Early-stage Ransomware Detection based on Pre-Attack Internal API Calls

Filippo Coglio, Ahmed Lekssays, Barbara Carminati, and Elena Ferrari

Abstract Ransomware attacks have become one of the main cyber threats to companies and individuals. In recent years, different approaches have been proposed to mitigate such attacks by analyzing ransomware behavior during the infection and post-infection phases. However, few works focused on early-stage ransomware detection. The analysis of recent ransomware has shown that they are designed to perform sensing activities to evade detection by known anti-viruses and anti-malware software. This paper proposes an early-stage ransomware detector based on a neural network model for multi-class classification. Our model achieves 80.00% accuracy on our dataset and 93.00% on another state-of-the-art dataset [11]. We show that our model performs better than the state-of-the-art approaches, especially on a challenging, large, and varied dataset we made publicly available.

1 Introduction

Ransomware is a malware designed to encrypt user information or lock access to infected devices and their resources. A ransomware exploits secure communication channels with C&C (Command and Control) servers to encrypt the victims' systems and force them to pay a ransom [9]. If the attacked entity refuses to pay the ransom, data is deleted or published on the web. Ransomware attacks have become one of the

---

Filippo Coglio
Università degli Studi dell'Insubria e-mail: `fcoglio@uninsubria.it`

Ahmed Lekssays
Università degli Studi dell'Insubria e-mail: `alekssays@uninsubria.it`

Barbara Carminati
Università degli Studi dell'Insubria e-mail: `barbara.carminati@uninsubria.it`

Elena Ferrari
Università degli Studi dell'Insubria e-mail: `elena.ferrari@uninsubria.it`

main cyber threats to both companies and individuals. In 2021, the average cost of a ransomware attack for companies was \$4.62 million, with an increase of 148% in the number of ransomware attacks from 2020 to 2021[1]. This increase was expected due to ransomware-as-a-service (RaaS) growth, where attackers sell their ransomware in underground markets, accepting payments in cryptocurrencies to preserve their anonymity [10]. This has turned ransomware into a lucrative tool for attackers who look for financial gains [11].

In recent years, different approaches have been proposed to mitigate such attacks using dynamic or static analysis to understand ransomware's code structure and behavior during infection and post-infection phases. Despite all the work, the defense against ransomware is challenging due to the lack of knowledge of newly detected ransomware.Therefore, there is a need to investigate effective approaches for detecting ransomware, keeping in mind their constant evolution.

In this paper, we focus on early-stage ransomware detection. The analysis of recent ransomware has shown that they are programmed to execute some functions and operations to evade detection by known anti-viruses and anti-malware software. These *paranoia activities* aim to sense the environment to understand whether the ransomware can run the malicious code [11].

Thus, based on pre-attack activities, we aim to detect ransomware before the encryption phase. We have dynamically analyzed more than 11,000 ransomware samples and 1200 benign samples from 23 different families to extract important API calls that ransomware mainly use before starting their attacks. These API calls help in classifying samples into their corresponding ransomware families or benign one. We have developed a neural network model for multi-class classification that achieves 80.00% accuracy on our dataset and 93.24% on another state-of-the-art dataset [11]. We show that our model performs better than the state-of-the-art approaches, especially on a challenging, large, and varied dataset. In addition, we show the effectiveness and feasibility of the proposed approach compared to previous work.

Contributions. The contributions of this work can be summarized as follows:

- we have compiled a dataset of 5203 benign and ransomware samples from 12 different families. To the best of our knowledge, it is the largest dataset available for ransomware detection;
- we have developed a neural network model that achieves an accuracy of 80.00% in a challenging, large, and varied dataset outperforming the state-of-the-art;
- we have made our source code and dataset publicly available[2] to reproduce the results.

Outline. The remainder of this paper is organized as follows. We discussed state-of-the-art approaches in Section 2. Section 3 presents background knowledge on ransomware detection. In Section 4, we discuss our methodology and the building blocks of our solution. Section 5 shows the obtained results and the comparison with state-of-the-art approaches. Finally, Section 6 concludes the paper.

---

[1] https://www.pandasecurity.com/en/mediacenter/security/ransomware-statistics/

[2] https://github.com/Ph1l99/RansomwareEarlyDetection

2 Related Work

In the last years, different techniques have been proposed for ransomware classification. [11] presents several machine-learning models for early-stage ransomware classification based on pre-attack paranoia activities using API calls as features. They have used different techniques for data representation: Occurrence of Words (OoW), representing the presence/absence of a feature, Bag of Words (BoW), expressing the frequency of a feature, and Sequence of Words (SoW), building a chain of API calls to take into consideration the order in which an API is executed.

The work in [2] presents an ML model for ransomware detection by comparing algorithms like Random Forest, Logistic Regression, Stochastic Gradient Descent, etc. After performing a dynamic analysis using the Intel PIN tool's dynamic binary instrumentation (DBI), features are extracted according to the CF-NCF (Class Frequency - Non-Class Frequency) technique. According to the authors, this process provides higher accuracy during classification experiments. [7] proposes a behavioral classification method by analyzing 150 samples and extracts a set of features and attributes based on reports from `VirusTotal`[3]. The authors of [9] presented a two-stage detection method based on dynamic analysis. The first stage relies on Markov chains, whereas the second on Random Forest. [17] relies on the Term Frequency-Inverse Document Frequency (TF-IDF) of the N-grams extracted from opcodes. They analyze different N-gram feature dimensions using various machine learning models. Similarly, [16] extracts N-grams features from opcodes; but it only uses a Self-Attention Convolutional Neural Network (SA-CNN) to test the approach, which worked well for some long sequences of opcodes.

Despite the promising results in the papers mentioned above, they have several limitations. For instance, the usefulness of the obtained results may be distorted by the limited number of analyzed samples, and the low variability of families included in the training phase may not represent the current ransomware landscape. We try to address these limitations by analyzing a more representative number of samples from 12 different families. Another limitation of some of the presented research is related to the type of analysis they have carried out; [17], and [16] use a static analysis approach for extracting N-grams for their models. This needs to deal with obfuscated ransomware samples, making reverse engineering the most complex step in the model generation process. We resolve this obfuscation problem by using a dynamic analysis approach to capture the pre-attack activities performed by ransomware samples.

Furthermore, our proposal focuses exclusively on ransomware detection, unlike the work in [2, 17, 16]. Second, it presents an early-stage ransomware detection, while, apart from [11], all other works focused on later stages of detection. However, it differs from [11] in the choice of the API calls considered in the detection phase. In addition, we tested our solution on, to the best of our knowledge, the largest ransomware detection dataset that includes 12 different ransomware fami-

---

[3] https://www.virustotal.com

lies, whereas the work on [11] has only been tested on a dataset of 5 ransomware families.

3 Background

In this section, we introduce the main characteristics of ransomware. We will then give a background on neural networks.

3.1 Ransomware

Ransomware is a type of malware that denies access to user files and demands a ransom from the user to regain access to the system and stored information [8]. Ransomware are mainly of two types:

- **Locker**: it prevents the victim from reaching their files by denying access to computing resources (e.g., locking the desktop or preventing the victim from logging in) [8].
- **Crypto**: it encrypts data on the target machine, taking it hostage until the victim pays the ransom and obtains the decryption key from the attacker. Some variants of crypto-ransomware will progressively delete hostage files or release them to the public if the victim fails to pay the ransom on time.

Ransomware can be organized in families depending on their behavior and the type of operations they perform. In the following, we present the main characteristics of well-known ransomware families:

- **Cerber:** it infects computers using common attack vectors, such as phishing e-mails. It comes bundled with free online software. Cerber mainly utilizes malicious Microsoft Office files with macros to spread. Once a victim opens a malicious Microsoft Office document and enables macros, the ransomware begins encrypting victim files.
- **CryptoWall:** once executed, CryptoWall writes its registry autorun keys in the Windows registry to maintain its persistence through reboots. It then searches for all system restore points and Volume Shadow Copy files and destroys them to prevent the victim from restoring any file. Then, it begins encrypting files using the RSA-2048 encryption algorithm.
- **WannaCry:** it is a crypto-ransomware that spreads by exploiting a Windows Server Message Block (SMB) vulnerability that provides unrestricted access to any computer running Windows. WannaCry is also able to propagate throughout corporate LANs automatically. It encrypts files of the infected device and tries to affect other devices in the network.

- **Locky:** the most common technique used by Locky to infect systems is through receiving an e-mail with a malicious Microsoft Word attachment. When this attachment is opened, an executable is downloaded from a C&C server, a private key is generated, and the ransomware starts encrypting files by infecting all connected devices.

All the above families target a single operating system, that is, Windows, that has been shown to be the most targeted operating system.[4] There are also ransomware that target different OSs, like macOS, GNU/Linux, and Android, but the percentage of attacks that target Windows-based machines is very high, compared to other operating systems.

3.2 Artificial Neural Networks

In this work, we rely on Artificial Neural Networks (ANN) for multi-class classification. We chose ANNs because they are lightweight and they give a good detection rate.

ANNs are mainly comprised of many interconnected computational nodes (referred to as neurons), which work in a distributed fashion to collectively learn from the input to optimize the final output. ANNs nodes are organized into layers: an input layer, an output layer, and hidden layers. Nodes in the input layer take a multidimensional vector that is sent to the hidden layers as input. Nodes in the hidden layers make decisions from the previous layer and weigh up how a stochastic change within itself detriments or improves the final output. This is referred to as the process of learning. Having multiple hidden layers stacked upon each other is commonly called *deep learning* [12].

Each layer can have an arbitrary number of neurons and their most important feature is the activation function. In order to output a value, a neuron takes the weighted sum of all its inputs and passes it through the activation function to obtain the results that will be handled to the next layer as input [14]. The role of the activation function is to decide whether a neuron's input is important or not in the process of prediction. The most commonly used activation functions are: ReLu, Linear, Sigmoid, and SoftMax [14]. Another important feature of standard ANNs is that they have a feed-forward architecture, i.e., data flow just in one direction (forward). This is a key characteristic since there are also more complex and advanced neural networks that have feedback: the outputs of the neurons are used as feedback inputs for other neurons [6].

---

[4] https://www.statista.com/statistics/701020/major-operating-systems-targeted-by-ransomware/

4 Methodology

In this section, we discuss the building blocks of our methodology which are: data collection, feature extraction, and classification.

4.1 Data Collection

Ransomware samples. Sample collection has been challenging for different reasons. First, there is no unique online repository that contains all existing ransomware, we had to merge all repositories to avoid duplication. Second, repositories use security vendors' scores and sandboxes results to map ransomware to their respective families. However, these classifications may be incorrect or not accurate, since some ransomware may have similar behavior but a completely different name. Finally, some families, like TeslaCrypt (and its variants like AgentTesla) may contain ransomware samples together with malware that affect the choice of features used for ransomware detection. Thus, they should not be considered for this research since they will affect its effectiveness.

For our samples collection, we used different online repositories (i.e., `VirusTotal`[5],`Malware Bazaar`[6], and `VirusShare`[7]) to obtain a total of 11,523 samples detected in the past few years (i.e., 2018-2022). To have a balanced dataset, the 11,523 collected samples are evenly split into 23 families namely *Ako, BB, Cerber, Conti, Cryptolocker, Cryptowall, Erica, Expiro, Gandcrab, Hive, Kryptik, Lockbit, Lockfile, Locky, Matrix, Matsnu, Shade, Stop, TeslaCrypt, Trik, Virlock, Wannacry, and Winlock*, where 501 samples represent each family.

Benign software. To properly identify ransomware, there is also a need to have some benign software to be included in the classification tasks. In total, we downloaded 1,111 benign samples from different sources (i.e, `The Portable Freeware Collection`[8] and `PortableApps`[9]). We focused on benign software that have similar behavior to ransomware and use a large number of API calls, such as file compressors, disk analyzers, anti-viruses, and password managers. Table 1 shows the distribution of the benign samples.

---

[5] https://www.virustotal.com

[6] https://bazaar.abuse.ch/browse/

[7] https://virusshare.com/

[8] https://www.portablefreeware.com/

[9] https://portableapps.com/

Table 1: Benign samples distribution

Category	Software	Samples
Anti-viruses	McAfee	100
	Others	3
Compressors	7-Zip	28
	PeaZip	99
	Others	3
Graphics	GIMP	79
	Blender	50
	JPEGView	21
	ScribusPortable	19
	Others	4
Text editors	AkelPad	36
	Geany	18
	Notepad 2	33
	Notepad++	100

Category	Software	Samples
Disk analyzers	CrystalDiskMark	36
	Others	4
Browsers	Google Chrome	20
	Others	5
Miscellaneous	Audacity	48
	FileZilla	73
	VeraCrypt	15
	Others	102
Messaging clients	TelegramDesktop	100
Media players	VLC	80
Mail clients	Various	3
Password managers	KeePassXC	23
PDF managers	Various	9
	Total	**1111**

4.2 Features Extraction

We perform a dynamic analysis of the collected samples using widely known tools and techniques for ransomware execution in a controlled environment. We select the features to be used for the classification task from the reports returned by dynamic analysis. In this step, we are interested in studying the usage of API calls that software use to communicate with the kernel. In our context, it is worth noting that a feature is a binary vector representing the usage of a specific API by the analyzed sample. The main challenge in feature extraction is the selection of representative features that could help distinguish different families. The similarity between samples belonging to different families leads to a set of similar features that affect the effectiveness of the developed ML models.

From 12,634 analyzed samples, we removed the ones that failed to execute. Moreover, we removed the ones that belong to underrepresented families (i.e., less than 200 samples), which were 11 families out of 23. We removed these families to keep the dataset balanced. The final dataset contains 5203 samples from 12 ransomware families, and one benign family (see Table 2). These samples contain at least one occurrence of the API calls specified in Table 3.

We have chosen these API calls, shown in Table 3, based on the most used evasion techniques adopted by ransomware, namely process injection, environment sensing, and unpacking. We present each of the evasion techniques in what follows.

Process injection. Code injection is the process of copying the code from an injecting entity ε_{inject} into a victim entity ε_{victim} and executing this code within the scope of ε_{victim} [3]. The definition of a code injection does not specify the place of residence of ε_{inject} and ε_{victim}. We can have two cases: if the attacker and the victim reside on the same system, we refer to Host-Based Code Injection, while if they reside on different systems, the process is called Remote Code Injection.

Table 2: Ransomware curated dataset

Family	**Samples**
Cerber	450
CryptoWall	450
Matsnu	450
Shade	450
Teslacrypt	450
Benign	450
Hive	443
Ako	432
Erica	377
Conti	359
Matrix	331
Gandcrab	295
Expiro	266
Total	**5,203**

Environment sensing. Before executing the malicious payload, usually, an attacker wants to determine if the environment is a virtual one or not [1]. Ransomware use different techniques for evading sandboxes and virtual analysis environments. The first one is fingerprinting, which aims to detect the presence of sandboxes by looking for environmental artifacts that could indicate a virtual/emulated machine. These signs can range from device drivers, overt files on disk, and registry keys, to discrepancies in emulated/virtualized processors. Another technique used in environment sensing is *Reverse Turing Test* which checks for human interaction with the system. This tactic capitalizes on the fact that sandboxes are automated machines with no human or operator directly interacting with them. Thus, if a malware does not observe any human interaction, it presumes to be in a sandbox. The malware waits indefinitely for any form of user input to test whether it is running on a real system. In a real system, eventually, a key would be pressed, or the user would move a mouse. If that occurs a specific number of times, the malware executes its malicious payload [1].

Unpacking. Packing is a widely used technique in malware development that allows attackers to conceal their code. Malware is then transmitted in a “scrambled” form, which is then restored to the original form just before execution using unpacking techniques [5]. Packers use different techniques for obfuscating malicious code. First, they use multi-level compression to obfuscate the payload of an executable, making it hard to perform reverse-engineering tasks on the executable [4]. Moreover, packers can achieve malware polymorphism by producing different binaries, i.e., different hash signatures for the same payload [4, 13]. Encryption is widely used to conceal some parts of the code, which are then decrypted during unpacking by using the encryption keys provided within the packed malware; finally, packers may use techniques like dead code insertion and instruction permutation that aim at making the unpacked malicious executable more challenging to analyze [13].

Table 3: Evasion APIs

Category	**Evasion techniques**	**Evasion API**	**Description**
Data access and storage	Unpacking	MoveFileWithProgressW	Move a file or directory, including its children
	Environment Sensing	NtCreateFile	Creates a new file or directory or opens an existing file
	Process Injection	NtWriteFile	Write data to an open file
		SetFileAttributesW	Sets the attributes for a file or directory
		GetDiskFreeSpaceExW	Retrieve information about the amount of space available on a disk
		GetDiskFreeSpaceW	Retrieves information about the specified disk
		ShellExecuteExW	Perform an operation on a specified file
		DeviceIoControl	Send a control code directly to a specified device driver
Generic OS queries	Environment Sensing	GetComputerNameW	Retrieve the name of the local computer
		NtQuerySystemInformation	Retrieve the specified system information
Memory management	Unpacking	GlobalMemoryStatusEx	Retrieve information about the system memory usage
	Environment Sensing	NtAllocateVirtualMemory	Reserve a region of pages within the user-mode virtual address space
	Process Injection	NtMapViewOfSection	Map specified part of Section Object into process memory
		NtProtectVirtualMemory	Change the protection on a region of committed pages
		NtUnmapViewOfSection	Unmap a view of a section from the virtual address space
		WriteProcessMemory	Writes data to an area of memory in a specified process
		LdrGetDllHandle	Loads a file in memory
Network	Unpacking	GetAdaptersAddresses	Retrieve the addresses associated with the adapters
	Environment Sensing	InternetOpenA	Initialize an application's use of the WinINet functions
Process	Process Injection	CreateProcessInternalW	Create a new process and its primary thread
		NtGetContextThread	Return the user-mode context of the specified thread
		NtResumeThread	Map specified part of Section Object into process memory
		NtSetContextThread	Set the user-mode context of the specified thread
		NtTerminateProcess	Terminate a process and all of its threads
		Process32NextW	Retrieve information about the next process recorded in a snapshot
		NtLoadDriver	Load a driver into the system
Registry	Process Injection	NtSetValueKey	Create or replaces a registry key's value entry
	Environment Sensing	RegOpenKeyExW	Open the specified registry key
		RegQueryValueExW	Retrieve the type and data for the specified value name of a key
		RegSetValueExW	Set the data and type of a specified value under a registry key
		NtCreateKey	Create a new registry key or opens an existing one
Security	Process Injection	CryptGenKey	Generate a random cryptographic session key or a key pair
		CryptExportKey	Export a cryptographic key or a key pair
		LookupPrivilegeValueW	Retrieve the identifier used to represent the specified privilege name
		CryptHashData	Add data to a specified hash object
Services	Environment Sensing	CreateServiceW	Create a service object and adds it to the specified service manager
		EnumServicesStatusW	Enumerate services in the specified service control manager database
UI artifacts	Environment Sensing	SetWindowsHookExW	Install an application-defined hook procedure into a hook chain
		FindWindowW	Retrieve a handle to the top-level window

In Table 3, APIs that end with *W* have twin API that ends with *A* with a similar goal. The difference in the names is due to the encoding. The APIs that end with *W* work with Unicode strings and the ones that end with *A* work with ANSI strings. For the sake of brevity, we included only the Unicode ones.

4.3 Classification

Since each ransomware family has its characteristics, we model the ransomware detection as a multi-class classification problem, where we have different classes (i.e., families), and the classifier will determine the belonging to a specific class. The state-of-the-art classifiers for this problem are Random Forest, Bernoulli Naive Bayes, k-Nearest Neighbors, and Artificial Neural Networks (ANNs). In this paper, we use an ANN (see Section 5.2), since it is lightweight and it gives good accuracy.

Our artificial neural network is composed of three layers with ReLu as an activation function. We use dropout on the input and hidden layers to drop nodes to reduce overfitting randomly. We also add a hidden layer with the Softmax activation function to the network's end.

Figure 1 depicts the model architecture.

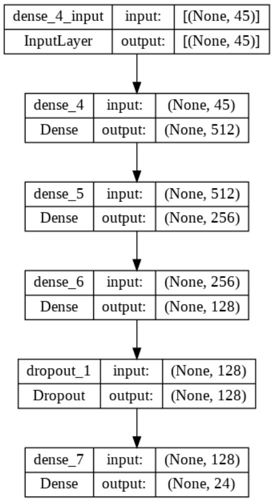
dense_4_input | input: | [(None, 45)]
InputLayer | output: | [(None, 45)]
dense_4 | input: | (None, 45)
Dense | output: | (None, 512)
dense_5 | input: | (None, 512)
Dense | output: | (None, 256)
dense_6 | input: | (None, 256)
Dense | output: | (None, 128)
dropout_1 | input: | (None, 128)
Dropout | output: | (None, 128)
dense_7 | input: | (None, 128)
Dense | output: | (None, 24)

Fig. 1: Classification Model Architecture

The hyperparameters for this network are the number of epochs (i.e., 50) and the batch size (i.e., 15): the former identifies how many times the model will iterate over the whole dataset. At the same time, the latter describes the number of samples after which the network will adjust its internal parameters.

5 Experimental results

To analyze the ransomware samples, we created an Ubuntu virtual machine on which we installed; `Cuckoo Sandbox` (version 2.0.7)[10] the sandbox on which we executed the samples, uses Windows 7 as an operating system with basic software like Internet Explorer and Windows Media Player and sample files like Word documents and PowerPoint slides.

Cuckoo Sandbox is one of the most widely used tools for analyzing the behavior of a malicious executable. The ransomware is run in a controlled virtual machine to capture all performed activities during its execution like API calls, files opened, registry keys, and dumped files. All the behavioral characteristics are then saved to a comprehensive report in JSON format. The report contains additional information about the analysis, such as machine name, operating system, internet access,

---

[10] https://cuckoosandbox.org/

and many other parameters. All the analyzed ransomware samples had access to the internet to contact, if required, their C&C servers for downloading additional malicious payloads.

5.1 Datasets

Our dataset (cfr. Table 2) consists of 5,203 samples distributed across 13 families (including a benign family). In addition, we have used the dataset provided by [11]. This dataset (described in Table 4) is composed of 2,994 ransomware samples from 5 families and 438 benign samples resulting in a total of 3,432 samples.

Table 4: Description of [11] dataset

Family	Reveton	TeslaCrypt	Cerber	Locky	Yakes	Benign	***Total***
Samples	600	600	600	600	594	438	***3432***

5.2 Multi-class Classification

As shown in Table 5, we tested several state-of-the-art classifiers (i.e., RF, BNB, KNN, and ANN). With ANN, we reached good results in terms of accuracy, especially in top-k accuracy ($k = 2$). ANN scores 80.00% in accuracy and 90.41% in top-2 categorical accuracy. For the reported results, we take the weighted average of all individual scores of the classes (i.e., families) we have. Similarly to [11], we used the default scikit-learn metrics[11].

Table 5: Multi-class classification results

Model	**Precision**	**Recall**	**F1-Score**	**Accuracy**	**Top-k Acc. (k=2)**
Random Forest	81.23%	78.38%	78.28%	78.38%	85.82%
Bernoulli Naïve Bayes	61.41%	56.38%	55.94%	56.38%	67.33%
K-Nearest Neighbors	78.39%	75.98%	76.07%	75.98%	82.03%
Artificial Neural Network	82.00%	80.00%	81.00%	**80.00%**	**90.41%**

We then compare our approach with the one presented in [11], since it is the only work we are aware of that that has a public dataset and source code available

---

[11] https://scikit-learn.org/stable/modules/model_evaluation.html

on GitHub[12]. We ran their Random Forest classifier model 5 times on their dataset. In the second step, we took their dataset and used it to train our Artificial Neural Network model. The results we have obtained are promising since the ANN performs very well even with a completely different dataset. The accuracy is 93.00%, and the top-2 categorical accuracy is 98.62%. Table 6 summarizes the results of the comparison.

Table 6: Comparison of our work with [11]

Approach	**Dataset**	**Precision**	**Recall**	**F1-Score**	**Accuracy**	**Top-k Acc. (k=2)**
[11]	[11]	92.36%	92.30%	92.19%	92.30%	97.82%
Our approach	[11]	93.00%	93.00%	93.00%	**93.00%**	**98.62%**
[11]	Our approach	79.29%	78.77%	78.78%	78.77%	86.74%
Our approach	Our approach	82.00%	80.00%	81.00%	**80.00%**	**90.41%**

6 Conclusion

In this paper, we proposed an early-stage ransomware detector based on a neural network model that achieves an accuracy of 80.00% in a challenging, large, and varied dataset, outperforming the state-of-the-art. The dataset we have compiled consists of 4753 ransomware samples from 12 different families and 450 benign samples. To the best of our knowledge, it is the largest dataset available for ransomware detection. We have made publicly available our source code and dataset, to reproduce the results. This work can be extended in many directions. First, we aim to make a decentralized version of it that runs over a blockchain. Second, we plan to explore the effect of adding other features, such as the registry and memory dumps, as input to our model. Third, we aim to explore other ML techniques, like transformers [15] that perform well with huge amounts of data.

Acknowledgements

The authors would like to thank the authors of [11] for their responsiveness and support. In addition, we would like to thank VirusTotal, VirusShare, and Bazaar for providing us with the ransomware samples.

This work has received funding from the Marie Skłodowska-Curie Innovative Training Network Real-time Analytics for Internet of Sports (RAIS), supported

---

[12] https://github.com/Rmayalam/Ransomware_Paranoia

by the European Union's Horizon 2020 research and innovation programme under grant agreement No 813162. Additionally, it has been partially supported by CONCORDIA, the Cybersecurity Competence Network supported by the European Union's Horizon 2020 research and innovation programme under grant agreement No 830927. The content of this paper reflects the views only of their author(s) and the European Commission/Research Executive Agency are not responsible for any use that may be made of the information it contains.

References

[1] Amir Afianian et al. "Malware dynamic analysis evasion techniques: A survey". In: *ACM Computing Surveys (CSUR)* 52.6 (2019), pp. 1–28.

[2] Seong Il Bae, Gyu Bin Lee, and Eul Gyu Im. "Ransomware detection using machine learning algorithms". In: *Concurrency and Computation: Practice and Experience* 32.18 (2020), e5422.

[3] Thomas Barabosch and Elmar Gerhards-Padilla. "Host-based code injection attacks: A popular technique used by malware". In: *2014 9th International Conference on Malicious and Unwanted Software: The Americas (MALWARE)*. IEEE. 2014, pp. 8–17.

[4] S Sibi Chakkaravarthy, D Sangeetha, and V Vaidehi. "A survey on malware analysis and mitigation techniques". In: *Computer Science Review* 32 (2019), pp. 1–23.

[5] Kevin Coogan et al. "Automatic static unpacking of malware binaries". In: *2009 16th Working Conference on Reverse Engineering*. IEEE. 2009, pp. 167–176.

[6] Ivan Nunes Da Silva et al. "Artificial neural networks". In: *Cham: Springer International Publishing* 39 (2017).

[7] Hajredin Daku, Pavol Zavarsky, and Yasir Malik. "Behavioral-based classification and identification of ransomware variants using machine learning". In: *2018 17th IEEE international conference on trust, security and privacy in computing and communications/12th IEEE international conference on big data science and engineering (TrustCom/BigDataSE)*. IEEE. 2018, pp. 1560–1564.

[8] Nihad A Hassan. "Ransomware Families". In: *Ransomware Revealed*. Springer, 2019, pp. 47–68.

[9] Jinsoo Hwang et al. "Two-stage ransomware detection using dynamic analysis and machine learning techniques". In: *Wireless Personal Communications* 112.4 (2020), pp. 2597–2609.

[10] Amin Kharraz et al. "Cutting the gordian knot: A look under the hood of ransomware attacks". In: *International conference on detection of intrusions and malware, and vulnerability assessment*. Springer. 2015, pp. 3–24.

[11] Ricardo Misael Ayala Molina et al. “On Ransomware Family Attribution Using Pre-Attack Paranoia Activities”. In: *IEEE Transactions on Network and Service Management* (2021).

[12] Keiron O’Shea and Ryan Nash. “An introduction to convolutional neural networks”. In: *arXiv preprint arXiv:1511.08458* (2015).

[13] Babak Bashari Rad, Maslin Masrom, and Suhaimi Ibrahim. “Camouflage in malware: from encryption to metamorphism”. In: *International Journal of Computer Science and Network Security* 12.8 (2012), pp. 74–83.

[14] Sagar Sharma, Simone Sharma, and Anidhya Athaiya. “Activation functions in neural networks”. In: *towards data science* 6.12 (2017), pp. 310–316.

[15] Ashish Vaswani et al. “Attention is all you need”. In: *Advances in neural information processing systems* 30 (2017).

[16] Bin Zhang et al. “Ransomware classification using patch-based CNN and self-attention network on embedded N-grams of opcodes”. In: *Future Generation Computer Systems* 110 (2020), pp. 708–720.

[17] Hanqi Zhang et al. “Classification of ransomware families with machine learning based onN-gram of opcodes”. In: *Future Generation Computer Systems* 90 (2019), pp. 211–221.