Ransomware's Early Mitigation Mechanisms

Routa Moussaileb
IMT Atlantique
routa.moussaileb@imt-atlantique.fr

Benjamin Bouget
LHS-PEC, Inria

Aurélien Palisse
LHS-PEC, Inria
aurelien.palisse@inria.fr

Hélène Le Bouder
IMT Atlantique
helene.le-bouder@imt-atlantique.fr

Nora Cuppens
IMT Atlantique
nora.cuppens@imt-atlantique.fr

Jean-Louis Lanet
LHS-PEC, Inria
jean-louis.lanet@inria.fr

ABSTRACT

Ransomware remains a modern trend. Attackers are still using cryptovirology forcing victims to pay. Notable attacks have been spreading since 2012, starting with Reveton's ransomware attack to the more recent 2017 WannaCry, Petya and Bad Rabbit cyberattacks. This Ransomware as a Service (RaaS) can lure criminals into developing tools to perform an attack without previous knowledge of the cryptosystem itself. We present in this paper a graph-based ransomware countermeasure to detect malicious threads. It is a new mechanism that doesn't rely on previously used metrics in the literature to detect ransomware such as Shannon's entropy or system calls. An accurate detection is achieved by our solution. The per-thread *file system traversal* is sufficient to highlight the malicious behaviors. To the best of our knowledge, no previous study has been conducted in this area. The ransomware collection used in our experiments contains more than 700 active examples of ransomware, that were analyzed in our bar metal sandbox environment.

KEYWORDS

Intrusion Detection System, Ransomware, File System Traversal/Monitoring

1 INTRODUCTION

The first ransomware appeared in 1989. Since 2012, the number of ransomware victims has increased significantly. As shown in the survey [1], different ransomware exist as Reveton, CryptoLocker, CryptoWall and WannaCry. The ransomware as a service field enables any individual to launch his/her own developed attack without a previous profound knowledge of the target system. It represents a major reason for this business growth. Ransomware payload not only attacks computers but also cellphones as described in [2].

There is a high demand to mitigate ransomware infection process. Indeed, it is no longer affecting users' personal data or computer, but it actually undermines many public services. For example, a hospital has been hit by ransomware and its servers have been encrypted exposing more than 9k patients[3]. Symantec, one of the leading cyber security companies worldwide, has been able to block in the first half of 2017, 319k ransomware as shown in their annual report 2017[4].

Motivation. Our motivation to join this arms race against malware is the increased number of attacks in recent years and the polymorphism of such malware registered by antivirus software and, therefore, went undetected/unmitigated in early stages.

What makes criminal exposure even harder is the use of cryptocurrency such as bitcoin for the trade, which is nearly impossible to trace. It remains a valid model since attackers are peer pressuring victims who are willing to pay any amount to retrieve their data. To make matters even worse, evasion techniques are spreading at a high rate. It is a challenge for antivirus software to adjust to the ongoing ransomware evolution. This kind of global economy is beneficial for cybercriminals and is fed by people's lack of information about spam mails and other mechanisms that enable the spread of ransomware at a very high rate.

In the ransomware battle, one main goal is to restrain file losses if no prior detection was achievable. Current detection mechanisms rely on limiting the number of lost files, after the encryption phase, by blocking any process that has similar behavior/features to one of a ransomware(API calls, Registry keys, Embedded strings in binaries, etc). Nonetheless, residual risks still exist. One assumption could be outlined: no alarm is raised prior to any suspicious behavior. That being said, if a prevention mechanism is not able to detect a malicious software while it is still in its "footprinting" phase, other precautions are compulsory to prevent and restrain further damage and data loss in the system.

Therefore, countermeasures are necessary to limit malware impact. Moreover, in order to ensure an efficient detection, countermeasures should not be present in the user space as they usually are but rather in kernel space, therefore having at least the same privileges as the latter.

Contribution. The first step in prevention mechanisms is previous knowledge of existent threats encountered by a user to mitigate them as soon as possible. Being a particular part of malware's family, a ransomware cannot go unnoticed. Indeed, its end is very clear and noticeable by the victim since this malware will lock/encrypt the files of the victim until a ransom is paid.

This paper introduces a ransomware detection technique that serves as an Intrusion Detection System (IDS). More precisely, it targets crypto-ransomware since it presents a higher threat than cryptoLocker.

It is based on a file system exploration, before the attack (encryption) takes place. Indeed, once it is unpacked, ransomware's payload goal is to explore the file system to find files to encrypt. This search is done from the root of the file system or directly from user's folder with a depth-first or a breadth-first search. As for the exploration phase, threads that traverse the file system behave similarly and predictably, enabling the possibility of an early detection of the ransomware and therefore deducing its family.

The method mentioned above can help a user to protect the confidentiality and availability of his/her data while limiting the probability of an attack and minimizing losses.

Outline. The context of the ransomware is presented in section 2, including malware detection phases in section 2.2 and the state of the art in section 2.3. Prevention and detection mechanisms of ransomware are developed in section 3. The results of the experiments are described in section 4. Limitations are presented in section 6. Finally the conclusion is drawn in section 7.

2 CONTEXT

2.1 Ransomware

Ransomware is a specific type of malware that locks the access to user's data or computer until a ransom is paid. Two types of ransomware exist nowadays:

- Desktop locker that blocks users' system without encrypting their files
- Cryptographic ransomware that encrypts user files

Cryptographic ransomware are themselves divided into multiple categories based on the cryptosystem used, like private-key or public-key or hybrid.

Despite having diversified attack vectors, crypto-ransomware typical infection process and payload execution is common among all families as described in [5]. The classic cryptographic ransomware behavior is described in Fig 1.

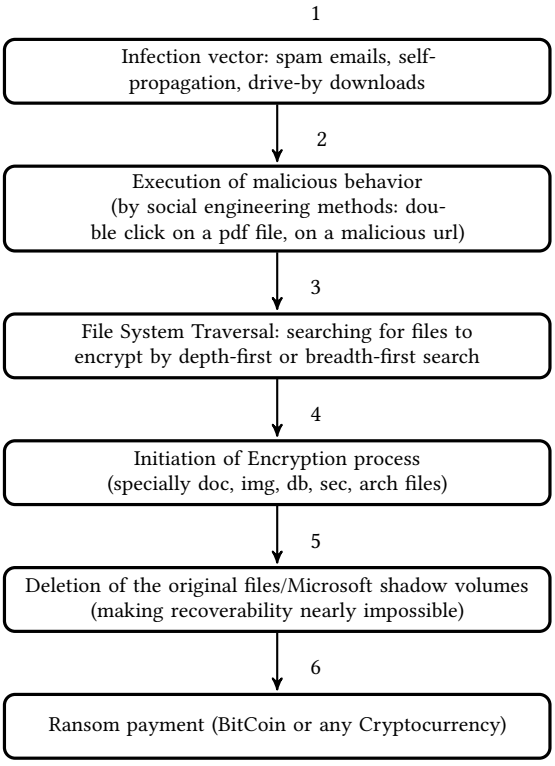

Figure 1: Ransomware's Workflow

2.2 Malware detection phases

Malware analysis is useful in the following three phases. The first one is related to prevention, *i.e.* avoiding to load a malicious binary file. The second one is related to run time behavior analysis, used to detect suspicious accesses. In that option, malicious code has bypassed the prevention mechanism's and the run time detection acts as a second line of defence. The last one is related to *post mortem* analysis, also known as forensic analysis. In this phase, the investigator tries to characterize the malware and understand its capability to bypass developed solution, to improve it. The first two options are considered as preventive actions while the last one is known as responsive action.

2.2.1 Prevention. Malware detection and prevention are critical for connected devices' protection across the Internet. Antivirus software uses signature files and heuristics to detect the threats. Traditional signature-based malware detectors fail to recognize polymorphic malware since they are obfuscated and zero-day executables. However, despite having multiple code versions, some semantics of the original malware are preserved such as its payload. Both static and dynamic analysis are convenient techniques to extract behavioral signature and classify malware.

2.2.2 Run Time analysis. Runtime malware analysis is also known as Intrusion Detection System (IDS). Host based

IDS collects and analyzes information gathered on a particular host or system, such as system logs, system processes, files, or network interface. Typical analysis techniques consist of function call/instruction trace monitoring and information flow tracking. These techniques monitor the program behavior during execution and potentially detect malware variants. Yet, they have a high false positive rate and an important performance overhead due to high resource demands.

2.2.3 Forensic Analysis. When malware is discovered on a system, before a shutdown, an analysis requires reconstructing a vivid picture of events surrounding the malware infection to gain a detailed understanding of the malware itself. This process needs an access to the volatile memory and logs if available. Malware can mitigate this dumping process by deleting a section of code as soon as it has finished executing. This technique is known as *stolen bytes*. These bytes must be restored if the dumped program is to be run again. The memory snapshot is thus incomplete or incoherent. The collected data can be used to understand the infection process or to classify the malicious file thanks to the data recovered in the memory.

2.3 State of the art

Previous contributions [6–12] are based on user space with cryptographic hooks or kernel space with file system encryption detection mainly based on the Shannon entropy. Both approaches suffer from false positives (*e.g.*, compression) and false negatives (*e.g.*, static libraries, encryption preserving the plaintext distribution). Moreover, the file system ransomware countermeasures [8, 9, 12] are impractical because of performance loss and unusable through limited false positives evaluation. Despite excellent detection results (*i.e.*, over 99%) with the existing countermeasures [8, 9, 11, 12], more advanced ransomware will pass through. Indeed, very simple and low cost techniques can be used to circumvent the state of the art solutions as their authors have acknowledged.

The previous ad hoc countermeasures [8–12] rely mainly on two features: the increase of Shannon entropy of the user's files and the cryptographic materials.

For historical reasons and practicality, the Microsoft's Cryptographic Application Programming Interface (CryptoAPI) is used by diverse ransomware families to perform file encryption. CryptoLocker and CryptoWall, use Microsoft CryptoAPI to encrypt victims' files. Chen *et al.*, in [13] suggest multiple classifiers (Random Forest, Support Vector Machine,etc.) to provide dynamic ransomware detection. To sum up their work, various Application Programming Interface (API) calls are collected when a software is executed in a virtual machine, then a Call Flow Graph is built. Preprocessing was made (normalization and feature selection) to train aforementioned classifiers. Palisse *et al.* [10] and Kolodenker et al. [11] each implemented a userland countermeasure based on this observation. This monitoring enables full control of key generation and encryption.

If backing up all the data is not possible, an individual can back up the key used during encryption. PAYBREAK [11] is at the heart of this idea. It is a proactive defence mechanism consisting of three different components. They enable the victim, by means of crypto function hooking to store different keys used by ransomware in its encryption process in a key vault. Later on, the victim can retrieve the key required for decrypting the files. Yet, PAYBREAK does not resist to all obfuscation threats.

Other features such as Hardware Performance Counters (HPC) [14] seem to be a viable option for distinguishing between benign and malicious processes. HPC was passed to a long-short-term-memory based auto-encoder for unsupervised anomaly detection. Fast Fourier Transformation was introduced to detect repetitive pattern of a ransomware that is opening/closing/deleting/encrypting a file. Homayoun *et al.* prefer using supervised learning algorithms (J48, Random Forest, and Bagging) proceeded by sequential pattern mining for software classification [15]. They rely on different patterns to expose ransomware (Registry, DLL and File System events).

A second approach exists and is based on file system monitoring through a driver in the kernel land. It has a very small chance to be detected and disabled by the ransomware. The file system drivers to date, monitor at least all the read and write operations on the disk [8, 9, 12]. Then for each read and write operation pre- or post-processing is performed (*e.g.*, Shannon entropy increase). Unfortunately, entropy fails when trying to distinguish between encryption and other operations such as compression. Another drawback of entropy is its inability to detect developed attacks described in Shukla *et al.*[16].

Findings of data-centric ransomware behavior in [8] revealed that a precise type of search is performed in order to encrypt victims' files. In addition to that, recent ransomware attacks were developed by Shukla *et al* [16] where they shared a common event, which is the "enumeration of all interesting files on volume". The research presented in our paper will extend this aspect making it a plausible feature not only for ransomware detection but also for classification.

ShieldFS [12], the latest file system minifilter driver, makes use of a supervised classifier (*i.e.*, random forest) trained with six features to detect malicious activities on the disk. Excellent detection rate (*i.e.*, over 99%) is achieved by each ad hoc countermeasure [8, 9, 11, 12]. But only few families have been tested compared to the dataset found by the antivirus companies. Moreover, we believe that more sophisticated ransomware will easily defeat the countermeasures [8–12].

At the outset, ransomware is capable of making use of the Intel AES-NI instructions [17] and thus passes through [10, 11]. Monitoring the disk activity with a file system driver seems to be the most promising approach for the live solutions, as [8, 9, 12] proceeded.

To compete in the arms race of ransomware, numerous behavioral aforementioned traits and others such as change of mime type and Shannon entropy are at our disposal. Our paper delves into file system traversal since, to the best of our knowledge, this approach has not been explored by researchers as an intrusion detection tool of ransomware. In its second part, the paper targets the fine distinction between ransomware's and benign process occurrence. As stated in [18], normal behavior has a high probability of occurrence compared to anomalies which have a low probability that stays below a certain threshold.

3 PREVENTION CAPABILITIES

Any prevention tool must be able to classify a given binary as suspicious or not. For this purpose, static or dynamic analysis can be used. Nevertheless, modern malware employs stealthy techniques in an attempt to remain undetected on diverse systems making it difficult to analyze. Static analysis may quickly reveal the presence of obfuscation (e.g., packing, virtualization) or other high level anti-static analysis techniques. Thus, static analysis may reveal just a glimpse of beneficial information to the analyst. For dynamic analysis mitigation, malware has the opportunity to examine the environment it is running in and alter its behavior if it detects an ongoing dynamic analysis.

To avoid these pitfalls, a new defence mechanism against ransomware is developed. Our approach does not rely on code examination, code structure or access to system's calls. A module that analyses suspicious software behavior is designed while accessing the file system. All ransomware of our provided collection, except one, evaluate users' data by traversing all folders except for some specific ones. Some ransomware avoid certain folders which is a signature of their original code. For this reason, execution traces and in particular traversal's order and avoided folders are collected.

In this section, different models to detect ransomware's attacks are presented. The novelty is the need of exclusively one information: *file system traversal*. Moreover, an accurate classification can be drawn with these observations. Thus any binary can be flagged as malicious or not.

3.1 Black- and Whitelists

Nowadays, most of the ransomware families available in multiple online databases implement a naive exploration of the file system. In other words, they explore the file system with well-known algorithms: depth-first or breadth-first search. Nevertheless, slight differences between them can be seen. Black- and whitelists of folders are embedded by ransomware's authors into malicious binaries. They represent two ways of filtering access to folders. Whitelists represent a set of folders that can be accessed by a ransomware. For example, a Cerber sample especially targets multimedia folders (*e.g.*, C:/program files (x86)/steam/). Whereas blacklists is the reverse of whitelists: folders that are omitted and denied from ransomware's path. For instance, C:/Windows/ system folders are avoided by malware to let the machine run normally. Finally, the environment variables allow the attacker to directly attack users' documents rather than beginning the exploration from the root as most of the analyzed ransomware do. The above information will be used for ransomware detection in the following parts.

3.2 Decoy score

The first prevention mechanism we suggest relies on the idea that ransomware, intrusively or passively, scans specific files and folders that enable not only their detection but also categorization[19]. In fact, some directories and files are rarely visited by the user or by one of the system's regular tools and thus can be considered as a trap. If these files are manipulated by a software it can indicate an illegal and unwanted access. They are referred to as Decoy Folders.

Recycle_bin	;	(C:\$Recycle.Bin)
MSI_Config	;	(C:\MSI_Config)
FoldMaj	;	(C:\FoldMaj)
Perf_log	;	(C:\PerfLogs)
Prog_data	;	(C:\Prog_data)

Figure 2: The list of decoy folders used to compute the per-thread score.

The main algorithm is the following:

Algorithm 1 Ransomware Detection

```
1:  procedure
2:      def detect_suspicious_behavior(thread, threshold):
3:      label_array ← {Decoy Folders}
4:      score_array ← {false, false, false, false, false}
5:      for path ∈ thread.paths do
6:          if path ∈ label_array then
7:              index ← label_array.index(path)
8:              score_array[index] ← true
9:      if evaluate_score(score_array) >= threshold then
10:         return is_suspicious
11:     return is_not_suspicious
```

Our suggested solution checks if a thread passes in specific folders. If so, it marks them and then increments the decoy folder counter. The final score is normalized (*i.e.*, divided by the number of "decoy folders" that was added in the beginning), in case the threshold is reached, the thread is recognized as malicious. The detailed algorithm is presented in Algorithm 1. It is unlikely that a normal thread will pass through at least 3 of these Decoy Folders. This is why the threshold has been set to 0.6. If it is lower some ransomware will not be detected (for example if a ransomware directly modifies the file it will not go through bin recycle), in addition, many benign threads will be flagged. If it is greater, others will also escape detection.

3.3 Graph Similarity

Each ransomware's file system traversal can be compared to others in order to know if they belong to the same family, or if they share some code concerning the paths exploration. As a first step, we build a graph of the explored folders for each sample. Then, the similarity matrix corresponding to the ransomware dataset is computed seen in Figure 3. Finally, a classification of the samples is done based on the similarity matrix using hierarchical clustering technique, which provides us a dendrogram.

3.3.1 Hierarchical Graph. All machines used to collect the data from the bare metal platform have the same configuration and a similar hardware. Each line of raw data used in the experiment correspond to the nature of the operation on the file system (read file and open directory), the thread `pid` and a time stamp. This data is enough to trace the complete graph traversal even if the write operation is useless for this phase.

Each node represents a file system folder that has been opened by a suspicious thread. File system traversals are represented as oriented graphs (via time stamps). Edges represent transition from parent to child folders. In order to be scalable (*i.e.*, graph size) and generic (*i.e.*, distinct Windows installations), subgraphs have been pruned at specific file system positions (*e.g.*, C:/Program_Files/). The number of pruned sub-folders is stored within the graphs' edges.

A graph is defined as the tuple $G = (V, E, \mu, \nu)$ where:

- V the set of nodes
- E the set of edges
- η the set of nodes not pruned covered by a ransomware
- θ the set of paired nodes with the number of pruned sub-folders
- $\mu : V \rightarrow \eta$
- $\nu : E \rightarrow \theta$

3.3.2 Adjacency similarity. A comparison between a given trace (*i.e.* a graph) with other graphs is needed. In this case, graph (or sub-graph) homomorphism is not a viable solution. Each version of a malware can exclude some paths leading to a slightly different graph. The concept of matching cost to penalize structural differences is introduced. The closer the structures of the two graphs are, the lower the cost to match them. Graph similarity techniques can be classified into three main categories: edit distance/graph isomorphism, feature extraction, and iterative methods [20]. The drawback of graph isomorphism is that the algorithms are exponential and, thus, not applicable to the large graphs that are of interest to us. The feature extraction approach relies on graph properties such as degree distribution, diameter, etc. This method scales well, but depending on the metrics that are chosen, it is possible to get high similarity between two graphs that have very different node set size. Iterative methods are based on the fact that two nodes are similar if their neighbourhoods are also similar. This latter method is chosen using an adjacency similarity algorithm.

To compute the similarity matrix between all the graphs, *Graph-tool* [21] has been used. It is a free framework for creating and manipulating graphs. The core of this framework is written in $C++$ which makes it fast even for large graphs. A built-in *Graph-tool* function computes the adjacency similarity between two graphs. It corresponds to the number of edges that have the same source and destination in both graphs. The labels of vertices are used to build the adjacency matrices and thus make the comparison. The higher the score is, the higher the similarity.

3.3.3 Classification. Then, to classify the samples, unsupervised hierarchical clustering over this similarity matrix is used. A dendrogram is used to represent the classification. It is a visual representation of the compound correlation data. The individual compounds are arranged along the bottom of the dendrogram and referred to as leaf nodes. Compound clusters are formed by joining individual compounds or existing compound clusters with the join point referred to as a node. The leafs of the tree are the name of the classified ransomware.

3.4 Supervised Machine Learning

In this section, an improvement of the previous classification is made by using a supervised approach. Thread level granularity is maintained throughout the whole experiment. In addition to the access to previously mentioned decoy folders in Figure 2, other features were taken into consideration to perform a supervised learning on the collected information.

This method is not limited to file system's traversal but overall and per decoy folder velocity is taken into consideration. In fact, it is not sufficient that a thread explores only decoy folders to be marked as malicious, the time spent in

each decoy folder and in total is crucial and needs to be considered for a better classification.

Paths_total	Total number of explored paths
Time_total	The file systems traversal duration
{Decoy_folder}_paths	The number of subfolders explored in the current decoy folder (Updated Decoy Folders: Recycle_bin, Perf_log, Windows, Python, Prog_data, Prog_files, Recovery)
{Decoy_folder}_time	The timestamp of the first subfolder explored in the current decoy folder

Table 1: The list of features used to train the classifiers.

3.5 File System Traversal Velocity

Another behavioral property is additionally investigated, the execution time (the velocity of the file traversal) of a family. Indeed, the samples issued from the same families have similar patterns. Each payload can be packed or obfuscated individually, but the system impact will remain the same for a particular family with the exception of new versions.

4 EXPERIMENTAL RESULTS

4.1 Data Collection

Needed data is collected from an automated bare metal malware analysis platform built from scratch. This data is represented in the JavaScript Object Notation (JSON) format and provides, for each userland thread of the system, a complete list of explored folders. Similar to [8, 9, 12], a file system driver is used to monitor the runtime behavior of each thread.

A crawler downloads a ransomware from well-known databases, then it is executed on windows 7/10 machines for a period of 15 minutes. A dump corresponding to this malware behavior is saved for analysis.

Moreover, during the analysis, thread's write operations are passed through an indicator of compromise: Shannon's entropy. As a result, within each dump, malicious threads are explicitly marked. A manual verification of all threads marked as malicious for all the dumps is then implemented (e.g., eliminates the false positives). The ransomware collection contains more than 700 active ransomware spread to across twenty families.

Sample labelling is achieved through Avclass tool [22] and VirusTotal [23]. A bare metal platform has been preferred to the solutions based on virtualization because of the numerous techniques used by the malware to fingerprint well-known sandboxes(e.g., Cuckoo Sandbox [24]). For the time being, the countermeasure presented in this paper is post-mortem (i.e., based on the sandbox analysis), however it could be incorporated into a live solution.

Benign data is collected from various users utilizing their computers for work purposes. Technically speaking, the information gathered corresponds to web browsing, software development, file encryption… The same information is collected for benign and malicious software: Files Traversed by threads.

For scalability reasons, parallel machines could be used to perform the tests as well as an improved disk image distribution.

4.2 Data set

4.2.1 Training Dataset. As previously mentioned, 770 active ransomware were executed on Windows OS both 7 and 10. However, 76 ransomware's binary hash corresponded to multiple ransomware categories such as Yakes or Teslacrypt or Shade and Barys. Therefore, these records are omitted from ransomwares family classification during the supervised learning phase. They serve as ID specific to each family of ransomware. For example,(1eb412a5f6400eb490a8698dc08129da) hash or (46b9fc70dc137c0d978ce16364a15c27) hash.

Since the overall database of malware collection contains 694 active ransomware, 417 ransomware records were used to act as training set completed with 417 records of benign computer usage. Benign data is collected on Windows 10 computers where a user is usually surfing the Internet, playing online games, developing a software, etc. They correspond to end user's daily activity.

Family	Samples	Family	Samples
Teslacrypt	115 (27.58%)	Firefox.exe	53 (12.74%)
Cerber	79 (18.94%)	Explorer.exe	47 (11.30%)
Xorist	74 (17.75%)	Svchost.exe	37 (8.89%)
Bitman	59 (14.15%)	Pnamain.exe	31 (7.45%)
Deshacop	13 (3.12%)	Receiver.exe	29 (6.97%)
Zerber	13 (3.12%)	Avp.exe	26 (6.25%)
Yakes	13 (3.12%)	Mscorsvw.exe	17 (4.09%)
Locky	7 (1.68%)	WmiPrvSE.exe	16 (3.85%)
Gpcode	6 (1.44%)	BackgroundTask	14 (3.37%)
Rest	38 (9.11%)	Rest	146 (35.10%)

Table 2: An overview of the active ransomware families and goodwares used in the experiments, ranked in descending order according to their samples number.

The holdout method is used to evaluate different supervised machine learning models. In order to apply those algorithms and evaluate their performance, Python [25] is used.

More specifically, scikit-learn library since it represents an efficient tool for data mining and data analysis. In our case classification and clustering [26]. Avoiding overfitting on the non malicious threads is crucial to have better results which will enable us to generalize our model.

4.2.2 Learning Phase. The training set also consists of equally partitioned data across all families of ransomware and different types of benign processes. The same distribution is kept to remove any bias in the data. Multiple classifiers have been trained: k-nearest neighbors [27], decision tree [28] and random forest [29]. The decision tree builds a graph-tree based on the features and gives a result with the output variable (*i.e.*, malicious or benign) on the leaf, whereas random forest builds decision trees with a subset of the features selected randomly. The final result is given with a majority vote.

4.3 Results

4.3.1 Decoy Folders. The objective in this first detection mechanism is a binary classification of a record: benign or malicious. 99.35% of malicious threads are correctly detected. It highlights the fact that the majority of the ransomware collection begins file system's exploration from the hard disk root. Less than 1% of benign data have been classified as malicious.

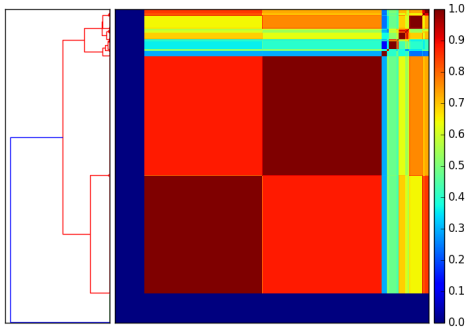

Figure 3: Malicious threads file system's traversal similarity matrix

4.3.2 Ransomware's Graph. The computation for all pairs of graphs, gives a similarity matrix as represented in Figure 3. A dozen of ransomware groups (*i.e.*, families) can be seen. Two groups represent 80% of the file systems traversal distribution. However, 10% of the samples are uncorrelated to others (*i.e.*, the blue block). The distance matrix shows that ransomware up to date have little diversity concerning the file system's exploration. This can be explained by the fact that most of them use the Windows API for accessing the files.

We used the Avclass tool [22] to obtain the labels of the evaluated binaries. We present here a partial view of the dendrogram (12 classified over the ransomware samples). We can notice that the `TeslaCrypt` and the `bitman` are very close. This means that they share the same traversal algorithm, white list and black list. Another similarity is between Cerber and Zerber ransomware. The latter can be considered as a simple variant of the first.

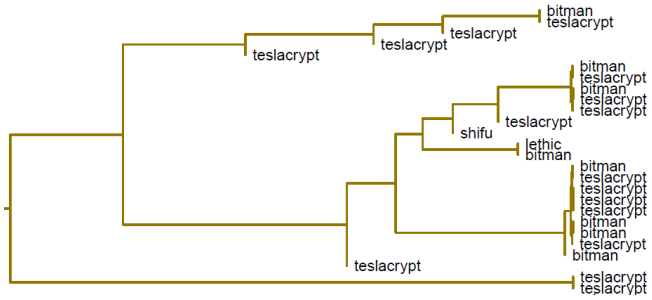

Figure 4: Families Graphical classification dendrogram

The (partial) dendrogram presented in Figure 4, shows that samples of the same family are successfully grouped on the same branch. The families that are closed to each other are also grouped together (*i.e.*, same branch).

But a difference is still noteworthy. That indicates that ransomware can be classified according to their file systems traversal, even when thin differences are present between families. In the next section, another feature is used to classify the samples into families.

The list of abbreviations used in Figure 5 is presented below:

HD	:	HardDiskVolume2
P_D	:	ProgramData
Py	:	Python26
$R.B	:	$Recycle.Bin
Rec	:	Recovery
Win	:	Windows

4.3.3 Supervised Learning. All of the supervised learning algorithms are able to distinguish between a ransomware and benign application file system traversal as seen in Table 3. Binary classification (benign vs ransomware) is efficient in this case. However, to take one step further than the decoy folder detection, an analysis of ransomware families is carried out. For the Random Forest Classifier, 68.63 % of malicious records were correctly classified as so: a binary is no longer flagged as benign or malicious, we were also

Supervised Learning Algorithm	True Positive Rate	True Negative Rate	False Positive Rate	False Negative Rate	Training Time (seconds)
K nearest neighbor (n=3)	97.67	97.11	2.89	2.33	0.0039
Decision Tree	100	100	0	0	0.0095
Random Forest	100	100	0	0	0.0611
Gaussian naive bayes	100	97.47	2.53	0	0.0146

Table 3: Classifiers Performance Metrics

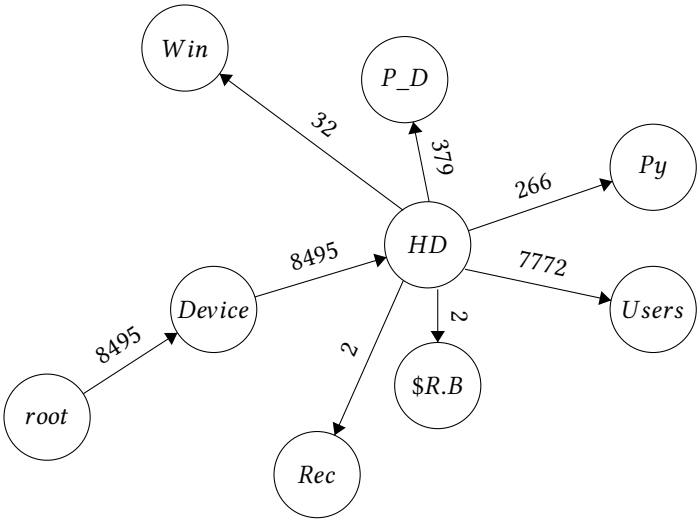

Figure 5: Xorist's File Traversal Subgraph

family	Bitman	Teslacrypt	Normal
nb_paths	8199	8199	8
time_total	996315740	987726399	872039
RECYCLE_BIN_aggreg	2	2	0
RECYCLE_BIN_time	1460	3799	0
PERF_LOG_aggreg	1	1	0
PERF_LOG_time	4742522	4274485	0
PYTHON_aggreg	266	266	0
PYTHON_time	4987588	4519400	0
PROG_DATA_aggreg	188	188	5
PROG_DATA_time	131810705	132116153	772436
PROG_FILES_aggreg	0	0	1
PROG_FILES_time	0	0	455083
WINDOWS_aggreg	0	0	2
WINDOWS_time	0	0	450114
RECOVERY_aggreg	2	2	0
RECOVERY_time	131625192	131958939	0

Table 4: Benign and Ransom Records

able to identify ransomware's family (eg. Locky, Yakes). Indeed, there's a similarity between Bitman and Teslacrypt ransomware on one hand, and Zerber and Cerber on the other hand. The main reason behind this correlation is that they belong to the same family, thus will most probably behave in similar ways to traverse the file system and encrypt their files. Decision Tree Classifier achieved a 61.25 % of correct ransomware classification.

Figure 6 illustrates decision tree's rules to perform the split, thus to classify the records. It is limited to 4 leaf nodes to be able to represent it. Indeed, normal application does not pass constantly through those decoy folders. The majority of benign records have non negligible values in the Prog_Files time and Windows_time. It shows the time spent by some applications such as firefox.exe or explorer.exe in those decoy folders.

The similarity between a Teslacrypt and Bitman's execution is shown through the records presented in Table 4.

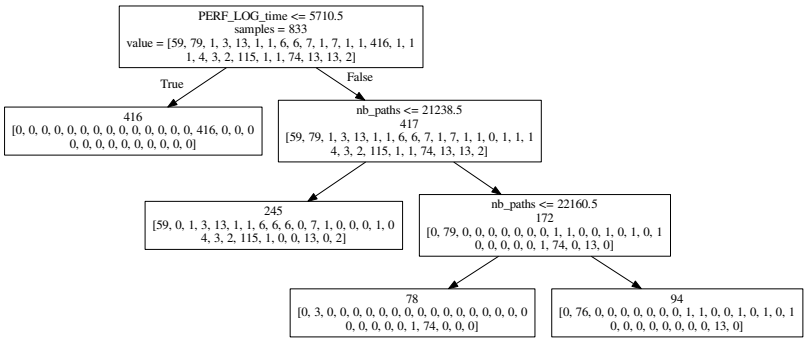

Figure 6: Decision Tree Classification Parameters

4.3.4 File System Traversal Velocity. Figure 7 illustrates such statement for the `xorist` malware. Figure 8 demonstrates the `bitman` malware. We can observe that two implementations of the file traversal algorithm exist.

The raw data used in the experiment have been gathered from the same machines. These tests were made from a

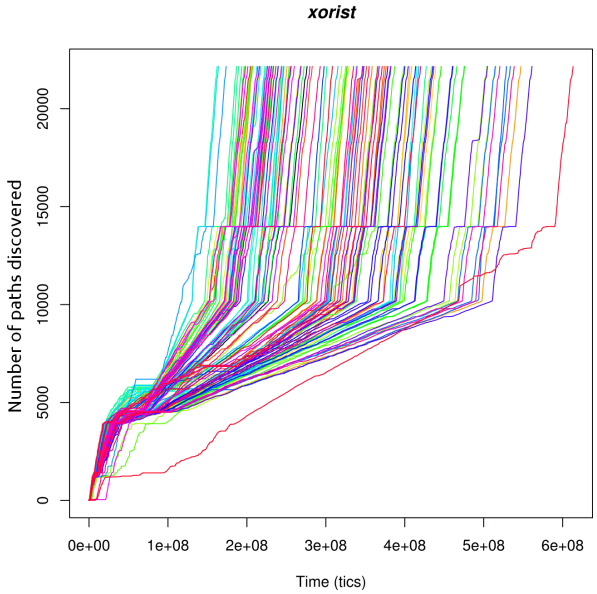

Figure 7: The file system's traversal velocity of the 125 Xorist samples

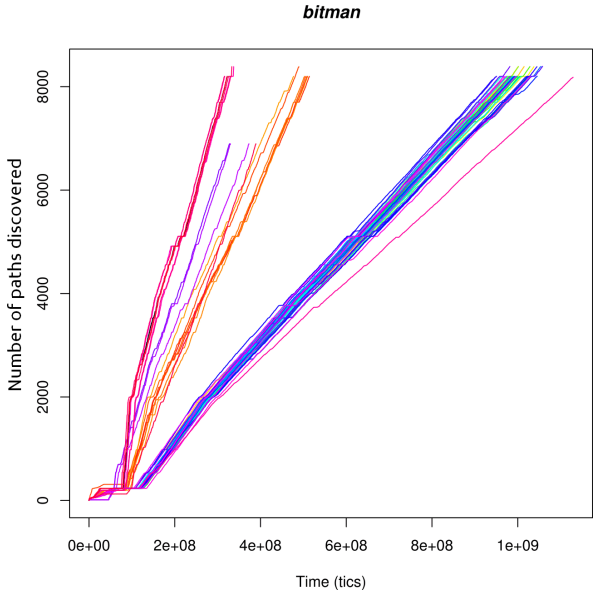

Figure 8: The file system's traversal velocity of the 57 `bitman` samples

sample of the available database. The time unit is the performance counter value (*i.e.*, OS internal) in units of processor ticks since the beginning of the session which is comparable across the analyses. The speed depends on the ransomware design. As an example, the `Cerber` family searches files in one thread and encrypts them in another, whereas, the `xorist` family uses the same thread to search and encrypt the files. Moreover, the programming or compiling choices cause some differences at runtime. To conclude, the velocity indicator provides an additional signature to distinguish between ransomware families. For other uninfected applications, no multiple file opening is shown.

5 ACKNOWLEDGEMENTS

The authors would like to thank CPER SSI for the infrastructure provided and partial funding.

6 LIMITATIONS

Since ransomware behave similarly in the file system traversal, more features need to be taken into consideration for families classification. In addition to that, any software that mimics the behavior of ransomware's traversal will be classified as malicious so our proposal raises false positives in this case. Another limitation occurs when no file system traversal is done prior any encryption.

7 CONCLUSION

In this work, we were able to detect ransomware behavior based only on monitoring *file system traversal*. We concluded that the majority of ransomware start their encryption process from the root of the hard disk. To get a precise ransomware's classification, machine learning techniques were used. Based only on decoy folders, we were able to detect ransomware from various families. As for our future work, we will gather additional features to be able to distinguish accurately between various ransomware families even if they share some specific behaviors such as file system traversal noticed for example between the Cerber and Zerber families. Furthermore, a signature of each ransomware could be extracted.

REFERENCES

[1] A Gandhi Krunal. Survey on ransomware: A new era of cyber attack.
[2] Pavol Zavarsky, Dale Lindskog, et al. Experimental analysis of ransomware on windows and android platforms: Evolution and characterization. *Procedia Computer Science*, 94:465–472, 2016.
[3] HIPAA Journal. Cardiology Center of Acadiana Ransomware Attack Impacts 9,700 Patients. hipaajournal.com, April 2017.
[4] ISTR. Symantec annual report, 2017.
[5] Daniel Gonzalez and Thaier Hayajneh. Detection and prevention of crypto-ransomware. In *Ubiquitous Computing, Electronics and Mobile Communication Conference (UEMCON), 2017 IEEE 8th Annual*, pages 472–478. IEEE, 2017.
[6] Adam L Young and Moti M Yung. An implementation of cryptoviral extortion using microsoft's crypto api. 2005.
[7] Amin Kharraz, William Robertson, Davide Balzarotti, Leyla Bilge, and Engin Kirda. Cutting the gordian knot: a look under the hood of

ransomware attacks. In *Detection of Intrusions and Malware, and Vulnerability Assessment*, pages 3–24. Springer, 2015.

[8] Nolen Scaife, Henry Carter, Patrick Traynor, and Kevin RB Butler. Cryptolock (and drop it): stopping ransomware attacks on user data. In *Distributed Computing Systems (ICDCS), 2016 IEEE 36th International Conference on*, pages 303–312. IEEE, 2016.

[9] Amin Kharraz, Sajjad Arshad, Collin Mulliner, William Robertson, and Engin Kirda. UNVEIL: A Large-Scale, Automated Approach to Detecting Ransomware. In *Proceedings of the 25th USENIX Security Symposium, Austin Texas*, pages 757–772. Usenix, 2016.

[10] Aurélien Palisse, Hélène Le Bouder, Colas Le Guernic, Axel Legay, and Jean-Louis Lanet. Ransomware and the legacy crypto api. In *The 11th International Conference on Risks and Security of Internet and Systems*, 2016.

[11] Eugene Kolodenker, William Koch, Gianluca Stringhini, and Manuel Egele. Paybreak: Defense against cryptographic ransomware. In *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security*, pages 599–611. ACM, 2017.

[12] Andrea Continella, Alessandro Guagnelli, Giovanni Zingaro, Giulio De Pasquale, Alessandro Barenghi, Stefano Zanero, and Federico Maggi. Shieldfs: a self-healing, ransomware-aware filesystem. In *Proceedings of the 32nd Annual Conference on Computer Security Applications*, pages 336–347. ACM, 2016.

[13] Zhi-Guo Chen, Ho-Seok Kang, Shang-Nan Yin, and Sung-Ryul Kim. Automatic ransomware detection and analysis based on dynamic api calls flow graph. In *Proceedings of the International Conference on Research in Adaptive and Convergent Systems*, pages 196–201. ACM, 2017.

[14] Manaar Alam, Sarani Bhattacharya, Debdeep Mukhopadhyay, and Anupam Chattopadhyay. Rapper: Ransomware prevention via performance counters. *arXiv preprint arXiv:1802.03909*, 2018.

[15] Sajad Homayoun, Ali Dehghantanha, Marzieh Ahmadzadeh, Sattar Hashemi, and Raouf Khayami. Know abnormal, find evil: frequent pattern mining for ransomware threat hunting and intelligence. *IEEE Transactions on Emerging Topics in Computing*, 2017.

[16] Manish Shukla, Sutapa Mondal, and Sachin Lodha. Poster: Locally virtualized environment for mitigating ransomware threat. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 1784–1786. ACM, 2016.

[17] PolarToffee. Found a sample of the aes-ni ransomware. twitter.com/PolarToffee, April 2017.

[18] Varun Chandola, Arindam Banerjee, and Vipin Kumar. Anomaly detection: A survey. *ACM computing surveys (CSUR)*, 41(3):15, 2009.

[19] Jeonghwan Lee, Jinwoo Lee, and Jiman Hong. How to make efficient decoy files for ransomware detection? In *Proceedings of the International Conference on Research in Adaptive and Convergent Systems*, pages 208–212. ACM, 2017.

[20] Danai Koutra, Ankur Parikh, Aaditya Ramdas, and Jing Xiang. Algorithms for graph similarity and subgraph matching. In *Proc. Ecol. Inference Conf.*, 2011.

[21] Tiago de Paula Peixoto. Graph-tool: Efficent network analysis with Python. graph-tool.skewed.de.

[22] Marcos Sebastián, Richard Rivera, Platon Kotzias, and Juan Caballero. Avclass: A tool for massive malware labeling. In *International Symposium on Research in Attacks, Intrusions, and Defenses*, pages 230–253. Springer, 2016.

[23] Virustotal. https://www.virustotal.com/.

[24] Cuckoo Foundation. Cuckoo Sandbox: Automated Malware Analysis. cuckoosandbox.org.

[25] Python. https://www.python.org/.

[26] Scikit-learn. http://scikit-learn.org/.

[27] Daniel T Larose and Chantal D Larose. k-nearest neighbor algorithm. *Discovering Knowledge in Data: An Introduction to Data Mining, Second Edition*, pages 149–164, 2005.

[28] S Rasoul Safavian and David Landgrebe. A survey of decision tree classifier methodology. *IEEE transactions on systems, man, and cybernetics*, 21(3):660–674, 1991.

[29] Tin Kam Ho. Random decision forests. In *Document Analysis and Recognition, 1995., Proceedings of the Third International Conference on*, volume 1, pages 278–282. IEEE, 1995.