Investigation of Malicious Portable Executable File Detection on the Network using Supervised Learning Techniques

Rushabh Vyas, Xiao Luo, Nichole McFarland, Connie Justice
Department of Information and Technology, Purdue School of Engineering and Technology
IUPUI,
Indianapolis, IN, USA 46202
Emails: rushvyas@umail.iu.edu; luo25@iupui.edu; mcfarlni@umail.iu.edu; cjustice@iupui.edu

***Abstract*—Malware continues to be a critical concern for everyone from home users to enterprises. Today, most devices are connected through networks to the Internet. Therefore, malicious code can easily and rapidly spread. The objective of this paper is to examine how malicious portable executable (PE) files can be detected on the network by utilizing machine learning algorithms. The efficiency and effectiveness of the network detection rely on the number of features and the learning algorithms. In this work, we examined 28 features extracted from metadata, packing, imported DLLs and functions of four different types of PE files for malware detection. The returned results showed that the proposed system can achieve 98.7% detection rates, 1.8% false positive rate, and with an average scanning speed of 0.5 seconds per file in our testing environment.**

I. Introduction

Malware is an application that is harmful to your computer. Malware analysis is the process of analyzing the behaviors of malicious code and then create signatures to detect and defend against it. Generally, there are two types of malware analysis and detection mechanisms: static feature based and dynamic feature based. Literature shows that static feature based analysis is effective and efficient [1] [2] [3], and enables network detection by loading the algorithm into the memory. However, it is challenging when the malicious file or code is packed or encrypted. Hence, the CPU instructions need to be unpacked or decrypted then dynamic feature analysis needs to be employed. One of the requirements of dynamic analysis is to execute the file in a controlled environment such as virtual machine or sandbox. On the other hand, given that the network traffic is high speed, the dynamic analysis might not be feasible for network malware detection.

In this paper, we investigated the static feature based malware detection by using different supervised learning algorithms, and proposed a network malware detection process for real time malware detection on the network. In this work, we targeted malicious PE file detection with a small number of features. We investigated how much we could push the supervised learning techniques towards malware detection while minimizing the computational cost for network malware detection. Four supervised learning techniques were used: Random Forest, Decision Tree, k-Nearest-Neighbor and Support Vector Machines. The ten folder cross validation results showed that random forest learning technique achieved best detection performance than the other three learning techniques. The achieved detection rate and false alarm rate on experimental data set were 98.7% and 1.8% respectively. We compared the performances of the four learning techniques on four types of PE malware - backdoors, viruses, worms and trojans. The results showed that the same learning technique show the similar performance on different types of malwares. It demonstrated the features that are extracted from the files have no bias for a specific type of malware. Lastly, we proposed a network detection process and evaluated the feasibilities of network detection of the best performed learning technique identified in the experiments. Based on the computational environment we used for this work, we achieved an average scanning rate of 0.5 seconds per file on average.

The rest of paper is organized as follows. Section II summarizes the related work. The feature extraction and feature construction are presented in section III. The supervised learning techniques that are explored in this work are introduced in Section IV. Section V presents the experimental setups and results. Conclusions and future work are discussed in Section VI.

II. Related Work

In the literature, there are several attempts to improve the performance of PE malware detection using machine learning algorithms. Shafiq *et al.* proposed PE-Miner in 2009 for PE malware detection. They first extracted 189 features from the PE file segments [1]. Then, feature selection/reduction algorithms, e.g. Principle Component Analysis (PCA) were used to select the most relevant features. The five supervised learning algorithms which include IBk, J48, NB, RIPPER, and SMO, are deployed for seven types of malicious executable detection. The seven types are backdoor + sniffer, Constructor + Virtool, DoS + Nuker, Flooder, Exploit + Hacktool, Work, Trojan and Virus. The best results (99% detection rate and 0.5% false positive rate) were achieved on detection virus. However, no discussion has been given on how to deploy these learning algorithms for network detection.

Lo, Pablo, and Carlos investigated the minimum features that need to be used for PE malware detection [2]. They

978-3-901882-89-0 @2017 IFIP

concluded that with “9” features, they can achieve 99% accuracy on malware detection by using an assembly classification schema. They also compared their performance against the other algorithms in the literature. However, their base feature pool was generated with an external software named virualtotal [4]. So, the overall performance relies on viraultotal. Additionally, the system was not compared against different types of malware detection.

Liu *et al.* compared Naive Bayes, Support Vector Machines, Decision Tree, Boosted Decision Trees (Adaboost + J48) and individual Cost Sensitive Twin One-Class Classifier by making use of both static features and dynamic features: Op-code n-grams, API n-grams and embedded behavior graphs [5]. They gained about 95% accuracy with about 2% false positive rate. The feasibility of these algorithms for in network detection was not discussed.

Shalaginov *et al.* compared Neuro-Fuzzy and Artificial Neural Network (ANN) on detection of ten malware families and ten malware categories [3]. Similar to the work of Lo *et al.* [2], they have relied on viraultotal [4] to extract features. It is not applicable for network malware detection. They achieved detection rate and false positive rate are 81% and 20% respectively on malware family hupigon.

Ding *et al.* considered using opcode based n-gram features for PE file malware detection [6]. They proposed a deep belief networks learning algorithm and compared it against Support Vector Machines (SVMs), k-NN, and Decision Tree by using up to 400 n-grams. Their results showed 96.7% accuracy.

A gap in the literature showed no investigation has been done using a small number of static features that were directly extracted from the PE files, so that on network malware detection can be done in an efficient manner. The contribution of this research is investigation and exploring on network PE malware detection using a small number of features and four different supervised learning techniques.

III. Feature Extraction and Feature Representation

The PE files were those files generally run on the Windows platform with extension of .exe or .dll. PE files are divided into the PE file header, section table and sections. In the PE file header, the compile timestamp, number of entries in section table, required processor type can be found. There are multiple sections, for example text section (executable code), data sections (.bss, .rdata, and .data), resource section (.rsrc),export section (.edata), import section (.idata) and so on [7].

A. Feature Extraction

In this work, based on the features that are extracted from PE header and sections, we re-grouped the extracted features into four categories - file metadata, file packing, imported dlls, and imported functions. The details of features from each category are given in the following sub-sections.

TABLE I
Meta Data Features

Feature Name	Explanation and Examples
Number Of Sections	The NumberOfSections field in the PE header. It is the number of entries in section table.
Number of DLLs	Total number of DLL files imported in the import section.
Number of Functions	Total number of functions imported in the import section.
Compile Time	The TimeDateStamp field in the PE header. It is the time and date when the file was compiled.
Compile Time Indicator	1 - TimeDateStamp $>$ current date; 0 - otherwise.
Address Of Entry Point	The AddressOfEntryPoint field in the PE header. It is the Offset of the entry point of the PE file relative to the image’s base address when it is loaded into memory.
Base Of Code	The BaseOfCode field in the PE header. It is the Offset of the beginning of the code section relative to the image’s base address when loaded into memory.
Number Of Symbols	The NumberOfSymbols field in the PE header. It is the number of entries in the symbol table.

1) File Metadata: The file metadata category contains some basic information about the PE file, for example the date on which the file was compiled, the total number of DLLs that are imported and so on. Specifically, we extracted eight metadata features from a PE file, which are given in table I. The basic information features may not independently imply that a PE file is malicious or not. However, based on the literature [8], the combinations of them might be strong indicators of malware. Based on the TimeDateStamp, we added a derived feature ‘Compile Time Indicator’. When binaries are compiled, the compiler can put compile date and time in the file. Compile date and time can be manipulated by the attacker. If a binary file was compiled in the future, it is suspicious and the feature ‘Compile Time Indicator’ is marked as 1. NumberOfSymbols was selected as a feature due to the fact that sometimes malware authors strip the symbols from the binary to make analysis harder.

2) File Packing: Sometimes, the author of the malicious file uses a packer software to first parses PE internal structures. Then, it reorganizes PE headers, sections, import tables, and export tables into new structures. During packing, a packer software sometimes encrypts the code and resource sections using the compression and encryption libraries [9]. The PE files are packed in a way that it makes it very difficult to reverse engineer, or the anti-virus programs to figure out whether the PE file is malicious or not. If PE files are packed using a packer software, the packing related or generated attributes are extracted. We extracted three file packing related features which are given in table II. Based on the literature [10], high entropy may imply packed malware. Hence, Shannon entropy for each file is calculated as a feature. The formula of the Shannon entropy calculation is given as formula 1, where f is the binary representation of the PE file, $p(i)$ is the probability of i^{th} unit of the binary file f with total n binary values.

TABLE II
FILE PACKING FEATURES

Feature Name	Explanation and Examples
Shannon Entropy	Shannon Entropy of the PE file binary.
SizeOfRawData	The SizeOfRawData field in each section. 0 - If one SizeOfRawData in any section is 0, 1 - otherwise.
Section name	1 - Contains UPX, ASPack, FSG, or MPRESS, 0 - otherwise.

$$H(f) = -\sum_{i=1}^{n} p(i) log_2 p(i) \tag{1}$$

Sometimes PE files are packed and the way they are packed causes them to have 0 as SizeOfRawData. When this is the case, virtual size for the section is higher. When the executable runs, the space allocated in the memory by the OS is the same size as virtual size, the malicious code then unpacks code into the memory [11]. So, SizeOfRawData for each section is extracted and examined whether the value is 0 or not. In addition to the entropy of the file and SizeOfRawData of each section, section names were also examined to see if any of them contain popular packers such as UPX, ASPack, FSG, MPRESS, since they are commonly utilized by malware actors to pack their malicious files. These features were assigned values 1 or 0 depending on if the PE file contains these names or not.

3) Imported DLL Files: DLL files can quickly tell the functional intention of the PE file. For example, if wsock32.dll is imported, it was assumed that the PE file does something network related. Hence, in this work, the specific DLLs or the combinations of the DLLs were extracted and used as one of the feature. The DLLs include kernel32.dll, advapi32.dll, user32.dll, gdi32.dll, ws2_32.dll, ntdll.dll, crypt32.dll, shell32.dll, wsock32.dll, wininet.dll and msvcrt.dll.

4) Imported Functions: Other than the imported DLLs, imported functions are also extracted from the PE files. Although there are no functions that are specific to malware, some of the functions are typically used by malware [12]. The function types that were investigated were registry, anti-analysis, packing, command or process execution, keylogging, networking, screenshot, cryptographic, privilege escalation, process or memory manipulation, service manipulation, information gathering, file or process creation, DLL related, finding, and persistence. The specific functions that were extracted for each function type are detailed in table III.

B. Feature Representation

After feature extraction, each file was represented in a proposed vector format. Each entry of the vector presented the corresponding feature. Figure 1 demonstrated the proposed vector presentation of a file. There are total of 28 entries in the vector presenting the features in the four categories described in the previous section.

It is worth to noting that instead of using eleven entries in figure 1 to present the existence of each, we only used

TABLE III
IMPORTED FUNCTION TYPES AND FUNCTIONS

Function type	Functions
Registry	RegCloseKey, RegOpenKey, RegQueryValue, RegSetValue, RtlCreateRegistryKey, RtlWriteRegistryValue
Anti-Analysis	CheckRemoteDebuggerPresent, FindWindow, GetLastError, IsDebuggerPresent, sleep, OutputDebugString, GetAdaptersInfo, FindWindow, GetTickCount, NtSettInformationProcess DebugActiveProcess, QueryPerformanceCounter, NtQueryInformationProcess
Packing	VirtualAllocEx, LoadLibrary, VirtualFree, GetProcAddress, LdrLoadDll, LoadResource VirtualProtectEx
Execution	CommandLineToArg, ShellExecute, system, WinExec
KeyLogging	SetWindowsHook, RegisterHotKey, GetKeyState, MapVirtualKey
Networking	listen, socket, accept, bind, connect, send, recv, FtpPutFile, InternetOpen, InternetOpenUrl, InternetWriteFile, ConnetNamedPipe, PeekNamedPike, gethostbyname, inet_addr InternetReadFie
Screenshot	BitBlt, GetDC
Crypto	CryptDecrypt, CryptGenRandom, CryptAcqureContext
Privilege Escalation	SetPrivilege, LookupPrivilege
Process Manipulation	CreateRemoteThread, WriteProcessMemory, ReadProcessMemory, OpenProcess, NtOpenProcess, NtReadVirtualMemory, NtWriteVirtualMemory
File/process creation	CreateFile, CreateFileMapping, CreateMutex, CreateProcess
Service manipulation	CreateService, ControlService OpenSCManager, StartServiceCtrlDispatcher
Process/memory manipulation	CreateRemoteThread, WriteProcessMemory, ReadProcessMemory, OpenProcess, NtOpenProcess, NtReadVirtualMemory, NtWriteVirtualMemory, MapViewofFile, Module32First, Module32Next, OpenMutex, OpenProcess, QueueUserAPC, SetFileTime, SfcTerminateWeatherThread, SuspendThread, Thread32First, Thread32Next, WriteProcessMemory ResumeThread
DLL	DllCanUnloadNow, DllGetClassObject, DllInstall, DllRegisterServer, DllUnregisterServer
Persistent	NetScheduleJobAdd
Information Gathering	FindFirstFile, FindNextFile, FindResource, WSAStartup

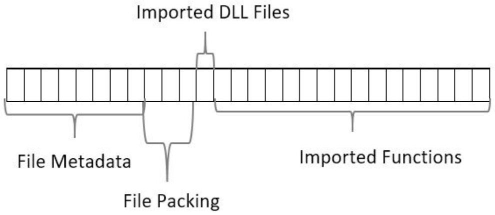

Fig. 1. The Proposed File Representation

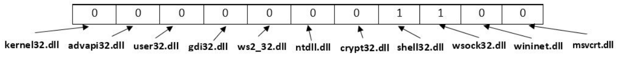

Fig. 2. Example of Imported DLLs feature representation

one entry in the vector to present all the extracted imported DLLs. We proposed to use a binary string to present the imported DLLs in the file, and then transfer the binary to an integer. The reason behind was to reduce the size of the vector for file representation, so that the computational cost of training can be reduced. For example, if a PE file imported only crypt32.dll and shell32.dll, figure 2 represents the binary string of existence of these DLLs. Only two entries in the binary string are 1, the rest are 0. The corresponding integer value is 12. Noted that transferring the binary presentation to decimal presentation might effects the similarity measures between the instances, we have done experiments to compare the two presentations. We identified that this transformation has very minimum affects on the detection performances of the learning techniques we chose for this study. Same strategy was used to present the value of the imported function types as described in table III.

IV. Learning Algorithms

The task of malware detection classified files into two classes: malware or benign. Hence, in this research, we deployed four supervised learning techniques (classification models) for the task of malware detection. Specifically, they were k-Nearest Neighbors (kNN), decision tree, support vector machines and random forest. The overviews of the learning techniques are given in the following sections.

A. k-Nearest Neighbors

k-Nearest Neighbors (k-NN) learning technique has been used in pattern recognition and classification since the beginning of 1970s as a non-parametric technique. It has been studied extensively for data mining and malware analysis tasks [13] [14]. k-NN uses a majority vote of its neighbors to classify an object. The object being classified is most common among its k nearest neighbors. k is a positive integer. If $k = 1$, then the object is simply assigned to the class of that single nearest neighbor. Typically, we experiment with a different k to find the optimal k value to classify a given data set. The neighborhood distance function varies from different implementation of k-NN. The typical one is Euclidean distance which was used in this work. The k value was three in this work.

B. Decision Tree

Decision tree learning techniques have been used for malware detection in the literature and archived competitive performances [1] [15]. There were different decision tree algorithms, such as ID3, C4.5, C5.0 and CART (Classification and Regression Trees). One of the differences between decision tree algorithms and other supervised learning algorithms is that the trained model of the decision tree can be interpreted and visualized as a tree structured form, while others are more like black boxes, the trained model could not be easily interpreted. In this work, optimized CART which was implemented in python Scikit-Learn package, was employed. CART was very similar to C4.5. The difference was that it supports numerical target variables (regression) and does not compute rule sets. CART constructs binary trees using the feature and threshold that yielded the largest information gain at each node [16].

C. Support Vector Machines

Support Vector Machines (SVMs) [17] was one of the classification algorithms evaluated. SVMs was a large margin classifier, and has been widely used in many different data analytic tasks and malware detections [13] [15]. The SVMs classifier aims to separate the input data using hyperplanes. To generate a less complex hyperplane function for classification, the maximum margin between the hyperplane and the support vectors was required. When the samples were not linearly separable, SVMs were used to non-linearly transform the training features from a two dimensional space 'x' to a higher dimensional feature space '$\phi(x)$' using a factor $\varphi : x \rightarrow \phi(x)$ and a function called 'Kernel', an inner product of two examples in the feature space. The ability to learn from large feature spaces and the dimensionality independence make the SVMs a universal learner for data classification [18].

D. Random Forests

Random forests algorithm was very similar to decision tree algorithms. The one main difference was the number of trees that were used to determine the error. Research has shown that when decision trees grow very deep to learn irregular patterns, they can over fit to the training sets [19]. Random forests take one third of the data from the sample and used them to calculate an unbiased estimate of error. Then, it used the information while creating the next tree to improve accuracy. The random forests algorithm provided a way of averaging deep decision trees with the goal of reducing the variance [19]. The combination of the multiple trees created a "forest" which gave an accurate approximation of the data. On the other hand, they also showed tree like models for visualizing the results. Random forests have also been used for malware detection in literature [15].

V. Experimental Setups and Results

The malware samples that were included in the experiments conducted were originally collected by VXHeaven [20]. The datasets were published as VX Heaven Virus Collection in 2010. The total amount of malware that was originally collected was 47 GB when compressed. VX Heaven collected known malware samples and gathered them for informational and educational purposes. 10,400 malicious files of four different categories were used in this research. The categories were worm, backdoor, virus and trojan. We also collected 1,100 benign files from a Windows 7 system, Ninite.com and downloaded applications that were known benign. The details

TABLE IV
OVERVIEW OF THE DATA COLLECTION

	Malware Type	Instances
Malware	Virus	2,600
	Backdoor	2,600
	Worm	2,600
	Trojan	2,600
Benign		1,100

TABLE V
DETECTION RATES OF DIFFERENT TYPES OF MALWARE

Malware Type	kNN	Decision Tree	Random Forest	SVM
worm	0.971	0.979	0.989	0.726
trojan	0.970	0.972	0.987	0.726
backdoor	0.970	0.980	0.990	0.726
virus	0.961	0.979	0.983	0.726
Weighted-Average	0.968	0.978	0.987	0.726

of each type of malicious and benign files were noted in table IV.

To fully evaluate the learning algorithms for malware detection, ten folder cross validation was used to train and build the learning models. The false positive rate (FPR) and detection rate (DR) were the most common metrics for evaluating malware detection or anomaly detection systems. They were estimated as equations 2 and 3. Tables V and VI show detection rates and false positive rates of the four learning algorithms on different types of malware respectively.

$$FPR = \frac{Number\ of\ Benign\ Detected\ as\ Malware}{Total\ Number\ of\ Benign} \tag{2}$$

$$DR = \frac{Number\ of\ Detected\ Malware}{Total\ Number\ of\ Malware} \tag{3}$$

Based on the returned results, random forest performed better than the other three algorithms in all cases. It gained 98.7% of weighted-average detection rate and 1.8% of weighted-average false positive rate. The SVMs performed the worse in all the cases. One reason was in building the learning models, the linear kernel for the SVMs was deployed. On the other hand, all the learning algorithms gained better detection rates on backdoor detection and better false positive rates on worm detection. The performances of each learning technique on all four types of malware were consistent which demonstrates that the extracted features presented characteristics of each malware type.

A. Network Malware Detection

Today, files and information are transferred and exchanged through the network and Internet, hence network malware detection is critical. This research investigated the feasibility of using supervised learning algorithm for network malware detection. A proposed network malware detection process is demonstrated in figure 3. For any incoming network traffic that passed through the router to the internal network, BroIDS [21], a network analysis tool was used to extract the PE files from the network traffic. Afterwards, the feature extraction module was used to extract the features that were described in the previous section. Once the features are extracted and the file is represented using the features, a REST API is used to submit the newly represented file to the malware detection engine which has the trained learning algorithm loaded into the memory. If it was detected as malware, an alert will be triggered and sent to the operator.

TABLE VI
FALSE POSITIVE RATES OF DIFFERENT TYPES OF MALWARE

Malware Type	kNN	Decision Tree	Random Forest	SVM
worm	0.029	0.021	0.011	0.274
trojan	0.030	0.028	0.013	0.274
backdoor	0.010	0.030	0.030	0.274
virus	0.039	0.021	0.017	0.274
Weighted-Average	0.027	0.025	0.018	0.274

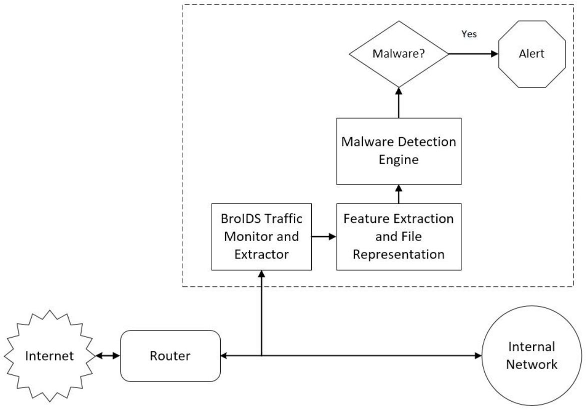
Malware?
Yes
Alert
Malware Detection Engine
BroIDS Traffic Monitor and Extractor
Feature Extraction and File Representation
Internet
Router
Internal Network

Fig. 3. Proposed Online Malware Detection Framework

From the previous results, we concluded that random forest outperformed the other algorithms. Hence, random forests learning technique was chosen for the network file analysis. To keep file analysis and detection time minimal for the network detection, we experimented with the number of trees from 1 to 100 to find an optimal number of trees without sacrificing the performances and limiting the computational cost. Figure 4 shows that the performance of the random forest stabilized when the number of trees reached approximately 10. Consequently, we deployed random forest model with 10 trees in our process. After the training process was done, the 10 tree structures were loaded into the memory for malicious file detection. The learning model will be re-trained periodically by including more input PE file instances and then reloaded into the memory.

We used a laptop machine that has 12GB memory and an Intel Core i72670 QM Processor which has 6 MB Cache, 4 cores and up to 3.10 GHz of Max Turbo Frequency as malware detection server to test the efficiency of file analysis and detection process. The training time for all four learning techniques: k-NN, Decision Tree, SVMs and Random Forests are 0.1, 0.1, 5 and 0.3 seconds respectively. k-NN and Decision Tree took the shortest time of 0.1 seconds and SVM took the longest time of 5 seconds. The table VII shows the averaged feature extraction and analysis time in seconds for each file

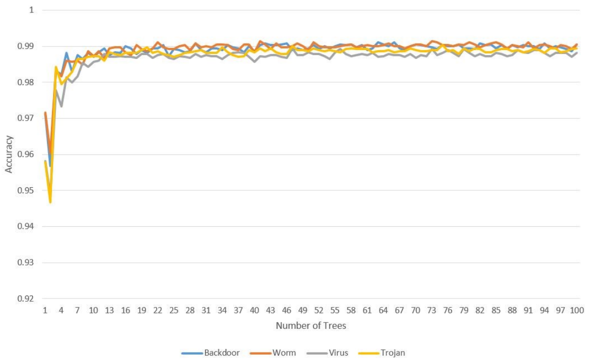

Fig. 4. The performance of Random Forest with Number of Trees

TABLE VII
EFFICIENCY OF ONLINE FILE ANALYSIS

Benign			
	Average Time	**# of Files**	**Total Volume**
	1.178 s	1157	1.6 GB
Malware			
	Average Time	**# of Files**	**Total Volume**
Worm	0.173 s	2641	490 MB
Trojan	0.224 s	2713	546 MB
Backdoor	0.251 s	2846	943 MB
Virus	0.188 s	2898	725 MB

type, total number of files and total volume of the files. It demonstrated that the proposed network malware detection process can detect malware in 0.21 seconds if the network traffic can be tapped and forwarded to the malware detection module. Also noted is the longer time on analyzing the benign files. The reason was some of the benign PE files are larger in size. It takes longer time to extract the features.

VI. CONCLUSION AND FUTURE WORK

This research explored four supervised techniques: Decision Tree, k-NN, SVMs and Random Forests for malware detection using the constructed 28 static features. Techniques were evaluated on four types of malware: backdoor, virus, trojan and worm. Random forests performed better than other three and achieved a weighted average detection rate of 98.7% and false positive rate of 1.8%. The results were very competitive or even better compare to some research that used more sophisticated learning algorithms or much more features in the literature [3] [6] [5].

The proposed network malware detection process achieved an average file analysis speed of 0.5 seconds per file on a laptop machine. The process can detect the malware within 0.21 seconds. Network malware detection efficiency has never been investigated in the literature. In the future, we will evaluate the proposed malware detection system on a server with more computing resources and on different network traffic settings. We will also consider expanding this system to detect portable executable files for other OS platforms, such as Unix. Other future work includes: extensively evaluating the proposed system on big data sets and using multiple evaluation metrics, such as Receiver Operating Characteristic (ROC) curve and F1-measure and investigating the minimal number of features that include dynamic features for network malware detection.

ACKNOWLEDGMENT

This research is supported by and is conducted as part of the IUPUI Living Lab at: https://livlab.org/.

REFERENCES

[1] M. Z. Shafiq, S. M. Tabish, F. Mirza, and M. Farooq, PE-Miner: Mining Structural Information to Detect Malicious Executables in Realtime, Recent Advances in Intrusion Detection, Volume 5758 of the series Lecture Notes in Computer Science, pp 121-141, 2009

[2] Chia Tien Dan Lo, Ordonez Pablo, and Cepeda Carlos, Feature Selection and Improving Classification Performance for Malware Detection, IEEE International Conferences on Big Data and Cloud Computing (BDCloud), Social Computing and Networking (SocialCom), Sustainable Computing and Communications (SustainCom), pp 560 - 566, 2016

[3] A. Shalaginov, L. S. Grini, and K. Franke, Understanding Neuro-Fuzzy on a class of multinomial malware detection problems, IEEE International Joint Conference on Neural Networks (IJCNN), pp 684-691, 2016

[4] VirusTotal, https://www.virustotal.com/

[5] J. C. Liu, J. F. Song, Q. G. Miao, Y. Cao, and Y. N. Quan, An Ensemble Cost-Sensitive One-Class Learning Framework for Malware Detection. International Journal of Pattern Recognition and Artificial Intelligence, 29(5), 2015

[6] Yuxin Ding, Sheng Chen and Jun Xu, Application of Deep Belief Networks for Opcode Based Malware Detection, IEEE International Joint Conference on Neural Networks (IJCNN), pp 684-691, 2016

[7] Steven Roman, Win32 API Programming with Visual Basic, O'Reilly Media; 1 edition, ISBN-10: 1565926315, 1999

[8] D. K. R. Chhabra, Feature selection and clustering for malicious and benign software characterization, Master Thesis, Master of Science in Computer Science Information Assurance, University of New Orleans, 2014

[9] W. Yan, Z. Zhang, N. Ansari, Revealing Packed Malware, IEEE Security and Privacy, pp. 72-76, 2008

[10] R. Lyda and J. Hamrock, Using Entropy Analysis to Find Encrypted and Packed Malware. IEEE Security and Privacy Magazine, 5(2), pp 40 - 45, 2007

[11] M. Sikorski and A. Honig, Practical Malware Analysis: The Hands-On Guide to Dissecting Malicious Software, No Starch Press, 1st edition (March 3 2012), ISBN-10 1593272901

[12] Windows Functions in Malware Analysis - Part 1, http://resources.infosecinstitute.com/windows-functions-in-malware-analysis-cheat-sheet-part-1/

[13] Ivan Firdausi, Charles Lim, Alva Erwin and Anto Satriyo Nugroho, Analysis of Machine Learning Techniques Used In Behavior-based Malware Detection, The Second IEEE International Conference on Advances in Computing, Control, and Telecommunication Technologies, pp. 201-203, 2010

[14] Y. Yang and X. Liu, A Re-examination of Text Categorization Methods, Proceedings of the ACM SIGIR, pp 42-49, 1999

[15] Richard R. Yang, V. Kang, S. Albouq and M. A. Zohdy, Application of Hybrid Machine Learning to Detect and Remove Malware, Transactions on Machine Learning and Artificial Intelligence, Vol 3, No 4, 2015

[16] Python SciKit Library, http://scikit-learn.org/stable/modules/tree.html#tree-algorithms

[17] V. N. Vapnik, The Nature of Statistical Learning Theory, Springer, New York, 1995

[18] T. Joachims, Text categorization with support vector machines: learning with many relevant features, ECML, LNCS, vol. 1398, pp. 137142, 1998

[19] T. Hastie, R. Tibshirani and J. Friedman, The Elements of Statistical Learning (2nd ed.), Springer, ISBN0-387-95284-5

[20] VX Heaven Virus Collection, http://vxheaven.org/vl.php

[21] The Bro Network Security Monitor, https://www.bro.org/index.html