AJBAR Vol. 3(4), August 2024: 96-111, ISSN: 2811-2881

Arid Zone Journal of Basic and Applied Research

Faculty of Science, Borno State University Maiduguri, Nigeria

Journal homepage: https://www.azjournalbar.com

Research Article

PEDLA Detection Model for Crypto-Ransomware

Oluwasogo Adekunle Okunade[1], Raymond Ternenge Igbudu[1] and Emmanuel Gbenga Dada[2]

[1]Department of Computer Science, Faculty of Computing, National Open University of Nigeria, Abuja, Nigeria

[2]Department of Computer Science, Faculty of Physical Sciences, University of Maiduguri, Nigeria

*Corresponding author's Email: igbuduraymond@gmail.com, doi.org/10.55639/607.05040303

ARTICLE INFO:

Keywords:
Crypto-ransomware,
Encryption,
Early detection,
Pre-encryption stage,
Signature-based

ABSTRACT

Crypto ransomware is a challenging cybersecurity threat that encrypts the files of the victim and demands a ransom in exchange for the decryption key. Traditional signature-based protection methods, such as antivirus and anti-malware, have proven in-effective at preventing crypto-ransomware attacks; therefore the production of ransomware is on the rise. Existing methods for early detection of crypto-ransomware attacks during the pre-encryption phase before encryption happens rely on a timing thresholding methodology to set the border of the pre-encryption stage. However, the fixed time threshold strategy, suggests that the samples begin encryption at the exact moment. This is not always the case since timing varies between crypto-ransomware families as a result of the obfuscation techniques used to evade detection. This research work therefore, proposed the creation of a Pre-Encryption Detection-Learning Algorithm (PEDLA). PEDLA monitors the pre-encryption stage for every case separately relying on the initial appearance of any API's related to cryptography to establish the pre-encryption stage boundary, whereby features are extracted and used in training a prediction model using the Random Forest machine learning algorithm. The sample data was obtained from widely used ransomware repositories such as VirusShare, Virus total and kaggle.com. The model achieved a detection accuracy of 98.6% with False Positive Rate (FPR) of 1.9%. Four classifiers including Support Vector Machines (SVM), K-Nearest Neighbour (KNN), Multi-Layer Perceptron (MLP), and AdaBoost, were used to evaluate the model's classification abilities. Furthermore, a comparison was done between the related works and PEDLA. The findings show that PEDLA performed better across most calculated metrics, such as accuracy, precision and recall.

Corresponding author: Raymond Ternenge Igbudu, **Email:** igbuduraymond@gmail.com
Department of Computer Science, National Open University of Nigeria, Abuja, Nigeria

INTRODUCTION

Cyber security is the practice of safeguarding data, networks, and devices from cyber-attacks. These cyber-attacks are normally launched with the intent to alter, destroy, or access the data of the user or produce intrusion in the users' business processes. The implementation of cyber-security has become critical and demanding because there are very few individuals against many computing devices. These cyber attackers are typically evil-minded people who are usually interested in altering, destroying, and/or accessing the data and reputation of the user. Cyber-security would not be possible if attention is not given to malware attacks (Tariq, 2020). The creation of intrusion detection systems and anti-viruses has prompted cyber-criminals to also create very powerful and advanced malware which can evolve; causing some serious infections and can even metamorphose (Gazet, 2010). Malware is a program developed to cause damage to the network of a computer, client, a computer, server and/or some other resources of the computer user. This malware usually disables and causes damage to computer resources without the knowledge of the user and contravenes the rights of the user. Malware has several forms such as Viruses, Trojan, Rootkit, Worms, crypto ransomware, and so on.

Crypto-ransomware is one of today's most dangerous types of malware. Once infected, the malware encrypts the victim's data and blocks access until ransom is paid, resulting in multi-million dollar cyber-extortion each year. This type of malware has caught the interest of cybercriminals due to numerous success stories with global ramifications, such as cryptowall, Wannacry, and NotPetya (Oz et al., 2021). Because of the huge level of interest, plenty of other new crypto-ransomware have been developed, as well as improving existing ransomware with new variants.

Crypto-ransomware is distinguished by its irreversible effect even after detection and removal. As such, early detection is critical to safeguarding user data and files from being held for ransom. Cybercriminals have refined crypto-ransomware attack aspects such as greater encryption algorithms, worm-like capabilities, pseudo-anonymous payment methods, and the availability of Ransomware-as-a-Service (RaaS) on the dark-web, which facilitates creation of new ransomware variants (Oz et al., 2021). As a result, crypto-ransomware attacks are on the rise. However, conventional malware detection methods are ineffective for detecting crypto-ransomware (Al-Rimy et al., 2021).

Many commercial and open-source anti-crypto-ransomware solutions rely on signature-based detection methods, which are quick and accurate for detecting known malware but far too restrictive for detecting zero-day attacks (Kok et al., 2020). The signature repositories must be updated on a regular basis, and due to cybercriminals' great interest in ransomware, new variants of ransomware that can bypass antiviruses continue to emerge at a rapid pace. Despite continuous improvements or updates, malware authors maintain a one-step advantage because new variants are produced quicker than new signatures can be generated, tested, and added to malicious signature repositories (Kok et al., 2020).

Detection methods capable of dealing with zero-day crypto-ransomware attacks strive to detect the infection based on broader features such as ransomware-specific operations rather than just file signatures alone. In this regard, several strategies for detecting crypto-ransomware attacks have been proposed; they can be classified as data-centric or process-centric (Al-Rimy et al., 2021).

The data-centric solutions monitor the victim's computer's digital assets and sound an alarm if any suspicious changes are found (Al-Rimy et al., 2021). They examine the file structure changes to see whether they are suspicious. This approach, however, cannot tell if the file structure change was caused by a crypto-ransomware attack or by benign application, resulting in a high rate of false alarms (Al-Rimy et al., 2021). Furthermore, the data-centric method may not entirely guard against ransomware attacks since it sacrifices a portion of the files before detection (Scaife et al., 2016). These files may be worth more to the victim than the other data.

Process-centric solutions, on the other hand, are classified into two groups. One, by monitoring system activities such as, file system access for example privilege elevation, network activity, resource usage and interactions with the operating system, and triggering an alarm when particular

encryption-related events occur (Kharaz et al., 2016). However, relying on ad hoc incidents for crypto-ransomware attack detection raises the likelihood of false alerts because crypto-ransomware is not always the cause; benign applications can also cause them. (Al-Rimy et al., 2021). Additionally, there is no assurance that such ad hoc events will always occur prior to encryption; they may occur after encryption due to changes in attack techniques (Kharaz et al., 2016). As a result, this strategy is ineffective for early detection.

The second type of process-centric solutions monitors the running process's behaviour and gathers various forms of behavioural data which are subsequently utilized to train different machine learning classifiers. The key obstacle with the existing methods for early detection of crypto-ransomware is lack of adequate data during the initial stages of an attack, which limits the capacity of feature extraction algorithms in early detection solutions to discover attack features, resulting in data loss, low detection accuracy, and a high false-positive rate (Al-Rimy et al., 2020). Furthermore, they adopt a set time-based thresholding method to determine the pre-encryption stage borders (Scaife et al. 2016: Hwang et al. 2020).

However, the set-time thresholding strategy, suggests that the samples begin encryption at the exact moment. This is not always the case since timing varies between crypto-ransomware families as a result of the obfuscation techniques used to evade detection (Al-Rimy et al., 2020). As a result, this strategy may miss the start of the encryption operation, resulting in encryption of several files before detection (Scaife et al., 2016).

Regardless of the efforts put forth, the existing approaches, inevitably, have limitations. Crypto-ransomware is still an intricate problem that requires further study to improve current detection methods. In light of this, this work proposes development of a model capable of detecting crypto-ransomware during the pre-encryption stage, before encryption happens.

Aslan (2020) explains malicious software (malware) detection as the act of evaluating the content of a program to find out if it is malicious or benign. Generally, detection methods are grouped into 3 kinds:

- Behavior-based: those that make use of dynamic analysis,
- Signature-based: those that make use of static analysis, and
- Hybrid: those that use both dynamic and static analysis.

Signature-based detection techniques examine the code of an application before it is implemented to adjudge if it has the capability of malicious action. If the static analysis identifies any malicious code, then the executable shall be halted from executing. The signature analysis entails getting code string patterns (known as signatures) from codes of the target application and matching them to the malicious code patterns database (Nieuwenhuizean, 2019).

This technique is effective and quick in identifying ransom-ware that is known. The failure of signature-based method to identify ransomware that is unknown is its main shortcoming. A malicious executable code could be detected only after it has been confirmed as being malicious and included in the database of malicious signatures (Nieuwenhuizean, 2019). This definitely has numerous resultant effects for the effectiveness of static-based detection;

- Firstly, it is not effective against code obfuscation; ransomware authors make use of code-obfuscation techniques so as to keep on changing malware so that every variant appears different from others in order to avoid detection by the signature-based methods.
- Secondly, the signature-based method of detection is not effective against ransom-ware that has short cycles of development. This has become a challenge for signature-based detection systems because new ransomware forms are developed faster than new ransom-ware signatures can be collected, tested and included in the malware signature repositories (Nieuwenhuizean, 2019).
- Thirdly, signature-based detection is not effective in the face of targeted ransomware. Making use of the RaaS (Ransomware-as-a-Service) paradigm, bots can change signatures in order to target specific companies. This makes it easy for the creation of a very personalized ransomware variant with

the capability of evading detection by signature-based methods.

Various authors have put forward some static analysis-based techniques for identifying cryptographic ransomware attacks.

Zhang (2019) put forward a model for transforming operation code (*opcode*) sequences into N-gram sequences that were then utilized for training the machine learning model: the accuracy of the model was 91.43%.

In Baldwin (2018), Static analysis was utilized to remove operation codes (*opcodes)* from benign and malicious Portable Executable files. Then, the characteristics of the extracted *opcode* were used as input to Support Vector Machine (SVM) machine learning classifier. Here, the very best accuracy gotten from 5 cryptographic ransom-ware variants was around 96.5%.

Alzahrani (2018) put forward a static-analysis technique known as *Ran-Droid*. This approach searches for a possibly suspicious information in the application code which could be an image, or a text. The main reason behind this is that ransomware's major aim is to extort monetary ransom from its victims. Because of this, these ransomware variants need to have a message of threat in its codes. The major drawback of this technique is this; the message of threat may be released after the data of the victim has been encrypted as a payload.

To manage the drawbacks of signature-based detection techniques, the methods of detection with the capability of handling cryptographic ransomware should have its focus more on broader features like ransomware-specific processes instead of file-signatures only.

A behavior-based method of detection involves evaluating the behaviors and interactions of a process against their surroundings in realtime so as to identify malicious intention. Every executable files are normally taken to be unknowns and it is the responsibility of the executable file to show it is safe and it is not malicious. Maliciously behaving operations will be detected and quarantined (Nieuwenhuizean, 2019).

Behavior-based methods of detection are categorically grouped as process-centric or data-centric. The process-centric approach offers two solutions. Firstly, by evaluating the resources of a system like file system access for instance, resource usage, network activity, privilege elevation and communication with OS. Amongst these, the cryptographic ransom-ware system activity is differentiated by the serious encryption of the data of the victim, which results in a file system activity that is unusual (Kharaz, 2016). Secondly, by evaluating the behavior of a process that is running, examining each line of code, and all potential activities carried out by the code are evaluated, such as having access into any crucial files or files that are not relevant, processes and internal services, and collecting different kinds of behavioral data that are then used to train various machine-learning classifiers (Al-Rimy, 2021).

The data-centric technique monitors the digital assets of the victim's computer, and alerts the user if any changes are found to be suspicious. The data-centric technique also monitors changes in file structures so as to know if they behave suspiciously. The data-centric techniques employ strategies such as file entropy, decoy techniques, and contents similarity measures. They do this to know what is going on in the structure of the file both before and after access (Al-Rimy, 2021).

The main drawback of behavior-based method of detection is that it is very difficult to execute; and, dynamic analysis on many dimensions causes delay that affects performance negatively. Also, methods of advanced code obfuscation make it impossible to conduct proper ransom-ware examination. Finally, the behavior of some ransomware variants in virtual machines or sandboxes is not proper.

Alzahrani (2018) proposed Unveil, which is a dynamic analysis technique that runs applications in a virtual environment and examines the activities of file system and the interactions of desktop for suspicious behavior that may identify ransom-ware infections. Alzahrani's Unveil had a rate of detection of 96.3%. However, Unveil could confuse heavy activity of the file system as the presence of ransom-ware.

Hwang (2020) propounded a mixed-detection model for ransom-ware that is two-phase; the Markov model, and a Random Forest (RF) model. The authors tested Application Programming Interface classification and disclosed that it had a 97.3% accuracy and a 1.5% FNR with a 4.8% FPR.

Takeuchi (2018) presented a ransom-ware detection technique based on SVM's, which learn the ransom-ware Application

Programming Interface calls as features, giving way to the support vector machines to identify ransom-ware that were previously unknown. The authors used 276 malware samples and 312 benign samples and got a rate of detection of 97.48%.

Wecksten (2016) examined the behavior of 4 kinds of cryptographic ransom-ware in a virtual-machine that was running the Win 7 operating system. These scientists discovered that the executable, vssadmin.exe is so much utilized in cryptographic ransom-ware attacks, and they proposed that users should stop the use of"vssadmin.exe" to avoid this cyber attack.

The developers of Continella (2016) attempted to find the Advanced Encryption Standard (AES) key in process memory through the interception of file access system calls. If the original file is somehow already encrypted, this technique gives room for its restoration. Unfortunately, this technique works only with ransom-ware that makes use of AES symmetric encryption. This technique will fail if the ransom-ware uses another encryption technique. Ransom-ware with administrator privileges can also uninstall this technique.

The hybrid system of ransom-ware detection detects cryptographic ransom-ware by using a combination of signature-based and behavior-based detection approaches.

Subedi (2018) propounded a technique that is integrated and is a combination of static analysis and run-time analysis to connect code segments with the dynamic behavior of ransom-ware. These scientists developed CRSTATIC, which is a reverse engineering-based tool for analysis used for identifying cryptographic ransom-ware.

Shaukat (2018) presented an (STL), Strong Trap Layer, which combines static analysis and dynamic analysis to produce a set of attributes that makes use of machine-learning strategies to depict the behavior of ransom-ware. The rate of detection was reported as 98% when making use of the GradientTreeBoostingAlgorithm (GTBA).

Other authors proposed to fight the cryptographic ransom-ware threat by the use of key escrow. This key escrow is a technique of saving keys and other cryptographic info for future use. Using the ransom-ware viewpoint, key escrow explores to gather secret keys and other materials while the ransom-ware encrypts files such that the files can be gotten back after the attack. This technique is predicated on the basis that should the encryption keys be recovered after the attack, the ransom-ware-encrypted files could be retrieved.

Kolodenker (2017) introduced PayBreak, which is a crypto-API monitoring strategy that excerpts cryptographic variables and encryption keys, and saves the keys inside a key vault such that when there is a ransom-ware attack, the encrypted files could be decrypted with the use of brute-force techniques, making use of the encryption keys and other cryptographic variables gotten at the key vault. According to the creators, ransom-ware can evade PayBreak by using third-party libraries and code obfuscation strategies (Kolodenker, 2017).

METHODOLOGY

Shown in Figure 1, is a research framework designed for this research work designed to meet all the objectives of the research. There shall be five blocks of research activity in this work, and they are Research Samples, Database, Cuckoo Sandbox, Machine Learning, and Data. An explanation of each of the blocks is detailed below (Kok *et al*, 2020)

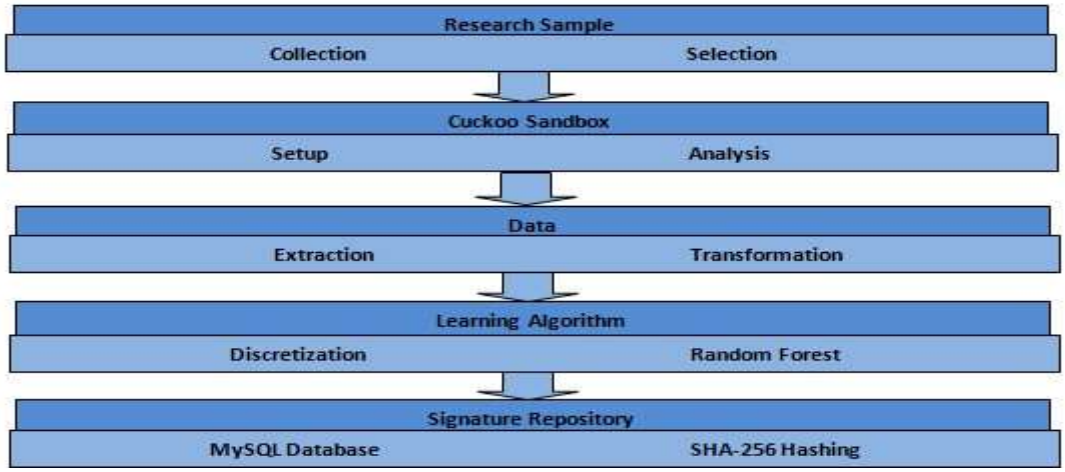

Figure 1: *PEDLA Model Framework*

Research Samples

The researchers started off by obtaining samples (ransom-ware and good-ware) from VirusShare (VS). This ransomware data-set was composed of 357 samples. These samples were grouped based on their families and defined as cryptographic malware by many of the anti-virus programs present on Virus-total. Those research samples stand for various families such as Petya, CryptoWall, Cerber, WannaCry, and TeslaCrypt. The goodware data-set had 75 samples collected from kaggle.com, which is a data science and machine learning online community platform.

VirusShare is a malicious software repository which shares malware collections with forensic analysts, incident responders, security researchers, and those who are curious morbidly. VS had 68,671,265 malware samples as at the time this research work was conducted (Kok et al, 2020).

Table 1: *Dataset*

S/No.	Crypto-ransomware Family	Number of Samples
1	Petya	35
2	CryptoWall	28
3	Cerber	57
4	WannaCry	40
5	TeslaCrypt	31
6	CryptoLocker	48
7	Cyrar	67
8	Satana	51

Through VS the researcher successfully obtained 357 new cryptographic ransom-ware samples.

For pre-encryption sample selection, the researcher passed all the samples collected in the Cuckoo Sandbox. This sandbox produced a record that stated the sequence of all API's (Application Program Interface) that were called by the sample. Those samples that had the 'crypt' keyword in there API sequence were chosen. The API's with the 'crypt' keyword were found to carry out encryption function as denoted in Table 2.

Table 2: API's with the 'crypt' keyword

S/No.	Application Program Interface (API)	DESCRIPTION
1	CryptHashData	To append data to a hash object that allowed long or discontinuous data streams
2	CryptGenKey	To generate a random encryption key in symmetrical encryption or public/private key pair in asymmetrical encryption
3	CryptCreateHash	To start hashing data stream that allowed communication in a secured session
4	CryptEncrypt	To perform encryption function using encryption key as specified by CSP using *hkey* parameter
5	CryptDecodeObjectEx	To decode according to type of structure as specified by *ipszstructType* parameter
6	CryptExportKey	To export an encryption key in symmetrical encryption or a pair of encryption keys in asymmetrical encryption from specific *CSP* securely
7	CryptAcquireContextW	
8	CryptAcquireContextA	To obtain key container from a specific cryptographic service (CSP) that was called by CryptoAPI

The Cuckoo Sandbox

The Cuckoo Sandbox is a malicious software (malware) analysis system which is used to check a program's behavior by running the program in a sandbox. As soon as the program has been deposited in the sandbox, Cuckoo Sandbox tracks all the Application Program Interface (API) calls made by the program. Thereafter, the Cuckoo generates a report that has a sequence of all the API's that are being called (Kok et al, 2020).

The Environment Setup

The dynamic analysis of the crypto ransomware was carried out in a controlled environment built on a computer with Core i5 processor @ 2.3 GHZ and 8 GB RAM. The operating system for the host computer is Linux Ubuntu 18 whereas the guest machine runs a Windows 7 guest operating system. To build a convincing setting in the guest machine where the samples were run, several applications were installed, including Google Chrome, Mozilla Firefox, Adobe Acrobat Reader, and Microsoft Office. Additionally, some files, such as MS Word documents, PPT, Excel, PDF, GIF, and JPG, were created in multiple places on the guest machine's local storage. The proposed methodologies, analysis, and findings were implemented using Python 3.6 modules (Njoroge, 2022).

Before using Cuckoo Sandbox for the analysis, the researchers had to configure the virtual environment using VirtualBox. This was to ensure that the analysis can be carried out in a safe condition. The guest computer system that was installed in the VirtualBox runs Windows 7 (Win7). After developing the Win7 guest VirtualBox, the researchers followed the steps in Table 3 to configure the network to allow the communication between the host machine (Cuckoo Sandbox) and the guest machine (Win7). Next, the researchers took a screenshot of the guest machine to allow us to reset it to this clean state after each sample analysis. Cuckoo Sandbox configurations were also changed as illustrated in Table 4

In order to run the sample analysis in the Cuckoo Sandbox, the guest machine needs to be firstly started in a VirtualBox. Then we can start the Cuckoo Sandbox service, followed by the Cuckoo Web service. Once the analysis has been completed, a report would be generated. This report can be extracted by copying the report file from this location "/.cuckoo/storage/analyses/1/report/report.json ".

Table 3: Configuration on Guest Machine

Step	Description
1	Configured the Network Address Translation (NAT) Network
2	Disabled the Dynamic Host Configuration Protocol (DHCP)
3	Disabled the Host-only Ethernet Adapter
4	Disabled the firewall in guest machine
5	Lowered UAC (User Account Control) in guest machine
6	Set static IPv4 in guest machine
7	Installed Python 2.7.16 in guest machine
8	Installed Pillow
9	Copied agent.py from Cuckoo Sandbox into the guest machine
10	Executed agent.py in guest machine

Table 4: Cuckoo Sandbox Configuration

Step	Description
1	Enabled .MongoDB in reporting.conf file
2	Setup the guest profile to the guest's machine name, "Win7SP1x64", in memory.conf file
3	Setup same static IP in guest machine and screenshot the name created previously in virtualbox.conf file

Data

This report produced by the Cuckoo Sandbox contained important information about pre-encryption APIs, that is, the entire API's before the call of any encryption function. However, before the extraction begins, the researcher had to make sure that the code can rightly extract the needed information. To achieve this, the researchers first of all developed a code which was used to extract all the API's, and then compared the summary of the extracted API with 'apistats' which could be found in the Cuckoo report. Once the researchers had verified that the code had gotten a similar result, we went ahead to check for the API with the keyword 'crypt', as the stop point of extraction. Shown below in *Algorithm 1* is the final pseudo code (Kok et al, 2020).

// data extraction

1. Open file
2. While there is line
3. Goto next line
4. Search for "*api"*
5. If found, get the next word
6. If the word has "crypt", then stop
7. Else store word
8. Else next
9. End while
10. Close file

Algorithm 1: Data Extraction

The data extraction algorithm above produced a 'txt' file which had to be converted to a *data* format ('csv' file) that can be later fed into the LA (Learning Algorithm). The 'csv' file had about 235 columns. The first column registered the sample's name. The second column specified whether the sample was goodware (zero) or crypto ransomware (one). The third column would specify the sample's source. The fourth column onwards consisted of API's that were compiled after extracting all the APIs from all the samples; a total of 232 APIs were found. Based on the data extracted above, the researchers produced three data-sets. The first data-set consisted of the full API's from all the samples. The second data-set consisted of ransomware found to have the encryption function on, and the pre-encryption API's were the only ones extracted. The third data-set was one from a previous research utilized in (Sgandurra, 2016), for extracting data-set of API's that are related. All the data-sets contained similar 'goodware' data that was extracted from (Sgandurra, 2016).

Learning Algorithm (LA)

The Learning Algorithm (LA) was a key component which could determine unknown and known ransomware attacks. The LA consisted of two steps: first step was a pre-processing of data by discretization; and the

second step was to use Random Forest (RF) to train the prediction model.

Random Forest was chosen for this study because it performs well, particularly for discrete datasets, and can also categorize a large amount of data with a low error rate. Moreover, since the algorithm chooses trees at random, over-fitting is avoided, and scaling is not required for Random Forest to train efficiently. According to Fawagreh et al. (2014), "Random Forest approach has proved its high accuracy and superiority."

In order to verify how effective the LA model is, the researchers had used all three data-sets in two different ratios of training and testing. At first, the ratio used was 80:20, whereby 80% of data were used for the training and 20% of data used for testing. The second ratio used was 70:30, where 70% of data was used for training and 30% of data were used for testing. Additionally, the researchers also carried out a 10-fold verification test. We had also carried out a test on only Random Forest with no discretisation pre-processing. Shown in Figure 2 below is the scenario utilized to analyze the Learning Algorithm.

The metrics used for evaluation are True Positive Rate (TP Rate), False Positive Rate (FP Rate), F-measure, Recall, Precision-Recall Curve (PRC), Matthews Correlation Coefficient (MCC), Precision, and Receiver Operating Characteristics (ROC). The TP Rate examined how well the Learning Algorithm model could correctly predict the positives, whereas the FP Rate evaluated how well the Learning Algorithm model incorrectly predicts positives. *Precision* evaluated how true a positive prediction will be. *Recall* evaluated the correctness of the positive prediction. *F-measure* calculated the balance between recall and precision.*MCC* examined how well the prediction of the Learning Algorithm model correlates to the actual result. The *ROC* is an area that is under the curve of *FP Rate* versus the *TP Rate*, whereas *PRC* is an area that is under the curve of Recall versus Precision.

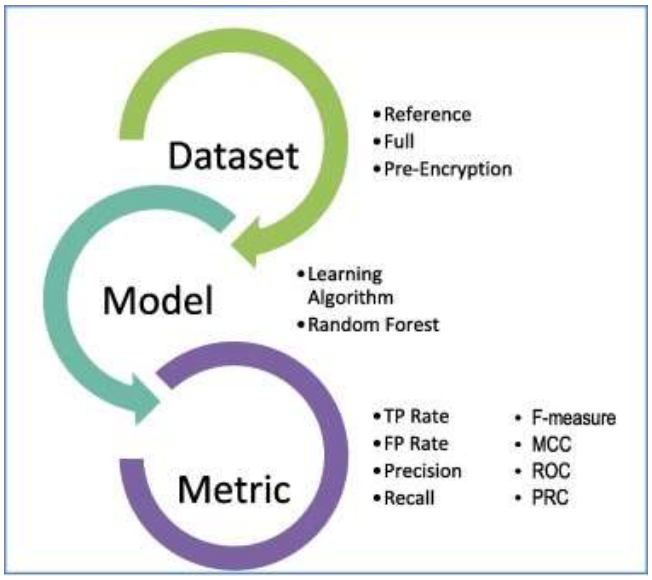

Figure 2: *Analysis of the learning algorithm model*

The Signature Repository

Once a ransomware has been identified by the Learning Algorithm above, the file's signature and other key information are usually saved in a database of MySQL. The signatures of ransomware were produced by hashing the ransomware file with SHA-256, which produced a fixed length of 64-characters code. The Hashing method paved the way for a fast and easy matching of the content of the file. The technique could identify ransomware immediately and with no need to be examined by the Cuckoo Sandbox. In average, Cuckoo Sandbox takes approximately 1 min to analyze a file.

Hence, the method was very accurate, considerably faster, and safer. Nevertheless, it made use of a signature repository to work, which can be attained by making use of the Learning Algorithm. The method is specific to the ransomware sample and also very rigid - any slightest changes would make the method not to be effective again.

Shown in Fig. 3 is the MySQL database design for the Signature Repository. Whenever the Learning Algorithm identifies a file to be ransomware, the file would be quarantined, and key data will be saved in the two tables in the database; specifically tables *Repeat* and *Sample*. If the ransomware was detected by the Learning Algorithm, this signified that it was the very first time that this ransomware had attacked the victim.

The table called *Sample* has eight fields: first field was *Hash-Value* which has the file's hash code. *File-Name* field saved the file's original name. The *Attack* field stored the *n*-times that the ransomware has attempted to attack the victim. *Date* and *Time* fields saved the date and time of the very first attack by the ransomware. The *Size* field stored the size of the file in MB (megabytes). The *Created* field stored the developer of the file. And lastly, *Location* field stored the location of the quarantined file which had its extension removed to make it not to be active again.

Whenever the same ransomware attacks the victim for the second time, the signature matching method will detect it. If this happens, information of the second attack would be stored in the table *Repeat*.

Additionally, *Attacks* field in table *Sample* will add one value and the file will be removed (Kok et al, 2020).

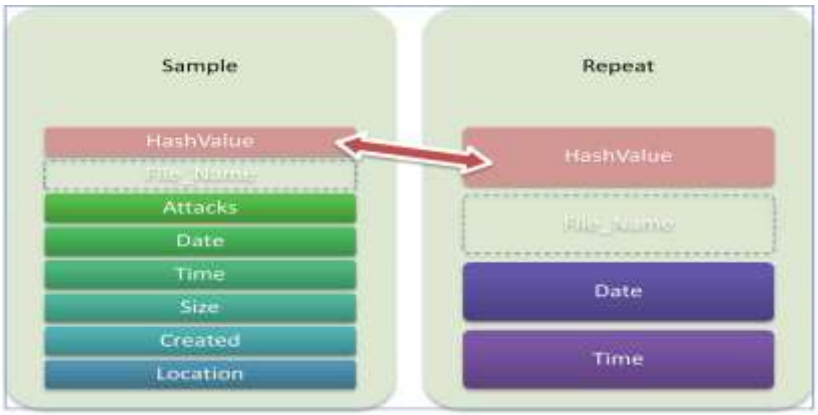

Figure 3: Signature repository data structure

RESULTS

PEDLA detection model has been designed in four stages, namely, features extraction, features selection, training/testing and prediction.

Features Extraction

The crypto-ransomware and benign samples are introduced one by one into the host machine's cuckoo sandbox instance. The cuckoo agent runs the binary files provided and captures the attack's features.

Figure 4: Cuckoo Sandbox dashboard

The activity report in JSON format for each file is stored in the sample's own trace-file. The API list and API sequences, which are required for the detection model, are then retrieved from the activity reports.

Features Selection

Rocchio relevance feedback with TF-IDF generated the pre-encryption border vector, as described in section 3.4.1 of the methodology. This procedure resulted in the definition of the

pre-encryption border vector, as seen in Table 4.2. Every API signifies a border vector entry. Each entry's weight is also indicated as determined by equation (1).

$$\nu_f = \nu_{initial} + \frac{1}{\eta}\left(\Sigma_{relevant} d_j - \frac{1}{2}\Sigma_{irrelevant} d_j\right) \quad - - - - - \quad (1)$$

Whereby ν_f stands for the feedback-vector; $\nu_{Initial}$ represents the original-vector; $\Sigma_{relevant}\ d_j$ stands for the relevant-group, and $\Sigma_{irrelevant}\ d_j$ is the irrelevant-group.

Table 5: The pre-encryption border vector

API	Weight (feature importance)
Cryptencrypt	0.896
Cryptdecrypt	0.821
Cryptacquirecontexta	0.812
Cryptacquirecontextw	0.779
Cryptunprotectdata	0.723
Cryptcreatehash	0.604
Crypthashdata	0.593
CryptgenKey	0.581
CryptexportKey	0.556
Encryptmessage	0.524
Cryptdecodeobjectx	0.509
CryptGetObjectUrl	0.503
CryptReleaseContext	0.499
CryptGetHashParam	0.474
CertGetNameStringW	0.476
Certcontrolstore	0.401
Certopenstore	0.401
Ntcreateprocessex	0.379
Removedirectorya	0.347
Ntdeletefile	0.343
NtdeletevalueKey	0.303
Rtlcompressbuffer	0.297
Internetgetconnectedstate	0.290
Ntqueryfullattributesfile	0.289
Openservicea	0.273

The variables considered to be significant were:
API calls usage: The most prevalent APIs that were invoked by the ransomware. Table 6 shows the names of the common APIs being called.

Table 6: Names of cryptoAPI's captured

CryptUnprotectData	CryptGenKey	CryptEncrypt
CryptDecrypt	EncryptMessage	CryptExportKey
CryptDecodeObjectEX	CryptReleaseContext	CryptHashData
CryptCreateHash	CryptAcquireContextA	CryptGetHashParam
CertGetNameStringW	CryptAcquireContextW	CryptGetObjectUrl

API calls input arguments: The APIs that accept cryptography related APIs and/or functions as parameters. They indicate the file's precise behaviour. Among them are, CertGetNameStringW, Certcontrolstore, Certopenstore, Ntcreateprocessex, Removedirectorya, Ntdeletefile, and Ntdeletevaluekey

API calls frequency: The APIs that invoke excessive and redundant cryptography related APIs. The weights and rank for the API occurrences are shown in table 5.

Table 5 demonstrates that the cryptoAPIs that are explicit were scored higher than other APIs related to cryptography. This means that the proposed method was effective in identifying the cryptograph-related APIs more precisely among all APIs included in the InitialAt subset.

Training and Testing

The random forest machine learning algorithm is used to train the model. However, four more learning algorithms were also trained for comparison purposes. They include Support Vector Machines (SVM), K-Nearest Neighbour (KNN), Multi-Layer Perceptron (MLP), and AdaBoost.

The effectiveness of the proposed model was assessed using several performance evaluation metrics including accuracy, f-score, recall, precision, false positive rate (FPR), false negative rate (FNR), ROC-AUC and cross-validation. We tested PEDLA against other machine learning algorithms used in malware classification and also conducted a comparative analysis with models presented by related works.

The first comparison is between PEDLA and four different learning algorithms as shown in table 7. According to the findings, the model attained a detection accuracy of 98.6%, Precision of 98.3%, recall of 99.1%, F score of 99.3% and a ROC-AUC of 99.1%.

Accuracy represents the ratio of all correct predictions. A higher precision means the lower the number of false positive errors committed by the classifier. The higher the recall value, the fewer cases misclassified as negative. A high ROC-AUC value indicates that the model can differentiate between negative and positive occurrences well.

Table 7: Experiment results using different classifiers

Metric	PEDLA	SVM	AdaBoost	KNN	MLP
Accuracy	0.986	0.915	0.975	0.978	0.935
F1-score	0.993	0.968	0.988	0.989	0.977
Precision	0.983	0.899	0.969	0.973	0.925
Recall	0.991	0.993	0.967	0.979	0.998
ROC-AUC	0.991	0.935	0.959	0.961	0.942

The highest accuracy on unseen test-set within the first 20 seconds of execution is shown in Table 8, along with the related false positive rate (FPR) and false negative rate (FNR). The highest accuracy of 98.6% was achieved in the fifth second of execution. However, as shown in figure 5, performance is consistently high (greater than 97%) even in the first few seconds. The model's sustained stability at various time intervals is an indication that it can predict the threat's appearance earliest possible.

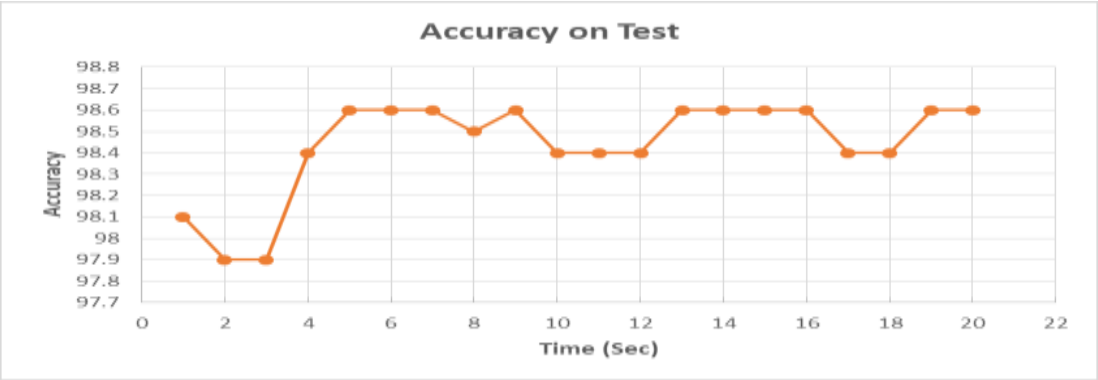

Figure 5: PEDLA performance at varying time stamps

Table 8: The highest accuracy on unseen test set

Classifier	Accuracy (%)	Time (s)	FPR (%)	FNR (%)
PEDLA	98.6	5	1.9	2.7
SVM	91.5	10	3.4	5.1
AdaBoost	97.5	8	4.7	7.2
KNN	97.8	4	1.9	3.2
MLP	93.5	14	3.9	5.5

A low FPR indicates that the model is less prone to generating false alarms, while a low FNR indicates that it is less likely to predict a negative class wrongly.
The highest average accuracy during 10-fold cross validation on the training-set within the first 20 seconds of execution is shown in Table 9, along with the related false positive rate (FPR) and false negative rate (FNR).

Table 9: Highest average accuracy during 10-fold cross validation

Classifier	Accuracy (%)	Time (s)	FPR (%)	FNR (%)
PEDLA	98.4	4	2.3	2.1
SVM	90.1	10	4.5	3.6
AdaBoost	95.6	4	6.7	5.2
KNN	97.1	3	6.4	5.7
MLP	92.9	13	7.2	5.8

The researchers also conducted a comparative analysis with few models presented by related works. This is shown in Table 10 and Figure 6.

Table 10: Comparison of detection results with related works

Technique	Accuracy (%)	Recall (%)	F-Score (%)	Precision (%)	FNR (%)	FPR (%)
PEDLA	98.9	99.1	99.3	98.3	2.6	1.8
Rhode et al (2021)	96.1	-	-	-	4.7	3.1
Sgandurra et al (2016)	97.9	96.3	-	-	2.3	1.6
Scaife et al (2016)	93.0	-	-	-	6.0	-
Homayoun et al (2017)	98.3	98.0	98.0	-	-	0.9

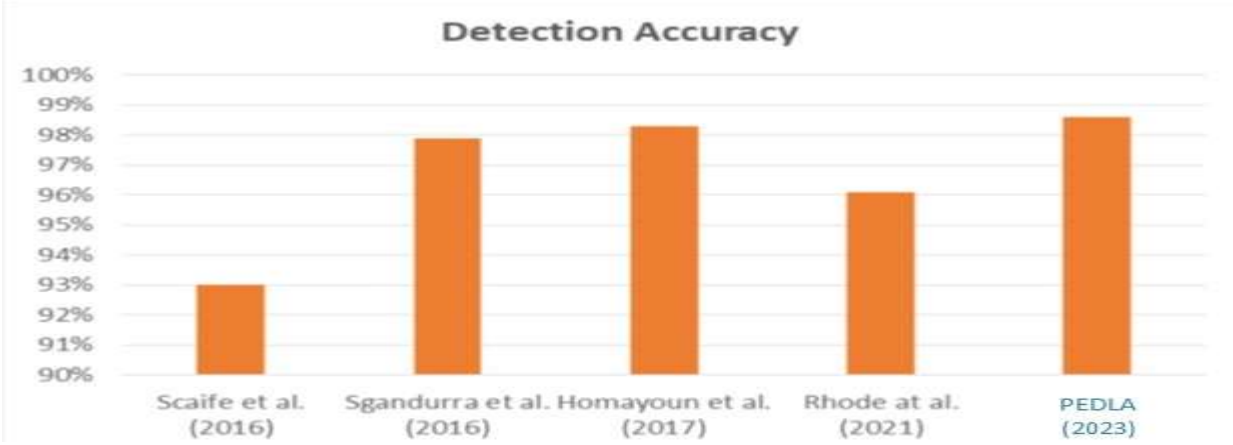

Figure 6: Comparison of detection accuracy

The detection accuracy is the proportion of all correct predictions. Accuracy, however, isn't necessarily a good indicator of a model's ability to make correct predictions. Despite being a useful assessment metric, it may not be adequate owing to the nature of the dataset. If a dataset is imbalanced, for instance, there is a bias in favour of the largest represented group. To address this issue, other performance metrics including F-score, recall, precision, FPR and FNR were used in this study.

Prediction

The prediction capability in our early detection of crypto-ransomware model is explained by the good results produced. It can be shown from the results that the model achieved a detection accuracy of 98.6%, F score of 99.3%, ROC-AUC of 99.1%, Precision of 98.3% and a recall of 99.1%.

DISCUSSION OF RESULTS

This research proposed the idea of early crypto-ransomware detection depending on dynamic pre-encryption phase boundaries. The PEDLA was proposed and developed to dynamically define the pre-encryption stage border in the lifecycle of crypto-ransomware and extract the features before encryption that were used in training the detection model. In contrast to fixed time-based thresholding, PEDLA uses a dynamic thresholding approach, which entails creating a border vector prior to encryption that contains the cryptography-related APIs being called. This vector's entries were selected based on their computed weights utilizing the proposed approach.

The comparative findings in Figure 6 show that our approach was more effective in identifying the pre-encryption border than the fixed-time thresholding method used in related works. This is due to the dependence on cryptography-related APIs to monitor the start of encryption for every sample, irrespective of how long it took for encryption to begin. Table 7 experimental findings show that the cryptoAPIs that are explicit were scored higher than other APIs related to cryptography. This means that the proposed method was effective in identifying the cryptography-related APIs more precisely among all APIs included in the Initial subset.

Table 10 and Figure 6 also demonstrate how well the model performed compared to the related works in Scaife et al. (2016), Sgandurra et al. (2016), Homayoun et al. (2017) and Rhode et al. (2021). Accordingly, the PEDLA dynamic thresholding proved more successful than other thresholding strategies in capturing the behavioural component of crypto-ransomware attacks at earlier stages. This is due to the PEDLA's ability to incorporate more data than other strategies since it monitors the onset of encryption for each instance separately. It utilizes the supplementary traits that certain ransomware samples in the dataset may have, allowing it to collect more pre-encryption data. Despite the fact that some ransomware instances begin encryption pretty fast, there are certain cases when the encryption begins late. As a result, dynamic thresholding compensates for a lack of data in samples that begin encryption quite fast with data gathered by samples that begin encryption late.

CONCLUSION

The goal of this research was to create a scheme for detecting crypto-ransomware attacks during the pre-encryption stage. The solution was effective in defining the crypto-ransomware lifecycle's pre-encryption borders, extracting the distinctive features that depict the attack patterns in this stage, and incorporating the features into the creation of the crypto-ransomware detection model.

Crypto ransomware is a particularly dangerous kind of malware. It is distinguished by its irreversible effect even after detection and removal. As such, early detection is critical to safeguarding user data and files from being held for ransom. However, the idea of early crypto-ransomware detection is constrained in how it determines the pre-encryption border. The existing studies either use a set threshold to define the pre-encryption border or utilize the entire data obtained during the attack which is ineffective in early detection.

In this research work, an early detection model for crypto ransomware was proposed. The model focuses on API calls and functions to detect malware encryption operation. APIs allow programs to interact and send information to one another; once we know which APIs are being called, the machine learning model can identify and halt the process. This gives us control and the ability to examine requesting application to determine whether it is a ransomware and then terminate it before it encrypts any files.

The model creates the pre-encryption border vector, which comprises all cryptography-

related APIs responsible for determining the pre-encryption stage border during the crypto ransomware lifecycle. The model proved more effective than previous works in defining the pre-encryption stage border of crypto ransomware attacks.

Four classifiers including Support Vector Machines (SVM), K-Nearest Neighbour (KNN), Multi-Layer Perception (MLP), and AdaBoost, were utilized to evaluate the model's classification abilities. Furthermore, a comparison was done between the related works and our model. The findings show that PEDLA performed better across most calculated metrics, including accuracy, precision and recall. PEDLA also has a low FPR of 1.9%, indicating that it is highly unlikely to misclassify a benign program. This demonstrates the effectiveness of the proposed approach in precisely defining and extracting the features most closely related with the pre-encryption stage, and then using these features to build the detection model. Consequently, we can conclude that the proposed scheme may well be used for early detection of crypto ransomware attacks during the pre-encryption stage.

FUTURE WORK

PEDLA usage has its limitations. One of the notable limitations is its dependence on use of Windows API, which provided more efficient encryption process by crypto ransomware. PEDLA does not have the ability to detect crypto ransomware that uses its own native encryption code. Hence, PEDLA should be utilized as a supplement detection system rather than the only crypto ransomware detection system. More so, PEDLA was designed to detect only one type of ransomware, whereas there are numerous kinds of malware in the wild.

The results coming from PEDLA are very good but it requires further improvements for it to be used by the public. This is due to the fact that the configuration and installation of supporting applications like MySQL and Cuckoo Sandbox can be challenging for many people. Hence, for future work, PEDLA should be built such that it can be implemented as a standalone application without the need for a separate configuration of supporting applications.

CONFLICT OF INTEREST

We have no conflicts of interest to disclose.

REFERENCES

Al-Rimy, B. A. S., Maarof, M. A., Alazab, M., Shaid, S. Z. M., Ghaleb, F. A., Almalawi, A., &Al-Hadhrami, T. (2021). Redundancy coefficient gradual up-weighting-based mutual information feature selection technique for crypto-ransomware early detection. *Future Generation Computer Systems*, *115*, 641-658.

Aslan, O. A., & Samet, R. (2020). A comprehensive review on malware detection approaches. *IEEE Access*, *8*, 6249-6271.

Alzahrani, A., Alshehri, A., Alshahrani, H., Alharthi, R., Fu, H., Liu, A., &Zhu, Y. (2018, May). RanDroid: structural similarity approach for Detecting ransomware applications in android platform. In *2018 IEEE International Conference on Electro/Information Technology (EIT)* (pp. 0892-0897). IEEE.

Baldwin, J., & Dehghantanha, A. (2018). Leveraging support vector machine for opcode density based detection of crypto-ransomware. In *Cyber threat intelligence* (pp. 107-136). Springer, Cham.

Continella, A., Guagnelli, A., Zingaro, G., De Pasquale, G., Barenghi, A., Zanero, S., & Maggi, F. (2016, December). ShieldFS: a self-healing, ransomware-aware file-system. In *Proceedings of the 32nd Annual Conference on Computer Security Applications* (pp. 336-347).

Gazet, A. (2010). Comparative analysis of various ransomware virii. *Journal in computer virology*, *6*(1), 77-90.

Hwang, J., Kim, J., Lee, S., & Kim, K. (2020). Two-stage ransomware detection using dynamic analysis and machine learning techniques. *Wireless Personal Communications*, *112*(4), 2597-2609.

Kharaz, A., Arshad, S., Mulliner, C., Robertson, W., & Kirda, E. (2016). {UNVEIL}: A large-scale, automated approach to detecting ransomware. In *25th {USENIX} Security Symposium ({USENIX} Security 16)* (pp. 757-772).

Kok, S.H, Azweem, A., Jhanjhi, N.Z. (2022). Early detection of crypto-ransomware using pre-encryption detection algorithm. Journal of King Saud University-Computer and Information Sciences, (pp. 1987-1993).

Kolodenker, E., Koch, W., Stringhini, G., & Egele, M. (2017, April). Paybreak: Defense against cryptographic ransomware. In *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security* (pp. 599-611).

Mohurle, S. & Patil, M. A Brief Study of WannaCry Threat: Ransomware Attack 2017. International Journal of Advanced Research in Computer Science, vol. 8, No. 5, 2017.

Njoroge, P.W. (2022). A Crypto-Ransomware Detection Model For The Pre-Encryption Stage Using Random Forest Algorithm (Masters Dissertation, Kenya College of Accountancy University)

Nieuwenhuizen, D. (2017). A behavioural-based approach to ransomware detection. *Whitepaper. MWR Labs Whitepaper*.

Oz, H., Aris, A., Levi, A., & Uluagac, A. S. (2021). A Survey on Ransomware: Evolution, Taxonomy, and Defense Solutions. *arXiv preprint arXiv:2102.06249*.

Rhode, M. (2021). *Racing demons: Malware detection in early execution* (Doctoral dissertation, Cardiff University).

Sgandurra, D., Muñoz-gonzález, L., Mohsen, R., Lupu, E.C., 3026. Automated dynamic analysis of ransomware: benefits, limitations and use for detection.

Shaukat, S. K., & Ribeiro, V. J. (2018, January). RansomWall: A layered defense system against cryptographic ransomware attacks using machine learning. In *2018 10th International Conference on Communication Systems & Networks (COMSNETS)* (pp. 356-363). IEEE.

Subedi, K. P., Budhathoki, D. R., & Dasgupta, D. (2018, May). Forensic analysis of ransomware families using static and dynamic analysis. In *2018 IEEE Security and Privacy Workshops (SPW)* (pp. 180-185). IEEE.

Takeuchi, Y., Sakai, K., & Fukumoto, S. (2018, August). Detecting ransomware using support vector machines. In *Proceedings of the 47th International Conference on Parallel Processing Companion* (pp. 1-6).

Tariq, M. I. A Review of Deep Learning Security and Privacy Defensive Techniques, Mobile Information Systems, vol. 2020, 2020.

Taylor, M. (2017). *Ransomware Detection Using Machine Learning and Physical Sensor Data*. Southern Methodist University.

Wecksten, M., Frick, J., Sjöström, A., & Järpe, E. (2016, October). A novel method for recovery from Crypto Ransomware infections. In *2016 2nd IEEE International Conference on Computer and Communications (ICCC)* (pp. 1354-1358). IEEE.

Zhang, H., Xiao, X., Mercaldo, F., Ni, S., Martinelli, F., & Sangaiah, A. K. (2019). Classification of ransomware families with machine learning based on N-gram of opcodes. *Future Generation Computer Systems*, *90*, 211-221.