JOURNAL OF CYBERSECURITY

Journal of Cybersecurity, 2025, tyaf009
https://doi.org/10.1093/cybsec/tyaf009

Research Paper

Not on my watch: ransomware detection through classification of high-entropy file segments

Fran Casino [1,2], Darren Hurley-Smith [3], Julio Hernandez-Castro [4], Constantinos Patsakis [2,5,*]

[1]Department of Computer Engineering and Mathematics, Universitat Rovira i Virgili, Avinguda dels Països Catalans, 26, 43007, Tarragona, Spain
[2]Information Management Systems Institute, Athena Research Centre, Artemidos 6, Marousi 15125, Greece
[3]University of Kent, Giles Ln, Canterbury CT2 7NZ, United Kingdom
[4]Universidad Politécnica de Madrid, Alan Turing, s/n, 28031 Madrid, Spain
[5]Department of Informatics, University of Piraeus, 80 Karaoli & Dimitriou str., 18534 Piraeus, Greece

*Corresponding author. Information Management Systems Institute, Athena Research Centre, Artemidos 6, Marousi 15125, Greece. E-mail: kpatsak@athenarc.gr

Received 20 March 2024; revised 31 December 2024; accepted 27 February 2025

Abstract

The double-edged sword of continuous digitization of services and systems opens the door to a myriad of beneficial opportunities, as well as challenging threats. Currently, ransomware is catalogued as the first threat in cybersecurity due to its impact on organizations, critical infrastructure, industry, and society as a whole. Thus, devoting efforts toward developing methodologies to effectively prevent and mitigate ransomware is crucial. In this article, we present an accurate method to identify encrypted bit streams by differentiating them from other high-entropy streams (e.g. compressed files), which is a critical task to detect potentially malicious file write events on the file system in current operating systems. After extensive evaluation, our findings demonstrate that the proposed solution outperforms the current state of the art in both adaptability and accuracy, enabling it to be integrated into current Endpoint Detection and Response systems.

Keywords: ransomware; high-entropy sources; endpoint detection and response systems; randomness; encryption

Introduction

Ransomware has become a serious concern for modern organizations, resulting in direct costs of hundreds of thousands of dollars (USD) and severe service disruptions. Additionally, reputations may be harmed, and down-chain costs caused by the cessation of services can reach millions. In 2023, the average recovery cost was slightly above $2 million according to a report surveying over 3000 companies across 14 countries [1]. Even more alarmingly, by 2031, ransomware is estimated to cause damages of $265 billion [2]. In most cases, the modus operandi (Fig. 1) is relatively straightforward: an adversary penetrates the network of an organization, either by exploiting a vulnerability or through phishing emails, and then performs lateral movement to find as many possible hosts and servers to encrypt with a subset of their files with a random key and a strong encryption algorithm [3]. Next, the adversary leaves a notice to the victim notifying how the ransom can be paid to receive a decryptor to recover the file. On specific occasions, e.g. MAZE, CONTI, adversaries may use a "double extortion" scheme in which, apart from the extortion to recover the encrypted files, they also threaten the victim to disclose files with sensitive information. In other cases, the attacker may also threaten to communicate with clients or perform a denial of service attack to put even more pressure on the victim and pay the ransom. The above is part of a general crime scheme that operates under the *Malware as a Service* model in which criminals outsource services and products, e.g. malware, hacking tools, to their peers [4].

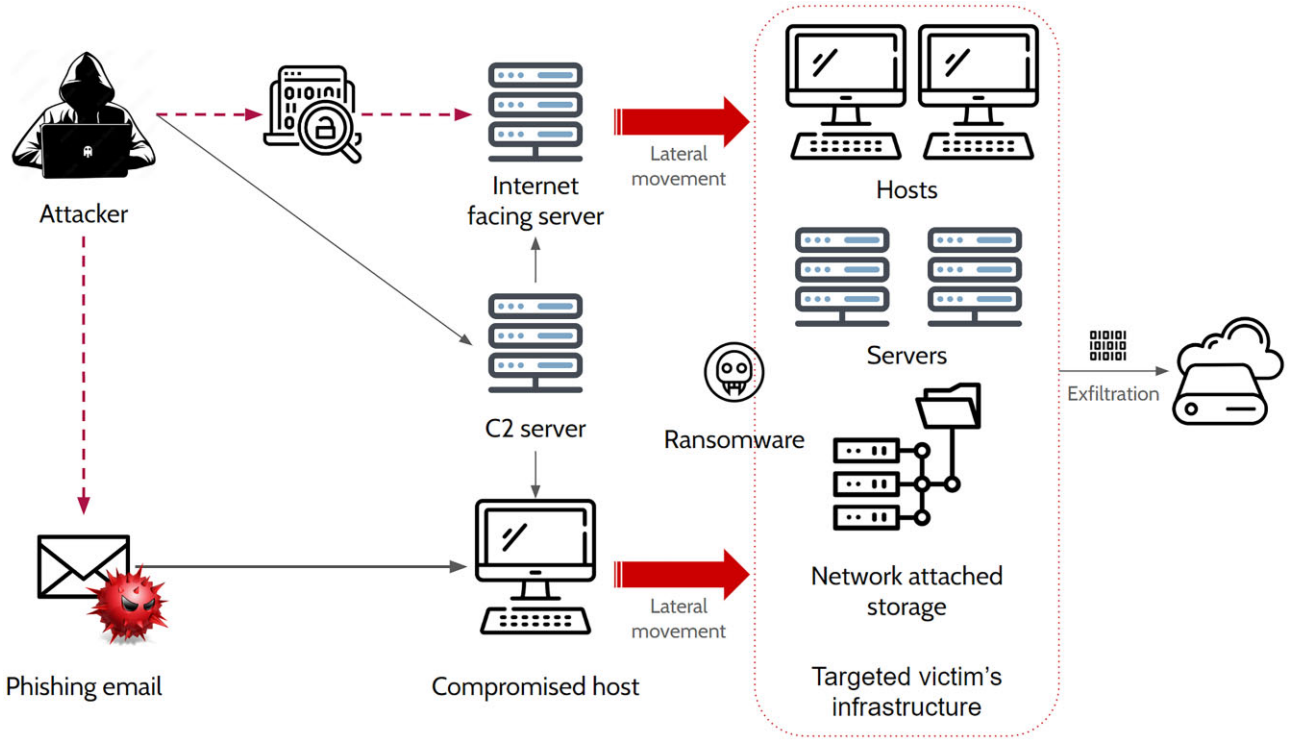


Figure 1. Ransomware modus operandi.

The primary objective of this work is the early detection and blocking of the encryption process of ransomware to prevent the encryption of more files, considering that an adversary has managed to get access to an endpoint and has established a connection to her C2 server; stage 6 of the cyber kill chain. There are many options in the literature to achieve this objective, e.g. hook cryptographic-related API calls, monitor the status of honey files, and prevent deletion of shadow copies. Nevertheless, common practice has shown that threat actors bypass these measures. We argue that the most effective way to achieve this is to monitor file system changes and determine whether the new created files are encrypted. To resolve this, one must be able to efficiently distinguish encrypted files from those that are not. Clearly, due to their contents, such files look more random and essentially have high entropy. The core idea is that if one monitors file system changes and investigates the write events, one could identify ransomware attacks, as the encrypted files would write more files with content looking "random" than others.

However, such an approach is inefficient, prone to high false positives, and introduces significant performance overhead. Clearly, processing each file that is modified in a system, and each time this happens, is a huge computational overhead that may render a computer unresponsive. Sampling methods may alleviate this problem by analysing fragments of the modified files and basing their assessment on the extracted sample. Nevertheless, while entropy is a good indication that something looks random, using it alone is not an accurate measure to determine that a file is encrypted. The reason is that high-entropy files are not only encrypted files but also compressed files. Note that the term compressed files does not refer only to traditional compressed files, e.g. ZIP, GZ, and RAR. Many audio and video file formats, as well as MS Office, JAR, and other proprietary formats, use compression to shrink the file size. As a result, additional metrics and tests must be performed on the extracted sample to prevent false alerts when processing such files.

In this work, we first analyse the accuracy and performance of several randomness tests in distinguishing encrypted from nonencrypted files, to distil a set of them to be used as features to train machine learning models. By properly selecting these tests, we may accurately and timely classify encrypted and nonencrypted data streams in real time. In addition, we provide a dataset with an equally distributed number of file types, which can be used as a benchmark by researchers to assess the utility of our proposal and facilitate comparisons with the state of the art and research in the field.

The rest of this article is organized as follows. The next section provides the reader with the relevant background on ransomware detection and existing approaches. Section Proposed approach describes our method, the dataset creation, and the feature extraction methodology. Section Experiments is devoted to the experimental setup and the tests performed to evaluate our method. Beyond merely theoretically testing our approach, we test and validate our proof of concept approach using recent malware samples. Section Discussion analyses the outcomes and findings. Finally, section Conclusion summarises the contributions and identifies potential directions for future research.

Background

In this section, we describe the most widely used randomness evaluation methods to determine whether a specific data stream is encrypted, with special regard to the tests used in this article.

Entropy

According to Information Theory [5], the measurement of entropy sets a numeric value on the unpredictability of data of an informa-

tion source. Entropy tests, however, tend to focus on either the identical or independent distribution properties of a sequence. Therefore, the higher the entropy, the more independently (Correlation between Successive Entropy Values [6]) or identically (Shannon-entropy) distributed bits are in the sample.

Chi-square test

This test measures the uniformity of the distribution of 1-byte or 1-bit expressed as integers within any $\mathbb{Z}(A)$. This method determines how much a dataset conforms to a particular distribution, expressed as an absolute value alongside a confidence percentage $\chi\%$. There exist three possibilities [7]:

(1) $1\% > \chi\%$ or $\chi\% > 99\%$, the stream is not random.
(2) $1\% < \chi\% < 5\%$ or $95\% < \chi\% < 99\%$, the stream is "suspected" to be random.
(3) $5\% < \chi\% < 10\%$ or $90\% < \chi\% < 95\%$, the stream is likely not to be random.

As can be seen, this is a two-tailed test; it is a common error to assume that only extremely low-confidence values indicate nonrandomness. In this manner, unlike Shannon-entropy measurements, the Chi-square test can identify any excessively uniform $\mathbb{Z}(A)$, such as a counter (which would report a high Shannon-entropy).

Autocorrelation test

This test analyses correlations to discover cyclic or periodic patterns. The randomness is computed by calculating autocorrelation for the values of the data stream at different time lags [8]. Correlation values close to zero correspond to a highly random pattern.

Jarque–Bera test

This test matches the skewness and kurtosis of the data and compares them with the values of a normal distribution. More concretely, the null hypothesis is a joint hypothesis that assumes zero values for skewness and excess kurtosis, as expected from a normal distribution. Any deviation from this increases the Jarque–Bera statistic, which results in values far from zero if the data do not follow a normal distribution.

Shapiro–Wilk test

The Shapiro–Wilk test [9] estimates whether a random sample comes from a normal distribution. The null hypothesis of this test assumes that the population is normally distributed. Thus, the hypothesis is rejected when the P-value $\leq .05$ (threshold value), denoting that the data tested are not normally distributed.

Kolmogorov–Smirnov test

Given two empirical cumulative distribution functions, this test quantifies the maximum absolute difference between them as a measure of disagreement [10].

Anderson–Darling test

This test is closely related to the Kolmogorov–Smirnov test. However, it performs better when applied to small data streams as seen in [11].

Monobit test

This computes the balance of ones and zeros in a bit stream. Given a sequence of n bits, it tests whether:

$$erfc\left(\frac{|\#zeroes - \#ones|}{n\sqrt{2}}\right) < 0.01.$$

Poker test

This test evaluates the number of repetitive patterns found in a data stream. First, the input is split into 4-bit segments, which belong to [0,15] when converted into an integer. Let us denote as $f(i)$ the occurrences of each number i. Next, we evaluate:

$$X = \frac{16}{5000}\sum_{i=0}^{15} f(i)^2 - 5000.$$

According to FIPS-2-140, the test is passed if $2.16 < X < 46.17$.

Runs test

This test analyses consecutive patterns in a bit stream. Each run is denoted as a set of consecutive bit patterns, which are counted to assess the proportion of repeating patterns.

Long runs test

This test determines if there are runs of length above 25, which could denote nonrandom bit streams.

FIPS-2-140 test

This test is a set of four empirical experiments to analyse the randomness of binary data streams. For our experiments, we utilize the FIPS-2-140 cryptographic module test with a minimum block size of 20 000 bits. This ensures that the tests are independently applied to each data block. The set comprises the monobit, poker, runs, and long runs tests described above.

The definitions for SP800-22 can be found in the official documentation for NIST SP800-22 [13]. Full equations and test descriptions are available in that document. We provide a short-form overview of the four tests used below:

Block frequency

This test identifies the number of 1's within an M-bit block. It is effectively a χ^2 test on each M-bit block in a sequence of size N. The proportion of blocks for which an approximately identical distribution (0.5) of 1s is identified determines whether this test passes or fails. A minimum input size of 100 bits is recommended, and M must be some smaller value by which N can be exactly divided.

Frequency (Monobits) test

Similar to FIPS-140-2 described above, this test counts the occurrences of ones and zeroes throughout the tested sequence, and checks for an identical distribution (0.5 rate of occurrence for both 0 and 1). This test requires a minimum of 100 bits to function, and fails if P-values resulting from the test fall below .01.

Overlapping templates test

This test uses an M-bit sliding window over an N-bit sequence to search for specific M-bit patterns. This is effectively a test for the

occurrence of specific M-bit strings within a sample. Should the M-bit pattern not be identified within M, M shifts by one bit position in the N-bit sequence. Unlike the nonoverlapping template test, it also slides by only one bit position if the pattern is found. A table of occurrences of values in the set M is produced, and the frequency of specific patterns within an N-bit sequence is identified.

Cumulative sum

This is a test of excursion from a normal value (0 at initialization of the test). If a 0 is detected, the index is incremented by 1, and a new value of $x - 1$ is recorded. If a 1 is detected, the index is incremented by 1 and $x + 1$ is recorded. This process is repeated for the length of N in the input sequence. This test is a form of random walk, which will detect some forms of nonrandomness, where sequences have an over large, or periodic excursion from the normal.

Related work

It is the ideal of every organization's information security team to identify and prevent an attack before it has a chance to execute its intended operations on a target system [14]. However, this is easier said than done. Signature detection, even when enhanced with semantic models and machine learning, cannot identify all potential attacks [15,16]. Furthermore, the most sophisticated solutions are restricted (by cost, required expertise, and resources) to the best-equipped organizations. Proof of prevention for a piece of malware is not proof of prevention for the entirety of a ransomware attack. Contemporary ransomware is a composite affair, with multiple infection vectors, highly variable malware loaders, and further malware that executes in parallel with crypto-ransomware binaries [17]. The latter is perfectly illustrated in the latest "Ransomware Activity Report" [18] highlighting that since January 2020, the uploaded malware samples belong to 130 unique families. Notably, these samples can be grouped into more than 30 000 different similarity clusters. Sophos has identified that attacks are increasingly targeted, a trend first observed in 2018 that has become the dominant form of crypto-ransomware extortion since 2021 [1,19]. Such attacks include data theft (for resale) alongside the ransomware itself. This makes the prevention of malware on an individual basis a poor guarantor of continued system integrity.

As a result, in-line monitoring of the file-system state has received increasing attention from the security community. The randomness of encrypted files has often been used to detect ransomware attacks in progress [20–22]. The core idea is that one may detect unusual increases in entropy on a local file system to identify whether mass encryption is underway. Then, accounting for any scheduled encrypted backups and other mass encryption operations, discern whether this entropy increase is legitimate or potentially malicious. Theoretically, this would allow one to identify a crypto-ransomware attack in progress, potentially stopping it before many files have been encrypted.

Nevertheless, there are several criticisms of this line of research. For instance, both McIntosh et al. [23] and Pont et al. [24] discuss why such approaches are inefficient in practical applications. The simple fact is that organizations are combating ransomware, hitting a threshold value of encrypted files. Value, as opposed to a number, as once business-critical operations are significantly disrupted, the attacker has the leverage required to demand ransom. This threshold can be understood as a "denial of capability." Computational efficiency is low in the proposed statistical approaches: a purely statistical approach to entropy measurement is prone to false positives where data compression and encryption are both expected on some scale. Web services often use compression for media to reduce the throughput resulting from queries to their servers. Formats such as WEBP are extremely hard to differentiate from encryption using the few efficient statistical test batteries (FIPS 140-2, some subtests of NIST SP800-22) [25]. Even JPEG, RAR, and ZIP files can be complex for purely statistical approaches to differentiate from encrypted data without further calls for file-specific metadata. These deficiencies result in a slow, inaccurate classification that may identify encryption in progress but not in a timely enough manner to prevent a malicious actor from achieving the aforementioned "denial of capability": the point at which an organization must consider whether or not to pay the ransom to restore services [26].

The open problem addressed by the authors in this article is not the design of novel tests but the implementation of novel, high-speed classification of randomness subtypes to differentiate legitimate processes from potential ransomware activity. Tools like Paybreak [27] rely on process hooking to identify interactions in dynamically linked libraries to detect and attempt to thwart calls to system crypto-libraries. This does not; however, solve the problem of obfuscated calls or ransomware, which leverages trusted execution environments to hide their activities [26]. To prevent file restoration, the bulk of ransomware deletes the shadow copies in Windows systems. Thus, Raccine [28] intercepts such calls to `vssadmin` and kills the invoking process.

Beyond ransomware detection, the level of interest in distinguishing between encrypted and compressed data streams has risen significantly. This can be partly attributed to the continuous integration of end-to-end encryption in online communications. Commonly employed approaches involve measuring Shannon entropy or the chi-square test on fixed-size data segments (e.g. 1, 2, and 4 KB) to differentiate various data types, including compressed and encrypted data. However, when there are limited samples, the entropy estimation approach is ineffective [29–31]. Moreover, high dependence on entropy measures is not the best option in the presence of other high-entropy data sources. Typical examples of such sources are compressed files, MP3, PDF, or even MS Office files. Beyond the accurate classification of high-entropy files, the time and resource allocation for this operation in a continuous monitoring setup presents one of the most challenging aspects of the problem. Clearly, such monitoring at the network and file system level may introduce serious processing bottlenecks if the necessary computations cannot be performed efficiently enough.

Beyond ransomware detection, the same issue is also found in traffic analysis [32]. Existing mechanisms rely on continuous traffic monitoring, information about *complete* packet transmission, the beginning (e.g. magic headers) and the end of a connection or a file, and so on. Evidently, real-time monitoring is inefficient using such strategies due to the huge volumes of data to be analysed. However, these strategies demonstrate their usefulness when examining previous events or focusing solely on specific connections. Therefore, we harness the most accurate yet fast to extract and process features to unleash real-time classification when monitoring the payload of randomly selected packet fragments. In the literature, there are a few approaches in this research line, such as [33–38].

Proposed approach

In the following paragraphs, we define the capabilities and requirements of our proposal. First, we present the basic assumptions and model. On this ground, we discuss efficient methods for implement-

ing the proposed solution on Windows systems, which is the primary target of ransomware attacks. Then, we detail the feature selection procedure and the methodology adopted for our study.

Basic architecture

As already discussed, our goal is to keep track of the file system changes and determine whether a binary is writing encrypted files. Due to the amount of file system changes in modern operating systems, this effort is rather high, and not all changes can be monitored in terms of their whole content. Moreover, some processes are expected to write encrypted files, e.g. system processes may perform memory dumps, browsers and other processes may keep sensitive data in encrypted files, and so on. Therefore, detecting encrypted content does not necessarily mean that the process belongs to malware. The decision must be based on the context of the write operations. Note, however, that the latter is difficult as malware sometimes injects into legitimate processes, and, in the case of Windows, the Controller Access Folder feature may be tampered with malicious intent. The basic factors can be stripped down to which process, of which binary writes an encrypted file, to which folder, and how many times this has been performed already. Note that for efficiency, we want file segments. One may argue that since some file formats may locally contain an encrypted fragment (e.g. a digital signature), more than one fragment may have to be collected to determine whether a file is encrypted. However, we argue that, while this is true, such fragments are highly unlikely to span across the length of the segments we consider in our experiments.

Based on the above, in our model, we assume that a service S, which runs in the background with kernel-level permissions, monitors file system changes. To prevent computational overheads, S monitors only file write events and may also have a whitelist of processes that are not monitored, based, of course, on the location of their binaries and or hashes. For each file write event, S keeps track of the process `p` with process ID `pid`, the user who initiated this process, its filename and path, as well as the filename and path of the affected file. This information is compared against a policy table PT, which will determine whether this change has to be further assessed. Then, S collects a sample of size SZ of the affected files and uses a set of methods to extract the necessary features from the file fragment. These measurements are then fed to a pretrained model to determine whether the affected file contains encrypted content. Should this be the case, S raises a counter, which is `pid` specific. Once the counter exceeds a threshold T, S considers that `p` is a ransomware-related process and pauses `p` and all the children of `pid`. Note that using fragments of size SZ prevents continuous computations, as changes to small files and minor file changes to the file system can be omitted, resulting in less resource consumption.

For Windows-based systems, S can be implemented with minifilter drivers [39], which have kernel-level access and may efficiently collect I/O changes without introducing significant computational overhead. Due to its access, such a driver not only monitors the file system changes but can also determine which is the process of making the change and pausing or even killing it. It should be noted that PT may have fine-grained policies that prevent the usage of specific processes from specific users or apply constraints based on time or type of access (e.g. RDP). Similarly, PT may whitelist specific actions from users/processes, significantly reducing the number of samples that must be collected by S.

The basic flow is illustrated in Fig. 2. In essence, a file system watcher monitors I/O changes. Once a change is performed, an event is triggered, and the watcher collects the user who performed the action, the process ID of the processes performing the action, the name of the process, its path, the path of the affected file, and the performed filesystem action; that is delete, write, and so on. This information is compared against a policy table which contains a set of rules. For instance, the table contains a list of binaries, which are allowed to perform specific actions in specific folders. If the process performing these changes does not conform to these rules, a scanning process is requested. In this scanning, a file segment is selected and the corresponding features are extracted. The features are then passed on to our trained machine learning to assess whether the file is encrypted. Should the model determine that the file is encrypted, since the process does not belong to the ones with the allowed policies, the process with the corresponding PID is killed.

Finally, we define a set of desired properties to be fulfilled by our method that will guide the selection process of the features and machine learning models:

- Accuracy: the proposed method should be able to accurately distinguish encrypted from nonencrypted high entropy data segments.
- Efficiency: the outcomes must be fast and robust to allow real-time responses.
- Adaptability: the proposed method must be versatile to allow customization and fine-tuning of parameters/features, depending on the size of the samples, to enable lightweight and faster classification.
- Reproducibility: ensuring easy deployment and verification, the proposed methodology should be user-friendly and allow seamless integration with existing solutions. To this end, we use state-of-the-art methods and a set of rigorously defined strategies for the collection, analysis, and subsequent classification of the bit streams.

Benchmark dataset

As a typical procedure, the creation of a statistically sound benchmark is crucial to ensure the robustness of the outcomes [36]. Hence, we use files collected from reputable and widely used sources, detailed in Table 2, to create a benchmark dataset. We have carefully curated our dataset to include an equal number of files for each file type (text, image, binary, video, audio, and PDF) to eliminate possible biases. Note that the randomness of compressed files depends on the original uncompressed file (see Section 6). Moreover, we highlight that the selected file types are the most representative according to the state-of-the-art [34,36,40,41].

Next, since one of the goals is to classify high entropy data streams, we generate a set of fixed-size compressed and encrypted bit streams that range from 64 up to 1024 KB, including all intermediate powers of 2. As several encryption and compression methods are used in real-life scenarios, we selected a set of widely used ones and summarized them in Table 1. The dataset creation procedure is described in Algorithm 1. In the context of Algorithm 1, both arrays *Sizes* and *Methods* employed in *line 1* correspond to bit stream sizes ranging from 64 to 1024 KB, and the components of Table 1, respectively. Once we apply Algorithm 1 to the data extracted from the benchmarks described in Table 2, the resulting output is an array of datasets (one for each file size), each storing ∼1.2 GB of compressed and encrypted files. More concretely, for each file type, we first generate 100 MB of encrypted files and 100 MB of compressed files, that will be later split iteratively from 1024 to 64 KB sizes, conforming to a total of 6 GB, which is the size of the whole dataset.

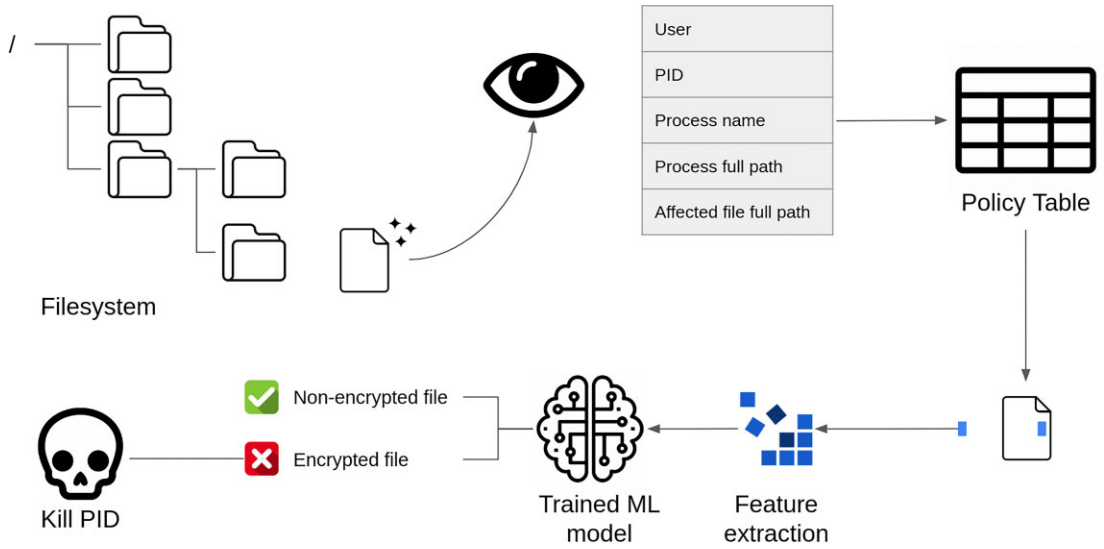


Figure 2. Overview of the proposed approach.

Table 1. Selected encryption and compression methods. Considering the possible combinations, each input generates 10 different new files.

Encryption	Compression
AES (128 / 192 / 256) Camelia (128 / 192 / 256)	ZIP RAR BZIP2 GZIP

Table 2. Source datasets used to obtain the randomly selected files for our benchmark.

File type	Benchmark
`IMG`	COCO Dataset [42]
`IMG`	Microsoft RGB-D Dataset 7 Scenes [43]
`PDF`	ArXiv.org e-Print Archive
`TXT`	Project Gutenberg [44]
`MP3`	Several classical music symphonies
`VIDEO`	YouTube-8M [45]
`BIN/EXEC`	`%SystemRoot%\System32` in Win10 x64 `/sbin` in Ubuntu 16.04

To achieve a balanced representation of files and avoid biases during experimentation, we use random selection to form a subset with an equal number of encrypted and compressed files for our experiments. Therefore, we generate five datasets (i.e., one for each file size denoted as 64, 128, 256, 512, and 1024), each composed of exactly the same number of encrypted and compressed files. Note that, in the case of extremely unbalanced samples, we could obtain a very high overall accuracy, yet it could cover the fact that the method cannot classify correctly the underrepresented class, which is a classical problem [46]. Therefore, despite the potential real-world prevalence of imbalanced datasets, such imbalance can hinder the robustness of the performance of learning systems [47], and thus, several balancing strategies can be used [48]. In our case, as we created the dataset, we opted for a balanced one, to fulfil the previous claims in the literature. We made our dataset available in Zenodo [49] to ease further comparisons and ensure the reproducibility of the results.

Feature selection

After assessing the randomness tests, only some of them met the requirements to be adopted in our system based on their performance and computational cost. In particular, the chi-square test proved to be an efficient mechanism to distinguish true identically distributed random bit streams from nonrandom ones. Moreover, the chi-square test achieved better accuracy than other distribution-based methods such as Kolmogorov–Smirnov and Anderson–Darling. It is; however, unable to identify correlations between bits in a sequence without substantial further analysis [50]. FIPS-140-2 tests are efficient and reliable for ensuring that encrypted bit streams exhibit basic levels of bit-level independence and identical distribution. However, FIPS-140-2 is known to be a poor detector of partially structured data: the poker and run tests cannot differentiate between partially embedded counters and truly ransom sequences [25]. However, they are extremely fast and can detect egregious divergences from randomness, making them an ideal component of our statistical test battery. Furthermore, the Monobit test can help identify potential bias in a file or Random Number Generator output, providing a simple and fast method of checking whether 1 or 0 occurs more frequently. This does not allow for any detailed characterization of said bias, but is a good first step and low enough in computational resource costs that it is worth including as the first stage of more rigorous statistical testing.

After selecting a set of unique tests (i.e., SP 800-22 and FIPS-140-2 have overlapping tests), we tested a subset of SP 800-22 tests according to their accuracy and efficiency in distinguishing between encrypted and compressed files. The frequency within a block test, the approximate entropy test, and the cumulative sums test were selected for inclusion in the classification tool set due to their performance in preliminary tests.

The average and correlation tests exhibited slightly better accuracy and less variability in encrypted streams than in compressed ones. However, they failed to provide reliable results in the case of high-entropy small data files. The remaining methods were not selected since either (1) they were unreliable—both encrypted and com-

Algorithm 1 Database Generation.

```
1: function CREATE DATASET(DataSet D, Array Sizes, Array Methods)   ▷ The bit stream sizes and the set of comp. and enc. methods
2:
3:     while (FilesToProcess) do
4:         f_i = SelectTheNextFile(D);                              ▷ Next source raw file
5:         V = CreateFileVariants( f_i, Array Methods);             ▷ Processes f_i to create Encrypted and Compressed variants
6:         S = SplitFiles(V, Array Sizes);                          ▷ Files in V are split into different sizes
7:     end while
8: end function
```

Table 3. Initial set of features used in our approach.

Notation	Description
`Shannon-Entropy`	The entropy of the sample
`Jarque-Bera`	The Jarque–Bera statistical test
`Shapiro-Wilk`	The Shapiro–Wilk statistical test
`Block_freq`	NIST SP800-22
`Freq_average`	NIST SP800-22
`Cumu_sum`	NIST SP800-22
`Overlapping_template`	NIST SP800-22
`Chi_score`	The Chi-square statistical test
`Monobit`	Monobit test, as part of the FIPS-140-2 test
`Long Run`	Long run test, as part of the FIPS-140-2 test
`Poker`	Poker test, as part of the FIPS-140-2 test
`Run`	Runs test, as part of the FIPS-140-2 test
`Fips_out`	The final outcome of the FIPS-140-2 test

pressed files had indistinguishable results from a statistical point of view—or (2) their computational cost is prohibitive. For instance, a subset of the Diehard tests (birthday spacing, parking lot, and random spheres) provide meaningless outcomes since they are usually passed by both compressed and encrypted files. Additionally, Diehard tests and many tests in the Crush batteries of TestU01 [51] require considerable computational resources, and, as such, they are not efficient enough to be used for real-time purposes. The selected features are detailed in Table 3.

Experiments

Feature analysis

In this section, we analyse the values obtained by the features described in Section 4.3 for each database. As it can be observed, this analysis does not include the battery of FIPS tests, which are analysed in Section 5.2. Therefore, Figs 3–5 represent the different features' values for all datasets. Since the distribution of values in the case of encrypted files was very stable across the different file types, we considered all the encrypted files of each dataset to compute them. Note that each dataset was normalized before computing all values; thus, the values range between 0 and 1.

In the case of the 64 KB dataset (cf. Fig. 3), we observe that the binary and image compressed files have similar values for all features, which are close to 0 for `Block_freq`, `Freq_average`, and `Cumu_sum`. The compressed PDF, Video, and TXT files show higher values in the aforementioned features. Finally, MP3 compressed files obtain the highest range of values in the `Freq_average`, and `Cumu_sum` tests. The 128 KB dataset values are similar to those obtained in the 64 KB dataset. The most noticeable difference is the higher range of values obtained by `Shapiro-Wilk` and that the `Overlapping_template`'s range is the highest among all features. According to the previous observations and the values of the 256 KB dataset (cf. Fig. 4), the range of values of `Shapiro-Wilk` grows according to the file size. Moreover, we can observe that the range of `Entropy` and `Chi_score` values for the 256 KB dataset is also higher than in previous file sizes. In the 512 KB dataset, we observe a notable growth in the value range of `Entropy`, `Shapiro-Wilk`, `Jarque-Bera` and `Chi_score`. Interestingly enough, the range values of other features such as `Block_freq`, `Freq_average`, `Cumu_sum` are reduced with respect to smaller file sizes. Finally, the growth patterns observed in the 512 KB are extended in the case of the 1024 KB dataset (cf. Fig. 5).

FIPS-140-2 test analysis

The well-known FIPS-140-2 (*rng-tools rngtest* utility in Linux) is a battery of four tests. Their efficacy in distinguishing between compressed and encrypted small file size bit streams has been proven in the past [36]. However, an analysis of the performance of each of the four FIPS-140-2 tests has not been done previously.

To analyse the accuracy of the FIPS-140-2 battery, we created a uniform factor, namely a γ factor, to relax the threshold values of such a test by applying them as a multiplier of each test's boundary values. For instance, given a boundary value a, the new value will be set as $a + a * \gamma$, or $a - a * \gamma$ in the case of a lower bound. The aim of such γ factor is to find the optimal relationship between the different file sizes evaluated in this article and the strictness of FIPS-140-2. Therefore, we applied the FIPS-140-2 test according to a set of γ values to all datasets and depicted the outcomes in Fig. 6.

As it can be observed, the values of γ modify the outcomes according to each file size. Overall, strict values (i.e. low values of γ) yield the best outcomes, except in the case of the Run test (cf. Fig. 6e), which affects the overall outcome of the test (cf. Fig. 6a). Since we applied a fixed set of γ values, the thresholds are modified by multiplying the boundaries of each test. Thus, in the specific case of Long Run (cf. Fig. 6c), the upper threshold value is 26 until $\gamma = 0.04$, in which case it changes to 27, which explains the sudden reduction in accuracy.

The outcome of the analysis is quite straightforward. Even though the Runs test can be relaxed to obtain a more efficient outcome for large files, the truth is that the most accurate test is Monobit. More concretely, Monobit achieves better accuracy in its most strict form (i.e., $\gamma = 0$, thus the normal setup) than the rest of the tests in any configuration and file size. Note that we tried negative γ values, but the accuracy decreased dramatically, and thus, they were not included in the possible range of values.

Therefore, given the outcome of this experiment, we enhanced our approach as follows: (i) we increased the accuracy due to the reliability of Monobit by discarding the rest of FIPS-140-2 outcomes, and (ii) we reduced the number of computations and features of the system, improving the performance of the classification models.

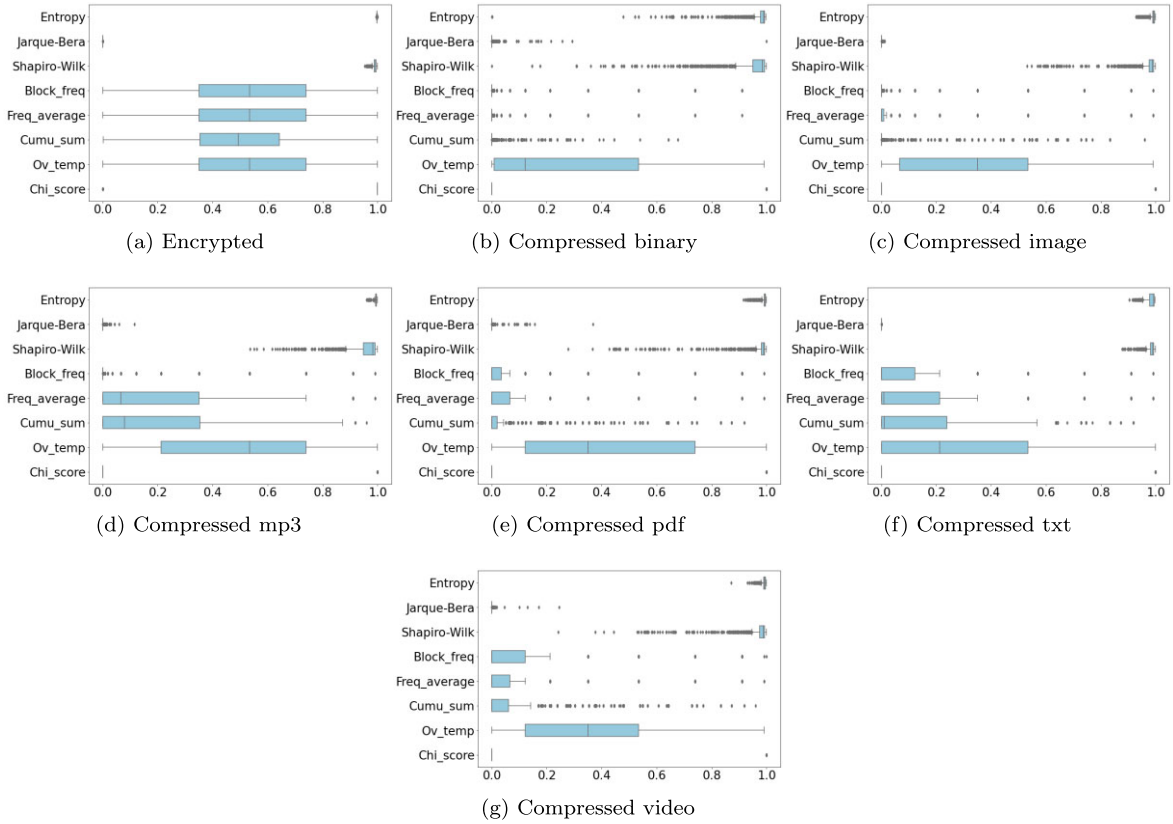


Figure 3. Detail of the features' values for 64 KB files. *x*-axis correspond to the normalized values between 0 and 1 for each feature measured in *y*-axis. The larger the box, the higher the feature's variability.

Classification experiments

To measure the capabilities of our proposed features for binary classification (i.e. to differentiate between compressed and encrypted samples), we selected some well-known and widely used machine learning methods. We used a Support Vector Classifier (SVC), a nonparametric ensemble classifier (Random Forest), and XGBoost, which implements gradient-boosted decision trees.

The selected classification methods are shown to be efficient (i.e. requiring minimal processing of the input data) and accurate with both small and large feature sets, being less prone to overfitting than deep learning methods, especially when working with tabular data [52].

We tuned the hyperparameters with grid search, using 10-fold cross-validation over the full dataset in an independent experiment to maximize classification performance. Table 4 describes the features that achieved the best performance. In the case of the SVC model, the best-performing configuration was achieved when using a radial basis function (RBF) kernel. We used 10-fold cross-validation and repeated the experiments three times to get an unbiased estimate of the performance of the predictive models.

For this and the rest of the experiments performed in the article, we selected the popular platform Google Colab[1] in its free version (2x 2.3GHz CPU and 12 GB of RAM), while we utilized the implementations of the `scikit-learn`[2] library, to ensure the replicability of our experiments. Such specifications establish a baseline that can be easily outperformed by that low-mid performance desktop computers. For the sake of reproducibility of the experiments, the sources are available on GitHub[3].

To simplify comparisons, we use standard classification metrics to evaluate the performance of the trained classifiers. Hence, the outcomes achieved by each model were computed in terms of precision, recall, accuracy, and F_1 score, and are summarized in Tables 5 and 6. As can be observed, the results improve according to the file size, with Random Forest and XGBoost exhibiting the best performance compared to the SVC model, which is clearly outperformed in all experiments. The low values of standard deviation obtained by all the classifiers indicate the robustness of the selected features.

In general, according to individual experiments, the most difficult file types to capture are Video, Image, and MP3, while the easiest ones are Binary and TXT. The misclassification errors occur exclusively in the case of compressed files (i.e. the precision in the case of encrypted files is 100%), which sometimes exhibit values that resemble those of encrypted files. Therefore, the system is able to capture all encrypted files due to the specific range of their feature's values, which is the level of restriction that we aimed for in our system (i.e. misclassifying an encrypted file would incur further security issues than misclassifying a compressed file).

Regardless of the file type, the outcomes show minimal errors on average in 256 KB files (i.e. average F1-score is 0.98, and it is equal to 1 in TXT files) and close to none in 512 KB files (i.e. average F1-score is 0.9965). Note that the experiments reported in the *All* rows correspond to blind experiments (i.e. training the models with all the possible file types, thus a more challenging experiment) considering

1 https://research.google.com/colaboratory/
2 https://scikit-learn.org
3 https://github.com/francasino/Ransomware_analysis

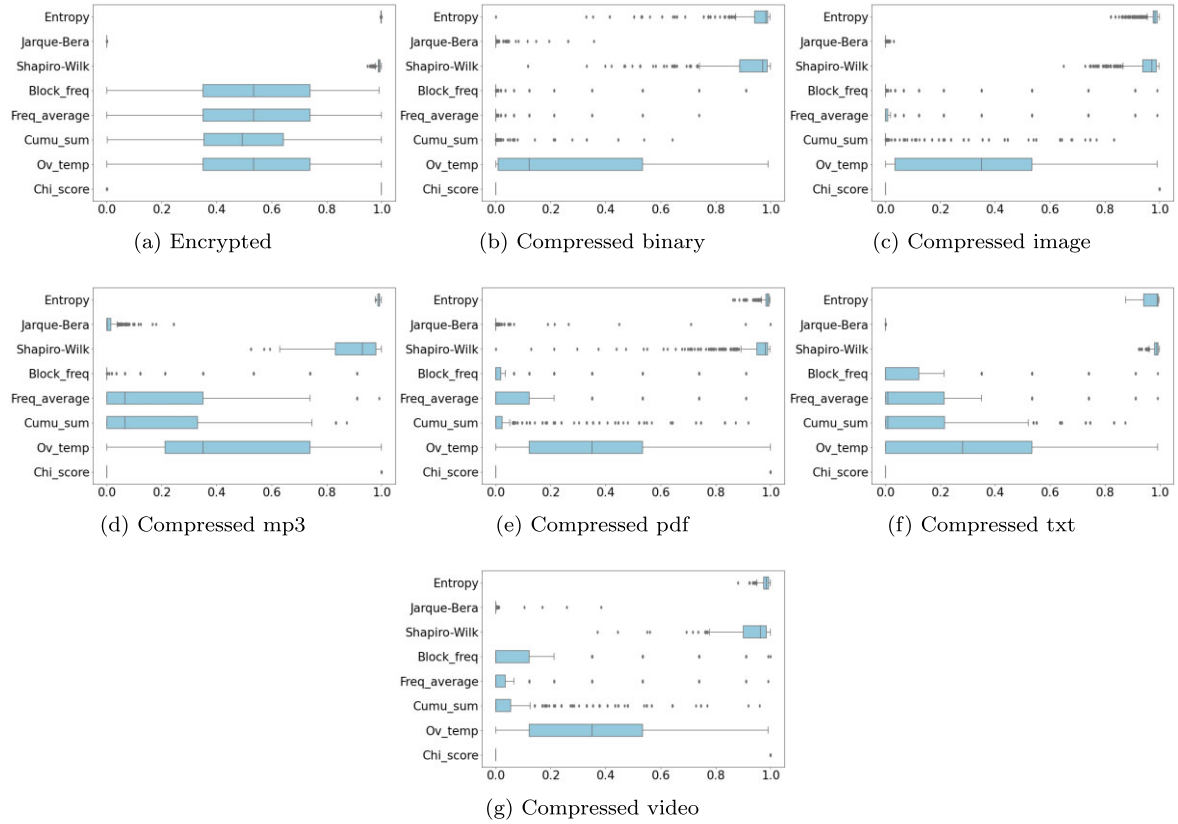


Figure 4. Detail of the features' values for 256 KB files. *x*-axis correspond to the normalized values between 0 and 1 for each feature measured in *y*-axis. The larger the box, the higher the feature's variability.

the whole dataset for training and testing, and hence are not a direct average computed from the rest of values.

Since each model processes data and features differently, we selected the Random Forest due to its high interpretability [53,54] to derive the relevance of each feature in the tree decision. The relevance evaluation of the features in the Random Forest model for each dataset is depicted in Fig. 7. As it can be observed, a specific subset of features always has the most significant relevance regardless of the file size. We noticed that `Entropy`, `Chi_score`, `Monobit`, and `Cumu_sum` are the most relevant features regardless of the model used.

Regarding entropy, we observed that encrypted bit streams always provide the highest possible value of entropy, yet this is not always the case for compressed bit streams. The challenge arises when we find compressed bit streams that achieve indistinguishable entropy levels from those of encrypted bit streams. Thus, entropy is a reliable indicator in most cases but not for a percentage of them, explaining why it has a high relevance but a low variation according to Figs 3–5.

Interestingly, we further noticed that the relevance of some features varied according to data size. More concretely, the relevance of `Entropy` and `Chi_score` slightly grows according to the file size, becoming more relevant than the rest in the 1024 KB dataset.

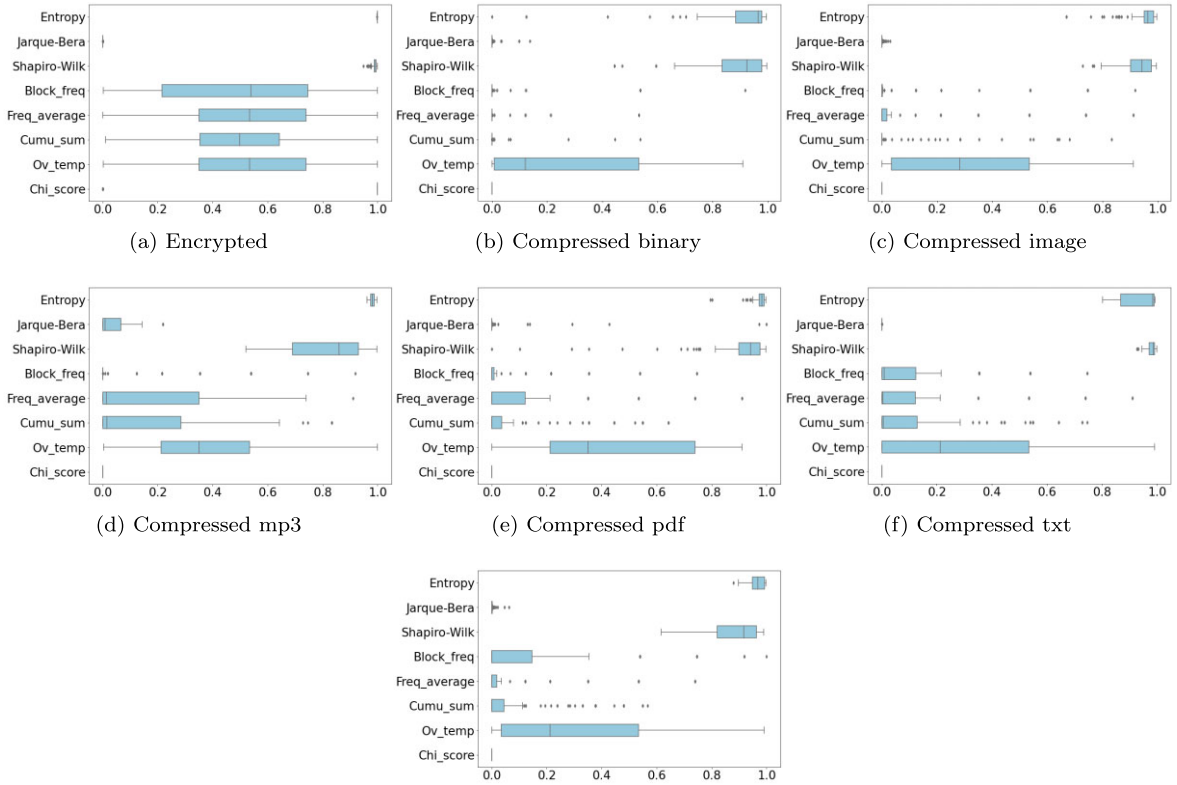
Model performance and optimizations

To further assess how well the model would perform in contexts where computing power and memory are scarce (e.g. IoT devices), we studied the accuracy of the Random Forest model when considering a small subset of features according to their relevance. Therefore, we repeated the classification experiments by creating a new subset of datasets (i.e. considering one, two, three, and all features), only including the selected features in each case. Table 7 shows the accuracy obtained in each case.

As it can be observed, the more features used, the more accuracy, yet by using only one feature, the model already achieves outstanding accuracy, especially for large file sizes, aligned with the feature relevance observed in Fig. 7. Note that the fact that `Entropy` achieves such high accuracy does not diminish the rest of the features since, for instance, `Chi_score` obtains similar values when applied independently. Due to the high correlation between `Entropy` and `Chi_score`, the improvement is reduced when using both features. When `Monobit` is added to the feature set, the improvement is greater, since the model has a richer set of observations to operate with. The main result of this experiment is that selecting a subset of features according to each file size enables adaptable configurations when considering the trade-off between computational cost and accuracy.

The latter can be used to establish different sets of policies according to the system under surveillance and the hardware capabilities, thus enabling devices with low computing resources (e.g. edge IoT devices) to perform faster analysis (e.g. using larger sets of data and not computing the whole feature set when possible), at the cost of having a slightly lower accuracy. Such dynamism enhances the adoption of our solution and provides efficient use of resources regardless of the device under analysis.

Regarding performance, for instance, the XGBoost model required less than 20 s to train and 0.1 s to classify all the values for a

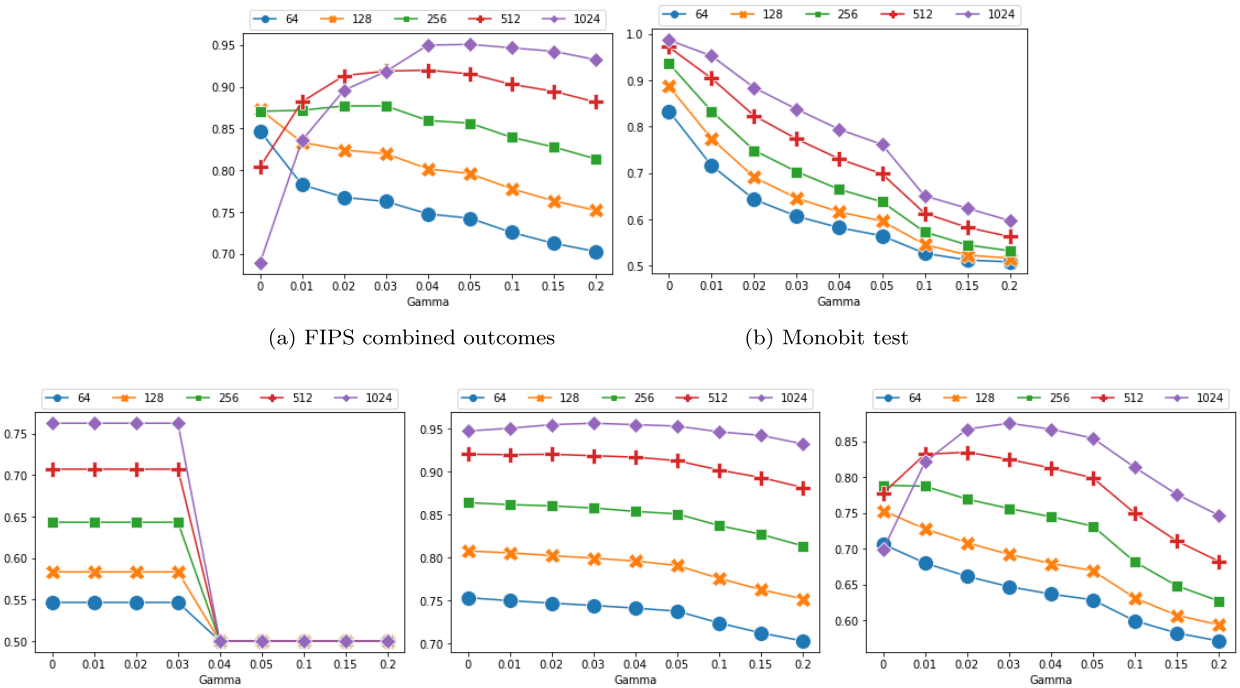


(a) Encrypted (b) Compressed binary (c) Compressed image

(d) Compressed mp3 (e) Compressed pdf (f) Compressed txt

(g) Compressed video

Figure 5. Detail of the features' values for 1024 KB files. *x*-axis correspond to the normalized values between 0 and 1 for each feature measured in *y*-axis. The larger the box, the higher the feature's variability.



(a) FIPS combined outcomes (b) Monobit test

(c) Long run test (d) Poker test (e) Run test

Figure 6. Detail of the FIPS tests' accuracy for each file size and gamma.

Table 4. Configuration parameters of each model.

Model	Best configuration
Random Forest	n_estimators = 100, max_depth = 10
XGBoost	learning_rate = 0.01, max_depth = 4, subsample = 0.6
SVC	kernel = 'rbf'

full 10-fold cross-validation loop in the case of the 64 KB dataset (i.e. considering a model with all features, thus the worst-case scenario). Therefore, assuming that such a dataset contains 19 200 records, the individual prediction time is far below the order of milliseconds. Note that, due to the richness of our dataset, the models need to be trained only once at the beginning. Nevertheless, if new file types were added to the dataset, the model would only require some seconds to be updated. It is relevant to note that the reported times consider that no parallelization is applied, thus enabling a large margin of improvement, also considering the possibility of reducing the number of features of the system according to the file size, as previously discussed.

To further illustrate computational load and efficiency, we computed the feature computation times for the most challenging file size, namely 64 KB. Figure 8 shows the average times for each feature. In particular, all features require times on the order of milliseconds to be computed, with the Cumu_sum test being the one requiring more time. Next, we computed the training time for each file size and the features selected, as shown in Fig. 9. As it can be seen, due to the dataset size as described in Section 4.2, the number of files used in the dataset increases the training times, paired with the number of features used. The latter, combined with the feature computation times, enables the creation of optimization strategies, as previously discussed. Overall, considering that the prediction time of the models is far below the order of milliseconds, we observe that our full pipeline requires from milliseconds to a fraction of a second in the worst case to compute the features and classify a sample. Note, however, that we used the a baseline Google Colab CPU instance without parallelization, and, thus, these measurements could be lower in better performing CPUs.

Experiments with raw and compound files

To further assess the challenging nature of distinguishing between high-entropy files, we performed a test over the raw documents of our dataset. We selected the most challenging size, namely 64 KB, and created file chunks of the original file types. Next, we computed the entropy and the Monobit test, and depicted the outcomes in Figs 10 and 11. As shown in Fig. 10, each file type has a different distribution of features (e.g. only compressed and encrypted files achieve entropy values very close to 8 consistently). Furthermore, only encrypted files pass the Monobit test (i.e. as seen in Fig. 11) and can thus be distinguished with efficacy. The latter showcases that distinguishing raw files from high-entropy ones is a far less challenging experiment than the previous ones, where we focus only on compressed and encrypted files.

As an additional experiment, we wanted to observe the behaviour of compound files containing different data types. We selected a set of MS Office files from a known dataset [55] and created a small database with them, following our database creation procedure, obtaining files from 64 to 1024 KB. The outcomes of the classification can be seen in Table 8. As it can be observed, the behavior is again close to compressed TXT files, as internally these documents are mostly code and text, even though they could contain some images and compressed parts.

Comparison with the state-of-the-art

In this section, we discuss the most relevant state-of-the-art approaches and discuss their benefits and drawbacks compared to our solution.

The first approach dealing with high entropy random data traffic classification was proposed by Hahn et al. [34]. The authors proposed *k*-NN and convolutional Neural networks to classify small files (1 KB) with an accuracy of 0.669. Lin [33] proposed a Support Vector Machine (SVM) classifier to classify compressed and encrypted files. This approach considers several features related to traffic, achieving an accuracy of 0.798 with variable-sized packets bigger than 1 KB. In [35], Kozachok and Spirin [35] used file chunks of 600 KB encrypted and compressed files generated from a set of meaningful text files (i.e. thus a less representative set of files than in our approach, noting that compressed `txt` files are easier to classify), and used a large number of features for classification (~250 features), to obtain an accuracy of 0.97. Moreover, when using fewer features (e.g. 10 features), their accuracy drops to 0.81, a fact that highlights the performance of the features selected in our article (i.e. we need nine features to provide 0.99 accuracy with even smaller file chunks). Considering smaller files (e.g. 50 KB), their accuracy is close to 0.81, while the accuracy obtained by our approach for 64 KB files is above 0.95. De Gaspari et al. [38] presented EnCoD [37], a deep neural network classifier that provided a highly accurate classification of small-size files. In their approach, the largest file size tested is 8 KB with an accuracy of 0.94, with the drawback that the dataset contains only `txt` compressed files, as seen in [35]. Later, in [38], the same authors created a content-type detector based on a neural network model that uses autoencoders. In this case, they experimented with a richer and more balanced dataset, reporting binary classification accuracies between 0.83 and 0.94 for files between 512 Bytes and 8 KB. Table 9 illustrates a descriptive comparison with the current state-of-the-art. As it can be observed, our method provides a rich and reproducible database and achieves higher accuracy than the rest of the methods, considering the specific file sizes tested. Moreover, we performed experiments considering longer file sizes than the rest of the methods, achieving the maximum possible accuracy and, thus, establishing the required lengths to guarantee unequivocal classification. Note that, in the case of production file systems that use frequent compression and encryption, even a low percentage of errors could translate into tenths or hundreds of failed operations in short periods. Thus, this article is the first to provide a complete study in this regard, showing the adaptability of our method.

Casino et al. [36] proposed HEDGE, a threshold-based approach to classify small files up to 64 KB. Their approach considered the Chi-squared statistical test and the FIPS-140-2 test to leverage a classification, thus enabling a memory-efficient simple system that could be deployed regardless of hardware constraints. Nevertheless, to observe the reliability of this method when larger files are processed, we leveraged a comparison in Table 10. As it can be observed, HEDGE has been applied with different parameters (i.e. γ variability as defined by the authors) to provide a fair comparison. The method proposed in this article clearly outperforms HEDGE in all cases due to the difficulty of larger files passing the FIPS-140-2 tests, which increases the number of false negatives (i.e. encrypted files not passing the test) dramatically. This behavior can be observed in the analysis performed in Section 5.2 and in Fig. 6(a), in which the FIPS-140-2

Table 5. Average outcomes for file sizes between 64 and 256 KB, and their corresponding standard deviation σ. The outcomes of each row correspond to independent experiments performed according to each dataset, file type, and classification model. The best $F_1 - score$ has been highlighted in each combination of the dataset and file type experiment.

			Precision		Recall		Accuracy		F_1-score	
Dataset (KB)	File type	Model	Average	σ	Average	σ	Average	σ	Average	σ
		Random Forest	0.9927	0.0680	0.9941	0.0078	0.9934	0.0042	0.9934	0.0042
64	Binary	XGBoost	0.9911	0.0084	0.9964	0.0056	0.9937	0.0047	0.9937	0.0047
		SVC	0.9742	0.0138	0.9920	0.0059	0.9828	0.0073	0.9830	0.0071
		Random Forest	0.8886	0.0222	0.9779	0.0106	0.9273	0.0155	0.9310	0.0140
64	Image	XGBoost	0.8827	0.0235	0.9856	0.0691	0.9269	0.0154	0.9311	0.0136
		SVC	0.8751	0.0214	0.9916	0.0064	0.9247	0.0144	0.9296	0.0126
		Random Forest	0.8864	0.0218	0.9877	0.0093	0.9302	0.0132	0.9341	0.0117
64	MP3	XGBoost	0.8812	0.0229	0.9950	0.0060	0.9301	0.0145	0.9345	0.0128
		SVC	0.8762	0.0233	0.8893	0.0197	0.8815	0.0163	0.8825	0.0158
		Random Forest	0.9096	0.0178	0.9887	0.0079	0.9451	0.0115	0.9474	0.0105
64	PDF	XGBoost	0.9039	0.0189	0.9887	0.0074	0.9416	0.0129	0.9443	0.0119
		SVC	0.8675	0.0186	0.9889	0.0072	0.9187	0.0135	0.9241	0.0120
		Random Forest	0.9900	0.0077	0.9902	0.0076	0.9901	0.0047	0.9901	0.0048
64	TXT	XGBoost	0.9876	0.0090	0.9925	0.0072	0.9900	0.0050	0.9900	0.0049
		SVC	0.8293	0.0209	0.9589	0.0188	0.8804	0.0157	0.8892	0.0138
		Random Forest	0.8744	0.0162	0.9795	0.0095	0.9192	0.0103	0.9239	0.0092
64	Video	XGBoost	0.8657	0.0180	0.9816	0.0118	0.9144	0.0124	0.9199	0.0112
		SVC	0.8450	0.0204	0.9893	0.0080	0.9036	0.0151	0.9113	0.0130
		Random Forest	0.9190	0.0097	0.9893	0.0038	0.9510	0.0055	0.9528	0.0051
64	All	XGBoost	0.9144	0.0100	0.9909	0.0028	0.9490	0.0058	0.9511	0.0053
		SVC	0.8577	0.0090	0.9819	0.0039	0.9094	0.0067	0.9156	0.0059
		Random Forest	0.9967	0.0064	0.9954	0.0083	0.9960	0.0050	0.9960	0.0050
128	Binary	XGBoost	0.9950	0.0076	0.9966	0.0065	0.9958	0.0050	0.9958	0.0050
		SVC	0.9746	0.0191	0.9950	0.0070	0.9843	0.0104	0.9846	0.0102
		Random Forest	0.9117	0.0276	0.9795	0.0137	0.9418	0.0169	0.9441	0.0154
128	Image	XGBoost	0.9097	0.0287	0.9850	0.0124	0.9431	0.0189	0.9456	0.0172
		SVC	0.8933	0.0249	0.9912	0.0087	0.9360	0.0170	0.9395	0.0153
		Random Forest	0.9136	0.0268	0.9873	0.0146	0.9425	0.0170	0.9446	0.0158
128	MP3	XGBoost	0.9117	0.0278	0.9841	0.0135	0.9439	0.0172	0.9463	0.0159
		SVC	0.9071	0.0316	0.9133	0.0270	0.9093	0.0214	0.9098	0.0209
		Random Forest	0.9260	0.0284	0.9791	0.0174	0.9500	0.0176	0.9515	0.0166
128	PDF	XGBoost	0.9222	0.0286	0.9800	0.0184	0.9481	0.0173	0.9498	0.0162
		SVC	0.8893	0.0327	0.9900	0.0115	0.9327	0.0224	0.9366	0.0201
		Random Forest	0.9967	0.0077	0.9962	0.0066	0.9964	0.0051	0.9965	0.0050
128	TXT	XGBoost	0.9955	0.0087	0.9962	0.0087	0.9958	0.0055	0.9958	0.0055
		SVC	0.8558	0.0265	0.9766	0.0181	0.9054	0.0163	0.9118	0.0142
		Random Forest	0.9180	0.0252	0.9687	0.0217	0.9408	0.0194	0.9425	0.0187
128	Video	XGBoost	0.9144	0.0265	0.9812	0.0182	0.9443	0.0199	0.9464	0.0189
		SVC	0.8759	0.0326	0.9929	0.0112	0.9254	0.0222	0.9304	0.0196
		Random Forest	0.9410	0.0099	0.9911	0.0045	0.9644	0.0054	0.9653	0.0051
128	All	XGBoost	0.9355	0.0102	0.9922	0.0039	0.9618	0.0053	0.9629	0.0049
		SVC	0.8799	0.0116	0.9871	0.0051	0.9261	0.0067	0.9304	0.0058
		Random Forest	0.9975	0.0074	0.9975	0.0076	0.9975	0.0050	0.9975	0.0050
256	Binary	XGBoost	0.9975	0.0075	0.9975	0.0076	0.9975	0.0051	0.9975	0.0051
		SVC	0.9951	0.0116	0.9900	0.0155	0.9925	0.0084	0.9924	0.0084
		Random Forest	0.9556	0.0270	0.9741	0.0266	0.9641	0.0217	0.9645	0.0214
256	Image	XGBoost	0.9531	0.0275	0.9816	0.0206	0.9662	0.0177	0.9668	0.0170
		SVC	0.9394	0.0324	0.9900	0.0140	0.9625	0.0196	0.9637	0.0184
		Random Forest	0.9784	0.0165	0.9716	0.0224	0.9750	0.0135	0.9748	0.0137
256	MP3	XGBoost	0.9801	0.0162	0.9800	0.0178	0.9800	0.0137	0.9799	0.0138
		SVC	0.8994	0.0466	0.9658	0.0231	0.9275	0.0292	0.9307	0.0266
		Random Forest	0.9484	0.0299	0.9716	0.0252	0.9587	0.0161	0.9593	0.0155
256	PDF	XGBoost	0.9511	0.0242	0.9750	0.0217	0.9620	0.0152	0.9626	0.0149
		SVC	0.9033	0.0372	0.9900	0.0155	0.9412	0.0264	0.9443	0.0239
		Random Forest	1	0	1	0	1	0	1	0
256	TXT	XGBoost	1	0	1	0	1	0	1	0
		SVC	0.8932	0.0348	0.9866	0.0204	0.9337	0.0258	0.9373	0.0240
		Random Forest	0.9689	0.0242	0.9775	0.0239	0.9729	0.0207	0.9730	0.0206
256	Video	XGBoost	0.9717	0.0255	0.9833	0.0211	0.9770	0.0185	0.9772	0.0184
		SVC	0.9206	0.0370	0.9950	0.0101	0.9537	0.0225	0.9559	0.0206
		Random Forest	0.9716	0.0105	0.9905	0.0063	0.9807	0.0048	0.9809	0.0047
256	All	XGBoost	0.9700	0.0109	0.9912	0.0059	0.9802	0.0062	0.9805	0.0060
		SVC	0.9209	0.0144	0.9901	0.0058	0.9524	0.0091	0.9542	0.0084

Table 6. Average outcomes for 512 and 1024 KB file sizes, and their corresponding standard deviation σ. The outcomes of each row correspond to independent experiments performed according to each dataset, file type, and classification model. The best $F_1 - score$ has been highlighted in each combination of the dataset and file type experiment.

			Precision		Recall		Accuracy		F_1-score	
Dataset (KB)	File type	Model	Average	σ	Average	σ	Average	σ	Average	σ
512	Binary	Random Forest	1	0	1	0	1	0	1	0
		XGBoost	1	0	1	0	1	0	1	0
		SVC	1	0	0.9800	0.0385	0.9900	0.0192	0.9895	0.0204
512	Image	Random Forest	0.9857	0.0221	1	0	0.9924	0.0116	0.9926	0.0113
		XGBoost	0.9888	0.0204	1	0	0.9941	0.0107	0.9943	0.0104
		SVC	0.9951	0.0147	0.9800	0.0337	0.9875	0.0182	0.9871	0.0188
512	MP3	Random Forest	0.9856	0.0252	0.9883	0.0215	0.9866	0.0170	0.9867	0.0168
		XGBoost	0.9856	0.0251	0.9933	0.0217	0.9891	0.0181	0.9892	0.0182
		SVC	0.9349	0.0380	0.9800	0.0249	0.9550	0.0221	0.9563	0.0211
512	PDF	Random Forest	0.9803	0.0380	0.9850	0.0325	0.9816	0.0236	0.9818	0.0232
		XGBoost	0.9803	0.0381	0.9900	0.0203	0.9841	0.0212	0.9845	0.0201
		SVC	0.9675	0.0433	0.9750	0.0286	0.9700	0.0281	0.9705	0.0265
512	TXT	Random Forest	1	0	1	0	1	0	1	0
		XGBoost	1	0	1	0	1	0	1	0
		SVC	0.9760	0.0271	0.9900	0.0203	0.9824	0.0187	0.9827	0.0185
512	Video	Random Forest	0.9937	0.0200	0.9983	0.0091	0.9958	0.0115	0.9959	0.0111
		XGBoost	0.9906	0.0225	1	0	0.9950	0.0121	0.9951	0.0116
		SVC	0.9768	0.0356	0.9900	0.0203	0.9824	0.0198	0.9828	0.0192
512	All	Random Forest	0.9950	0.0063	0.9980	0.0035	0.9965	0.0036	0.9965	0.0036
		XGBoost	0.9934	0.0072	0.9986	0.0031	0.9959	0.0035	0.9959	0.0035
		SVC	0.9617	0.0167	0.9841	0.0124	0.9723	0.0101	0.9727	0.0098
1024	Binary	Random Forest	1	0	1	0	1	0	1	0
		XGBoost	1	0	1	0	1	0	1	0
		SVC	1	0	0.9800	0.0484	0.9900	0.0242	0.9892	0.0262
1024	Image	Random Forest	1	0	1	0	1	0	1	0
		XGBoost	1	0	1	0	1	0	1	0
		SVC	1	0	0.9700	0.0466	0.9850	0.0233	0.9842	0.0245
1024	MP3	Random Forest	0.9909	0.0277	1	0	0.9950	0.0152	0.9952	0.0145
		XGBoost	0.9909	0.0277	1	0	0.9950	0.0152	0.9952	0.0145
		SVC	0.9812	0.0382	0.9700	0.0466	0.9750	0.0314	0.9747	0.0317
1024	PDF	Random Forest	1	0	1	0	1	0	1	0
		XGBoost	1	0	1	0	1	0	1	0
		SVC	1	0	0.9800	0.0406	0.9900	0.0203	0.9894	0.0214
1024	TXT	Random Forest	1	0	1	0	1	0	1	0
		XGBoost	1	0	1	0	1	0	1	0
		SVC	1	0	0.9900	0.0305	0.9950	0.0152	0.9947	0.0160
1024	Video	Random Forest	1	0	1	0	1	0	1	0
		XGBoost	1	0	1	0	1	0	1	0
		SVC	0.9909	0.0277	0.9800	0.0406	0.9850	0.0233	0.9847	0.0237
1024	All	Random Forest	1	0	1	0	1	0	1	0
		XGBoost	1	0	1	0	1	0	1	0
		SVC	0.9950	0.0098	0.9783	0.0181	0.9866	0.0094	0.9864	0.0096

accuracy (i.e. for $\gamma = 0$, thus the classical test) for large sizes is lower for smaller sizes.

Proof of concept implementation

To validate the efficacy of our approach in a real-world scenario, we implemented our proposed solution in Python. To monitor filesystem changes, we used *watchdog*[4] and native Python libraries without further optimizations, e.g. parallelization, pypy, and so on. Moreover, we set our timer to 10 ms and the sampling to 64 KB. Then, we created a virtual machine with Windows 10, 12 GB of RAM, and allocated 12 Cores from the processor, a 13th Gen Intel Core i9-13900K with 32 cores. We made all the necessary updates to the operating system, populated the host with several files that would be found in a typical user host, e.g. PDF and MS Office files, and added some history to the browser to show that this is an actual system. Nevertheless, we did not try to hide that the system is a virtual machine or introduce additional measures to hide our monitoring activity. The latter led some samples, e.g. Akira, not to encrypt the files as they understood that they were monitored and seized their actions, a common malware behavior to evade their analysis [56,57]. Yet, despite the possible computational improvements that Virtual Box Guest Additions could introduce, we opted not to use them to avoid making the virtual machine fingerprint bigger. Given that the scope of this testing is not to counter the antianalysis mechanisms of ransomware, we omit these samples from our report. Furthermore, to ensure that the ransomware will be executed and we will not have interference with

4 https://github.com/gorakhargosh/watchdog

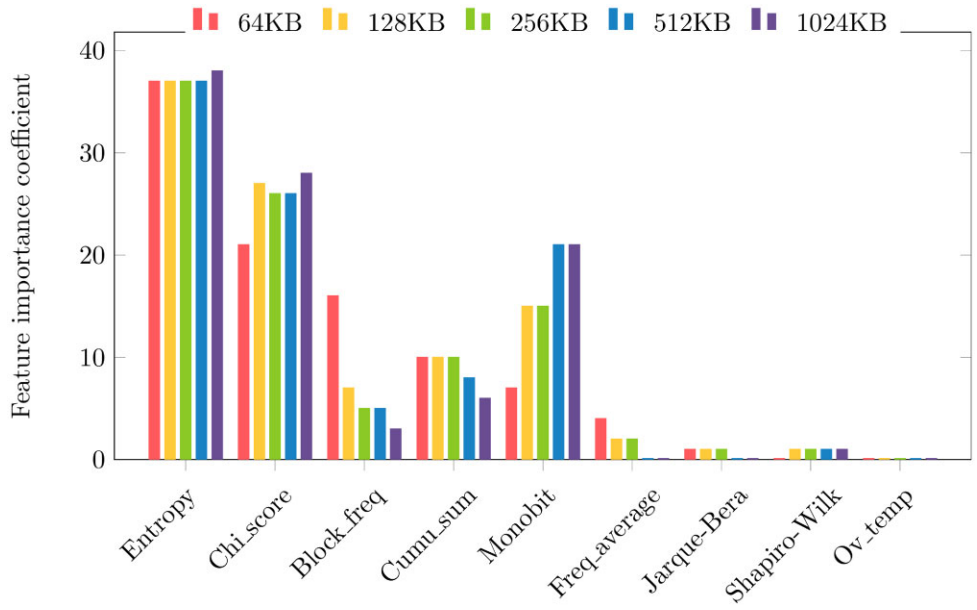


Figure 7. Details of the feature relevance for each dataset in the Random Forest classifier setup.

Table 7. Outcomes when using a different number of features in the Random Forest model.

Dataset	Feature set 1 (Entropy)	Feature set 2 (Entropy and Chi_score)	Feature set 3 (Entropy and Chi_score and Monobit)	All features
64	0.9192	0.9202	0.9353	0.9528
128	0.9425	0.9441	0.9599	0.9653
256	0.9662	0.9680	0.9753	0.9809
512	0.9925	0.9930	0.9955	0.9965
1024	1	1	1	1

other security mechanisms, we have disabled Microsoft Defender and the firewall. Nevertheless, to avoid further propagation, the virtual machine was blocked from the network and the Internet by detaching the network card from the virtual machine. Finally, we collected recent ransomware samples from Malware Bazaar of Abuse.ch[5] to validate our detection capabilities in a real scenario. In Table 11, we report the ransomware family, the average time it took to determine that a file is encrypted, and the standard deviation in seconds.

Evidently, using a minifilter, a more efficient programming language, and parallelization, the reported times would be drastically lower. Yet, the accurate identification of ransomware encryption on the scale of a second from our approach in our proof-of-concept implementation showcases the efficacy and validity of our approach in a real-world scenario. The results illustrate huge discrepancies between the ransomware families. For instance, there are detections that need just a couple of milliseconds, while others take around a second. We attribute these discrepancies to the different ways that each ransomware family encrypts the files. The use of parallelization, different file prioritization for encryption, as well as the cryptographic primitives used from each ransomware family are obvious factors that variate when each ransomware makes the filesystem changes that we monitor. In this regard, we expect that when files are processed sequentially, we will have a shorter detection time, but when the files are processed in parallel, a longer delay is expected.

5 https://bazaar.abuse.ch/

Discussion

As previously seen in the literature [36,37], there is a strong correlation between the randomness of the file type and the randomness of the corresponding generated compressed file. As seen in Tables 5 and 6, the best outcomes were obtained by the binary and `txt` files, which were the ones that exhibited more identifiable patterns according to our selected features (and thus, less randomness), as depicted in Figs 3–5. Moreover, as discussed in Section 5.1, such identifiable patterns expand from small file sizes to larger ones, yet with specific changes that ease the distinction between compressed and encrypted bit streams when their length grows. The above can be justified by the fact that these are the most "structured" files in our dataset. That is because image files, PDFs, MP3s, and videos already contain in some forms compressed information. For instance, PDF stores PDF commands and text and vector objects in ZIP format, JPEG uses Huffman coding, and PNG format uses Huffman and Lempel-Ziv 77 compression algorithms. As a result, possible structures are local, e.g. PDF trailer and cross-reference tables (xref), so most of these files are more random-looking. Recalling the necessity to minimize the cost of misclassification, our system is able to detect and flag all encrypted files without error due to their stability, as seen in Fig. 5. Thus, the challenge appears in zip files that resemble encrypted ones. As stopping a compression process is less critical than letting a malicious encryption one pass, we assume that these misclassifications are assumable errors, which minimize the compromise of the system as the opposite would do.

By recalling the design objectives stated in Section 4.1, we can claim that our proposal fulfilled them remarkably. First, as discussed in Section 5.6, the accuracy of our system outperforms the current state of the art, and it does so efficiently, enabling real-time classification. Next, our system can be dynamically adapted to different file sizes and features to reduce the number of computations while obtaining outstanding accuracy, as described in Section 5.4. In this re-

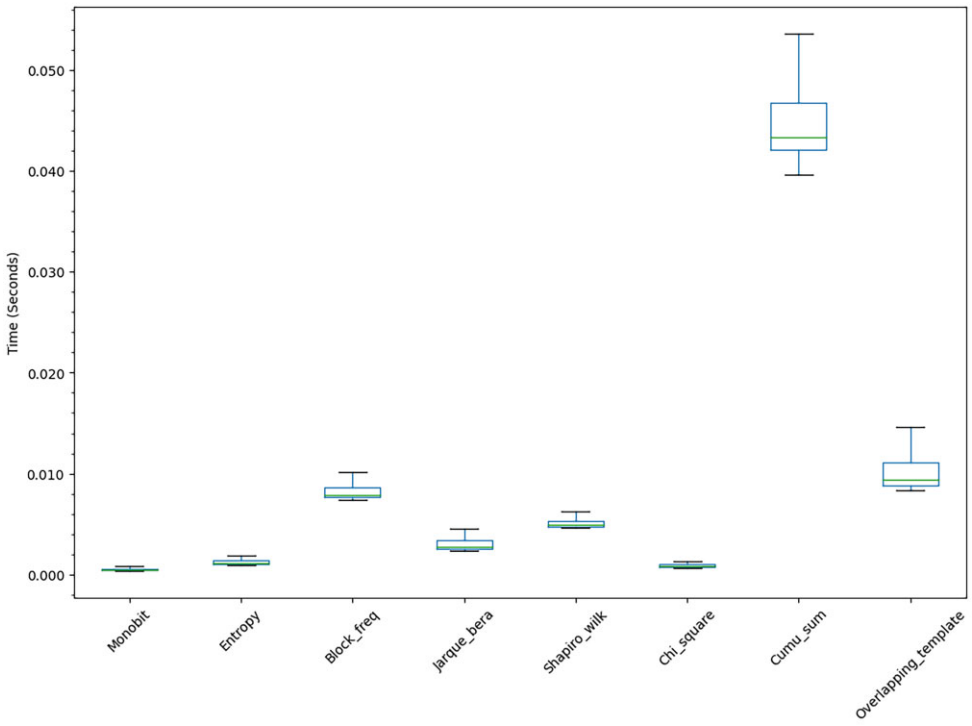


Figure 8. Average feature computation times for 64 KB file size.

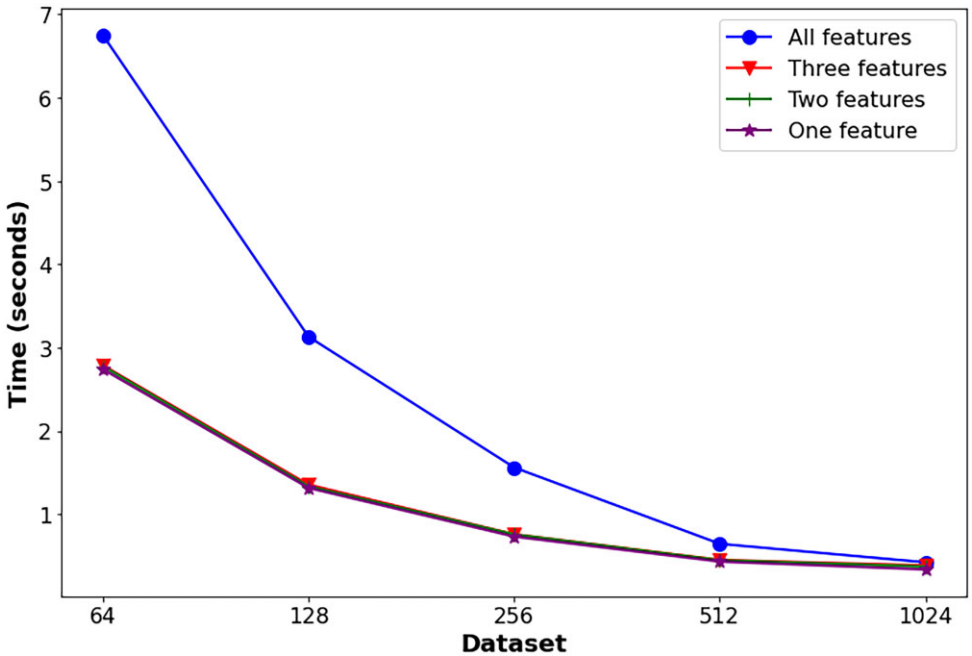


Figure 9. Training time for each file size and number of features in the Random Forest model.

gard, a versatile approach to enable efficient intrusion detection systems in the context of resource-constrained devices is implementing Multi-Agent Systems (MAS). In MAS, multiple autonomous agents operate within a network, each possessing specialized capabilities [58]. They can communicate and collaborate to achieve a common goal, such as detecting and mitigating ransomware attacks by sharing information to ease the cybercrime fight [59]. In MAS, the flexibility to adaptively select features based on the nature of data and constraints of devices, as demonstrated in our model, becomes a critical aspect. Thus, each agent could be equipped with multiple versions of the model discussed in this article, having a flexible model that maintains performance while being adaptable in its feature set [60,61]. Furthermore, using MAS allows for decentralized processing, reducing the load on individual IoT devices [62].

Finally, we publicly shared our dataset and experiments, and described the features and parameters we used for the classification

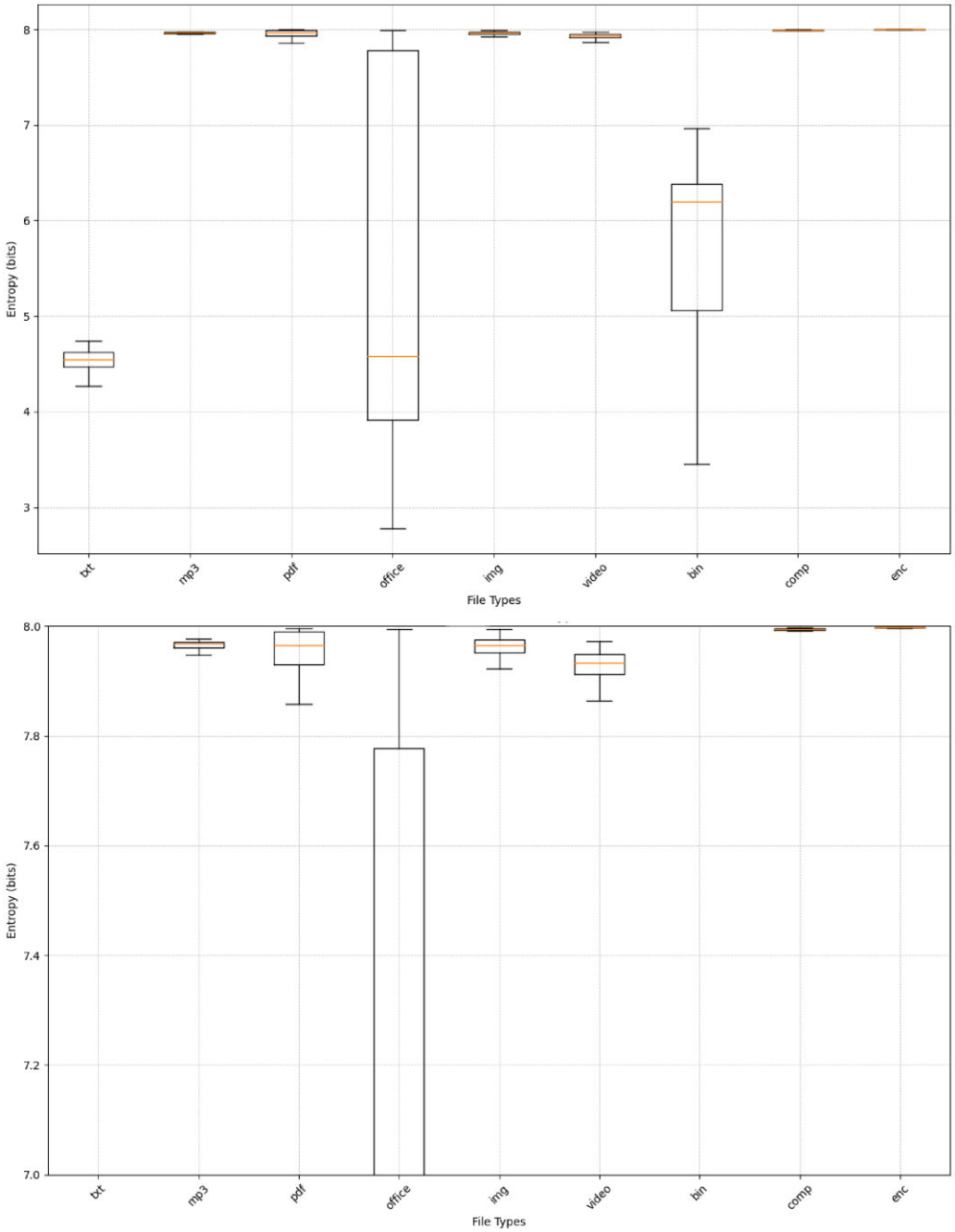


Figure 10. Detail of the entropy values of nonencrypted, compressed, and encrypted files. The second figure represents a zoomed version of the outcomes for clarity.

methods in Section 4. While this eases the reproducibility of the experiments and further comparisons, we also allow fellow researchers to use a significantly richer baseline dataset, fostering the progress of the state of the art. Note that dependence on small and unrepresentative datasets carries inherent risks [46]. For instance, this shortcoming may foster biases, distorting the understanding and interpretation of the actual data, and, as a result, lead to wrong analysis and misleading conclusions.

A further distinctive characteristic of our design is that, contrary to the current state of the art, it can be used both in traffic analysis and in the context of file write operations monitoring. The latter, paired with the adaptable feature selection, results in a versatile solution toward malware detection, with a particular focus on ransomware.

Due to its minimal computational overhead, the proposed approach could be integrated into existing Endpoint Detection and Response (EDR) solutions and increase their detection capabilities. Notably, while many of them use honey files [63] to detect ransomware infection or prevent the execution of binaries with high entropy, threat actors have found ways to bypass these measures. On the contrary, our solution accurately detects ransomware execution by focusing on the very nature of the outcomes and the timely detection of encrypted files. In fact, identifying the running process from the

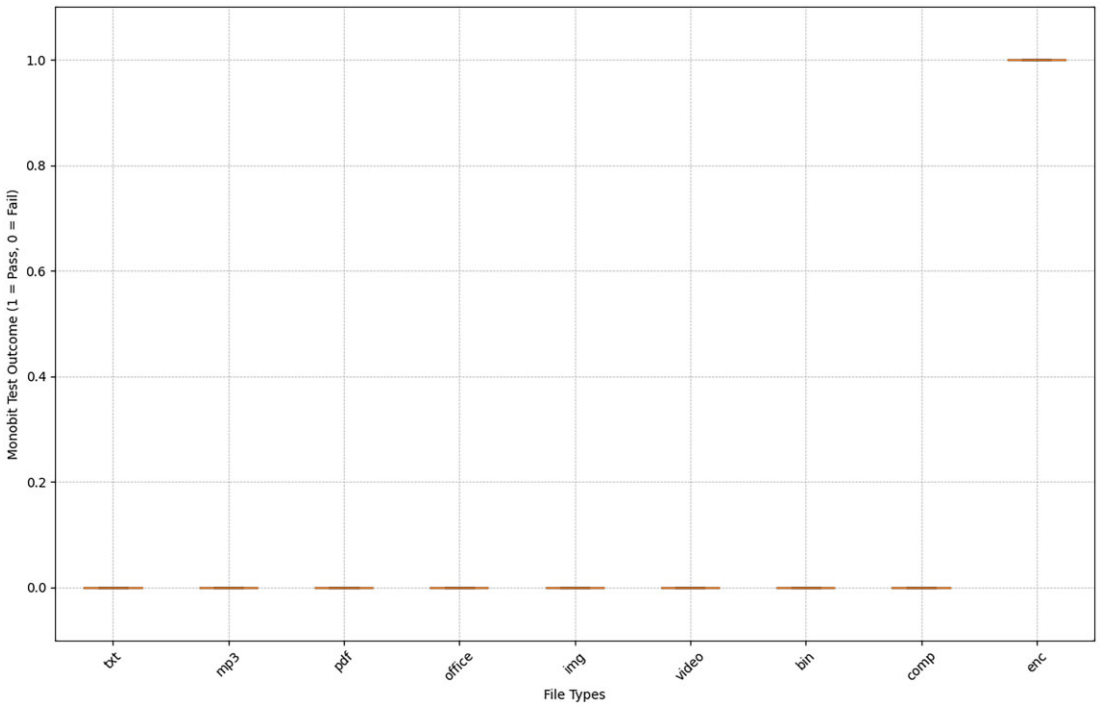


Figure 11. Outcomes of the Monobit test when applied to raw files, compressed, and encrypted ones.

Table 8. Average outcomes for the classification of compressed office files versus encrypted office files dataset and their corresponding standard deviation σ when using the Random Forest classifier.

	Precision		Recall		Accuracy		F1-score	
File size	Average	σ	Average	σ	Average	σ	Average	σ
64	0.9818	0.0370	0.9900	0.0305	0.9850	0.0233	0.9852	0.0230
128	0.9909	0.0277	0.9900	0.0305	0.9900	0.0203	0.9900	0.0204
256	0.9909	0.0277	1	0	0.9950	0.0153	0.9952	0.0145
512	0.9909	0.0277	1	0	0.9950	0.0153	0.9952	0.0145
1024	0.9944	0.0304	1	0	0.9967	0.0183	0.9970	0.0166

Table 9. Descriptive comparison with the state-of-the-art.

References	Model	Comments
[33]	SVM classifier	Small file sizes, with accuracies below 0.80
[34]	*k*-NN and convolutional neural networks	Classification accuracy of 0.67 with small file sizes (1 KB).
[36]	Threshold-based	Classification accuracy between 0.69 and 0.95 for files between 1 and 64 KB. Accuracy decreases with file size above 64 KB.
[37]	Deep neural network	Only small file sizes tested (from 512Bytes to 8 KB files) over a less representative dataset than ours.
[35]	Random Forest	Accuracies ranging between 0.81 and 0.97 in the case of 600 KB file size, with a less representative dataset and a much higher number of features.
[38]	Deep neural network with autoencoders	Only small file sizes tested (from 512Bytes to 8 KB files) with accuracies between 0.83 and 0.94.
Current work	Random Forest	Accuracies ranging between 0.95 and 1 for files between 64 KB and 1024 KB.

file system changes can solve another issue that some EDR blocking mechanisms face. EDR systems utilize a variety of techniques for detecting malicious activities. These techniques generally include signature-based detection, behavioral analysis, heuristic analysis, and machine learning. According to the MITRE ATT&CK framework, common techniques observed in malware are often tied to specific behavioral patterns, such as process injection, process hollowing, privilege escalation, and lateral movement. EDR solutions rely on these

Table 10. Comparison with HEDGE [36].

Dataset	Model	F1	Dataset	Model	F1
64	HEDGE $\gamma = 0$	0.7488	128	HEDGE $\gamma = 0$	0.7211
	HEDGE $\gamma = 1$	0.8762		HEDGE $\gamma = 1$	0.8598
	HEDGE $\gamma = 2$	0.9424		HEDGE $\gamma = 2$	0.9078
	Our approach (RF)	0.9528		Our approach (RF)	0.9653
256	HEDGE $\gamma = 0$	0.7025	512	HEDGE $\gamma = 0$	0.6654
	HEDGE $\gamma = 1$	0.8339		HEDGE $\gamma = 1$	0.7645
	HEDGE $\gamma = 2$	0.8818		HEDGE $\gamma = 2$	0.7995
	Our approach (RF)	0.9809		Our approach (RF)	0.9965
1024	HEDGE $\gamma = 0$	0.5983			
	HEDGE $\gamma = 1$	0.6591			
	HEDGE $\gamma = 2$	0.6833			
	Our approach (RF)	1			

Table 11. Statistics from our proof of concept implementation. Time is reported in seconds.

Ransomware	Sample	Average	SD
Chaos	`4dd53a1b9a5bc8e1c327abfa7774e287`	1.165	2.613
Conti	`71d43bb68ae566de0d8183d223b56e5d`	0.081	0.141
Dharma	`32e3001eb783b182de6b45e5f729d3ba`	0.822	0.588
Fog	`d72c3508cbb968c478e0bd91e0f11424`	0.005	0.005
InterLock	`f7f679420671b7e18677831d4d276277`	0.735	0.461
Mammon	`ccaa87a7a44fa59ae536138e2313bc3e`	0.021	0.041
Phobos	`6096dec7644520ba1a4fdc04183bb62f`	0.009	0.032
Termite	`6b06aae5ec596cdbc1b9d4c457fd5f81`	0.931	0.327

techniques to detect and respond to malware in real-time. In several instances, EDRs may detect malicious behavior of a process and kill the process; however, the process may spawn another instance, so while the original process is sacrificial and is killed, the actual encryption is performed by the spawned process, which is not monitored any more. In other instances of human-operated ransomware, the EDR may have initially detected and blocked the malicious binary that would encrypt the victim's files, the process is killed, and the binary is sent for automated scanning. However, suppose the proper input (e.g. used arguments) is not sent. In that case, the ransomware performs a graceful exit on the sandbox environment so the EDR does not consider it a threat and may allow its execution the next time. On the contrary, by integrating our approach into EDR rules, file system monitoring could be configured to trigger alerts whenever encryption-like patterns are detected (e.g. sudden high-entropy changes to files). The latter would allow the EDR to react immediately by issuing alerts, terminating the rogue processes responsible for encryption, and quarantining the malicious binaries before significant damage occurs. Indeed, as illustrated with our proof of concept experiments, our approach can timely identify the ransomware encryption.

Conclusions

In the context of cyber security, the accurate and efficient identification of encrypted bit streams is an open challenge that affects several areas, such as file system security and network traffic analysis, to name a few. Thus, leveraging automated systems such as the one presented in this article is crucial, especially considering the current threat landscape, in which ransomware seems to be a long-term menace.

This work performs a thorough evaluation of available randomness tests to determine the ones that are efficient and allow their output to be used as features to accurately classify random bit streams into either encrypted or compressed ones. We created and used a statistically sound dataset to test our experiments, achieving an accuracy between 0.9528 (64 KB bit streams) and 1. We also observed and analysed the correlation between the different file types and the randomness of the corresponding compressed files. We concluded that (compressed) binary and text files can be easily identified by our method with almost no errors, even in 64 KB bit streams. This is validated with our proof of concept using recent ransomware samples. Although our method can detect the nature of high-entropy bit streams, some applications, including compressors, may use encryption at some stages. The latter opens the door for optimization mechanisms in the form of OS-level policies, such as setting up whitelisted applications or directories and performing an overall control of the file system's entropy to detect an attack as early as possible. These interesting paths will be explored in future work.

The efficiency of our method was highlighted, exhibiting better performance than competing state-of-the-art works while enabling feature-based adaptability to achieve real-time classification in different contexts. The latter enables it to be used in existing endpoint security solutions through, e.g. minifilters and the inotify API[6] in Windows and linux-based hosts, respectively, as illustrated with our proof of concept implementation testing.

In future works, we plan to (i) refine our method to increase its accuracy further; especially for short length bit streams, (ii) study further learning strategies that can be combined with our method to efficiently detect ransomware-related behaviours in real time, and (iii) explore other feature generation methods. Another line of research will explore ad-hoc models to maximize the detection accuracy for file types and sizes used in particular scenarios. We consider incorporating the model optimizations discussed in this article into a MAS to enable an efficient and potent solution for ransomware detection in distributed, resource-constrained environments. Finally, we will explore the possibility of detecting ransomware in the early stages of the binary execution, e.g. when the binary is unpacked, and the initial system calls are made before encryption.

Acknowledgements

The content of this article does not reflect the official opinion of the European Union. Responsibility for the information and views expressed therein lies entirely with the authors.

6 https://man7.org/linux/man-pages/man7/inotify.7.html

Author contributions

Fran Casino (Conceptualization, Investigation, Methodology, Project administration, Software, Validation, Writing – original draft, Writing – review & editing), Darren Hurley-Smith (Software, Validation, Writing – original draft), Julio Hernandez-Castro (Conceptualization, Formal analysis, Methodology, Writing – review & editing), and Constantinos Patsakis (Conceptualization, Formal analysis, Investigation, Methodology, Validation, Writing – original draft, Writing – review & editing)

Conflict of interest: The authors reported no potential conflict of interest.

Funding

This work was supported by the European Commission under the Horizon Europe Programme, as part of the projects SAFEHORIZON (grant agreement number 101168562) and LAZARUS (grant agreement number 101070303). This work was also supported by the European Union's Internal Security Fund as part of the ALUNA project (grant agreement number 101084929). This work was also supported by the COST Action GoodBrother, Network on Privacy-Aware Audio- and Video-Based Applications for Active and Assisted Living, (CA 19121). This work was partially supported by Ministerio de Ciencia, Innovación y Universidades, Gobierno de España (Agencia Estatal de Investigación, Fondo Europeo de Desarrollo Regional -FEDER-, European Union) under the research grant PID2021-127409OB-C33 CONDOR. Fran Casino was supported by the Spanish Ministry of Science and Innovation under the "Ramón y Cajal" programme (RYC2023-044857-I), and by AGAUR with the project ASCLEPIUS (2021SGR-00111).

References

1. Sophos. Sophos the state of ransomware 2023. https://assets.sophos.com/X24WTUEQ/at/c949g7693gsnjh9rb9gr8/sophos-state-of-ransomware-2023-wp.pdf (4 January 2024, date last accessed).
2. Braue D. Global ransomware damage costs predicted to exceed $265 billion by 2031. Cybersecurity Ventures, 2022. https://cybersecurityventures.com/global-ransomware-damage-costs-predicted-to-reach-250-billion-usd-by-2031 (11 February 2023, date last accessed).
3. Cohen A, Nissim N. Trusted detection of ransomware in a private cloud using machine learning methods leveraging meta-features from volatile memory. *Expert Syst Appl* 2018;**102**:158–78.
4. Patsakis C, Arroyo D, Casino F. The Malware as a service ecosystem. In: *Malware: Handbook of Prevention and Detection*. Berlin: Springer Nature, 2024, 371–94.
5. Shannon CE. Communication theory of secrecy systems. *Bell Syst Tech J* 1949;**28**:656–715.
6. L'Écuyer P, Compagner A, Cordeau JF. Entropy tests for random number generators. In: GERAD report G-96-41. Montréal: GERAD, 1996.
7. D'Agostino RB. *Goodness-of-fit-techniques*. London: Routledge, 2017.
8. Ghosh BK, Sen PK. *Handbook of Sequential Analysis*. Boca Raton, FL: CRC Press, 1991.
9. Shapiro SS, Wilk MB. An analysis of variance test for normality (complete samples). *Biometrika* 1965;**52**:591–611.
10. Lopes RHC. Kolmogorov–Smirnov test. In: *International Encyclopedia of Statistical Science*. Berlin: Springer, 2011, 718–20.
11. Razali NM, Wah YB. Power comparisons of Shapiro–Wilk, Kolmogorov–Smirnov, Lilliefors and Anderson–Darling tests. *J Stat Model Anal* 2011;**2**:21–33.
13. Rukhin A, Soto J, Nechvatal J. A statistical test suite for random and pseudorandom number generators for cryptographic applications. NIST DTIC Document. NIST DTIC Document NIST SP800-22. Gaithersburg, MD: National Institute of Standards and Technology, 2010.
14. Casino F, Dasaklis TK, Spathoulas GP. *et al.* Research trends, challenges, and emerging topics in digital forensics: a review of reviews. *IEEE Access* 2022;**10**:25464–93.
15. Yang W, Kong D, Xie T. *et al.* Malware detection in adversarial settings: exploiting feature evolutions and confusions in android apps. In: *Proceedings of the 33rd Annual Computer Security Applications Conference*. New York, NY: ACM Digital Library, 2017, 288–302.
16. Kara I. Fileless malware threats: recent advances, analysis approach through memory forensics and research challenges. *Expert Syst Appl* 2022;**214**:119133.
17. Kara I, Aydos M. The rise of ransomware: forensic analysis for windows based ransomware attacks. *Expert Syst Appl* 2022;**190**:116198.
18. VirusTotal. Ransomware activity report. 2021. https://storage.googleapis.com/vtpublic/vt-ransomware-report-2021.pdf. (4 January 2024, date last accessed).
19. Kara I, Aydos M. The rise of ransomware: forensic analysis for windows based ransomware attacks. *Expert Syst Appl* 2022;**190**:116198.
20. Jung S, Won Y. Ransomware detection method based on context-aware entropy analysis. *Soft Comput* 2018;**22**:6731–40.
21. Cuzzocrea A, Martinelli F, Mercaldo F. A novel structural-entropy-based classification technique for supporting android ransomware detection and analysis. In: *Proceedings of the 2018 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE)*. Piscataway, NJ: IEEE, 2018, 1–7.
22. Lee K, Lee SY, Yim K. Machine learning based file entropy analysis for ransomware detection in backup systems. *IEEE Access* 2019;**7**:110205–15.
23. McIntosh T, Jang-Jaccard J, Watters P. *et al.* The inadequacy of entropy-based ransomware detection. In: *Proceedings of the International Conference on Neural Information Processing*. Berlin: Springer, 2019, 181–9.
24. Pont J, Arief B, Hernandez-Castro J. Why current statistical approaches to ransomware detection fail. In: *Proceedings of the International Conference on Information Security*. Berlin: Springer, 2020, 199–216.
25. Hurley-Smith D, Patsakis C, Hernandez-Castro J. On the unbearable lightness of FIPS 140-2 randomness tests. In: *Proceedings of the IEEE Transactions on Information Forensics and Security*. Piscataway, NJ: IEEE, 2020, 1–1.
26. Bhudia A, O'Keeffe D, Sgandurra D. *et al.* RansomClave: ransomware key management using SGX. In: *Proceedings of the 16th International Conference on Availability, Reliability and Security*. New York, NY: ACM Digital Library, 2021, 1–10.
27. Kolodenker E, Koch W, Stringhini G. *et al.* Paybreak: defense against cryptographic ransomware. In: *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security*. New York, NY: ACM, 2017, 599–611.
28. Roth F, Lambert J, Yom-Tov E. *et al.* Raccine. GitHub, 2021. https://github.com/Neo23x0/Raccine. (4 January 2024, date last accessed).
29. Paninski L. Estimation of entropy and mutual information. *Neur Comput* 2003;**15**:1191–253.
30. Paninski L. Estimating entropy on m bins given fewer than m samples. *IEEE T Inf Theor* 2004;**50**:2200–3.
31. Davies SR, Macfarlane R, Buchanan WJ. Comparison of entropy calculation methods for ransomware encrypted file identification. *Entropy* 2022;**24**:1503.
32. Cunha VC, Zavala AZ, Magoni D. *et al.* A complete review on the application of statistical methods for evaluating internet traffic usage. *IEEE Access* 2022;**10**:128433–55.
33. Wang Y, Zhang Z, Guo L. *et al.* Using entropy to classify traffic more deeply. In: *Proceedings of the 2011 IEEE Sixth International Conference on Networking, Architecture, and Storage*. Piscataway, NJ: IEEE, 2011, 45–52.
34. Hahn D, Apthorpe N, Feamster N. Detecting compressed cleartext traffic from consumer internet of things devices. 2018. arXiv:1805.02722. (4 January 2024, date last accessed).
35. Kozachok AV, Spirin AA. Model of pseudo-random sequences generated by encryption and compression algorithms. *Prog Comput Softw* 2021;**47**:249–60.
36. Casino F, Choo KKR, Patsakis C. Hedge: efficient traffic classification of encrypted and compressed packets. *IEEE T Inf Foren Secur* 2019;**14**:2916–26.
37. De Gaspari F, Hitaj D, Pagnotta G. *et al.* Encod: distinguishing compressed and encrypted file fragments. In: *Proceedings of the International*

Conference on Network and System Security. Berlin: Springer, 2020, 42–62.
38. De Gaspari F, Hitaj D, Pagnotta G. *et al.* Reliable detection of compressed and encrypted data. *Neur Comput Appl* 2022;**34**:20379–93.
39. Microsoft. Filter manager concepts. Redmond, WA, 2021. https://docs.microsoft.com/en-us/windows-hardware/drivers/ifs/filter-manager-concepts. (4 January 2024, date last accessed).
40. White AM, Krishnan S, Monrose F. Clear and present data: opaque traffic and its security implications for the future. In: *Proceedings of the NDSS*. San Diego, CA: Network and Distributed System Security, 2013.
41. Khakpour AR, Liu AX. An information-theoretical approach to high-speed flow nature identification. *IEEE/ACM Trans Netw* 2013;**21**:1076–89.
42. Lin TY, Maire M, Belongie S,. *et al.* Microsoft Coco: common objects in context. In: *Proceedings of the European Conference on Computer Vision*. Berlin: Springer, 2014, 740–55.
43. Criminisi A. Rgb-d dataset 7-scenes. Redmond, WA: Microsoft, 2013. https://www.microsoft.com/en-us/research/project/rgb-d-dataset-7-scenes/. (4 January 2024, date last accessed).
44. Hart M. Project Gutenberg. 1971. https://www.gutenberg.org/. (17 December 2023, date last accessed).
45. Abu-El-Haija S, Kothari N, Lee J. *et al.* Youtube-8m: a large-scale video classification benchmark. *arXiv:1609.08675*. 2016.
46. Casino F, Lykousas N, Homoliak I. *et al.* Intercepting hail hydra: real-time detection of algorithmically generated domains. *J Netw Comput Appl* 2021;**190**:103135.
47. Karatas G, Demir O, Sahingoz OK. Increasing the performance of machine learning-based idss on an imbalanced and up-to-date dataset. *IEEE Access* 2020;**8**:32150–62.
48. Batista GE, Prati RC, Monard MC. A study of the behavior of several methods for balancing machine learning training data. *ACM SIGKDD Explor Newsl* 2004;**6**:20–9.
49. Casino F. Distinguishing between high entropy bit streams. GitHub, 2021.
50. Hurley-Smith D, Hernandez-Castro J. Certifiably biased: an in-depth analysis of a Common Criteria EAL4+ certified TRNG. In: *Proceedings of the IEEE Transactions on Information Forensics and Security*. Piscataway, NJ: IEEE, 2017, 99.
51. L'Écuyer P, Simard R. TestU01: AC library for empirical testing of random number generators. *ACM T Math Softw* 2007;**33**:1–40.
52. Shwartz-Ziv R, Armon A. Tabular data: deep learning is not all you need. *Inform Fusion* 2022;**81**:84–90.
53. Panigutti C, Hamon R, Hupont I. *et al.* The role of explainable AI in the context of the AI act. In: *Proceedings of the 2023 ACM Conference on Fairness, Accountability, and Transparency*. New York, NY: ACM, 2023, 1139–50.
54. Cabitza F, Campagner A, Malgieri G. *et al.* Quod erat demonstrandum? Towards a typology of the concept of explanation for the design of explainable AI. *Expert Syst Appl* 2023;**213**:118888.
55. Koutsokostas V, Lykousas N, Orazi G. *et al.* Malicious MS Office documents dataset. Zenodo, 2021.
56. Afianian A, Niksefat S, Sadeghiyan B. *et al.* Malware dynamic analysis evasion techniques: a survey. *ACM Comput Surv* 2019;**52**:1–28.
57. Geng J, Wang J, Fang Z. *et al.* A survey of strategy-driven evasion methods for pe malware: transformation, concealment, and attack. *Comput Secur* 2024;**137**:103595.
58. Machin J, Batista E, Martínez-Ballesté A. *et al.* Privacy and security in cognitive cities: a systematic review. *Appl Sci* 2021;**11**:4471.
59. Casino F, Pina C, López-Aguilar P. *et al.* Sok: cross-border criminal investigations and digital evidence. *J Cybersecur* 2022;**8**:tyac014.
60. Mayuranathan M, Saravanan SK, Muthusenthil B. *et al.* An efficient optimal security system for intrusion detection in cloud computing environment using hybrid deep learning technique. *Adv Eng Softw* 2022;**173**:103236.
61. Qasem MH, Hudaib A, Obeid N. *et al.* Multi-agent systems for distributed data mining techniques: an overview. In: *Big Data Intelligence for Smart Applications*. Cham: Springer, 2022, 57–92.
62. Javadpour A, Pinto P, Ja'fari F. *et al.* DMAIDPS: a distributed multi-agent intrusion detection and prevention system for cloud IoT environments. *Cluster Comput* 2023;**26**:367–84.
63. Yuill J, Zappe M, Denning D. *et al.* Honeyfiles: deceptive files for intrusion detection. In: *Proceedings from the Fifth Annual IEEE SMC Information Assurance Workshop*. Piscataway, NJ: IEEE, 2004, 116–22.