Early-Stage Ransomware Detection Based on Pre-attack Internal API Calls

Filippo Coglio, Ahmed Lekssays(✉), Barbara Carminati, and Elena Ferrari

Università degli Studi dell'Insubria, Varese, Italy
{fcoglio,alekssays,barbara.carminati,elena.ferrari}@uninsubria.it

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 [10]. 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 type of malware that encrypts user data or restricts 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 [8]. If the attacked entity refuses to pay the ransom, data is deleted or published on the web. Ransomware attacks have become one of the 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, and accept payments in cryptocurrencies to preserve their anonymity [9]. This has turned ransomware into a lucrative tool for attackers who look for financial gains [10].

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.

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

L. Barolli (Ed.): AINA 2023, LNNS 654, pp. 417–429, 2023.
https://doi.org/10.1007/978-3-031-28451-9_36

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 [10]. Thus, based on pre-attack activities, we aim to detect ransomware before the encryption phase. We dynamically analyzed over 11,000 ransomware samples and 1,200 benign samples from 23 different families to extract key API calls used by ransomware before launching attacks. These API calls help in classifying samples into their corresponding ransomware families or benign ones. 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 [10]. 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.

The contributions of this work can be summarized as follows: (i) 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; (ii) 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; (iii) we have made our source code and dataset publicly available[2] to reproduce the results. The remainder of this paper is organized as follows. We discussed state-of-the-art approaches in Sect. 2. Section 3 presents background knowledge on ransomware detection. In Sect. 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, Sect. 6 concludes the paper.

2 Related Work

In the last years, different techniques have been proposed for ransomware classification. [10] 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

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

authors, this process provides higher accuracy during classification experiments. [6] proposes a behavioral classification method by analyzing 150 samples and extracting a set of features and attributes based on reports from [3].

The authors of [8] presented a two-stage detection method based on dynamic analysis. The first stage relies on Markov chains, whereas the second relies on Random Forest.

[16] 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, [15] 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 of the above-mentioned papers, 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 is that some solutions (i.e., [15,16]) 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. 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 [2,15,16]. Second, it focuses on early-stage ransomware detection, whereas all other works, with the exception of [10], focus on later stages of detection. However, our work differs from [10] 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 including 12 different ransomware families, whereas the work on [10] has only been tested on a dataset of 5 ransomware families.

3 Background

We introduce the ransomware and 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 users to regain access to the system and stored information [7]. 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 blocking the logging in) [7].

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

Crypto: it encrypts data on the target machine, holding 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 into 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 and encrypt victim files.

CryptoWall: it 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 on 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, which 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

We rely on Artificial Neural Networks (ANN) for multi-class classification, as these are lightweight and give a good detection rate. ANNs are mainly composed of many interconnected computational nodes (aka neurons), working in a distributed fashion to collectively learn from the input. ANN nodes are divided into layers: input and output layers, as well as various hidden layers between them. Nodes in the input layer take a multidimensional vector as input and send it to the hidden layer. Here, nodes perform nonlinear transformations of inputs

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

and send the results to another hidden layer or to the final output layer. In order to output a value, a neuron in the hidden layer takes the weighted sum of all its inputs and passes it through the activation function to obtain the results [13]. 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 [13]. Having multiple hidden layers stacked upon each other is commonly called *deep learning* [11].

4 Methodology

We discuss the building blocks of our methodology: data collection, feature extraction, and classification.

4.1 Data Collection

Ransomware Samples. Sample collection has been challenging for different reasons. First, since 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 sandbox 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. We obtained a total of 11,523 samples detected in the last few years (i.e., 2018–2022) by using different online repositories (i.e., [5], [6], and [7]). Moreover, 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, we need to have some benign software for the classification tasks.

We downloaded 1,111 benign samples from various sources (i.e., [8] and [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	Category	Software	Samples
Anti-viruses	McAfee	100	Disk analyzers	CrystalDiskMark	36
	Others	3		Others	4
Compressors	7-Zip	28	Browsers	Google Chrome	20
	PeaZip	99		Others	5
	Others	3	Miscellaneous	Audacity	48
Graphics	GIMP	79		FileZilla	73
	Blender	50		VeraCrypt	15
	JPEGView	21		Others	102
	ScribusPortable	19	Messaging clients	TelegramDesktop	100
	Others	4	Media players	VLC	80
Text editors	AkelPad	36	Mail clients	Various	3
	Geany	18	Password managers	KeePassXC	23
	Notepad 2	33	PDF managers	Various	9
	Notepad++	100		**Total**	**1111**

4.2 Features Extraction

We used widely known tools and methods for running ransomware in a controlled environment to perform a dynamic analysis of samples. 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 that shows how the analyzed sample uses the specific APIs. In this step, the main challenge 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 the 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.

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**

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.

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 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 common way for attackers to hide their code when they create malware. Malware is then transmitted in a "scrambled" form, which is then restored to its 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 [4,12]. 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 [12].

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

(*continued*)

Table 3. (*continued*)

Category	Evasion techniques	Evasion API	Description
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 unique characteristics, we model ransomware detection as a multi-class classification problem where the classifier determines which class (i.e., family) the ransomware belongs to. 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 Sect. 5.2), since it is lightweight and gives good accuracy. Our artificial neural network has three layers with ReLu as an activation function. We use dropout on the input and hidden layers to drop nodes and reduce overfitting randomly. We also add a hidden layer with the Softmax activation function to the network's end.

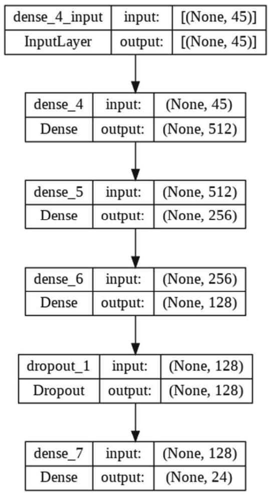
dense_4_input | InputLayer | input: [(None, 45)] | output: [(None, 45)]
dense_4 | Dense | input: (None, 45) | output: (None, 512)
dense_5 | Dense | input: (None, 512) | output: (None, 256)
dense_6 | Dense | input: (None, 256) | output: (None, 128)
dropout_1 | Dropout | input: (None, 128) | output: (None,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

Fig. 1. Classification model architecture

Figure 1 depicts the model architecture. The hyperparameters for this network are: the number of epochs (i.e., 50), indicating how many times the model will iterate over the whole dataset, and the batch size (i.e., 15), denoting the number of samples after which the network will adjust its internal parameters.

5 Experimental Results

As an environment, we used an Ubuntu virtual machine with installed the (version 2.0.7)[10] to execute ransomware samples. It is one of the most widely used tools for analyzing the behaviors of malicious executables. We set up the sandbox with Windows 7 and basic software (e.g., Internet Explorer, Windows Media Player), as well as sample files (e.g., Word documents, PowerPoint slides). The ransomware is run in a controlled virtual machine, keeping track of everything it does, like API calls, files opened, registry keys and files dumped. All the behavioral characteristics are then saved into a JSON report, which also contains the machine name, operating system, internet access, 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.

[10] https://cuckoosandbox.org/.

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 [10], which is composed of 2,994 ransomware samples from 5 families and 438 benign samples resulting in a total of 3,432 samples (cfr. Table 4).

Table 4. Description of [10] dataset

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

5.2 Multi-class Classification

We tested several state-of-the-art classifiers, i.e., RF, BNB, KNN, and ANN (cfr. Table 5. With ANN, we reached good accuracy, especially in top-k accuracy ($k = 2$) (cfr. Table 5. ANN scores 80.00% in accuracy and 90.41% in top-2 categorical accuracy. We took the weighted average of all individual scores of the classes (i.e., families) we have. Similarly to [10], 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 compared our approach with [10], since it is the only work with a public dataset and source[12] We ran their Random Forest classifier 5 times on their dataset. Then, we used their dataset to train our Artificial Neural Network model. The obtained results 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% (see Table 6).

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

[12] Available on Github https://github.com/Rmayalam/Ransomware_Paranoia.

Table 6. Comparison of our work with [10]

Approach	Dataset	Precision	Recall	F1-score	Accuracy	Top-k Acc. (k = 2)
[10]	[10]	92.36%	92.30%	92.19%	92.30%	97.82%
Our approach	[10]	93.00%	93.00%	93.00%	**93.00%**	**98.62%**
[10]	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 [14] that perform well with huge amounts of data.

Acknowledgements. The authors would like to thank the authors of [10] 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 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.

References

1. Afianian, A., Niksefat, S., Sadeghiyan, B., Baptiste, D.: Malware dynamic analysis evasion techniques: a survey. ACM Comput. Surv. **52**(6), 1–28 (2019)
2. Bae, S.I., Lee, G.B., Im, E.G.: Ransomware detection using machine learning algorithms. Concurr. Comput. Pract. Exp. **32**(18), e5422 (2020)
3. Barabosch, T., Gerhards-Padilla, E.: Host-based code injection attacks: a popular technique used by malware. In: 2014 9th International Conference on Malicious and Unwanted Software: The Americas (MALWARE), pp. 8–17. IEEE (2014)
4. Chakkaravarthy, S.S., Sangeetha, D., Vaidehi, V.: A survey on malware analysis and mitigation techniques. Comput. Sci. Rev. **32**, 1–23 (2019)
5. Coogan, K., Debray, S., Kaochar, T., Townsend, G.: Automatic static unpacking of malware binaries. In: 2009 16th Working Conference on Reverse Engineering, pp. 167–176. IEEE (2009)

6. Daku, H., Zavarsky, P., Malik, Y.: 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), pp. 1560–1564. IEEE (2018)
7. Hassan, N.A.: Ransomware families. In: Ransomware Revealed, pp. 47–68. Apress, Berkeley, CA (2019). https://doi.org/10.1007/978-1-4842-4255-1_3
8. Hwang, J., Kim, J., Lee, S., Kim, K.: Two-stage ransomware detection using dynamic analysis and machine learning techniques. Wirel. Pers. Commun. **112**(4), 2597–2609 (2020). https://doi.org/10.1007/s11277-020-07166-9
9. Kharraz, A., Robertson, W., Balzarotti, D., Bilge, L., Kirda, E.: Cutting the Gordian knot: a look under the hood of ransomware attacks. In: Almgren, M., Gulisano, V., Maggi, F. (eds.) DIMVA 2015. LNCS, vol. 9148, pp. 3–24. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-20550-2_1
10. Molina, R.M.A., Torabi, S., Sarieddine, K., Bou-Harb, E., Bouguila, N., Assi, C.: On ransomware family attribution using pre-attack paranoia activities. IEEE Trans. Netw. Serv. Manag. **19**(1), 19–36 (2021)
11. O'Shea, K., Nash, R.: An introduction to convolutional neural networks. arXiv preprint arXiv:1511.08458 (2015)
12. Rad, B.B., Masrom, M., Ibrahim, S.: Camouflage in malware: from encryption to metamorphism. Int. J. Comput. Sci. Netw. Secur. **12**(8), 74–83 (2012)
13. Sharma, S., Sharma, S., Athaiya, A.: Activation functions in neural networks. Towards Data Sci. **6**(12), 310–316 (2017)
14. Vaswani, A., et al.: Attention is all you need. Adv. Neural Inf. Process. Syst. **30** (2017)
15. Zhang, B., et al.: Ransomware classification using patch-based CNN and self-attention network on embedded N-grams of opcodes. Futur. Gener. Comput. Syst. **110**, 708–720 (2020)
16. Zhang, H., et al.: Classification of ransomware families with machine learning based on N-gram of opcodes. Futur. Gener. Comput. Syst. **90**, 211–221 (2019)