Journal of Computer Security 28 (2020) 337–373
DOI 10.3233/JCS-191346
IOS Press

Multilayer ransomware detection using grouped registry key operations, file entropy and file signature monitoring

Brijesh Jethva [a], Issa Traoré [a,*], Asem Ghaleb [a], Karim Ganame [b] and Sherif Ahmed [c]
[a] *Department of Electrical and Computer Engineering, University of Victoria, BC, Canada*
E-mails: bjethva@uvic.ca, itraore@ece.uvic.ca, aghaleb@uvic.ca
[b] *Efficient Protections Inc., QC, Canada*
E-mail: ganame@streamscan.io
[c] *Department of Computer Science, University of Windsor, ON, Canada*
E-mail: Sherif.SaadAhmed@uwindsor.ca

Abstract. The last few years have come with a sudden rise in ransomware attack incidents, causing significant financial losses to individuals, institutions and businesses. In reaction to these attacks, ransomware detection has become an important topic for research in recent years. Currently, there are two broad categories of ransomware detection techniques: signature-based and behaviour-based analyses. On the one hand, signature-based detection, which mainly relies on a static analysis, can easily be evaded by code-obfuscation and encryption techniques. On the other hand, current behaviour-based models, which rely mainly on a dynamic analysis, face difficulties in accurately differentiating between user-triggered encryption from ransomware-triggered encryption. In the current paper, we present an upgraded behavioural ransomware detection model that reinforces the existing feature space with a new set of features based on grouped registry key operations, introducing a monitoring model based on combined file entropy and file signature. We analyze the new feature model by exploring and comparing three different linear machine learning techniques: SVM, logistic regression and random forest. The proposed approach helps achieve improved detection accuracy and provides the ability to detect novel ransomware. Furthermore, the proposed approach helps differentiate user-triggered encryption from ransomware-triggered encryption, allowing saving as many files as possible during an attack. To conduct our study, we use a new public ransomware detection dataset collected in our lab, which consists of 666 ransomware and 103 benign binaries. Our experimental results show that our proposed approach achieves relatively high accuracy in detecting both previously seen and novel ransomware samples.

Keywords: Ransomware detection, machine learning, file entropy, file signature

1. Introduction

Ransomware is a type of malicious software that prevents users from accessing systems or files; this is done either by locking the user's device or encrypting the user's files. Users have to pay a required amount of money (ransom) to get back access to the locked systems or encrypted files. Attackers usually demand the payment in Bitcoins because of the anonymity associated with this cryptocurrency [20]. Recently, the number of ransomware attacks has grown significantly, and ransomware variants have adopted sophisticated techniques for spreading, encrypting and evading defensive mechanisms. In addition, autonomous ransomware variants are becoming popular; in these cases, the encryption is done

*Corresponding author. E-mail: itraore@ece.uvic.ca.

offline using legitimate OS features, such as Crypto APIs, eliminating the need for communication with the command and control (C&C) server. As the number of attacks has increased, ransomware has become a prominent threat to businesses and individuals [24]. The success of ransomware is because it gives attackers the ability to make huge monetary gains with limited effort and a low risk exposure, especially when critical or embarassing information for the victims is at stake. Ransomware represents one of the most disruptive forms of security threats in the world. WannaCry, one of the latest strains of ransomware, has reportedly infected over 200,000 machines around the globe [30]. Recently, the Danish shipping company A.P. Moller-Maersk was hit by the NotPetya ransomware attacks, disrupting its business processes and costing the company over $200 million in lost revenue [22].

Despite the growing number of ransomware infections, their increasing sophistication, and their significant financial and operational impacts, the current defensive mechanisms face challenges in curbing the threat. Most of the existing approaches are signature based and hence struggle with the evolving nature of ransomware, of which there are currently over 160 different strains [34].

Ransomware detection techniques fall under the same general categories as existing general malware detection approaches, which broadly include signature-based analysis and behavioural analysis. Signature-based detection can detect only known ransomware; it is not helpful in detecting novel ransomware. A behavioural analysis consists of the live monitoring of processes to identify anomalous behaviours. This involves analysing all requests access to specific files, processes, connections or services, including each low-level instruction executed at the operating system level or any other programmes that have been invoked. Until now, most of the work done on ransomware detection using behavioural analyses have focused on training a machine learning model on a limited types of features. The existing approaches rely on extracting and analysing API call sequence and/or checking entropy at kernel level for file operations (e.g., file read, delete, etc.). One of the main issues with these approaches is that some legitimate applications (e.g., 7-zip) exhibit the same behaviour as ransomware when it comes to file operations. Also, API calls are system dependent, raising the issue of custom libraries (e.g., for encryption, file operations, etc.) used by some ransomware to evade detection. Furthermore, many current behaviour-based detection approaches rely on features that are specific to a particular ransomware family or ransomware binary referred to as binary features. Binary features are not helpful in detecting new variants of the ransomware because these may contain a new set of processes.

The purpose of the current research is to detect both existing and previously unseen ransomware by creating a forward-looking system for monitoring the ransomware system activity. We make a step forward by proposing, implementing and evaluating an approach that combines automatic detection and file backup on a Windows system. We introduce an upgraded behavioural based ransomware detection system by exploring different machine learning classifiers and reinforce the existing feature space by introducing a new set of features based on grouped registry key operations, along with a monitoring model based on combined file entropy and file signature.[1] To mitigate the impact of custom libraries, we introduce a set of new features based on group registry key operations, focusing on registry key hives. The existing features extracted from registry operations are binary variables (i.e., check presence or not of registry entry for specific key), which is not effective in detecting new ransomware variants. To better distinguish ransomware from benign applications, we introduce a new monitoring model based on combined entropy and file signature. High-entropy operations during ransomware attacks are helpful in detecting anomalous behaviour. However, sometimes, files are encrypted by users for legitimate security purposes. In this case, current detection models based on file entropy calculation generate false

[1] Not to be confused with a malware signature.

positives, identifying non malicious operations as malicious. Although file entropy and registry key operations have been considered in one way or another in the literature, there has not been a systematic focus on how to utilize these features to improve ransomware detection accuracy and novel ransomware detection. The remainder of the paper is structured as follows: Section 2 reviews and discusses current ransomware detection approaches. In Section 3, a new public benchmark ransomware detection dataset is introduced. Section 4 presents our proposed detection architecture and model. Section 5 presents our proposed feature model. Section 6 presents the experimental evaluation of the proposed approach and discusses the results. Finally, Section 7 makes some concluding remarks and summarises our future work.

2. Related work

Ransomware detection approaches can be classified as being set-up behaviour based, file system behaviour based, network traffic based, and analysis based. In this section, we review and provide insights into existing ransomware detection techniques by discussing sample works under the aforementioned categories.

2.1. Set-up behaviour-based detection

The changes that happen to a system during a ransomware set-up, along with the large number of changes to the file system that occur when the ransomware performs its malicious actions, can be useful for ransomware detection. For example, Nieuwenhuizen in [29] presented a ransomware detection scheme by using a behaviour analysis in conjunction with machine learning. Although the exact features were not divulged, their constructed feature set combined the features extracted from the behaviour of a malicious set-up, such as payload persistence, anti-system restore, stealth techniques, environment mapping, network traffic and privilege elevation. In addition to the behavioural features associated with data transformation behaviour, such as mass file encryption, they used the support vector machine (SVM) algorithm as the classification scheme. Their model was trained on about 300 different ransomware families. In addition, they collected a controlled benign dataset that comprised examples similar to ransomware, such as file image compression and bulk file compression, to minimise false positives. No performance data of the approach were provided, and the authors only focused on explaining how a reduced dimensionality of their feature space lead to well-defined clusters of benign and ransomware samples. In addition, because no detection experiments were conducted, the ability to detect new ransomware has not been validated.

Sgandurra et al. presented a machine learning approach called EldeRan for analysing and detecting ransomware by monitoring the different actions performed by applications during ransomware set-up or installation [35]. The authors dynamically analysed the ransomware in a sandbox environment and collected several features, such as API calls, registry operations, file system operations, operations per file extension, directory operations, and the strings embedded in the ransomware binary. The mutual information criterion method was used for feature selection while using a regularised logistic regression for classification. In their work, the authors used a dataset consisting of 582 ransomware samples and 942 good applications. The achieved detection rate by EldeRan was 96.3%, and the false positive rate was 1.6%. In their experiments for detecting new ransomware, the weighted average detection rate was 93.3%, when using 100 features and 87.1% when using 400 features. In this work, all the experiments

were conducted offline on the features extracted from the data generated by running ransomware samples in a sandbox environment. No module has been implemented to conduct the online detection of ransomware attacks. In addition, the ransomware samples were executed in an environment where no other benign applications were running. This can be a possible reason behind the low false positive rate.

2.2. File system behaviour-based detection

The file system behaviour-based techniques detect ransomware based on massive file system transformations, such as systems calls triggered by ransomware activity on the infected host. For instance, Kharraz et al. in [16] presented a study of ransomware attacks by analysing various samples of ransomware belonging to different ransomware families that emerged between 2006 and 2014; they concluded that it is possible to design pratical ransomware defense techniques by monitoring file systems' activities, such as I/O requests and changes in the files' meta data. For example, a classifier can be used to detect the abnormal activities of the file system by training it on normal and malicious MFT entries. Moreover, API functions are used by a large number of ransomware families to lock the infected machine. Therefore, a classifier can be modelled and trained to detect a suspicious sequence of API calls. In the work in [16], the focus was on analysing a group of ransomware families, and based on the analysis, the authors provided several ideas that can be used for building ransomware classifiers for detecting ransomware; however, the work did not involve any ransomware detection experiments, and no detection approach was implemented.

Scaife et al. presented a detection system, CryptoDrop, that tracks changes in the victim's data and then alerts the user [33]. The authors used three primary criteria to detect malicious file changes: file type modifications, degree of similarity and entropy. Two other secondary indicators were used in addition to the primary indicators: the deletion of a large number of files and file type funnelling. By tracking these indicators, a user is alerted as soon as a threshold value has been reached. A Windows kernel driver was instrumented by the detector, which allowed for intercepting system calls between processes and file system. CryptoDrop was tested against 492 ransomware samples belonging to 14 families and achieved a 100% detection rate with only a few files lost. However, CryptoDrop can not distinguish whether a ransomware or a user is encrypting the files. In addition, CryptoDrop detected benign applications that generate high entropy and read large number of files, such as 7-zip, as ransomware.

Kharraz et al. presented a dynamic analysis system, UNVEIL, that can detect ransomware attacks [15]. The designed system creates an artificial execution environment and monitors the processes interacting with a file system to characterise cryptographic ransomware behaviour. Furthermore, the system detects a locked desktop by tracking changes made to the desktop that indicates ransomware behaviour; this is done by using the dissimilarity scores of screenshots taken before, during and after ransomware attacks. The system was implemented as a Windows kernel driver that provides monitoring capabilities for the file system. The evaluation of UNVEIL showed good performance results because it detected 13,637 ransomware samples with zero false positives. However, a weakness of this work is that the proposed system detects ransomware after it starts encrypting files, so the files encrypted before detection cannot be recovered. In addition, the designed execution environment can be fingerprinted by attackers and then evaded. UNVEIL is not designed to detect ransomware that does partial encryption of files or that shuffles the files' content. In addition, UNVEIL runs in the kernel and aims to detect user-level ransomware; hence, any ransomware that can run at the kernel level may affect the functionality of UNVEIL because it uses hooks to monitor the file system.

Soto et al. presented a signature-less method to detect malicious behaviour before the ransomware payload can infect the machine [38]. The authors used a dataset of labelled exploits and benign behaviour to train a machine learning classifier using the random forest algorithm. The dataset consisted of 386 labelled exploit examples from Contagio, the CRIME database, ransomware samples with some call back, and file-system-level indicators and anonymised bluecoat traffic. The machine learning detection model is coupled with the automated generation of group policy objects (GPOs), and in this way, the authors demonstrated an automated way to take action and create a policy based on the observed indicators of compromise (IOCs) that were detected in a zero-day exploit pattern. Currently, no more information or results are available about this work; only the presentation slides are available online.

Mbol et al. proposed an approach to detect files being affected by encryption because of ransomware (e.g., TorrentLocker) by considering the ransomware's inherent behaviour, where a huge amount of files are transformed very quickly [23]. The presented method works on a potential TorrentLocker by detecting processes with abnormal behaviour that open a large amount of files and checking if there is a difference between the structure of the input and output streams. Their proposed work focused on JPEG files. They used the Kullback-Leibler divergence to distinguish between 2,000 encrypted files and 2,000 normal JPEG files. The detection is based on a specified threshold, and the file is identified as an encrypted file when the Kullback-Leibler divergence of the input stream is below this predefined threshold. JPEG files were distinguished from encrypted files at a detection rate of 99.95%. The focus of this work was on JPEG files only, and the proof of concept has been done offline. No implementation of the approach was provided that performs online monitoring of the processes running on a computer system.

Chen et al. proposed a ransomware dynamic detection model using data-mining techniques that monitor the software's dynamic behaviour and then generate API call flow graphs (CFG) [5]. The graphs are transferred into feature space, and then, the most informative features are selected. To classify the data, four different classification techniques were studied, including support vector machine (SVM), simple logistic (SL), random forest (RF) and naive Bayes (NB). The authors used a dataset of 83 ransomware samples and 85 benign samples. The SL classifier outperformed the other classifiers with a 98.2% accuracy and 97.6% detection rate, with the lowest false positive rate being 1.2%. In addition, all the experiments were performed offline using Weka.

Continella et al. proposed an approach, ShieldFS, for detecting ransomware-like behaviours and retrieving the affected users' data [6]. ShieldFS monitors low-level file system activities and extracts a set of features, including the amount of folder listing, number of files reads, amount of files written, amount of files renamed, amount of files accessed and write entropy. Detecting the ransomware is based on comparing the collected features with that of benign applications. ShieldFS keeps a backup area of the files modified in a storage that is protected by allowing only read operations and preventing any I/O modification requests from the user space. However, the implemented approach only detects AES encryption, and the proposed approach may fail to detect ransomware variants that do not perform massive encryption of files at once and encrypt instead a small number of files per time slot. Moreover, a ransomware could evade detection by injecting a number of normal processes and letting each process performs a small part of the encryption activity. Another limitation of ShieldFS is the possibility for a sophisticated ransomware to prevent the ShieldFS (software) service from starting at system boot.

Mehnaz et al. introduced a detection mechanism, RWGuard, for detecting crypto ransomware [25]. The detection mechanism employs three modules: decoy files monitor, process monitor, and file change monitor. A number of decoy files are deployed, and by monitoring these decoy files, a ransomware process that writes on them can be identified. The running processes' I/O request packets (IRPs) are

monitored by the process monitor, while the changes performed on the files are detected by the file change monitor. To distinguish ransomware encryption from benign changes on files, a classification model that hooks relevant CryptoAPI routines is used. The same module is used to restore encrypted files; however, it can recover the files encrypted by only three ransomware families and cannot restore files when the ransomware employs custom cryptographic libraries. As a weakness, it is possible for ransomware to recognise the decoy files by monitoring file activities in the file system and determining which ones were not modified by the end users. In addition, code obfuscation makes the identification of cryptographic primitives in the ransomware binary difficult [25]. Another limitation of this work is that the files and processes monitors can miss some of the malicious activities because of the time lag between logging IRP calls and file activities and checking them through the related modules.

Some studies have focused on Android ransomware detection. Song et al. proposed a ransomware prevention technique on the Android platform to detect and block Android ransomware with anomalous I/O and CPU activity levels [37]. The approach was proposed as being able to reduce the damage caused by ransomware attacks and to detect modified or new versions of ransomware. It detects ransomware based on monitoring processor status information and file I/O events that occur when the ransomware accesses files. In addition, the approach depends on information stored in the DB, enabling users to specify which files must be shielded from ransomware attacks. The proposed method can be implemented in the Kernel and Android framework sources. However, CPU and I/O operations monitoring can produce significant system overhead, as noted when the proposed technique was used for evaluation. Furthermore, the proposed approach can be evaded using mimicry attacks [39] because of its dependency on the abnormal behaviours of the CPU and I/O.

2.3. Network traffic-based detection

The network-traffic-based approaches rely on the characteristics of network traffic during communication with the C&C server to exchange keys or information, in addition to communication with other agents, such as kill switches.

Ahmadian et al. proposed a detector for tracking high survivable ransomware (HSR), along with the detection of ransomware that exchanges a key for its operation [1]. The work was based on the idea that after infection, ransomware tries to contact an active C&C server to receive a public key by joining domains created by a domain generation algorithm (DGA). In the first version of their framework, the authors designed a connection monitor to detect DNS requests generated by DGAs. Markov chains were used for model training. With this approach, all HSRs that make DNS requests to domains that are generated by DGAs will be detected. In their extended framework, the authors implemented a connection monitor (CM) and connection breaking (CB) that check all the traffic of executables, enabling users to accept the connection or to ignore it.

Cabaj et al. presented a ransomware detection approach based on software-defined networking (SDN) [4]. In their work, the authors focused on crypto ransomware that utilises asymmetric ciphers. They analysed the behaviour of two types of ransomware: cryptoWall and locky. The characteristic features were then extracted from the outgoing HTTP messages exchanged by the compromised machine during ransomware execution. For each sample, they extracted a feature vector consisting of the content sizes from the POST requests generated by the analysed sample and then fine-tuned it. The distance between the constructed vector and the median point for a given type of ransomware was computed, and a decision was made based on a threshold distance limit. They obtained detection rates and false positive rates ranging from 97–98%, and 4–5%, respectively.

Cusack et al. proposed a detection approach that processes the network traffic between the compromised machine and the C&C server to identify malicious connections and prevent the delivery of the encryption key [7]. The approach is based on building five-tuple flows and then extracting the flow features used to train the detection model, which is done by using a random forest classifier. Two main types of flow features are extracted: direction-independent, including the duration, interarrival times, number of packets in the flow and their total size, and direction-dependent features, including burst lengths, the ratio of outbound to inbound packets and vice versa. The authors were able to achieve an 87% detection accuracy rate and 10% false positive rate.

Hasan and Rahman [11] proposed a framework called RansHunt that combines static and dynamic analysis to detect ransomware. The proposed model was evaluated using a total of 1,283 different binaries which included 360 ransomware binaries of 21 different families and 923 benign binaries, achieving 97.1% accuracy. The authors introduced new network related features in the dynamic analysis, which did not contribute much to improve the detection rate. Also, the features used for the dynamic analysis were almost similar to ElderRan's features. The model is ineffective for new variants of ransomware.

2.4. Analysis-based detection

Analysis-based detection techniques use information gathered and then conduct either a static or dynamic analysis or by combining both of these methods. For example, Gharib et al. presented a two-layer real-time detection framework for an Android platform and called their method DNA-Droid [10]. The framework starts by statically analysing a sample, and if detected as suspicious, it continues by dynamically analyzing the sample via monitoring and profiling the run-time behaviour in a simulated environment. DNA-Droid will terminate the process or programme under analysis if its profile can be matched to a similar group of malicious profiles. The static module evaluates the APK files by parsing the APK files and then extracting strings, detecting and counting the number of logos and nude images, and then extracting the list of permissions and API methods. Natural language features are extracted, and machine learning tasks are applied in the static module. The dynamic module differentiates between benign and malicious samples by analysing the API call sequences. However, a sophisticated ransomware can use code obfuscation in DNA-Droid to escape the static filter and shield itself from being dynamically analysed.

Another analysis-based detection work that explored Android ransomware detection can be found in [21]. In this work, the authors presented a system, R-PackDroid, which uses system API packages to classify an Android application into one of three classes: ransomware, malware or trusted. The authors identified the different API packages by inspecting the Android application, and based on this, they constructed a vector of the number of occurrences of the different system API packages. This feature vector can be passed to a statistical classifier that yields the application label. However, the proposed system relies on a static analysis, and this exposes some shortcomings, such as the programme not being able to analyse applications in which the code is dynamically linked or loaded during run-time or for fully encrypted classes. In addition, using a static analysis is not proper for early detection.

Andronio et al. introduced HelDroid, a tool for recognising Android scareware and ransomware, including unknown ransomware samples [2]. The proposed tool analyses Android applications statically and dynamically by looking for flows of function calls, which are indicators of suspicious behaviours, such as file encryption or device locking. The tool employs a text classifier and lightweight emulation to detect locking techniques. In addition, the tool detects file-encrypting flows by applying taint tracking. HelDroid starts by parsing code and resource files and then analyzing strings. If it fails to detect any

threatening strings, then it executes the sample in a sandbox environment, captures the network traffic, decodes the protocols working at the application layer, and finally extracts strings. The system uses a text classification model trained using sample ominous sentences, imitating those appearing in the ransomware variants. The evaluation of the system achieved accuracy over 97% with a dataset consisting of 650 ransomware and about 81,000 benign samples. However, the detection of threatening pharses is not much useful as by the time the user gets a ransom note on the screen the data is already encrypted. Also, detecting encrypted files is limited to Android primitive API and can be easily evaded using a custom code for encrypting the files.

Yang et al. presented a theoretical solution based on static and dynamic analyses [40]. The proposed design consists of two layers. In the first layer, the solution employs a static analysis based on features matching. These features include the following: API-invoking sequence, permissions, resources and APK structure. In the second layer, the model is based on dynamic behaviour monitoring of the running Android application. The four behaviours collected during the dynamic analysis include the following: sensitive paths and data flow, malicious malware access, malicious charges and bypassing Android permission (getting administration privileges). The work presented was only a theoretical design and did not present any implementation, experiments or analysis.

Mercaldo et al. developed a detector that uses formal methods to detect ransomware on Android smart phones [26]. The model also identifies the characteristic instructions in Android ransomware that implement the malicious payload of ransomware and that are related to the infection and payload's activation. The proposed model first parses the Java Bytecode of the application and generates the formal models of the system. Then, in the second sub process, the behaviour of the ransomware is defined by means of temporal logic properties. Finally, the ransomware family is recognised by invoking a model checker that verifies the logic properties constructed in the second sub process.

Shaukat and Ribeiro [36] conducted a comprehensive analysis of 574 ransomware samples from 12 different ransomware families. They also presented a layered defense system named RansomWall for the detection and mitigation of cryptographic ransomware. The proposed solution follows an hybrid method that combines dynamic and static analysis to identify and characterize the behavior of ransomware. The initial layer of the system checks for the malicious behavior and the next layer initiates a backup of user files changed by malicious processes detected in the initial layer. The authors explored the following supervised machine learning algorithms: Logistic regression, Support Vector Machine, Artificial Neural Network, Random Forest and Gradient Tree boosting. Among all tested algorithms, the approach achieved the best detection rate of 98.25% and nearly zero false-positive rate with gradient tree boosting algorithm. Although the model achieved high detection rate, ransomware detection is highly dependent on the modification of deployed honey files and honey directories. For early detection of ransomware by the first layer, it must encrypt the honey files. Ransomware that do not encrypt or modify the honey files will be able to evade the early detection phase of the system.

In addition to the studies that have focused on the detection of ransomware, some studies have analysed the possibility of building defense techniques that allow recovering from ransomware infections without considering the detection of ransomware attacks [13,18,19,27]. Although the works proposed in [18,19,27] presented software-based solutions for the recovery of the files encrypted by ransomware, the work in [13] presented a hardware-based solution to defend against ransomware attacks by providing a firmware-level recovery mechanism. The hardware-based defense approach has the benefit of being robust even against kernel-level ransomware [17].

Table 1 summarizes and compares our work with the previous works which are most relevant in terms of feature selection and machine learning detection. Our approach is unique in its use of grouped

Table 1
Summary of most relevant works

Name and reference	Number of ransomware samples	Number of benign samples	Number of families	Number of features	Groups of features	Classifiers	DR (%)	FPR (%)
EldeRan [35]	582	942	11	400	API calls, registry key operations, file-system operations, file operations per file extension, directory operations, dropped files, embedded binary strings	SVM, naive Bayes	96.3	1.61
API call flow graph [5]	83	85	4	3600	API calls presented as call flow graph	SVM, naive Bayes, random forest, logistic regression	97.6	1.2
RansHunt [11]	360	460	21	67	Dynamic features: API calls, registry key operations, file operations static features: function length frequency, printable string information	Random forest, naive Bayes	97.1	<3
RansomWall [36]	574	442	12	15 categories	API calls, file system operations	Logistic regression SVM, neural network, random forest, gradient tree boosting	98.2	≈0
HellDroid [2]	650	81000	Not specified	120	API calls, lightweight behavioral patterns, heuristics, assets and source code stats	J48 random forest, stochastic gradient descent, decision tables, rule learners	84.6	≈0

Table 1
(Continued)

Name and reference	Number of ransomware samples	Number of benign samples	Number of families	Number of features	Groups of features	Classifiers	DR (%)	FPR (%)
DNA-Droid [10]	1928	2500	8	Not specified	API call sequences, text classification, image classification	Naive Bayes, SVM, random forest, AdaBoost, deep neural networks	97.5	<1.5
Multilayer [current paper]	666	103	20	400	API calls, grouped registry key operations, Windows DLLs, directory, enumerations, embedded strings	Logistic regression, random forest, SVM	100	1.41

registry keys, and it achieves some of the best performances. Also, to the best of our knowledge none of the abovementioned work have published openly their datasets.

3. Ransomware detection dataset

In this section, we present a new benchmark ransomware detection dataset, that can be used to design and evaluate ransomware detection models. We did not find any publicly available ransomware dataset. Various authors have built their own datasets to establish their work. Building a benchmark dataset of ransomware attacks would be a valuable addition to the field of ransomware detection. Towards this goal, we have built a standard ransomware detection dataset. The construction of the dataset was performed according to the scientific guidelines and best practices suggested in [31]. This section highlights the collected dataset and its construction process.

Most ransomware samples were obtained from Virustotal[2] under academic license, and several samples were obtained from anti malware companies. To build a representative ransomware dataset, the variety of the ransomware families and variants is more important than the quantity of the collected samples. Hence, we tried to collect samples from as many ransomware families as possible. We focused on gathering both old and recent ransomware samples that represent most of the popular ransomware families and variants that have emerged in the wild. At first, we ended up with a collection of about 4,000 ransomware binaries. To decide the family name of each sample before running the samples and starting the ransomware behaviour data collection process, we followed the approach used in [16] to decide the family name of the samples based on the Virustotal scan results of each ransomware binary. The popular name used by most of the antivirus (AV) engines is assigned as the family name of the

[2] http://virustotal.com

Table 2
Ransomware families and number of samples involved in the dataset

Family	Samp	%	Family	Samp	%
TeslaCrypt	348	52.3	Mole	4	0.597
Cerber	122	18.23	Satan	2	0.299
Locky	129	19.28	CTBLocker	2	0.299
CryptoShield	4	0.597	Win32.Blocker	18	2.69
Unlock26	3	0.448	Spora	5	0.747
WannaCry	1	0.149	Jaff	3	0.448
CryptoMix	2	0.299	Zeta	2	0.149
Sage	5	0.747	Striked	1	0.149
Petya	2	0.299	GlobeImposter	4	0.598
Crysis	8	1.196	Xorist	2	0.299
Flawed	1	0.149			
Total			**666**		
Benign			**103**		

sample. After executing the samples, we noticed that several samples were wrongly classified based on their Virustotal results. Because of this, we followed another approach for assigning family names, one that is more accurate though time-consuming. The family name of each sample was confirmed based on either the ransom note generated during the dynamic analysis of the samples or from the extensions of the encrypted files. By doing this, each sample was assigned the correct family label based on the results from running the sample. To the best of our knowledge, there is no other work that has followed this approach. Table 2 shows the families included in the dataset, the total count of samples per family, and the percentage of samples per family compared with the total count of samples in the dataset.

All collected samples were analysed using the Cuckoo sandbox [32]. In our setup, we used Windows 7 (64 bit) to run the ransomware samples and collect the behaviour data. To build a realistic analysis environment similar to a real-use environment, we collected a group of files (of size 1.51 GB) representing the different file types targeted by the various ransomware families [9,28]. The different file types, the number of files per type and the percentage of files per file type compared with the total number of files are depicted in Table 3. These files were distributed over the entire file system by following the studies that examined the distributions of files within directories [12] and across the file system [8]. In addition, common applications, such as MS Office, Adobe Reader, Chrome Browser, and so on, were installed on the analysis machine. The virtual networking between the host running the Cuckoo software and the analysis machine (guest) uses a host-only adapter networking layout. Full Internet access was provided for the analysis machine to enable required communications with other agents involved in the attack, such as C&C servers, kill switches and so forth, hence allowing the analysed ransomware samples to run successfully. The outbound network traffic to the LAN network was restricted to protect against the ransomware trying to spread to other machines.

To capture the behaviour and traces of the targeted families and variants, each sample was allowed to run for the time needed for the attack to happen successfully until its completion, with a maximum analysis time of 60 minutes.[3] Because some samples will never carry out an attack, as explained later in this section, we kept monitoring the analysis logs generated by the sandbox to see if the sample was doing its work (e.g., encrypting files) and also checking if any ransom note was created in case there

[3]We found that most of the samples finish their attacks in less than 60 minutes.

Table 3
Files distribution in the analysis machines

Type	Number	%	Type	Number	%
pdf	412	18.72	ps	23	1.04
html	462	21	mov	24	1.09
gif	38	1.72	rtf	5	0.22
2 jpg	188	8.54	swf	40	1.81
xls	75	3.4	txt	233	10.59
csv	17	0.77	log	6	0.27
xlsx	32	1.45	avi	24	1.09
ppt	117	5.31	gz	11	0.5
pptx	132	6	zip	8	0.36
doc	170	7.7	xml	18	0.81
docx	97	4.4	unk	6	0.27
mp4	2	0.09	wp	2	0.09
m4a	24	1.09	dbase3	2	0.09
mp3	22	1	png	2	0.09
java	1	0.04	other	7	0.31
Total			**2,200**		

was not much activity. The analysis of the sample was cancelled after 30 minutes if it did not start the attack within this time frame. During each run, different types of data were collected. The following is a summary of the collected data:[4]

(1) Network traffic dump generated during the analysis of each sample.
(2) List of operations performed on the file system.
(3) Information of the system calls initiated during the sample analysis with the arguments passed and return values.
(4) Information about the processes initiated during the sample analysis.
(5) Information about the different operations on the file system.
(6) Information about the different operations on the Windows registry.
(7) Full memory dump of the analysis machine.
(8) List of all the files targeted by the sample being analysed.

The analysis machine was reverted to a clean state after each experiment (run) to remove the possibility of interference between experiments. As mentioned above, we collected around 4,000 ransomware samples to be analysed and collected their behaviour data. However, not all of the samples ran successfully because of several reasons. For example, some of the samples were corrupted executable files, some were not able to connect with the C&C servers because those servers were no longer reachable and down, and another group of samples detected the analysis machine and refused to run to avoid being analysed. Therefore, a group of 666 ransomware samples belonging to 20 different families were run successfully and did their attacks to completion. Our dataset consists of the behaviour data for those 666 various ransomware samples of 20 different families, making (to our knowledge) our dataset the most comprehensive dataset publicly available to date. In addition to the data collected for the ransomware samples, we collected behaviour data for a collection of benign applications as well. A set of 103 benign

[4]The dataset can be obtained at https://www.uvic.ca/engineering/ece/isot/datasets/index.php.

applications representing the most popular software applications used by Window users were downloaded from a software repository website.[5] To avoid suspicious applications, only software applications hosted by trustworthy sources were used. The collected benign applications included applications that behave like ransomware, such as compression, encryption, disk cleaning, and secure deletion. The total size of the dataset is 428 GB.

Upon observation of our ransomware dataset of 666 ransomware samples from 20 different families, one can notice that a major portion of our dataset is shared by three ransomware families making our dataset imbalanced. However, after close observation of execution logs of ransomware samples, we noticed that although ransomware samples were from different families, they broadly shared the same characteristics in terms of execution operations such as windows API calls, targeting registry key areas and file system operations. Machine learning classification accuracy heavily depends on the features the model is trained on. The more features are generalized, more there are chances to detect a variant of similar kind. Hence, our machine learning model is trained on generalized behavior set which is collected from analysis logs and common across all ransomware families. The reason for following this method is to make a generalized predictive model that can detect the previously seen ransomware with high classification accuracy and can detect new ransomware families without affecting the accuracy of our model.

4. Proposed detection approach and architecture

We propose a ransomware detection system that consists of two layers of defense:

(1) The first layer uses a strengthened feature model and machine learning classification for detection.
(2) The second layer tracks and detects the ransomware samples that escape the previous layer by using both the file entropy and file signature.

In this section, we present the combined file entropy/file signature model and then introduce the architecture of our proposed detection system.

4.1. Combined file entropy and file signature model

Entropy in digital systems is a measure of the randomness in a file. A file is compressed by replacing large patterns of bits with shorter patterns of those bits. Compressed and encrypted files have a high degree of randomness. Shannon provided a formula to calculate the theoretical maximum amount for digital file compression. As per Shannon, the maximum entropy occurs when all bytes are distributed equally across the file. The entropy value is a calculation of the predictability of the next character in the file based on previous characters. It is measured on a scale of 1 to 8, where encrypted and compressed files have a high value, and standard text files have a low value. The Shannon entropy formula allows for calculating the average minimum number of bits required to encode the string of symbols based on the frequency of symbols and the alphabet size. The Shannon entropy H is given by the following,

$$H(S) = -\sum_{i} p_i \log(p_i) \tag{1}$$

where p_i is the probability of character i appearing in the alphabet stream.

[5] http://software.informer.com/software

To calculate the entropy of a file, we calculate the frequency of all ASCII characters, which include standard ASCII characters (0–127) and extended ASCII characters (128–255), in a given file, and then use this as the probability in the Shannon entropy formula.

File signatures, or magic numbers, are the first few bytes in a file that are different for each file type. These bytes are used by the operating system to recognise the files without depending on the file extension. For instance, JPEG image files begin with FFD8 and end with FFD9. A file signature is not visible to users, but by using a hex editor, it can be seen. Changing or corrupting these bytes makes a file useless because they are essential for a file to be opened.

Some legitimate files, such as MS Office, 7-zip and pdf files, are also highly compressed and have a high entropy value. Therefore, file entropy calculation alone does not help to differentiate between user-triggered encryption and ransomware-triggered encryption. However, most of the file types have a file header and/or file footer corresponding to their signature or magic numbers, and through which the actual format of the file can be identified [14]. There are two strong characteristics of ransomware, as follows:

(1) Ransomware usually encrypts the whole file, meaning it also clobbers the file signature of the data files.
(2) Ransomware generally applies a decent encryption algorithm to the files. As a result of this, the file entropy will be very high.

Therefore, monitoring the combination of file signature and file entropy can effectively help identify ransomware-triggered encryption. To study the impact of ransomware infection on file entropy and file signature, we deployed some user files in our sandbox environment. After the successful execution of ransomware and benign binaries in a sandbox environment, we analysed a total number of 157,187 user files from infected and normal Windows machines. The dataset was a combination of regular user files (i.e., .docx, .pdf, .jpeg, etc.) and ransomware-encrypted files. Most of the user files after ransomware execution were encrypted. On the other hand, after benign binary execution, the files were unmodified. We also noticed that most of the ransomware-encrypted files were missing file signatures. We then calculated the Shannon entropy of all files and filtered out the files where the file signatures were missing. The steps to calculate the entropy are outlined in Algorithm 1.

We grouped the filtered files by ransomware families and calculated the average entropy of files per family. On the one hand, for all ransomware families, the average entropy values were above 7. On the other hand, after executing the benign binaries and because the files remain unchanged, the average entropy of the files was around 4.5. Figure 1 shows the average entropy for the different ransomware families. As we can see from Fig. 1, the filtered files of ransomware-infected machines have very high average entropy values compared with the files in uninfected machines.

4.2. Proposed multilayer detection architecture

Our multilayer detection model involves two separate detectors that work in tandem. The first detector consists of a pre trained machine learning (ML) classifier, while the second detector makes its decision based on a file entropy threshold and file signature database. Figure 2 presents the architecture designs for the two detectors in separate tracks.

Although (as shown later in the experimental evaluation) our machine learning model achieves a considerably high detection rate, there might be a case when a new variant of the ransomware evades detection. Our system continuously monitors the high-entropy operations of unknown file signatures.

Algorithm 1 File entropy calculation process

Input: File {A file from the file system}
Output: Output {Binary decision whether the encryption is by a ransomware or a legitimate process}

Open(File)
FileSize = GetFileSizeInBytes(File)
threshold = 7
(float) entropy = 0
for C = 0 To 255 **do**
  $(float)\, p = \frac{number\ of\ occurrences\ of\ character\ C\ in\ File}{FileSize}$
  if (p > 0) **then**
    $entropy = entropy - p \times \log_2(p)$
  end if
end for
$entropy = -entropy$
if $entropy > threshold$ **then**
  if File signature = Missing **then**
    Output = Ransomware Encryption
  end if
else
  Output = Benign Behavior
end if
return Output

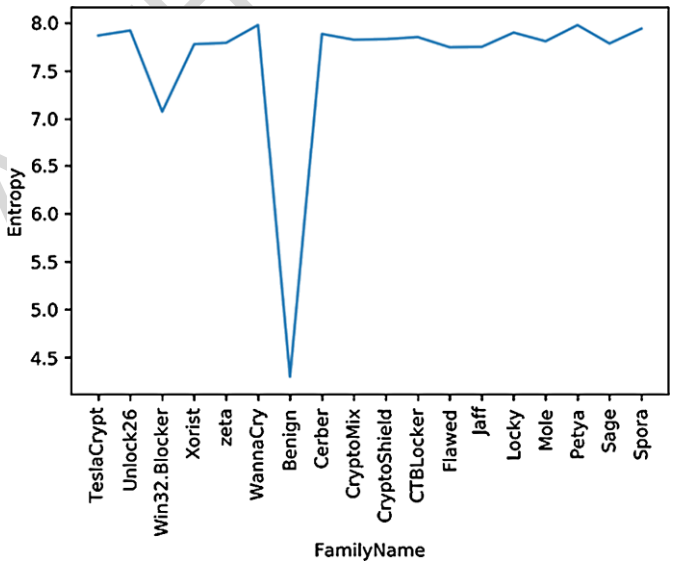

Fig. 1. Average entropy of the encrypted files per family.

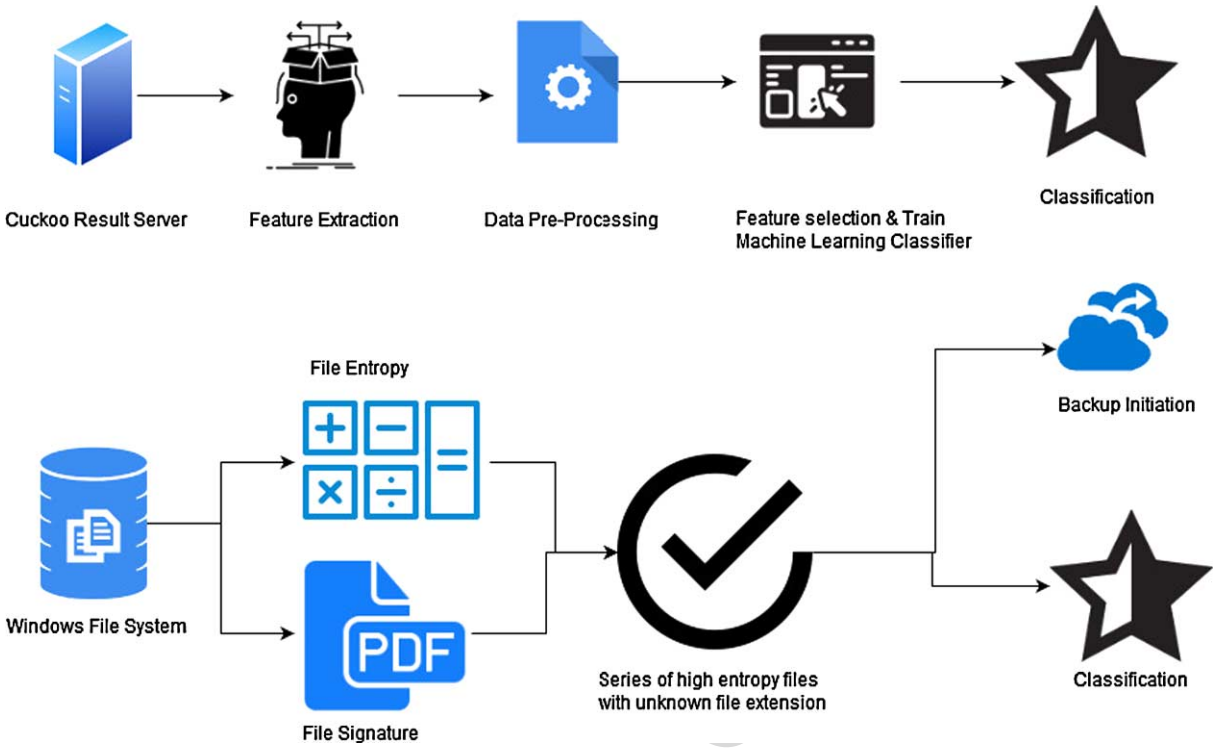

Fig. 2. ML and file entropy/signature detectors.

When a series of high-entropy operations with an unknown file signature takes place in the device, our system initiates immediately the backup of the user files and classifies the process as malicious. By utilising this system, user data can be saved with minimum file loss.

Figure 3 presents the system architecture showing the working of the two detectors in tandem.

When a sample binary is received, features are extracted and then classified using the first detector into either legitimate/normal or ransomware. In the case where a ransomware decision is made by the first detector, an automatic backup is triggered for the files being modified. At the same time, the second detector computes the file entropy for the corresponding files and compares them against a predefined threshold. If the entropy is larger or equal to the threshold, the detector then checks the file signatures. If the file signatures do not match the expected signatures, are unknown or are unreadable, then the initial ransomware classification (i.e., made by the ML classifier) is confirmed, and the modified files are automatically replaced by using their original version stored in the backup. Otherwise, if the file signatures match the expected ones, then the sample binary will be reclassified as normal (i.e., by reverting the initial ML decision). In the case where the initial classification for the sample binary by the ML model is normal, the second detector will apply the same process as above by computing and comparing the file entropy against the threshold and checking the file signatures. If the file entropy is lower, then the original decision of normal will be confirmed. Similarly, if the entropy is higher while the signatures do match, then the classification as normal will be confirmed as well. Otherwise, if the entropy is larger than the threshold and the signatures do not match, a warning will be generated, and the backup will be triggered automatically.

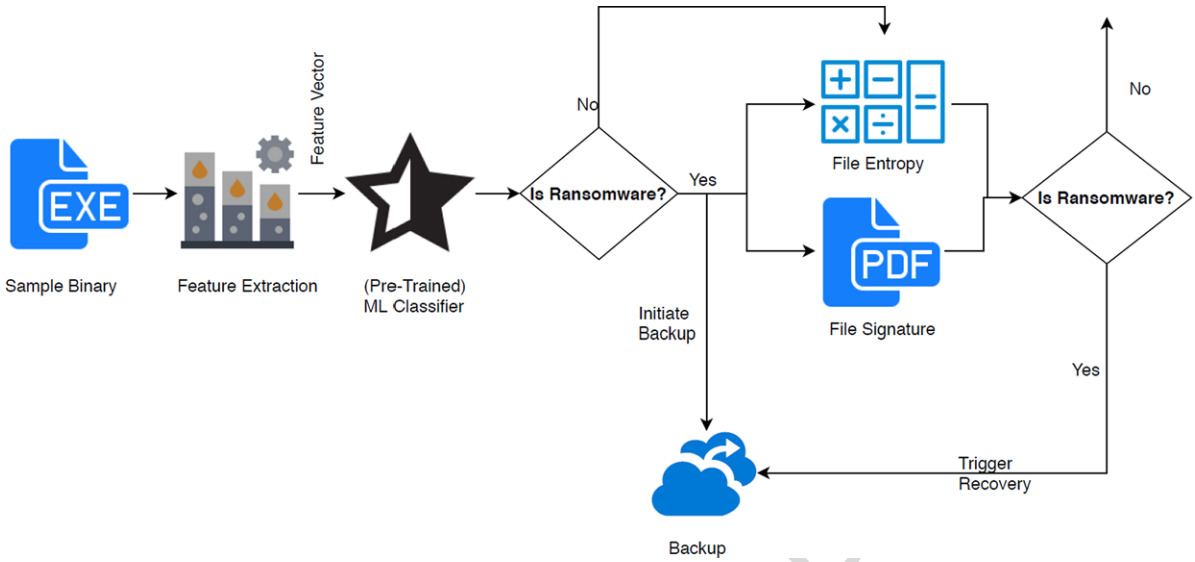

Fig. 3. Multilayer detection process.

A legitimate question about our model can be why keep the ML layer because all its decision will be challenged by the second detector, whether such a decision is ransomware or normal. A straightforward answer to that question is that the ML model provides early warning of possible ransomware and then triggers the backup immediately without delay. This way a malware can be detected with minimum file loss. In the case where the ML detector fails to detect the ransomware (i.e., in the case of a false negative), the second detector would then catch the corresponding sample and trigger the response accordingly. Of course, in the latter case the backup would be slightly delayed, and there would be more chance of loosing some files.

5. Feature model

Our proposed feature model consists of a new set of features based on grouped registry key operations, and additional features inspired from the literaure. In this section, we present the different features groups involved in our model.

5.1. Registry key operations

Whenever the user installs any software program, the initial configurations are stored as key-value pairs in the registry. When a user runs the software, the system components retrieve their run-time configuration from the registry database. Our sandbox analysis shows that the software executes four types of registry key operations to maintain the persistency of the configurations across the reboots. We collected a total number of 27,739 unique registry key operations from the collected cuckoo reports (benign and ransomware). Table 4 provides a breakdown of the collected registry operations.

Registry key operations can be unique per software. Two different software might not use the same registry key operations. To analyse the most impacted registry hives during the ransomware attack, we counted the total number of registry key operations done on each registry hive. There might be a case

Table 4
Registry key operations and their counts

Registry key operation	Count
Opened	5201
Deleted	199
Read	17693
Written	4646
Total count	27739

Table 5
Registry key hives and their counts

Registry key hives	Count
Opened	1211
Deleted	29
Read	1826
Written	931
Total count	3997

where one registry hive has multiple child registry hives and where only one of the child registry hives is impacted during ransomware execution. In this case, considering the child class and parent class of the registry hives both as features increases the chances of selecting repetitive data. To reduce the chances of training a model on repetitive data, we identified the registry key hives that are in linear correlation with each other. We calculated the Pearson correlation coefficient for each parent hive and its child hives. The Pearson correlation coefficient is a measure of the linear relation between two variables [41]. The value of the Pearson correlation coefficient lies between $+1$ and -1. A value of 0 indicates that there is no linear relation between two variables. Values of 1 and -1 indicate positive and negative linear relations, respectively. The Pearson correlation between two variables x and y is given by the following formula:

$$r_{x,y} = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \bar{x})^2}\sqrt{\sum_{i=1}^{n}(y_i - \bar{y})^2}} \tag{2}$$

where,

- n is the sample size
- x_i, y_i are the individual sample points indexed with i
- $\bar{x} = \frac{i}{n}\sum_{i=1}^{n} x_i$ (the sample mean); and analogously for $\bar{y}$

We identified the child hives having a correlation value of 1 with its parent hive and removed those hives from our dataset. The reason for truncating the dataset to the best features is to avoid training the model on repetitive data, which ultimately overfits the model. The final breakdown of the registry hives selected as features is shown in the Table 5.

5.2. API calls

From the previous work done on ransomware detection using dynamic analyses, we observed that most ransomware variants use standard Windows cryptographic APIs to encrypt the files. To study the

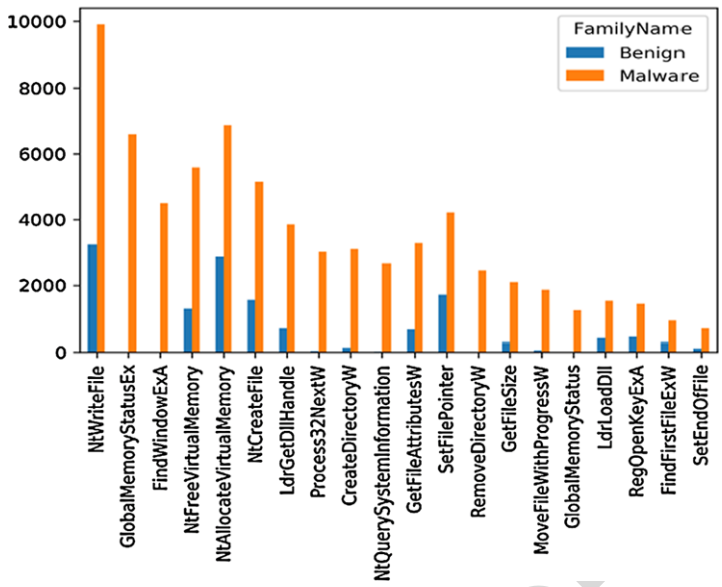

Fig. 4. API call frequency comparison.

importance of API calls in ransomware detection and how the requests generated by ransomware in the Windows operating system are different from those generated by benign applications, we extracted the frequency of each of the API calls initiated by ransomware and benign applications during their execution in the sandbox. We identified 286 API calls of interest from our dataset, including both benign and ransomware. The analysis of API calls revealed that API calls related to file system activities are heavily used in ransomware files compared with benign files, and certain API calls are only present in ransomware files. By examining the calls, some API calls are present in both benign and ransomware files, but their use frequency varies in both benign and ransomware applications. The comparison of API call frequency in benign and ransomware applications is depicted in Fig. 4. We also observed that not all ransomware families use the same API calls to achieve their goal.

On further examination of the calls, some API calls are present in both benign and ransomware files, but their use frequency varies in both benign and ransomware applications. For example, API *FindWindowExA* was heavily used by the Cerber ransomware family. CTBLocker and Sage ransomware families also used this API, but the frequency of usage was low compared to Cerber. On the other side, the usage frequency of this API in benign applications was very low, almost close to none.

5.3. Command-line operations

The command prompt is generally used to automate the tasks, troubleshoot operating system issues or perform administrative functions. Because most of the users use a graphical user interface for convenience, in the backend ransomware leverages a part of an operating system that computer users rarely come in contact with. The ransomware utilises this functionality to achieve goals such as delete the master boot record, delete windows shadow copies and so forth. We analysed and extracted in total 2,770 important command line operations from our benign and ransomware binary execution reports.

5.4. Windows DLLs

Windows APIs are implemented as a set of DLL files. In the backend, all Windows APIs use dynamic linking libraries. We extracted 404 common DLL files used by ransomware and benign applications from generated JSON reports.

5.5. Directories enumerated

During encryption process, ransomware goes through all directories or a specific set of directories of the file system to encrypt the files. During this period, the number of directory changes are very high because ransomware tries to encrypt as many files as possible. From collected reports, we identified 34,809 directory enumerations and converted them into numeric features to generate a count of directory enumerations in each binary execution.

5.6. Mutex

A mutex is a programme object generally used by malicious software as a locking mechanism to avoid simultaneous access to a resource in the system and to avoid infection of the same system more than once. To evade detection, some malware avoids the usage of common mutex names and dynamically generate mutexes at runtime. Ransomware also utilises this functionality to avoid infection of the system more than once. We extracted 603 important mutexes from the analysis of ransomware samples.

5.7. Embedded strings

String extraction from malware binaries can give a clue about the functionality of the programme. For example, if malware tries to resolve the domain name, it might be stored as a string in a binary. It can also give important information such as IP addresses, file names, registry key operations, etc. We extracted a total of 8,582 strings from ransomware and benign binaries.

5.8. Miscellaneous features

Ransomware also exhibits some common malware characteristics, such as gaining control of keyboard shortcuts, changes in the boot sequence, stealing information from browsers and so forth. We identified 105 such characteristics and used them as binary features. Some of the most important among these features are as follows:

- Packer entropy: To bypass the signature detection, malware authors use code obfuscation techniques. One of the purposes of packing malware is compressing and ciphering the real code. High-entropy values in packed executables indicate a random distribution of the bytes, a prevalent property in compressed and ciphered data. High packer entropy is one of the measures for the detection of packed executables.
- BCDEdit: BCDEdit is a command line tool available in Windows operating systems that is used to manage boot configuration data. Some of the ransomware variants use this command line option to change the boot menu options.
- Deletion of Windows shadow copy: Ransomware deletes all system backup files to make sure files cannot be recovered once encrypted.

Table 6 provides a breakdown of all the extracted features, their types and the number of features per category.

Table 6
Feature space breakdown

Feature name	Feature type	Number of features
API calls	Numeric	286
Registry key operations	Numeric	3,997
Command line operations	Binary	2,770
DLLs	Binary	404
Directories enumerated	Numeric	34,809
Mutex	Binary	603
Embedded strings	Binary	8,582
Miscellaneous	Binary	105

6. Experimental evaluation

In this section, we give a brief outline of the techniques used for data analysis and then present various experiments conducted to evaluate our proposed detection system.

6.1. Data analysis techniques

6.1.1. Data standardisation

Data standardisation, also known as data scaling, is a method used to normalise the range of features values. To achieve feature scaling, we used a min–max scaler to transform the data so that all have values between 0 and 1.

6.1.2. Feature selection

There are three different types of methods available for feature selection: filter methods, wrapper methods and embedded methods. Filter methods, which are also called univariant selection methods, apply statistical measures to all available features and assign scores to them according to their importance and the target class. The wrapper techniques, on the other hand, prepare different combinations of different features, and evaluate and compare them with other combinations. The recursive feature elimination method is an example of a wrapper method where unimportant features are discarded one by one recursively. Embedded methods learn the best features of the model while the model is being created. The most commonly used embedded feature selection methods are regularisation methods [3]. From the initial feature set, we wanted to identify the features that are the most important to classify the ransomware. This requirement makes filter methods the best option for us. Also, we regularise our models manually at the later stage to reduce overfitting. In this way, we used two feature selection methods: filter methods in the data preprocessing stage and embedded methods while training a machine learning model. An example of filter methods is the chi-square statistical test. We used Scikit-learn library SelectKbest with a chi-square test to select the relevant features. The calculation of the chi-square statistic is quite straightforward and done as follows:

$$x^2 = \sum \frac{(f_0 - f_e)^2}{f_e} \tag{3}$$

where f_0 is the observed frequency and f_e is the expected frequency if no relationship exists between the variables. As per the equation, the chi-square statistic is based on the difference between what is observed in the data and what would be expected if there is no relationship between the variables. A very small

chi-square value means observed data fits expected data very well. In this case, there is a relationship. A very large chi-square value means that the data do not fit well, so there is no relationship. All features are assigned a score after calculation of chi-square statistics between every feature variable and the target variable. We discarded features with low scores, and features with high scores were marked as important and selected to train a machine learning model.

6.1.3. Machine learning classification

In our dataset, malware data dominate benign data, which is not the case in the real world. In machine learning and data science, this scenario is called imbalanced class distribution. There is a chance that the predictive model developed using this dataset could be biased and inaccurate. Various data sampling techniques are available in data science to overcome these challenges. We evaluated our detection model using both the original dataset and the resampled dataset. We studied and compared three different machine learning models: linear SVM, logistic regression and random forest classifiers. We used python Scikit-learn libraries for training our data.

6.2. Machine learning in imbalanced dataset

We explored a chi-square feature selection range between 50 to 500 features. Once feature selection is applied to the model, the final step is to train a classifier for detecting ransomware. In our case, because of having a high number of features, linear classifiers are a good choice. We chose logistic regression, SVM, and random forest for training a machine learning model and applied 10-fold cross-validation on the trained models. Figure 5 depicts the accuracy of the classifiers when varying the number of features. In the case of SVM, the average accuracy of the model stayed around 94%, while for logistic regression and random forest, the accuracy varied from 94% to about 97%. It can be noted that for logistic regression and random forest classifiers, as the number of features increases, the accuracy also increases. However, with an increase in the number of features, the chance of overfitting also increases. We observed that with 400 features, we achieved the highest average accuracy in all models. Therefore, we used this number as the cut-off to select our features.

Table 7 shows the distribution of the 400 important features selected through the chi-square feature selection method. From the distribution, it can be observed that API calls and registry key operations

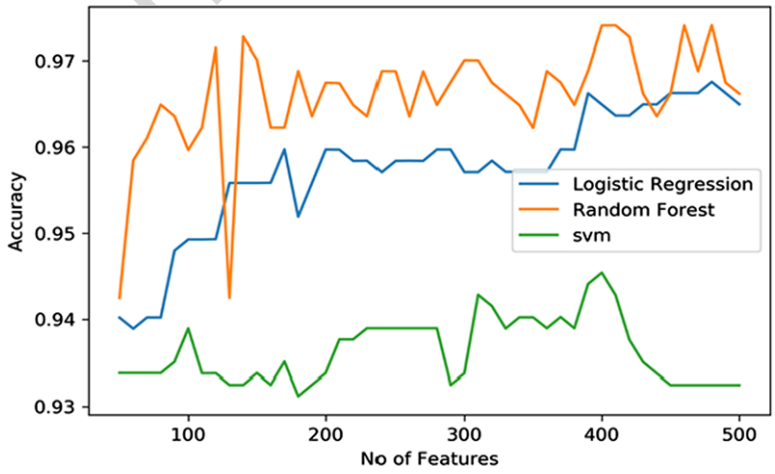

Fig. 5. Classification accuracy when varying the number of features.

Table 7
Top 400 features distribution

Feature type	Top 400
API calls	159
Registry operations	225
Command line operations	1
Directories enumerated	1
Miscellaneous	7
Mutex	5
Embedded strings	2

Table 8
Classification performance for the logistic regression

	Precision (%)	Recall (%)	f1-score (%)	Support
Benign	92	73	81	15
Ransomware	97	99	98	140
Micro average	97	97	97	155
Macro average	94	86	90	155
Weighted average	97	97	97	155

cover major portions of the selected features. However, the presence of other features also plays an important role in achieving a high detection rate.

From Fig. 5, we can conclude that our detection models achieved the highest accuracies while using random forest and logistic regression algorithms. Also, random forest and logistic regression algorithms are easy to train and execute compared with SVM. Compared with random forest, logistic regression is less prone to overfitting. However, the maximum likelihood estimation of posterior probability used in the logistic regression algorithm is also prone to overfit if a large number of features are used to train the classifier [35]. This problem can be addressed through model regularisation. We first trained a logistic regression model without any regularisation and checked the performance metrics. The training was done on 80% of the dataset, and the remaining 20% was kept aside for testing. Table 8 and Fig. 6 show the classification results and the confusion matrix, respectively, for the standard logistic regression classifier.

From Fig. 6, we can observe that the model has falsely predicted one ransomware and four benign samples. The classification results yield an accuracy of 96.7%. We also wanted to compare the achieved results with a regularised logistic regression classifier. Regularisation is a technique that allows finding a good bias-variance trade-off by tuning the complexity of the model. Regularisation does not improve classification performance on the dataset on which the algorithm initially learns. However, it can improve the prediction accuracy of new and unseen data. By adding the bias, we can remove overfitting; but adding too much bias also can result in underfitting. Hence, the concept behind regularisation is to introduce additional bias to penalise extreme parameter weights.

Note that in our system, we have two mechanisms to prevent overfitting: the feature selection with the chi-square and the regularisation of the logistic regression. Regularised logistic regression is competitive with SVM and easy to train when using different regularisation values. To apply regularization to our logistic regression term, we used C as a regularisation parameter, which is available in the Scikit-learn library. The regularisation parameter available in the Scikit-learn library is inverse to the regularisation parameter λ: $C = \frac{1}{\lambda}$.

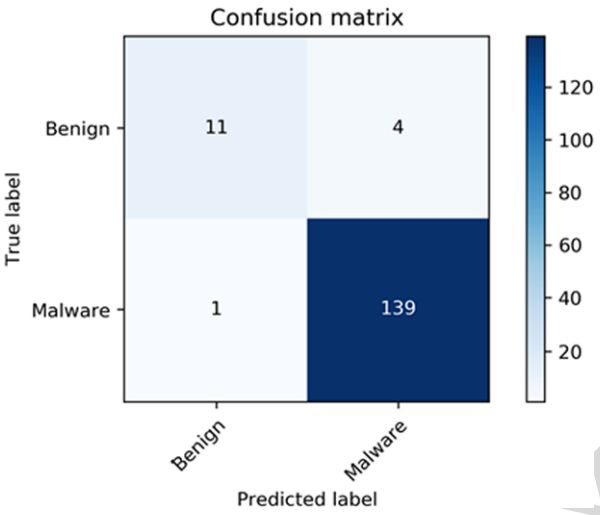

Fig. 6. Confusion matrix for the logistic regression classifier.

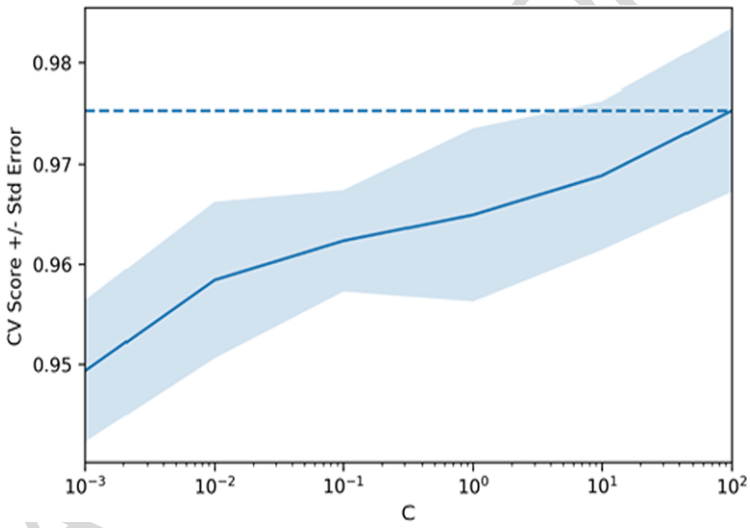

Fig. 7. Logistic regression accuracy for different values of the regularization parameter C.

Lambda (λ) controls the trade-off between allowing the model to increase its complexity as much as possible while trying to keep it simple. For example, if λ is very low or null, the model will have enough power to increase its complexity (overfit) by assigning high values to the weights for each parameter. On the other hand, if we increase the value of λ, the model will tend to underfit because it will become too simple. Parameter C will work conversely. For small values of C, the model will underfit the data, and for large values of C, the model will overfit the data. To make an appropriate selection of value for C, we ran our logistic regression model through 10-fold cross validation in the narrow range of C between 0.001 and 100. Figure 7 depicts the model accuracy when varying C. As shown in Fig. 7, with an increase in the value of C, the model accuracy also increases. From the graph, we can observe that with the value of $C = 100$, we obtain the highest accuracy at around 97.5%.

Table 9
Classification performance for the regularized logistic regression classifier

	Precision (%)	Recall (%)	f1-score (%)	Support
Benign	100	87	93	15
Ransomware	99	100	99	140
Micro average	99	99	99	155
Macro average	99	93	96	155
Weighted average	99	99	99	155

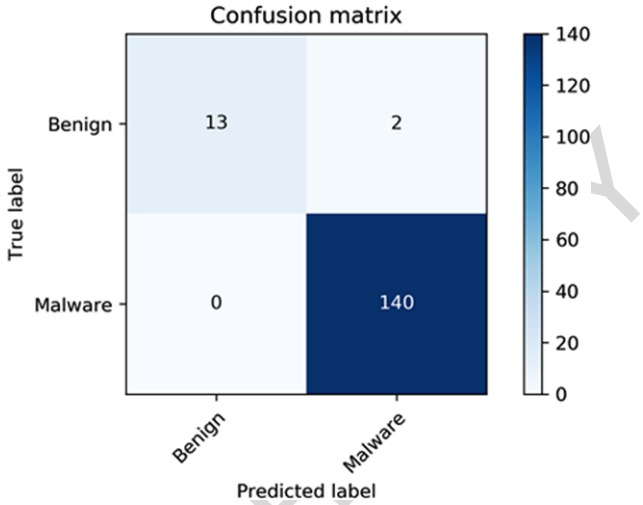

Fig. 8. Confusion matrix for the regularized logistic regression classifier.

We divided the dataset into an 80:20 ratio to evaluate the regularised logistic regression classifier. We trained a logistic regression model with the value of $C = 100$ using 80% of the data, and the remaining 20% data was used to test the trained classifier. Table 9 shows the evaluation results for the classifier. For ransomware, we achieved a recall of 100%, meaning we were able to detect all ransomware in the testing stage. From the confusion matrix shown in Fig. 8, we can also observe that our model achieved an accuracy of 98.7% with a false positive rate of 1.41% because the model misclassified two benign applications as ransomware applications.

6.2.1. Hyper-parameter tuning

Although we managed to improve the base logistic regression model, we also wanted to check the accuracy of the random forest and SVM models by tuning the model parameters. A hyper-parameter is a parameter whose value is set before training the machine learning model. The model parameters are configuration variables internal to the model, and as a note here, a model can run with default parameters. However, the performance of the model can be improved by tuning the parameters. To achieve the best performance, we identified the parameters that have the best impact on the classification results for both models. For random forest, we chose n_estimators and max_depth hyper-parameters, and for SVM, we tuned the parameter C available in Scikit-learn library. Figures 9 and 10 depict the accuracy of the random forest through 10-fold cross-validation when varying the value of n_estimators and max_depth parameters, respectively.

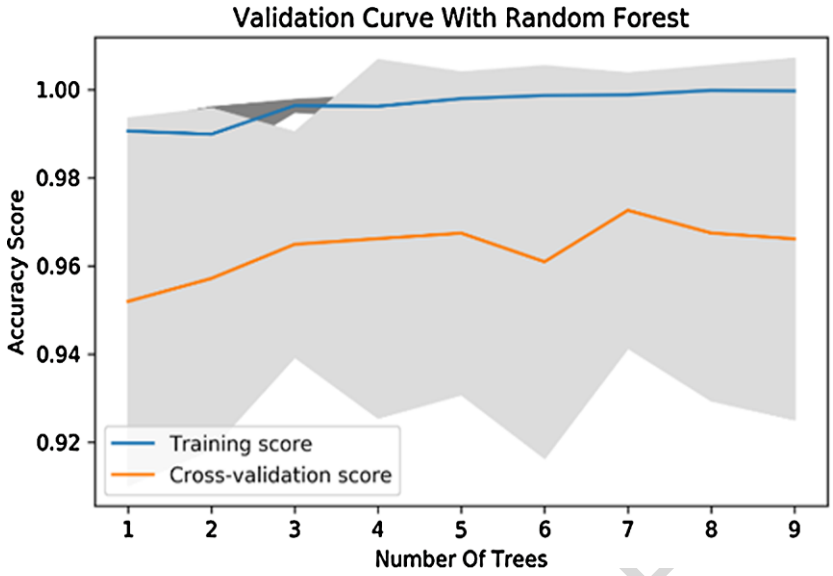

Fig. 9. Random forest 10-fold cross validation score for different values of n_estimators.

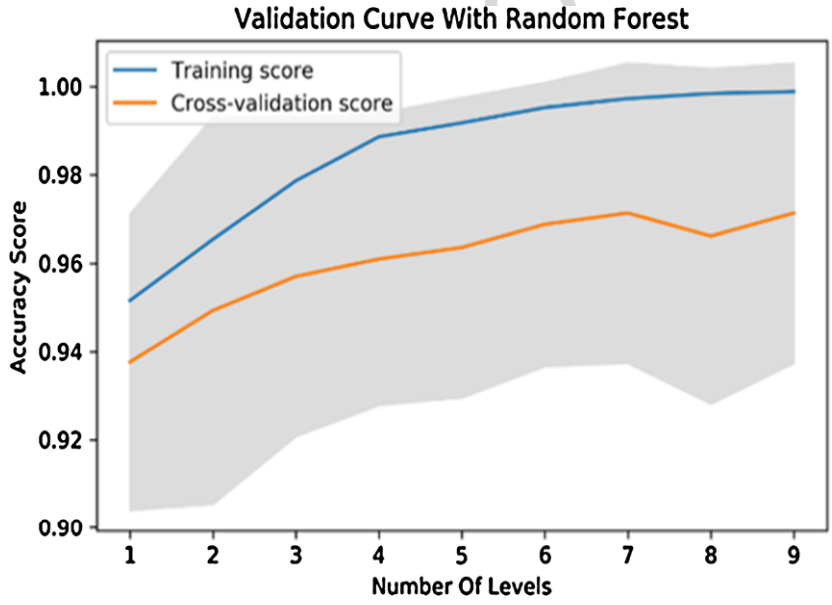

Fig. 10. Random forest 10-fold cross-validation score for different values of max_depth.

We can observe from both figures that the classifier achieved the highest accuracy at value 7 in both parameters. Finally, we trained our random forest model using the aforementioned best parameters; Table 10 summarises the obtained results. Random forest was also successful in detecting all ransomware samples. However, the number of false positives increased here. From the confusion matrix shown in Fig. 11, we can see that four benign applications were misclassified as ransomware.

Table 10
Classification performance for the fandom forest classifier before hyper-parameter tuning

	Precision (%)	Recall (%)	f1-score (%)	Support
Benign	100	73	85	15
Ransomware	97	100	99	140
Micro average	97	97	97	155
Macro average	99	87	92	155
Weighted average	97	97	97	155

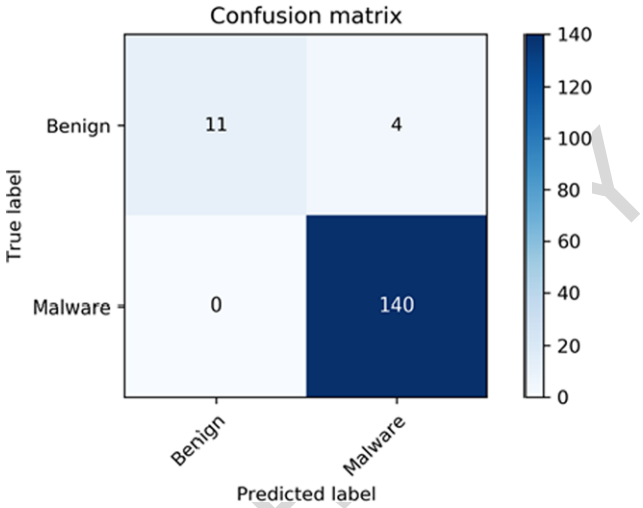

Fig. 11. Confusion matrix for the random forest classifier after hyper-parameter tuning.

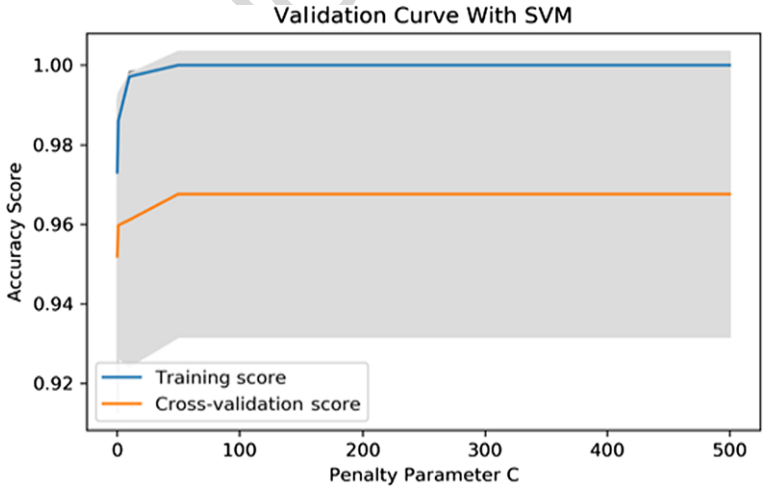

Fig. 12. SVM 10-fold cross-validation score for different values of parameter C.

For the support vector machine, we chose parameter C, the penalty parameter of the error term. This parameter controls the trade-off between classifying the training samples correctly and achieving a smooth decision boundary. We did 10-fold cross-validation by varying the value of this parameter in

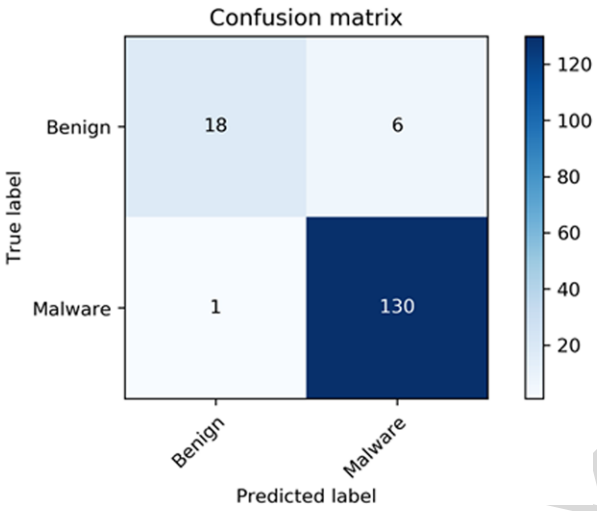

Fig. 13. Confusion matrix for SVM after hyper-parameter tuning.

Table 11
Classification performance for SVM after hyper-parameter tuning

	Precision (%)	Recall (%)	f1-score (%)	Support
Benign	95	75	84	24
Ransomware	96	99	97	131
Micro average	95	95	95	155
Macro average	95	87	91	155
Weighted average	95	95	95	155

the range of 0 to 500. Figure 12 shows that after reaching a value of 50, the accuracy becomes steady. We chose 50 as a tuned value for SVM and generated the classification performance for the tuned model. As shown in Fig. 13, the classification accuracy decreased to 95% in this case. SVM was not able to detect one ransomware sample, and it misclassified six benign samples as ransomware. Table 11 lists the performance results for SVM after hyper-parameter tuning.

6.3. Machine learning using a balanced dataset

We balanced the dataset using data resampling based on the synthetic minority oversampling technique (SMOTE). SMOTE uses the K-nearest neighbours algorithm to generate new data for underrepresented classes in the dataset. Instead of just replicating the minority class, this statistical technique takes samples of each target class nearest to the decision boundary and generates the new samples, combining the features of the target class and its neighbors. In our dataset, we marked benign samples as a minority class. Before applying the SMOTE technique, we split our data into an 80:20 ratio for testing and training purposes. After data split, our dataset consisted of 616 samples in total, consisting of 536 ransomware and 80 benign samples. SMOTE oversampled our minority class and balanced the dataset in equal distribution for each class, as shown in Fig. 14. Using a dataset of 1,072 benign and ransomware samples, we trained a regularised machine learning classifier. Testing was done on a total of 155 samples consisting of 132 ransomware and 23 benign samples. Table 12 and Fig. 15 show the performance obtained for the logistic regression when using SMOTE. As shown in Fig. 15, the accuracy decreased compared with the previous classifier. The model achieved an accuracy of about 97.5%, and it generated

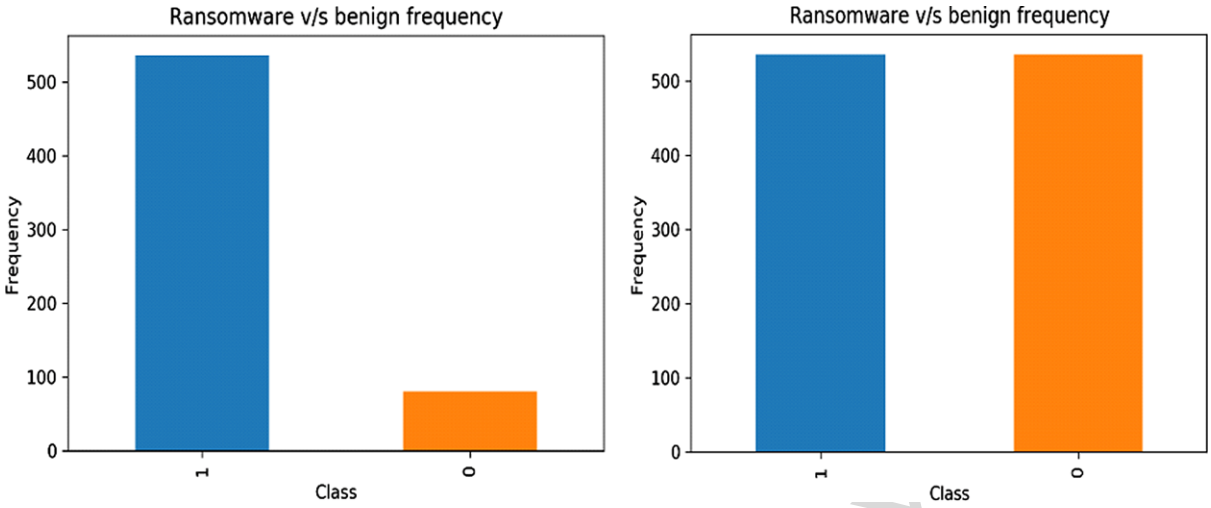

Fig. 14. Class distribution before and after SMOTE.

Table 12
Classification performances for the regularised logistic regression after using SMOTE

	Precision (%)	Recall (%)	f1-score (%)	Support
Benign	88	96	92	23
Ransomware	99	98	98	132
Micro average	97	97	97	155
Macro average	94	97	95	155
Weighted average	98	97	97	155

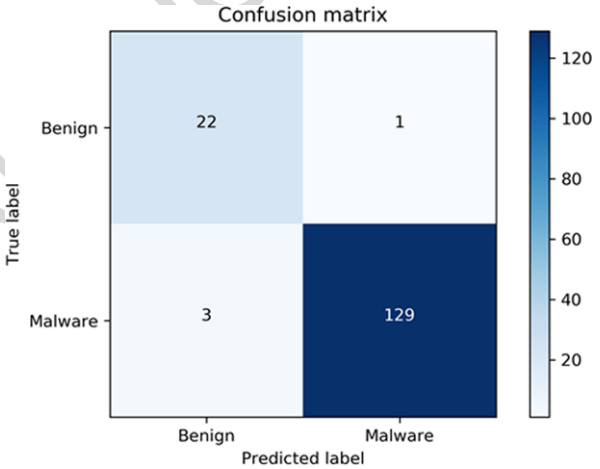

Fig. 15. Confusion matrix of the regularised logistic regression after SMOTE.

three false negatives and one false positive. However, these results are always debatable because SMOTE blindly generates the minority classes without regard to the majority class. As a result, it can increase overlapping of the classes and introduce additional noise.

6.4. Novel ransomware detection

Machine learning has great potential to detect the new variants of ransomware. We achieved the best results with the regularised logistic regression model and decided to use it for further data testing. To test our machine-learning model for novelty detection, we excluded some ransomware families while training the regularised logistic regression classifier. We excluded two families one by one: cerber and locky. We tested each family individually and calculated the performance metrics for each test. The test dataset also contained 30% of the benign samples from our dataset that were not part of the training dataset. Benign data were randomly selected from our current dataset to evaluate each family. As shown in Fig. 16, for each family, the model classified all ransomware samples correctly. However, the model that evaluated the cerber family identified two benign samples as ransomware. Correspondingly, the model that was used to evaluate locky ransomware family identified four benign samples as ransomware. Tables 13 and 14 show the classification performances for both ransomware families. Recall for new ransomware families was 100%, meaning our model successfully detected all previously unseen ransomware samples. However, the machine learning models trained for cerber and locky ransomware families achieved false-positive rates of 1.61% and 3%, respectively.

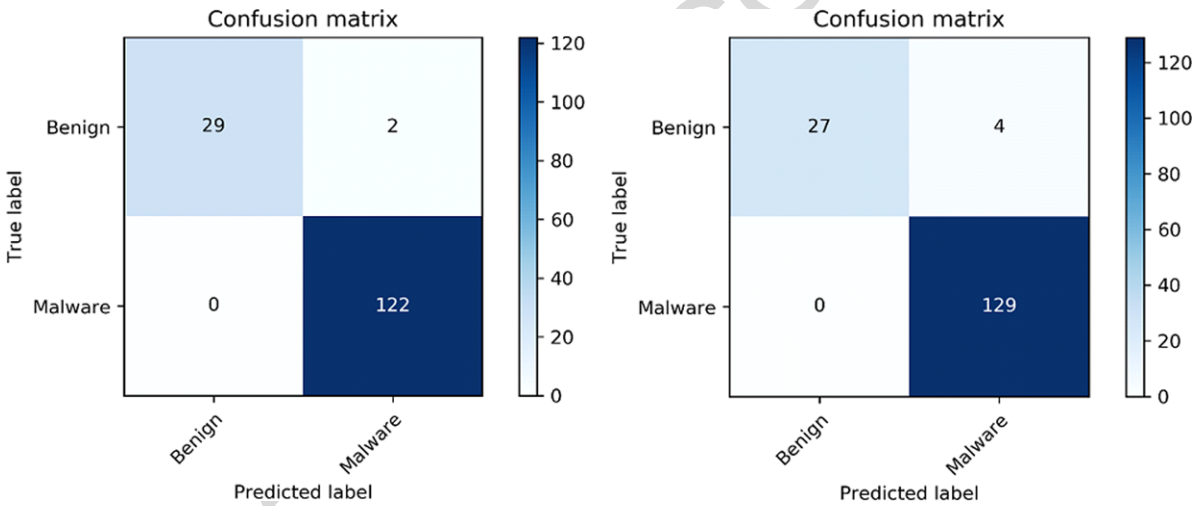

Fig. 16. Confusion matrices for cerber (left) and locky (right) ransomware families.

Table 13
Classification performances for cerber ransomware family

	Precision (%)	Recall (%)	f1-score (%)	Support
Benign	100	94	97	31
Ransomware	98	100	99	122
Micro average	99	99	99	153
Macro average	99	97	98	153
Weighted average	99	99	99	153

Table 14
Classification performances for locky ransomware family

	Precision (%)	Recall (%)	f1-score (%)	Support
Benign	100	87	93	31
Ransomware	97	100	98	129
Micro average	97	97	97	160
Macro average	98	94	96	160
Weighted average	98	97	97	160

	A	B	C	D	E	F	G	H	I
1	FamilyNan	ID	FileName	Time	Entropy	Extension	FileSize	MinimumFileSize	
2	TeslaCrypt	18	binary	2018-04-04 22:42:46.506609974-04:00					
7	TeslaCrypt	18	2d8d62c25e4b3c58_threading.py.mp3	2018-04-04 23:03:05.522609829-04:00	7.995653772	data	48832	48805.47	
9	TeslaCrypt	18	45bc84d034dc6957_time_hashlib.py.mp3	2018-04-04 23:03:06.714609828-04:00	7.894587103	data	3312	3268.359	
10	TeslaCrypt	18	6793f33f1914938b___phello__.foo.py.mp3	2018-04-04 23:03:06.718609828-04:00	6.697723749	data	448	375.0725	
11	TeslaCrypt	18	fbeb89dab4af5eea_test_simplehttpserver.py.mp3	2018-04-04 23:03:07.142609828-04:00	7.767438222	data	1760	1708.836	
13	TeslaCrypt	18	9e166adcc6e4b5c0_unicode_escape.py.mp3	2018-04-04 23:03:07.142609828-04:00	7.747634891	data	1600	1549.527	
14	TeslaCrypt	18	f841c710f6332348_nonmultipart.py.mp3	2018-04-04 23:03:07.178609828-04:00	7.563486816	data	1088	1028.634	
15	TeslaCrypt	18	994be0c490ae905e_000310.ppt.mp3	2018-04-04 23:03:07.178609828-04:00	7.999364235	data	255360	255339.7	
16	TeslaCrypt	18	47412c4362fb98f8_099459.pdf.mp3	2018-04-04 23:03:07.178609828-04:00	7.998479654	data	133392	133366.6	
17	TeslaCrypt	18	8b075c5c07de7d69_queues.py.mp3	2018-04-04 23:03:07.202609828-04:00	7.980874128	data	13072	13040.75	
19	TeslaCrypt	18	0535d532d3bf0759_sbcharsetprober.py.mp3	2018-04-04 23:03:07.290609828-04:00	7.944850374	data	5168	5132.373	
20	TeslaCrypt	18	d348b026f7910598_raw_unicode_escape.py.mp3	2018-04-04 23:03:07.346609828-04:00	7.750226346	data	1632	1581.046	
21	TeslaCrypt	18	62f738440e72bc85_repr.py.mp3	2018-04-04 23:03:07.346609828-04:00	7.940089159	data	4800	4764.053	
22	TeslaCrypt	18	b4637ade823bc5d3_fix_except.py.mp3	2018-04-04 23:03:07.366609828-04:00	7.912052352	data	3824	3781.961	
23	TeslaCrypt	18	085da15528f96858__structures.py.mp3	2018-04-04 23:03:07.662609828-04:00	7.748179611	data	1792	1735.592	
24	TeslaCrypt	18	3e159c6e0e5287fe_glob.py.mp3	2018-04-04 23:03:07.662609828-04:00	7.945073124	data	5584	5545.661	
25	TeslaCrypt	18	8a85ec2ca73d39d3_palmimageplugin.py.mp3	2018-04-04 23:03:07.682609828-04:00	7.973190906	data	9520	9488.097	
26	TeslaCrypt	18	092b341d493b2804_099594.pdf.mp3	2018-04-04 23:03:07.834609828-04:00	7.995892352	data	53184	53156.69	
27	TeslaCrypt	18	0c01634b2fbfe292_test_codecencodings_hk.py.mp3	2018-04-04 23:03:07.866609828-04:00	7.61181349	data	1168	1111.325	
28	TeslaCrypt	18	e070362ba456c1aa_000481.txt.mp3	2018-04-04 23:03:07.878609828-04:00	7.73072889	data	1664	1607.992	
29	TeslaCrypt	18	0d6dad3190740da7_file_util.py.mp3	2018-04-04 23:03:07.898609828-04:00	7.968077517	data	8752	8717.077	
30	TeslaCrypt	18	9d44798a384a07ef_099146.pdf.mp3	2018-04-04 23:03:08.338609828-04:00	7.99674945	data	59232	59207.93	
31	TeslaCrypt	18	597f2fc3c292d905_099155.pdf.mp3	2018-04-04 23:03:08.338609828-04:00	7.999183443	data	246240	246214.9	
32	TeslaCrypt	18	38fdb213d3357241_test_code.py.mp3	2018-04-04 23:03:08.370609828-04:00	7.937437339	data	5184	5143.459	

Fig. 17. Teslacrypt encrypted files with timestamp.

6.5. *Ransomware-triggered vs. user-triggered encryption*

We wanted to identify the difference between ransomware-triggered and user-triggered encryption. To do this, we collected all user files from guest machines after the binary execution. We also collected the timestamps when ransomware and benign binaries were executed in a sandbox environment. Then, we sorted all the files according to their last access time and filtered out the files for which the entropy values were greater than 7 and where the file signatures were missing. To check the file signature, we used python-magic library available in python. As depicted in Fig. 17, the teslacrypt binary was executed around 10:42 PM on 4 April 2018. After almost 21 minutes of the successful execution of the binary, this teslacrypt variant started generating the new files by appending .mp3 in the filename. The files were identified as *data* files by the python-magic library. The extension value *data* means python-magic library was unable to identify the file type. In addition, the files that were missing the file signature had an entropy value of close to 8.

Figure 18 shows the filename, timestamp, entropy, and extension values of the files after the execution of one of the ransomware binaries from the zeta family. After 30 minutes of binary execution, this variant also started generating .scl files of high entropy and unknown file signatures. We did a timestamp analysis for all generated user files and all ransomware binaries. After the analysis, we came to the conclusion that ransomware, after specific period of time from execution, starts generating random files of high entropy

A	B	C	D	E	F	G	H
FamilyName	ID	FileName	Time	Entropy	Extension	FileSize	MinimumFileSize
zeta	563	binary	2018-04-18 11:09:10.066854722-04:00				
zeta	563	965b6a4bce52fd73_ar_jo.msg.id_60f3beebafa7737_email_enc6@dr.com_.scl	2018-04-18 11:39:34.934854505-04:00	7.902285941	data	1824	1801.721195
zeta	563	5d4ea3dda699f8de_fractions.py.id_60f3beebafa7737_email_enc6@dr.com_.scl	2018-04-18 11:39:35.026854504-04:00	7.992503791	data	23008	22986.4409
zeta	563	eb402003cd883f9f_es_pa.msg.id_60f3beebafa7737_email_enc6@dr.com_.scl	2018-04-18 11:39:37.926854504-04:00	7.082140418	data	256	226.6284934
zeta	563	b42559f788114a34_test_funcattrs.py.id_60f3beebafa7737_email_enc6@dr.com_.scl	2018-04-18 11:39:37.962854504-04:00	7.986876318	data	13104	13082.50341
zeta	563	5f777056d34558dd_pythread.h.id_60f3beebafa7737_email_enc6@dr.com_.scl	2018-04-18 11:39:37.986854504-04:00	7.843095081	data	1200	1176.464262
zeta	563	f43381fa94795bc3_albumartsmall.jpg.id_60f3beebafa7737_email_enc6@dr.com_.scl	2018-04-18 11:39:38.026854504-04:00	7.964215647	data	5264	5240.453896
zeta	563	da3be7867462eb63_099483.pdf.id_60f3beebafa7737_email_enc6@dr.com_.scl	2018-04-18 11:39:38.046854504-04:00	7.995059822	data	42624	42597.67873
zeta	563	43beebdfd465dfb3_test_pstats.py.id_60f3beebafa7737_email_enc6@dr.com_.scl	2018-04-18 11:39:38.046854504-04:00	7.80933471	data	992	968.3575041
zeta	563	f98990fd10b58834_dummy_threading.py.id_60f3beebafa7737_email_enc6@dr.com_.scl	2018-04-18 11:39:38.046854504-04:00	7.93856987	data	2896	2873.762293
zeta	563	3d119652fb11b276_es_pe.msg.id_60f3beebafa7737_email_enc6@dr.com_.scl	2018-04-18 11:39:38.050854504-04:00	7.14326831	data	256	228.5845859
zeta	563	7b70fffa7ee35b4d_592572.pptx.id_60f3beebafa7737_email_enc6@dr.com_.scl	2018-04-18 11:39:38.062854504-04:00	7.999946469	data	3817248	3817222.457
zeta	563	f1036e8f256693c4_lv.msg.id_60f3beebafa7737_email_enc6@dr.com_.scl	2018-04-18 11:39:38.066854504-04:00	7.856969143	data	1232	1209.973248
zeta	563	dd20f209d0f3afa5_test_dictviews.py.id_60f3beebafa7737_email_enc6@dr.com_.scl	2018-04-18 11:39:38.090854504-04:00	7.980893713	data	9824	9800.53748
zeta	563	7fbb677e14b5989a_test_dbtables.py.id_60f3beebafa7737_email_enc6@dr.com_.scl	2018-04-18 11:39:38.110854504-04:00	7.987711596	data	15760	15735.79184
zeta	563	bd605e68cc1e38d1_099097.jpg.id_60f3beebafa7737_email_enc6@dr.com_.scl	2018-04-18 11:39:38.146854504-04:00	7.995059013	data	40304	40279.10731
zeta	563	aa342ed4eaf446db_es_cr.msg.id_60f3beebafa7737_email_enc6@dr.com_.scl	2018-04-18 11:39:38.150854504-04:00	7.232001855	data	256	231.4240594
zeta	563	73d32bbd4fee1f40_pyarena.h.id_60f3beebafa7737_email_enc6@dr.com_.scl	2018-04-18 11:39:38.170854504-04:00	7.932857751	data	2768	2744.768782
zeta	563	e23bbc85ce6e25c7_help_decrypt_your_files.txt	2018-04-18 11:39:38.178854504-04:00	3.406880718	data	3234	1377.23153
zeta	563	fb1ea9f4924e7a2a_pt_br.msg.id_60f3beebafa7737_email_enc6@dr.com_.scl	2018-04-18 11:39:38.186854504-04:00	7.299413229	data	288	262.7788762
zeta	563	484f7a326332a9e5_099558.pdf.id_60f3beebafa7737_email_enc6@dr.com_.scl	2018-04-18 11:39:38.202854504-04:00	7.995738436	data	38448	38427.51892
zeta	563	bacf574884e1b93f_descrobject.h.id_60f3beebafa7737_email_enc6@dr.com_.scl	2018-04-18 11:39:38.222854504-04:00	7.923410014	data	2576	2551.338024
zeta	563	368f62e59bb05339_test_gdbm.py.id_60f3beebafa7737_email_enc6@dr.com_.scl	2018-04-18 11:39:38.230854504-04:00	7.925162191	data	2624	2599.453198
zeta	563	926fdb20d5993f11_msgcat-1.5.2.tm.id_60f3beebafa7737_email_enc6@dr.com_.scl	2018-04-18 11:39:38.250854504-04:00	7.987986306	data	16416	16391.3479
zeta	563	126eb279a8b46219_keycert2.pem.id_60f3beebafa7737_email_enc6@dr.com_.scl	2018-04-18 11:39:38.258854504-04:00	7.906033681	data	1824	1802.575679
zeta	563	23a7e2b7dfa7cea3_replacedialog.py.id_60f3beebafa7737_email_enc6@dr.com_.scl	2018-04-18 11:39:38.266854504-04:00	7.972262811	data	6864	6840.201492
zeta	563	657bb3123cb79d35_test_setups.py.id_60f3beebafa7737_email_enc6@dr.com_.scl	2018-04-18 11:39:38.298854504-04:00	7.989469752	data	16976	16953.65481

Fig. 18. Zeta encrypted files with timestamp.

with unknown file signatures. On the one hand, the files encrypted with legitimate encryption tools like 7-zip, WinRAR, and so on, correspond to known file formats whose file signatures can easily be identified. On the other hand, ransomware generates unknown file extensions. This behaviour of ransomware can be helpful in identifying the ransomware not detected by current anti-virus programmes and can help save the user files with minimum file loss.

To be assured about the files encrypted by ransomware are high entropy values, we filtered out the files which had missing file signatures and calculated the average entropy per ransomware family. Table 15 shows the average entropy of the files which are missing file signatures. We can see from the table that the average entropy for almost all the ransomware families is greater than 7. Crysis and striked families are the exceptions in our case, having an average entropy of 2.21 and 5.32, respectively. Upon checking the files of the crysis and striked ransomware families, we found out that despite the successful execution of the ransomware binaries, the binaries were unable to start the encryption process. Possible reasons for such failure may include the inability of these binaries to connect to their C&C servers, or the fact that the 30-minute threshold we used to execute the ransomware might be very low for these particular ransomware to start encryption, and so forth.

To evaluate the performance of the file entropy/signature detector, our second detector, we used the same ransomware and benign samples which were used to evaluate the ML detector. The test dataset used to evaluate the regularized logistic regression model contained 155 samples, out of which 15 were benign and 140 were ransomware. For all 140 ransomware samples, we filtered out the files which were missing file signatures and calculated the average entropy of the files per ransomware sample. On the one hand, the average entropy of the files, after the execution of the 134 ransomware samples, were higher than seven. On the other hand, for all the benign samples, the average entropy remained below seven. Under the above scenario, the new confusion matrix and classification performances achieved after passing the samples through our second detector are shown in Fig. 19 and Table 16, respectively.

We can see from the confusion matrix that we were successfully able to reduce the false positives by using our two detectors in tandem. The second detector successfully reclassified two ransomware samples that were falsely classified as benign by the ML detector. However, the second detector also increased false negatives by classifying six ransomware samples as benign. Those six ransomware be-

Table 15
Average file entropy per family

Family	Average entropy
CTBLocker	7.85
Cerber	7.88
CryptoMix	7.82
CryptoShield	7.83
Crysis	2.21
Flawed	7.75
GlobeImposter	7.45
Jaff	7.75
Locky	7.90
Mole	7.81
Petya	7.98
Sage	7.79
Satan	7.29
Spora	7.94
Striked	5.32
TeslaCrypt	7.87
Unlock26	7.92
WannaCry	7.98
Win32.Blocker	7.07
Xorist	7.78
zeta	7.79
Benign	4.30

longed to the teslacrypt, crysis and striked ransomware families. The above scenario makes detection more critical as ransomware samples were identified as benign. Upon checking the files, it was observed that these ransomware samples did not start any encryption process yet. As a result, the average entropy stayed below the threshold. In this case, when the first detector classifies a binary as a ransomware while the second detector classifies it as legitimate, our system will still raise a warning by notifying about the suspicious binary.

6.6. Detection system efficiency

Our ransomware detection system has two modules for ransomware detection: File entropy and file signature module and Machine learning module. The machine learning module classifies the ransomware through features like grouped registry key operations, directory enumerations, windows API calls, etc. While the file entropy and file signature module continuously checks the file signature and entropy of modified, created or deleted files in Windows environment. To check the applicability of our proposed architecture in real environment, we kept the multilayer detection system running for 24 hours on our analysis machine (with characteristics: Windows 7 – 64 bit, 8 Gb of RAM, 512 Gb of hard drive, Intel Core i7 7700 HQ 2.8 GHz).

The detection system in a normal state when no backup is initiated by the system and no software is getting installed uses less than 1% of the CPU. However, when the backup in the system is initiated the machine's CPU usage goes slightly up and stays below 3%.

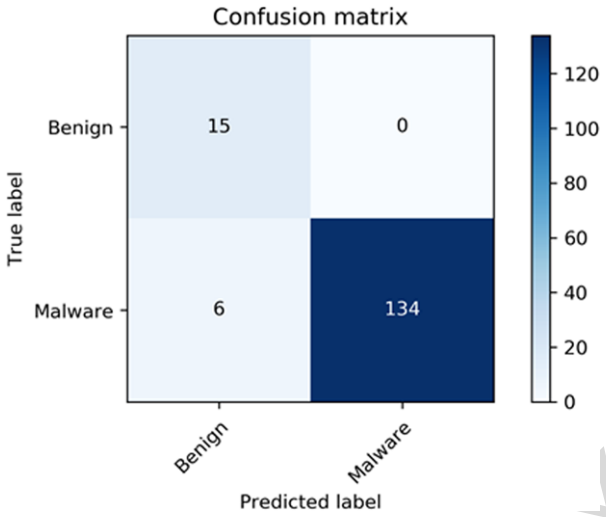

Fig. 19. Confusion matrix for the file entropy/signature detector.

Table 16
Classification performance for the file entropy/signature detector

	Precision (%)	Recall (%)	f1-score (%)	Support
Benign	71	100	83	15
Ransomware	100	96	98	140
Micro average	96	96	96	155
Macro average	86	98	91	155
Weighted average	97	96	96	155

Also, the detection time of our algorithm is totally dependent on our machine learning model. The machine learning detector has the capability of detecting the ransomware right after installation in the Windows environment. However, if a new ransomware variant can evade the first detector (i.e. the ML detector) then to classify the sample our detection system has to wait until ransomware starts the encryption of the user files.

In case of false negative when our ML detector fails to detect the ransomware, our second detector continuously checks the high entropy operations of missing file signatures. The model will check for the encrypted files with entropy greater than seven and missing file signatures. When the counter reaches 10 files, the system will automatically start backing up the files that have been modified and generate an alert to notify the user about the suspicious process. A legitimate limitation of our system is that if the ML detector fails to detect the ransomware then the user will loose the first 10 files encrypted by the ransomware. However, all the files modified after triggering the backup is triggered would be recovered easily.

7. Conclusion

Over the years, researchers have proposed various approaches for detecting ransomware. Unfortunately, the major works done on dynamic ransomware detection are biased towards limited feature space

(e.g., API calls). To fill this gap, we introduce a new set of features based on grouped registry key operations, and a monitoring model based on combined file entropy and file signature tracking. Through extensive experimentation, we demonstrated that grouped registry key operations play an essential role along with other features in ransomware detection and help achieve a 100% detection rate with a false positive rate of 1.4%. We have also shown that the proposed method is capable of detecting all the known and new ransomware. Furthermore, we proposed a detection architecture that helps differentiate user-triggered encryption from ransomware-triggered encryption, thereby saving as many files as possible during an attack. Our future work will consist of expanding our features space by closely monitoring the latest ransomware and exploring more compelling features. Future work will also include an extension of the current collected ransomware dataset.

There are many ways to improve the final accuracy and false positive rate of our ransomware detection system. One such way is combining our system with a static-based detection system and making the system a hybrid one. If code obfuscation is not done, the static-based detection system would be adequate in detecting previously seen ransomware samples. If the ransomware is detected at an early stage through static-based detection, it would not have to pass through the complex process of feature extraction and machine learning classification. The results might also help to reduce the false positive rate and increase the detection accuracy of our system. Additionally, there are several advantages of extending our current dataset. One advantage is to classify the ransomware based on family. One of the future scopes of our work consists of covering all possible ransomware families and identifying the expected ransomware class based on execution behaviour in the Windows operating system. We will be evaluating our system on large-scale data and experimenting with other machine learning algorithms, such as deep learning.

To more effectively classify the ransomware-triggered encryption, we are also planning to build a kernel-level driver that will monitor the file system changes at the MFT (master file table) level where the file signature is missing. The driver will continuously detect the files that are deleted or overwritten. For the files currently being modified, the driver will make a temporary copy and hold it into temporary storage for a predefined time T. The time T will be defined after the analysis of different ransomware families and the time taken by them to encrypt the user files. If the ransomware has accessed a large number of files in a short time, the threshold of the files will be low. If the ransomware encrypts the files and stops for a while, the value of the time parameter T can be defined as very large. At the end of the defined period T, the decision can be made to classify the binary as ransomware or benign. Also, the encrypted or deleted files can be restored from the backup.

References

[1] M.M. Ahmadian, H.R. Shahriari and S.M. Ghaffarian, Connection-monitor & connection-breaker: A novel approach for prevention and detection of high survivable ransomwares, in: *12th International Iranian Society of Cryptology Conference on Information Security and Cryptology (ISCISC)*, IEEE, 2015, pp. 79–84. doi:10.1109/ISCISC.2015.7387902.

[2] N. Andronio, S. Zanero and F. Maggi, Heldroid: Dissecting and detecting mobile ransomware, in: *International Workshop on Recent Advances in Intrusion Detection*, Springer Verlag, New York, 2015, pp. 382–404.

[3] J. Brownlee, An introduction to feature selection, 2014. https://machinelearningmastery.com/an-introduction-to-feature-selection.

[4] K. Cabaj, M. Gregorczyk and W. Mazurczyk, Software-defined networking-based crypto ransomware detection using HTTP traffic characteristics, *Computers & Electrical Engineering* (2017).

[5] Z.-G. Chen, H.-S. Kang, S.-N. Yin and S.-R. 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*, ACM, New York, NY, USA, 2017, pp. 196–201. doi:10.1145/3129676.3129704.

[6] A. Continella, A. Guagnelli, G. Zingaro, G. De Pasquale, A. Barenghi, S. Zanero and F. Maggi, ShieldFS: A self-healing, ransomware-aware filesystem, in: *Proceedings of the 32nd Annual Conference on Computer Security Applications*, ACM, 2016, pp. 336–347.
[7] G. Cusack, O. Michel and E. Keller, Machine learning-based detection of ransomware using SDN, in: *Proceedings of the 2018 ACM International Workshop on Security in Software Defined Networks & Network Function Virtualization*, ACM, 2018, pp. 1–6.
[8] J.R. Douceur and W.J. Bolosky, A large-scale study of file-system contents, *ACM SIGMETRICS Performance Evaluation Review* **26**(4) (2016), 212–221.
[9] S. Garfinkel, P. Farrell and G. Dinolt, Bringing science to digital forensics with standardized forensic corpora, in: *Digital Forensic Research Conference*, Montreal, Canada, 2009.
[10] A. Gharib and A. Ghorbani, Dna-droid: A real-time Android ransomware detection framework, in: *International Conference on Network and System Security*, Springer, 2017, pp. 184–198. doi:10.1007/978-3-319-64701-2_14.
[11] M.M. Hasan and M.M. Rahman, RansHunt: A support vector machines based ransomware analysis framework with integrated feature set, in: *20th International Conference of Computer and Information Technology (ICCIT)*, 2017.
[12] B.J. Hicks, A. Dong, R. Palmer and H.C. Mcalpine, Organizing and managing personal electronic files: A mechanical engineer's perspective, *ACM Transactions on Information Systems (TOIS)* **26**(4) (2008), 23. doi:10.1145/1402256.1402262.
[13] J. Huang, J. Xu, X. Xing, P. Liu and M.K. Qureshi, FlashGuard: Leveraging intrinsic flash properties to defend against encryption ransomware, in: *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, ACM, 2017, pp. 2231–2244.
[14] G.C. Kessler, File Signatures, 2019. https://www.garykessler.net/library/file_sigs.html.
[15] A. Kharraz, S. Arshad, C. Mulliner, W.K. Robertson and E. Kirda, Unveil: A large-scale, automated approach to detecting ransomware, in: *USENIX Security Symposium*, 2016, pp. 757–772.
[16] A. Kharraz, W. Robertson, D. Balzarotti, L. Bilge and E. Kirda, Cutting the gordian knot: A look under the hood of ransomware attacks, in: *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*, Springer, Cham, 2015, pp. 3–24. doi:10.1007/978-3-319-20550-2_1.
[17] A. Kharraz, W. Robertson and E. Kirda, Protecting against ransomware: A new line of research or restating classic ideas?, *IEEE Security & Privacy* **16**(3) (2018), 103–107. doi:10.1109/MSP.2018.2701165.
[18] E. Kolodenker, W. Koch, G. Stringhini and M. Egele, PayBreak: Defense against cryptographic ransomware, in: *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security*, ACM, 2017, pp. 599–611.
[19] J. Lee, J. Lee and J. Hong, How to make efficient decoy files for ransomware detection?, in: *Proceedings of the International Conference on Research in Adaptive and Convergent Systems*, ACM, 2017, pp. 208–212. doi:10.1145/3129676.3129713.
[20] LogRhythm, The ransomware threat: A guide to detecting an attack before it's too late, Technical report, LogRhythm EMEA, 2018. http://www.ncstrl.org:8900/ncstrl/servlet/search?formname=detail&id=oai%3Ancstrlh%3Amitai%3AMIT-LCS%2F%2FMIT%2FLCS%2FTR-200.
[21] D. Maiorca, F. Mercaldo, G. Giacinto, C.A. Visaggio and F. Martinelli, Rpackdroid: Api package-based characterization and detection of mobile ransomware, in: *Proceedings of the Symposium on Applied Computing*, ACM, 2017, pp. 1718–1723. doi:10.1145/3019612.3019793.
[22] L. Mathews, NotPetya ransomware attack cost shipping giant Maersk over $200 million, *Forbes Magazine* (2017). https://www.forbes.com/sites/leemathews/2017/08/16/notpetya-ransomware-attack-cost-shipping-giant-maersk-over-200-million/#3c3ec1eb4f9a.
[23] F. Mbol, J.-M. Robert and A. Sadighian, An efficient approach to detect torrentlocker ransomware in computer systems, in: *International Conference on Cryptology and Network Security*, Springer, 2016, pp. 532–541. doi:10.1007/978-3-319-48965-0_32.
[24] J. McKnight, The evolution of ransomware and breadth of its economic impact, Master's thesis, Utica College, 2017.
[25] S. Mehnaz, A. Mudgerikar and E. Bertino, RWGuard: A real-time detection system against cryptographic ransomware, in: *International Symposium on Research in Attacks, Intrusions, and Defenses*, Springer, 2018, pp. 114–136. doi:10.1007/978-3-030-00470-5_6.
[26] F. Mercaldo, V. Nardone, A. Santone and C.A. Visaggio, Ransomware steals your phone. Formal methods rescue it, in: *International Conference on Formal Techniques for Distributed Objects, Components, and Systems*, Springer, 2016, pp. 212–221. doi:10.1007/978-3-319-39570-8_14.
[27] R. Moussaileb, B. Bouget, A. Palisse, H. Le Bouder, N. Cuppens and J.-L. Lanet, Ransomware's early mitigation mechanisms, in: *Proceedings of the 13th International Conference on Availability, Reliability and Security*, ACM, 2018, p. 2.
[28] N.B. University of Texas San Antonio, University of Texas San Antonio, FILETYPES1 File Type Samples, 2014. http://digitalcorpora.org/corp/nps/files/filetypes1/.
[29] D. Nieuwenhuizen, A behavioural-based approach to ransomware detection, *Whitepaper. MWR Labs Whitepapery* (2017). https://pdfs.semanticscholar.org/93b6/e2fdf2a79608e44ab64e37bddd6973f54f1d.pdf.

[30] M. Reynolds, Ransomware attack hits 200,000 computers across the globe, *New Scientist* (2017). https://www.newscientist.com/article/2130983-ransomware-attack-hits-200000-computers-across-the-globe.
[31] C. Rossow, C.J. Dietrich, C. Grier, C. Kreibich, V. Paxson, N. Pohlmann, H. Bos and M. Van Steen, Prudent practices for designing malware experiments: Status quo and outlook, in: *2012 IEEE Symposium on Security and Privacy (SP)*, IEEE, 2012, pp. 65–79. doi:10.1109/SP.2012.14.
[32] C. Sandbox, Cuckoo Sandbox – Automated malware analysis, 2017. https://cuckoosandbox.org.
[33] N. Scaife, H. Carter, P. Traynor and K.R. Butler, Cryptolock (and drop it): Stopping ransomware attacks on user data, in: *IEEE 36th International Conference on Distributed Computing Systems (ICDCS)*, IEEE, 2016, pp. 303–312.
[34] T. Seals, Troldesh Nabs top ransomware spot, *Infosecurity Magazine* (2017). https://www.infosecurity-magazine.com/news/troldesh-nabs-top-ransomware-spot.
[35] D. Sgandurra, L. Muñoz-González, R. Mohsen and E.C. Lupu, Automated dynamic analysis of ransomware: Benefits, limitations and use for detection, Preprint, 2016. arXiv:1609.03020.
[36] S.K. Shaukat and V.J. Ribeiro, RansomWall: A layered defense system against cryptographic ransomware attacks using machine learning, in: *10th International Conference on Communication Systems and Networks (COMSNETS)*, 2018, pp. 356–363.
[37] S. Song, B. Kim and S. Lee, The effective ransomware prevention technique using process monitoring on android platform, *Computer Fraud & Security* **2016** (2016).
[38] R. Soto and J. Zadeh, Automated prevention of ransomware with machine learning and GPOs, 2017. https://www.rsaconference.com/events/us17/agenda/sessions/6695-automated-prevention-of-ransomware-with-machine.
[39] D. Wagner and P. Soto, Mimicry attacks on host-based intrusion detection systems, in: *Proceedings of the 9th ACM Conference on Computer and Communications Security*, ACM, 2002, pp. 255–264.
[40] T. Yang, Y. Yang, K. Qian, D.C.-T. Lo, Y. Qian and L. Tao, Automated detection and analysis for Android ransomware, in: *2015 IEEE 7th International Symposium on Cyberspace Safety and Security (CSS), 2015 IEEE 12th International Conferen on Embedded Software and Systems (ICESS)*, IEEE, 2015, pp. 1338–1343.
[41] K. Yeager, LibGuides: SPSS Tutorials: Pearson Correlation, 2019. https://libguides.library.kent.edu/SPSS/PearsonCorr.