Toward an Automatic, Online Behavioral Malware Classification System

Raymond Canzanese and Moshe Kam
Dept. of Electrical and Computer Engineering
Drexel University
Philadelphia, PA, USA
{rcanzanese,kam}@minerva.ece.drexel.edu

Spiros Mancoridis
Dept. of Computer Science
Drexel University
Philadelphia, PA, USA
spiros@drexel.edu

***Abstract*—Malware authors are increasingly using specialized toolkits and obfuscation techniques to modify existing malware and avoid detection by traditional antivirus software. The resulting proliferation of obfuscated malware variants poses a challenge to antivirus vendors, who must create signatures to detect each new malware variant. Although the many variants in a malware family have different static signatures, they share characteristic behavioral patterns resulting from their common function and heritage. We describe an automatic classification system that can be trained to accurately identify new variants within known malware families, using observed similarities in behavioral features extracted from sensors monitoring live computers hosts. We evaluate the accuracy of the classifier on a live testbed under a heavy computational load. The described classification system is intended to perform classification online, using the computed classes of newly detected malware variants to guide the automatic mitigation of infected hosts.**

I. Introduction

Malicious software or malware are designed to perform various malevolent tasks, such as providing unauthorized access, stealing confidential information, or disrupting the normal operation of computer systems. The traditional defense against malware has been antivirus software, which use static signatures to detect known malware samples. Antivirus software represent a reactive approach to malware detection, where detection signatures for new malware are typically created after the malware have been successfully deployed. Malware authors are increasingly using specialized software to create large numbers of obfuscated malware variants to exploit this weakness and evade detection. In some cases, malware authors have released tens of thousands of variants of a malware sample in rapid succession in an attempt to overwhelm antivirus vendors [1]. Malware authors have also developed metamorphic and polymorphic malware, which automatically modify themselves as they propagate [2].

The continuing rise of obfuscated malware variants presents challenges both to antivirus vendors, who must analyze, classify, and create signatures for each newly discovered malware variant; and system administrators, who must find ways to defend their computer systems against the influx of malware variants that evade traditional detection mechanisms. Our research focuses on the latter problem, that of defending computer systems against previously unseen malware, especially malware that are variants of, or behaviorally similar to, known malware. Accordingly, our goal is to create self-protecting servers that can quickly detect the execution of new malware variants, classify them according to their behavior, and use the computed classes to guide the automatic mitigation of infected hosts.

Whereas our previous work has focused the problem of behavioral malware detection [3], [4], this study focuses on the problem of classification. That is, given that we have detected a new malware sample, can we quickly and accurately classify the malware in order to determine its category (*e.g.,* Trojan, rootkit, worm) or family (*e.g.,* ZBot, Fynloski, Kelihos)? We present the design and experimental evaluation of an online, host-based malware classification system, making the following contributions:

- *Feature Selection:* We select a set of observable features that are easily extracted with minimal overhead from live computer hosts, and whose values can be used to infer whether a detected malware sample belongs to a particular category or family. We evaluate three different types of features in terms of their usefulness for malware classification, namely data collected from performance monitors that report resource usage, the frequency of calls to specific kernel functions, and the frequency of calls to specific sequences of kernel functions.
- *Classification:* We present the design of a classification system that uses random forests to identify the category and family of detected malware. The classifier is designed to classify new malware variants according to their behavioral similarity to known malware.
- *Experimental Evaluation:* We evaluate the accuracy of the classification system on a corpus of more than 800 malware samples. We perform the evaluation on a custom testbed designed to mimic the challenges of online malware classification in a real-world, production environment.

The remainder of the paper describes the related work in behavioral malware classification in Section II, the experimental setup in Section III, the classification procedure in Section IV, and the experimental results and analyses in Section V.

II. Related Work

Previous studies of behavioral malware detection and classification have identified features that provide discrimination among different classes of malware and benign software, including sequences of calls to kernel functions [5], resource monitors [6], and features extracted from static executable files [7]. Furthermore, there have been efforts to address the problem of malware classification for reasons of expediting the analysis of (and signature creation for) newly discovered malware samples. The classification systems described in this section are mostly systems designed for such offline analysis rather than rapid, online classification, which is the focus our study.

Classification systems generally fall into one of two categories: those that rely on features extracted from static files, or those that execute malware and use behavioral features to classify malware. Static approaches sometimes use low-level features such as calls to external libraries, strings, and byte sequences for classification [8]. Other static approaches extract more detailed information from binaries, including sequences of API calls, the graphical representations of control flow [9]–[11], and the structure of the functions composing the malware [12]. Recent work represents the byte structure of malware samples as images and uses image classification techniques to classify the malware [13].

Reliable extraction of the features required for static classification represents a considerable challenge. Malware are often encrypted, compressed, or otherwise designed to complicate such analyses. Accordingly, many techniques have been explored for extracting features from malware while they execute. A subset of these techniques involve executing the malware in an isolated sandbox environment and extracting information about the files, registry entries, and processes that the malware create or modify [14]–[18]. Another body of work focuses solely on network-based features, extracting network flow information for classification [19], [20]. Still other techniques include using API hook information to classify rootkits [21]. Recent work has also been successful in combining both static and behavioral features to leverage the strengths of each type of classifier to increase overall classification accuracy [22], [23].

Our present work draws from previous work, using behavioral features previously shown to be useful for malware detection and classification as inputs to a random forest classifier. We extend previous studies, which focus on offline analysis, by focusing our efforts on the online classification of new, previously unseen malware variants that evade signature based detection on production hosts. To evaluate the usefulness of our classification system, we study the performance of the classifier using five different malware labeling schemes. We aim to classify new malware variants online as soon as they are detected to guide immediate, automatic mitigation. Our goal is to create self-protecting servers that can automatically detect and mitigate infections caused by previously unseen malware.

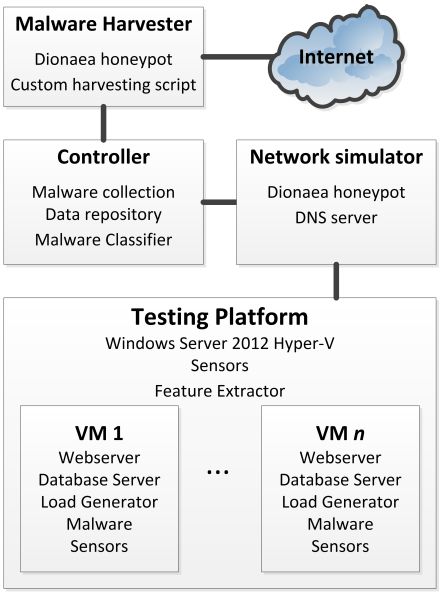

Fig. 1. Malware classification testbed

III. Experimental Setup

This section describes the experimental setup we use to evaluate the accuracy of the described classification system, including an overview of the malware testbed, details of the experimental procedures used for data collection, a description of the collected features, and a description of the malware corpus used for experimental evaluation.

A. Malware Classification Testbed

Our malware classification testbed is designed to provide an environment for testing the malware classification system that is analogous to the environment under which classification would occur in a production environment: namely, a noisy environment where the host computer is performing the heavy computation necessary to perform its intended function.

The virtual machine (VM) hosts composing the testbed operate as web and database servers performing a series of functional tests provided by the Drupal[1] Content Management System (CMS). The test suite provides 55 different categories of tests, each exercising a different component of the CMS (*e.g.,* file system access, database operations). Our testbed continuously executes a randomly selected series of tests. The result is a complex computational load that varies significantly in time and complicates classification by introducing significant noise into the feature data used for classification. During testing, each VM starts in a clean state and is infected with malware at a randomly distributed infection time (5 to 15 minutes after boot), to ensure that the malware infection is not correlated with any specific system events.

A block diagram of the malware classification testbed is provided in Figure 1, which indicates the four distinct components that compose the system:

[1] Drupal CMS, http://www.drupal.org

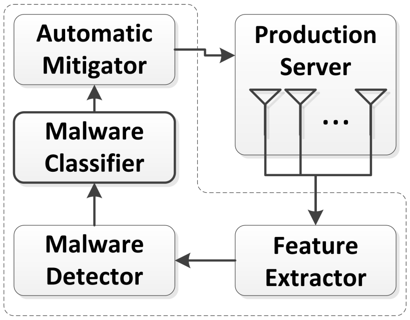

Fig. 2. Self-protecting system showing detector, classifier, and mitigator

- The *testing platform* hosts a collection of VMs, each running Microsoft Windows[2] Server 2012 and executing the computation described above. The sensors monitoring the VMs are installed both on the VMs and on the testing platform, which is where feature extraction occurs. During testing, these virtual machine hosts are infected with malware.
- The *network simulator* provides means for malware that normally communicate over the Internet to exercise some of their network-centric features on our isolated testbed. It redirects all outgoing network connections to a honeypot. We use the Dionaea[3] honeypot, which works by completing incoming connection requests and attempting to obtain malicious payloads from connected malware.
- The *controller* receives the feature data from the testing platform and performs classification.
- The *malware harvester* collects malware samples from the wild to use for testing, using both the Dionaea honeypot and a custom harvester that downloads files from blacklisted sites. Each malware sample is analyzed to determine its file type, identifying information, and whether it is detected by various commercial antivirus software, using the free online scanning service VirusTotal[4].

The testbed depicted in Figure 1 is designed for experimental evaluation only. In a production environment, the malware classification system would be deployed as depicted in Figure 2. The figure shows a self-protecting system with a server being monitored by a series of sensors whose raw feature data are sent to a feature extractor. The feature extractor processes the raw data and sends the extracted features to a malware detector. If malware are detected, the malware classifier is used to classify the malware, and the identifying information output by the classifier is used to determine appropriate mitigation. The closed-loop system is designed to work with both VM and bare metal hosts. Feature extraction, detection, and classification are all intended to be performed outside the monitored environment to minimize the risk of the malware tampering with those components. The focus of this paper is on the *Malware Classifier* component of the system depicted in Figure 2.

B. Features and Datasets

The goal of this work is to classify malware according to a set of observable features extracted from the hosts on which the malware execute. In order to be effective for online malware classification, these features must be easy to extract and must provide sufficient information to discriminate among different malware classes. Accordingly, we select three distinct types of features to use for classification, namely performance monitor, system call, and system call sequence features.

Performance monitors are a feature of the Microsoft Windows family of operating systems that provide a mechanism for remote data collection [24]. Available data include CPU, disk, network and memory usage statistics; information about individual processes and threads; information about individual VM instances; and application-specific information, such as web server and database server statistics. For our study we consider a set of 653 distinct performance monitor sensors. This set of sensors was chosen after a preliminary study of the data reported by all of the available performance monitor sensors on the described testbed. We eliminated from consideration those sensors that provided monotone non-decreasing or constant outputs, since those sensors would provide no discriminability for detection or classification.

Two distinct types of features are extracted from kernel traces. The *system call* features are the number of calls made to each kernel function per second and the *system call sequence* features are the number of calls made to each unique sequence of two kernel function calls per second. Extraction of these two types of features is performed by a custom-built application that uses the NT Kernel Tracer provided by the Event Tracing for Windows (ETW) facility [24] to track kernel function calls. In total, we observed 235 unique kernel functions and 13,290 unique sequences of two kernel function calls.

For each sensor, we sample the data provided by the sensor once every second, resulting in a time series of measurements, one for every second the server is active. For classification, we divide the time series of data collected from each sensor into two distinct sets: one set labeled *clean*, containing the data collected before the malware sample executes, and another set labeled *infected*, containing the data collected after the sample executes. We refer to the time that the malware begins execution as the *infection time* t_I.

C. Malware Collection

In order to establish ground truth with which to compare our classification results, we use the antivirus (AV) scan results provided by VirusTotal to label the datasets. The AV results consist of the identifying information associated with the detection signatures matching the detected malware. This information typically includes the general category of the sample, the platform the sample targets, and the malware family of which the sample is a member. We only consider malware that match one of the detection signatures provided

[2] Microsoft, http://www.microsoft.com

[3] Dionaea low-interaction honeypot, http://dionaea.carnivore.it

[4] Quintero, Bernardo. VirusTotal, http://www.virustotal.com

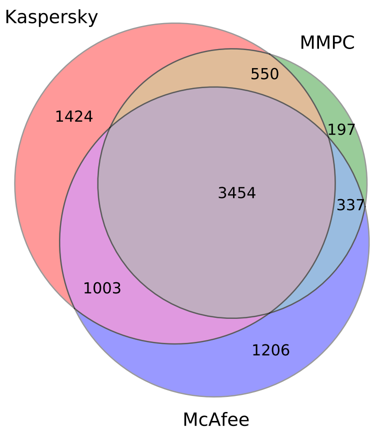

Fig. 3. Venn diagram showing the relationship among the sets of malware samples positively identified by three different AV vendors. The values indicate the number of distinct malware samples in each set.

by Kaspersky Lab[5], the Microsoft Malware Protection Center (MMPC)[6], or McAfee.[7]

Evaluating the accuracy of the classifier is complicated by the absence of a universal, complete, and consistent labeling scheme against which the classification results can be compared. Incompleteness in the AV labeling schemes exists where malware detected by one AV vendor are not detected by another. For example, Figure 3 shows that of the 8,171 identified by *at least one* of the three vendors under consideration, only 3,454 samples are identified by *all three* vendors, while 2,827 are identified by *only one* of the three. Inconsistency in the labeling schemes exists where different AV vendors classify the same sample as belonging to different categories and families. For example, one of the malware samples in our collection is identified as `Trojan.Yakes` by Kaspersky, `Worm.Gamarue` by MMPC, and `PWS.Zbot` by McAfee.

For our evaluation, we consider only malware samples in the portable executable (PE) file format targeting both 32-bit and 64-bit versions of the Windows family of operating systems. As part of our preliminary testing, we execute each of the malware samples in one of the VMs on the testing platform to ensure compatibility of the sample with our system configuration. As a result of this testing, we choose 882 distinct malware samples to use in the experimental evaluation.

Since it is not evident whether the labels provided by one vendor are more accurate than another's, we consider the labels provided by each vendor separately, and we consider the category and family labels separately for each vendor. We consider five different labeling schemes; namely the categories reported by MMPC and Kaspersky, and the families reported by MMPC, Kaspersky, and McAfee. We format the family names as `Category.Family` to distinguish among similarly named families from separate categories. We use the labels from each scheme as ground truth to train and evaluate the accuracy of the classifier.

IV. Classification

In our previous work, we demonstrated how decentralized change-point detection algorithms can be used to infer the execution of malware on a live computer host by detecting changes in the distribution of performance monitor data correlated with malware execution [4]. That system provided only a binary decision indicating that a computer host is either *clean* or *infected*. Here, we describe how such a system can be extended to include additional information about the detected malware, namely its category or family, identifying information that could help guide mitigation of the infected host.

We assume that we are using a detection algorithm that can detect the execution of malware and can infer the approximate time that the infection occurs. The described classification methods can be adapted to deal with uncertainty in the start time by removing from consideration the data samples collected around the estimated start time.

Once a malware infection has been detected and we know its approximate start time, we perform classification first by extracting a new feature vector from a buffer of the most recent feature data. We consider the time series containing the T seconds of clean data collected immediately before the infection time t_I and the time series containing the T seconds of infecteddata collected immediately after t_I. We consider the clean data for the i^{th} feature to be a vector $X_i = \{x_{i,t_I-T}, ..., x_{i,t_I-2}, x_{i,t_I-1}\}$ and the corresponding infected data to be vector $Y_i = \{y_{i,t_I}, y_{i,t_I+1}, ...y_{i,t_I+T-1}\}$.

For each sensor, we quantify that changes that occur about the infection time t_I and construct a new feature vector V containing information about that changes that occur for all of the sensors. First, we consider the normalized change in the mean of the data for the i^{th} sensor:

$$m_i = \frac{\text{mean}(Y_i) - \text{mean}(X_i)}{\sqrt{\text{var}(X_i)}}, \tag{1}$$

where $\text{mean}(X)$ is the sample mean:

$$\text{mean}(X) = \frac{1}{T}\sum_{j=1}^{T} x_j, \tag{2}$$

and $\text{var}(X)$ is the sample variance:

$$\text{var}(X) = \frac{1}{T-1}\sum_{j=1}^{T} (x_j - \text{mean}(X)). \tag{3}$$

Next, we consider the change in the variance of the data for the i^{th} sensor:

$$v_i = \text{var}(Y_i) - \text{var}(X_i). \tag{4}$$

We form a feature vector V composed of these two quantities, $V = \{m_1, m_2, ..., m_N, v_1, v_2, ...v_N\}$, where N is the total number of sensors. We construct this vector V separately for each malware sample and use the resulting set of vectors for

[5] Kaspersky Lab, http://www.kaspersky.com

[6] MMPC, http://www.microsoft.com/security/portal

[7] McAfee, http://www.mcafee.com

training and testing the classifier. The underlying assumption for the classification task is that these vectors V contain information that can be used to discriminate among different classes of malware. We use the changes in the means and variances of the feature data that occur about the infection time t_I to build a decision tree for classification.

A. Decision Trees and Random Forests

To perform classification, we use binary decision trees that use the computed change vectors V as inputs. The binary decision trees are composed of a set of nodes and edges, where each interior node represents a simple threshold test performed using one of the components m_i or v_i of the vector V, each edge indicates whether the threshold was exceeded at a particular node, and each leaf node corresponds to a label. We classify newly observed malware by evaluating the decision tree for the vector V computed for a particular malware sample, arriving at a leaf node that indicates the computed class of the newly observed malware.

We take a supervised learning approach to decision tree creation, wherein we use the AV class or family labels as ground truth and train the decision tree using the change vectors V collected for a set of malware samples whose variants we wish to identify. We use the labeled training data to construct the decision tree using the CART decision tree algorithm [25].

The CART algorithm builds decision trees recursively. It begins by considering all of the malware and their corresponding labels and vectors V at the root node of the tree. It selects a component of V and a threshold such that after performing the threshold test on the selected component, the malware are split into two distinct sets associated with the two children of the root node. The goal is to select the component and threshold such that the sets of malware that share the same label are grouped together after the resulting split. This is accomplished by selecting a component and threshold that minimize the weighted average of the Gini impurity of the two resulting sets. Consider a set of malware $\mathcal{X}$ with labels in the set $\{c_1, c_2, ...c_K\}$ where K is the number of distinct labels. We define p_{c_k} to be the fraction of the malware in set $\mathcal{X}$ with label c_k. The Gini impurity G of the set $\mathcal{X}$ is defined as:

$$G(\mathcal{X}) = \sum_{k=1}^{K} p_{c_k} \left(1 - p_{c_k}\right), \tag{5}$$

where minimum impurity (0) is achieved when a set contains only malware samples having the same label. Once a node contains only malware sharing the same label, it becomes a leaf node of the decision tree. The algorithm terminates based on tunable parameters that determine the complexity of the tree, including the minimum number of training instances assigned to each leaf node and the minimum number of training instances required to split a node.

We selected decision tree models for classification due to the white box decision models they provide, their low computational complexity during classification, and the promising accuracy they provided during preliminary testing. Since decision trees tend to suffer from overfitting, we use a modified version of the decision tree classifier, the random forest classifier [26]. The random forest classifier is an ensemble learner that trains a *collection* of decision trees to use for classification. To ensure that each of the decision trees is different, the decision tree training algorithm is modified to consider a only randomly selected subset of features at each node. To perform classification, the algorithm computes the predicted label for each individual tree and averages over the results. In our evaluation, we use the implementation of the CART decision tree algorithm and random forest algorithm provided by Pedregosa *et al.* [27]

B. Feature Selection

In order to prevent over-fitting and reduce complexity, we performed feature selection prior to our experimental evaluation of the classifier. Feature selection reduces the number of features that must be monitored, thereby decreasing the overhead introduced by the monitoring, feature extraction, and classification components of the system. Feature selection is accomplished by ranking features according to the amount of information they convey for the classification task.

We accomplish feature selection using the same random forest algorithm that we use for classification. We train a forest of $1{,}000$ randomized trees, and compute the importance of each feature as a function of the number of times it appears in each decision tree [27]. We then rank the features in terms of their importance and select the most important features to use for classification.

V. Experimental Results

In this section, we present experimental results obtained using the described classifier on the data gathered using the testbed. First, we examine the accuracy of the classifier as a function of the time elapsed between the malware executing and classification being performed. Next, we present the accuracy of the system as a function of the number of features chosen during feature selection. Finally, we present the accuracy of the classifier using each of the labeling schemes. The results presented in this section were obtained using 4-fold stratified cross validation. For each labeling system, we considered only those malware classes for which we have four or more distinct samples sharing the same label.

A. Classification Accuracy vs. Time

First, we examine the effect that the amount of time elapsed after the infection time t_I has on the accuracy of the classifier. Since our goal is ultimately to use the classification results to guide automatic mitigation, making a classification decision as quickly as possible is desired to limit the damaging effects of the malware.

We define *classification accuracy* as the fraction of the total malware samples correctly labeled by the classifier during cross-validation. We say a malware sample is correctly labeled by the classifier if its computed label matches the ground truth

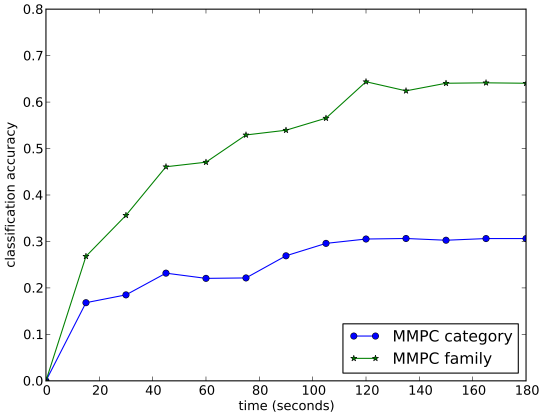

Fig. 4. Classification accuracy as a function of the time elapsed between infection and classification

label under the chosen labeling scheme. Figure 4 shows the empirical accuracy of the classifier as a function of T, the number of seconds of feature data collected both before and after t_I used for classification. These results were obtained using the top 100 of each type of feature chosen during feature selection. The plot shows that the detection accuracy increases as a function of time. For both the MMPC category and family labels, the accuracy reaches 99% of its maximum observed value after 120 seconds.

Because more accurate results require the malware to be allowed to execute for a longer period of time, it may be desirable to perform preliminary mitigation, such as the isolation or imaging of the infected host, before a more accurate decision is reached. Furthermore, it may be possible to make a quicker determination depending on the label of the detected malware. For example, the classifier correctly labels all of the `Backdoor.Kelihos` and `Adware.Kremiumad` samples using 60 seconds of data; but requires 120 seconds of data to correctly label all the `TrojanSpy.Zapemli` and `TrojanClicker.Clidack` samples. These results indicate that certain families of malware may be more easily discernible and therefore not require as much time or data to be correctly identified by the classifier.

The time required to extract the feature data from the raw sensor data, compute the change vectors, and perform classification using the current system implementation is under five seconds. Thus, the low complexity of the decision tree classification algorithm – $O(\log(n))$, where n is the number of training samples – enables classification accuracy in excess of 60% within 125 seconds of the malware executing on the testbed. Extending the system to use additional features or a larger corpus of training data will result in a more complex decision tree an an increased classification time.

B. Feature Selection

The accuracy of the classification system is largely dependent on the choice of features. Choosing too few features leads to insufficient discrimination among classes while selecting too many features introduces unnecessary overhead and leads to overfitting. Figure 5 shows the detection accuracy of the classifier as a function of the number of features chosen from each type for the MMPC family labels. The features are chosen in order of decreasing importance, using the importance values computed during feature selection. These results were obtained using $T = 120$ seconds of data.

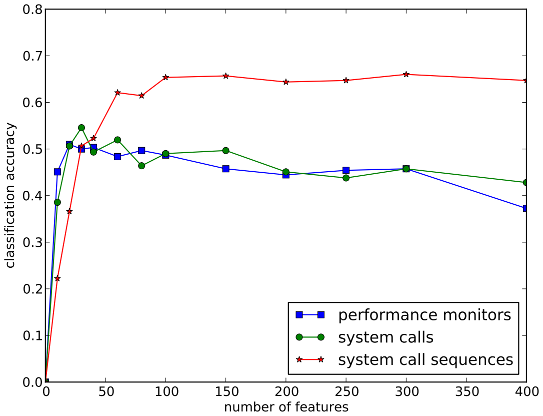

Fig. 5. Detection accuracy comparison for different feature sets

TABLE I
DETECTION ACCURACY OF CLASSIFIER USING EACH LABELING SCHEME

Label	Distinct	Accuracy(%)
Kaspersky categories	16	35.9
McAfee families	17	40.1
MMPC categories	13	44.9
Kaspersky families	35	45.0
MMPC families	32	66.8

Figure 5 indicates that the system call sequences provide the most discrimination among the classes, while the system call and performance monitor features provide approximately the same amount of discrimination. The plot also indicates that each set of features exhibits a peak beyond which the accuracy decreases, likely as a result of overfitting the decision tree models to features that provide little to no discrimination. The peak occurs at approximately 50 features for the performance monitors, 30 features for the system calls, and 150 features for the system call sequences. We use this reduced set of features to obtain the detailed detection results presented in the following section.

C. Labeling System Comparison

Finally, we compare the performance of our classifier using each of the five different labeling systems. In this section, we present results achieved using the 230 features described in the preceding section and using $T = 180$ seconds of data. Table I presents the classification accuracy achieved for each of the five labeling schemes along with the number of distinct labels in each scheme. The accuracy results indicate that, for the Kaspersky and MMPC labels, the classification accuracy increases with the specificity of the labeling systems, *i.e.,* more accurate results were achieved with the family labels than with the category labels.

Table I also indicates that the classifier achieved its highest accuracy when using the MMPC family labels, with the MMPC labels providing significantly better accuracy than the McAfee and Kaspersky family labels. To see why this is the case, we examine the the per-label accuracy of the classifier in terms of *precision* and *recall*.

We consider a particular class c_k and the following three quantities: tp, the number of malware samples in class c_k correctly labeled by the classifier (*true positives*); fn, the number of malware samples in class c_k incorrectly labeled by the classifier (*false negatives*); and fp, the number of malware samples not in class c_k that are incorrectly classified as belonging to c_k (*false positives*). We define the *precision* and *recall* in terms of these three quantities as:

$$Precision = \frac{tp}{tp + fp} \tag{6}$$

$$Recall = \frac{tp}{tp + fn}. \tag{7}$$

Precision quantifies how often malware samples are incorrectly identified as belonging to class c_k, where low precision means the classifier inforrectly classified a large contingent of malware samples as belonging to c_k. Recall is the fraction of malware samples in class c_k that are correctly identified by the classifier.

We begin by discussing the McAfee family labels, which, as shown in Table I, afforded an accuracy comparable to the Kasperky and MMPC *category* labels. The reason that the classifier using the McAfee labels underperformed the other family labeling schemes lies in the fact that a large number of the malware have generic labels in the McAfee family labeling scheme: for example, `Artemis` (236 samples) and `Generic` (122 samples). These labels exhibited both poor precision (0.59 and 0.28) and recall (0.39 and 0.26), while other labels defining specific malware families, *e.g.,* `Clidak` and `PWS-Zbot`, exhibited a higher precision (1.00 and 0.74) and recall (0.75 and 0.69). The generic labels suffer from the same deficiency of the category labels, namely that a single generic label is often insufficient to describe a specific malware sample.

Table II presents the precision and recall for each label in the Kaspersky family labeling scheme. When compared to the McAfee labels, the Kaspersky labels provide more specific information about individual malware families, resulting in an overall improvement in classification accuracy. The results in Table II show that specific families, such as `Trojan-Downloader.Genome` and `Trojan-PSW.Tepfer` exhibit both a high precision and recall, indicating that the classifier was able to discern them from the rest of the group. Still others, such as `Backdoor.DarkKomet` and `Constructor.Zbot` exhibit high recall but low precision, indicating that the classifier often misclassified other malware as belonging to these two classes. Similar to the McAfee labels, the Kaspersky labels also contain generic labels that exhibit a particularly low precision and recall, such as `Trojan.Generic`.

TABLE II
DETAILED DETECTION RESULTS FOR KASPERSKY FAMILY LABELS

Label	Precision	Recall	Samples
Backdoor.Agent	0	0	4
Backdoor.DarkKomet	0.56	0.93	15
Constructor.Zbot	0.41	1	7
Server-Proxy.CCProxy	0.88	1	7
Trojan-Banker.Agent	0.67	0.5	4
Trojan-Banker.Banbra	0.33	0.4	10
Trojan-Banker.Banker	0.33	0.14	7
Trojan-Clicker.Agent	0	0	5
Trojan-Downloader.Adload	0.75	0.43	7
Trojan-Downloader.Agent	0.28	0.22	23
Trojan-Downloader.Andromeda	0.4	0.86	7
Trojan-Downloader.Banload	0	0	6
Trojan-Downloader.Delf	0.33	0.2	5
Trojan-Downloader.Generic	0.15	0.4	10
Trojan-Downloader.Genome	0.88	1	22
Trojan-Dropper.Agent	0.33	0.47	19
Trojan-Dropper.Dapato	0.67	0.29	7
Trojan-Dropper.Injector	0.25	0.11	9
Trojan-PSW.Tepfer	0.94	0.79	19
Trojan-Ransom.Blocker	1	0.25	4
Trojan-Spy.KeyLogger	1	0.71	7
Trojan-Spy.Zbot	0.88	0.93	30
Trojan.Agent	0.05	0.04	28
Trojan.Agent2	0	0	6
Trojan.Bublik	0	0	6
Trojan.Crypt	0.07	0.14	7
Trojan.Generic	0.32	0.11	53
Trojan.Genome	0	0	4
Trojan.Inject	0	0	4
Trojan.Jorik	0.22	0.24	29
Trojan.StartPage	0.68	0.88	24
Trojan.VB	0	0	4
Trojan.VBKrypt	0.32	0.58	12
Worm.AutoRun	0.5	1	4
Worm.Ngrbot	0.5	0.86	7
Average	0.43	0.45	

The precision and recall values for the MMPC family labels are presented in Table III. The major reason that the classifier using the MMPC labels outperforms the classifiers using the McAfee or Kaspersky labels is the lower concentration of malware with generic labels. While the MMPC labeling scheme does include some generic labels that exhibit a particularly low precision and recall, such as `Trojan.Comrerop`, `Trojan.Malagent`, `TrojanDownloader.Banload`, and `Trojan.Meredrop`, the number of malware samples described by these labels is much lower than in the Kaspersky or McAfee labeling schemes.

Table III also indicates that the classification performance varies for different labels, with some achieving perfect precision or recall (*e.g.,* `TrojanSpy.Zapemli`) and others having zero precision and recall (*e.g.,* `Backdoor.Farfli`). In general, the classes with the highest precision and recall are those whose family labels define specific a specific set of functionality or specific heritage, such as `TrojanSpy.Zapemli`, `PWS.Zbot`, and `Backdoor.Kelihos`, and those exhibiting the lowest precision and recall were those whose family labels define only a broad category of functionality. For example,

TABLE III
DETAILED DETECTION RESULTS FOR MMPC FAMILY LABELS

Label	Precision	Recall	Samples
Adware.Kremiumad	0.84	1	16
Backdoor.Farfli	0	0	4
Backdoor.Fynloski	0.71	0.96	28
Backdoor.IRCbot	0.4	0.5	4
Backdoor.Kelihos	0.94	1	15
HackTool.CCProxy	0.88	1	7
PWS.Fareit	0.8	0.67	6
PWS.OnLineGames	0.83	0.83	6
PWS.Zbot	0.9	0.95	39
Trojan.Comrerop	0	0	4
Trojan.Danginex	0.57	0.5	8
Trojan.Dynamer!dtc	0.29	0.29	7
Trojan.EyeStye	0.5	0.4	5
Trojan.Ircbrute	0.67	0.33	6
Trojan.Malagent	0	0	4
Trojan.Meredrop	0	0	5
Trojan.QHosts	1	0.8	5
Trojan.Sisron	0.5	0.25	8
Trojan.Urelas	1	0.25	4
TrojanClicker.Clidak	0.5	1	13
TrojanClicker.Delf	0.56	1	5
TrojanDownloader.Banload	0	0	8
TrojanDownloader.Small	0.59	0.76	17
TrojanProxy.Banker	0.5	0.2	5
TrojanSpy.Bancos	0.22	0.25	8
TrojanSpy.Banker	0	0	4
TrojanSpy.Zapemli	1	1	4
VirTool.DelfInject	0.33	0.12	8
VirTool.Obfuscator	0	0	6
VirTool.VBInject	0.62	0.62	8
Worm.Dorkbot	0.89	0.8	10
Worm.Gamarue	0.58	0.92	12
Average	0.61	0.67	

the TrojanDownloader.Banload label is generically defined as identifying Trojans that download other malware. In order to achieve better classification performance for malware in these categories, we will likely have to consider subfamily identifiers within these categories that have a much more narrow definition of the malware they contain. This is left as the subject of future work as it requires multiple samples within each of the distinct subfamilies.

Figure 6 shows the normalized confusion matrix for the classifier using the MMPC family labels, where the row labels indicate the ground truth and the column labels indicate the labels assigned by the classifier. The shade intensity indicates the percentage of the malware in each row that are classified as the label indicated by the column name, where black indicates 100% and white indicates 0%. For example, it shows that the malware labeled Backdoor.Farfli by MMPC were misclassified most often as Worm.Gamarue and Backdoor.IRCbot. That the malware labeled Backdoor.IRCbot were also often misclassified as Backdoor.Farfli is an indication that these two malware families share behavioral similarities in terms of the features used for classification. In fact, these two families do share common features, such as DDoS functionality.

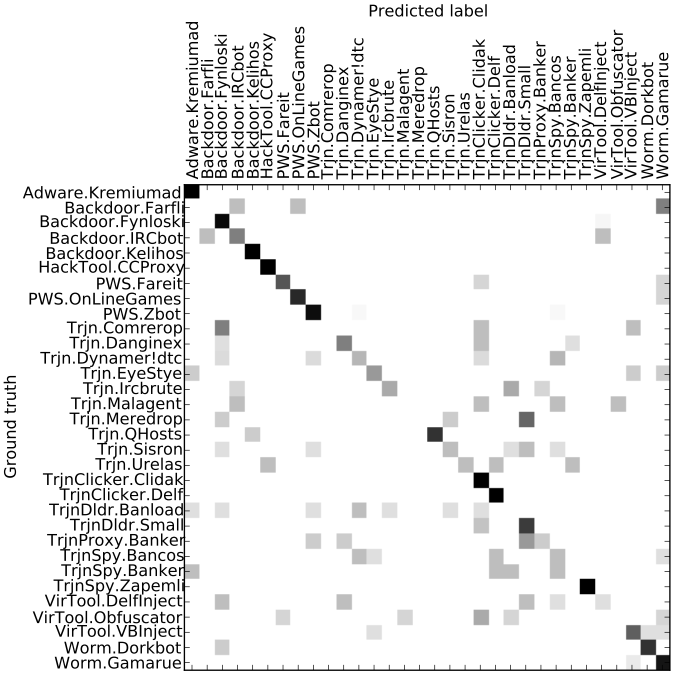

Fig. 6. Normalized confusion matrix for MMPC family labels

The confusion matrix also shows some useful information about the generic labels. For example, it shows that the Trojan.Comrerop samples were most often misclassified as Backdoor.Fynloski, indicating that these samples are either behaviorally similar to, or variants of, Backdoor.Fynloski. It also shows that the Trojan.Malagent, TrojanDownloader.Banload, and Trojan.Meredrop samples are often misclassified as a variety of different labels, indicating that the samples within those families are behaviorally dissimilar and may belong to different families.

The confusion matrix in Figure 6 lacks a category-level, block-wise diagonal structure that would arise if misclassifications were more common among malware of the same *category*. For example, the TrojanDownloader.Banload samples are misclassified as Adware, Backdoors, and Trojans. In this case, the misclassification may be due to the nature of TrojanDownloaders, whose primary function is to install additional malware. However, this lack of a block-wise diagonal structure is a further indication that the category labels are insufficient for describing newly classified malware.

The results presented in this section indicate that malware whose labels define a specific heritage or set of behavioral features are classified more accurately than malware whose labels define a broad range of malicious behavior. The insufficiency of the category labels, combined with the fact that the best performance was achieved for the families with the most specific definitions, indicates that the overall performance of the classifier might be able to be improved by separating more general family definitions into subfamilies.

D. Automatic Mitigation

The goal of this work is to create a malware classification system that determines the class of new, previously unseen malware on production hosts by computing their behavioral similarity to known malware. Once a new malware sample has been classified, our goal is to apply automatic mitigations to prevent data loss, data theft, or further spread of the malware.

To illustrate how the classifier using the MMPC family labels can be used for automatic mitigation, we consider the example of the fifteen `Backdoor.Kelihos` malware samples used during our experimental evaluation. Kelihos is a botnet used primarily for BitCoin theft and sending spam. Accordingly, appropriate mitigation might include blocking the ports used for sending spam or used for the command and control of the botnet. Furthermore, the 15 variants we studied during our experimental evaluation also exhibited common behaviors, including using the same names for the processes created during execution and making the same configuration changes to the infected computer. Thus, terminating the processes suspected to be associated with Backdoor.Kelihos, deleting the executables that spawn the processes, and undoing known configuration changes might also be effective mitigations to apply automatically.

Thus, the strategy for automatic mitigation is the following: For canonical examples of malware families, appropriate mitigations are first developed in a controlled laboratory setting through detailed malware analysis (or using existing documentation such as that provided by AV vendors). The described malware classification system will then apply the mitigations whenever a newly detected malware sample is determined to belong to one of these known families.

While our empirical observation indicates that such a set of strategies might exist for the `Backdoor.Kelihos` samples (and other samples within well-defined categories), the inconsistency in antivirus labeling systems and the inaccuracy of our classifier for certain malware family labels indicates that generic mitigation might be infeasible for other families using the described labels. However, it might be possible to perform mitigation at the subfamily level, where malicious behaviors are more well-defined, or apply generic mitigations such as isolation when a malware sample cannot be accurately classified.

VI. Discussions

We presented experimental results demonstrating how behavioral feature data can be used for online malware classification on hosts under heavy background computational load and provided an example of how such classification results could be used for mitigation. Yet, this study also raises a few key questions and presents opportunities for future research.

The described classifier uses three distinct types of observable features for the classification task, the majority of which characterize host-wide behavior. It may be desirable consider additional features, such as longer system call sequences, performance data provided by other applications, or more specific information about access to critical system resources, such as the network and filesystem. Extracting features at the process or thread level may also provide more accurate classification results while also helping to identify the source of a detected malware infection and thereby aiding in mitigation.

Next, the described system uses decision trees for classification, which label each newly detected malware sample as one of the existing labels learned during training. If a new malware sample does not belong to one of the existing labels, the decision trees will simply label it (incorrectly) with one of the existing labels. To address the issue of new malware samples that are not well-described by an existing label, we have begun to evaluate other types of classifiers better suited to this situation. For example, we have begun exploring the use of the nearest centroid algorithm [28], which can be used to establish a notion of distance from the centroids that define the labels. Malware whose distance to the closest centroid exceeds a threshold might be considered new malware that warrant further analysis. To deal with the behavioral diversity of malware having a particular label, we are exploring using the k-means clustering algorithm [29] to establish multiple centroids for characterizing each malware label and also exploring the use of subfamily labels for classification.

For the 882 malware samples considered in our experimental evaluation, the number of samples described by each label varied significantly. For example, in the MMPC family naming scheme, we studied four samples of Backdoor.Farfli and 39 samples of PWS.Zbot. This disparity is largely driven by the malware samples available to us for the evaluation: For many malware samples, we only had a small number of distinct variants within a class. The evaluation of the described classifier on a larger malware corpus containing more distinct malware variants within each family is a subject of our ongoing work. Considering a larger corpus of malware is also important for evaluating the the accuracy and usefulness of the system for deployment in production environments.

Finally, the described experimental procedures were designed to test the performance of the classifier under a heavy and diverse load. While the type and intensity of the load varies for each of the experimental runs, the entire experiment is limited to a single system configuration. Thus, additional work must be performed to characterize the performance of the classifier on system with different configurations and different load conditions. Particularly, a systematic study of the background noise caused by the benign workload on a host computer, and the effects the background noise has on the detection and classification tasks, is necessary.

The work presented here is part of an ongoing effort to create self-protecting servers that can automatically detect and mitigate new malware threats. As part of this work, we intend to continue to evaluate the described classifier on a larger malware corpus and on a variety of different hosts, evaluating the effects of changes in usage patterns and configuration on the performance of the classifier. Particularly, we are interested in establishing whether the classifier can be generalized to a wide variety of host configurations and uses.

VII. Conclusions

We presented a behavioral malware classification system designed to be deployed on production servers, using changes in sets of observable behavioral features for online malware classification. The system uses performance monitor data, system call frequencies, and frequencies of pairs of system calls as input features for the classifier. We presented the results of an empirical evaluation of the classification system using random forests to detect previously unseen malware variants on live systems under heavy workloads, and analyzed the accuracy of our system using different feature sets and labeling schemes.

The presented results indicate that the behavioral classifier can correctly identify new malware variants within certain families of malware in the presence of a complex computational load. We intend to continue to develop and test the described classification system, expanding our malware corpus and refining our classification techniqes, evaluating the classifier's accuracy under different load conditions, and developing training procedures appropriate for more general deployment. This work is part of a larger effort to develop a self-protecting system that can detect, classify, and mitigate malware infections, especially infections caused by new variants of known malware that evade traditional antivirus software.

VIII. Acknowledgments

This research is sponsored by a Secure and Trustworthy Cyberspace (SaTC) award from the National Science Foundation (NSF), under grant CNS-1228847.

References

[1] M. Venable, A. Walenstein, M. Hayes, C. Thompson, and A. Lakhotia, "Vilo: a shield in the malware variation battle," in *Virus Bulletin*, 2007.

[2] G. Jacob, H. Debar, and E. Filiol, "Behavioral detection of malware: from a survey towards an established taxonomy," *Journal in Computer Virology*, vol. 4, pp. 251–266, 2008, 10.1007/s11416-008-0086-0. [Online]. Available: http://dx.doi.org/10.1007/s11416-008-0086-0

[3] R. Canzanese, M. Kam, and S. Mancoridis, "Inoculation against malware infection using kernel-level software sensors," in *Proceedings of the 8th ACM international conference on Autonomic computing*, ser. ICAC '11. New York, NY, USA: ACM, 2011, pp. 101–110.

[4] ——, "Multi-channel change-point malware detection," in *Seventh International Conference on Sofware Security and Reliability (SERE)*, 2013.

[5] S. A. Hofmeyr, S. Forrest, and A. Somayaji, "Intrusion detection using sequences of system calls," *Journal of Computer Security*, vol. 6, no. 3, pp. 151–180, 1998.

[6] R. Moskovitch, Y. Elovici, and L. Rokach, "Detection of unknown computer worms based on behavioral classification of the host," *Comput. Stat. Data Anal.*, vol. 52, pp. 4544–4566, May 2008.

[7] M. Schultz, E. Eskin, F. Zadok, and S. Stolfo, "Data mining methods for detection of new malicious executables," in *Security and Privacy, 2001. S P 2001. Proceedings. 2001 IEEE Symposium on*, 2001, pp. 38 –49.

[8] J. Z. Kolter and M. A. Maloof, "Learning to detect and classify malicious executables in the wild," *J. Mach. Learn. Res.*, vol. 7, pp. 2721–2744, December 2006.

[9] S. Cesare, Y. Xiang, and W. Zhou, "Malwise: An effective and efficient classification system for packed and polymorphic malware," *Computers, IEEE Transactions on*, vol. 62, no. 6, pp. 1193–1206, 2013.

[10] K. Iwamoto and K. Wasaki, "Malware classification based on extracted api sequences using static analysis," in *Proceedings of the Asian Internet Engineeering Conference*, ser. AINTEC '12. New York, NY, USA: ACM, 2012, pp. 31–38.

[11] Y. Park and D. Reeves, "Deriving common malware behavior through graph clustering," in *Proceedings of the 6th ACM Symposium on Information, Computer and Communications Security*, ser. ASIACCS '11. New York, NY, USA: ACM, 2011, pp. 497–502.

[12] Y. Zhong, H. Yamaki, and H. Takakura, "A malware classification method based on similarity of function structure," in *Applications and the Internet (SAINT), 2012 IEEE/IPSJ 12th International Symposium on*, 2012, pp. 256–261.

[13] L. Nataraj, V. Yegneswaran, P. Porras, and J. Zhang, "A comparative assessment of malware classification using binary texture analysis and dynamic analysis," in *Proceedings of the 4th ACM workshop on Security and artificial intelligence*, ser. AISec '11. New York, NY, USA: ACM, 2011, pp. 21–30.

[14] K. Rieck, P. Trinius, C. Willems, and T. Holz, "Automatic analysis of malware behavior using machine learning," *Journal of Computer Security*, 2011.

[15] M. Bailey, J. Oberheide, J. Andersen, Z. M. Mao, F. Jahanian, and J. Nazario, "Automated classification and analysis of internet malware," in *Proceedings of the 10th International Symposium on Recent Advances in Intrusion Detection (RAID'07)*, September 2007.

[16] I. Firdausi, C. Lim, A. Erwin, and A. Nugroho, "Analysis of machine learning techniques used in behavior-based malware detection," in *Advances in Computing, Control and Telecommunication Technologies (ACT), 2010 Second International Conference on*, 2010, pp. 201 –203.

[17] J. Hegedus, Y. Miche, A. Ilin, and A. Lendasse, "Methodology for behavioral-based malware analysis and detection using random projections and k-nearest neighbors classifiers," in *Computational Intelligence and Security (CIS), 2011 Seventh International Conference on*, 2011, pp. 1016–1023.

[18] T. Lee and J. J. Mody, "Behavioral classification," in *Proceedings of the 15th European Institute for Computer Antivirus Research (EICAR 2006) Annual Conference*, 2006.

[19] S. Nari and A. A. Ghorbani, "Automated malware classification based on network behavior," in *Computing, Networking and Communications (ICNC), 2013 International Conference on*, 2013, pp. 642–647.

[20] N. Stakhanova, M. Couture, and A. Ghorbani, "Exploring network-based malware classification," in *Malicious and Unwanted Software (MALWARE), 2011 6th International Conference on*, 2011, pp. 14–20.

[21] D. Lobo, P. Watters, and X. Wu, "Rbacs: Rootkit behavioral analysis and classification system," in *Knowledge Discovery and Data Mining, 2010. WKDD '10. Third International Conference on*, 2010, pp. 75–80.

[22] B. Anderson, C. Storlie, and T. Lane, "Improving malware classification: bridging the static/dynamic gap," in *Proceedings of the 5th ACM workshop on Security and artificial intelligence*, ser. AISec '12. New York, NY, USA: ACM, 2012, pp. 3–14.

[23] M. Neugschwandtner, P. M. Comparetti, G. Jacob, and C. Kruegel, "Forecast: skimming off the malware cream," in *Proceedings of the 27th Annual Computer Security Applications Conference*, ser. ACSAC '11. New York, NY, USA: ACM, 2011, pp. 11–20.

[24] M. E. Russinovich, D. A. Solomon, and A. Ionescu, *Windows Internals*, 6th ed. Microsoft Press, April 2012, no. Part 1, 0735648735.

[25] L. Breiman, J. Friedman, C. J. Stone, and R. Olshen, *Classification and Regression Trees*. Chapman and Hall/CRC, 1984.

[26] L. Breiman, "Random forests," *Mach. Learn.*, vol. 45, no. 1, pp. 5–32, Oct. 2001.

[27] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[28] Q. Zhang and S. Sun, "A centroid k-nearest neighbor method," in *Proceedings of the 6th international conference on Advanced data mining and applications: Part I*, ser. ADMA'10. Berlin, Heidelberg: Springer-Verlag, 2010, pp. 278–285.

[29] C. M. Bishop, *Pattern Recognition and Machine Learning*. Springer, 2006.