49), relaxes the security requirements. This means that 49 partition changes occur before an alert is issued. This might cause the false negative rate to increase. The user can configure the partition size and the threshold value according to the nature of the environment used.

5. Conclusions

In summary, this paper has discussed a detection and monitoring paradigm for minimizing ransomware's impact using various methods and approaches to detecting ransomware based on different hashing techniques. We developed a modular system with components for hashing, file generation, benchmarking, graphing, ransomware emulation, and detection. Our results show that each technique has its merit; for example, the full hash algorithm correlates directly to the size hashed, regardless of the number of files. FSH shows that the execution time is purely a function of the number of files, while SPH performance is comparable with the added benefit of a better descriptive capability. These findings guide further work on the use of hashing algorithms for high-performance and low-latency ransomware detection.

We recognize this research as the potential seed for a new consumer-grade ransomware detection tool that can be suitable for endpoints such as mobile phones, laptops, and even low-end server devices due to its speed and effectiveness. While our proof-of-concept implementation was written in Python, a commercial implementation of our research should be implemented using a more performant language that introduces lower memory and lower time overhead; C++, Go, and Rust are possible candidates. Other considerations include using a hybrid approach that adds ML capabilities to our detector, writing a kernel-level component responsible for terminating offending processes upon detection (our current system is discovery only), using OS-based triggers (instead of interval-based triggers) and other improvements that all fall in the optimization category, such as a dynamic component for determining and fine-tuning parameters such as the partition size and detection threshold based on the target environment. Another important aspect that cannot be neglected is testing against modern malware with ultra-rapid encryption capabilities, such as LockBit. Finally, like any anti-malware software, SPH requires the scanning of system and user files, which raises privacy issues. Data collection and misuse can be a concern, so user consent must be taken into consideration.

Author Contributions: Conceptualization, A.A., A.E., O.S. and H.A.; methodology, A.A., A.E., O.S. and H.A.; software, A.E., O.S. and H.A.; validation, A.A., A.E., O.S. and H.A.; formal analysis, A.A., A.E., O.S. and H.A.; investigation, A.A., A.E., O.S. and H.A.; data curation, A.A., A.E., O.S. and H.A.; writing—original draft preparation, A.A., A.E., O.S. and H.A.; writing—review and editing, A.A., A.E., O.S. and H.A.; visualization, A.E., O.S. and H.A.; supervision, A.A.; project administration, A.A.; funding acquisition, A.A. All authors have read and agreed to the published version of the manuscript.

Funding: The work in this paper was supported, in part, by the Open Access Program from the American University of Sharjah.

Institutional Review Board Statement: This paper represents the opinions of the author(s) and does not mean to represent the position or opinions of the American University of Sharjah.

Informed Consent Statement: Not applicable.

Data Availability Statement: The original contributions presented in the study are included in the article, further inquiries can be directed to the corresponding author.

Conflicts of Interest: The authors declare no conflicts of interest.

Abbreviations

The following abbreviations are used in this manuscript:

Malicious software	Malware
File integrity monitor	FIM
Application programming interface	API
Machine learning	ML
File-selective hashing	FSH
Similarity-preserving hashing	SPH
Similarity-digest hashing	SDHASH
Advanced encryption standard	AES

References

1. Anghel, M.; Racautanu, A. A note on different types of ransomware attacks. *Cryptol. Eprint Arch.* **2019**, *in press*.
2. Lessing, M. Case Study: WannaCry Ransomware. Available online: <https://www.sdxcentral.com/security/definitions/what-is-ransomware/case-study-wannacry-ransomware/> (accessed on 3 August 2024).
3. Bensaoud, A.; Kalita, J.; Bensaoud, M. A survey of malware detection using deep learning. *Mach. Learn. Appl.* **2024**, *16*, 100546. [\[CrossRef\]](#)
4. Gaber, M.G.; Ahmed, M.; Janicke, H. Malware detection with artificial intelligence: A systematic literature review. *ACM Comput. Surv.* **2024**, *56*, 1–33. [\[CrossRef\]](#)
5. Cen, M.; Jiang, F.; Qin, X.; Jiang, Q.; Doss, R. Ransomware early detection: A survey. *Comput. Netw.* **2024**, *239*, 110138. [\[CrossRef\]](#)
6. Aslan, O.A.; Samet, R. A Comprehensive Review on Malware Detection Approaches. *IEEE Access* **2020**, *8*, 6249–6271. [\[CrossRef\]](#)
7. Arora, P.; Gupta, R.; Malik, N.; Kumar, A. Malware Analysis Types & Techniques: A Survey. In Proceedings of the 5th International Conference on Information Management & Machine Intelligence (ICIMMI '23), Jaipur, India, 23–25 November 2023; Association for Computing Machinery: New York, NY, USA, 2024. [\[CrossRef\]](#)
8. Patil, J. Malware Detection of Portable Executable Using Machine Learning and Neural Networks. *Int. J. Res. Appl. Sci. Eng. Technol.* **2024**, *12*. [\[CrossRef\]](#)
9. Shaukat, K.; Luo, S.; Varadharajan, V. A novel deep learning-based approach for malware detection. *Eng. Appl. Artif. Intell.* **2023**, *122*, 106030. [\[CrossRef\]](#)
10. Bazrafshan, Z.; Hashemi, H.; Hazrati Fard, S.M.; Hamzeh, A. A survey on heuristic malware detection techniques. In Proceedings of the IKT 2013—2013 5th Conference on Information and Knowledge Technology, Shiraz, Iran, 28–30 May 2013; pp. 113–120. [\[CrossRef\]](#)
11. Manavi, F.; Hamzeh, A. A new approach for malware detection based on evolutionary algorithm. In Proceedings of the Genetic and Evolutionary Computation Conference Companion (GECCO '19), Prague, Czech Republic, 13–17 July 2019; Association for Computing Machinery: New York, NY, USA, 2019. [\[CrossRef\]](#)
12. Dumitrasc, V.; Serrall-Gracià, R. User Behavior Analysis for Malware Detection. In *Computer Security, Proceedings of the ESORICS 2023 International Workshops, The Hague, The Netherlands, 25–29 September 2023*; Katsikas, S., Abie, H., Ranise, S., Verderame, L., Cambiaso, E., Ugarelli, R., Praça, I., Li, W., Meng, W., Furnell, S., et al., Eds.; Springer: Cham, Switzerland, 2024; pp. 92–110.
13. Kaya, Y.; Chen, Y.; Saha, S.; Pierazzi, F.; Cavallaro, L.; Wagner, D.; Dumitras, T. Demystifying Behavior-Based Malware Detection at Endpoints. *arXiv* **2024**, arXiv:2405.06124. [\[CrossRef\]](#)

14. Aslan, O.; Samet, R. Investigation of Possibilities to Detect Malware Using Existing Tools. In Proceedings of the 2017 IEEE/ACS 14th International Conference on Computer Systems and Applications (AICCSA), Hammamet, Tunisia, 30 October–3 November 2017; pp. 1277–1284. [\[CrossRef\]](#)
15. AlMajali, A.; Qaffaf, A.; Alkayid, N.; Wadhawan, Y. Crypto-ransomware detection using selective hashing. In Proceedings of the 2022 International Conference on Electrical and Computing Technologies and Applications (ICECTA), Ras Al Khaimah, United Arab Emirates, 23–25 November 2022; pp. 328–331.
16. Al-Muntaser, B.; Mohamed, M.A.; Tuama, A.Y. Real-Time Intrusion Detection of Insider Threats in Industrial Control System Workstations Through File Integrity Monitoring. *Int. J. Adv. Comput. Sci. Appl.* **2023**, *14*. . [\[CrossRef\]](#)
17. Oujezsky, V.; Novak, P.; Horvath, T.; Holik, M.; Jurcik, M. Data Backup System with Integrated Active Protection Against Ransomware. In Proceedings of the 2023 46th International Conference on Telecommunications and Signal Processing (TSP), Prague, Czech Republic, 12–14 July 2023; pp. 65–69.
18. Novak, P.; Kaura, P.; Oujezsky, V.; Horvath, T. Ransomware File Detection Using Hashes and Machine Learning. In Proceedings of the 2023 15th International Congress on Ultra Modern Telecommunications and Control Systems and Workshops (ICUMT), Ghent, Belgium, 30 October–1 November 2023; pp. 107–110.
19. Naik, N.; Jenkins, P.; Gillett, J.; Mouratidis, H.; Naik, K.; Song, J. Lockout-Tagout Ransomware: A Detection Method for Ransomware using Fuzzy Hashing and Clustering. In Proceedings of the 2019 IEEE Symposium Series on Computational Intelligence (SSCI), Xiamen, China, 6–9 December 2019; pp. 641–648. [\[CrossRef\]](#)
20. Abbasi, M.S.; Al-Sahaf, H.; Mansoori, M.; Welch, I. Behavior-based ransomware classification: A particle swarm optimization wrapper-based approach for feature selection. *Appl. Soft Comput.* **2022**, *121*, 108744. [\[CrossRef\]](#)
21. Kang, Q.; Gu, Y. Enhancing Ransomware Detection: A Windows API Min Max Relevance Refinement Approach. *Preprint* **2023**. [\[CrossRef\]](#)
22. Kok, S.; Abdullah, A.; Jhanjhi, N. Early detection of crypto-ransomware using pre-encryption detection algorithm. *J. King Saud Univ.-Comput. Inf. Sci.* **2022**, *34*, 1984–1999. [\[CrossRef\]](#)
23. Lee, K.; Lee, S.Y.; Yim, K. Machine Learning Based File Entropy Analysis for Ransomware Detection in Backup Systems. *IEEE Access* **2019**, *7*, 110205–110215. [\[CrossRef\]](#)
24. Douze, M.; Guzhva, A.; Deng, C.; Johnson, J.; Szilvasy, G.; Mazaré, P.E.; Lomeli, M.; Hosseini, L.; Jégou, H. The Faiss library. *arXiv* **2024**, arXiv:2401.08281.
25. McIntosh, T.; Jang-Jaccard, J.; Watters, P.; Susnjak, T. The Inadequacy of Entropy-Based Ransomware Detection. In *Proceedings of the Neural Information Processing*; Gedeon, T., Wong, K.W., Lee, M., Eds.; Springer: Cham, Switzerland, 2019; pp. 181–189. [\[CrossRef\]](#)
26. O'Connor, J.; Aumasson, J.P.; Neves, S.; Wilcox-O'Hearn, Z. *BLAKE3: One Function, Fast Everywhere*; Technical Report, BLAKE3 Team, Version 20211102173700; Technical Report; 2019. Available online: <https://github.com/BLAKE3-team/BLAKE3-specs/blob/master/blake3.pdf> (accessed on 4 August 2024).
27. Cicala, F.; Bertino, E. Analysis of Encryption Key Generation in Modern Crypto Ransomware. *IEEE Trans. Dependable Secur. Comput.* **2022**, *19*, 1239–1253. [\[CrossRef\]](#)
28. Davis, S. Gone in 52 Seconds...and 42 Minutes: A Comparative Analysis of Ransomware Encryption Speed. Available online: https://www.splunk.com/en_us/blog/security/gone-in-52-seconds-and-42-minutes-a-comparative-analysis-of-ransomware-encryption-speed.html (accessed on 3 August 2024).
29. Garfinkel, S.; Farrell, P.; Roussev, V.; Dinolt, G. Bringing science to digital forensics with standardized forensic corpora. *Digit. Investig.* **2009**, *6*, S2–S11. [\[CrossRef\]](#)

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.