

Data Encryption Battlefield: A Deep Dive into the Dynamic Confrontations in Ransomware Attacks

Arash Mahboubi^{a,*}, Hamed Aboutorab^b, Seyit Camtepe^{b,d}, Hang Thanh Bui^b, Khanh Luong^c, Keyvan Ansari^e, Shenlu Wang^{b,d}, Bazara Barry^f

^aCharles Sturt University, New South Wales, 2444, Australia

^bUniversity of New South Wales, Canberra, 2600, Australia

^cQueensland University of Technology, Queensland, 4000, Australia

^dCSIRO Data 61, Sydney, New South Wales, 2000, Australia

^eMurdoch University, Western Australia, 6150, Australia

^fCyber Security NSW – NSW Department of Customer Service

Abstract

In the rapidly evolving domain of cybersecurity threats, ransomware stands out as a formidable challenge. Adversaries are increasingly employing advanced encryption techniques, such as entropy reduction using Base64 encoding, along with partial and intermittent encryption, to bypass traditional security measures and maximize their illicit gains. This study delves into the nuanced battleground between these adversaries, who are adept at refining encryption strategies to evade detection, and the defenders, who are constantly developing sophisticated countermeasures to safeguard vulnerable data assets. At the heart of our investigation is the application of online incremental machine learning algorithms designed to predict file encryption activities, even in the face of evolving adversaries' complex obfuscation tactics. Our research is underpinned by an extensive dataset, which encompasses 32.6 GB of data across 11,928 distinct files, including Microsoft Word documents (.doc), PowerPoint presentations (.ppt), Excel spreadsheets (.xlsx), and various image formats (.jpg, .jpeg, .png, .tif, .gif), PDF files (.pdf), audio files (.mp3), and video files (.mp4), all encrypted by a wide variety of 75 ransomware families. This dataset facilitates a comprehensive empirical analysis, enabling the assessment of various machine learning classifiers' effectiveness in predicting encryption events amid a range of adversarial strategies. The study's results highlight the exceptional performance of the Hoeffding Tree algorithm, which stands out for its incremental learning capabilities, making it particularly adept at identifying conventional and AES-Base64 (e.g., encryption-encoding used to reduce the entropy values) adversarial methods. In contrast, the Random Forest classifier, augmented with a warm-start feature, proves to be highly effective against the more elusive intermittent encryption techniques, underscoring the significance of bespoke machine learning solutions in navigating the dynamic and sophisticated landscape of ransomware threats.

Keywords: ransomware, intermittent, partial encryption, incremental online learning, Base64 encoding, file system, Hoeffding Tree

1. Introduction

In the ever-evolving landscape of cybersecurity, the battle against ransomware represents one of the most challenging frontiers [1]. This digital plague, characterized by its ability to encrypt victims' files and demand ransom for their release, has become a significant threat to individuals, organizations, and even national security. The dynamic nature of this threat is underscored by the cat-and-mouse game between cyber adversaries, notably ransomware developers, and defenders. This confrontation is emblematic of an asymmetrical battlefield where attackers, needing only to find a single vulnerability, consistently remain one step ahead of defenders who must secure every possible point of entry.

Ransomware developers are in a constant state of innovation, refining their tactics, techniques, and procedures (TTPs)

to exploit the inevitable delay in defensive responses to new threats. This relentless advancement ensures their strategies are not just current but often pioneering, placing defenders in a perpetual state of catch-up. The inherent challenge in cybersecurity defense is the difficulty of anticipating and defending against the unknown. As adversaries introduce new and more complex ransomware variants, defenders are forced into a reactive stance, struggling to react effectively post-incident. For example, NetWalker, also known as Mailto, is a ransomware targeting Windows systems, first seen in 2019 and impacting healthcare, education, and government sectors. Operating as a ransomware-as-a-service (RaaS), it allows affiliates to execute attacks and share ransom profits. Its fileless nature, the fileless malware like NetWalker, however, operates directly within the computer's memory, leveraging legitimate system tools and processes to perform its malevolent actions, including file encryption and data theft. This stealthy approach not only enhances its evasion capabilities but also complicates efforts to mitigate and eradicate the ransomware from infected systems

*Corresponding author

Email address: amahboubi@csu.edu.au (Arash Mahboubi)

[2].

The heart of this challenge lies in the limitations of current ransomware detection methodologies, including machine learning models, static analysis, user behavior analytics, and dynamic analysis. Each of these approaches, while comprehensive in their scope, often lags behind the sophisticated and continuously evolving offensive tactics employed by ransomware developers. For instance, machine learning models, which are trained on known malware samples, find it difficult to identify zero-day attacks that display previously unseen behaviors [3]. Static analysis tools, on the other hand, are increasingly evaded by ransomware that employs sophisticated obfuscation techniques. User behavior analytics can result in high false positive rates due to the variability in legitimate user activities, and dynamic analysis might be circumvented by ransomware that can detect and alter its behavior in sandboxed environments.

The future of ransomware defense is poised at a critical juncture, necessitating a departure from traditional detection methodologies towards the adoption of more predictive and adaptive strategies. Our research endeavors to explore the sophisticated tactics employed by adversaries to undermine the efficacy of existing defense mechanisms, particularly those operating at the file system level. These adversaries skillfully navigate around conventional security measures by employing strategies such as entropy value reduction, intermittent, and partial encryption, thereby diminishing the impact and detection capabilities of current defense systems.

Furthermore, our study delves into the potential of online incremental learning as a pivotal technology to enhance the differentiation between encrypted and normal files within hot data storage environments [4]. This approach is particularly relevant in the context of ransomware attacks that utilize intermittent and partial encryption techniques to evade detection. Our research aims to continuously update the detection models with new data (encrypted and normal), enabling them to adapt to evolving ransomware encryption tactics dynamically.

Our focus lies on the identification of unique file attributes, represented as feature vectors, that serve as effective countermeasures against adversarial actions. Several related works rely on entropy-based detection [5, 6, 7, 8], involve frequent file system operations [9] and system logs frequent pattern mining [10]. These examples of mitigation strategies, used by defenders, can be evaded using tactics such as intermittent encryption, partial encryption, and memory mapping. These tactics are discussed in detail in Section 3. We place a spotlight on features revealing the diversity of file types within systems, particularly under the online incremental machine learning framework [11]. The contributions of this paper are manifold and can be delineated as follows:

1. We formalize existing adversarial encryption techniques used by ransomware developers, highlighting their capability to circumvent traditional security measures.
2. We identify unique feature vectors capable of distinguishing between encrypted and unencrypted data at the file system level.
3. Investigating ML models that can reduce the computational costs associated with the need for CPU- and memory-intensive tasks.
4. Through empirical research, we investigate strategic approaches for both attackers, aiming to minimize entropy to evade detection, and defenders, using entropy as a pivotal metric for spotting encrypted data. Our study particularly examines the impact of Base64 encoding on entropy reduction and evaluates its potential exploitation by adversaries to lower data entropy successfully.
5. We evaluate the efficacy of machine learning models in real-world ransomware scenarios, aiming to thwart ransomware attacks during their encryption phase. This includes determining alert activation thresholds and developing adaptive strategies to respond to evolving data patterns.

The structure of this paper is organized as follows: In Section 2, we outline the threat model and foundational assumptions underpinning this study. Section 3 outlines methodologies to understand strategies in adversarial and defensive scenarios, focusing on advanced encryption techniques used by ransomware developers. Moving on to Section 4, we describe our methodology and approach. In Section 5, we conduct a comprehensive analysis of machine learning paradigms, with a specific focus on both shallow and deep learning classifiers. To evaluate their efficacy, we perform targeted micro-experiments. Additionally, we articulate the core principles associated with online incremental machine learning, tailored for the identification of encrypted files within a file-level system. Subsequently, Section 6 reviews existing literature and elucidates the principles that inform it. Finally, Section 7 offers concluding remarks and outlines prospective directions for future research.

2. Threat model

In our threat model, we presume that adversaries possess comprehensive knowledge of the proposed system and the features extracted from the files. This is grounded in the practical realities of the cybersecurity landscape, where sophisticated adversaries are often privy to significant information about the systems they target. For our experiments, we employ real-world ransomware to encrypt files, replicating the tactics commonly deployed by adversaries, such as partial encryption and intermittent encryption.

It should be clearly stated that our systematization of knowledge investigation does not aim to detect the ransomware binary or its behavior patterns. Rather, our main objective is to ascertain the encryption status of a multitude of file types within a file system. By focusing our efforts on this aspect, we aim to develop a robust and accurate tool that can contribute to the mitigation of damage and data loss from ransomware attacks, rather than identifying the ransomware itself. This approach serves as a complement to other strategies and models focusing on ransomware detection and behavior analysis.

To understand countermeasures and adversaries' techniques, we elucidate the interplay between adversaries and defenders using the framework of strategic games. By framing the cybersecurity landscape as a strategic game, we can formally analyze both defenders' and adversaries' system vulnerabilities and weak points, particularly during the initial stages of ransomware file encryption. Viewing this through the lens of strategic games allows us to evaluate decision-making processes, simulate possible responses, and predict potential outcomes.

We operate under the assumption that online learning is particularly suitable for data streams. After conducting a systematic exploration of adversarial tactics and defensive strategies, our objective is to incorporate machine learning, specifically online learning algorithms, into backup systems and network file-sharing drives, including, but not limited to, NFS and SMB protocols. The model serves the purpose of detecting unauthorized encryption activities before the initiation of any protocol-based encryption processes. In the realm of file-level systems, this integration can be achieved through a variety of open-source file systems capable of embedding the proposed online incremental machine learning model. Prominent examples are FUSE (File system in Userspace) and Puffs (Pass-to-Userspace Framework File System), which are elaborated upon in [12] and [13], respectively.

3. Adversarial strategies

In this section, we delineate formal methodologies for comprehensively understanding the strategies employed in both adversarial and defensive scenarios, with a particular focus on sophisticated encryption techniques devised by ransomware developers.

3.1. Legitimate encryption process usage

The complexity of the game is accentuated by the attacker's strategy of co-opting legitimate system processes to carry out malicious activities. This deceptive approach, commonly termed a 'stealth attack,' obfuscates the distinction between benign and malicious operations, thereby posing a significant challenge for the defender in neutralizing the threat without adversely affecting regular system functions.

- Let us denote (E) as the set of legitimate encryption processes that can be used by the operating system to encrypt files and (R) as the ransomware process. Additionally, let (A) be the adversary or ransomware attacker, (D) as the system defender, and (P) as the set of all system processes. Furthermore, (M) is a mapping function that maps a process to its legitimacy status, such that ($M : P \rightarrow \{0, 1\}$), where '0' indicates malicious and '1' indicates benign.

In a typical scenario, a system defender (D) tries to neutralize malicious processes, i.e., for any process (p) in (P), if ($M(p) = 0$), then neutralize (p). However, the complexity arises when ransomware (R) employs legitimate processes (E) for encryption, i.e., ($R(E)$). In this case,

($M(R(E)) = 1$), making it seem benign to the system defender (D).

Given this, it becomes a challenge for the defender to distinguish between benign and malicious activities, as the ransomware attacker (A) has essentially turned the scenario into a game where (A) and (D) have conflicting objectives. While (A) tries to maximize the use of legitimate processes (E) for malicious deeds, (D) attempts to neutralize malicious processes without disrupting regular operations.

Therefore, there is a need for a more nuanced approach that can distinguish between the legitimate use of processes E and their malicious use by R , possibly by considering additional context or behavioral patterns. However, this task is non-trivial and represents a significant challenge in the current landscape of ransomware detection and neutralization.

3.2. Gradual write I/O to storage

Typically, defenders scrutinize file system operations, including reading, writing, deletion, and renaming, by utilizing predetermined threshold values as a means to detect ransomware activity. Nevertheless, ransomware employs a strategy that resorts to memory mapping to circumvent the file I/O read and write behaviors. In such circumstances, the existing countermeasures applied by defenders are prone to failure.

Memory mapping is a technique that allows programs to interact with data in storage as if it were in the computer's main memory. The operating system creates a mapping between the program's address space and the storage, within the context of virtual memory. This enables the program to use standard memory access instructions for file operations, potentially simplifying and improving efficiency over traditional file access methods.

- Let (O) be the set of file operations including reading, writing, deletion, and renaming, and (T) be the predetermined threshold values used to detect ransomware activity. Traditional detection methods focus on monitoring these operations such that if any operation (o) in (O) exceeds the corresponding threshold (t) in (T), an alarm is raised. Formally, for any operation (o) in (O), if ($o > t$), then an alarm is raised.

However, ransomware employs memory mapping which is a process by which the operating system creates a mapping (M) between the address space of the process (P) (the range of addresses that the process can use to address memory) and the storage object (S). This can be represented as ($M : P \rightarrow S$). When this mapping is in place, file I/O operations are transformed into memory access operations.

Consequently, if ransomware (R) utilizes memory mapping, the file I/O operations become memory operations and can evade detection by systems strictly monitoring file access. So for any file operation (o) in (O) performed by

ransomware (R), if (R) employs memory mapping, (o) is transformed into memory operation (m), i.e., ($o \rightarrow m$). Since defenders are monitoring (O) and not (m), these operations can potentially bypass detection.

There are number of research studies in [14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24] using monitoring methods often focus on file system changes, and when ransomware uses memory mapping, these file I/O operations are essentially translated into memory operations. This means that changes that would ordinarily be tracked (like file creation, deletion, renaming, and modifications) may not be detected, leading to a potential blind spot in ransomware detection mechanisms.

3.3. Partial data encryption

Ransomware strategies, distinguished by their swift and elusive encryption techniques, exhibit intriguing characteristics, such as the adaptable partial encryption of files exceeding 5.245 MB [25]. Although partial file encryption is not an innovative tactic - various ransomware programs deploy this to accelerate the process - the novel feature lies in its ability to specify the quantum of a file to encrypt. This capability bears significant implications for security programs that traditionally monitor file modifications to identify potential ransomware incursions. The ensuing fragmentation and probable low percentage of encrypted file content reduces the likelihood of detection by defenders.

This encryption methodology, along with other tactics employed by Royal ransomware, bears resemblance to Conti ransomware. For instance, the Conti ransomware also utilized 5.24MB as a threshold for partial encryption, subsequently segmenting the file into several equal parts, encrypting one part and leaving the next unencrypted. However, Conti's approach diverges by encrypting 50% of those segments. This partial encryption can be dynamic. Therefore, defensive strategies may fail if they rely on changes in file system attributes or CPU arithmetic operations like the Exclusive OR (XOR) logic gate.

The several potential modes of partial encryption used by real-world ransomware include:

- **Skip-step encryption mode:** Let F denote the file to be encrypted, where F_i represents the i^{th} megabyte (MB) of the file F . Let N and Y be non-negative integers. Then, in the skip-step mode, the file F is encrypted such that:

$$\forall i, F_i \text{ is encrypted} \iff i \bmod (N + Y) \geq N$$

This represents an encryption process that skips the first N MB, then encrypts the next Y MB, and so on.

- **Fast encryption mode:** Let F and F_i denote the same as above, and N be a non-negative integer. Then, in the fast mode, the file F is encrypted such that:

$$\forall i, F_i \text{ is encrypted} \iff i < N$$

This indicates an encryption process that encrypts only the first N MB of the file F .

- **Percent encryption mode:** Let (F) and (F_i) denote the same as above, and (N) be a non-negative integer, and (P) be a percentage ($0 \leq P \leq 100$). Also, let (S) be the size of the file in MB, and ($P_{MB} = \frac{P}{100} \times S$). Then, in the percent mode, the file (F) is encrypted such that:

$$\forall i, F_i \text{ is encrypted} \iff i \bmod (N + P_{MB}) \geq P_{MB}$$

This represents an encryption process that skips (P_{MB}) (which equals ($P\%$) of the total file size), then encrypts every (N) MB, and so on.

3.4. Intermittent encryption

Intermittent encryption strategies involve selectively encrypting data based on specific criteria. This approach allows for dynamic encryption, optimizing security and efficiency by adapting to varying contexts.

The intermittent encryption strategy [26], as used by Lock-File and Black Basta ransomware, significantly differs from the partial encryption techniques adopted by LockBit 2.0, Dark-Side, and BlackMatter. While partial encryption, targeting the initial segments of documents, aims to expedite the process, intermittent encryption focuses on encrypting alternate 16-byte segments of a file. This results in scrambled and untouched data alternating throughout the file.

Although this strategy is slower than partial encryption, it disrupts statistical analysis which is a key tool in ransomware detection. Detection programs typically block any process from modifying additional files if the statistical analysis test indicates encryption. With encrypted files appearing significantly different from unencrypted ones in statistical analysis, a clear difference can be seen in the chi-squared (χ^2) test scores or entropy value for files encrypted by ransomware.

The Intermittent Encryption strategy used by ransomware brings an additional layer of complexity for defenders where the ransomware encrypts pieces of a file at regular intervals. Ransomware can use the option to intermittently encrypt data, which is a feature that can be adjusted and customized.

For instance, BlackCat ransomware introduces a versatile implementation of intermittent encryption, enabling operators to select from an array of byte-skipping patterns. This flexibility allows for diverse encryption modes that can enhance the complexity and potentially the security of the encrypted data, such as:

- **SmartPattern [N,P]:** Encrypts N megabytes of the file in percentage steps. For example, as its default setting it starts from the file's header and encrypts 10 megabytes every 10%.
- **Auto mode:** This mode amalgamates multiple encryption methods for a more convoluted result. The encryption pattern in this mode is a complex function of the different encryption methods available, potentially including all the previously mentioned modes and their parameters.

Another intermittent encryption is Black Basta ransomware, a prominent figure in the cybercrime space that operates differently. Its strain of ransomware makes decisions based on the size of the file rather than offering operator-selected modes.

- **For small files:** For a file F of size S , F is encrypted if $S < 704$ bytes.
- **For medium size files:** For a file F with S in the range $704 \text{ bytes} \leq S < 4 \text{ KB}$, for every b^{th} byte F_b , F_b is encrypted if $b \bmod 256 < 64$.
- **For larger files:** For a file F with $S > 4 \text{ KB}$, for every b^{th} byte F_b , F_b is encrypted if $b \bmod 192 < 64$.

Notably, if the defender strategy is calibrated to react only to significant statistical differences to prevent false positives, it could fail to detect the encryption performed by ransomware. Additionally, the ransomware strategy complicates incident response efforts by deleting itself after completing the encryption process, making it difficult for defenders to locate a ransomware binary for analysis and system cleansing.

3.5. Adversaries avoidance of decoy techniques

In the ongoing struggle against nefarious file system activities, both commercial and research anti-ransomware strategies have adopted the use of deception-based methodologies, particularly the strategic placement of decoy files amongst authentic user files. This approach introduces an additional layer of complexity to ransomware detection efforts, predicated on the understanding that any interaction with a decoy file inherently signifies a malicious activity. However, in order to illustrate how ransomware can effectively circumvent existing deception-based detection strategies, researchers have put forth a proof-of-concept for anti-decoy ransomware in a scholarly publication [27]. This lab-developed ransomware is equipped with a decision engine that employs a minimal set of rules, thereby successfully evading decoys. Here an abstract analysis of the approach to bypass the decoy strategy can be used by an adversary.

- **Detecting static decoys through heuristics:** This strategy involves ransomware using heuristics to identify patterns that suggest the presence of decoy files. These patterns could be files filled with empty values or those with static creation dates and content. By fingerprinting these checks at run-time, ransomware can exclude these decoy files from the target files to encrypt. It notes that if ransomware mistakenly identifies a user file as a decoy and excludes it, the impact on the overall strategy is minimal as the ransomware will still encrypt other files. Two heuristic methods are suggested; one targets hidden and empty files, while the other aims at non-regular files like symbolic links or named pipes.
- **Distinguishing decoys using statistical methods:** This method is based on understanding file storage on the Windows operating system to discern file attributes and metadata. The paper mentions that decoy files, which are not typically accessed by users, will show different access patterns than genuine files. By analyzing these differences statistically, it's possible to distinguish between decoy and

genuine files. The technique uses 36 metrics for each directory based on file attributes and their standard deviations, creating a feature vector of 72 metrics. This statistical method seems effective at finding discrepancies, especially with time and date information.

- **Monitoring User to Reveal Non-decoy Files:** This technique focuses on monitoring user activities to identify genuine files. Two strategies are suggested. The first involves injecting a spy module into Explorer.exe to monitor which files are accessed by user applications. The second proposes enumerating all processes and injecting an interceptor module, replacing the WriteFile API with encryption routines.

3.6. File entropy and data manipulation strategy

The strategy employed by defenders to monitor potential shifts in file entropy, which might serve as indicators of ransomware's cryptographic activities, has been scrutinized and studied [5, 23, 28, 24]. In one study [8], it was claimed that no instance of ransomware was discovered that manipulates the entropy values of encrypted content. Nevertheless, we have highlighted certain ransomware families that utilize techniques such as partial encryption and intermittent encryption. Moreover, Maze ransomware [12], is known for its insertion of "Null" characters as a strategy to reduce the entropy of an encrypted file prior to its storage.

The referenced study [8] evaluates the entropy-based changes experienced by diverse file types as a result of Base64-Encoding and either partial or full encryption. Our extensive analysis suggests the existence of at least three methodologies by which file entropy values can be manipulated. This supposition adds another layer to the complex landscape of ransomware evasion tactics and challenges the robustness of entropy-based detection strategies. Therefore, an attacker's strategy might involve reducing the file entropy values after the encryption and encoding phases, while the defender could rely on measuring the uncertainty, unpredictability, or randomness of the file at the storage level.

3.6.1. Base64 encoding

Various studies have suggested approaches for countering ransomware detection by using a Base64 encoding algorithm to lower file entropy [7, 8]. The Base64 algorithm converts binary data into ASCII text by transforming it into a radix-64 format [29].

The objective of Base64 encoding is to transform a binary data stream D comprised of n 8-bit bytes b_1, b_2, \dots, b_n into an ASCII string S . The algorithm partitions the n bytes into blocks of three, concatenates each block to form a 24-bit block, and divides this 24-bit block into four 6-bit segments. Each 6-bit segment is then mapped to an ASCII character from the Base64 alphabet, usually composed of 64 printable ASCII characters ranging from 'A' to 'Z', 'a' to 'z', '0' to '9', '+', and '/'. Padding is applied when the last block contains fewer than 3 bytes, filling with zero bits and appending one or two '=' characters as needed.

Mathematically, for a data stream $D = b_1b_2 \dots b_n$, it is partitioned into k blocks D_1, D_2, \dots, D_k where $D_i = (b_{3i-2}, b_{3i-1}, b_{3i})$. Each D_i produces a 24-bit block $B_i = b_{3i-2} \parallel b_{3i-1} \parallel b_{3i}$, which is subdivided into four 6-bit segments $S_{i1}, S_{i2}, S_{i3}, S_{i4}$. These segments are mapped to ASCII characters $C_{ij} = \text{Base64}(S_{ij})$. The output S is the concatenation $C_{11}C_{12}C_{13}C_{14}C_{21}C_{22} \dots C_{k4}$. This facilitates the safe transfer of binary data over systems optimized for text-based data.

We conduct an experiment to determine the most effective strategy for an adversary aiming to reduce data entropy. But, first, it is well-known that Base64 encoding increases the size of the data by approximately 33% [30]. Our focus is on a sequential data transformation process aimed at entropy reduction. Initially, we gathered raw binary data from a dataset consisting of 3,200 files (8.13 GB) across multiple formats, including JPG, PDF, Microsoft documents, and TIFF. The entropy of this raw data is calculated using Shannon’s entropy formula to establish a baseline. The data then undergoes a first layer of transformation via Base64 encoding, which serves to standardize the data representation and generally aims to reduce its entropy. This stage is represented by the blue bar in Figure 1. Subsequently, the data is encrypted using AES, a step that significantly increases its entropy. This transformation is depicted by the red bar in Figure 1. Finally, we apply a combination of AES encryption and Base64 encoding. This multi-step process successfully reduces the entropy of the files, as illustrated by the purple bar in Figure 1. However, this results in a 33.38% increase in the original file size.

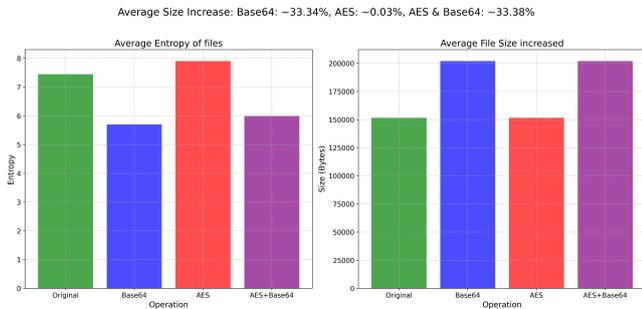


Figure 1: Base64 encoding reduces the entropy from 7.99 to 5.99. Adversary seeking to minimize entropy, a sequential AES → Base64 encoding approach is a tangible approach. However, this results in a 33.38% increase in the original file size.

In our empirical study, we observed that the Shannon entropy values for data processed through an Encrypt-Base64 encoding pipeline consistently fall within the range of 5.99 to 6.0. These findings imply that defensive mechanisms could leverage this specific range of entropy values to improve the detection of encrypted data stores. Figure 2 presents the Shannon entropy measurements across 3200 different types of files. But the question is, will adversaries risk incorporating Base64 encoding into their strategy, despite being aware that an increase in file size may potentially trigger other defense systems?

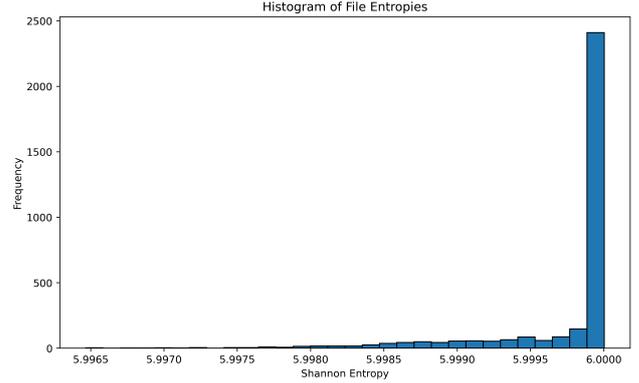


Figure 2: Base64 encoding reduces the entropy from 7.99 to 5.99. However, in a sequential AES → Base64 encoding approach, the Shannon entropy results fall in the range of 5.99 to 6.

3.7. Adversary falsifies the defender’s strategy by maintaining a uniform distribution of symbols

Many proposed models rely on measuring of file entropy for detecting ransomware [31, 32, 33, 7, 24, 34, 21]. Entropy in information theory facilitates the establishment of a uniform distribution of symbols, an essential precondition for ensuring high encryption security. Under ideal circumstances, every possible outcome within a uniform distribution should hold an equivalent likelihood, thereby fostering maximum unpredictability. If an encrypted file upholds this uniform distribution, it indicates successful randomization, making it challenging for cryptanalysis to detect patterns or infer original data.

However, failure to maintain a uniform distribution might suggest the existence of identifiable patterns in the encryption. These patterns could be manipulated by either a defender or an adversary, leading to decreased security. This is particularly critical when an adversary is attempting to implement robust encryption while preserving the ability to decrypt, highlighting an inherent paradox within the ransomware business model.

On the one hand, the attacker must implement robust encryption to prevent access to the victim’s data. Conversely, they must also retain the ability to reliably reverse this encryption when necessary (if and only if the victim pays the ransom). An inability to accomplish this dual objective threatens the viability of the ransomware business model, removing the victim’s incentive to pay the ransom.

An adversary’s strategic move in this confrontation often involves the use of partial or intermittent encryption, a tactic aiming to lower the entropy value while maintaining the legitimacy of the ransomware business model. This strategy’s successful execution could signify a potential loss for the defender in this strategic standoff.

4. Methodology

In our research, we meticulously aim to evaluate the compatibility and efficacy of various machine learning models, with a primary focus on file system data encryption realms. Our methodological framework is robustly constructed, guiding the

identification and validation of optimal machine learning models. These models are anticipated to make substantial contributions to future works aimed at detecting and mitigating the challenges posed by the cryptographic landscapes of ransomware.

Phase 1: Baseline Analysis with Traditional Machine Learning Models. Initially, we assess a range of popular non-online machine learning classifiers to understand their efficacy in differentiating between encrypted and unencrypted data buffers at the storage level [35]. The classifiers chosen for this phase represent a diverse set of well-established machine learning paradigms, as detailed in previous literature [36, 37]. Our primary goal here is to establish a baseline and comprehend the limitations of traditional machine learning techniques in real-time classification tasks.

Phase 2: Hypothesis Formation and Real-time Use Cases. Drawing from the foundational insights garnered in Phase 1, we have developed a hypothesis that online learning methods could be advantageous in real-time detection scenarios, specifically in identifying ransomware attacks (i.e., file encryption). We are exploring the possibility of deploying classifiers, based on online learning, in dynamic environments such as large corporate offices or multi-tenant cloud storage systems.

Phase 3: Selection of Online Learning Models. Subsequently, we narrow our focus to first-order online learning models, which promise the real-time adaptation essential for dynamic environments. For a more in-depth investigation, we select three leading online learning algorithms: Stochastic Gradient Descent (SGD), Perceptron, and Passive-Aggressive algorithms [38]. The choice of these models is not arbitrary but is influenced by the limitations we identified in traditional machine learning classifiers. To refine our model selection further, we also explore other compatible linear models and ensemble methods for online learning [36, 37].

Phase 4: Addressing the Challenges with Stream Data. We recognize that both traditional and online learning algorithms may still fall short in capturing the complexities of stream data, which is often encountered in the scenarios we consider. The dynamic nature of these scenarios—ranging from ransomware attacks to routine organizational data flow—calls for specialized algorithms designed to handle continuous data streams.

Phase 5: Final Model Selection. Given the challenges outlined in Phase 4, the Hoeffding algorithm is employed for our study. This decision tree learning method is specifically designed for classifying stream data. It provides a robust mechanism for real-time classification of encrypted and unencrypted data, thereby aligning with our objective for immediate detection and response in various scenarios.

Approach to handling file operations and the core components

Integrating DeltaFile Guard with the FUSE file system enables a sophisticated approach to handling file operations with

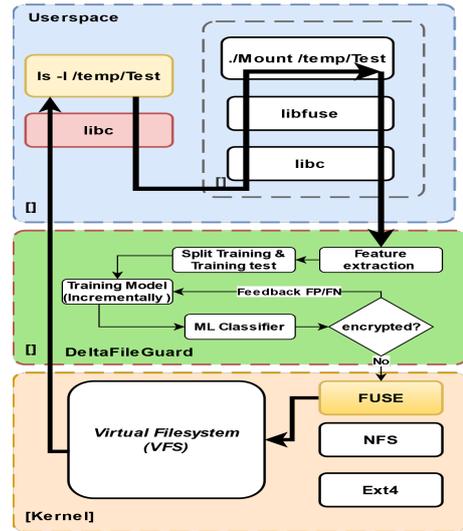


Figure 3: Conceptualize diagram: DeltaFile Guard represents the utilization of online incremental learning within the FUSE file system.

an added layer of intelligence. We have developed a Middle Layer component to accommodate online incremental learning illustrated in Figure 3. Here is a detailed breakdown of how this integration facilitates online incremental learning to predict the encryption status of files based on extracted features:

- **DeltaFile Guard:** This represents the specialized component designed to intercept file operations within the FUSE file system. Its main role is to apply machine learning algorithms for real-time analysis and prediction. DeltaFile Guard focuses on identifying whether files are encrypted, leveraging online incremental learning to adapt its models based on incoming data.
- **FUSE File System:** A flexible user space file system framework that allows custom file system operations to be implemented without altering the kernel. It serves as the foundation upon which DeltaFile Guard operates, providing the necessary infrastructure for file manipulation and access.

Workflow

As files are accessed by the write function within the FUSE file system, DeltaFile Guard intercepts these operations. This interception is crucial for analyzing file content and metadata without disrupting the user’s interaction with the file system.

- **Feature Extraction:** For each file operation, DeltaFile Guard extracts relevant features. These features are discussed in the following feature engineering section.
- **Online Incremental Learning:** Utilizing the extracted features, the embedded machine learning model within DeltaFile Guard performs real-time predictions to determine if a file is encrypted. The "online incremental" aspect refers to the model’s ability to learn and update its parameters dynamically as new data arrives, ensuring the system

evolves and adapts to new patterns or encryption methods over time.

- **Prediction and Action:** Based on the prediction outcome, DeltaFile Guard can take predefined actions. For encrypted files, it flags them for further review and trigger alerts. For non-encrypted files, it ensures they are handled per standard file system operations.

5. Empirical evaluation and results

5.1. Dataset

In this section, we implement a structured empirical approach to appraise the performance of a multitude of machine learning classifiers. We require an appropriate classification function with the intent of augmenting the probability of detecting encrypted data whilst concurrently reducing the occurrence of false positives and false negatives.

For this study, a comprehensive set of 75 unique ransomware families was curated from publicly available malware repositories. Utilizing controlled environments, each ransomware variant was executed against a dataset comprising approximately 32.6 GB, encompassing a total of 11,928 files. This dataset spans a broad array of file formats, including but not limited to mp3, mp4, docx, pptx, png, pdf, jpeg, gif, xls, and csv. The assembled dataset features a mixture of encrypted and non-encrypted files, designed to mimic a genuine computational setting. Notably, the encrypted portion of this dataset has been generated using real-world ransomware, ensuring the inclusion of the latest ransomware families in our analysis.

5.2. Ransomware dataset

In the course of our research, we amassed a comprehensive collection of ransomware samples sourced from various public repositories, including but not limited to VirusShare, MalwareBazar, MalwareDB, and the Zoo. Our ensemble consists of approximately 1,500 distinct samples. To gain insights into the characteristics and behavior of these malicious entities, we employed both static and dynamic analysis techniques. Prior to in-depth analysis, each sample underwent rigorous testing to ensure its ability to execute, thereby allowing us to accurately observe and document its inherent behavior.

5.2.1. System specification

Our experimental environment is powered by the AMD Ryzen ThreadRipper Pro 5975WX 32-core, paired with 32 GB of RAM and a 1TB SSD, providing ample processing power and storage capacity for handling extensive datasets and complex models. This setup operates on Windows 11 and leverages Python 3.10 and the Spyder integrated development environment (IDE) for seamless machine learning code development and experimentation.

5.3. Feature engineering

We preprocess the data by extracting relevant features from the files, preprocessing data through extraction of pertinent features plays a significant role in enabling classifiers to effectively distinguish between encrypted and unencrypted files. Each selected feature offers a distinctive perspective on the characteristics of the file content, thus providing the classifier with a comprehensive understanding of its structure and composition.

Our approach seeks to augment traditional detection methods, which primarily focus on entropy and file size, with a suite of more nuanced features. Traditional methods are increasingly circumvented by advanced encryption schemes designed to evade detection, highlighting the need for a broader set of characteristics that can reveal the presence of encryption. While file size and byte entropy are foundational, they offer limited insight into sophisticated evasion techniques. Our inclusion of content-based features such as byte frequency, variance, kurtosis, skewness, and the analysis of strings and file content patterns, delves deeper into the structural intricacies of files. These features are designed to detect subtle anomalies and patterns (or their absence) indicative of encryption, thereby providing a more robust framework for distinguishing encrypted files from their unencrypted counterparts.

Moreover, our methodology introduces novel features such as entropy variance and percentiles utilization, which are not commonly employed in existing solutions. The entropy variance feature aims to capture the uniformity of randomness across a file, a characteristic trait of encrypted content, while percentiles utilization examines the distribution of byte values at specific intervals, offering insights into the uniformity expected from encrypted files versus the varied distributions of plaintext files. These novel features, particularly when combined with traditional metrics, enhance the detection of encrypted files by identifying characteristics that evade simpler detection methods. This comprehensive approach not only addresses the limitations of relying solely on entropy and file size but also sets a new standard for encrypted file detection by incorporating a more detailed analysis of file characteristics, thereby improving the ability to detect advanced encryption schemes.

1. **File Size:** This attribute provides a trivial understanding of a file's intricacy and potential data volume. Notably, encrypted files, contingent on the applied encryption methodology, may exhibit unique size patterns that starkly contrast those of their unencrypted counterparts. Later in the context of online learning, we opt to exclude this feature since the file size may be subject to constant changes.
2. **Byte Entropy:** This feature, indicative of a file's randomness or unpredictability, generally presents higher values in encrypted files. The calculation of byte entropy serves as a tool for the identification of encrypted files due to their inherently unpredictable nature.

$$H(X) = - \sum_{i=1}^n p(x_i) \log_b p(x_i) \quad (1)$$

Where X is a random discrete variable, x_i is a possible value of X , $p(x_i)$ is the probability mass function of X at x_i , and b is the base of the logarithm.

3. **Content-Based Features:** Including the frequency or scarcity of specific characters or sequences within file content, these features aid in unveiling patterns or consistencies present in unencrypted files but absent in encrypted ones.

Byte frequency:

$$f_i = \frac{\text{count}(x_i)}{N} \quad (2)$$

In Byte frequency, x_i is a byte value and N is the total number of bytes.

4. **Byte Variance:** As a measure of dispersion, byte variance can offer valuable insights into the range of byte values in a file. The high variance may suggest a broader range of byte values, a characteristic potentially indicative of encrypted files.

$$\sigma^2 = \frac{1}{N} \sum_{i=1}^N (x_i - \mu)^2 \quad (3)$$

5. **Byte Kurtosis:** This measure of data point distribution can provide a statistical fingerprint of encrypted data. Elevated kurtosis values may denote a distribution with pronounced tails or sharper peaks, characteristics potentially representative of encrypted data. In other words, elevated Byte Kurtosis in an unencrypted file may hint at a concentrated distribution of byte values. In contrast, for an encrypted file, Byte Kurtosis would typically approach zero, reflecting the uniform distribution of byte values.

$$K = \frac{N(N+1)}{(N-1)(N-2)(N-3)} \frac{1}{N} \sum_{i=1}^N \left(\frac{x_i - \mu}{\sigma} \right)^4 - \frac{3(N-1)^2}{(N-2)(N-3)} \quad (4)$$

6. **Byte Skewness:** This measure of distributional asymmetry can assist in differentiating between the recurring patterns found in unencrypted data and the randomness intrinsic to encrypted data. Encrypted data, due to its random characteristics, is anticipated to exhibit a skewness value approximating zero, while unencrypted data may manifest significant skewness if it contains repeated elements or patterns.

$$S = \frac{N(N-1)}{(N-2)} \frac{1}{N} \sum_{i=1}^N \left(\frac{x_i - \mu}{\sigma} \right)^3 \quad (5)$$

In Variance, skewness, and kurtosis of byte content, x_i is the byte value, N is the total number of bytes, μ is the mean of the byte values, and σ is the standard deviation of the byte values.

7. **Strings (Average String Length):** The diversity in string lengths within unencrypted files can provide hints regarding their nature. Encrypted files may manifest a uniform distribution of strings, whereas unencrypted files may exhibit varied string lengths attributable to the presence of natural language structures or formatting.

$$\text{avg_string_length} = \frac{1}{N} \sum_{i=1}^N |s_i| \quad (6)$$

In Average string length, s_i is a string in the file and N is the total number of strings.

8. **File Content Analysis:** This feature encompasses a detailed examination of the file content, such as the frequency of specific patterns or the presence of distinct structures (e.g., headers, footers, metadata), providing additional clues that can assist in differentiating between encrypted and unencrypted files.

$$\text{avg_word_length} = \frac{1}{N} \sum_{i=1}^N |w_i| \quad (7)$$

In Average word length, w_i is a word in the file and N is the total number of words.

9. **Entropy Variance:** By measuring the fluctuation in randomness across different sections of a file, this feature can aid in identifying encrypted files. Encrypted files may exhibit a comparatively consistent entropy level, whereas unencrypted files may show greater entropy variability. Note that the entropy is a measure of the disparity of the density function $f(x)$ from the uniform distribution. On the other hand, the variance measures an average of distances of outcomes of the probability distribution $f(x)$ from the mean [39].

$$\sigma_H^2 = \frac{1}{N} \sum_{i=1}^N (H(X_i) - \mu_H)^2 \quad (8)$$

In Entropy variance, $H(X_i)$ is the entropy of the i -th 512-byte block, N is the total number of 512-byte blocks, and μ_H is the mean entropy of the blocks.

10. **Percentiles utilization:** We assume that percentiles serve to appraise the dispersion of byte values within a file. Within a data cohort, the Nth percentile is characterized as the value beneath which N percent of the data resides. Hence, the 25th, 50th, and 75th percentiles epitomize the values beneath which 25%, 50%, and 75% of the data reside, respectively. Plaintext files predominantly possess a structured framework, engendering the recurrence of particular byte values. For instance, within a text file, ASCII values corresponding to common alphabets and spaces occur with high frequency. In the event of plotting the distribution of byte values, one would observe peaks at certain values, thereby inducing a skew in the distribution. As a consequence, the 25th, 50th, and 75th percentiles in a plaintext file may not adhere to a uniform distribution owing to these peaks and troughs.

Encryption algorithms are conceived to metamorphose plaintext into a semblance that radiates randomness to any entity devoid of the decryption key. Optimally, this transformation engenders a uniform distribution of byte values spanning the feasible range (0 to 255 for 8-bit bytes). In such a distribution, the likelihood of each byte value is equitably distributed, and thus, the distribution’s percentiles would settle at regular intervals across this range. To elucidate:

- The 25th percentile would approximate the value 64 (since 25% of 256 is 64).
- The 50th percentile (also recognized as the median) would approximate the value 128 (since 50% of 256 is 128).
- The 75th percentile would approximate the value 192 (since 75% of 256 is 192).

This percentile analysis conjecturally can offer a proper feature to distinguish between normal and encrypted files predicated on the distribution of their byte values at the file level system.

5.3.1. Feature extraction computational resources

The process of extracting statistical features from individual files, as detailed in the data, involves varying degrees of computational resource utilization, which is dependent on the file type and whether it is in its normal or encrypted state. For normal files, the resource demand is largely influenced by the file’s size and type. For example, feature extraction from a large Excel file is considerably resource-intensive, requiring up to 441.827 seconds and utilizing 0.0352 MB of memory. Conversely, simpler files such as JPEG images and mp3 audio files demand significantly less computational power, with JPEG extraction taking a mere 0.0090 seconds and mp3 files requiring 0.927 seconds, both utilizing a minimal memory footprint of 0.0039 MB.

When files are encrypted by ransomware, the computational requirements for feature extraction exhibit slight variations. Some file types show a decrease in processing time, albeit with an increment in memory usage. For instance, the memory requirement for an encrypted Excel file slightly increases to

0.0391 MB. Table 1 illustrates the differences in time and memory usage for extracting information from various file types, both in their normal state and when encrypted by ransomware.

Table 1: Computational resources required for feature extraction from normal and encrypted files by ransomware.

| File Type | Size | Time (Second) | Memory Usage (MB) |
|-----------------|---------|---------------|-------------------|
| PDF Normal | 82.1 MB | 11.526 | 0.0039 |
| mp4 Normal | 182 MB | 26.845 | 0.0039 |
| JPEG Normal | 23.7 KB | 0.0090 | 0.0039 |
| Excel Normal | 3.58 GB | 441.827 | 0.0352 |
| Doc Normal | 89.8 MB | 12.772 | 0.0039 |
| mp3 Normal | 6.59 MB | 0.927 | 0.0039 |
| PDF Encrypted | 82.1 MB | 11.215 | 0.0156 |
| mp4 Encrypted | 182 MB | 26.069 | 0.0155 |
| JPEG Encrypted | 24.0 KB | 0.0040 | 0.0039 |
| Excel Encrypted | 3.58 GB | 421.884 | 0.0391 |
| Doc Encrypted | 89.8 MB | 12.760 | 0.0195 |
| mp3 Encrypted | 6.59 MB | 0.924 | 0.0039 |

5.4. Benchmarking supervised learning algorithms for classifications: A comparative analysis

In the empirical analysis, we employed a series of batch classifiers, as delineated in the preceding section, to assess the efficacy of both traditional shallow and deep learning algorithms in discerning encrypted from non-encrypted files across multiple file formats. Subsequently, the feature vectors were extracted for both the normal and encrypted files using our supervised machine learning tool, leading to a comprehensive dataset consisting of 65.1 GB (70,008,138,878 bytes) files.

Quantitative metrics are showcased in Table 2, furnishing a comparative performance analysis of the various machine learning classifiers under review. These metrics encompass Accuracy, Precision, Recall, and F1-score, providing a thorough understanding of each model’s performance capabilities. Notably, the training of these classifiers was rigorously based on the feature variables outlined in section 5.3. Our findings suggest that certain machine learning classifiers, particularly Random Forest, Decision Trees, AdaBoost, and Gradient Boosting, offer exceptional capabilities in the classification of files based on their encryption status, thereby providing valuable insights into effective strategies for the detection of ransomware-encrypted files. We partitioned the dataset, allocating 70% for training and the remaining 30% for evaluation. However, our assumption is that online learning, when incorporated with Decision Trees, can be suitable in such a file system (i.e., data stream) environment for learning and predicting encrypted and unencrypted buffers.

The necessity for online incremental learning approaches to improve performance, even with such high accuracy rates presented in Table 2, can be justified on several fronts:

1. **File system Dynamic Environments:** In practical scenarios, the characteristics of files, user behavior, and system environments are not static. An online learning approach can adjust to changes in the environment that might affect the model’s performance, such as new file types or legitimate encryption practices by users, reducing false positives and improving detection accuracy over time.

Table 2: The table presents a comparative assessment of multiple machine learning classifiers, including Logistic Regression, Support Vector Machines (SVM), Decision Trees, Random Forests, k-Nearest Neighbors, AdaBoost, Gradient Boosting, Multilayer Perceptron, and Naive Bayes, in classifying files as encrypted or normal. Decision Trees emerged as the most accurate classifier, with an accuracy of 98.08% and an F1-score of 98.20%. Random Forests and Gradient Boosting also showcased robust performance, closely following Decision Trees with comparable precision and recall. These results underscore the effectiveness of Decision Trees, Random Forests, and Gradient Boosting in maintaining a harmonious balance between precision and recall, affirming their position as leading classifiers in this specific classification task.

| Classifier | Accuracy | Precision | Recall | F1 |
|-----------------------|---------------|---------------|---------------|---------------|
| Logistic Regression | 0.9155 | 0.8748 | 0.9878 | 0.9246 |
| SVM | 0.9502 | 0.9324 | 0.9793 | 0.9535 |
| Decision Trees | 0.9808 | 0.9496 | 0.9880 | 0.9820 |
| Random Forests | 0.9806 | 0.9703 | 0.9947 | 0.9786 |
| k-Nearest Neighbors | 0.9353 | 0.9226 | 0.9591 | 0.9385 |
| AdaBoost | 0.9719 | 0.9607 | 0.9879 | 0.9732 |
| Gradient Boosting | 0.9800 | 0.9686 | 0.9941 | 0.9808 |
| Multilayer Perceptron | 0.7897 | 0.9160 | 0.9801 | 0.8395 |
| Naive Bayes | 0.8918 | 0.8346 | 0.9967 | 0.9055 |

- Model Drift:** Over time, the performance of models trained in an offline setting might degrade due to model drift. This phenomenon occurs when the underlying data distribution changes, making previously learned patterns less applicable. Online learning continuously updates the model, mitigating the effects of model drift and maintaining high accuracy levels.
- Resource Efficiency:** Online learning models can be more resource-efficient in certain contexts. They require less memory and computational power to update with new data, compared to retraining an entire offline model with a growing dataset. This efficiency is particularly valuable in environments with limited resources.

5.5. Exploring the efficacy of incremental learning in predicting encrypted and unencrypted buffer at file system

In traditional machine learning paradigms, particularly within supervised learning contexts, the modus operandi is predominantly batch-oriented. In this framework, an accumulated dataset serves as the initial substrate for the algorithmic training process. This necessitates the availability of the entire training dataset in advance of the model’s learning phase, often relegating the training process to offline execution due to the substantial computational and temporal overheads involved [38]. Despite their ubiquity, batch learning techniques are fraught with inherent limitations, including (a) suboptimal efficiency concerning both time and computational resources, and (b) a lack of scalability, especially in large-scale applications where the introduction of new data typically mandates a comprehensive retraining of the existing model.

Incremental online learning represents a significant advancement in machine learning, offering unparalleled adaptability, efficiency, and real-time processing capabilities, especially in dynamic environments like file systems. This document addresses the concerns regarding its advantages over traditional ML schemes.

- Adaptability to New Data:** Incremental online learning algorithms are designed to adapt to new data in real-time or near-real-time, without the need for retraining the model from scratch [40]. This is particularly advantageous in dynamic environments like file systems, where new file types may be introduced over time, and the system must quickly adapt to recognize and process these new structures. Traditional ML schemes, in contrast, often require batch processing and retraining on the entire dataset, including both old and new data, which is computationally expensive and time-consuming.
- Efficiency in Handling Large Datasets:** File systems typically contain a vast amount of data. Incremental online learning algorithms can process data in smaller chunks, making them more memory-efficient than traditional approaches that may require the entire dataset to be loaded into memory. This efficiency is crucial for learning file types’ structures and predicting data encryption status, where models must scale to accommodate large datasets without significant increases in computational resources.
- Real-time Prediction and Detection:** The ability to update models incrementally allows for real-time prediction and detection of encrypted versus unencrypted data. This is essential for cybersecurity applications where timely detection of encrypted files (potentially indicative of ransomware activity) is critical. Traditional ML models, due to their batch-learning nature, cannot easily provide real-time insights or adapt quickly to newly emerging encryption patterns.
- Experimental Comparison and Justification:** To empirically demonstrate the advantages of incremental online learning, we propose a comprehensive experimental comparison involving several datasets representative of real-world file system structures, including a mix of encrypted and unencrypted data.
- Application-Specific Benefits:** For file system monitoring and encryption detection, the incremental learning approach allows for continuous learning from incremental changes in file structures or content, enhancing the model’s ability to detect subtle shifts towards encryption patterns. This contrasts with traditional ML models that might miss or delay detecting these patterns due to the static nature of their training process.

In contrast, online learning provides a more adaptive machine learning paradigm well-suited for data streams that are sequentially ordered. The underlying objective is the iterative refinement and dynamic updating of the model to maximize predictive performance on future, as-yet-unseen data. This adaptability effectively ameliorates the limitations of batch learning by enabling incremental model updates as new instances of data become available. Consequently, online learning algorithms demonstrate enhanced efficiency and scalability, attributes particularly beneficial in large-scale, real-world data analytics sce-

narios where data is not only voluminous but also arrives at a high velocity [38] [41].

5.6. Computational cost comparison between the traditional and online learning models

The comparison between the traditional and the online learning models, based on evaluations conducted using our full dataset, reveals significant differences in computational efficiency and resource utilization. The Traditional Model, while delivering high accuracy, requires a considerably longer CPU time of approximately 3.76 seconds and consumes around 7.47 MB of memory. This indicates a substantial computational effort and resource allocation to achieve its predictive performance, highlighting the model’s reliance on the extensive dataset for training and evaluation. On the other hand, the online learning model stands out for its remarkable efficiency, needing only about 0.27 seconds of CPU time and a minimal memory usage of 0.016 MB. Despite the differences in accuracy metrics, the online learning model’s performance showcases its potential for applications requiring rapid responses and minimal resource consumption. The utilization of the full dataset for obtaining these results underscores the efficiency of the online learning model in handling large volumes of data swiftly and with significantly lower resource requirements, making it particularly suitable for real-time or resource-constrained environments. Table 3 compares the performance of a traditional machine learning model, specifically, a Random Forest Classifier, and an online learning model, specifically, SGDClassifier in terms of CPU time and memory usage.

Table 3: Computational costs: The traditional model is more resource-intensive, while the online learning model is highly efficient, demonstrating lower CPU time and minimal memory usage, highlighting its suitability for real-time applications with strict resource constraints.

| Model | CPU Time (s) | Memory Usage (MB) |
|-----------------|--------------|-------------------|
| Traditional | 3.755 | 7.473 |
| Online Learning | 0.266 | 0.016 |

5.6.1. Online learning benchmarks

To rigorously assess the most effective machine learning classifiers for distinguishing between encrypted and unencrypted files, we conducted a comprehensive empirical study using a large and diverse dataset. The dataset featured 11,928 files, totaling 32.6GB in size, encrypted through various techniques from 75 unique ransomware families. Noteworthy among these were sophisticated encryption strategies like partial and intermittent encryption, commonly employed by ransomware variants such as Black Basta, BlackMatter, Grand Crab, lab-developed ransomware AES-Base64, Paradise, and LockBit 3.0.

5.7. First-order online learning

In the context of online learning, First-order algorithms are particularly useful for large-scale learning tasks and data streams due to their incremental model updating using only

gradient information (i.e., first-order derivative). In our investigation, we leveraged conventional machine learning classifiers that fall under the category of first-order online learning algorithms. These include a *Stochastic Gradient Descent (SGD)* (logistic regression as the linear classifier), *Passive-Aggressive techniques*, and the *Perceptron model*. These algorithms were strategically employed for predictive analytics on a file system-based dataset, enabling us to rigorously explore the inherent differences between categories of encrypted and unencrypted data within a complex data corpus for large-scale learning tasks and data streams.

5.7.1. Mathematical formulation in an online learning context

Let $\mathcal{D} = \{(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)\}$ represent a heterogeneous dataset sourced from a file system. Here, each x_i denotes a data buffer, and $y_i \in \{0, 1\}$ signifies whether the buffer is encrypted (1) or unencrypted (0). We applied machine learning classifiers, particularly focusing on algorithms such as Stochastic Gradient Descent (SGD), Passive-Aggressive algorithms, and the Perceptron model. The algorithms aim to minimize the objective function $J(\theta)$ concerning the model parameters θ .

For the SGD algorithm, the update rule at time t is as follows:

$$\theta_{t+1} = \theta_t - \alpha_t \nabla J(\theta_t)$$

In the case of the Passive-Aggressive algorithm, the update rule can be defined as:

$$\theta_{t+1} = \theta_t + \tau_t (y_t - \text{sgn}(\theta_t^T x_t)) x_t$$

Here, τ_t is selected to minimize $J(\theta)$, subject to specific constraints.

For the Perceptron algorithm, the update occurs conditionally upon an error:

$$\theta_{t+1} = \theta_t + \alpha_t (y_t - \text{sgn}(\theta_t^T x_t)) x_t$$

This update is executed only when $y_t \neq \text{sgn}(\theta_t^T x_t)$.

utilizing these algorithms, we continually adjust θ with each incoming sample (x_t, y_t) .

The dynamic adaptability and real-time learning capabilities of these algorithms render them highly effective for the real-time detection of encrypted versus unencrypted data buffers at the file system level. The methodology thus provides a detailed and nuanced understanding of the intrinsic variances between encrypted and unencrypted data, making it possible to detect encrypted buffers in real time. This paves the way for detecting between encrypted and unencrypted data buffers.

5.8. Hoeffding Tree algorithm and key advantages

The Hoeffding Tree Algorithm, also known as VFDT (Very Fast Decision Tree Learner), serves as a cutting-edge approach for the incremental induction of decision trees from unbounded data streams. This algorithm is designed to analyze each incoming data instance exactly once, thereby eliminating the need to store past instances once they have been incorporated into the evolving decision tree. Within this computational framework,

the only entity that necessitates in-memory storage is the decision tree itself, which accumulates relevant metadata within its terminal nodes to not only allow for future growth but also to enable real-time predictive inference. The mathematical robustness of the Hoeffding Tree Algorithm has been formally corroborated by Domingos and Hulten, who have demonstrated that the algorithm produces decision trees closely approximating those generated through conventional batch learning methods [42]. This finding lends empirical weight to the argument that the Hoeffding Tree is highly efficacious in producing high-quality decision trees, thereby aligning with the prevailing consensus that batch-learned decision trees are among the most effective model architectures in the field of machine learning. Here are some key advantages, especially in scenarios involving streaming data, such as a file system space:

- **Adaptability to Heterogeneous Data:** In environments where data characteristics can change over time due to various factors such as user behavior, system updates, or evolving threat landscapes, the ability of online learning models like the Hoeffding tree to adapt and learn incrementally from each new batch of data becomes invaluable. This adaptability ensures that the model remains effective even as the nature of the data it analyzes changes, providing a level of resilience that static offline models may lack.
- **Resource Efficiency:** One of the significant advantages of the Hoeffding tree, highlighted in the performance metrics, is its efficiency in terms of CPU and memory usage. This efficiency is critical in real-time file system environments where resources are limited, and the overhead of running complex detection models can be prohibitive. The ability of the Hoeffding tree to deliver effective performance while minimizing resource consumption makes it particularly suitable for deployment in such contexts, where maintaining system performance and responsiveness is paramount.
- **Real-Time Detection Capabilities:** The Hoeffding tree’s online incremental learning approach enables it to update its knowledge base in real-time as new data arrives. This capability is crucial for the timely detection of ransomware attacks, where early detection and response can significantly mitigate the impact. In contrast, offline models, despite their high accuracy, require periodic retraining and deployment, which can introduce delays in adapting to new threats.
- **Continuous Improvement and Learning:** Although the performance of the Hoeffding tree may appear to stabilize after a certain point, this does not preclude the potential for future improvements as more diverse data is encountered. The model’s design to shuffle and learn from the data in a way that reflects the heterogeneous nature of file systems ensures that it remains sensitive to subtle changes in data patterns that could indicate new or evolving threats.
- **Scalability:** The scalability of online learning models like the Hoeffding tree, given their incremental nature, is an-

other factor that contributes to their standout performance. As the volume of data in file systems grows, the model can continue to learn and adapt without the need for complete retraining, making it well-suited to environments with expanding data volumes.

5.9. Online learning evaluations and results

We have developed a tool designed to evaluate and compare online learning algorithms, specifically addressing the challenges presented by large-scale or real-time data within environments like enterprise-level corporate offices or multi-tenant cloud storage. This tool employs an incremental learning paradigm, leveraging a sliding window mechanism to manage the data files. This operational approach enables the incremental training of classifiers such as SGD (Stochastic Gradient Descent), Perceptron, Passive Aggressive, and Hoeffding Tree, making it exceptionally suitable for situations where the data’s size prohibits in-memory storage or when data is generated in real-time, such as at the file system level. By reading, processing, and using a subset of the data for model training during each computational iteration, the tool overcomes limitations related to computational resources. This strategy allows for the dynamic ingestion of data, ensuring that the system remains efficient and effective even as data volumes grow or as new data streams in continuously.

To assess the effectiveness of online learning models, we encrypted a dataset with 75 ransomware families using a lab-developed AES-Base64 encryption-encoding model. The results, presented in Table 4, showcase the performance of these online learning models when confronted with well-known ransomware, alongside the encryption-encoding model. Here brief descriptions of each online learning evaluation conducted against the encrypted dataset are provided.

GrandCrab Ransomware evaluation: GandCrab, a notorious ransomware, has significantly impacted global cybersecurity by encrypting user files and demanding ransom payments for decryption keys. It distinguished itself through its ability to evade detection with frequent updates and a variety of distribution methods. In our study, we encrypted data using the latest GandCrab variant to test the efficacy of online learning models. The results illustrated that these models frequently achieve over 90% accuracy in distinguishing between normal and encrypted files. Particularly noteworthy is the performance in Batch 4, where the Hoeffding Tree classifier achieved perfect classification (accuracy = 100%). This suggests that the Hoeffding Tree is exceptionally effective at capturing the underlying data distribution, making it a potent tool for identifying and mitigating threats posed by sophisticated ransomware like GandCrab. Result illustrated in Table 4 row A.

AES-Base64 encryption-encoding evaluation (lab-developed): One malicious method to reduce entropy involves using AES-Base64 for encryption and encoding. We developed a tool for encrypting and encoding data into Base64 and evaluated its effectiveness with online learning algorithms. Our empirical experiment, detailed in the second

Table 4: Comparative Performance of Online Learning Classifiers against Adversarial Encryption Techniques. This table illustrates the incremental evaluation results of various classifiers, with data fed in rounds of 1000, in a simulated file system-like environment. Four encryption techniques were analyzed: Grand Crab Ransomware’s Strong Encryption, AES-Base64’s Suggested Entropy Reduction, Paradise’s Partial Encryption, and CryptoFile2 ransomware which uses encrypted by a strong encryption with RSA-2048.

A particular focus is placed on the standout performance of the Hoeffding classifier, which exhibited superiority across all tested contexts. **Row A** illustrates the results from data encrypted with GrandCrab Ransomware. Row B illustrates data first encrypted with the AES algorithm and then encoded using Base64 encoding. **Row C** illustrates the results from data encrypted by Paradise ransomware, which uses partial encryption. **Row D** illustrates the results from encrypted data by CryptFile2 ransomware. The Hoeffding algorithm shows outperforming accuracy results compared to other online learning algorithms.

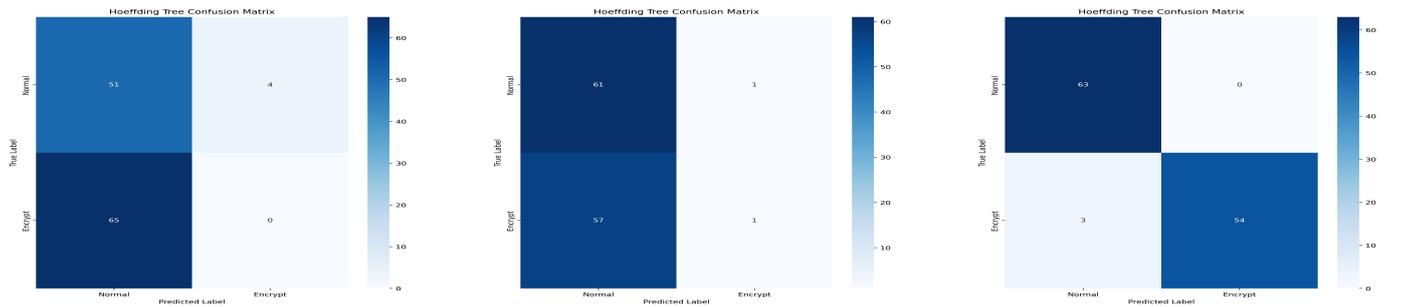
| # | Comparison of Classifier Performance Graph | Comparison of Classifier Performance (Accuracy) | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|----------|--|---|--------------------|----------------|------------|--------------------|----------------|---|--------|--------|--------|---------------|---|--------|--------|--------|---------------|---|--------|--------|--------|---------------|---|--------|--------|--------|---------------|---|--------|--------|--------|---------------|---|--------|--------|--------|---------------|---|--------|--------|--------|---------------|---|--------|--------|--------|---------------|---|--------|--------|--------|---------------|----|--------|--------|--------|---------------|----|--------|--------|--------|---------------|----|--------|--------|--------|---------------|
| A | | <table border="1"> <thead> <tr> <th>Batch</th> <th>SGD</th> <th>Perceptron</th> <th>Passive Aggressive</th> <th>Hoeffding Tree</th> </tr> </thead> <tbody> <tr><td>1</td><td>0.4917</td><td>0.35</td><td>0.6333</td><td>0.9250</td></tr> <tr><td>2</td><td>0.4917</td><td>0.4667</td><td>0.65</td><td>0.9833</td></tr> <tr><td>3</td><td>0.5250</td><td>0.6750</td><td>0.7833</td><td>0.9833</td></tr> <tr><td>4</td><td>0.4167</td><td>0.3333</td><td>0.7417</td><td>1.0000</td></tr> <tr><td>5</td><td>0.7333</td><td>0.2667</td><td>0.6333</td><td>0.9833</td></tr> <tr><td>6</td><td>0.4917</td><td>0.5750</td><td>0.6833</td><td>0.9000</td></tr> <tr><td>7</td><td>0.6000</td><td>0.5167</td><td>0.7250</td><td>0.9167</td></tr> <tr><td>8</td><td>0.7250</td><td>0.3667</td><td>0.5167</td><td>0.9667</td></tr> <tr><td>9</td><td>0.4167</td><td>0.6250</td><td>0.5750</td><td>0.9583</td></tr> <tr><td>10</td><td>0.5083</td><td>0.5583</td><td>0.6833</td><td>0.9583</td></tr> <tr><td>11</td><td>0.5000</td><td>0.7250</td><td>0.6833</td><td>0.9250</td></tr> <tr><td>12</td><td>0.3839</td><td>0.5268</td><td>0.8036</td><td>0.9554</td></tr> </tbody> </table> | Batch | SGD | Perceptron | Passive Aggressive | Hoeffding Tree | 1 | 0.4917 | 0.35 | 0.6333 | 0.9250 | 2 | 0.4917 | 0.4667 | 0.65 | 0.9833 | 3 | 0.5250 | 0.6750 | 0.7833 | 0.9833 | 4 | 0.4167 | 0.3333 | 0.7417 | 1.0000 | 5 | 0.7333 | 0.2667 | 0.6333 | 0.9833 | 6 | 0.4917 | 0.5750 | 0.6833 | 0.9000 | 7 | 0.6000 | 0.5167 | 0.7250 | 0.9167 | 8 | 0.7250 | 0.3667 | 0.5167 | 0.9667 | 9 | 0.4167 | 0.6250 | 0.5750 | 0.9583 | 10 | 0.5083 | 0.5583 | 0.6833 | 0.9583 | 11 | 0.5000 | 0.7250 | 0.6833 | 0.9250 | 12 | 0.3839 | 0.5268 | 0.8036 | 0.9554 |
| Batch | SGD | Perceptron | Passive Aggressive | Hoeffding Tree | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 1 | 0.4917 | 0.35 | 0.6333 | 0.9250 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 2 | 0.4917 | 0.4667 | 0.65 | 0.9833 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 3 | 0.5250 | 0.6750 | 0.7833 | 0.9833 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 4 | 0.4167 | 0.3333 | 0.7417 | 1.0000 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 5 | 0.7333 | 0.2667 | 0.6333 | 0.9833 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 6 | 0.4917 | 0.5750 | 0.6833 | 0.9000 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 7 | 0.6000 | 0.5167 | 0.7250 | 0.9167 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 8 | 0.7250 | 0.3667 | 0.5167 | 0.9667 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 9 | 0.4167 | 0.6250 | 0.5750 | 0.9583 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 10 | 0.5083 | 0.5583 | 0.6833 | 0.9583 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 11 | 0.5000 | 0.7250 | 0.6833 | 0.9250 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 12 | 0.3839 | 0.5268 | 0.8036 | 0.9554 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| B | | <table border="1"> <thead> <tr> <th>Batch</th> <th>SGD</th> <th>Perceptron</th> <th>Passive Aggressive</th> <th>Hoeffding Tree</th> </tr> </thead> <tbody> <tr><td>1</td><td>0.5250</td><td>0.3833</td><td>0.5583</td><td>0.5333</td></tr> <tr><td>2</td><td>0.6333</td><td>0.5417</td><td>0.6500</td><td>0.6083</td></tr> <tr><td>3</td><td>0.4917</td><td>0.6500</td><td>0.6667</td><td>0.9750</td></tr> <tr><td>4</td><td>0.5833</td><td>0.6500</td><td>0.5750</td><td>0.9833</td></tr> <tr><td>5</td><td>0.6000</td><td>0.6083</td><td>0.5500</td><td>1.0000</td></tr> <tr><td>6</td><td>0.6167</td><td>0.4250</td><td>0.6167</td><td>0.9750</td></tr> <tr><td>7</td><td>0.5083</td><td>0.4500</td><td>0.5250</td><td>0.9833</td></tr> <tr><td>8</td><td>0.4583</td><td>0.5333</td><td>0.6083</td><td>0.9750</td></tr> <tr><td>9</td><td>0.6000</td><td>0.4500</td><td>0.6750</td><td>0.9750</td></tr> <tr><td>10</td><td>0.6833</td><td>0.3667</td><td>0.4667</td><td>0.9833</td></tr> <tr><td>11</td><td>0.3417</td><td>0.4417</td><td>0.7750</td><td>0.9583</td></tr> <tr><td>12</td><td>0.7207</td><td>0.5766</td><td>0.5856</td><td>0.9640</td></tr> </tbody> </table> | Batch | SGD | Perceptron | Passive Aggressive | Hoeffding Tree | 1 | 0.5250 | 0.3833 | 0.5583 | 0.5333 | 2 | 0.6333 | 0.5417 | 0.6500 | 0.6083 | 3 | 0.4917 | 0.6500 | 0.6667 | 0.9750 | 4 | 0.5833 | 0.6500 | 0.5750 | 0.9833 | 5 | 0.6000 | 0.6083 | 0.5500 | 1.0000 | 6 | 0.6167 | 0.4250 | 0.6167 | 0.9750 | 7 | 0.5083 | 0.4500 | 0.5250 | 0.9833 | 8 | 0.4583 | 0.5333 | 0.6083 | 0.9750 | 9 | 0.6000 | 0.4500 | 0.6750 | 0.9750 | 10 | 0.6833 | 0.3667 | 0.4667 | 0.9833 | 11 | 0.3417 | 0.4417 | 0.7750 | 0.9583 | 12 | 0.7207 | 0.5766 | 0.5856 | 0.9640 |
| Batch | SGD | Perceptron | Passive Aggressive | Hoeffding Tree | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 1 | 0.5250 | 0.3833 | 0.5583 | 0.5333 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 2 | 0.6333 | 0.5417 | 0.6500 | 0.6083 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 3 | 0.4917 | 0.6500 | 0.6667 | 0.9750 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 4 | 0.5833 | 0.6500 | 0.5750 | 0.9833 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 5 | 0.6000 | 0.6083 | 0.5500 | 1.0000 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 6 | 0.6167 | 0.4250 | 0.6167 | 0.9750 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 7 | 0.5083 | 0.4500 | 0.5250 | 0.9833 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 8 | 0.4583 | 0.5333 | 0.6083 | 0.9750 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 9 | 0.6000 | 0.4500 | 0.6750 | 0.9750 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 10 | 0.6833 | 0.3667 | 0.4667 | 0.9833 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 11 | 0.3417 | 0.4417 | 0.7750 | 0.9583 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 12 | 0.7207 | 0.5766 | 0.5856 | 0.9640 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| C | | <table border="1"> <thead> <tr> <th>Batch</th> <th>SGD</th> <th>Perceptron</th> <th>Passive Aggressive</th> <th>Hoeffding Tree</th> </tr> </thead> <tbody> <tr><td>0</td><td>0.5583</td><td>0.7417</td><td>0.7667</td><td>0.5167</td></tr> <tr><td>1</td><td>0.5667</td><td>0.4500</td><td>0.6333</td><td>0.5750</td></tr> <tr><td>2</td><td>0.8167</td><td>0.7417</td><td>0.6583</td><td>0.9750</td></tr> <tr><td>3</td><td>0.5333</td><td>0.5500</td><td>0.4083</td><td>0.9917</td></tr> <tr><td>4</td><td>0.4250</td><td>0.6167</td><td>0.6667</td><td>0.9667</td></tr> <tr><td>5</td><td>0.6917</td><td>0.5583</td><td>0.5667</td><td>0.9833</td></tr> <tr><td>6</td><td>0.5417</td><td>0.4750</td><td>0.9000</td><td>0.9667</td></tr> <tr><td>7</td><td>0.1917</td><td>0.7500</td><td>0.5333</td><td>0.9667</td></tr> <tr><td>8</td><td>0.4833</td><td>0.4083</td><td>0.4583</td><td>0.9917</td></tr> <tr><td>9</td><td>0.5000</td><td>0.2333</td><td>0.7917</td><td>1.0000</td></tr> <tr><td>10</td><td>0.4167</td><td>0.2917</td><td>0.8417</td><td>0.9667</td></tr> <tr><td>11</td><td>0.8036</td><td>0.3571</td><td>0.6964</td><td>1.0000</td></tr> </tbody> </table> | Batch | SGD | Perceptron | Passive Aggressive | Hoeffding Tree | 0 | 0.5583 | 0.7417 | 0.7667 | 0.5167 | 1 | 0.5667 | 0.4500 | 0.6333 | 0.5750 | 2 | 0.8167 | 0.7417 | 0.6583 | 0.9750 | 3 | 0.5333 | 0.5500 | 0.4083 | 0.9917 | 4 | 0.4250 | 0.6167 | 0.6667 | 0.9667 | 5 | 0.6917 | 0.5583 | 0.5667 | 0.9833 | 6 | 0.5417 | 0.4750 | 0.9000 | 0.9667 | 7 | 0.1917 | 0.7500 | 0.5333 | 0.9667 | 8 | 0.4833 | 0.4083 | 0.4583 | 0.9917 | 9 | 0.5000 | 0.2333 | 0.7917 | 1.0000 | 10 | 0.4167 | 0.2917 | 0.8417 | 0.9667 | 11 | 0.8036 | 0.3571 | 0.6964 | 1.0000 |
| Batch | SGD | Perceptron | Passive Aggressive | Hoeffding Tree | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 0 | 0.5583 | 0.7417 | 0.7667 | 0.5167 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 1 | 0.5667 | 0.4500 | 0.6333 | 0.5750 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 2 | 0.8167 | 0.7417 | 0.6583 | 0.9750 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 3 | 0.5333 | 0.5500 | 0.4083 | 0.9917 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 4 | 0.4250 | 0.6167 | 0.6667 | 0.9667 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 5 | 0.6917 | 0.5583 | 0.5667 | 0.9833 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 6 | 0.5417 | 0.4750 | 0.9000 | 0.9667 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 7 | 0.1917 | 0.7500 | 0.5333 | 0.9667 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 8 | 0.4833 | 0.4083 | 0.4583 | 0.9917 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 9 | 0.5000 | 0.2333 | 0.7917 | 1.0000 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 10 | 0.4167 | 0.2917 | 0.8417 | 0.9667 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 11 | 0.8036 | 0.3571 | 0.6964 | 1.0000 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| D | | <table border="1"> <thead> <tr> <th>Batch</th> <th>SGD</th> <th>Perceptron</th> <th>Passive Aggressive</th> <th>Hoeffding Tree</th> </tr> </thead> <tbody> <tr><td>0</td><td>0.4583</td><td>0.5083</td><td>0.5583</td><td>0.4250</td></tr> <tr><td>1</td><td>0.6416</td><td>0.4666</td><td>0.7000</td><td>0.5166</td></tr> <tr><td>2</td><td>0.8083</td><td>0.5000</td><td>0.7833</td><td>0.9750</td></tr> <tr><td>3</td><td>0.6250</td><td>0.5750</td><td>0.7500</td><td>0.9666</td></tr> <tr><td>4</td><td>0.6500</td><td>0.4666</td><td>0.5500</td><td>0.9833</td></tr> <tr><td>5</td><td>0.4583</td><td>0.5583</td><td>0.8500</td><td>0.9500</td></tr> <tr><td>6</td><td>0.6000</td><td>0.5916</td><td>0.6500</td><td>0.8500</td></tr> <tr><td>7</td><td>0.4333</td><td>0.7916</td><td>0.7916</td><td>0.9666</td></tr> <tr><td>8</td><td>0.5750</td><td>0.6000</td><td>0.5833</td><td>0.9000</td></tr> <tr><td>9</td><td>0.6416</td><td>0.4750</td><td>0.7166</td><td>1.0000</td></tr> <tr><td>10</td><td>0.6000</td><td>0.4500</td><td>0.6500</td><td>0.9250</td></tr> <tr><td>11</td><td>0.5625</td><td>0.4821</td><td>0.6250</td><td>0.9553</td></tr> </tbody> </table> | Batch | SGD | Perceptron | Passive Aggressive | Hoeffding Tree | 0 | 0.4583 | 0.5083 | 0.5583 | 0.4250 | 1 | 0.6416 | 0.4666 | 0.7000 | 0.5166 | 2 | 0.8083 | 0.5000 | 0.7833 | 0.9750 | 3 | 0.6250 | 0.5750 | 0.7500 | 0.9666 | 4 | 0.6500 | 0.4666 | 0.5500 | 0.9833 | 5 | 0.4583 | 0.5583 | 0.8500 | 0.9500 | 6 | 0.6000 | 0.5916 | 0.6500 | 0.8500 | 7 | 0.4333 | 0.7916 | 0.7916 | 0.9666 | 8 | 0.5750 | 0.6000 | 0.5833 | 0.9000 | 9 | 0.6416 | 0.4750 | 0.7166 | 1.0000 | 10 | 0.6000 | 0.4500 | 0.6500 | 0.9250 | 11 | 0.5625 | 0.4821 | 0.6250 | 0.9553 |
| Batch | SGD | Perceptron | Passive Aggressive | Hoeffding Tree | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 0 | 0.4583 | 0.5083 | 0.5583 | 0.4250 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 1 | 0.6416 | 0.4666 | 0.7000 | 0.5166 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 2 | 0.8083 | 0.5000 | 0.7833 | 0.9750 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 3 | 0.6250 | 0.5750 | 0.7500 | 0.9666 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 4 | 0.6500 | 0.4666 | 0.5500 | 0.9833 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 5 | 0.4583 | 0.5583 | 0.8500 | 0.9500 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 6 | 0.6000 | 0.5916 | 0.6500 | 0.8500 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 7 | 0.4333 | 0.7916 | 0.7916 | 0.9666 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 8 | 0.5750 | 0.6000 | 0.5833 | 0.9000 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 9 | 0.6416 | 0.4750 | 0.7166 | 1.0000 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 10 | 0.6000 | 0.4500 | 0.6500 | 0.9250 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 11 | 0.5625 | 0.4821 | 0.6250 | 0.9553 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |

differentiate between normal and encrypted data, with a slight dip in Batch 6 before resuming high performance in subsequent batches.

The intermittent encryption and online learning Challenges:

Given the pronounced advantages conferred to adversarial entities and the feasibility of its deployment, it is imperative to concentrate on the exploration of intermittent encryption,

which constitutes a minor fraction of files encrypted by ransomware. Consequently, we posit that contemporary online learning paradigms may not adequately adapt to the nuances of intermittent encryption. In this section, we subject our dataset to encryption via the Black Basta algorithm. Table 5 delineates the performance metrics of various online learning classifiers, namely SGD, Perceptron, Passive Aggressive, and Hoeffding Tree algorithms, spanned across 12 distinct batches. It is salient

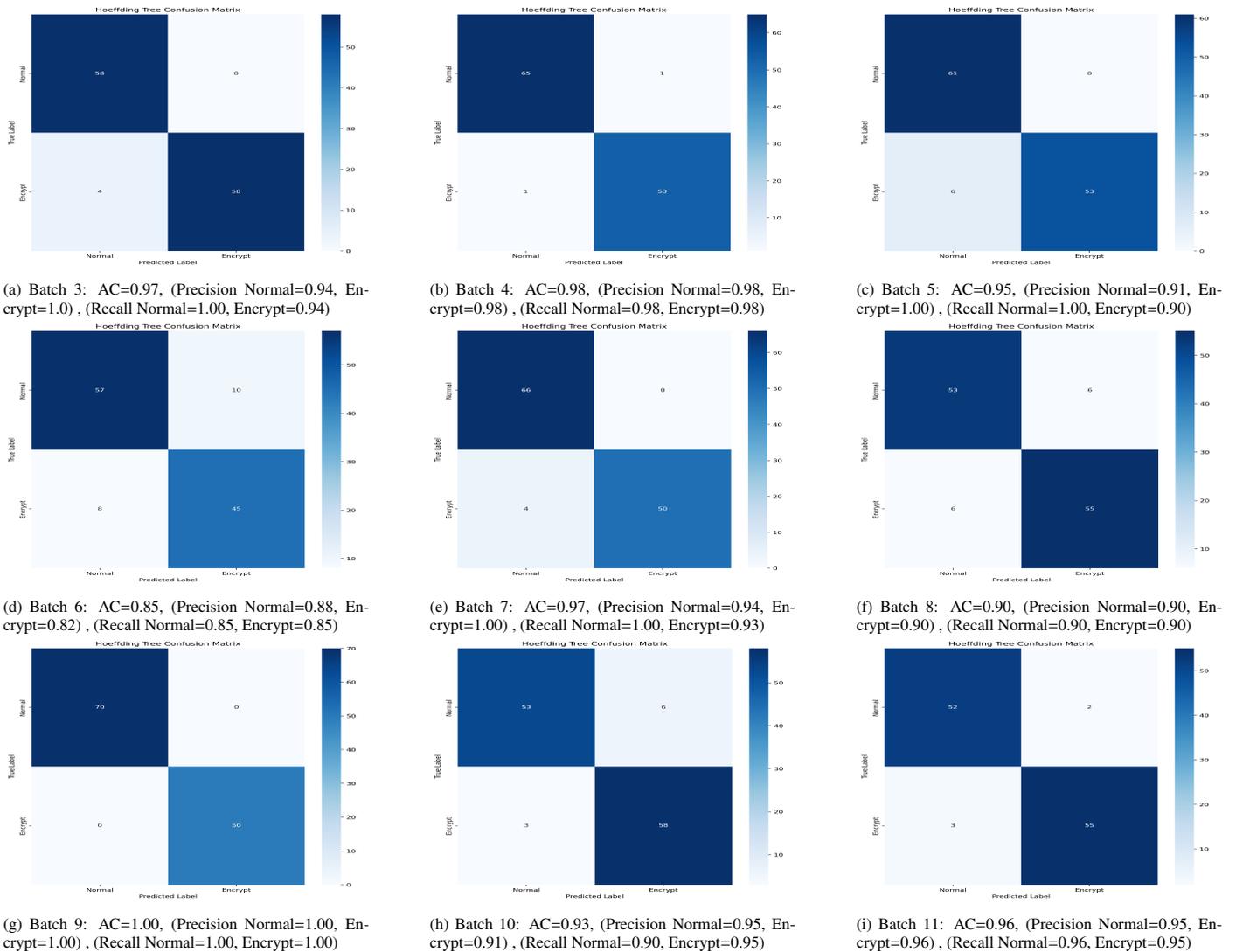


(a) Batch 0: AC=0.42, (Precision Normal=0.44, Encrypt=0.0), (Recall Normal=0.93, Encrypt=0.0)

(b) Batch 1: AC=0.52, (Precision Normal=0.52, Encrypt=0.50), (Recall Normal=0.98, Encrypt=0.2)

(c) Batch 2: AC=0.97, (Precision Normal=0.95, Encrypt=1.0), (Recall Normal=1.0, Encrypt=0.95)

Figure 5: The results of the Hoeffding Tree's performance on a dataset encrypted by CryptoFile ransomware involved analyzing the confusion matrix, precision, and recall across the first three batches.



(a) Batch 3: AC=0.97, (Precision Normal=0.94, Encrypt=1.0), (Recall Normal=1.00, Encrypt=0.94)

(b) Batch 4: AC=0.98, (Precision Normal=0.98, Encrypt=0.98), (Recall Normal=0.98, Encrypt=0.98)

(c) Batch 5: AC=0.95, (Precision Normal=0.91, Encrypt=1.00), (Recall Normal=1.00, Encrypt=0.90)

(d) Batch 6: AC=0.85, (Precision Normal=0.88, Encrypt=0.82), (Recall Normal=0.85, Encrypt=0.85)

(e) Batch 7: AC=0.97, (Precision Normal=0.94, Encrypt=1.00), (Recall Normal=1.00, Encrypt=0.93)

(f) Batch 8: AC=0.90, (Precision Normal=0.90, Encrypt=0.90), (Recall Normal=0.90, Encrypt=0.90)

(g) Batch 9: AC=1.00, (Precision Normal=1.00, Encrypt=1.00), (Recall Normal=1.00, Encrypt=1.00)

(h) Batch 10: AC=0.93, (Precision Normal=0.95, Encrypt=0.91), (Recall Normal=0.90, Encrypt=0.95)

(i) Batch 11: AC=0.96, (Precision Normal=0.95, Encrypt=0.96), (Recall Normal=0.96, Encrypt=0.95)

Figure 6: The analysis of the Hoeffding Tree's performance on a dataset encrypted by CryptoFile2 ransomware extended to include the confusion matrix, precision, and recall measurements across an additional nine batches, following the initial three illustrated in Figure 5.

Table 5: Comparative Analysis of Online Classifiers Against Intermittent Encryption Adversarial Threats. This table delineates the performance distinctions among various online classifiers when contending with data encrypted using intermittent encryption. Initial observations reveal a suboptimal performance by the Hoeffding classifier in contrast to other encryption strategies. However, upon deeper examination, it emerges that the Random Forest classifier notably surpasses its counterparts, exhibiting a superior predictive accuracy concerning intermittent encryption. **Row A** illustrates the use of intermittent encryption, which results in underperforming the Hoeffding Tree. **Row B** illustrates benchmarking classifiers with differential entropy as an additional feature, resulting in the random forest classifier outperforming others. **Row C** illustrates a comparison between the Random Forest and Hoeffding Tree classifiers.

| # | Comparison of Classifier Performance Graph | Comparison of Classifier Performance | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|-------------------------|--|--|--------------------|---------------------|----------------------|--------------------|----------------|---------------------|--------|--------|--------|--------|-------------------------|--------|--------|--------|--------|----------------|--------|--------|--------|---------------|-----------------------|---------------|---------------|---------------|---------------|---------------------|--------|--------|--------|--------|----------|--------|--------|--------|---------------|-------------------|--------|--------|--------|--------|-----------------------|--------|--------|--------|--------|-------------|--------|--------|--------|---------------|---|--------|--------|--------|--------|----|--------|--------|--------|---------------|----|--------|--------|--------|--------|
| A | | <table border="1"> <thead> <tr> <th>Batch</th> <th>SGD</th> <th>Perceptron</th> <th>Passive Aggressive</th> <th>Hoeffding Tree</th> </tr> </thead> <tbody> <tr><td>0</td><td>0.5167</td><td>0.5250</td><td>0.5500</td><td>0.4917</td></tr> <tr><td>1</td><td>0.4917</td><td>0.4333</td><td>0.5417</td><td>0.5333</td></tr> <tr><td>2</td><td>0.4500</td><td>0.5833</td><td>0.8500</td><td>0.9250</td></tr> <tr><td>3</td><td>0.5250</td><td>0.4500</td><td>0.4500</td><td>0.6417</td></tr> <tr><td>4</td><td>0.6333</td><td>0.4833</td><td>0.6833</td><td>0.7917</td></tr> <tr><td>5</td><td>0.6750</td><td>0.5250</td><td>0.5000</td><td>0.8750</td></tr> <tr><td>6</td><td>0.5333</td><td>0.4833</td><td>0.5500</td><td>0.8083</td></tr> <tr><td>7</td><td>0.5000</td><td>0.4750</td><td>0.6667</td><td>0.7667</td></tr> <tr><td>8</td><td>0.5750</td><td>0.6417</td><td>0.5500</td><td>0.9250</td></tr> <tr><td>9</td><td>0.5250</td><td>0.5333</td><td>0.5250</td><td>0.8333</td></tr> <tr><td>10</td><td>0.5833</td><td>0.5000</td><td>0.7167</td><td>0.8667</td></tr> <tr><td>11</td><td>0.4732</td><td>0.5000</td><td>0.6071</td><td>0.7589</td></tr> </tbody> </table> | Batch | SGD | Perceptron | Passive Aggressive | Hoeffding Tree | 0 | 0.5167 | 0.5250 | 0.5500 | 0.4917 | 1 | 0.4917 | 0.4333 | 0.5417 | 0.5333 | 2 | 0.4500 | 0.5833 | 0.8500 | 0.9250 | 3 | 0.5250 | 0.4500 | 0.4500 | 0.6417 | 4 | 0.6333 | 0.4833 | 0.6833 | 0.7917 | 5 | 0.6750 | 0.5250 | 0.5000 | 0.8750 | 6 | 0.5333 | 0.4833 | 0.5500 | 0.8083 | 7 | 0.5000 | 0.4750 | 0.6667 | 0.7667 | 8 | 0.5750 | 0.6417 | 0.5500 | 0.9250 | 9 | 0.5250 | 0.5333 | 0.5250 | 0.8333 | 10 | 0.5833 | 0.5000 | 0.7167 | 0.8667 | 11 | 0.4732 | 0.5000 | 0.6071 | 0.7589 |
| Batch | SGD | Perceptron | Passive Aggressive | Hoeffding Tree | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 0 | 0.5167 | 0.5250 | 0.5500 | 0.4917 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 1 | 0.4917 | 0.4333 | 0.5417 | 0.5333 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 2 | 0.4500 | 0.5833 | 0.8500 | 0.9250 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 3 | 0.5250 | 0.4500 | 0.4500 | 0.6417 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 4 | 0.6333 | 0.4833 | 0.6833 | 0.7917 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 5 | 0.6750 | 0.5250 | 0.5000 | 0.8750 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 6 | 0.5333 | 0.4833 | 0.5500 | 0.8083 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 7 | 0.5000 | 0.4750 | 0.6667 | 0.7667 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 8 | 0.5750 | 0.6417 | 0.5500 | 0.9250 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 9 | 0.5250 | 0.5333 | 0.5250 | 0.8333 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 10 | 0.5833 | 0.5000 | 0.7167 | 0.8667 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 11 | 0.4732 | 0.5000 | 0.6071 | 0.7589 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| B | | <table border="1"> <thead> <tr> <th>Classifier</th> <th>Accuracy</th> <th>Precision</th> <th>Recall</th> <th>F1-score</th> </tr> </thead> <tbody> <tr><td>Logistic Regression</td><td>0.6248</td><td>0.6000</td><td>0.7530</td><td>0.6636</td></tr> <tr><td>Support Vector Machines</td><td>0.6130</td><td>0.6304</td><td>0.6127</td><td>0.6134</td></tr> <tr><td>Decision Trees</td><td>0.7155</td><td>0.7170</td><td>0.7199</td><td>0.7229</td></tr> <tr><td>Random Forests</td><td>0.7832</td><td>0.7643</td><td>0.8225</td><td>0.7911</td></tr> <tr><td>k-Nearest Neighbors</td><td>0.6027</td><td>0.6001</td><td>0.6521</td><td>0.6233</td></tr> <tr><td>AdaBoost</td><td>0.6605</td><td>0.6549</td><td>0.7452</td><td>0.6839</td></tr> <tr><td>Gradient Boosting</td><td>0.7131</td><td>0.6918</td><td>0.8038</td><td>0.7366</td></tr> <tr><td>Multilayer Perceptron</td><td>0.5949</td><td>0.5659</td><td>0.8591</td><td>0.5991</td></tr> <tr><td>Naive Bayes</td><td>0.5448</td><td>0.5281</td><td>0.8939</td><td>0.6620</td></tr> </tbody> </table> | Classifier | Accuracy | Precision | Recall | F1-score | Logistic Regression | 0.6248 | 0.6000 | 0.7530 | 0.6636 | Support Vector Machines | 0.6130 | 0.6304 | 0.6127 | 0.6134 | Decision Trees | 0.7155 | 0.7170 | 0.7199 | 0.7229 | Random Forests | 0.7832 | 0.7643 | 0.8225 | 0.7911 | k-Nearest Neighbors | 0.6027 | 0.6001 | 0.6521 | 0.6233 | AdaBoost | 0.6605 | 0.6549 | 0.7452 | 0.6839 | Gradient Boosting | 0.7131 | 0.6918 | 0.8038 | 0.7366 | Multilayer Perceptron | 0.5949 | 0.5659 | 0.8591 | 0.5991 | Naive Bayes | 0.5448 | 0.5281 | 0.8939 | 0.6620 | | | | | | | | | | | | | | | |
| Classifier | Accuracy | Precision | Recall | F1-score | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Logistic Regression | 0.6248 | 0.6000 | 0.7530 | 0.6636 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Support Vector Machines | 0.6130 | 0.6304 | 0.6127 | 0.6134 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Decision Trees | 0.7155 | 0.7170 | 0.7199 | 0.7229 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Random Forests | 0.7832 | 0.7643 | 0.8225 | 0.7911 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| k-Nearest Neighbors | 0.6027 | 0.6001 | 0.6521 | 0.6233 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| AdaBoost | 0.6605 | 0.6549 | 0.7452 | 0.6839 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Gradient Boosting | 0.7131 | 0.6918 | 0.8038 | 0.7366 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Multilayer Perceptron | 0.5949 | 0.5659 | 0.8591 | 0.5991 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Naive Bayes | 0.5448 | 0.5281 | 0.8939 | 0.6620 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| C | | <table border="1"> <thead> <tr> <th>Batch</th> <th>Random Forest Score</th> <th>Hoeffding Tree Score</th> </tr> </thead> <tbody> <tr><td>1</td><td>0.9660</td><td>0.5080</td></tr> <tr><td>2</td><td>0.8400</td><td>0.5140</td></tr> <tr><td>3</td><td>0.9200</td><td>0.5080</td></tr> <tr><td>4</td><td>0.8080</td><td>0.6400</td></tr> <tr><td>5</td><td>0.9460</td><td>0.7840</td></tr> <tr><td>6</td><td>0.9200</td><td>0.7540</td></tr> <tr><td>7</td><td>0.8420</td><td>0.8120</td></tr> <tr><td>8</td><td>0.9480</td><td>0.7580</td></tr> <tr><td>9</td><td>0.8840</td><td>0.9740</td></tr> <tr><td>10</td><td>0.9340</td><td>0.8560</td></tr> <tr><td>11</td><td>0.9480</td><td>0.9160</td></tr> <tr><td>12</td><td>0.9375</td><td>0.8836</td></tr> </tbody> </table> | Batch | Random Forest Score | Hoeffding Tree Score | 1 | 0.9660 | 0.5080 | 2 | 0.8400 | 0.5140 | 3 | 0.9200 | 0.5080 | 4 | 0.8080 | 0.6400 | 5 | 0.9460 | 0.7840 | 6 | 0.9200 | 0.7540 | 7 | 0.8420 | 0.8120 | 8 | 0.9480 | 0.7580 | 9 | 0.8840 | 0.9740 | 10 | 0.9340 | 0.8560 | 11 | 0.9480 | 0.9160 | 12 | 0.9375 | 0.8836 | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Batch | Random Forest Score | Hoeffding Tree Score | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 1 | 0.9660 | 0.5080 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 2 | 0.8400 | 0.5140 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 3 | 0.9200 | 0.5080 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 4 | 0.8080 | 0.6400 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 5 | 0.9460 | 0.7840 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 6 | 0.9200 | 0.7540 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 7 | 0.8420 | 0.8120 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 8 | 0.9480 | 0.7580 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 9 | 0.8840 | 0.9740 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 10 | 0.9340 | 0.8560 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 11 | 0.9480 | 0.9160 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 12 | 0.9375 | 0.8836 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |

to note that the Hoeffding Tree algorithm manifests favorable outcomes in merely four of the delineated batches.

Shortcomings of the Hoeffding Tree algorithm in intermittent encryption: we advocate for the integration of differential entropy [44] as a pivotal feature in the classification process of intermittently encrypted data. The differential entropy allows for a flexible and adaptive measure of the data’s uncertainty, making it more suitable for dealing with the fluctuations and gaps caused by intermittent encryption.

The differential entropy, represented as $h(X)$, serves as a potent tool in quantifying the uncertainty of continuous random variables, articulated mathematically as:

$$h(X) = - \int_{-\infty}^{\infty} f(x) \log_2 f(x) dx \quad (9)$$

where $f(x)$ signifies the probability density function (pdf) of the variable X .

By employing differential entropy, we aim to harness its analytical acumen to navigate the encrypted buffers’ intricacies within the file system adeptly, thereby fostering a model imbued with enhanced precision and reliability in the face of intermittent encryption’s unpredictabilities.

Conversely, a more in-depth exploration reveals that the Random Forest classifier notably excels, manifesting predominant efficacy in predicting outcomes under the influence of intermittent encryption (See results in Table 5).

5.10. Baseline model and discussion

Recent literature emphasizes that the majority of ransomware detection mechanisms proposed for detecting encryption activities at the file system level predominantly utilize entropy analysis [6, 32, 8, 45, 46, 47, 17]. As delineated in Section 3, adversaries have evolved their strategies to either circumvent entropy detection measures or reduce the visibility of encryption indicators, thereby undermining the effectiveness of conventional

machine learning models. These models, which largely depend on static offline training, are increasingly ineffectual against the sophisticated techniques employed by modern adversaries. As part of our baseline evaluation, we assessed the performance of popular traditional classifiers such as SVM, Decision Trees, Random Forests, etc., all of which leverage Shannon entropy as a key feature. For this purpose, we utilized the Black Basta ransomware to encrypt a dataset, subsequently comparing the outcomes with our initial evaluation that employed the same traditional classifiers but augmented with a set of identified features, as discussed in Section 5.3. The findings from this evaluation are encapsulated in Table 6.

In the comparative analysis of classification algorithms based on their performance metrics—accuracy, precision, recall, and F1 score—it is evident that the inclusion of a multifaceted feature set encompassing byte entropy, byte variance, byte mean, byte kurtosis, byte skewness, differential entropy, and Shannon entropy, as opposed to relying solely on entropy, significantly enhances classifier performance. Table 6, characterized by its singular reliance on entropy as a feature, yielded modest performance metrics across various classifiers, with Decision Trees achieving the highest accuracy of 0.5331 and k-Nearest Neighbors demonstrating the best recall and F1 score, at 0.6710 and 0.5750, respectively. This suggests a limited capability of entropy alone to capture the intricate patterns within the data, thus constraining the classifiers’ effectiveness.

Conversely, the integration of a comprehensive feature set in the second dataset led to a remarkable improvement in all evaluated metrics (see table 2). Notably, Decision Trees exhibited a profound increase in accuracy to 0.9808, while Random Forests achieved superior precision and recall rates of 0.9703 and 0.9947, respectively. Such improvements underscore the critical importance of feature diversity in enhancing the predictive accuracy and reliability of classifiers. The multifaceted feature set evidently provides a richer, more nuanced representation of the data, thereby enabling more complex classifiers like Random Forests, Gradient Boosting, and AdaBoost to exploit this complexity to achieve near-optimal performance.

Table 6: Classification reports summary for various models that rely on entropy feature encrypted by Black Basta (intermittent encryption) and CryptFile2 ransomware that encrypted by a strong encryption with RSA-2048.

| Classifier | Accuracy | Precision | Recall | F1 |
|-----------------------|----------|-----------|--------|--------|
| Logistic Regression | 0.4881 | 0.4878 | 0.3807 | 0.4165 |
| SVM | 0.5018 | 0.5019 | 0.4006 | 0.4154 |
| Decision Trees | 0.5331 | 0.5390 | 0.4618 | 0.4948 |
| Random Forests | 0.5298 | 0.5331 | 0.4877 | 0.5068 |
| k-Nearest Neighbors | 0.5062 | 0.5064 | 0.6710 | 0.5750 |
| AdaBoost | 0.5012 | 0.4999 | 0.5952 | 0.5309 |
| Gradient Boosting | 0.5128 | 0.5110 | 0.5714 | 0.5339 |
| Multilayer Perceptron | 0.4875 | 0.4863 | 0.3769 | 0.4135 |
| Naive Bayes | 0.4940 | 0.4983 | 0.4804 | 0.4691 |

6. Related work

Over recent years, the field of ransomware detection has received significant attention within the academic community.

Traditional detection methodologies typically involve the analysis of malicious executable [48], an approach that is increasingly proving to be ineffective against the mounting sophistication of polymorphism and code obfuscation techniques employed by malicious actors [49].

In a concerted effort to overcome these increasingly complex obfuscation tactics, academic research has gradually transitioned toward behavior-based analysis techniques. These methods revolve around the scrutiny of runtime behaviors, such as system call sequences and Read/Write operations, which are intrinsically challenging to manipulate without fundamentally altering the malware’s core functionality [50, 34]. For instance, in the context of process-level monitoring, any ongoing process that exceeds a predefined trust threshold is identified and classified as a potentially malicious activity. However, it is important to note that these behavior-based detection techniques are not without their limitations. Indeed, sophisticated adversaries may circumvent these strategies by dividing the ransomware activity into multiple distinct processes. Each process performs only a fraction of the overall ransomware operations, thereby resulting in the same net output as a full-scale ransomware execution [51]. A prominent example of a ransomware family employing this evasion technique is LockerGoga [52].

The aforementioned behavior-based detection methodologies generally demonstrate efficacy in dealing with known threats, yet their effectiveness diminishes in the face of previously unseen ransomware samples. Given the propensity for ransomware to attack user files, recent scholarship has advocated for the implementation of a decoy file-based deceptive solution as a means to expediently detect cryptographic ransomware attacks. These decoy files, which can be created either automatically or manually, are strategically dispersed across the network [53].

Another direction within the field of ransomware detection involves leveraging the communication between the Command and Control (C&C) Server and the targeted victim as a behavioral characteristic to detect potential ransomware attacks [54, 55, 56]. Almashhadani et al. [56] empirically demonstrated this approach, utilizing network traffic data to investigate Locky Ransomware. In their study, they extracted 20 features from the network traffic data to construct a classification model. Experimental evaluations of this proposed detection system using Random Forests, Bayes Networks, and Support Vector Machines indicated high detection accuracy, a low false positive rate, and useful feature extraction. Additionally, the system proved highly effective in tracking the network activities of ransomware. However, there are important caveats to consider in the application of these techniques. Firstly, Almashhadani et al.’s work focuses solely on the Locky ransomware, which introduces potential issues of generalisability, as the findings might not apply to other types of ransomware. Secondly, while some ransomware families necessitate an internet connection to initiate the encryption process, numerous others do not require a connection to the C&C server to begin encrypting victim files, thereby limiting the effectiveness of this approach.

A different behavioral feature used for ransomware detection is the monitoring of hardware performance counters (HPCs)

such as cache-reference, cache misses, and instruction counts [57, 58]. This methodology is rooted in the fact that ransomware perpetrators, in their quest to maximize potential gains, seek to encrypt as many user files as possible. The aggressive execution of file encryption and renaming processes usually provokes a context switch, which in turn influences the CPU status, affecting elements such as the CPU cache and branch prediction. For instance, the authors in EGB [58] evaluated the features derived from hardware performance counters using several machine learning algorithms to classify applications into ransomware and non-ransomware categories. In their research, EGB [58] demonstrated that a model built using the Random Forest algorithm exhibited the highest detection rate.

Additionally, DeepGuard’s work [59] introduced an innovative method for capturing user activity. Their proposed mechanism relied on monitoring file interaction patterns and utilized a deep generative autoencoder architecture to reconstruct the input. DeepGuard’s approach is premised on the assumption that, with sufficient training on user file-interaction logs collected over several days, the deep generative autoencoder becomes proficient in reconstructing unseen inputs with minimal reconstruction error, provided the input resembles legitimate user activity. Consequently, this model can identify anomalous (or ransomware) activity by exclusively modeling the user interaction pattern, effectively distinguishing benign user activities from ransomware operations.

A study by Baek et al. [60] proposed a two-pronged ransomware protection system known as SSDinsider++. This system focused both on the detection of ransomware attacks and on data recovery. However, their approach was constrained by the practical infeasibility of assessing every I/O block, its header, and payload during runtime. SSDinsider++’s detection algorithm operates by observing the I/O patterns of a host system and determines whether the host is under ransomware attack at an early stage.

Meanwhile, in an effort to enhance the detection rate of crypto-ransomware, the authors in File-entropy [61] leveraged the entropy of encrypted files as a metric to identify ransomware activity. They encrypted and analyzed over 20 file formats using WannaCry, Phobos, GandCrab, and Globelmposter ransomware, gathering features essential for distinguishing ransomware tasks from file encryption activities, such as Zip and 7z. These extracted features were then fed into a support vector machine for classification tasks. Their experimental results showed that their model could detect ransomware activity with an average accuracy of 85.17%. However, their research was limited to only four ransomware samples, and may therefore lack the ability to detect other varieties of ransomware.

The deception-based [62] solution operates by setting up artificial computer system assets such as a web server or router across an enterprise network. These assets act as bait, enticing intruders and subsequently triggering alerts when accessed. While this methodology is still relatively novel, some recent studies have proposed deception-based strategies for ransomware detection [63, 64, 62, 65].

For example, Lee et al. [63] implemented a proof of concept involving the creation and placement of decoy files based

on ransomware file traversal patterns. In their study, various ransomware samples were decompiled and analyzed to understand how ransomware traverses file systems. They suggested planting two deceptive files in the root directory, one at the beginning and another at the end of an alphabetical list. Additionally, Gómez et al. [64] developed R-Locker, a tool designed to thwart ransomware attacks by creating decoy files. These files were generated using the ‘makeinfo’ command in Unix systems, filled with 3KB of data, and placed in the user’s home directory. R-Locker would then block any process attempting to access these files. However, the effectiveness of this approach was limited because all the decoy files were of the same type, allowing ransomware to potentially evade detection.

In Rwgward [65], the authors proposed a novel combination of a behavioral-based detection model with a decoy-based approach to combat cryptographic ransomware attacks. Rwgward randomly designated original user files as decoy files. Likewise, Cryptostopper [62] embedded arbitrarily generated decoy files in the file system to create a deceptive defense against ransomware attacks. Any attempt to alter the content of these deceptive files would trigger an alert, notifying the system administrator and closing the infected host. As Cryptostopper is a commercial product, its technical details are not publicly available.

Several studies have utilized dynamic analysis to examine the behavior of ransomware, including work by Kharraz et al., Song et al., Cabaj et al., and Andronio et al. [66, 34, 67, 68, 69]. These studies focused on various elements such as I/O data buffer entropy, access patterns, file system activities [70], CPU and memory usage, and network behavior. Despite its effectiveness against evasive ransomware types, dynamic analysis has inherent limitations. Notably, it necessitates the execution of ransomware in a safe and monitored environment; otherwise, the platform risks infection. This constraint may limit the feasibility of dynamic analysis in certain scenarios. We have summarized and critiqued related work approaches, as well as what this paper offers, in Table 7.

Table 8 serves as a concise overview of various ransomware detection approaches, highlighting their strengths and limitations. It offers a quick reference for researchers and practitioners in the field of cybersecurity, providing insights into the key criteria used to evaluate these approaches. The table aims to assist in understanding the relative effectiveness of each method in combating ransomware threats and what this paper is aiming to achieve. The paper examines techniques employed by adversaries to diminish the indication of compromise threshold by manipulating various features and activities, including factors like entropy, input and output data, and the exhaustive utilization of applications.

7. Conclusion and future directions

This study delves deeply into the ever-expanding realm of ransomware, meticulously investigating the dynamic confrontations between adversaries, who continuously evolve sophisticated strategies for optimizing illicit gains, and defenders, who

Table 7: Summary and Critique of Ransomware Detection Approaches

| Approach | Criteria | Summary and Critique |
|-------------------------------|---|--|
| Behaviour-Based Analysis | - Effectiveness against polymorphism and obfuscation techniques [49] - Efficacy against known threats [48] | Effective against evolving ransomware tactics, but less effective against unknown ransomware [50]. Demonstrates high efficacy. |
| Decoy File-Based Deception | - Detection speed and accuracy - Lack of deception diversity | Rapidly detects cryptographic ransomware attacks. Limited effectiveness when ransomware adapts to recognize decoy files as fake. |
| Communication Analysis | - Detection accuracy and false positive rate [56] - Limited applicability | High accuracy and low false positive rate. Limited effectiveness when ransomware doesn't rely on C&C servers or when dealing with new, previously unseen variants. |
| Hardware Performance Counters | - Detection rate [58] - Resource overhead [57] - Limited applicability | High detection rate with hardware performance counters. May introduce resource overhead, impacting system performance. May not be effective against non-resource-intensive ransomware. |
| User Activity Monitoring | - Anomaly detection capability [59] - Training data limitations | Effective at distinguishing benign activities from ransomware. Requires substantial training data and may struggle with rapidly evolving ransomware. |
| SSDinsider++ | - Early-stage detection capability [60] - Practicality [60] | Early detection based on I/O patterns. Practicality limited due to resource-intensive assessment. |
| File Entropy Analysis | - Detection accuracy [61] - Limited ransomware sample coverage [61] - Scalability [61] | Accurate in detecting ransomware activity. Limited to the ransomware samples used for training. May struggle to scale with a large number of file formats. |
| Deception-Based Strategies | - Intruder enticement and alert triggering [63] - Effectiveness against adaptive ransomware | Detects intruders and triggers alerts effectively. Limited effectiveness when ransomware evolves to recognize decoy assets. |
| Dynamic Analysis | - Effectiveness against evasive ransomware types [66] - Platform infection risk [66] - Variability among ransomware variants [28] | Effective but requires a safe and monitored environment. Necessitates executing ransomware in a controlled environment, which may not always be feasible due to virtualization detection, incubation period, and other evasion techniques [28]. Behaviours are specific to individual ransomware variants, requiring generic models or continuous training to adapt to new variants. |

carefully orchestrate countermeasures to mitigate risks associated with file system-level data encryption. Through a comprehensive analysis, the research illuminates nuanced adversary techniques designed to circumvent storage-level and behavioral countermeasures, and employs advanced methodologies, like online incremental machine learning algorithms, to unravel the complexities of these confrontations. The findings reveal that the Hoeffding Tree algorithm, renowned for its intrinsic incremental learning abilities, stands out with exemplary performance, particularly against strong, partial, and Base64 encoding encryption techniques, demonstrating an impressive accuracy average above 90%. In contrast, the Random Forest classifier, enriched with a warm-start feature set, prevails with remarkable efficacy, achieving a minimum of 80% accuracy, specifically in scenarios involving intermittent encryption. In the forthcoming phases of our research, the vision is to innovate further by deploying a hybrid model within our developmental framework, intending to fortify the proxy gateway designed to secure endpoints connecting to cloud storage. Data preprocessing involves extracting relevant features from files, a crucial step in enabling classifiers to effectively distinguish between encrypted and unencrypted files. After a comprehen-

sive review of all ransomware techniques outlined in Section 3, we identified and extracted features that can be roughly categorized into three groups: (1) Features related to file entropy; (2) Features related to file size; and (3) Feature related to file content/pattern. We also investigate how Asymmetric Numeral Systems (ANS), which provide both efficient and near-optimal entropy coding, could be leveraged by ransomware developers as part of their tactics [71, 72].

References

- [1] D. Palmer, Ransomware is the biggest global cyber threat. and the attacks are still evolving, zDNet (2022).
URL <https://www.zdnet.com/article/ransomware-attacks-are-the-biggest-global-cyber-threat-and-still-evolving/>
- [2] A. Mahboubi, S. Camtepe, H. Morarji, Reducing usb attack surface: A lightweight authentication and delegation protocol, in: 2018 International Conference on Smart Computing and Electronic Enterprise (ICSCEE), 2018, pp. 1–7. doi:10.1109/ICSCEE.2018.8538400.
- [3] Y. Wang, M. A. Bashar, M. Chandramohan, R. Nayak, Exploring topic models to discern cyber threats on twitter: A case study on log4shell, Intelligent Systems with Applications 20 (2023) 200280. doi:<https://doi.org/10.1016/j.iswa.2023.200280>.
URL <https://www.sciencedirect.com/science/article/pii/S2667305323001059>

Table 8: Summary and Critique of Ransomware Detection Approaches: In this context, **IE** stands for Intermittent Encryption, **PE** stands for Partial Encryption, and **AB** stands for AES-Base64 encryption and encoding.

| Approach | Contributions | Challenges | IE | PE | AB |
|-------------------------------|--|---|----|----|----|
| Behaviour-Based Analysis | - Polymorphism and obfuscation techniques [49] - Efficacy against known threats [48] | - Limited effectiveness against unknown ransomware [50]. | ✗ | ✗ | ✗ |
| Decoy File-Based Deception | - Detection speed and accuracy | - Limited effectiveness when ransomware adapts to recognize decoy files as fake - Lack of deception diversity | ✗ | ✗ | ✗ |
| Communication Analysis | - Detection accuracy and false positive rate [56] | - Limited effectiveness when ransomware does not rely on C&C servers or when dealing with new variants - Limited applicability | ✗ | ✗ | ✗ |
| Hardware Performance Counters | - High detection rate with hardware performance counters [58] | - Resource overhead [57] - Limited applicability - Limited effectiveness against non-resource-intensive ransomware | ✗ | ✗ | ✗ |
| User Activity Monitoring | - Anomaly detection capability [59] - Effective at distinguishing benign activities from ransomware | - Requires substantial training data and may struggle with rapidly evolving ransomware | ✗ | ✗ | ✗ |
| SSDinsider++ | - Early-stage detection on I/O patterns capability [60] | - Practicality limited due to resource-intensive assessment [60] | ✗ | ✗ | ✗ |
| File Entropy Analysis | - High detection accuracy [61] | - Limited ransomware sample coverage [61] - Scalability [61] | ✗ | ✗ | ✗ |
| Deception-Based Strategies | - Intruder enticement and alert triggering effectively [63] | - Limited effectiveness against adaptive ransomware - Limited effectiveness when ransomware evolves to recognize decoy assets | ✗ | ✗ | ✗ |
| Dynamic Analysis | - Effectiveness against evasive ransomware types [66] | - Platform infection risk that necessitates executing ransomware in a controlled environment, which may not always be feasible [66] - Variability among ransomware variants [28], specifically behaviors are specific to individual ransomware variants, requiring generic models or continuous training to adapt to new variants. - Behaviours are specific to individual ransomware variants, requiring generic models or continuous training to adapt to new variants. | ✗ | ✗ | ✗ |
| This paper | - Effectiveness against encryption evasive methods, statistical analysis of content data | - Online learning involves acting as a file share proxy within the file system, such as FUSE, and constantly training on input data from a network shared drive. attack adaptability and the use of not yet known techniques | ✓ | ✓ | ✓ |

- [4] F. Sabbaghi, A. Mahboubi, S. H. Othman, Hybrid service for business contingency plan and recovery service as a disaster recovery framework for cloud computing, *Journal of Soft Computing and Decision Support Systems* 4 (4) (2017) 1–10.
- [5] C.-M. Hsu, C.-C. Yang, H.-H. Cheng, P. E. Setiasabda, J.-S. Leu, Enhancing file entropy analysis to improve machine learning detection rate of ransomware, *IEEE Access* 9 (2021) 138345–138351. doi:10.1109/ACCESS.2021.3114148.
- [6] M. J. May, E. Laron, Combating ransomware using content analysis and complex file events, in: 2019 10th IFIP International Conference on New Technologies, Mobility and Security (NTMS), IEEE, 2019, pp. 1–5.
- [7] J. Lee, K. Lee, A method for neutralizing entropy measurement-based ransomware detection technologies using encoding algorithms, *Entropy* 24 (2) (2022). doi:10.3390/e24020239. URL <https://www.mdpi.com/1099-4300/24/2/239>
- [8] T. McIntosh, J. Jang-Jaccard, P. Watters, T. Susnjak, The inadequacy of entropy-based ransomware detection, in: T. Gedeon, K. W. Wong, M. Lee (Eds.), *Neural Information Processing*, Springer International Publishing, Cham, 2019, pp. 181–189.
- [9] A. Continella, A. Guagnelli, G. Zingaro, G. De Pasquale, A. Barengi, S. Zanero, F. Maggi, Shieldfs: A self-healing, ransomware-aware filesystem, in: *Proceedings of the 32nd Annual Conference on Computer Security Applications, ACSAC '16*, Association for Computing Machinery, New York, NY, USA, 2016, p. 336–347. doi:10.1145/2991079.2991110. URL <https://doi.org/10.1145/2991079.2991110>
- [10] S. Homayoun, A. Dehghantaha, M. Ahmadzadeh, S. Hashemi, R. Khayami, Know abnormal, find evil: Frequent pattern mining for ransomware threat hunting and intelligence, *IEEE Transactions on Emerging Topics in Computing* 8 (2) (2020) 341–351. doi:10.1109/TETC.2017.2756908.
- [11] D. Nallaperuma, R. Nawaratne, T. Bandaragoda, A. Adikari, S. Nguyen, T. Kempitiya, D. De Silva, D. Alahakoon, D. Pothuhera, Online incremental machine learning platform for big data-driven smart traffic management, *IEEE Transactions on Intelligent Transportation Systems* 20 (12) (2019) 4679–4690. doi:10.1109/TITS.2019.2924883.
- [12] A. Mahboubi, K. Ansari, S. Camtepe, J. Duda, P. Morawiecki, M. Pawłowski, J. Pieprzyk, Digital Immunity Module: Preventing Unwanted Encryption using Source Coding (1 2022). doi:10.36227/techrxiv.17789735.v1.

- URL <https://shorturl.at/jGIV9>
- [13] A. Kantee, Rump file systems: Kernel code reborn, in: 2009 USENIX Annual Technical Conference (USENIX ATC 09), USENIX Association, San Diego, CA, 2009.
URL <https://www.usenix.org/conference/usenix-09/rump-file-systems-kernel-code-reborn>
- [14] M. E. Ahmed, H. Kim, S. Camtepe, S. Nepal, Peeler: Profiling kernel-level events to detect ransomware, in: E. Bertino, H. Shulman, M. Waidner (Eds.), Computer Security – ESORICS 2021, Springer International Publishing, Cham, 2021, pp. 240–260.
- [15] A. Continella, A. Guagnelli, G. Zingaro, G. De Pasquale, A. Barengi, S. Zanero, F. Maggi, Shieldsf: A self-healing, ransomware-aware filesystem, in: Proceedings of the 32nd Annual Conference on Computer Security Applications, ACSAC '16, Association for Computing Machinery, New York, NY, USA, 2016, p. 336–347. doi:10.1145/2991079.2991110.
URL <https://doi.org/10.1145/2991079.2991110>
- [16] M. Hirano, R. Hodota, R. Kobayashi, Ransap: An open dataset of ransomware storage access patterns for training machine learning models, Forensic Science International: Digital Investigation 40 (2022) 301314. doi:<https://doi.org/10.1016/j.fsidi.2021.301314>.
URL <https://www.sciencedirect.com/science/article/pii/S2666281721002390>
- [17] M. Hirano, R. Kobayashi, Machine learning based ransomware detection using storage access patterns obtained from live-forensic hypervisor, in: 2019 Sixth International Conference on Internet of Things: Systems, Management and Security (IOTSMS), 2019, pp. 1–6. doi:10.1109/IOTSMS48152.2019.8939214.
- [18] J. Huang, J. Xu, X. Xing, P. Liu, M. K. Qureshi, Flashguard: Leveraging intrinsic flash properties to defend against encryption ransomware, in: Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS '17, Association for Computing Machinery, New York, NY, USA, 2017, p. 2231–2244. doi:10.1145/3133956.3134035.
URL <https://doi.org/10.1145/3133956.3134035>
- [19] H. L. Jeong, S. K. Ahn, S. H. Baek, K.-W. Park, Anomaly detection technology using potential difference displacement detection of data bus., J. Internet Serv. Inf. Secur. 9 (4) (2019) 68–77.
- [20] A. Kharaz, S. Arshad, C. Mulliner, W. Robertson, E. Kirda, UNVEIL: A Large-Scale, automated approach to detecting ransomware, in: 25th USENIX Security Symposium (USENIX Security 16), USENIX Association, Austin, TX, 2016, pp. 757–772.
URL <https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/kharaz>
- [21] S. Mehnaz, A. Mudgerikar, E. Bertino, Rwgard: A real-time detection system against cryptographic ransomware, in: M. Bailey, T. Holz, M. Stamatogiannakis, S. Ioannidis (Eds.), Research in Attacks, Intrusions, and Defenses, Springer International Publishing, Cham, 2018, pp. 114–136.
- [22] S. M. Milajerdi, B. Eshete, R. Gjomemo, V. Venkatakrishnan, Piroit: Aligning attack behavior with kernel audit records for cyber threat hunting, in: Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security, CCS '19, Association for Computing Machinery, New York, NY, USA, 2019, p. 1795–1812. doi:10.1145/3319535.3363217.
URL <https://doi.org/10.1145/3319535.3363217>
- [23] D. Min, Y. Ko, R. Walker, J. Lee, Y. Kim, A content-based ransomware detection and backup solid-state drive for ransomware defense, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 41 (7) (2022) 2038–2051. doi:10.1109/TCAD.2021.3099084.
- [24] N. Scaife, H. Carter, P. Traynor, K. R. B. Butler, Cryptolock (and drop it): Stopping ransomware attacks on user data, in: 2016 IEEE 36th International Conference on Distributed Computing Systems (ICDCS), 2016, pp. 303–312. doi:10.1109/ICDCS.2016.46.
- [25] L. Constantin, New royal ransomware group evades detection with partial encryption, CSO (December 2022).
URL <https://www.csoonline.com/article/574223/new-royal-ransomware-group-evades-detection-with-partial-encryption.html>
- [26] B. Toulas, Ransomware gangs switching to new intermittent encryption tactic, Bleeping Computer (September 2022).
URL <https://www.bleepingcomputer.com/news/security/ransomware-gangs-switching-to-new-intermittent-encryption-tactic/>
- [27] Z. A. Genç, G. Lenzi, D. Sgandurra, On deception-based protection against cryptographic ransomware, in: R. Perdisci, C. Maurice, G. Giacinto, M. Almgren (Eds.), Detection of Intrusions and Malware, and Vulnerability Assessment, Springer International Publishing, Cham, 2019, pp. 219–239.
- [28] A. Kharaz, E. Kirda, Redemption: Real-time protection against ransomware at end-hosts, in: M. Dacier, M. Bailey, M. Polychronakis, M. Antonakakis (Eds.), Research in Attacks, Intrusions, and Defenses, Springer International Publishing, Cham, 2017, pp. 98–119.
- [29] A. Kumbhojkar, Base64 encoding algorithm, published in The Startup, 5 min read, Jan 15, 2021 (2021).
URL <https://medium.com/swlh/base64-encoding-algorithm-42abb92908>
- [30] D. Maguire, C. Davis, M. Penna, Configure client-specific message size limits, microsoft, Article. Accessed: 2023-12-10 (Jan. 2023).
URL <https://learn.microsoft.com/en-us/exchange/configure-client-specific-message-size-limits-exchange-2013-10>
- [31] S. R. Davies, R. Macfarlane, W. J. Buchanan, Comparison of entropy calculation methods for ransomware encrypted file identification, Entropy 24 (10) (2022).
URL <https://www.mdpi.com/1099-4300/24/10/1503>
- [32] K. Lee, S.-Y. Lee, K. Yim, Machine learning based file entropy analysis for ransomware detection in backup systems, IEEE Access 7 (2019) 110205–110215. doi:10.1109/ACCESS.2019.2931136.
- [33] S. R. Davies, R. Macfarlane, W. J. Buchanan, Differential area analysis for ransomware attack detection within mixed file datasets, Computers & Security 108 (2021) 102377. doi:<https://doi.org/10.1016/j.cose.2021.102377>.
URL <https://www.sciencedirect.com/science/article/pii/S0167404821002017>
- [34] A. Kharaz, S. Arshad, C. Mulliner, W. Robertson, E. Kirda, {UNVEIL}: A [Large-Scale], automated approach to detecting ransomware, in: 25th USENIX security symposium (USENIX Security 16), 2016, pp. 757–772.
- [35] A. Mahboubi, S. Camtepe, K. Ansari, M. Pawłowski, P. Morawiecki, H. Aboutorab, J. Pieprzyk, J. Duda, Shared file protection against unauthorised encryption using a buffer-based signature verification method, Journal of Information Security and Applications 86 (2024) 103873. doi:<https://doi.org/10.1016/j.jisa.2024.103873>.
URL <https://www.sciencedirect.com/science/article/pii/S2214212624001753>
- [36] J. Alzubi, A. Nayyar, A. Kumar, Machine learning from theory to algorithms: an overview, in: Journal of physics: conference series, Vol. 1142, IOP Publishing, 2018, p. 012012.
- [37] K. Das, R. N. Behera, A survey on machine learning: concept, algorithms and applications, International Journal of Innovative Research in Computer and Communication Engineering 5 (2) (2017) 1301–1309.
- [38] S. C. Hoi, D. Sahoo, J. Lu, P. Zhao, Online learning: A comprehensive survey, Neurocomputing 459 (2021) 249–289. doi:<https://doi.org/10.1016/j.neucom.2021.04.112>.
URL <https://www.sciencedirect.com/science/article/pii/S0925231221006706>
- [39] A. Toomaj, A. Di Crescenzo, Generalized entropies, variance and applications, Entropy 22 (6) (2020) 709.
- [40] M. H. Hilman, M. A. Rodriguez, R. Buyya, Task runtime prediction in scientific workflows using an online incremental learning approach, in: 2018 IEEE/ACM 11th International Conference on Utility and Cloud Computing (UCC), 2018, pp. 93–102. doi:10.1109/UCC.2018.00018.
- [41] Y. Wu, Y. Chen, L. Wang, Y. Ye, Z. Liu, Y. Guo, Y. Fu, Large scale incremental learning, in: 2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 2019, pp. 374–382. doi:10.1109/CVPR.2019.00046.
- [42] G. Hulten, L. Spencer, P. Domingos, Mining time-changing data streams, in: Proceedings of the Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '01, Association for Computing Machinery, New York, NY, USA, 2001, p. 97–106. doi:10.1145/502512.502529.
URL <https://doi.org/10.1145/502512.502529>
- [43] A. Cocomazzi, Custom-branded ransomware: The vice society group and

- the threat of outsourced development, accessed: 2023-12-10 (2022).
URL <https://shorturl.at/nsxF7>
- [44] J. V. Michalowicz, J. M. Nichols, F. Bucholtz, Handbook of differential entropy, Crc Press, 2013.
- [45] J. Bang, J. N. Kim, S. Lee, Entropy sharing in ransomware: Bypassing entropy-based detection of cryptographic operations, *Sensors* 24 (5) (2024). doi:10.3390/s24051446.
URL <https://www.mdpi.com/1424-8220/24/5/1446>
- [46] G. Y. Kim, J.-Y. Paik, Y. Kim, E.-S. Cho, Byte frequency based indicators for crypto-ransomware detection from empirical analysis, *Journal of Computer Science and Technology* 37 (2) (2022) 423–442.
- [47] J. von der Assen, C. Feng, A. H. Celdrán, R. Oleš, G. Bovet, B. Stiller, Guardfs: a file system for integrated detection and mitigation of linux-based ransomware (2024). arXiv:2401.17917.
- [48] B. Zhang, W. Xiao, X. Xiao, A. K. Sangaiyah, W. Zhang, J. Zhang, Ransomware classification using patch-based cnn and self-attention network on embedded n-grams of opcodes, *Future Generation Computer Systems* 110 (2020) 708–720.
- [49] A. Moser, C. Kruegel, E. Kirda, Limits of static analysis for malware detection, in: Twenty-third annual computer security applications conference (ACSAC 2007), IEEE, 2007, pp. 421–430.
- [50] R. Tian, R. Islam, L. Batten, S. Versteeg, Differentiating malware from cleanware using behavioural analysis, in: 2010 5th international conference on malicious and unwanted software, Ieee, 2010, pp. 23–30.
- [51] F. De Gaspari, D. Hitaj, G. Pagnotta, L. De Carli, L. V. Mancini, The naked sun: Malicious cooperation between benign-looking processes, in: Applied Cryptography and Network Security: 18th International Conference, ACNS 2020, Rome, Italy, October 19–22, 2020, Proceedings, Part II 18, Springer, 2020, pp. 254–274.
- [52] D. Marcus, P. Greve, S. Masiello, D. Scharoun, et al., McAfee threats report: Third quarter 2009 (2010).
- [53] B. Whitham, Automating the generation of enticing text content for high-interaction honeyfiles (2017).
- [54] K. Cabaj, W. Mazurczyk, Using software-defined networking for ransomware mitigation: the case of cryptowall, *Ieee Network* 30 (6) (2016) 14–20.
- [55] O. M. Alhawi, J. Baldwin, A. Dehghantanha, Leveraging machine learning techniques for windows ransomware network traffic detection, *Cyber threat intelligence* (2018) 93–106.
- [56] A. O. Almashhadani, M. Kaiiali, S. Sezer, P. O’Kane, A multi-classifier network-based crypto ransomware detection system: A case study of locky ransomware, *IEEE access* 7 (2019) 47053–47067.
- [57] M. Alam, S. Bhattacharya, S. Dutta, S. Sinha, D. Mukhopadhyay, A. Chattopadhyay, Ratafia: Ransomware analysis using time and frequency informed autoencoders, in: 2019 IEEE International Symposium on Hardware Oriented Security and Trust (HOST), IEEE, 2019, pp. 218–227.
- [58] S. Aurangzeb, R. N. B. Rais, M. Aleem, M. A. Islam, M. A. Iqbal, On the classification of microsoft-windows ransomware using hardware profile, *PeerJ Computer Science* 7 (2021) e361.
- [59] G. O. Ganfure, C.-F. Wu, Y.-H. Chang, W.-K. Shih, Deepguard: Deep generative user-behavior analytics for ransomware detection, in: 2020 IEEE International Conference on Intelligence and Security Informatics (ISI), IEEE, 2020, pp. 1–6.
- [60] S. Baek, Y. Jung, D. Mohaisen, S. Lee, D. Nyang, Ssd-assisted ransomware detection and data recovery techniques, *IEEE Transactions on Computers* 70 (10) (2020) 1762–1776.
- [61] C.-M. Hsu, C.-C. Yang, H.-H. Cheng, P. E. Setiasabda, J.-S. Leu, Enhancing file entropy analysis to improve machine learning detection rate of ransomware, *IEEE Access* 9 (2021) 138345–138351.
- [62] G. O. Ganfure, C.-F. Wu, Y.-H. Chang, W.-K. Shih, Rtrap: Trapping and containing ransomware with machine learning, *IEEE Transactions on Information Forensics and Security* 18 (2023) 1433–1448. doi:10.1109/TIFS.2023.3240025.
- [63] J. Lee, J. Lee, J. Hong, How to make efficient decoy files for ransomware detection?, in: Proceedings of the International Conference on Research in Adaptive and Convergent Systems, 2017, pp. 208–212.
- [64] J. Gómez-Hernández, L. Álvarez González, P. García-Teodoro, R-locker: Thwarting ransomware action through a honeyfile-based approach, *Computers & Security* 73 (2018) 389–398. doi:<https://doi.org/10.1016/j.cose.2017.11.019>.
URL <https://www.sciencedirect.com/science/article/pii/S0167404817302560>
- [65] S. Mehnaz, A. Mudgerikar, E. Bertino, Rvguard: A real-time detection system against cryptographic ransomware, in: International symposium on research in attacks, intrusions, and defenses, Springer, 2018, pp. 114–136.
- [66] K. Cabaj, P. Gawkowski, K. Grochowski, D. Osojca, Network activity analysis of cryptowall ransomware, *Przełąd Elektrotechniczny* 91 (11) (2015) 201–204.
- [67] F. Mbol, J.-M. Robert, A. Sadighian, An efficient approach to detect torrentlocker ransomware in computer systems, in: Cryptology and Network Security: 15th International Conference, CANS 2016, Milan, Italy, November 14–16, 2016, Proceedings 15, Springer, 2016, pp. 532–541.
- [68] S. Song, B. Kim, S. Lee, et al., The effective ransomware prevention technique using process monitoring on android platform, *Mobile Information Systems* 2016 (2016).
- [69] N. Andronio, S. Zanero, F. Maggi, Heldroid: Dissecting and detecting mobile ransomware, in: Research in Attacks, Intrusions, and Defenses: 18th International Symposium, RAID 2015, Kyoto, Japan, November 2–4, 2015. Proceedings 18, Springer, 2015, pp. 382–404.
- [70] A. Mahboubi, K. Ansari, S. Camtepe, Using process mining to identify file system metrics impacted by ransomware execution, in: S. Bouzefrane, M. Laurent, S. Boumerdassi, E. Renault (Eds.), Mobile, Secure, and Programmable Networking, Springer International Publishing, Cham, 2021, pp. 57–71.
- [71] S. Camtepe, J. Duda, A. Mahboubi, P. Morawiecki, S. Nepal, M. Pawłowski, J. Pieprzyk, Ans-based compression and encryption with 128-bit security, *International Journal of Information Security* 21 (5) (2022) 1051–1067.
- [72] J. Pieprzyk, J. Duda, M. Pawłowski, S. Camtepe, A. Mahboubi, P. Morawiecki, The compression optimality of asymmetric numeral systems, *Entropy* 25 (4) (2023). doi:10.3390/e25040672.
URL <https://www.mdpi.com/1099-4300/25/4/672>