On Ransomware Family Attribution Using Pre-Attack Paranoia Activities

Article *in* IEEE Transactions on Network and Service Management · September 2021

DOI: 10.1109/TNSM.2021.3112056

7 authors, including:

Elias Bou-Harb
University of Texas at San Antonio
120 PUBLICATIONS **1,857** CITATIONS
SEE PROFILE

Nizar Bouguila
Concordia University Montreal
478 PUBLICATIONS **5,790** CITATIONS
SEE PROFILE

Chadi Assi
Concordia University Montreal
443 PUBLICATIONS **8,589** CITATIONS
SEE PROFILE

On Ransomware Family Attribution Using Pre-Attack Paranoia Activities

Ricardo Misael Ayala Molina*, Sadegh Torabi†, Khaled Sarieddine*, Elias Bou-Harb‡, Nizar Bouguila*, and Chadi Assi*
*The Security Research Centre, Concordia Institute for Information Systems Engineering, Concordia University, Montreal, Canada
†The Center for Secure Information Systems, George Mason University, Fairfax, VA, United States
‡The Cyber Center For Security and Analytics, University of Texas at San Antonio, San Antonio, United States

***Abstract*—Ransomware attacks are among the most disruptive cyber threats, causing significant financial losses while impacting productivity, accessibility, and reputation. Despite their end goals (encryption/locking), ransomware are often designed to evade detection by executing a series of pre-attack API calls, namely "paranoia" activities, for determining a suitable execution environment. In this work, we present a first-of-a-kind effort to utilize such paranoia activities for characterizing ransomware distinguishable behaviors. To this end, we draw-upon more than 3K samples from recent/prominent ransomware families to fingerprint their uniquely leveraged paranoia activities. Specifically, by leveraging techniques rooted in Natural Language Processing (NLP) such as Occurrence of Words (OoW), we model ransomware-generated evasion API calls while tailoring various machine and deep learning algorithms to perform ransomware classification. The thoroughly conducted evaluations demonstrate the effectiveness of the implemented approach, with the Random Forest (RF) and OoW techniques producing an optimal classification accuracy (94.92%). The insights/findings from this work not only shed light on contemporary ransomware-specific evasion methods, but also (i) indicates that such tactics could be employed effectively as features for ransomware family attribution while (ii) laying the foundation for implementing proactive and portable countermeasures for further ransomware attack detection/mitigation by solely utilizing ransomware-generated paranoia activities.**

***Index Terms*—Ransomware API, Ransomware analysis, Cyber forensics, Machine/deep learning.**

I. Introduction

RANSOMWARE is an extortion-type of malware that is designed to encrypt user information and/or lock targeted devices [1–3]. The main objective of such attacks is to request a ransom in exchange for releasing/decrypting user data and/or unlocking the affected devices. The past few years have proved ransomware as a major threat towards electronic assets (e.g., data and machines), where individuals, governments, companies, hospitals, and various industries have fallen victims of this menace [2]. For instance, recently, in May 2021, Colonial Pipeline, which is one of the most important companies in the US, operating the most extensive US fuel pipeline on the east coast, suffered a ransomware attack [4, 5], forcing this organization to stop its normal operations, work offline, and pay 4.4 M to free their hijacked resources. Furthermore, in the same month, JBS, the world's largest meat company and which supplies more than 20% of all beef in America, had to pause their typical operations in the US due to a ransomware attack [4], besides paying 11M to liberate their captured resources. Moreover, during the ongoing COVID pandemic, several hospitals across the U.S. were targeted by ransomware attacks, which were noted as an imminent and increased cybercrime threat by the U.S. Cybersecurity and Infrastructure Security Agency (CISA) [6].

Further, such attacks have been targeting various industries in the U.S. and around the world, causing huge security implications and concerns for involved parties [7, 8]. More importantly, the introduction of Ransomware-as-a-Service (RaaS) along with digital cryptocurrencies (e.g., bitcoins), which preserve attackers' anonymity [2], have turned ransomware into a lucrative tool for attackers who look for financial gains. Consequently, the frequency and cost incurred by ransomware attacks have been on the rise in recent years, resulting in billions of dollars worth of losses throughout the data-dependent industries and users [9, 10].

Researchers have proposed several approaches to detect and prevent ransomware attacks by applying different static/dynamic analysis methods to understand ransomware's code structure, actions that are done during its life cycle, and its post-infection behaviors [1, 2, 11]. Meanwhile, researchers leveraged several machine learning and deep learning techniques to stop this menace [12–15]. Despite the previous work, defending against the increasing number of ransomware attacks is considered as a challenging task due to the lack of knowledge about the newly detected ransomware and the constantly evolving families/variants [15, 16]. Indeed, the frequency and impact of such attacks demonstrate the lack of effective measures for ransomware detection and mitigation. Therefore, to address these challenges, there is a need to investigate effective approaches for detecting and mitigating ransomware attacks by analyzing their behavioral characteristics and attributing them to known malicious families/variants whenever possible.

The analysis of recently detected ransomware families showed that they are programmed to evade detection by executing a sequence of functions and operations, which aim at sensing the execution environment prior to executing the malicious payload [12, 16]. Such detection evasion techniques are devised by performing a series of pre-attack API calls to fingerprint the environment and avoid execution in a virtual environment. These pre-attack activities, which are known as "paranoia" activities, are hypothesized to present distinguishable behavioral characteristics that can be used to classify ransomware families, and thus, achieving the first steps towards identifying and mitigating such threats.

Motivated by the prevalence of the pre-attack paranoia activities among the detected ransomware, in this paper, we analyze over 3,000 real instances of ransomware belonging to five prominent families. We execute these malicious executables in a controlled environment to capture their pre-attack behavioral characteristics, namely paranoia activities [17]. It is important to note that the analyzed ransomware might not execute their full attack in the deployed analysis environment. Nevertheless, their pre-attack paranoia activities can still be captured and recorded for further analysis. We leverage Natural Language Processing (NLP) techniques to represent the obtained activities and extract behavioral features. Indeed, our analysis of the invoked evasion API calls indicates that ransomware families produce distinguishable fingerprints in terms of the invoked API calls and their sequence/frequencies.

We leverage such fingerprints for large-scale ransomware classification and family attribution using machine and deep learning models. Our analysis results demonstrate the effectiveness of the proposed approach with classification accuracy that is better or comparable to the state-of-art approaches. Specifically, the implemented RF model produced the highest accuracy, as compared to other ML/DL models. Finally, while this work sheds light on a new approach for effective characterization and classification of ransomware families, it provides means for building preemptive defenses against ransomware attacks using solely their paranoia activities, which is necessary to safeguard devices from the risk of getting infected.

In this context, we frame the main contributions of this research as follows:

- We perform dynamic malware analysis on a large corpus of real ransomware extracted from recent prominent families to build a better understanding about their behavioral characteristics through the analysis of their paranoia activities represented by their pre-attack API function calls. Indeed, our analysis highlights distinguishable behavioral characteristics in terms of the frequency of API calls, the sequence of such evasion APIs, and the presence/absence of them in the majority of ransomware families.
- We motivate, through this work, the need for early detection of ransomware while the malware sample goes through its cyber-kill-chain levels, thus mitigating the risk of encrypting the user's data. We propose a ransomware family attribution approach by implementing effective ML/DL classifier models that utilize the unique characteristics of the obtained ransomware paranoia activity (pre-attack evasion API function calls). Our analysis using various tailored ML/DL models demonstrates the effectiveness of the approach to attribute suspicious behaviors to known ransomware families (about 95% accuracy), which can trigger early detection/mitigation countermeasures.
- We perform an in-depth analysis of the implemented ML/DL classifiers and evaluate/compare their performance using various measures such as processing speed, accuracy, F-measure, precision and recall.
  Additionally, we explore the effectiveness of the implemented model and compare their overall outcomes to identify the most effective combinations of classifier and feature representations. Finally, we discuss the practical implications and cyber-security benefits of our approach, which can be utilized to build effective countermeasure for detecting ransomware attacks and preventing them from causing tangible damage.

The rest of the work is organized as follows:In Section II, we introduce related works in the context of ransomware family classification. The proposed methodology is detailed in Section III, including data collection, dynamic analysis, feature extraction/selection, classifier implementation/evaluation, and limitations of the proposed methodology. Section IV describes the conducted experiments and the results obtained from each classifier, discussing them simultaneously. Section IV-D presents a comparison between the classifiers' attained outcomes. Section V discusses the main findings, limitations, future endeavors, applications of our framework in the real world, and information about the robustness of our approach. Finally, the work is concluded in Section VI.

II. Background and Literature Review

A. Ransomware Classification

A number of research has utilized static malware analysis techniques for extracting features that can be used along with ML/DL methods for the purpose of ransomware detection and classifications. For instance, Zhang et al. [16] conducted comprehensive experiments on real-world datasets by calculating the Term Frequency-Inverse document of the N-grams that are transformed from ransomware opcodes. They performed analysis with different N-gram feature dimensions and evaluated their approach using five different machine learning techniques. Similarly, the authors in [12] leveraged opcodes for their classification models using extracted N-grams features. However, while they implemented multiple ML classification models, they also utilized DL models such as the Self-Attention Convolutional Neural Network (SA-CNN) to test their framework, which worked well for long opcode sequences. From a different perspective, Subedi et al. [13] utilized static and dynamic analysis techniques to perform forensic investigation of ransomware families by analyzing 450 ransomware samples. Specifically, they applied data mining techniques to the components derived from reverse engineering processes (i.e., assembly instructions, DLL libraries, and function calls) to find unique association rules, which can characterize ransomware samples and perform family attribution.

Despite the presented results in the aforementioned papers, they suffer from a number of limitations. For instance, the generalizability of the results might be hampered by the limited number of analyzed ransomware samples, which might not be representative of the overall ransowmare threat landscape (Table I). We try to address this limitation by analyzing a more representative number of ransomware samples from 5 predominant families. To the best of our knowledge, our proposed work represents experimentation with the largest empirical dataset of ransomware samples. Moreover, Subedi et al. [13] did not implement any ML/DL classifications, which may limit their forensic investigation findings to the studied data. Additionally, their approach might not scale to large

Table I: Summary of the state-of-the-art ransomware classification results from literature (NA: Not Applicable, RF: Random Forest, DT: Decision Tree, NB: Naive Bayes, LR: Logistic Regression).

	Reference	Approach	Classifier	Ransom./ Malware	Features	Accuracy
Ransomware Classification	H. Zhang et al. [16]	Static	DT, KNN, RF, NB, GBDT	1,787	N-grams	91.43%
	B. Zhang et al. [12]	Static	DT, KNN, NB, SA-CNN	1,887	N-grams	89.50%
	Subedi et al. [13]	Static/Dynamic	NA	450	Assembly instruction, PE libraries, function calls	NA
	Hajredin et al. [15]	Dynamic	NB, J48 DT, KNN	150	12 features extracted from `VirusTotal` reports	78%
	Vinayakumar et al. [14]	Dynamic	LR, NB, DT, RF, KNN, SVM, MLP	974	131 APIs and their frequencies	98%*
Malware Classification	Onwuzurike et al. [18]	Static	RF, SVM, 1-NN, 2-NN, 3-NN	44K	10K API calls	99%
	Jiaqi et al. [19]	Static	DG-CNN	20K	Control Flow Graphs	99.25%
	Suarez et al. [20]	Static	Extra Trees	100K	Obfuscation induced Artifacts	99.26%
	Dash et al. [21]	Dynamic	SVM	5,246	System Calls	94%
	Cai et al. [22]	Dynamic	RF, SVM, Naive Bayes Decision Trees, kNN	34,343	Method Calls	97%
	This work	**Dynamic**	**Bernoulli NB, KNN, RF ANN, LSTM, Bi-LSTM**	**3,432**	**23 evasion APIs (Paranoia Activities)**	**94.92%**

* We discuss the main limitations of this work, which affects the reliability of the reported results in Section II.

number of analyzed ransomware samples that represent various families. Furthermore, while both approaches implemented in [12, 16] result in reasonable classification accuracy (91.43% and 89.50%, respectively), they rely on static malware analysis techniques to extract N-grams and assembly code features for their classifiers, which is extremely difficult to obtain when dealing with the predominantly obfuscated ransomware samples that are difficult to reverse-engineer. Conversely, in order to defeat the ransomware's obfuscation problem, we use dynamic analysis techniques to capture their pre-attack environment sensing actions, which represent distinguishable features for effective classification and threat detection, as detailed in Section III-B).

In line with that, some researchers relied on dynamic malware analysis techniques to obtain behavioral characteristics that are used for further ransomware detection and classification. For instance, Hajredin et al. [15] proposed a behavioral classification method by analyzing 150 ransomware samples from 10 different families. The authors relied on collected behavioral reports from `VirusTotal` to extract 12 behavioral attributes/features that were used with three implemented ML-based classifiers, respectively. Surprisingly, the authors used a very small number of ransomware samples to test and evaluate their implemented models, which undermines the overall generalizability of the findings. Moreover, they reported a relatively low classification accuracy (about 78%), which may raise further questions regarding the overall effectiveness of the approach for ransomware detection and classification.

Additionally, we found the work done by Vinayakumar et al. [14] to be closely related to our approach, where they leveraged invoked API calls and their frequencies to characterize ransomware samples while evaluating the effectiveness of a series of shallow and deep learning techniques for classifying ransomware families and distinguishing them from benign applications. Specifically, they collected 974 ransomware from 7 different families, while identifying 131 invoked API calls and their frequencies to evaluate their implemented classifiers. Despite their significant reported results (Table I), our thorough analysis of their work highlighted a number of major limitations, which significantly hamper the quality and reliability of the presented results. For instance, the work lacks adequate details and justifications about the executed methodology in terms of the data collection, analysis, feature extraction, and model evaluation. Moreover, while the relatively small number of ransomware samples are not representative of the overall population, it is not clear how the studied samples were selected among all existing ransomware data. Additionally, the choice of 131 invoked API calls and their effect in relation to the selected samples on the overall classification results is not discussed in the work. As a result, these aforementioned major limitations raise fundamental questions about the threats associated to the validity and generalizability of the reported findings. In other words, the reported significant classification results might be limited to their dataset and not necessarily be valid/reliable when used with other datasets/features.

B. Malware Classification

While there is a limited number of previous work related to ransomware classification, malware detection/classification has been extensively studied in recent years. For instance, Android malware classification has been discussed in several previous work. Specifically, Onwuzurike et. al. [18] developed a static-analysis-based tool that utilizes Markov chains to represent the sequences of abstract API calls, which are used in the classification process. Jiaqi et. al. [19] utilized semantic and structural dependencies in the control flow graph (CFGs) to build a deep graph convolutional neural network for malware classification. Suarez et. al. [20] proposed a fast, reliable, and obfuscation-resistant Android malware classifier based on static analysis. This framework takes advantage of functionality and objects introduced by obfuscation mechanisms and are obfuscation-agnostic. Dash et. al. [21] introduced a classification approach

that leverages Support Vector Machines with Conformal Prediction to generate high-accuracy malware family prediction using features extracted from runtime behaviors. In line with that, Cai et al. [22] employed dynamically extracted features such as method calls and inter-component communication to overcome the challenges associated with previous approaches and achieved a more robust malware classification outcome.

Despite the promising malware classification outcomes that were presented in [18–20], the main limitation of such approaches is when dealing with malware obfuscation, which hinders the collection of features and thus, affecting the robustness of such approaches. Moreover, while Suarez et al. [20] utilize obfuscation artificats in their approach, we cannot be sure if the mechanism is detecting malicious activity or just obfuscation, which do not convey the actual intent of the software sample. Moreover, the amount of data stored in the continuous features used by their approach leads to the exponential growth of the dataset, which hampers the scalability of the approach. Finally, the detection approach presented in [21] relies on different metrics that are collected throughout the cyber-kill-chain, which increases the risk malware (ransomware) execution before its detection.

C. Overall Comparison to Our Approach

In this paper, we highlight the lack of previous work towards ransomware classification and family attribution using paranoia activities. More importantly, we demonstrate the overall effectiveness of the proposed approach towards ransomware classification and family attribution, as compared to previous work. Indeed, we show that our approach, which was tested with a significantly larger number of ransomware samples, can achieve better or comparable classification accuracy, as illustrated in Table I. More specifically, we discuss a number of main limitations that makes our work more practical and effective towards early detection and mitigation of ransomware attacks. For instance, the work done in [21] and [22] falls short in terms of their ability to perform early detection of ransomware attacks since they rely on system/method calls that are invoked during or after the execution of the attack. Moreover, despite the high-level similarity of the approach presented in [22] to our work in terms of capturing the behavioral characteristics of the malicious executable, their detection mechanism depends on the whole behavioral characteristics throughout the cyber-kill-chain. Whereas, our approach requires the pre-attack behavioral characteristics only, which allows early detection before the execution of the actual attack.

Additionally, as presented in Table I, our classification approach relies on extracting behavioral features through observing 23 evasion API calls, which are associated with malicious ransomware activities. More importantly, we utilize a significantly smaller number of features as compared to other approaches such as [14] and [18], which use 131 and 10K API calls, respectively. This indeed, gives us a competitive advantage over previous work where we can guarantee high accuracy with fewer behavioral features, which can improve scalability by reducing training time and the required computational resources (e.g., memory and CPU). Finally, our approach can be utilized in practice to detect the ransomware after delivery and before the execution of the malicious payload. More importantly, our approach can be leveraged for research and analysis purposes as it can still capture the pre-attack behaviors even if the ransomware detects the analysis environment and abort its intended operations/attacks.

It is worth noting that we make our used dataset and all the developed methods publicly available to the research community at large to facilitate reproducibility of the obtained results.[1]

III. Methodology

Given the limited number of previous work, which discussed ransomware classification and family attribution, in this paper, we aim at leveraging pre-attack paranoia activities to characterize various ransomware families to implement effective ML and DL classification approaches. More specifically, the devised methodology attempts to answer the following research questions (RQs):

1) *What are the main behavioral characteristics of different ransomware families in terms of their paranoia activity?*
2) *How can we leverage the identified behavioral characteristics to propose an effective ransomware classification approach?*

To achieve our objectives, we follow the proposed approach, which consists of three main components, as presented in Figure 1. We collect 19,499 ransomware samples from multiple resources such as `VirusShare` and `VirusTotal`. Furthermore, we perform dynamic malware analysis on a randomly selected sample, which represents 5 different ransomware families. We execute the ransomware in a controlled environment using `Cuckoo` sandbox [23] to extract their behavioral features and characteristics. Finally, we leverage the extracted/curated features to implement various ML/DL classification models while evaluating them using de-facto methods. In what follows, we present further details about our data collection and analysis methodologies.

A. Data Collection and Preprocessing

In this work, we leveraged multiple resources to obtain real instances of ransomware. We obtained a total of 129,500 malware samples that were detected between 2010 and 2019. More specifically, we obtained 90,364 malware samples from `VirusTotal` [24] (2017-2018), in addition to 39,136 samples from `VirusShare` [25] (2010-2016, and 2019). As depicted in Figure 2, we utilized AVClass [26] to label ransomware samples in our data. AVClass is a tool that leverages the API of `VirusTotal` to identify malware family names/labels. The original dataset contains various malicious software and malware families such as Mirai, Locky, Nemucod, Ramnit, Phish, Pdfka, Redir, and Faceliker, to name a few. We followed the processes depicted in Figure 1 (i.e, Labeling Malware Samples and Filtering Samples) to extract 19,499 ransomware samples from the original malware samples that belong to 21 ransomware families. As shown in Table II,

[1] https://github.com/Rmayalam/Ransomware_Paranoia.git

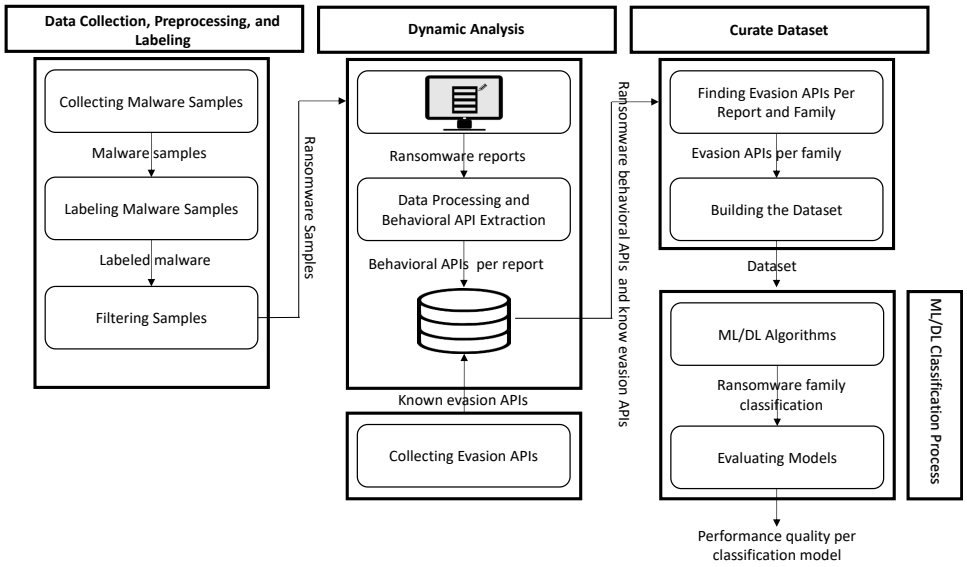

Figure 1: The overall workflow of the proposed methodology.

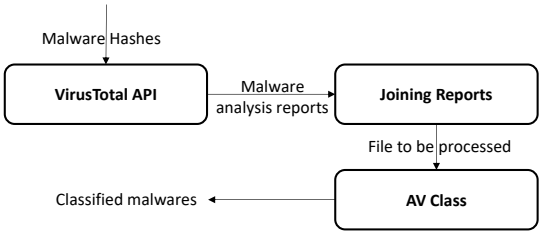

Figure 2: Initial labeling of the ransomware samples in the harvested data using AVClass.

the distribution of the identified ransomware samples across the top 10 families indicates that Reveton, Xorist, and Locky correspond to the largest number of samples, respectively.

Table II: Distribution of samples across ransomware families.

Family	Samples	%
Reveton	4,374	22.43
Xorist	3,636	18.65
Locky	2,887	14.81
Teslacrypt	2,846	14.60
Bitman	1,045	5.36
Cryxos	1,020	5.23
Yakes	922	4.73
Cerber	594	3.05
Urausy	570	2.92
Razy	567	2.91
Others	1,038	5.32

Given the imbalanced nature of the collected data, and the fact that some ransomware samples have been widely utilized to perform large-scale attacks [27, 28], we perform our experiments with the following five main ransomware families: Reveton, Locky, Teslacrypt, Yakes, and Cerber. Note that we selected these ransomware families due to the fact that there is a lack of ransomware repository that manages a balance distribution with a considerable number of samples for each possible ransomware family. Consequently, considering the different number of samples from the selected ransomware families, we applied an undersampling technique to obtain a balanced dataset and generate consistent results in our experiments. It is worth noting that our ransomware dataset is the largest that has been employed until now to realize ransomware family classification. In general, these ransomware families represent crypto and/or locker ransomware, which encrypts data files on a host or locks the targeted device/resources without encrypting its content/data [3]. Additionally, we explore and create a benign dataset by collecting over 4.5k executable samples from a new installation of Windows 10 operating system.[2]. We leverage Cuckoo Sandbox to explore possible paranoia activities generated by each sample. Our analysis resulted in over 1.4k benign samples that perform some sort of fingerprinting to recognize their execution environment. These benign applications/software include utilities (e.g., Zoom, Windows Media Player, Anydesk, and WinZip), web browsers (e.g., Google Chrome), and windows drivers (e.g., msiexec, msconfig, and NetInstaller). Overall, we use 6 selected classes (5 ransomware families and a benign dataset) in our experimental analysis throughout the paper.

B. Dynamic Analysis

As shown in Figure 1, we follow a dynamic malware analysis approach to capture the pre-attack paranoia activities generated by the ransomware. We leverage a common characteristic of the analyzed ransomware, which tend to evade detection by invoking a number of API calls (paranoia activities) to realize the existence of dynamic analyzers (e.g., Cuckoo Sandbox) before executing its attack payload. In practice, this is considered as a limitation of existing analyzers. Nevertheless, we take advantage of such malicious pre-attack activities to detect the presence of a ransomware and characterize its behaviors, while

[2]Windows is the most ransomware-targeted OS [1, 29] We collected the benign samples to distinguish between them and the ransomware families in the different classification models utilized in this work.

Table III: Different categories of inferred evasion APIs and their description.

Category	Evasion API	Description
Generic OS queries	DeviceIoControl	Retrieve information concerning a hard disk.
	GetComputerNameA	Retrieve the NetBIOS name of the local computer.
	GetDiskFreeSpaceExW	Retrieves information about the available space on a disk volume.
	GetComputerNameW	Retrieve the NetBIOS name of the local computer.
	GetSystemInfo	Retrieve information about the current system.
	GetSystemMetrics	Retrieve the specified system metric or system configuration setting.
	GetUserNameA	Retrieve the name of the user associated with the current thread.
	GetUserNameW	Retrieve the name of the user associated with the current thread.
	NtQuerySystemInformation	Retrieve the specified system information.
Global OS Objects	NtCreateFile	Create a new file or directory, or opens an existing device, file or directory.
	NtOpenDirectoryObject	Open an existing directory object.
Hooks	ReadProcessMemory	Read memory.
	SetWindowsHookExA	Installs an application-defined hook procedure into a hook chain.
	SetWindowsHookExW	Installs an application-defined hook procedure into a hook chain.
Network	GetAdaptersAddresses	Retrieve the addresses associated with the adapters on the local computer.
OS features	CreateToolhelp32Snapshot	Take a snapshot of the processes, memory heaps, modules, and used threads.
Processes	LdrGetProcedureAddress	Retrieve the address of an exported function or variable from the specified dynamic-link library (DLL).
Registry	NtEnumerateKey	Enumerate sub keys of an open key.
	NtClose	Close the specified handle.
	NtOpenKey	Open a registry key.
	NtQueryValueKey	The ValueName, type, and data for any one of a key's value entries may be queried with this API.
	RegCloseKey	Close a handle to the specified registry key.
UI artifacts	EnumWindows	Enumerate all top-level windows on the screen.

utilizing extracted features for further family classification and attribution. More importantly, our analysis completely relies on capturing the pre-attack activities of the ransomware and will not be affected by its consequent behaviors, which is key for building effective tools for early detection/prevention of ransomware attacks.

We execute the selected ransomware samples and analyze their behavioral characteristics by configuring Cuckoo Sandbox 2.0.7 in a virtual environment (e.g., VirtualBox 6.1 with Windows 7 operating system), leveraging its modularity to avoid possible crashes. Cuckoo Sandbox [23] is one of the most widely used tools for analyzing the behaviors of a diversity of malware. The ransomware is run in a controlled virtual box to capture all performed activities during its execution (e.g., API calls, files opened, registry keys, etc.). These activities, which represent the behavioral dynamics of the ransomware, are captured in a comprehensive report generated by Cuckoo. Along with that, the report contains information about the analysis such as machine name, operating system, Internet access, along with additional static and signature-based analysis outcomes. Given the generated reports, we extract a list of behavioral API calls per analyzed sample (e.g., GetNativeSystemInfo, LdrUnloadDll, NtOpenKey, NtClose, etc.), which we use for further analysis, as detailed in the following sub-sections.

C. Inferring Evasion API Calls

As stated by Yan et al. [30], extracting features has a vital role in a classification malware task. Therefore, one of the most important phases in this work is to identify API calls that are used by the ransomware to evade detection (i.e., paranoiac activities). We postulate that ransomware samples from each family will generate distinguishable footprints in terms of such API calls, and thus, can be used as a behavioral feature for further classification of ransomware samples. Given the list of behavioral API calls per ransomware sample, we utilized a number of known/common detection evasion API [31–33] that are used by malicious executables to identify the sample-specific evasion API calls. After that, we analyze the reports to identify all evasion APIs that were invoked across the selected ransomware families. To this end, we identified 23 unique evasion APIs that were used by the analyzed samples. As summarized in Table III, we categorize the identified APIs into 8 different categories, which reflect API calls (paranoia activities) that are normally invoked to sense the execution environment and avoid detection.

D. Feature Extraction

Given the obtained dataset of evasion APIs coupled with their frequencies per ransomware samples, we adopt three feature selection/representation approaches: Occurrence of Words (OoW), Bag of Words (BoW), and Sequence of Words (SoW).

OoW is a flexible technique that can be used to extract features from text documents and has demonstrated great success with machine learning applications to classify documents. This technique does not take into consideration the order or the structure of the words. It only contemplates the words' presence/absence by a binary count, where 1 means present and 0 means absent. Moreover, we employ the BoW technique to capture the frequency of the API calls. Both OoW and BoW were proved to be effective when used with statistical machine learning models such as the Bernoulli Naive Bayes [34], KNN algorithms [35], or RF [36].

While these techniques were adopted to capture the frequency and the occurrence of evasion APIs, they do not take into account the sequence of API calls nor do they conserve

their order. Therefore, we also considered the SoW method. Since most of the API calls are repetitive, sometimes indefinitely, we ought to choose unique API calls while conserving the order in which they were called by each sample. Using this technique, we aim at decreasing our feature space from millions to a sequence of unique APIs.

E. Classification Models

To this end, we implement and evaluate 5 different ML/DL classification models. Specifically, we use Bernoulli Naive Bayes and KNN algorithms to capture the occurrence and frequency of evasion API calls without considering the order and structure of the calls. Moreover, to capture the order of the API calls, we chose Artificial Neural Network (ANN), Long Short-Term Memory (LSTM), and BiDirectional LSTM models to perform further classification of ransomware samples.

Bernoulli Naive Bayes: First, we adopt the Bernoulli Naïve Bayes, a probabilistic classifier that predicts the probability of an input belonging to a specific class. It uses binary variables such as 1/0 or true/false to represent the occurrence or absence of features, making it quite fast in contrast to other classification models. Moreover, when the assumption of class conditional independence is fulfilled, this classifier performs better than others [34]. Furthermore, it has been demonstrated that this probabilistic method presents a good performance in text classification [37].

K-Nearest Neighbors (KNN): Additionally, we use the K-Nearest Neighbors algorithm, a supervised machine learning model that can be leveraged for multi-label classification and has demonstrated its effectiveness in different classification tasks [38–40]. The KNN algorithm is based on the feature similarity approach, where it classifies new samples based on a selected similarity measure (e.g., distance functions) [41, 42].

Random Forest (RF): Moreover, we test random decision forest which is an ensemble learning method mostly used for classification. It operates by constructing a multiple decision trees at training time and then average voting to select the class of the sample at hand. RF is widely adopted in the malware classification realm, due to its easily understood structure while maintaining a high accuracy [14, 16].

Artificial Neural Network (ANN): This method mimics the biological neural network that constitute the brain and how it operates. ANN represents a system of interconnected neurons, or nodes with a nonlinear mapping between the variable input and the output vector. The nodes are connected by weights and output signals, which are a function of the sum of the inputs to the node modified by a simple nonlinear transfer, or activation function [43]. The ANN is a simple deep learning algorithm that is used to classify and extract intelligence from a set of features similar to the works done in [14, 44].

Long-Short Term Memory (LSTM): We also leverage a special type of Recurrent Neural Networks, namely the Long Short Term Memory (LSTM). RNNs represent powerful and robust artificial neural networks that use sequences to evolve models that simulate the neural activity in the human brain. Some of the applications of RNN is to recognize patterns in sequences of data such as texts and handwriting, to name a few. However, RNN often suffers from gradient explosion that occurs through exponential growth by repeatedly multiplying gradients through the network layers. RNNs apply weights to the current and previous inputs. They also tweak their weights through gradient descent and back-propagation. LSTMs preserve the errors that will be back propagated through layers. Maintaining a constant error allows LSTMs to continue learning over many time steps. LSTM is unique in their capability to learn what information to store in long term memory. It also uses gates to control the learning and memorization process (learns what to keep or get rid of). LSTM also allows the neural network to identify patterns and sequences in the data by learning spatial and temporal relationships between features while utilizing memory gates. To capture the temporal order between the API calls, we adopt LSTM and its variants, which have been widely used for text-based data classification [45].

Bidirectional Long Short Term Memory (Bi-LSTM): We further explore a special type of bidirectional RNN, namely the bidirectional LSTM, to overcome a major limitation of the unidirectional RNNs, which rely only on learning from previous time steps. Note that the Bi-LSTM addresses this limitation by learning representations from future time steps along with previous ones. This helps to better understand the content and eliminate ambiguity. In some cases, we would need to look ahead to better understand and identify sequences. The Bi-LSTM will take the input in normal order, and at the same time it will take the same input and reverse it to allow for forward and backward learning from the feature space [46].

F. Model Evaluation and Comparison

We follow a number of standard methods to evaluate the overall effectiveness of the implemented classification models to compare their outcomes. More specifically, we use metrics such as accuracy, recall, precision, and F-measure. Further, we use the confusion matrix, which is a useful method for discussing the effectiveness of the implemented ML/DL models, where true positive and true negative specify the number of samples that the model has classified correctly, while false positive and false negative state the number of samples that were misclassified. In addition, we compute the speed of each model as measure of their computational performance.

G. Limitations

In this work, we encountered (and had to manage) a number of limitations in terms of data collection and analysis. First, while we leveraged several public threat repositories (i.e., `VirusShare` and `VirusTotal`) to obtain a large number of real ransomware instances from different families, the obtained dataset was not balanced due to the different number of ransomware samples per family. To address this, we applied undersampling by (i) scoping down on 5 ransomware families with the most recent attack incidents, and (ii) randomly selecting a subset of malicious executables from the selected families to create a balanced dataset. Second, we rely on AVClass and `VirusTotal` APIs to obtain the ransomware family

names and create our labeled dataset. It is important to realize that such labels might not necessarily reflect accurate class labels, and thus, might hamper the classification results. Despite that, the internal mechanics of AVClass, and the implemented majority rule system seems to be effective in identifying reliable labels, especially when the malicious executables are detected and analyzed by many AV vendors over time. Finally, we extract and rely upon a list of known/common evasion API calls to characterize the behaviors of the ransomware and capture its paranoia activities. Nevertheless, ransomware could implement other techniques that might not be reflected within such API calls, thus hindering the classification accuracy of the proposed approach. Our empirical evaluation demonstrates that the analyzed APIs are highly accurate and that they can easily be extended in the future to mitigate any current shortcomings.

IV. Experimental Results

As described in Section III-A, we focus on 5 main ransomware families for our experimental analysis. Furthermore, as noted, given the imbalance dataset in terms of the sample distribution per family (Table IV), we applied undersampling by randomly selecting a subset of samples from the selected families to create a balanced dataset. Moreover, we chose benign applications as the sixth family class in our experiments, where 30% of them were shown to invoke one or more evasion API calls. The remaining benign application, which did not invoke any evasion API calls were omitted from further analysis. Similarly, while most of the ransomware samples (99.5%) in each family invoked one or more evasion API call, a small percentage of them did not use any evasion API calls, and thus were also removed from further analysis. We deem that these small percentage could be easily detected (and attributed) by other complementary methods, given that they do not leverage evasive methods. As summarized in Table IV, the remaining 3,432 executable samples, which belong to 6 groups (5 ransomware families and one benign), were used for further analysis throughout the work. Note that we select 600 samples from each family, except Carber and benign, were we ended up with 594 and 438 samples, respectively.

A. Feature Selection via Evasive APIs

Recall that we hypothesize that each ransomware family can be identified by its paranoia activity, which is represented by the used pre-attack API calls. To investigate our hypothesis, we summarize the invoked evasion API calls and their frequencies (on average) for the analyzed ransomware families, as shown in Figures 3a–3e. It is clear that samples within each ransomware family use a specified set of evasion API calls and frequencies, which can possibly be used as a distinguishable feature for further analysis. Moreover, despite the specificity of the used evasion API calls per family, some evasion API calls were common across different families, as shown in Figure 4. For instance, `getSystemInfo` API call was invoked by all the analyzed samples. As previously noted, we leverage three feature selection/representation approaches (OoW, BoW, and SoW) to represent the obtained behavioral characteristics in terms of the unique evasion API calls per sample. In what follows, we utilize these three feature representation to test and assess our hypotheses by performing two sets of experiments: (i) Frequency and occurrence based, and (ii) Sequence-based.

Table IV: Number of ransomware samples per family (pre- and post-processing).

Family Name	Collected Data	Selected Samples
Reveton	4,374	600
Locky	2,887	600
TelesCrypt	2,846	600
Yakes	922	600
Cerber	594	594
Benign	1,419	438

B. Frequency and Occurrence-Based Experiments

We use Bernoulli Naive Bayes, KNN and, RF machine learning models with OoW and BoW techniques to capture the frequency and the occurrence of the evasion API calls per ransomware family. Moreover, we apply 10-fold cross-validation to avoid overfitting the models. Furthermore, we divided our dataset (3,432 samples) into training, validation, and testing sets. We leverage 80% of the samples for training purposes, whereas the remaining 20% were utilized for testing/validating the models. In general, the RF model produced the highest values for accuracy (94.61%) and F-measure (94.92%), respectively. In what follows, we present the analysis results with respect to the implemented machine learning models:

1) Bernoulli Naive Bayes (BNB): This statistical model was combined with the OoW technique to test our framework's performance to classify each ransomware sample considering only the presence or absence of the evasion APIs. The input data referred to as the "Training" set represent vectors composed of 1s and 0s where each binary value represents the presence or absence of specific evasion APIs. The length of these vectors is 23 positions, which correspond to the number of evasions APIs used in this work to classify each ransomware family. Numerical measures presented in Table V show the F-measure, recall, and precision per each family. The obtained F-measure score indicates that the classifier's quality is acceptable in general, because the value associated with each family is greater than 84%, except for the Benign and Reveton families.

The Benign family obtained a value of 62.50%; Indeed, this demonstrates that this family may require special attention, as its recall and precision measurements recorded 54.54% and 57.69, respectively. An observation concerning the Benign family was that some samples generate overlapping evasion APIs, which could explain why the Benign family presents low values in all their measurements. Moreover, the Reveton family displayed variations in their evasion APIs patterns, which also justifies the 72.73% of its F-measure and the 64% of its recall value. Furthermore, the Locky and Yakes families showed some differences in their behavioral paranoia activity, affecting the precision (79.76%) for the Locky family and the recall (74.07%) for the Yakes family.

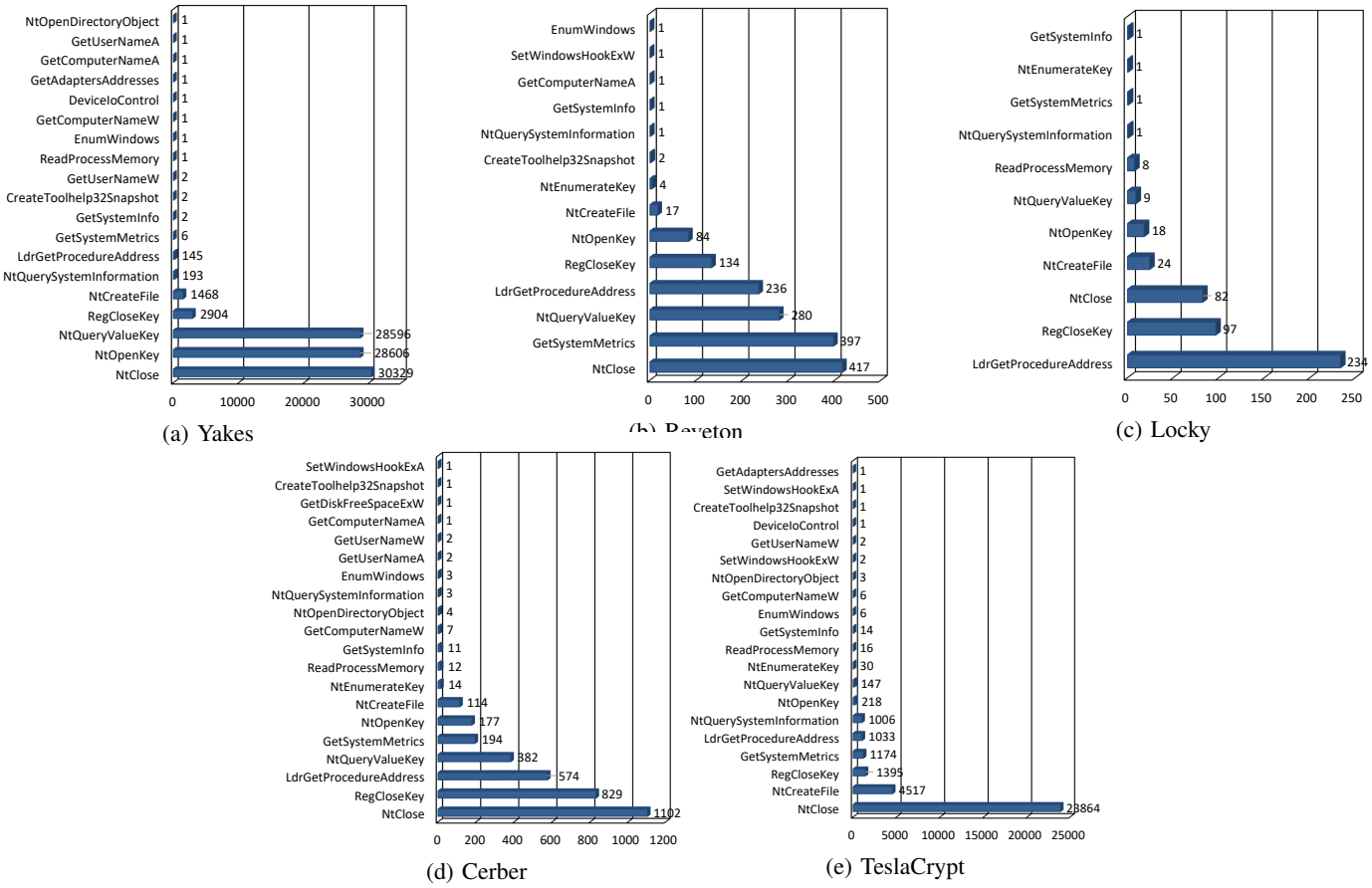

Figure 3: Average number of the invoked evasion APIs for the analyzed ransomware families.

On the other hand, the Cerber and TeslaCrypt families performed well with this model, which is reflected in their F-measure values. Table VI shows the confusion matrix's outcomes derived from this model. It is observed that the numbers presented in this matrix confirm that the classification model needs to pay special attention to the Benign and Reveton families, given that in the context of the Benign family, 14 out of 44 samples (31.82%) were not classified correctly, and for the Reveton family 18 out of 50 samples (36%) were not classified correctly. Further, the misclassification in the Yakes family samples are observed in this matrix, where about 25.9% of them were not classified correctly. Conversely, the remaining families demonstrated satisfactory performance. The Bernoulli Naive Bayes model produced an overall accuracy of about 85.11%, 256 ransomware samples out of 336 correctly classified, indicating relatively good classification outcomes.

In addition to the classification accuracy, we evaluated the performance of this classifier in terms of its speed. Our results shows that the model needed about 0.195 seconds to accomplish the ransomware family classification, indicating a timely process and good performance. Interestingly, this model was ranked second-best in terms of the speed, as compared to the rest of the tested models throughout the work (Figure 5). Nevertheless, Bernoulli Naive Bayes produced the lowest accuracy and f-score measures compared to the rest classifiers.

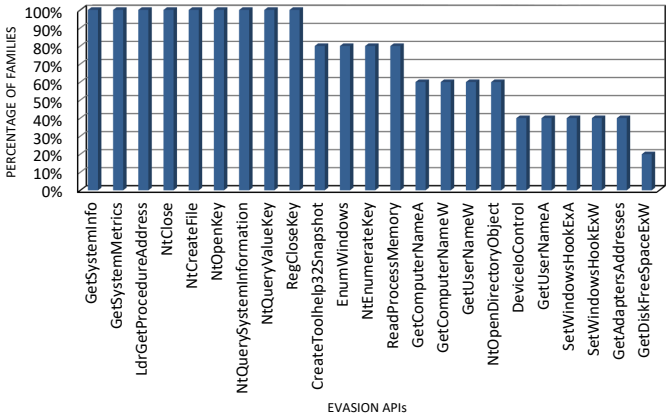

Figure 4: Common evasion APIs invoked by different ransomware families.

2) K-Nearest Neighbors (KNN): We use the KNN model with BoW to assess our ransomware family classification framework's performance, examining the evasion APIs coupled with their frequencies without considering their sequences. The input data of this classification model are vectors that contain the frequency of each evasion APIs. Like Bernoulli Naive Bayes, the length of these vectors is 23 positions, representing the numbers of evasions APIs utilized in this research.

Table V: Per-family evaluation results for BNB, KNN, and RF machine learning models.

	BNB			KNN			RF		
Family	**F-mea.(%)**	**Recall(%)**	**Precis.(%)**	**F-mea.(%)**	**Recall(%)**	**Precis.(%)**	**F-mea.(%)**	**Recall(%)**	**Presic.(%)**
Benign	62.5	54.54	57.69	90.7	88.64	92.8	87.80	81.81	94.73
Cerber	99.08	98.18	100.0	98.15	96.36	100.0	98.14	96.36	100
Locky	88.16	98.53	79.76	94.66	91.18	98.41	94.73	92.64	96.92
Reveton	72.73	64.0	84.21	84.0	84.0	84.0	99.01	100	98.03
TeslaCrypt	95.45	96.92	94.03	97.67	96.92	98.44	100	100	100
Yakes	84.21	74.07	97.56	89.83	98.15	82.81	88.13	96.29	81.25

Table VI: Confusion matrices' outcomes for BNB, KNN, and RF machine learning models.

	BNB						KNN						RF					
	B	C	L	R	T	Y	B	C	L	R	T	Y	B	C	L	R	T	Y
B	**30**	0	11	3	0	0	**39**	0	0	2	0	3	**36**	0	1	0	0	7
C	0	**54**	0	1	0	0	0	**53**	0	2	0	0	1	**53**	0	0	0	1
L	1	0	**67**	0	0	0	1	0	**62**	4	0	1	1	0	**63**	0	0	4
R	17	0	0	**32**	1	0	2	0	1	**42**	0	5	0	0	0	**50**	0	0
T	0	0	1	0	**63**	1	0	0	0	0	**63**	2	0	0	0	0	**65**	0
Y	4	0	5	2	3	**40**	0	0	0	0	1	**53**	0	0	1	1	0	**52**

B=Benign, C=Cerber, L=Locky, R=Reveton, T=TelesCrypt,Y=Yakes

Table V presents interesting outcomes for this model, its F-measure, recall, and precision scored over 90% on 3 families. It is worth mentioning that in this experimental setup the Benign and Reveton Families improved on all the measurement values compared to the previous model. The Benign family recorded 90.70% of F-measure, 88.64% of recall, and 92.86% of precision, while the Reveton family obtained 84% for all its measurements. In general, all the families showed a performance improvement with this classification model, which could be due to the fact that we use evasion APIs' frequencies instead of their presence/absence. As presented in Table VI, the KNN's confusion matrix results demonstrates improvement in the Benign and Reveton family classification results. For the Benign family, only 5 out 44 samples (11.36%) were misclassified, while for the Reventon family, 8 out of 50 (16%) were misclassified. The Yakes family also exhibits appreciable improvement because there exist one misclassified instance related to its labeling samples. Its F-measure and precision were affected by errors made with other families' samples classification. Overall, the implemented model's accuracy reached a high value of 92.85%, meaning 312 ransomware samples classified without mistakes, thus, representing 7.74% more effective than Bernoulli Naive Bayes' accuracy, confirming the effectiveness of the KNN model towards classifying this type of malware in our experiments.

Moreover, this classifier's computational performance in terms of speed was measured to compare it with the previous one. KNN spent 0.264 seconds to process and classify the ransomware samples. It means 1.35 times the Bernuille Naïve's. It is due to Bernoulli Naive Bayes's feature vector values, which were binary values (e.g., 0 and 1), while the KNN's feature vector values were integers from 0 to over 30,000. Nevertheless, KNN's performance measures outcomes are much better than the ones obtained from Bernoulli Naïve Bayes.

3) Random Forest (RF): Like Bernoulli Naive Bayes, RF was coupled with the OoW technique to obtain a vector of 23 binary features (e.g., 1s and 0) representing the identified specific evasion API shown in Table III. We present the performance analysis results for the RF model in Table V. It is observed that the F-measure, recall, and precision recorded values over 94% for four ransomware families (e.g., Cerber, Locky, Reveton, TeslaCrypt), indicating better accuracy as compared to the KNN model, which classified three families with values over 94% (e.g., Cerber, Locky, TeslaCrypt). We believe that this is due to the simplicity of the RF model and its internal logic when dealing with uncorrelated models (e.g., ensemble prediction) that reduce data variability. Furthermore, the classification results for Reveton improved notoriously with values better than KNN in all the three mentioned evaluation measures, recording 99.01%, 100%, and 98.03%, respectively. Moreover, the classification results for the Benign and Yakes families maintain a similar tendency as KNN.

We present the classification results for the RF model in the confussion matrix, as illustrated in Table VI. It is interesting to see how the Benign family keeps its trend pattern through the models analyzed until this moment, with only 8 misclassified samples out of all 44 benign samples (18%), confirming the overlapping pattern generated for this type of software. On the other hand, the rest of the families presented a high percentage of true positives, representing correctly classified samples. In general, this model scored an accuracy value of 94.92%, with 319 correctly labeled ransomware samples, hence, slightly better than the KNN's in terms of accuracy. Finally, as illustrated in Figure 5, the RF model required 0.206 seconds to accomplish the classification process. Therefore, while RF produced the highest accuracy so far, it was slightly slower in terms of processing time as compared to the BNB, but faster than KNN.

C. Sequence-Based Experiment

In this section, we use SoW feature representation by taking the sequence/order of the generated API calls into consideration. We also implement several deep learning models (e.g.,

Table VII: Optimized hyper-parameters for the implemented models.

	Learning Rate	Drops	Decay	Batches	Units1	Units2	Epochs
Neural Network 1	0.00117094	0.143239267	0.000402171	44	289	132	13
Neural Network 2	0.001075841	0.152889602	0.000408151	46	259	115	12
LSTM 1	0.000536543	0.183308741	1.04E-07	48	298	146	19
LSTM 2	0.000490795	0.184130793	1.02E-07	45	326	155	17
Bi-LSTM 1	0.000561684	0.163749935	0.000133455	47	121	294	18
Bi-LSTM 2	0.000582273	0.165189885	0.000143483	42	129	287	17

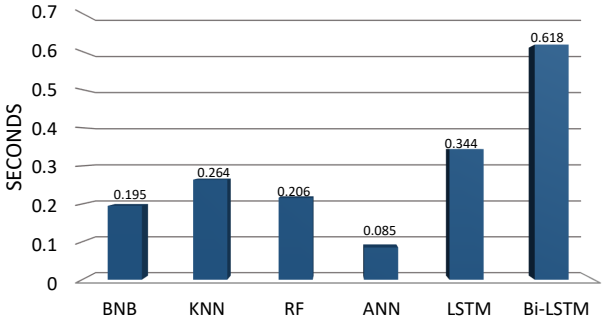

Figure 5: Performance comparison (computational time).

LSTM, Bi-LSTM, and ANN), which are known to be effective when used with temporal/sequential data.

Data pre-processing. To feed our data to the deep learning algorithms, we encoded our features by converting them into a numerical machine-readable form. In this context, we replaced the categorical value (API calls) with a numeric value between 0 and the number of unique API calls minus 1. If the categorical variable value contains 5 distinct API calls, we use (0, 1, 2, 3, and 4). Moreover, we represent our categorical variables as binary vectors. First, we map categorical classes to integer values. Then, each integer value is represented as a binary vector that is all zero values except the index of the integer, which is marked with a 1. For each label, we created a new column with binary encoding (0 or 1) to denote whether a particular row belongs to this category. After that, we split our data to training (80%) and testing/validation (20%).

Model Selection and Evaluation. We have studied different deep learning models to classify ransomware families based on their paranoia activity. The structure and layers of these models are described in the following sub-sections, along with the evaluation results. Moreover, we chose RMSprop [47] as an optimizer for this classification problem, given that it is a de-facto approach and has shown to be significantly effective when employed with Recurrent Neural Networks (RNN). To iteratively enhance the outcomes, we started with simple models (fewer layers) and systematically added further layers until we reached a relatively good fit (no under-fitting and over-fitting). Consequently, we performed hyper-parameter tuning using the random search algorithm. Finally, we identified the parameters that yielded the highest F-measure among the runs; then we refined our random search. As summarized in Table VII, the parameters that were tuned are the learning rate, the decay in the RMSprop optimizer, the proportion of drops, number of neurons in a layer, size of batches, and number of epochs.

Hyper-Parameter Tuning. It is worth mentioning that different techniques were applied to decrease over fitting and achieve a good fit on the training data. First of all, we use dropping techniques [48], which refers to dropping out/ignoring units (i.e., neurons) during the training phase with a certain probability. Moreover, we use batch normalization technique [49] to standardize the inputs to a layer for each mini-batch. This helps in stabilizing the learning process and dramatically reducing the number of training epochs required to train deep networks. Moreover, we applied hyper-parameter tuning in order to find the best parameters to achieve a good fit since there exist a large number of variables that can be tuned to enhance the training. Moreover, we performed 10-fold cross-validation, which helps in reducing the number of training samples and decrease over-fitting. Finally, we use categorical cross entropy as a loss function.

Applied Random Search. For each model, we applied a random search and a refined random search, where the hyper-parameters that we discovered are summarized in Table VII. In a deep learning model, various parameters contribute to finding the best fit for the training data (e.g., learning rate, decay, batch size, etc.). We generate 500 different combinations out of these parameters. The random search uses a random combination of these values and performs similar to meta-heuristics and grid search, however, with a lower computational cost [50]. We performed random searches and several refined searches; nevertheless, since the values on average were similar for all trials on the second random search, we opted to apply two random searches, a general one, and a refined random search in the realm of -/+ 10% of the best performing parameters according to the F-measure. This was performed to all the deep learning models where they showed that two random searches is sufficient.

In the following sub-sections, we present the analysis of the performance of the deep learning algorithms that were implemented to classify ransomware families based on their evasion API calls. We select the deep learning model which shows the highest F-measure score, as it is more representative of minority classes in the data. As a result, our classification model will utilize the features learned by our best model, to attribute such ransomware.

1) Artificial Neural Network (ANN): We use the ANN model because of the scarcity of the data, where a simple model will be able to generalize better. The developed ANN architecture depicted in Figure 6 consists of the following layers:

- **Input layer**: The input of the network is a 23 array of encoded API calls.

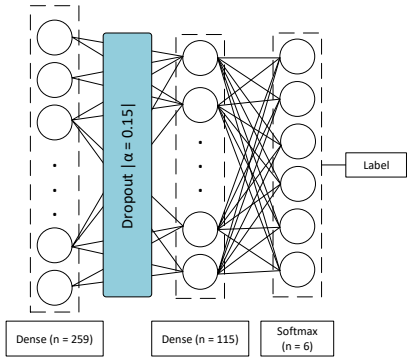

Figure 6: Structure of the ANN model.

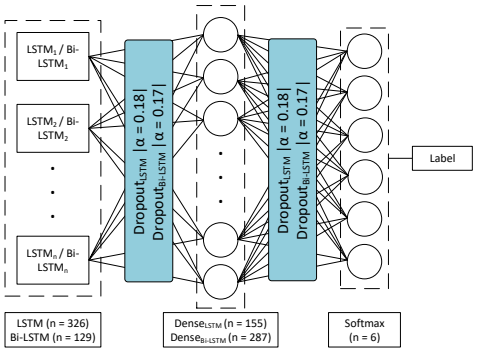

Figure 7: Implemented LSTM/Bi-LSTM model architectures.

- **Fully-Connected Layer 1**: The input is fed to a fully connected layer with the rectified linear activation function that overcomes the vanishing gradient problem, while allowing the models to learn faster and perform better. It is a piece-wise linear function that will output the input directly if it is positive, otherwise, it will output zero. This is followed by a drop layer with $\alpha = 0.18$ along with batch normalization that would help in decreasing over-fitting and handling noise.
- **Fully-Connected Layer 2**: The output of the first fully connected layer is inputted to another layer of a fully connected layer with a ReLu activation function. The outcome is then fed to a Softmax function to output the normalized probability distribution over the classes.

Initially, we started by running a random search algorithm (500 runs). We achieve an accuracy of 86.80% and an F-measure of 86.71%. To further explore this deep learning algorithm, we applied random search on the parameters discovered with our initial random search that yielded better F-measure score, with a rate of -/+ 10%. After applying the second random search to refine our parameters and improve the performance, we achieved an F-measure score of 87.74% and an accuracy of 87.80%, that is, 295 samples tagged correctly according to their family.

As summarized in Table VIII, the ANN model achieved F-measure values above 85% for 5 out the 6 studies families, indicating an overall satisfactory quality and classification effectiveness. Moreover, the recall and precision values obtained for classifying these 5 families were also over 85%, confirming this model's good performance. However, only the Benign family was associated with relatively lower F-measure and recall values (less than 80% for). This could be due to the smaller number of benign samples used to train the model, which was 27% less than the number of utilized samples for other families. Additionally, the ANN confusion matrix results (Table IX) shows that 25% of the benign samples were misclassified, thus confirms that the Benign family needs more in-depth analysis to improve the accuracy of the model. Moreover, we note that the benign samples generated some overlapping evasion APIs patterns with other ransomware samples, which could be the reason for such misclassification. Despite that, our analysis shows that the majority of the ransomware samples ($> 85\%$) for the remaining families were correctly attributed to the known families.

Furthermore, our performance analysis shows that the computational time to complete the classification experiments using the ANN model achieved an outstanding time of 0.085 seconds, with an overall classification accuracy above 85%. As illustrated in Figure 5, the ANN model outperforms all the previous machine learning models, with a significantly shorter time for completing the classification tasks. Therefore, despite the fact that RF model produces relatively higher classification accuracy, our analysis indicates that a customised ANN model can be considered as an effective alternative model, especially when the number of samples grows significantly.

2) Long-Short Term Memory (LSTM): In this RNN model, we stacked an LSTM layer, where each cell will output one hidden state for each input, on top of a fully connected hidden layer with a ReLu activation function, and a softmax output layer. Moreover, we use categorical cross entropy as a loss function since we have more than two classes in which we want to identify.

The architecture of the LSTM depicted in Figure 7 consists of the following layers:

- **Input Layer**: The input of the network is a 23 array of encoded API calls.
- **LSTM Layer**: This is the main building block of an LSTM deep neural network. It is responsible for learning order dependency in our evasion API feature space, followed by a dropout with $\alpha = 0.18$ to decrease over-fitting.
- **Fully-Connected Layer**: After the LSTM layer, we add a fully connected layer to complete the LSTM architecture. The output of the LSTM is three dimensional, however, the fully connected layer expects a one dimensional vector of numbers. Therefore, we create a copy of the output collapsed into one dimension. To decrease over fitting, we apply batch normalization along with a dropout. The last connected layer combines the features learned in the previous layers and applies the softmax function to output the normalized probability distribution over the classes.

After running our first random search, we achieve a relatively good accuracy (85.71%) and F-measure scores (85.63%). Consequently, we applied a second random search in the realm of the random parameters that generated this first model to check if we can enhance the learning by altering the parameters with a rate of 10%. By applying the second random

Table VIII: Per-family evaluation results for ANN, LSTM and Bi-LSTM models.

	ANN			LSTM			Bi-LSTM		
Family	**F-mea.(%)**	**Recall(%)**	**Precis.(%)**	**F-mea.(%)**	**Recall(%)**	**Precis.(%)**	**F-mea.(%)**	**Recall(%)**	**Presic.(%)**
Benign	79.52	75.0	84.62	85.06	84.09	86.05	83.15	84.09	82.22
Cerber	87.85	85.45	90.38	86.0	78.18	95.56	86.87	78.18	97.73
Locky	87.41	86.76	88.06	87.77	89.71	85.92	89.55	88.24	90.91
Reveton	90.72	88.0	93.62	85.44	88.0	83.02	84.62	86.27	83.02
TeslaCrypt	91.97	96.92	87.5	89.55	92.31	86.96	91.18	95.38	87.32
Yakes	86.73	90.74	83.05	88.07	88.89	87.27	89.29	92.59	86.21

Table IX: Confusion matrices's outcomes for ANN, LSTM, and Bi-LSTM

	ANN						LSTM						Bi-LSTM					
	B	C	L	R	T	Y	B	C	L	R	T	Y	B	C	L	R	T	Y
B	**33**	4	3	0	0	4	**37**	0	5	1	1	0	**37**	0	1	2	1	3
C	2	**47**	1	0	1	4	3	**43**	1	4	2	2	4	**43**	1	4	1	2
L	2	0	**59**	3	4	0	2	0	**61**	3	1	1	2	0	**60**	3	2	1
R	0	0	2	**44**	3	1	0	1	2	**44**	3	0	0	1	3	**43**	3	0
T	0	1	0	0	**63**	1	0	1	0	0	**60**	4	1	0	0	0	**62**	2
Y	2	0	2	0	1	**49**	1	0	2	1	2	**48**	1	0	1	0	2	**50**

B=Benign, C=Cerber, L=Locky, R=Reveton, T=TelesCrypt,Y=Yakes

search, we were able to improve the F-measure score from 85.63% to 87.17% and the accuracy from 85.71% to 87.20%. This indicates that 293 out of 336 ransomware samples were correctly classified.

The per-class performance metrics of the LSTM model are illustrated in Table VIII. Herein, the F-measure achieved a value for all the families between 85% and 90%, which could be interpreted as an acceptable model's performance. Furthermore, the recall and precision measurements obtained values between 84%–96% for 5 out of 6 families, which is aligned with the F-measure. It is worth noting that the LSTM model performance is assumed to improve when we increase the number of training/testing samples since having a complex feature vector of API calls necessitates more samples to better capture the relationship between the features.

The confusion matrix analysis results for the LSTM model presented in Table IX. In general, the LSTM produced an overall accuracy similar to the ANN model. This is also clearly reflected in the number of correctly classified samples from each family when comparing the two model. Nevertheless, LSTM seems to perform better than ANN in classifying Benign samples, while producing comparable results for classifying samples from other families. Further, the LSTM model required significantly longer time (0.344 seconds) to complete the classification process as compared to the ANN Model (Figure 5). This is mainly due to the complex internal structure of the LSTM model, which tries to find associations between learned features and thus, requiring more computational time and resources.

3) Bidirectional Long-Short Term Memory (Bi-LSTM): In this RNN model, we stacked a Bidirectional LSTM layer, where each cell will output one hidden state for each input, on top of a fully connected hidden layer with a ReLu activation function, and a softmax output layer.

The architecture of the Bi-LSTM that is depicted in Figure 7 consists of the following layers:

- **Input layer**: The input of the network is a 23 array of encoded API calls.
- **Bi-LSTM Layer**: The main building block of an bidirectional LSTM deep neural network. This layer is responsible for learning and identifying sequences in the inputted features, while taking into account forward and backward order. It is then followed by a dropout with $\alpha = 0.17$.
- **Fully-Connected Layer**: After the Bi-LSTM layer, we add a fully connected layer to wrap up the Bi-LSTM architecture. The output of the Bi-LSTM is three dimensional; we then convert the output of the Bi-LSTM layer into a 1-dimensional array, since the fully connected layer expects a one dimensional vector of numbers. Moreover, we also apply batch normalization along with a dropout. Finally, we apply a softmax function to output the normalized probability distribution over the classes using the last connected layer that combines the features learned in the previous layers.

After applying the first random search and achieving 86.18% for F-measure score and 86.31% for accuracy, we applied a second random search in the realm of the random parameters that resulted in an F-measure of 87.77% and an accuracy of 87.80%, which is similar to the accuracy scored by ANN and LSTM, representing 295 correctly classified samples. The per-class performance metrics are reported in Table VIII. Similar to LSTM, the Bi-LSTM is ideal for cases with a significantly larger number of ransomware samples, where it is assumed to perform effective classification with improved overall performance. This could also hamper the obtained F-measure values, which were under 90% for most of the families (Benign, Cerber, Locky, Reventon, Yakes). However, recall and precision exhibit values over 90%, indicating that with the increase number of samples, the F-measure score may increase while improving the overall performance.

The analysis of the confusion matrix outcomes for the Bi-LSTM model (Table IX) shows that it similar to the LSTM model, it outperforms the ANN model in classifying Benign samples. Moreover, the Bi-LSTM model produces high

Table X: Classifiers outcomes.

Classifier	Accur.(%)	F-mea.(%)	Recall(%)	Precis.(%)
RF	**94.92**	**94.61**	**94.44**	**95.16**
KNN	92.85	92.60	92.54	92.75
ANN	87.80	87.74	87.15	87.87
Bi-LSTM	87.80	87.77	87.46	87.90
LSTM	87.20	87.17	86.86	87.46
BNB	85.11	83.69	83.32	85.54

accuracy for most ransomware families. For instance, the 62 samples of the TeslaCrypt family were classified classified correctly, with about 95% per family classification accuracy. Moreover, the comparison of the results to the LSTM and ANN models shows comparable results that are not statistically significant. On the other hand, the processing time for the Bi-LSTM model was the longest among all tested models, with about 0.618 seconds required to classify all the samples (Figure 5). This is not surprising as the BiLSTM model is an extended version of LSTM with an internal mechanism that accounts for finding two-ways relational dependencies in the analyzed data/features.

D. Evaluation and Comparison

To attribute ransomware to their corresponding families, we devised and developed different ransomware family classification models based on the paranoia activity generated by the ransomware samples, whether they were malicious or legitimate. Consequently, we tailor three machine learning algorithms (BNB, KNN and RF) and three deep learning algorithms (ANN, LSTM, and Bi-LSTM) to classify such samples. We evaluate and compare the performance of the tailored classification models based on their accuracy/F-Measure (Table X) along with the overall computational performance represented by the time required to complete the classification process (Figure 5).

As shown in Table X, all machine learning models except BNB produced significantly higher accuracy and F-measure scores as compared to the deep learning models, with RF to have the highest accuracy (94.92%) and F-measure values (94.61%) among all. The KNN model came second with a score slightly above 92% for both measures, respectively. It is interesting to see that the BNB scored the lowest among all tested models (between 83%–85% for accuracy and F-measures). We attribute these low values to the partial fulfillment of the class conditional independence in the Bernoulli Naive Bayes model. On the other hand, our analysis indicates that the tested DL models resulted relatively lower scores for both accuracy and F-measure (about 87%). Additionally, while the tested DL models produced similar classification outcomes, we believe that their classification outcomes may improve when experimenting with a larger sample size.

The values of the precision and recall for the analyzed models is presented in Table X. Achieving high values for these two metrics indicates accurate results, which depicts how much we can trust these classifiers to identify all the samples. Our analysis shows that the RF and KNN produced the highest values for precision and recall, indicating that the proportion of correctly predicted samples is significant and thus, complementing and confirming the analysis done for the accuracy and F-measure. These results provide a measure of the effectiveness and quality of the analyzed models when attributing ransomware to their corresponding families based on their paranoia activities.

Further, to compare the computational performances of the devised classifiers, we measure their speed in terms of time required to complete the classification experiments. As illustrated in Figure 5, the ANN model performed significantly faster than all the other tested models, with 0.085 seconds to complete the classification. Additionally, the ML models such as BNB, RF, and KNN performed relatively slower, with a computational time of 0.195, o,206, and 0.264 seconds, respectively. On the other hand, it is interesting to see that the LSTM-based DL models performed significantly worst, with 0.344 and 0.618 seconds for LSTM and Bi-LSTM models, respectively.

In general, despite the fact that the RF model produced the best classification outcomes in terms of F-measure and accuracy measures, the ANN could be also considered as a potential option for our proposed ransomware family classification due to its computational performance and speed. This could be especially the case when dealing with a larger number of analyzed ransomware samples, which may eventually improve the overall learning outcomes in terms of model accuracy.

V. Discussion

To overcome the problem of ransomware family classification, we proposed an approach that leverages their paranoia activity through the development and application of various ML/DL models. Such models are capable of identifying a set of evasion APIs that characterize the behaviors of various ransomware families. Furthermore, the results demonstrate that some families invoke a unique set of evasion API calls with different frequencies. Inline with our previous finding, the executed experiments reveal that the classification of ransomware families based on their paranoia activities can be achieved using three different techniques, one of which is based on the mere presence and absence of an API calls, the second one is based on the frequency of the evasion API, and finally, a technique based on the sequence of API calls. Furthermore, our best model (RF) was able to identify and differentiate benign software from the different ransomware families with high precision and recall values (about 94%), as summarized in Tables X.

Practical Cyber-Security Benefits. Mitigating ransomware attacks requires detecting them prior to executing their payload, which typically results in encrypting/locking the targeted device/data. One way to achieve this is to analyze the behavioral characteristics of suspicious applications and identify adversarial activities, which might be attributed to known ransomware families while triggering appropriate actions to defend against them.

In line with that, our dynamic approach utilizes pre-attack paranoia activities along with tailored ML/DL models to offer effective and timely ransomware classification and family

attribution. More importantly, given the knowledge about existing ransomware families, and the fact that the paranoia activities are captured/analyzed prior to perpetuating the actual attacks, our approach can be leveraged to develop practical preventive capabilities to obstruct ransomware attacks at early stages (e.g., during their paranoia activity). Moreover, the behavioral characteristics associated with various ransomware families can be utilized to discover new emerging ransomware variants and/or detect new unknown families by feeding new ransomware variants into the ML/DL models and re-training them accordingly. Additionally, this work contributes toward identifying new ransomware variants/families by running the model and findings samples that do not belong/associate to known families based on their paranoia activity. Such findings however, require further investigation to confirm the observation.

It is also worth noting that our approach implements lightweight classifiers, which can be portable to other environments such as Linux and IoT-centric OSs. Further, this work can be leveraged to devise ransomware attack mitigation and prevention methods through learning the behavioral characteristics of benign application and comparing them to anomalous behaviors of newly installed applications (e.g., possible ransomware). This can be done by instrumenting a "learning" period for vetting/whitelisting the installation of default programs, while increasing the accuracy of the implemented ransomware classifier and its capabilities for ransomware detection purposes. Consequently, new signatures can be created for the detected ransomware based on their paranoia activities, which can be utilized to detect/mitigate further attacks including possible 0-day using conventional Intrusion Detection Systems (IDS). Finally, this research can also be leveraged to function as a defense mechanism inside a computer network in order to detect paranoia activity that are produced by different types of malicious software such as ransomware and/or other types of malware to determine the origin of this sensing actions and stop possible attacks.

Robustness. Considering that our approach's scope is only family classification of known ransomware based on its paranoia activity, we did not include unknown ransomware or any other types of malware in our experiment. We used 5 ransomware families in our experiments due to the lack of a balanced ransomware datasets. However, our approach is easily reproducible (it is available in GitHub[3]) to make it operational with other ransomware datasets given that they share similar semantics (i.e., paranoia activity). Further, fingerprinting actions to sense the environment has turned into a malware's tendency to detect dynamic analyzers [33]. Therefore, our approach can be tested with any other types of malware that employed paranoia activity. Moreover, benign samples can be always included in the experiments as long as they apply environmental detection actions.

Limitations and Future Work. Although this work contributes towards understanding ransomware and their behavior, it is realistic to note a few current limitations. For instance, the work relies on a supervised learning approach, which cannot classify new, previously unseen ransomware variants. Therefore, labeling our samples prior to training our ML/DL models to perform ransomware family classification is vital. To overcome this limitation, unsupervised learning approaches (e.g., clustering) can be considered as a complementary approach, which can be pursued to face the emergence of new ransomware families. Moreover, this work can be leveraged for future works by incorporating a combination of static and dynamic analysises, which would benefit from the advantages of these two approaches. Additionally, our classification approach can be leveraged as a stepping stone to develop new defense mechanisms that prevent and detect these types of malicious attacks. In addition, while this work aims at ransomware classification and family attribution, the proposed approach can be extrapolated to other types of malware, which invoke detection evasion API calls in their pre-attack routines.

Finally, our solution does not handle the sustainability/deterioration issue that most learning-based detectors suffer from. The issue is generated by the use of old samples to detect new samples, and this affects the robustness of the model [51]. Different approaches have been employed to handle the deterioration of the machine learning models. In [52] they proposed the use of API graphs, which helped in decreasing the deterioration of the model, whereas [53] proposed the use of online learning to handle the evolving nature of malware, along with an evolving feature sets and pseudo labels. Moreover, in [54] they assess the deterioration of the models and emphasize on the importance of the features used. The features should consistently differentiate malware from benign application for sustainable learning-based malware detection. And this aligns with our feature analysis and selection strategy were we used features that are predominantly associated with malicious software samples. For future work we will be detailing a study on our feature set to assess and detect the existence of concept drift that might affect the performance of the AI-based learning models using works similar to [55].

VI. Conclusion

In this work, we propose a dynamic analysis approach for attributing ransomware samples based on their pre-attack paranoia activities. We execute more than 3,000 ransomware samples that belong to 5 predominant families in a sandboxing environment to collect their behavioral characteristics/features in terms of 23 selected pre-attack evasion API calls that are associated with sensing the execution environment. To this end, we utilize BoW, SoW, and OoW to build feature vectors while implementing various ML/DL models such as RF, KNN, BNB, ANN, LSTM, and BI-LSTM to perform the family classification based on the collected paranoia activities.

In general, our analysis shows the prevalence of paranoia activities among ransomware samples. More importantly, our findings indicate that the paranoia activities can be in fact used as distinguishable features for classifying ransomware samples into known families. Moreover, our detailed evaluation and comparison of the deployed models showed that the RF model obtained the best classification outcomes (i.e., accuracy and F-measure), while the ANN model produced the

[3]https://github.com/Rmayalam/Ransomware_Paranoia.git

smallest computational times for completing the ransomware classification process. Finally, while this work helps towards building a better understanding of the ransomware threat landscape, it introduces a practical approach for building effective tools for detection and mitigation of ransomware attacks by monitoring their pre-attack activities and attributing them to known families during early phases.

ACKNOWLEDGMENT

This work has been supported by Natural Sciences and Engineering Research Council of Canada (NSERC) and Concordia University. This work was also partially supported by a grant from the U.S. National Science Foundation (NSF), Office of Advanced Cyberinfrastructure (OAC), #2104273.

REFERENCES

[1] E. Berrueta *et al.*, "A survey on detection techniques for cryptographic ransomware," *IEEE Access*, vol. 7, pp. 144 925–144 944, 2019.

[2] B. A. S. Al-rimy, M. A. Maarof, and S. Z. M. Shaid, "Ransomware threat success factors, taxonomy, and countermeasures: A survey and research directions," Jan 2018. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S016740481830004X

[3] Kaspersky, "What are the different types of ransomware?" Dec 2020. [Online]. Available: https://www.kaspersky.com/resource-center/threats/ransomware-examples

[4] "Fbi director sees 'parallels' between ransomware threat and 9/11," Dec 2020. [Online]. Available: https://amp.theguardian.com/us-news/2021/jun/04/fbi-christopher-wray-cyberattacks-9-11

[5] Z. Cohen and G. Sands, "Four key takeaways on the us government response to the pipeline ransomware attack," May 2021. [Online]. Available: https://www.cnn.com/2021/05/11/politics/colonial-pipeline-cyber-hearing-senate-homeland-security-committee/index.html

[6] V. Salama, A. Marquardt, and Z. Cohen, "Several hospitals targeted in new wave of ransomware attacks," Oct 2020. [Online]. Available: https://www.cnn.com/2020/10/28/politics/hospitals-targeted-ransomware-attacks/index.html

[7] E. M. Lab, "Ransomware statistics for 2020: Q1 report," Jun 2020. [Online]. Available: https://blog.emsisoft.com/en/36303/ransomware-statistics-for-2020-q1-report/

[8] "2020 ransomware statistics, data, & trends," Nov 2020. [Online]. Available: https://purplesec.us/resources/cyber-security-statistics/ransomware/

[9] D. Freeze, "Global ransomware damage costs predicted to reach $20 billion (usd) by 2021," Jan 2020. [Online]. Available: https://cybersecurityventures.com/global-ransomware-damage-costs-predicted-to-reach-20-billion-usd-by-2021/

[10] B. D. M. M. at phoenixNAP. Researcher *et al.*, "27 shocking ransomware statistics that every it pro needs to know," Feb 2021. [Online]. Available: https://phoenixnap.com/blog/ransomware-statistics-facts

[11] S. Kok *et al.*, "Ransomware, threat and detection techniques: A review," *Int. J. Computer Science and Network Security*, vol. 19, no. 2, p. 136, 2019.

[12] B. Zhang *et al.*, "Ransomware classification using patch-based cnn and self-attention network on embedded n-grams of opcodes," *Future Generation Computer Systems*, vol. 110, pp. 708–720, 2020.

[13] K. P. Subedi, D. R. Budhathoki, and D. Dasgupta, "Forensic analysis of ransomware families using static and dynamic analysis," in *2018 IEEE Security and Privacy Workshops (SPW)*. IEEE, 2018, pp. 180–185.

[14] R. Vinayakumar *et al.*, "Evaluating shallow and deep networks for ransomware detection and classification," in *Int. Conf. on Advances in Computing, Communications and Informatics (ICACCI)*. IEEE, 2017, pp. 259–265.

[15] H. Daku, P. Zavarsky, and Y. Malik, "Behavioral-based classification and identification of ransomware variants using machine learning," in *17th IEEE Int. Conf. On Trust, Security And Privacy In Computing And Communications/12th IEEE Int. Conf. On Big Data Science And Engineering (TrustCom/BigDataSE)*, 2018, pp. 1560–1564.

[16] H. Zhang *et al.*, "Classification of ransomware families with machine learning based on n-gram of opcodes," *Future Generation Computer Systems*, vol. 90, pp. 211–221, 2019.

[17] A. AlSabeh *et al.*, "Exploiting ransomware paranoia for execution prevention," in *IEEE Int. Conf. on Communications (ICC)*. IEEE, 2020, pp. 1–6.

[18] L. Onwuzurike *et al.*, "Mamadroid: Detecting android malware by building markov chains of behavioral models (extended version)," *ACM Transactions on Privacy and Security (TOPS)*, vol. 22, no. 2, pp. 1–34, 2019.

[19] J. Yan, G. Yan, and D. Jin, "Classifying malware represented as control flow graphs using deep graph convolutional neural network," in *2019 49th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. IEEE, 2019, pp. 52–63.

[20] G. Suarez-Tangil *et al.*, "Droidsieve: Fast and accurate classification of obfuscated android malware," in *Proceedings of the Seventh ACM on Conference on Data and Application Security and Privacy*, 2017, pp. 309–320.

[21] S. K. Dash *et al.*, "Droidscribe: Classifying android malware based on runtime behavior," in *2016 IEEE Security and Privacy Workshops (SPW)*. IEEE, 2016, pp. 252–261.

[22] H. Cai *et al.*, "Droidcat: Effective android malware detection and categorization via app-level profiling," *IEEE Transactions on Information Forensics and Security*, vol. 14, no. 6, pp. 1455–1470, 2018.

[23] "Automated malware analysis," Nov 2020. [Online]. Available: https://cuckoosandbox.org/

[24] "VirusTotal," Nov 2020. [Online]. Available: https://www.virustotal.com/gui/

[25] "VirusShare," Nov 2020. [Online]. Available: https://virusshare.com/

[26] Malicialab, "malicialab/avclass," Nov 2020. [Online]. Available: https://github.com/malicialab/avclass

[27] L. Loeb, "Cerber ransomware owns the market," Jan 2021. [Online]. Available: https://securityintelligence.com/news/cerber-ransomware-owns-the-market/

[28] "Averting ransomware epidemics in corporate networks with windows defender atp," Jul 2019. [Online]. Available: https://www.microsoft.com/security/blog/2017/01/30/averting-ransomware-epidemics-in-corporate-networks-with-windows-defender-atp/

[29] J. Johnson, "Major operating systems targeted by ransomware 2020," Feb 2021. [Online]. Available: https://www.statista.com/statistics/701020/major-operating-systems-targeted-by-ransomware/

[30] G. Yan, N. Brown, and D. Kong, "Exploring discriminatory features for automated malware classification," in *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*. Springer, 2013, pp. 41–61.

[31] "Dynamic malware analysis in the modern era-a state of the art survey dynamic malware analysis in the modern era-a state of the art survey," Mar 2021. [Online]. Available: https://dl.acm.org/doi/fullHtml/10.1145/3329786

[32] "Evasion techniques," May 2021. [Online]. Available: https://evasions.checkpoint.com/

[33] A. Afianian *et al.*, "Malware dynamic analysis evasion techniques: A survey," *arXiv preprint arXiv:1811.01190*, 2018.

[34] S.-B. Kim *et al.*, "Some effective techniques for naive bayes text classification," *IEEE transactions on knowledge and data engineering*, vol. 18, no. 11, pp. 1457–1466, 2006.

[35] R. Jodha *et al.*, "Text classification using knn with different features selection methods," *International Journal of Research Publications*, vol. 8, no. 1, pp. 8–8, 2018.

[36] K. Shah *et al.*, "A comparative analysis of logistic regression, random forest and knn models for the text classification," *Augmented Human Research*, vol. 5, no. 1, pp. 1–16, 2020.

[37] L. Sayfullina *et al.*, "Efficient detection of zero-day android malware using normalized bernoulli naive bayes," in *2015 IEEE Trustcom/BigDataSE/ISPA*, vol. 1. IEEE, 2015, pp. 198–205.

[38] Y. Wang and Z.-O. Wang, "A fast knn algorithm for text categorization," in *Int. Conf. on Machine Learning and Cybernetics*, vol. 6. IEEE, 2007, pp. 3436–3441.

[39] "Text classification using knn with different feature selection methods," May 2021. [Online]. Available: https://www.researchgate.net/publication/326893075_Text_Classification_using_KNN_with_different_Feature_Selection_Methods

[40] "An improved knn text classification algorithm based on clustering," Feb 2021. [Online]. Available: https://pdfs.semanticscholar.org/59dd/c6120c15b4327d675f1da6ff540727078c7a.pdf

[41] C. H. Wan *et al.*, "A hybrid text classification approach with low dependency on parameter by integrating k-nearest neighbor and support vector machine," *Expert Systems with Applications*, vol. 39, no. 15, pp. 11 880–11 888, 2012.

[42] N. Suguna and K. Thanushkodi, "An improved k-nearest neighbor classification using genetic algorithm," *Int. J.of Computer Science Issues*, vol. 7, no. 2, pp. 18–21, 2010.

[43] M. W. Gardner and S. Dorling, "Artificial neural networks (the multilayer perceptron)—a review of applications in the atmospheric sciences," *Atmospheric environment*, vol. 32, no. 14-15, pp. 2627–2636, 1998.

[44] S. Mitra, R. K. De, and S. K. Pal, "Knowledge-based fuzzy mlp for classification and rule generation," *IEEE Transactions on Neural Networks*, vol. 8, no. 6, pp. 1338–1350, 1997.

[45] C. Zhou *et al.*, "A c-lstm neural network for text classification," *arXiv preprint arXiv:1511.08630*, 2015.

[46] R. Alzaidy, C. Caragea, and C. L. Giles, "Bi-lstm-crf sequence labeling for keyphrase extraction from scholarly documents," in *The WWW Conf.*, 2019, pp. 2551–2557.

[47] K. Team, "Keras documentation: Rmsprop," May 2021. [Online]. Available: https://keras.io/api/optimizers/rmsprop/

[48] W. Wang, M. Zhao, and J. Wang, "Effective android malware detection with a hybrid model based on deep autoencoder and convolutional neural network," *J.of Ambient Intelligence and Humanized Computing*, vol. 10, no. 8, pp. 3035–3043, 2019.

[49] M. A. Kadri, M. Nassar, and H. Safa, "Transfer learning for malware multi-classification," in *Proc. of the 23rd Int. Database Applications & Engineering Symposium*, 2019, pp. 1–7.

[50] R. G. Mantovani *et al.*, "Effectiveness of random search in svm hyper-parameter tuning," in *Int. Joint Conf. on Neural Networks (IJCNN)*. Ieee, 2015, pp. 1–8.

[51] X. Fu and H. Cai, "On the deterioration of learning-based malware detectors for android," in *2019 IEEE/ACM 41st International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)*. IEEE, 2019, pp. 272–273.

[52] X. Zhang *et al.*, "Enhancing state-of-the-art classifiers with api semantics to detect evolved android malware," in *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, 2020, pp. 757–770.

[53] K. Xu *et al.*, "Droidevolver: Self-evolving android malware detection system," in *2019 IEEE European Symposium on Security and Privacy (EuroS&P)*. IEEE, 2019, pp. 47–62.

[54] H. Cai, "Assessing and improving malware detection sustainability through app evolution studies," *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 29, no. 2, pp. 1–28, 2020.

[55] R. Jordaney *et al.*, "Transcend: Detecting concept drift in malware classification models," in *26th USENIX Security Symposium (USENIX Security 17)*, 2017, pp. 625–642.

Ricardo Ayala is studying a Ph.D. in Information and Systems Engineering at Concordia University, Montreal, Canada. He received his M.Sc. degree from the Computer Engineering Department, Universitat Oberta de Catalunya (UOC), Spain, in 2017. Moreover, He obtained a Master of Science in Finance degree from the University of El Salvador (UES), El Salvador, in 2017. His research interests focus on cyber security, attacks detection/characterization, and the Internet of Things.

Sadegh Torabi received the Ph.D. degree in Information Systems Engineering from Concordia University, Montreal, Canada. He received the M.Sc. degree from the Electrical and Computer Engineering Department, University of British Columbia (UBC), Vancouver, BC, Canada, in 2016, and the B.Sc./M.Sc. degrees (with Distinction) from the Computer Engineering Department, Kuwait University (KU), Kuwait, in 2005 and 2009, respectively. He is currently a postdoctoral research fellow at the Center for Secure Information Systems (CSIS) at George Mason University (GMU). His research interests are in the areas of Internet measurements, network/systems security, usable security/privacy, and operational cyber security including the security of Internet of Things and cyber-physical systems.

Khaled Sarieddine is a Ph.D. student at Concordia University. He was an assistant instructor in the Department of Computer Science at the American University of Beirut (AUB), where he also obtained his bachelor's and master's degree. His research interests include Vehicular Ad hoc Networks (VANETs), Fog, Edge, and Cloud Computing. Moreover, his most recent research interests extend to malware and ransomware detection/prevention, along with security of the electric vehicle ecosystem.

Elias Bou-Harb received the Ph.D. degree in computer science from Concordia University, Montreal, Canada. He was a visiting research scientist at Carnegie Mellon University (CMU) in 2015–2016 before joining the Department of Computer Science at Florida Atlantic University (FAU) as an Assistant Professor in 2016. He is currently an Associate Professor at the Cyber Center For Security and Analytics at the Department of Information Systems and Cyber Security at the University of Texas at San Antonio (UTSA). His research interests are in operational cyber security, attacks detection/characterization, Internet measurement, cyber-physical systems security, and mobile network security.

Nizar Bouguila received the engineer degree from the University of Tunis, Tunis, Tunisia, in 2000, and the M.Sc. and Ph.D. degrees in computer science from Sherbrooke University, Sherbrooke, QC, Canada, in 2002 and 2006, respectively. He is currently a Professor with the Concordia Institute for Information Systems Engineering (CIISE) at Concordia University, Montreal, Quebec, Canada. His research interests include image processing, machine learning, data mining,, computer vision, and pattern recognition. Prof. Bouguila received the best Ph.D Thesis Award in Engineering and Natural Sciences from Sherbrooke University in 2007. He was awarded the prestigious Prix d'excellence de l'association des doyens des etudes superieures au Quebec (best Ph.D Thesis Award in Engineering and Natural Sciences in Quebec), and was a runner-up for the prestigious NSERC doctoral prize. He was the holder of a Concordia University research Chair Tier 2 from 2014 to 2019 and was named Concordia University research Fellow in 2020. He is the author or co-author of more than 400 publications in several prestigious journals and conferences. He is a regular reviewer for many international journals and serving as associate editor for several journals such as Pattern Recognition journal. Dr. Bouguila is a licensed Professional Engineer registered in Ontario, and a Senior Member of the IEEE.

Chadi Assi received his Ph.D. from the City University of New York (CUNY). During his PhD he worked on optical networks, and namely on lightpath provisioning and survivability. He spent a year as a visiting researcher at Nokia Research Center (Boston) where he worked on quality of service (QoS) in passive optical access networks. He joined Concordia University in 2003 as an Assistant Professor where he is currently a Full Professor. He was a Concordia research chair (Tier 2) between 2012 and 2017, then since 2017 he holds a research chair, Tier 1. He was elevated to an IEEE Fellow (class 2020) by the Communications Society for "contributions to resource allocation for optical and wireless networks". His research interests are in the general area of networks and telecommunications (both wired and wireless), (IoT) cyber security and smart grids. He serves or served on the Editorial Board of several flagship journals of the IEEE. He was the recipient of the Prestigious Mina Rees Dissertation Award from CUNY in 2002 for his research on wavelength division multiplexing in optical networks.