The Effect on Network Flows-based Features and Training Set Size on Malware Detection

Jarilyn M. Hernández Jiménez, Katerina Goseva-Popstojanova
Lane Department of Computer Science and Electrical Engineering, West Virginia University, Morgantown, WV 26506
jhernan7@mix.wvu.edu, katerina.goseva@mail.wvu.edu

***Abstract*—Although network flows have been used in areas such as network traffic analysis and botnet detection, not many works have used network flows-based features for malware detection. This paper is focused on malware detection based on using features extracted from the network traffic and system logs. We evaluated the performance of four supervised machine learning algorithms (i.e., J48, Random Forest, Naive Bayes, and PART) for malware detection and identified the best learner. Furthermore, we used feature selection based on information gain to identify the smallest number of features needed for classification. In addition, we experimented with training sets of different sizes. The main findings include: (1) Adding network flows-based features improved significantly the performance of malware detection. (2) J48 and PART were the best performing learners, with the highest F-score and G-score values. (3) Using J48, the top five features ranked by information gain attained the same performance as when using all 88 features. In the case of PART, the top fourteen features ranked by information gain led to the same performance as when all 88 features were used. None of the system logs-based features were included in these two models. (4) The classification performance when training on 75% of the data was comparable to training on 90% of the data. As little as 25% of the data can be used for training at an expense of somewhat higher, but not very significant performance degradation (i.e., less than 7% for F-score and 6% for G-score compared to when 90% of the data were used for training).**

***Index Terms*—malware detection, network traffic-based features, network flows, system logs-based features, supervised machine learning.**

I. Introduction

Polymorphic malware can bypass signature-based detection methods by slightly changing the instructions of the existing malware. These new malware variants appear to be different programs from the viewpoint of signature-based anti-virus scanners, and cannot be identified until their signatures are incorporated into the detection software [1].

Previous works have attempted to address this problem by using other methods that are more powerful than signature-based detection, such as byte frequency [2], byte randomness [3], and behavioral analysis [4]–[6]. The byte frequency of software refers to the frequency of the different unsigned bytes in the corresponding file, while byte randomness refers to the bytes distribution value of the instruction sequences that are obtained from randomness tests. The behavioral analysis is based on the identification of the actions performed by the malware examples rather than on their binary code patterns. The behavior of a software application, including a malicious application, can be characterized by its system and network activities. Extracting behavioral features is more expensive in comparison to code-based features, since the behavioral analysis of the binary is required. However, the behavioral properties are more resilient against evasion attacks in comparison to code-based characteristics.

978-1-5386-7659-2/18/$31.00

Network flows have been used in areas such as network traffic analysis [7]–[20]. However, most of these works were focused on network traffic classification for botnet detection [7]–[11], [19], [20], malware detection on Android devices [12], [13], worms detection [14], [15], and for detection of other types of malware (i.e., trojans and viruses) [16], [17]. Additionally, using both network traffic and system logs data was explored in [15], [18]. It should be emphasized that these works have done classification of the network traffic to malicious and non-malicious [7]–[20], while we are classifying software binaries as malware and non-malicious software. Specifically, we use the behavioral characteristics, both network traffic-based and system logs-based, for malware detection.

For our experiments, we selected examples of recent malware, including viruses, worms, trojans, backdoors, rootkits, and ransomware. In the case of the non-malicious software, we used some applications that are network intensive and other that are CPU and memory usage intensive. We used our own testbed to collect both the network traffic data and system logs while the malware and non-malicious software ran on the experimental machine. Both the network traffic data and system logs were pre-processed and seventy eight network traffic related features (i.e., 18 commonly used network traffic-based features and 60 network flows-based features) and ten system logs-based features were extracted.

Our machine learning experiments for malware detection were based on using four supervised machine learning algorithms (i.e., J48, Random Forest, Naive Bayes, and PART) and comparing their performance. We used ten-fold cross validation for the these experiments. In addition, we used feature selection based on information gain to identify the smallest number of features needed for successful classification. We also experimented with training sets of different sizes.

This work explores the following research questions:

RQ1: Does the network flows-based features improve the performance of malware detection? What is (are) the best performing learner(s)?

RQ2: What is the smallest number of features sufficient to successfully distinguish malware from non-malicious software? What are the types of the best predictor features?

RQ3: How much data must be set aside for training in order to attain acceptable detection results?

The contributions of this paper are as follows:

- Most of the previous works that have used network flows-based features [7]–[20] have done classification of the network traffic, while our study is focused on classifying the software running in a machine as malware and non-malicious software using the extracted dynamic behavioral features (i.e., network traffic-based features and system logs-based features).
- Feature selection methods were not commonly used by previous works on malware detection, with an exception of [16]. Determining a small subset of features that can provide predictions as good as when all features are used has a practical usefulness and importance because it allows building more efficient models.
- We experimented with different sizes of the training set (i.e., 90%, 75%, 50%, and 25% of the data) and found that smaller training sets produced very good classification results. Specifically, using 75% of the data for training has only slightly worse performance compared to when using 90% of the data for training (which is the standard ten-fold cross validation approach). Somewhat worse, but still very good classification performance was achieved with as little as 25% of the data used for training. This aspect of our work has a practical value because the manual labeling of the training set is a tedious and time consuming process.

The main findings of our work include:

- Adding network flows-based features provided better performance than using the baseline feature vector consisting of a combination of commonly used network traffic-based features and system logs-based features.
- J48 and PART were the best performing learners, with the highest F-score and G-score values.
- Relatively small number of features was sufficient to distinguish malware from non-malicious software. Specifically, for J48 the top five features ranked by information gain provided same performance as using all 88 features, while for PART the top fourteen features ranked by information gain led to same performance as using all 88 features. None of the system logs-based features were included in these two models.
- The performance of training with 75% of the data was comparable to using 90% of the data for training. As little as 25% of the data can be used for training if one is willing to sacrifice some of the performance (i.e., less than 7% for F-score and 6% for G-score compared to when 90% of the data were used for training).

The rest of this paper is organized as follows. Related work is summarized in Section II and our experimental set-up and data collection process are described in Section III. The feature extraction process is presented in Section IV. Our machine leaning approach for malware detection is presented in Section V. Results are discussed in Section VI. Finally, the conclusion is presented in Section VII.

II. Related Work

Various works have explored malware detection using machine learning techniques [21], [22]. Here, we present a literature review of those approaches that used network traffic classification [7]–[20] and system logs [23]–[27] for malware detection.

Network flows-based features have been mainly used for detection of botnets [7]–[11], [19], [20]. In the case of malware detection, network flows-based features have not been extensively explored, except for a few works that focused on network traffic classification for malware detection in Android devices [12], [13], detection of worms [14], [15], and detection of other types of malware such as trojans and viruses [16], [17]. From these approaches the methods used by Dubendorfer et al. [14], Dressler et al. [15], Bekerman et al. [16], and Yeo et al. [17] are the most relevant to our work as they used similar network flows-based features and malware types that are used in our work.

Dubendorfer et al. [14] proposed an approach that used network flows from high speed Internet backbones demonstrating worms can be detected by tracking the cardinality of sudden changes in the network traffic. Dressler et al. [15] developed a pattern based on the correlation of flow-based features with system logs data for worms detection. Bekerman et al. [16] presented a malware detection approach that classified malicious and non-malicious network traffic recorded in sandbox environments and in real networks, while Yeo et al. [17] used a convolution neural network to detect network flows generated by botnets, trojans, and viruses.

System logs monitoring has also been used for malware detection [23]–[27]. From these works, the approaches proposed by Arpan et al. [26] and Ozsoy et al. [27] are the most relevant to our work as they used similar system changes triggered by malware as those used in this paper. Arpan et al. [26] compared the lookup table from the experiment and raw log files based on common attributes like event ID, event source, and event description. They focused on changes caused by the Service Control Manager (SCM) and in those system services that were stopped and/or installed. Similarly, Ozsoy et al. [27] proposed a malware detection approach that used features based on machine code, memory address patterns, and architectural events such as the frequency of memory reads and/or writes.

Combining system logs with network traffic data has been explored by Masud et al. [18] for detection of botnets and by Dressler et al. [15] for detection of worms. In both works, network flows-based features were extracted by aggregating the packet level features.

There are several main differences that distinguish our work from previous approaches:

- Several methods [14]–[17] used network flows-based features for detection of worms, trojans, and viruses. For the work in this paper, we used recent malware samples from 2011-2017. Most of them, particularly ransomware samples, communicated with a command and control server.
- For the system logs features, previous approaches focused on which services were triggered by malware [26] and on the analysis of features extracted from machine code and/or memory [27]. In this paper, our system logs-based features are focused on changes that occur in the file system, the registry, and the system processes.
- Feature selection methods were not commonly used by previous works, with an exception of [16]. Our results showed that using a small subset of features is sufficient to provide as good predictions as if all features were used, thus leading to more efficient prediction models. Furthermore, network traffic-based features were better predictors than the system logs-based features.
- It appears that none of related works have experimented with different sizes of training sets. Our results showed that using 75% of the data for training had comparable performance to using 90% of the data for training (which is the standard ten-fold cross validation approach). Using as little as 25% of the data for training led to somewhat worse, but still very good classification performance with less than 7% degradation of the F-score and 6% degradation of the G-score compared to when 90% of the data were used for training.

III. Experimental Set-up and Data Collection

The objective of our experiments was to collect the network traffic data and system logs while running different malware and non-malicious applications on a general-purpose computer. The experiments were conducted on a Dell OptiPlex 755 computer with a clean installation of 32-bit Windows 7 Ultimate. The experimental machine had unfiltered Internet access, which is crucial for the malware samples to perform their full functionality, as most malware initiates network traffic (e.g., contacts the command and control servers). We designed a segregated network, which consisted of the experimental machine, the data collection repository machine, a switch, and a cellular data connection, as shown in Figure 1. The data collection repository was connected to the personal hotspot, and then through a network switch the wireless connection was shared with the experimental machine. The advantages from the use of a segregated network are: (1) allowing the malware to behave normally, while avoiding the possibility of infecting other machines on the network, and (2) allowing us to monitor and record the experimental machine's network traffic.

Several software tools were used for the experimental design. We used ClockSynchro [28] to synchronize the clocks

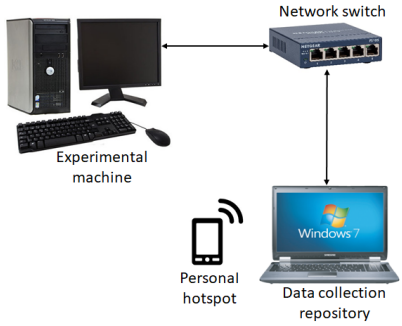

Figure 1: Experimental set-up

of the experimental machine and the data collection repository machine. We also used Wireshark [29], which ran on the data repository machine to collect the network traffic data. We used CaptureBAT [30] to collect the system logs (i.e., changes in the Windows operating system when either malware or non-malicious software was running on the experimental machine). Finally, we used the Clonezilla application [31] to ensure that the hard drive of the experimental machine contained a clean (i.e., uninfected) copy of the Windows operating system. Note that running malware and non-malicious software separately is a standard experimental set-up used by many malware detection approaches (e.g., [21], [22], [32]).

Each malware used in our experiments was executed in a virtual machine (on the experimental machine) to ensure that it was not corrupted and that it was functional for the Windows operating system. Also, we used the behavioral reports generated by a malware analysis service [33] to ensure the malware chosen for our experiments was entirely removed from the hard disk drive after formatting the machine using the Clonezilla tool. Malware was executed manually to infect the experimental machine. We used great care to allow malware to behave as intended. For each malware example, we ensured that it was active by monitoring the network traffic and by observing events such as files being encrypted, pop-ups with adult content, etc. The malware selected for our experiments have traits of viruses (e.g., Dexter [34]), worms (e.g., Gamarue [35]), trojans (e.g., Tofsee [36]), backdoors (e.g., Greencat [37]), rootkits (e.g., Alureon [38]), and ransomware (e.g., Locky [39]). Note that we used different types of malware to have a representative, diverse malware sample. Distinguishing among different malware types and/or malware families are beyond the scope of this paper.

In the case of the non-malicious software, we used applications such as Chrome [40], VLC [41], Windows Media Player [42] and benchmark tools such as IntelBurnTest [43], HeavyLoad [44], and XtremeMark [45]. Note that we used different types of non-malicious applications (some network intensive, other CPU and memory usage intensive). Furthermore, we ensured that each non-malicious application was provided with adequate inputs/workloads. For example, we used Chrome to navigate the Internet which generated network traffic, while IntelBurnTest was used to stress the CPU of the experimental machine.

Considering the randomness of different Windows operating system background processes, each malware and non-malicious software application was executed three times. Each ran lasted for thirty minutes. For each run, one .pcap and one system log file were collected. Note that other works that used behavioral characteristics for malware detection have executed the malware for five minutes in a controlled environment [46].

IV. Feature Extraction

After the data collection, both the system logs and the network traffic data were pre-processed and subsequently features were extracted.

A. System Logs

System logs (also known as syslogs) contain events that are logged by components of the operating system (OS). Syslogs are useful because they contain information about the software, hardware, system processes, and system components, as well as information on errors and warning events related to the OS.

We used CaptureBAT [30] to collect the system logs from the Windows OS. CaptureBAT monitors the file system, the registry, and the system processes. The file system monitor captures information such as the timestamp when the event occurred and the event type (e.g., read and/or write). The registry monitor focuses on the Windows Registry and captures events such as when a registry key is opened, modified, and/or deleted. The process monitor observes the creation and/or termination of processes. Note that the events recorded during our experiments are from the application-level (i.e., Ring 3). This means that we recorded all the system changes related to the code that ran outside the operating system's kernel [47]. We selected CaptureBAT because it has an exclusion list mechanism that allows to omit noise that occurs naturally in the system. Moreover, this tool has been recommended for conducting dynamic malware analysis [48]. CaptureBAT generated a text file for each run, which was then converted to a CSV file, pre-processed, and subsequently used to extract the system logs-based features listed in Table I.

TABLE I: System logs-based features extracted for each thirty minutes run of malware and non-malicious software

Feature name	***Description***
Changes	# of changes that occurred in the system
FileChgs	# of changes in the file system
RegistryChgs	# of changes in the registry
ProcessesChgs	# of changes in the process manager
FlsWrite	# of written files
FlsDelete	# of deleted files
CreatedPrcs	# of created processes
TerminatedPrcs	# of terminated processes
SetValueKeyChgs	# of times a method replaced or created a value entry under the open key
DelValueKeyChgs	# of times a method deleted a value entry under the open key

B. Network Traffic

The .pcap file of each run (collected using Wireshark [29]) was first exported as a CSV file. Subsequently, we extracted the commonly used network traffic-based features listed in Table II. Another important aspect of the data pre-processing concerns the flow decomposition of traces. Network flows can be defined as a summary of a connection between two hosts. Specifically, a network flow is defined using the following 5-tuple:

(ip_src, ip_dest, port_src, port_dst, and proto),

where *ip_src* refers to the source IP address, *ip_dest* refers to the destination IP address, *port_src* refers to the source port number, *port_dst* refers to the destination port number, and *proto* refers to the used protocol.

There are many tools that can be used to extract network flows from a .pcap file. For our work, we used Tranalyzer [49], a lightweight unidirectional flow exporter that collects packet information with common characteristics such as IP addresses and port numbers. This tool was chosen over the others for three reasons: (i) It is an extension of Netflow [50] which has been widely used by the research community as a flow exporter (aggregates packets into flows and export flow records) and as a flow collector (storage and pre-processing of flow data); (ii) It supports features that can be categorized into time, inter-arrival, packets and bytes, and flags groups; (iii) It has been used by previous works [19], [20] for detection of botnets.

Since multiple flows (typically over 100) were generated for each individual .pcap file, we performed aggregation of the network flows-based features to achieve our final network flows-based features. The list of the network flows-based features and the corresponding aggregation levels are given in Table III.

TABLE II: Commonly used network traffic-based features extracted for each thirty minutes run of malware and non-malicious software

Feature name	***Description***
Packets	# of received packets
BytesTransferred	Packets length in bytes
UniqueSourceIP	IP address of the device sending the packet
UniqueDestIP	IP address of the device receiving the packet
LLMNR	# of packets related to LLMNR
UDP	# of packets related to UDP
ARP	# of packets related to ARP
BROWSER	# of packets related to BROWSER
NBNS	# of packets related to NBNS
DHCP	# of packets related to DHCP
DHCPV6	# of packets related to DHCPV6
DNS	# of packets related to DNS
HTTP	# of packets related to HTTP
ICMP	# of packets related to ICMP
ICMPV6	# of packets related to ICMPV6
IGMPV3	# of packets related to IGMPV3
SSDP	# of packets related to SSDP
TCP	# of packets related to TCP

V. Machine Learning Approach

Common way of doing malware detection is based on using the feature vectors extracted from malware and non-malicious software as inputs to different classification techniques (such as machine learning and statistical analysis techniques). In this paper, we conducted a series of machine learning experiments for malware detection. The objective of our machine learning experiments was to classify between malware and non-malicious software.

TABLE III: Network flows-based features extracted for each thirty minutes run of malware and non-malicious software

Feature name	*Description*	*Aggregation levels*
Flows	# of flows	None
Duration	Time communication lasted	Max, Avg
L4ProtoUDP	# of flows related to UDP	None
L4ProtoIGMP	# of flows related to IGMP	None
L4ProtoTCP	# of flows related to TCP	None
L4ProtoICMP	# of flows related to ICMP	None
PktsSent	# of transmitted packets	Sum, Max, Avg
PktsRcvd	# of received packets	Sum, Max, Avg
BytesSnt	# of transmitted bytes	Sum, Min, Max, Avg
BytesRcvd	# of received bytes	Sum, Max, Avg
MinPktSize	Min layer 3 packet size	Min
MaxPktSize	Max layer 3 packet size	Max
AvgPktSize	Avg packet load ratio	Avg, Median
StdPktSz	Std packet load ratio	Std
Pktps	Packets sent per second	Max, Avg
Bytps	Bytes sent per second	Max, Avg
PktAsm	Packet stream asymmetry	Min, Avg
BytAsm	Byte stream asymmetry	Min, Avg
TcpPSeqCnt	TCP packet sequence count	Max, Avg
TcpSeqSntBytes	TCP sent sequence diff bytes	Max, Avg
TcpSeqFaultCnt	# of TCP sequence fault count	Max, Avg
TcpPAckCnt	TCP packet ack count	Max, Avg
TcpFlLAcRcByt	TCP flawless ack received bytes	Max, Avg
TcpAckFaultCnt	# of TCP ack fault count	Max, Avg
TcpInitWinSz	TCP initial window size	Max, Avg
TcpAveWinSz	TCP avg window size	Avg, Median
TcpWinSzDwCn	TCP window size change down count	Max, Avg
TcpWiSzUpCnt	TCP window size change up count	Max, Avg
TcpWiSzChDiCn	TCP window size direction change count	Max, Avg
FlowDirA	Flows direction is clnt to srvr	None
FlowDirB	Flows direction is srvr to clnt	None
AvgIAT	Average of IAT	Avg, Median
StdIAT	Standard deviation of IAT	Std

The baseline feature vector was created by combining the system logs-based features (given in Table I) and the commonly used network traffic-based features (given in Table II). Then, we added the network flows-based features (given in Table III) to the baseline feature vector to study their effect on the malware detection. Note that any feature (regardless of the type) which had all instances equal to 0 was removed from the learning process.

For classification, we used four supervised machine learning algorithms (i.e., J48, Random Forest, Naive Bayes, and PART) of different types, with a goal to identify the best performing learner(s). Table IV lists the names and types the four learners used in this work.

TABLE IV: Name and type of each learner used for this work

Learner	Type
J48 [51]	Tree
Random Forest (RF) [52]	Ensemble Tree
Naive Bayes (NB) [53]	Bayes Theorem
PART [54]	Rule + Tree

The J48 learner is an open source Java implementation of the C4.5 decision tree algorithm developed by Ross Quinlan [51]. Random Forest is an ensemble learning method that operates by constructing a multitude of decision trees and outputs the average prediction of the individual trees [52]. Naive Bayes is an algorithm based on the Bayesian theorem in which numeric estimator precision values are chosen based on analysis of the training data [53]. PART is a hybrid rule-and-tree algorithm that builds a partial C4.5 decision tree in each iteration and makes the "best" leaf into a rule [54]. We used the implementations of these four learners provided in Weka [55].

For malware detection experiments, we used ten-fold cross validation, which consists of using nine folds of the labeled malware and non-malicious software instances for training (i.e., 90% of the data) and the tenth fold (of unseen) malware and non-malicious instances for testing. We also experimented with smaller training set sizes (i.e., 75%, 50%, and 25% of the data).

To evaluate the learners performance, we used several metrics computed from the confusion matrix:

	Actual: *Non-malicious*	**Actual:** *Malware*
Predicted: *Non-malicious*	TN	FN
Predicted: *Malware*	FP	TP

where TN, FN, FP, and TP refer to the numbers of true negatives, false negatives, false positives, and true positives, respectively. We compute the following performance metrics that assess different aspects of the classification:

$$\text{Accuracy} = (TP + TN)/(TP + TN + FP + FN) \quad (1)$$

$$\text{Recall} = TP/(TP + FN) \quad (2)$$

$$\text{Precision} = TP/(TP + FP) \quad (3)$$

$$\text{FPR} = FP/(FP + TN) \quad (4)$$

$$\text{F-score} = \frac{2 \cdot Precision \cdot Recall}{Precision + Recall} \quad (5)$$

$$\text{G-score} = \frac{2 \cdot Recall \cdot (1 - FPR)}{Recall + (1 - FPR)} \quad (6)$$

The accuracy (see Equation (1)) provides the percentage of instances that were detected correctly. The Recall, defined by Equation (2), is the ratio of detected malware to all malware instances. Precision (see Equation (3)) determines the fraction of instances correctly classified as malware out of all instances classified as malware. False Positive Rate (FPR), defined by Equation (4), is the ratio of non-malicious software applications misclassified as malware to the number of all non-malicious applications. Values of all metrics are in the interval $[0, 1]$. Ideally, a good classifier would have Accuracy, Recall, and Precision of 1 and FPR of 0.

In addition to these metrics, we used two composite metrics: F-score and G-score. The F-score, defined by Equation (5), is the harmonic mean of the Recall and Precision. Similarly, G-score, given by Equation (6), is the harmonic mean of Recall and $(1 - FPR)$. Larger values of F-score and G-score correspond to better learner performance. An ideal learner would have both F-score and G-score of 1.

VI. Results

In this section we present the main findings as they pertain to the research questions given in Section I.

A. RQ1: Does the network flows-based features improve the performance of malware detection? What is (are) the best performing learner(s)?

To answer this research question, we compared the learners performance when the baseline features were used (see Figure 2) and when the network flows-based features were added to the baseline feature vector (see Figure 3). Because low FPR indicates better performance, $1 - FPR$ is shown in these Figures. The range of performance metrics for both Figures 2 and 3 is from 0.5 to 1.

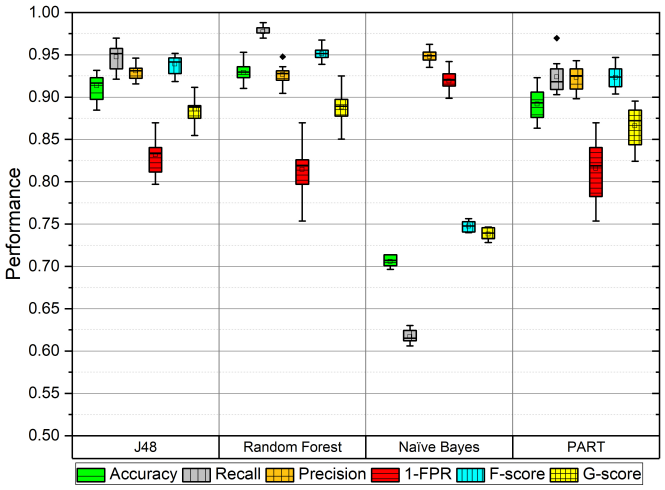

Figure 2: Box plots of the learners' performance metrics for the baseline feature vector

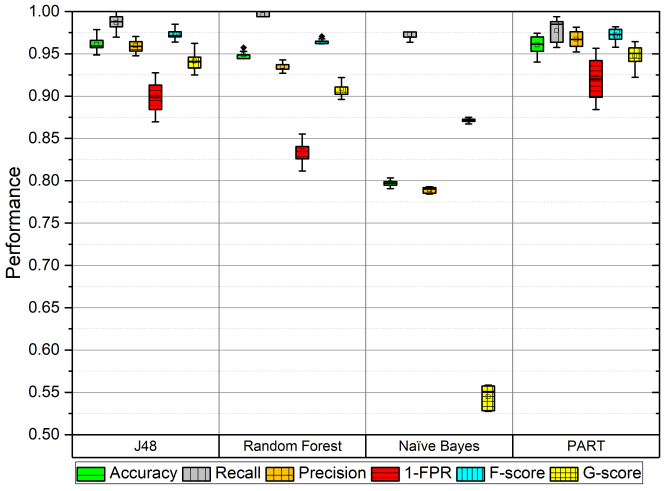

Figure 3: Box plots of the learners' performance metrics for all features (i.e., baseline feature vector + network flows-based features). Note that since $1-FPR$ for Naive Bayes was below 0.5, it is not shown in this figure.

In addition to box plots shown in Figures 2 and 3, we used the basic statistics (i.e, mean, median, variance, and interquartile range (IQR)) for the G-score (given in Table V) and F-score (shown in Table VI). Note that IQR is a measure of statistical dispersion, being equal to the difference between 75^{th} and 25^{th} percentiles.

In case of G-score, when network flows-based features were added to the baseline feature vector, J48, Random Forest, and PART showed a significant improvement of the mean and median G-score, as well as smaller variance and IQR. On the other side, the Naive Bayes algorithm experienced degradation of the G-score when all features were used. This was due to the increased FPR, which likely was a result of the fact that this learner assumes that features are conditionally independent from one another.

In the case of F-score, the performance of all learners was improved when using all features compared to when the baseline feature vector was used, that is, they had significantly higher mean and median F-scores and smaller variance and IQR. Note that the F-score of the Naive Bayes algorithm had significantly smaller mean and median values than the other three algorithms.

In summary, when all features were used for classification, J48 and PART were the best performing learners. PART had sightly higher median G-score than J48 (0.950 compared to 0.941), while they had similar median F-score values (0.973 and 0.972, respectively).

TABLE V: Basics Statistics of G-score

	Baseline				**All Features**			
Learners	Mean	Median	Variance	IQR	Mean	Median	Variance	IQR
J48	0.886	0.888	$3.12 \cdot 10^{-4}$	0.014	0.941	0.941	$1.21 \cdot 10^{-4}$	0.013
RF	0.888	0.889	$3.94 \cdot 10^{-4}$	0.019	0.908	0.911	$5.25 \cdot 10^{-5}$	0.008
NB	0.738	0.740	$5.23 \cdot 10^{-5}$	0.019	0.545	0.550	$2.17 \cdot 10^{-4}$	0.029
PART	0.866	0.872	$5.70 \cdot 10^{-4}$	0.037	0.947	0.950	$1.83 \cdot 10^{-4}$	0.016

TABLE VI: Basic Statistics of F-score

	Baseline				**All Features**			
Learners	Mean	Median	Variance	IQR	Mean	Median	Variance	IQR
J48	0.939	0.942	$1.25 \cdot 10^{-4}$	0.015	0.973	0.972	$3.69 \cdot 10^{-5}$	0.005
RF	0.951	0.951	$6.10 \cdot 10^{-5}$	0.008	0.965	0.965	$7.40 \cdot 10^{-6}$	0.002
NB	0.747	0.748	$4.11 \cdot 10^{-5}$	0.011	0.871	0.871	$6.21 \cdot 10^{-6}$	0.002
PART	0.923	0.924	$1.97 \cdot 10^{-4}$	0.018	0.972	0.973	$5.78 \cdot 10^{-5}$	0.011

B. RQ2: What is the smallest number of features sufficient to successfully distinguish malware from non-malicious software? What are the types of the best predictor features?

To answer this research question we used feature selection method on all features (i.e., the combined set of baseline features and network flows-based features). Specifically, we used a feature selection method called information gain [56], which ranks the features from the most descriptive to the least descriptive using the information gain as a measure. Table VII shows all features by their ranking order.

To determine the smallest number of features that can be used for malware detection without performance degradation we used the following approach. We started building the model with the highest ranked feature and included one feature at a time until reaching less than or equal to 1% difference of the Recall compared to when all 88 features were used.

For J48, the top five features ranked by information gain provided similar performance as when using all 88 features.

Four out of the five features were network flows-based features. In the case of PART, the first fourteen features ranked by information gain led to similar performance as when using all 88 features. Six out of the fourteen features were network flows-based features. In general, network traffic-based features were better predictors than the system logs-based features, which were not included in any of these two models. (The top most system logs-based feature was ranked the forty first, as shown in Table VII.)

TABLE VII: All features ranked using information gain. Features in gray are network flows-based features, bold features are system logs-based features and the remaining features are the commonly used network traffic-based features.

Rank	*Feature*	*Rank*	*Feature*
1	BytesSntMax	45	TcpWinSzDwnCntMax
2	PktsSentSum	46	TcpPAckCntMax
3	PktsSentMax	47	FlowDirB
4	Packets	48	TcpFlLAcRcBytMax
5	BytesSntSum	49	TcpAveWinSzAvg
6	L4ProtoIGMP	50	MinPktSizeMin
7	SSDP	51	DNS
8	NBNS	52	PktpsMax
9	ARP	53	BytAsmAvg
10	UDP	54	TcpWinSzUpCntMax
11	BytesSntAvg	55	**FileChgs**
12	IGMPV3	56	**FlsWrite**
13	ICMPV6	57	TcpSeqFaultCntMax
14	TCP	58	PktpsAvg
15	PktsRcvdSum	59	L4ProtoUDP
16	DurationAvg	60	TcpPSeqCntAvg
17	BytesTransferred	61	TcpWnSzChgDiCnMax
18	AvgPktSizeAvg	62	TcpPAckCntAvg
19	TcpAckFaultCntMax	63	**Changes**
20	PktsSentAvg	64	L4ProtoICMP
21	LLMNR	65	AvgIATAvg
22	BytesRcvdSum	66	AvgIATMedian
23	ICMP	67	**TerminatedPrcs**
24	UniqueSourceIP	68	**ProcessesChgs**
25	TcpAckFaultCntAvg	69	**CreatedPrcs**
26	PktsRcvdMax	70	DHCPV6
27	FlowDirA	71	DHCP
28	BytesRcvdMax	72	StdIATStd
29	BROWSER	73	**FlsDelete**
30	TcpInitWinSzAvg	74	HTTP
31	L4ProtoTCP	75	MaxPktSizeMax
32	Flows	76	TcpWnSzChDiCnAvg
33	AvgPktSizeMedian	77	TcpSeqSntBytesMax
34	TcpInitWinSzMax	78	PktAsmAvg
35	BytpsAvg	79	PktAsmMin
36	BytesRcvdAvg	80	BytpsMax
37	BytesSntMin	81	BytAsmMin
38	PktsRcvdAvg	82	TcpSeqSntBytesAvg
39	StdPktSzStd	83	TcpWinSzUpCntAvg
40	TcpFlLAcRcBytAvg	84	TcpSeqFaultCntAvg
41	**RegistryChgs**	85	TcpWnSzDwCnAvg
42	**SetValueKeyChgs**	86	DurationMax
43	UniqueDestIP	87	TcpAveWinSzMedian
44	TcpPSeqCntMax	88	**DelValueKeyChgs**

C. RQ3: How much data must be set aside for training in order to attain acceptable detection results?

Next, we focus on determining the amount of data that must be set aside for training in order to produce good malware detection results. For this part of our study, we restricted the experiments to the best performing learners J48 and PART, using all features. Table VIII shows the performance of J48 and PART using training sets with different sizes (i.e., 90%, 75%, 50%, and 25% of the data).

The results showed that the learners were able to produce similar performance with 75% of the data used for training as in the case when 90% of data were used for training, which is the commonly used 10-fold cross validation machine learning approach. The performance of the learners was more significantly affected when 50% of the data were used for training, with less than 3% degradation of the F-score and 5% degradation of the G-score compared to when 90% of the data were used for training. Even when only 25% of the data were used for training the malware detection performance was still satisfactory, with Accuracy, Precision, Recall, F-score, and G-score all around or above 90% and less than 7% degradation of the F-score and 6% degradation of the G-score compared to when 90% of the data were used for training. It should be noted that the FPR was significantly more affected by the smaller sizes of the training set than any other performance metric.

It appears that the amount of data used for training is a trade-off between somewhat better results at an expense of significantly more effort invested in labeling more data. The fact that smaller training sizes led to successful malware detection is an important result of our study, with a significant practical value.

TABLE VIII: J48 and PART performance on training sets with different sizes

Learner	*Performance Metrics*	*% of data used for training*			
		90%	*75%*	*50%*	*25%*
J48	*Accuracy*	96.11%	94.74%	92.56%	92.13%
	Precision	95.94%	94.91%	93.82%	91.25%
	Recall	98.67%	97.82%	95.88%	90.15%
	FPR	10.00%	12.61%	15.36%	10.23%
	F-score	97.28%	96.32%	94.80%	90.63%
	G-score	94.13%	92.26%	89.69%	89.43%
PART	*Accuracy*	96.03%	94.15%	92.74%	91.90%
	Precision	96.72%	94.35%	93.98%	91.85%
	Recall	97.70%	97.58%	95.88%	89.55%
	FPR	8.00%	14.06%	14.78%	9.93%
	F-score	97.20%	95.92%	94.90%	90.60%
	G-score	94.75%	91.33%	90.15%	89.20%

VII. Conclusion

In this paper we presented a malware detection approach based on using dynamic features extracted from the network traffic data and system logs, which were collected while running malware samples and non-malicious software applications on our experimental testbed. For the machine learning experiments we used two feature vectors: a baseline feature vector consisting of the system logs-based features and the commonly used network traffic-based features and another feature vector that integrated the baseline feature vector with the network flows-based features (that is, consisted of all features). Four supervised machine learning algorithms were used for classification, using both the baseline feature vector and the

feature vector with all features. The main findings include: (1) Adding network flows-based features improved significantly the performance of malware detection. (2) J48 and PART were the best performing learners, with the highest F-score and G-score values. (3) Using J48, the first five features ranked by information gain attained the same performance as when using the all 88 features, while in the case of PART the first fourteen features ranked by information led to same performance as when the all 88 features were used. None of the system logs-based features were included in these two models. (4) The classification performance when training on 75% of the data was comparable to training on 90% of the data. Using as little as 25% of the data for training led to somewhat worse, but still very good classification performance, with Accuracy, Precision, Recall, F-score, and G-score all around or above 90% and less than 7% degradation of the F-score and and 6% degradation of the G-score compared to when 90% of the data were used for training.

As part of our future work, we plan to extract additional features from both the network traffic data and system logs, as well as to explore if the results presented in this paper would generalize in a more diverse execution environment with multiple machines.

ACKNOWLEDGMENT

This work is funded by the National Science Foundation under the grant CNS-1618629. The authors thank Dr. Jeffrey A. Nichols and Dr. Stacy Prowell from the Cyber and Information Security Research group at Oak Ridge National Laboratory (ORNL) for their support with the initial hardware and software configuration of the experimental testbed.

REFERENCES

[1] P. O'Kane, S. Sezer, and K. McLaughlin, "Obfuscation: The hidden malware," *IEEE Security & Privacy*, vol. 9, no. 5, pp. 41–47, 2011.

[2] S. Yu, S. Zhou, and R. Yang, "Detecting malware variants by byte frequency," *Journal of Networks*, vol. 6, no. 4, pp. 638–645, Apr. 2011.

[3] S. Qi, M. Xu, and N. Zheng, "A malware variant detection method based on byte randomness test," *Journal of Computers*, vol. 8, no. 10, pp. 2469–2477, 2013.

[4] J. M. Hernández, A. Ferber, S. Prowell, and L. Hively, "Phase-space detection of cyber events," in *10th ACM Cyber and Information Security Research Conference (CISRC)*, 2015, p. 13.

[5] M. Ahmadi, A. Sami, H. Rahimi, and B. Yadegari, "Malware detection by behavioural sequential patterns," *Computer Fraud & Security*, vol. 2013, no. 8, pp. 11–19, 2013.

[6] A. M. Fawaz and W. H. Sanders, "Learning process behavioral baselines for anomaly detection," in *IEEE 22nd Pacific Rim International Symposium on Dependable Computing (PRDC)*, 2017, pp. 145–154.

[7] R. F. M. Dollah, M. Faizal, F. Arif, M. Z. Mas'ud, and L. K. Xin, "Machine learning for HTTP botnet detection using classifier algorithms," *Journal of Telecommunication, Electronic and Computer Engineering*, vol. 10, no. 1-7, pp. 27–30, 2018.

[8] S. Ding, "Machine learning for cybersecurity: Network-based botnet detection using time-limited flows," 2017, unpublished.

[9] Z. B. Celik, J. Raghuram, G. Kesidis, and D. J. Miller, "Salting public traces with attack traffic to test flow classifiers," in *4th USENIX Workshop on Cyber Security Experimentation and Test (CSET)*, 2011, pp. 3–3.

[10] D. Acarali, M. Rajarajan, N. Komninos, and I. Herwono, "Event graphs for the observation of botnet traffic," in *8th IEEE Annual Information Technology, Electronics and Mobile Communication Conference (IEMCON)*, 2017, pp. 628–634.

[11] D. Zhao, I. Traore, A. Ghorbani, B. Sayed, S. Saad, and W. Lu, "Peer to peer botnet detection based on flow intervals," in *IFIP International Information Security Conference (SEC)*. Springer, 2012, pp. 87–102.

[12] A. Arora, S. Garg, and S. K. Peddoju, "Malware detection using network traffic analysis in Android based mobile devices," in *8th IEEE Conference on Next Generation Mobile Applications, Security and Technologies (NGMAST)*, 2014, pp. 66–71.

[13] S. Wang, Q. Yan, Z. Chen, B. Yang, C. Zhao, and M. Conti, "Detecting Android malware leveraging text semantics of network flows," *IEEE Transactions on Information Forensics and Security*, vol. 13, pp. 1096–1109, 2018.

[14] T. Diibendorfer and B. Plattner, "Host behaviour based early detection of worm outbreaks in Internet backbones," in *14th IEEE International Conference on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE)*, 2005, pp. 166–171.

[15] F. Dressler, W. Jaegers, and R. German, "Flow-based worm detection using correlated honeypot logs," in *ITG-GI Conference in Communication in Distributed Systems*, 2007, pp. 1–6.

[16] D. Bekerman, B. Shapira, L. Rokach, and A. Bar, "Unknown malware detection using network traffic classification," in *IEEE Conference on Communications and Network Security (CNS)*, 2015, pp. 134–142.

[17] M. Yeo, Y. Koo, Y. Yoon, T. Hwang, J. Ryu, J. Song, and C. Park, "Flow-based malware detection using convolutional neural network," in *IEEE International Conference on Information Networking (ICOIN)*, 2018, pp. 910–913.

[18] M. M. Masud, T. Al-Khateeb, L. Khan, B. Thuraisingham, and K. W. Hamlen, "Flow-based identification of botnet traffic by mining multiple log files," in *IEEE International Distributed Framework and Applications Conference*, 2008, pp. 200–206.

[19] F. Haddadi, D. Le Cong, L. Porter, and A. N. Zincir-Heywood, "On the effectiveness of different botnet detection approaches," in *Information Security Practice and Experience (ISPEC)*. Springer, 2015, pp. 121–135.

[20] F. Haddadi and A. N. Zincir-Heywood, "Benchmarking the effect of flow exporters and protocol filters on botnet traffic classification," *IEEE Systems Journal*, vol. 10, no. 4, pp. 1390–1401, 2016.

[21] N. Idika and A. P. Mathur, "A survey of malware detection techniques," *Purdue University*, vol. 48, 2007.

[22] A. Souri and R. Hosseini, "A state-of-the-art survey of malware detection approaches using data mining techniques," *Human-centric Computing and Information Sciences*, vol. 8, no. 1, p. 3, 2018.

[23] E. N. Yolacan, J. G. Dy, and D. R. Kaeli, "System call anomaly detection using multi-HMMs," in *8th IEEE International Conference on Software Quality, Reliability and Security - Companion (QRS-C)*, 2014, pp. 25–30.

[24] R. S. Pirscoveanu, S. S. Hansen, T. M. Larsen, M. Stevanovic, J. M. Pedersen, and A. Czech, "Analysis of malware behavior: Type classification using machine learning," in *IEEE Cyber Situational Awareness, Data Analytics and Assessment (CyberSA)*, 2015, pp. 1–7.

[25] R. Canzanese, S. Mancoridis, and M. Kam, "System call-based detection of malicious processes," in *IEEE International Conference on Software Quality, Reliability and Security*, 2015, pp. 119–124.

[26] A. M. Sainju and T. Atkison, "An experimental analysis of Windows log events triggered by malware," in *ACM SouthEast Conference*, 2017, pp. 195–198.

[27] M. Ozsoy, C. Donovick, I. Gorelik, N. Abu-Ghazaleh, and D. Ponomarev, "Malware-aware processors: A framework for efficient online malware detection," in *21st IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, 2015, pp. 651–661.

[28] (2017) Clocksynchro. [Online]. Available: http://clocksynchro.com/

[29] (2016) Wireshark. [Online]. Available: https://www.wireshark.org/

[30] C. Seifert *et al.*, "Capture–a behavioral analysis tool for applications and documents," *Digital Investigation*, vol. 4, pp. 23–30, 2007.

[31] (2016) Clonezilla. [Online]. Available: http://clonezilla.org/

[32] R. Tian, R. Islam, L. Batten, and S. Versteeg, "Differentiating malware from cleanware using behavioural analysis," in *5th IEEE International Conference on Malicious and Unwanted Software (MALWARE)*, 2010, pp. 23–30.

[33] (2016) Payload Security. [Online]. Available: https://www.hybrid-analysis.com/

[34] L. Constantin. (2016) Dexter malware infects point-of-sale systems worldwide, researchers say. [Online]. Available: https://goo.gl/b3Tvjy

[35] (2016) Win32/gamarue. [Online]. Available: https://malwarefixes.com/threats/win32gamarue/

[36] E. Kovacs. (2016) Tofsee malware distribution switched from exploit kit to spam. [Online]. Available: https://goo.gl/zonRmP

[37] (2016) What is GREENCAT-2? [Online]. Available: https://www.solvusoft.com/en/malware/trojans/greencat-2/

[38] E. Wrenn. (2012) Warning from FBI: If you have 'Alureon' virus on your PC, you WILL get kicked off Internet on Monday. [Online]. Available: https://goo.gl/JCDc3K

[39] (2016) Locky. [Online]. Available: https://nakedsecurity.sophos.com/2016/02/17/locky-ransomwa-re-what-you-need-to-know/

[40] (2016) Welcome to Chromium. [Online]. Available: https://blog.chromium.org/2008/09/welcome-to-chromium_02.html

[41] (2017) VLC Media Player. [Online]. Available: https://www.videolan.org/vlc/

[42] (2017) Windows Media Player. [Online]. Available: https://www.computerhope.com/jargon/w/windows-media-player.htm

[43] (2017) IntelBurnTest. [Online]. Available: https://www.majorgeeks.com/files/details/intelburntest.html

[44] (2017) HeavyLoad. [Online]. Available: https://www.jam-software.com/heavyload/

[45] (2017) XtremeMark. [Online]. Available: http://www.xtreme-lab.net/en/xmark.htm

[46] P. Burnap, R. French, F. Turner, and K. Jones, "Malware classification using self organising feature maps and machine activity data," *Computers & Security*, vol. 73, pp. 399–410, 2018.

[47] (2018) User space. [Online]. Available: https://en.wikipedia.org/wiki/User_space

[48] E. Gandotra, D. Bansal, and S. Sofat, "Malware analysis and classification: A survey," *Journal of Information Security*, vol. 5, no. 02, p. 56, 2014.

[49] (2018) Tranalyzer. [Online]. Available: https://tranalyzer.com/

[50] B. Claise, "Cisco systems Netflow services export Version 9," Tech. Rep., 2004.

[51] J. R. Quinlan, *C4.5: Programs for Machine Learning*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1993.

[52] L. Breiman, "Random forests," *Machine Learning*, vol. 45, no. 1, pp. 5–32, 2001.

[53] G. H. John and P. Langley, "Estimating continuous distributions in bayesian classifiers," in *11th Conference on Uncertainty in Artificial Intelligence (UAI)*, 1995, pp. 338–345.

[54] E. Frank and I. H. Witten, "Generating accurate rule sets without global optimization," in *15th International Conference on Machine Learning*, 1998, pp. 144–151.

[55] M. Hall *et al.*, "The weka data mining software: An update," *ACM SIGKDD Explorations Newsletter*, vol. 11, no. 1, pp. 10–18, 2009.

[56] J. T. Kent, "Information gain and a general measure of correlation," *Biometrika*, vol. 70, no. 1, pp. 163–173, 1983.