Detecting Ransomware using Support Vector Machines

Yuki Takeuchi
Department of Electrical Engineering and Computer Science
Tokyo Metropolitan University
Hino, Tokyo, Japan
takeuchi-yuuki@ed.tmu.ac.jp

Kazuya Sakai
Department of Electrical Engineering and Computer Science
Tokyo Metropolitan University
Hino, Tokyo, Japan
ksakai@tmu.ac.jp

Satoshi Fukumoto
Department of Electrical Engineering and Computer Science
Tokyo Metropolitan University
Hino, Tokyo, Japan
s-fuku@tmu.ac.jp

ABSTRACT

Ransomeware is the most prevalent malicious software in 2017 that encrypts the files in a victim's machine and demands money, i.e., ransom, for decrypting the files. The global damage cost and financial losses of individuals and organizations due to ransomware is increasing year by year. Therefore, fighting against ransomware is an urgent issue. In this paper, we propose a ransomware detection scheme using support vector machines (SVMs), which is one of supervised machine learning algorithms. The key idea of the proposed scheme is to let a SVM learn the API calls of ransomware as its features so that the SVM detects unseen ransomware. Unlike the existing solutions, our scheme looks into the API call history in more detail. The testbeds using real 276 ransomware with Sanbox demonstrate that the proposed scheme improves the correct detection rate of ransomware.

CCS CONCEPTS

• **Security and privacy → Intrusion/anomaly detection and malware mitigation**; **Intrusion detection systems**;

KEYWORDS

Ransomware, support vector machines, malware detection.

ACM Reference Format:
Yuki Takeuchi, Kazuya Sakai, and Satoshi Fukumoto. 2018. Detecting Ransomware using Support Vector Machines. In *ICPP '18 Comp: 47th International Conference on Parallel Processing Companion, August 13–16, 2018, Eugene, OR, USA.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3229710.3229726

1 INTRODUCTION

At present, ransomware is the most prevalent malware spread all over the worlds, which encrypts the files in victim's machines and demands money, i.e., ransom, for the decryption key. The targets of ransomware include not only individuals, but also organizations. The total loss due to ransomware damage was $24 millions in 2015 [7]. However, within just three months between January and March, 2016, the loss reached $209 millions. Amazingly, the total loss further increased to $1 billion in 2017. One reason of this world wide trend is that the victims tend to pay money when the ransom is relatively small with respect to the importance of encrypted files. This motivates adversaries to spread ransomware for money.

Therefore, inventing the counter technologies against ransomware is an urgent issue that security research communities should address. The malware detection mechanisms are generally divided into three categories, surface analysis [15], static analysis [10], and dynamic analysis [3]. Among them, we are interested in dynamic analysis, where a malicious program is executed under a controlled environment, e.g., Sandbox [4], and its behavior was monitored. Then, malicious behaviors are detected in real time.

The existing solutions [12, 13] are based on dynamic analysis. In [13], several indicators such as file hashes, file sizes, and entropy-based metrics, are proposed to identify malicious behaviors. The work in [12] focuses on Microsoft Windows systems, and the API calls (a.k.a. system calls) that ransomware executes are monitored. Based on the characteristic of API calls unique to ransomware, malicious activities are detected. One of the issues of the existing solution is that the API execution logs that two different programs outputs may yield the same vector representations, which characterizes the behaviors of software.

In this paper, we propose a novel ransomware detection scheme using support vector machines (SVMs) for Microsoft Windows systems. SVMs are one of supervised machine learning algorithms which classify an unknown example (a malicious program in this paper) into either of two classes (malicious or benign). The contributions of this paper are listed as follows.

- First, we design vector representation model of the API calls, in which the number of individual API calls is counted by deeply looking into execution logs of malicious programs.
- Second, we further propose a standardized vector representation model to accommodate the diversity of programs.
- Thirds, we propose a ransomware detection scheme using SVMs, which first learns the feature of malicious behavior by using the proposed vector representation models. Then, a SVM is applied to classify unseen ransomare.
- Fourth, we conduct testbeds using real ransomware under a controlled environment with Cuckoo Sandbox [4]. The experimental results demonstrate that the proposed scheme improves the accurate ransomware detection rate.

The rest of this paper is organized as follows. Section 2 provides the preliminary including ransomware and SVMs. We propose a new ransomware detection scheme using SVMs in Section 3. In Section 4, testbeds using real ransomeware under a controlled

ICPP '18 Comp, August 13–16, 2018, Eugene, OR, USA

ACM ISBN 978-1-4503-6523-9/18/08.
https://doi.org/10.1145/3229710.3229726

environment are conducted for performance evaluation. The related works are reviewed in Section 5. Section 6 concludes this paper.

2 PRELIMINARY

2.1 Overview of ransomware

Ransomeware is one of the most destructive malwares that encrypts the files in an infected host and demands money, i.e., ransom, for decrypting the files. In general, a ransomware woks as follows. An adversary deploys a key server which generates a pair of public and private keys, and this sever also works as the command and control server (C& C server) of ransomware. Once a victim gets infected by randomware, the malicious process communicates the C&C server for ransomware to obtain a public key, and then, the files in the victim's machine are encrypted. The corresponding private key is stored in the C&C server, and it is computationally hard for the victim to decrypt the files by herself. The malicious process repeatedly pops a window or shows a text to demand money for the decryption key, i.e., the corresponding private key.

For money transfer, anonymous payment systems, such as cryptogcurrency (e.g., bitcoin [9]), gift cards, and so on, are used for adversaries to evade the low enforcement's investigation. When the amount of ransom is small with respect to the importance of the encrypted files, the victims often pay money. This is one of the reason why the amount of loss due to ransomware is drastically increasing year by year and encourages malicious attempts all over the world.

One variant of ransomware locks the file system. Should this happen, it is, unfortunately, impossible for the victims to access the infected host.

2.2 Support Vector Machines

Support vector machines (SVMs) are one of the most popular surpervised machine learning algorithms, which was invented by Crtes and Vapnik in 1995[1]. For given training data, a SVM algorithm divides the hyperplane of the feature space by the support vector which maximizes the margin. Thus, each training example belongs to either of one or the other category in the feature space. The application of SVMs includes classification, regression analysis, and outliers detection.

SVMs has higher generalization than the other machine learning algorithms. Note that *generalization* in machine learning means the ability to predict unknown data. For example, in Figure 2, a set of data are plotted on a 2-dimensional plane of the feature space. Each data is represented by circles and triangles and classified into either class A or class B. Consider test data represented by a star polygon is added into the future space. The support vector that classifies the test data into class A is represented by the black solid line labeled by support vector 1. On the other hand, when the test data is categorized into class B, the red solid line labeled by support vector 2 will be the support vector. Clearly, the margin of the black slid line is wider than that of the read line. Hence, the black line (support vector 1) is the one that maximizes the margin, and therefore, the test data is classified into class A.

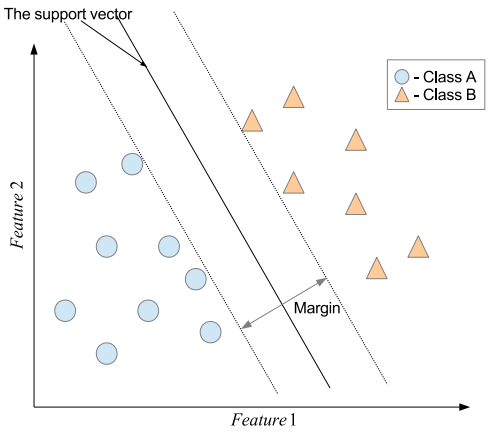

Figure 1: An example of classification with SVMs.

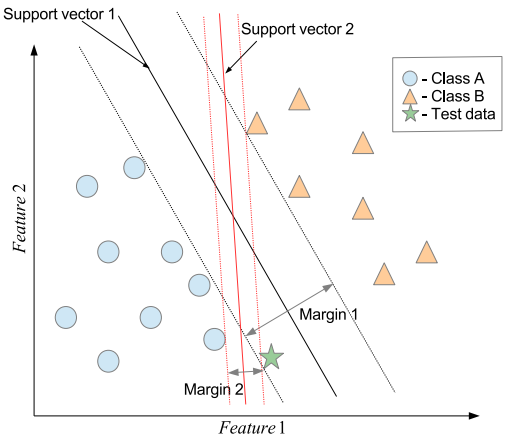

Figure 2: An example of generalization in SVMs.

3 PROPOSED METHOD

3.1 The Overview

In this paper, we propose a SVM-based ransomware detection scheme for Windows systems, which woks as follows. First, the API calls that the existing ransomware execute are transformed to a set of vector data. Then, a SVM learns these vector data as the feature of ransomware. Finally, the proposed scheme detects unknown ransomware as a malicious program.

The proposed detection scheme is different from the existing solutions in the sense how the API calls are represented by a vector. By deeply inspecting the API calls, ours improves the correct ransomware detection rate. The reason why we apply SMVs for learning and classifying ransomware is that SVMs genrally have higher generalization, i.e., the ability to predict unseen data, than the other machine learning algorithms. In other words, SVMs are appropriate for detecting unknown ransomware. Therefore, we apply SVMs to ransomware detection.

Table 1: Definition of notations.

Symbols	Definitions
A	A set of possible API calls
a_i	An API call ($1 \leq i \leq \lvert A \rvert$)
q	The size of q-grams
S	A set of all possible q-gram of API calls
s_i	The i-th q-gram in S
x	A sequence of executed API calls (i.e., a log)
ℓ	The number of executed API calls of log x

3.2 The Interpretation of API Calls

Windows API calls is the lowest level instructions, such as read, write, create, and so on, to use Windows system's functions. In UNIX families, such instructions are called system calls. The execution of any Windows applications is considered as a sequence of API calls. The set of API calls of a program is interpreted as the behavior of the program. For SVMs to learn the features of malicious programs, an API calls log needs to be quantified.

API calls can be represented by vectors. Let $A = \{a_1, a_2, ..., a_{|A|}\}$ be a set of all API calls. We define q consecutive API calls as a q-gram, denoted by $s = (a_1,a_2,...,a_q)$ where $a_i \in A$ and $1 \leq i \leq q$. Let S be a set of q-grams, then S is denoted by Equation 1

$$S = \{(a_1, a_2, \ldots, a_q) | a_i \in A, 1 \leq i \leq q\}. \tag{1}$$

A log of a program execution is represented by a sequence of ℓ API calls, denoted by $x = [a_1, a_2, ..., a_\ell]$, where $a_i \in A$ and $1 \leq i \leq |x|$. We define $\phi : A^\ell \rightarrow \{0, 1\}^{|S|}$ as a function that transforms a set of API calls to a vector with dimension $|S|$.

The key issue of ransomware detection is how to define ϕ. As one possible instance, the work in [12] formulates ϕ as Equations 2 and 3.

$$\phi(x) = (\phi_s(x))_{s \in S}, \tag{2}$$

where

$$\phi_s(x) = \begin{cases} 1 & \text{(if any } q\text{-gram of } x \text{ is equal to } s) \\ 0 & \text{(otherwise).} \end{cases} \tag{3}$$

That is, the i-th element of vector $\phi(x)$ is set to be 1, when the i-th q-gram in S is included in the q-gram representation of x. Otherwise, the i-th element is set to be 0. By doing this, log x of a program execution is transformed into a vector.

Example 3.1. Consider a system with two API calls, i.e., $A = \{a, b\}$. Assume that a program execution outputs log $x = \{a, b, a, b\}$. When $q = 2$, we will have a set of 2-gram defined by $S = \{(a, a), (a, b), (b, a), (b, b)\}$. As the 2-grams of x will be (a, b), (b, a), and (a, b), the corresponding vector of log x is represented by

$$\phi(x) = \begin{pmatrix} 0 \\ 1 \\ 1 \\ 0 \end{pmatrix}. \tag{4}$$

3.3 The Proposed API Calls Analysis

The issue of the existing API calls analysis is that the vector representation of a log provides binary information about q-grams, i.e., whether each q-gram element of API calls is included in x or not. This causes different sequences of API calls might be transformed into the same vector representation as shown in Figure 3. To be specific, consider that the logs ,x_1 and x_2, from two different program executions are recorded as $x_1 = [a, a, a, a, a, a, a, b, a]$ and $x_2 = [a, b, a, a, b, a, a, a, a]$, respectively. While x_1 and x_2 differ 3 points each other, the corresponding vectors will be $\phi(x_1)^T = (1, 1, 1, 0)$ and $\phi(x_2)^T = (1, 1, 1, 0)$, where superscript T indicates for the transposition of a vector. Hence, a malicious program and a benign program might have the same feature, which could be bad training examples for machine learning.

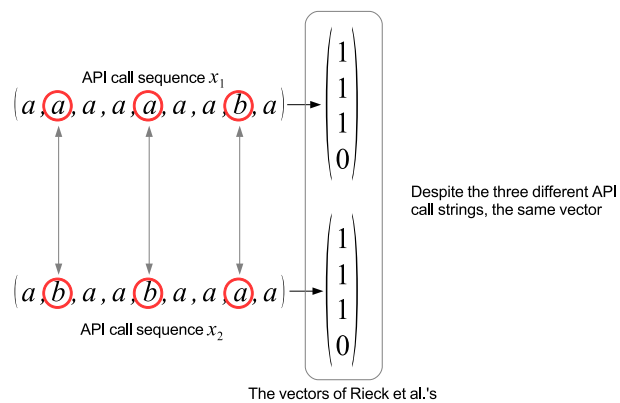

Figure 3: Representation of API calls of the previous research.

In this paper, we deeply looks into the logs and counts the number of q-grams. To this end, function ϕ is redefined. Let $\phi' : A^\ell \rightarrow \mathbb{Z}^{|S|}$ be the function that transforms a log with ℓ API calls to a vector with size $|S|$. Here, $\mathbb{Z}$ is a positive integer set. Let n be the number of q-grams appeared in log x. For given log x, ϕ' is defined as

$$\phi'(x) = (\phi_s{}'(x))_{s \in S}, \tag{5}$$

where $\phi_s{}'(x) = n$.

Example 3.2. In Example 3.1, we have $A = \{a, b\}$ and a set of 2-gram defined by $S = \{(a, a), (a, b), (b, a), (b, b)\}$. Let $x = \{a, b, a, b\}$ a log, and 2-gram of x will be (a, b), (b, a), and (a, b). Since (a, b) and (b, a) appear once and twice, respectively. The corresponding vector of log x is represented by

$$\phi'(x) = \begin{pmatrix} 0 \\ 2 \\ 1 \\ 0 \end{pmatrix}. \tag{6}$$

With the proposed function, two logs shown in Figure 3 yields in different vector representations. Given $x_1 = [a, a, a, a, a, a, a, b, a]$

and $x_2 = [a, b, a, a, b, a, a, a, a]$, the corresponding vectors with 2-grams will be $\phi'(x_1)^T = (6, 1, 1, 0)$ and $\phi'(x_2)^T = (4, 2, 2, 0)$, respectively as shown in Figure 4.

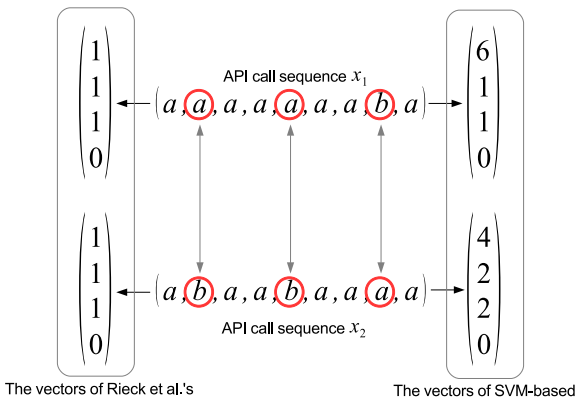

Figure 4: Comparison of vector representation

3.4 Extension to Standardized Vectors

The number of API calls is different from a program to a program. In some program, the number of API calls is of hundreds scale; in others, that could be hundreds of thousands. To accommodate such diversity, we extend the proposed vector representation scheme to the standardized vector representation.

We define $\phi'' : A^\ell \to \mathbb{Z}^{|S|}$ as a function that transforms a log with ℓ API calls to a vector with size $|S|$. For given execution log x, $\phi''(x)$ can be computed as follows. We first computes $\phi'(x)$ as described in Section 3.3. Let μ and σ be the mean and variance of the elements in $\phi'(x)$. Then, μ and σ are calculated by Equations 7 and 8, respectively.

$$\mu = \frac{1}{|S|} \sum_{i=1}^{|S|} \phi_i{}'(x) \tag{7}$$

$$\sigma = \sqrt{\frac{1}{|S|} \sum_{i=1}^{|S|} \left\{ \phi_i{}'(x) - \mu \right\}^2}. \tag{8}$$

Here, $|S|$ is the size of S and $|S| = |A|^q$. The average number of q-grams will equals to 0, and the variance is between 0 and 1. Now, ϕ'' can be obtained by

$$\phi''(x) = (\phi_s{}''(x))_{s \in S}, \tag{9}$$

where $\phi_s{}''(x) = \dfrac{\phi_s{}'(x) - \mu}{\sigma}$.

Example 3.3. In Example 3.2, we have $A = \{a, b\}$ and $S = \{(a, a), (a, b), (b, a), (b, b)\}$ with $q = 2$. Let $x = \{a, b, a, b\}$, and then $\phi'(x)^T = (0, 2, 1, 0)$. From $\phi'(x)$, the mean and variance can be computed by $\mu = 0.75$ and $\sigma = 2.75$, respectively. To derive $\phi''(x)$, we need to compute $\phi_s''(x)$ for $s \in \{(a, a), (a, b), (b, a), (b, b)\}$. For example, for $s = (a, a)$, we will have

$$\begin{aligned} \phi_{(a,a)}{}''(x) &= \frac{\phi_{(a,a)}{}'(x) - \mu}{\sigma} \\ &= \frac{0 - 0.75}{2.75} \\ &= -0.2727. \end{aligned} \tag{10}$$

Similarly, the corresponding vector of log x can be obtained as follows.

$$\phi''(x) = \begin{pmatrix} -0.2727 \\ 0.4545 \\ 0.0909 \\ -0.2727 \end{pmatrix}. \tag{11}$$

3.5 Ransomware Detection

In the proposed randomware detection scheme, a SVM learns the vector representation of the log consists of a sequence of the API calls that ransomware outputs as the feature of malicious programs. Then, for a given log of the API calls that unknown program outputs, the SVM classifies the program into malicious or benign software.

4 PERFORMANCE EVALUATION

In this section, we implement the proposed SVM-based ransomware detection scheme (for simplicity we call the proposed one SVM-based) as well as one of the existing schemes [12] (called Rieck et al.'s) for performance evaluation.

4.1 The Samples of Goodware and Ransomware

We first collected 312 benign software, called *goodware*, and 276 ransomware. Goodware includes office suits, media players, file sharing applications, the Internet browsers, antivirus software, and so on. These software are distributed by reliable vendors and their goodness is acknowledged by an online community [8].

Ransomeware are available at a malware research community [6]. We download the ransomware whose target is Microsoft Windows systems, such as WannaCry [2], Cerber [14], PETYA [11], CryptoLocker [13], and so on.

4.2 Testbed Environment

The testbeds are conducted with real ransomware in a common way for antivirus software developments. As the host machine, we use Ubuntu 16.04 LTS. Ransomware is executed under a safe environment, called *Cuckoo Sandbox* [4], that creates an independent virtual environment in the host machine. Note that Cukoo Sandbox is widely employed in web mail services. As the victim machine, Windows 7 is installed in Cukoo Sandbox.

In a Windows system, 10 folders and 1,000 subfolders are randomly deployed under Desktop, MyDocumnet, the C drive, and so on. These folders contains multiple files including Microsoft document files, pdf files, and so fourth. This emulates typical Windows users. In addition, Python 2.7 is installed in Windows 7 for the victim machine in Cuckoo Sandbox to communicate with the host machine. This configuration is defined as the initial state of the victim machine.

Starting the initial state of the victim machine, either goodware or ransomware is executed, and the logs of API calls are collected.

Table 2: The description of the results.

		Result	
		Ransomware	Goodware
Type	Ransomware	True positive	False negative
	Goodware	False positive	True negative

Table 3: The accuracy and false negative rate.

	SVM-based	Rieck et al.'s
Accuracy	97.48%	94.18%
Missing rate	1.64%	3.08%

The log file contains not only Windows API calls, but also communications status, file accesses, process trees, and so on. From these information, only the API calls are extracted and save as vectors by our custom scripts by the R language.

There are 588 logs, 312 among which are one generated by goodware and 276 among which are generated by ransomware. Randomly selected 294 logs are used for the learning phase of a SVM, and the other 294 logs are used to predict if the sampled program is either malicious or benign. By randomly generating test cases, 1,000 experiments are conducted.

4.3 Evaluation Metrics

For a given API call log of either goodware or ransomware, the proposed and existing ransomware detection algorithms determines if the program is malicious or benign. Thus, the result is either true positive, true negative, false positive, or false negative as shown in Table 2. The detection is considered as correct, when ransomware is detected as a malicious program (i.e., true positive) or goodware is detected as a benign program (i.e., true negative). On the other hand, the detection fails, if ransomware is detected as a benign program (i.e., false negative) or goodware is detected as a malicious program (i.e., false positive).

We uses two metrics for evaluating ransomware detection schemes. One is accuracy, which indicates how much a scheme correctly predicts; the other is the missing rate, which is the probability of ransomwere being undetected. A test is said to be *accurate* when the result is either true positive or true negative. The accuracy is defined by the number of true positive and true negative results divided by the total number of tests. The missing rate is simply computed by the number of false negative results divided by the total number of tested ransomware programs.

4.4 Testbed Results

Table 3 shows the accuracy and the missing rate for Rieck et al.'s and SVM-based. As can be seen the table, the accuracy of SVM-based and Rieck et al.'s are 97.48% and 94.18%, respectively. In other words, SVM-based successfully identifies ransomware as malicious programs and goodware as benign programs than Rieck et al.'s does by 3.3%. In addition, the missing rate of SVM-based decreases from 3.08% to 1.64%. Thus, the probability of ransomware being undetected decreases.

Table 4: The accuracy and false negative rate using standardized vectors.

	SVM-based	Rieck et al.'s
Accuracy	93.52%	96.42%
Missing rate	2.69%	4.06%

Table 4 illustrates the accuracy and missing rate for Rieck et al.'s and SVM-based, when the standardized vector representation is used. As seen in the table, the proposed SVM-based still provides higher accuracy than Rieck et al.'s does by 2.90%. In addition, the mission rate of the SVM-based is smaller than that of Rieck et al.'s by 1.37%. The standardized vector representation does not help much compared with the original vector representation. This is because the number of ransomware and goodware is only 588, and their characteristics of API calls are not diverse enough. However, we claim that a scaled metric such as the standardized vector representation is always required to accommodate diversity.

5 RELATED WORKS

5.1 Malware Analyses

Malware analyses are generally categorized into either of three types, surface analysis, static analysis, or dynamic analysis, as follows.

In surface analysis, quantified indicators are used to detect software as a suspicious program. To be specific, the hash value or file type of the software programs, which have been recognized as malware, are stored in the database, called virus definitions or signatures. Then, should suspicious programs have the same hash value or file type as the existing malware in the database, these programs are detected. The disadvantage of this type is that variants and/or subspecies of the existing malware cannot be detected, since they have a different hash value from their ancestors due to the modification of the original source codes.

In static analysis (a.k.a. white box analysis), the source codes of suspicious software is analyzed without execution. The advantage of this type is that the suspicious programs are not required to be executed, and thus, the analysis is safe. In addition, all the functions in the program are inspected without execution. Since the source code is inspected line by line, the analysis, unfortunately, take a long time and requires high leveled skills.

In dynamic analysis (a.k.a. black box analysis), suspicious software is executed and its behavior is analyzed. To do this, suspicious programs must be executed for analysis, which exposes the network or computer systems to danger. Therefore, the dynamic analysis is performed under a controlled environment, such as Sandbox [4].

In commercialized antivirus software, malware are detected by the combination of the above analyses. The ransomware detection scheme proposed in this paper is categorized into the dynamic analysis.

5.2 Ransomware Detection

Since ransomware is relatively a new type of malware, to the best of our knowledge, a only few works have been made so far. Nolen et al. [13] designs a set of indicators for ransomware detection,

including the change of a file type, the similarity of files, and the Shannon's entropy of files. For clarification, a file type is not just header information, but the bytes unique to a file. The file type of a targeted file shall change after encryption. The encrypted files that ransomware creates have something in common, which can be measured by the similarity. These indicators are quantified by the Shannon's entropy. To distinguish whether a file is encrypted by ransomware or by a user, the above indicators are combined for malware detection.

The works in [12] detects malicious activities of ransomware in Microsfot Windows systems by inspecting API calls. The API calls that ransomware executes are extracted as its feature, and then, the characteristics of API calls are used as indicator of malicious activities.

The work in [5] also proposes a malware scheme using one of the supervised machine learning algorithms, called *perceptrons*. The perceptron is a binary classifier that returns whether or not an input data belongs to a particular class based on a set of weights with feature vectors. The proposed scheme in [5] based on static analysis, which may take a long time. Thus, this category is out of scope of this paper.

6 CONCLUSION

In this paper, we propose a ransomware detection scheme using SVMs. The vector representations of the API call logs resulting form ransomware executions are used as training examples for a SVM. The key difference form the exiting solution is that the proposed SVM-based scheme deeply inspects the sequences of API calls, and our vector representations include the number of q-grams in the execution logs. By doing this, unknown ransomware is effectively detected in keeping with a smaller probability of malicious programs being undetected. We have conducted dynamic analysis experiments using real ransomware prevalent all over the world under a safe environment using Sandbox. The testbed results demonstrate that the proposed scheme improves the accuracy of predictions and decreases the missing rate of ranosmware.

ACKNOWLEDGMENT

This research has been funded in part by the Okawa Foundation for Information and Telecommications Grant Number 17-04.

REFERENCES

[1] Corinna Cortes and Vladimir Vapnik. 1995. Support-Vector Networks. *Journal of Machine Learning* 20, 3 (1995), 273–297.
[2] Jesse M Ehrenfeld. 2017. Wannacry, Cybersecurity and Health Information Technology: A Time to Act. *Journal of Medical Systems* 41, 7 (2017), 104.
[3] Ivan Firdausi, Alva Erwin, Anto Satriyo Nugroho, et al. 2010. Analysis of Machine Learning Techniques Used in Behavior-Based Malware Detection. In *Advances in Computing, Control and Telecommunication Technologies*. IEEE, 201–203.
[4] Cuckoo Foundation. Accessed 4-20-2018. Cuckoo Sandbox - Automated Malware Analysis. https://cuckoosandbox.org/. (Accessed 4-20-2018).
[5] Dragoş Gavriluţ, Mihai Cimpoeşu, Dan Anton, and Liviu Ciortuz. 2009. Malware Detection Using Machine Learning. In *International Multiconference on Computer Science and Information Technology*. IEEE, 735–741.
[6] Inc. Hybrid Analysis GmbH. Accessed 4-20-2018. Free Automated Malware Analysis Service - powered by Falcon Sandbox. https://www.hybrid-analysis.com/. (Accessed 4-20-2018).
[7] IBM. 2016. Ransomware: How Consumers and Businesses Value Their Data. (2016), 6 pages.
[8] Inc. Informer Technologies. Accessed 4-20-2018. Must-have software for windows - download for free. http://software.informer.com/software/. (Accessed 4-20-2018).
[9] Kevin Liao, Ziming Zhao, Adam Doupé, and Gail-Joon Ahn. 2016. Behind Closed Doors: Measurement and Analysis of CryptoLocker Ransoms in Bitcoin. In *Electronic Crime Research*. IEEE, 1–13.
[10] Andreas Moser, Christopher Kruegel, and Engin Kirda. 2007. Limits of Static Analysis for Malware Detection. In *Computer Security Applications Conference*. IEEE, 421–430.
[11] R. Richardson and M. North. 2017. Ransomware: Evolution, mitigation and prevention. *International Management Review* 13, 1 (2017), 10.
[12] Konrad Rieck, Philipp Trinius, Carsten Willems, and Thorsten Holz. 2011. *Journal of Computer Security* 19, 4 (2011), 639–668.
[13] Nolen Scaife, Henry Carter, Patrick Traynor, and Kevin RB Butler. 2016. Cryptolock (and drop it): stopping ransomware attacks on user data. In *International Conference on Distributed Computing Systems*. IEEE, 303–312.
[14] Daniele Sgandurra, Luis Muñoz-González, Rabih Mohsen, and Emil C. Lupu. 2016. Automated Dynamic Analysis of Ransomware: Benefits, Limitations and Use for Detection. *Computing Research Repository* (2016), 104.
[15] P. Vinod, Rajasthan Jaipur, V. Laxmi, and M. Gaur. 2009. Survey on Malware Detection Methods. In *3rd Hacker's Workshop on Computer and Internet Security*. 74–79.