# Run-time Classification of Malicious Processes Using System Call Analysis

Raymond Canzanese* and Spiros Mancoridis†
*Dept. of Electrical and Computer Engineering
†College of Computing and Informatics
Drexel University, Philadelphia, PA, USA
{rcanzanese,mancors}@drexel.edu

Moshe Kam
Newark College of Engineering
New Jersey Institute of Technology
Newark, NJ, USA
kam@njit.edu

***Abstract*—This study presents a malware classification system designed to classify malicious processes at run-time on production hosts. The system monitors process-level system call activity and uses information extracted from system call traces as inputs to the classifier. The system is advantageous because it does not require the use of specialized analysis environments. Instead, a 'lightweight' service application monitors process execution and classifies new malware samples based on their behavioral similarity to known malware. This study compares the effectiveness of multiple feature sets, ground truth labeling schemes, and machine learning algorithms for malware classification. The accuracy of the classification system is evaluated against process-level system call traces of recently discovered malware samples collected from production environments. Experimental results indicate that accurate classification results can be achieved using relatively short system call traces and simple representations.**

## I. Introduction

Automatic classification techniques are used to characterize newly discovered malware samples, with goals of generating mitigation procedures, remediation procedures, and detection signatures. Such techniques typically involve executing samples in a sandbox environment to observe their behaviors. These environments provide reports that include information about filesystem, registry, memory, library, and system call activities [1], [2]. Executing malware samples in such environments is a resource-intensive process, presenting a formidable challenge for antivirus (AV) vendors who seek to analyze every new malware sample. One vendor estimates that more than 6 new malware samples are discovered every second, a pace that is projected to increase through the end of 2015 [3]. Among such newly discovered malware samples are new malware families and new variants of existing malware families. Malware authors use specialized tools to generate new malware variants, with the intentions of overwhelming AV vendors and evading analysis [4]. Some malware samples are specifically crafted to alter themselves as they propagate, automatically creating new variants [5], [6].

Suspected malware samples are typically submitted to AV vendors during postmortem analysis, after they have been discovered on an infected host. The goal of this study is to leverage data collected from infected hosts to facilitate malware analysis. To this end, the described malware classification includes a 'lightweight' host-agent that automatically monitors the system call traces of every process executing on a host. A system call is a request for an operating system service, and a system call trace provides a listing of the system calls in the order in which they occur. The described system uses features extracted from system call traces to determine a suspected malware sample's behavioral similarity to known malware. This system is advantageous because it provides immediate classification results that can be used to guide postmortem analysis, mitigation, and remediation. It is intended to be used in conjunction with behavioral detection systems that use similar feature sets to infer malicious behaviors [7].

The contributions of this study are three-fold. First, this study considers multiple feature sets and classification algorithms that have previously been demonstrated to be useful for malware classification. These techniques are applied to system call traces collected from the execution of more than 76,000 distinct malware samples. The system call traces were collected on hosts running multiple versions of Microsoft Windows. Second, this study compares the effectiveness of various ground truth labeling schemes derived from AV naming systems. The experimental results identify which ground truth labels provide the highest classification accuracy. Finally, this study presents the design of a run-time malware classification system, with algorithms and parameters selected based on experimental evaluation.

## II. Related work

The related work in automatic malware classification describes both static and dynamic analysis techniques. Static analysis techniques use features extracted from executable files, while dynamic analysis techniques use observed characteristics of executing malware. A large portion of related work in malware classification focuses primarily on systems designed for off-line analysis.

Feature sets commonly used for static analysis include strings, byte sequences, and program structure [8]–[10]; API imports and API calls [11]; and control flow information [12]–[14]. Commonly used classification algorithms include naïve Bayes, decision trees, support vector machines (SVMs) [8], [10], image classification techniques [9], hierarchical clustering analysis [11], and graph matching and clustering algorithms [12]–[14].

Dynamic analysis techniques typically require malware samples to be executed in a sandbox environment. Nearest neighbor techniques have been used to classify malware based on the reports collected from such environments [2], [15]–[17]. Alternate algorithms have also been explored, including naïve Bayes, SVMs, decision trees, and neural networks [16]. Hierarchical clustering analysis based on file, registry, and process activity has also provided promising results [18]. Nearest neighbor techniques have demonstrated success using system call traces extracted using cwsandbox [19], [20]. Clustering algorithms have been successful in classifying malware based on activities derived from system call traces [21]. Two recent studies leverage the strengths of static analysis (ease of feature extraction) and dynamic analysis (richness of datasets) to provide improved classification results [22], [23]. Dynamic analysis has also been applied to specific malware subsets, including malware that communicate over a network [24], [25] and rootkits that use API hooks [26]. This study focuses on dynamic analyisis – particularly, system call analysis – but differs from the related work in that it performs classification on-line on production hosts without the use of any specialized sandbox environments.

## III. Feature extraction from system call traces

The system call traces used in this study were collected using a custom host-agent known as the System Call Service (SCS)[1]. The SCS is a service application that collects system call traces from running processes. For this study, system call traces are represented as vectors of system call $n$-gram frequencies. This representation is referred to as the bag-of-system-call-$n$-grams representation, adapted from the bag-of-words representation popular in document classification and intrusion detection [27]. A system call $n$-gram is a sequence of $n$ contiguous system calls appearing in a trace. The feature vectors are stored as TF-IDF transformed vectors of $n$-gram frequencies, where TF-IDF transformation is used to account for the fact that system call $n$-gram frequencies scale to different orders of magnitude. Given a vector of $\mathbf{x}$ of raw frequencies, the TF-IDF transformation considers the term frequency (TF) as the logarithmic frequency,

$$\mathrm{TF}(\mathbf{x}) = \log(\mathbf{x}+1)\,. \tag{1}$$

The inverse document transformation (IDF) applies weights to features inversely according to their frequency of appearance. Considering $p$ to be the number of traces in a set $\mathcal{X}$ and $\mathbf{d}$ to be a vector counting the traces in which each $n$-gram appears, the IDF is

$$\mathrm{IDF}(\mathcal{X}) = \log\left(\frac{1+p}{1+\mathbf{d}}\right) + 1\,. \tag{2}$$

The overall TF-IDF transformation is the element-wise product of the TF and the IDF, represented as $\hat{\mathbf{x}}$.

The feature space of the bag-of-system-call-$n$-grams representation has $465^n$ dimensions, because there are 465 distinct system calls tracked by the SCS. Since certain classification algorithms presented in this work perform poorly with very high-dimensional data (due to the so-called 'curse of dimensionality' [28]), this work considers two dimension reduction techniques. The first, truncated singular value decomposition (SVD), is a matrix factorization algorithm used to project the features onto a lower dimensional feature space [29]. The SVD defines a transformation of a vector from its original representation $\hat{\mathbf{x}}$, to a new representation $\hat{\mathbf{x}}_k$ in a $k$-dimensional space. The parameters of the transformation, $\Sigma_k$ and $U_k^T$, are selected to project the data onto a space defined by the $k$ largest eigenvalues of $\hat{X}\hat{X}^T$, where $\hat{X}$ is a matrix formed by concatenating the set of training vectors. The transformation is

$$\hat{\mathbf{x}}_k = \Sigma_k U_k^T \hat{\mathbf{x}}\,. \tag{3}$$

The second transformation, linear discriminant analysis (LDA), is a generalization of Fisher's Linear Discriminant [28]. Whereas SVD transforms the feature data to eliminate redundancy, LDA considers the class labels and transforms the data to separate the instances of different classes. Given data from $K$ classes, LDA is used to project the data onto a $(K-1)$-dimensional space. LDA assumes the class-conditional distribution of the features to be Gaussian and requires non-redundant input features. Therefore, LDA is only applied to the SVD-transformed feature vectors $\mathbf{x}_k$. LDA seeks a matrix $W$ to maximize the between-class covariance and minimize with within-class covariance of the data in the projected space. The projection is given by

$$\hat{\mathbf{x}}_{K-1} = W\hat{\mathbf{x}}_k\,. \tag{4}$$

For simplicity of representation, $\hat{\mathbf{x}}$ is used in the remainder of this work to refer to the output of the TF-IDF, SVD, and LDA transformations.

## IV. Malware classification algorithms

The classification techniques considered for this study are used in a supervised learning context, wherein the classification models are learned from labeled training data. Each technique includes a training and testing algorithm. The techniques were selected because of their previously demonstrated effectiveness in intrusion detection and malware classification.

### A. Multi-class logistic regression (LR)

LR is a linear classifier that provides a model for computing the probability that a malware samples belongs to a specific class [30]. The LR classifier is desirable because of its computational simplicity during testing and its efficient training process. This study uses a one-versus-all (OVA) approach, which treats the $K$-class classification problem as a collection of binary detection problems [31]. Each detector is trained to differentiate between the malware belonging to a class $\mathcal{C}_k$ and malware belonging to all other classes $\mathcal{C}_{\bar{k}}$. The detectors model the probability that a process with feature vector $\hat{\mathbf{x}}$ comes from a malware sample of class $\mathcal{C}_k$ as

$$p_{\mathcal{C}_k}(\hat{\mathbf{x}}) = \frac{1}{1+e^{-(\mathbf{w}^T\hat{\mathbf{x}})}}\,, \tag{5}$$

[1]SCS source code: https://github.com/rcanzanese/SystemCallService

and the probability that it comes from some other class as

$$p_{\mathcal{C}_{\bar{k}}}(\hat{\mathbf{x}}) = 1 - p_{\mathcal{C}_k}(\hat{\mathbf{x}})\,. \tag{6}$$

The OVA algorithm selects the most likely class $\hat{\mathcal{C}}_k$ as

$$\hat{\mathcal{C}}_k = \underset{\mathcal{C}_k}{\arg\max}\ \log\left(\frac{p_{\mathcal{C}_k}(\hat{\mathbf{x}})}{p_{\mathcal{C}_{\bar{k}}}(\hat{\mathbf{x}})}\right). \tag{7}$$

The parameter $\mathbf{w}$ in the preceding expressions is determined through minimization of a function that penalizes misclassification and high model complexity,

$$E(\mathbf{w}) = \frac{1}{p}\sum_{i=1}^{p} L(y_i, \mathbf{w}^T\mathbf{x}_i) + \alpha||\mathbf{w}||_2\,. \tag{8}$$

where

$$L(t, y) = \log\left(1 + \exp\left(-ty\right)\right). \tag{9}$$

### B. Naïve Bayes

The naïve Bayes algorithm is a log-likelihood ratio test that assumes conditional independence of the features and bases its detection thresholds on error costs and prior probabilities [32]. Given the prior probabilities $P(\mathcal{C}_k)$ and likelihoods $P(\hat{\mathbf{x}}|\mathcal{C}_k)$, the naïve Bayes classifier computes the most likely class label $\hat{\mathcal{C}}_k$ of a feature vector $\hat{\mathbf{x}}$ as the class with the highest posterior probability,

$$\hat{\mathcal{C}}_k = \underset{\mathcal{C}_k}{\arg\max}\, P(\mathcal{C}_k) \times P(\hat{\mathbf{x}}|\mathcal{C}_k)\,. \tag{10}$$

For this study, the prior probabilities are estimated from the training data. The likelihoods are parameterized using a multinomial distribution model for the TF-IDF data and a Gaussian distribution model for the SVD and LDA data.

### C. Random forests

The random forest algorithm is advantageous for its white box model and its ability to realize complicated decision surfaces. It uses a collection of binary decision trees for classification. A decision tree comprises of a set of nodes and edges. The interior nodes correspond to simple threshold tests. During classification, the tree is traversed by evaluating the threshold tests. The traversal ends at a leaf node, which indicates the predicted malware class $\hat{\mathcal{C}}_k$. The random forest classifier uses a collection of binary decision trees and selects the majority output of the component trees as its output [33]. Random forests are used because decision trees alone tend to overfit the training data.

This study uses the classification and regression trees (CART) algorithm for training [34]. The CART algorithm builds the decision trees recursively, starting at the root node of the tree. It begins by considering all of the malware samples and their corresponding labels and feature vectors $\hat{\mathbf{x}}$. The CART algorithm seeks the feature and threshold test that cause similar malware samples to be grouped together after the split. The algorithm terminates based on tunable parameters that determine the complexity of the tree. The trees in the forest are randomized by considering a randomly selected subset of the available features at each node.

### D. Nearest neighbors

The $k$-nearest neighbors classifier uses the entire training set $\mathcal{X}$ as its model [28]. The classifier identifies the vectors in the training set nearest to a test vector $\hat{\mathbf{x}}$. These vectors are considered its nearest neighbors. The output of the classifier is the class with the highest representation in the set of nearest neighbors. The nearest neighbor classifier is advantageous because of its simplicity and because it can realize complicated decision surfaces.

### E. Nearest centroid

The nearest centroid classifier models each malware class as its centroid, *i.e.*, the average of its feature vectors [35]. The output of the classifier is the class label of the centroid nearest to the vector. The nearest centroid classifier is advantageous because of its lower model and computational complexity compared to the nearest neighbor classifier. The classifier assumes convexity of its classes and equal variance along all dimensions. Therefore, it performs poorly when the data points within the classes do not form non-overlapping convex sets or when the feature variances differ significantly.

## V. Malware classifier evaluation

The effectiveness of the classifiers described in the previous section is measured in terms of precision, recall, and $F_1$ score. These three quantities are computed on a per-class basis. The precision for a class $\mathcal{C}_k$ is the fraction of processes classified as $\mathcal{C}_k$ that belong to $\mathcal{C}_k$,

$$\text{Precision}_{\mathcal{C}_k} = \frac{TP_{\mathcal{C}_k}}{TP_{\mathcal{C}_k} + FP_{\mathcal{C}_k}}\,. \tag{11}$$

$TP_{\mathcal{C}_k}$ and $FP_{\mathcal{C}_k}$ are the numbers of true positives and false positives reported by the classifier, *i.e.*, the number of processes correctly and incorrectly classified as belonging to $\mathcal{C}_k$. The precision measures the relevance of the positive classifications. A precision of 1 indicates that the classifier is always correct when it classifies a process as belonging to class $\mathcal{C}_k$, whereas a precision of 0 indicates it is never correct when it does so.

The recall of a class $\mathcal{C}_k$ is the fraction of the processes belonging to $\mathcal{C}_k$ in the ground truth that are correctly classified,

$$\text{Recall}_{\mathcal{C}_k} = \frac{TP_{\mathcal{C}_k}}{TP_{\mathcal{C}_k} + FN_{\mathcal{C}_k}}\,. \tag{12}$$

$FN_{\mathcal{C}_k}$ is the number of false negatives, *i.e.*, the number of instances of $\mathcal{C}_k$ misclassified as belonging to another class. The recall measures the sensitivity of the classifier. A recall of 1 indicates that a classifier correctly identifies every instance of class $\mathcal{C}_k$, whereas a recall of 0 indicates that a classifier never correctly identifies instances of $\mathcal{C}_k$.

The $F_1$ score of a class $\mathcal{C}_k$ is the harmonic mean of the precision and recall of that class. An $F_1$ score of 0 indicates 0 recall or 0 precision, whereas an $F_1$ score of 1 indicates perfect recall and precision. In this study, per-class $F_1$ scores are averaged over all the classes to provide an overall characterization of a classifier. Three averaging techniques are considered to account for the unbalanced representation of the malware classes used in this study.

Micro-averaged $F_1$ score:
: The $F_1$ score computed from the aggregate set, characterizing classifier performance on large classes.

Macro-averaged $F_1$ score:
: The average of the $F_1$ scores of the classes, characterizing classifier performance on small classes.

Weighted $F_1$ score:
: The weighted average of the $F_1$ scores of the classes, with weights proportional to heir support in the ground truth.

## VI. Experimental results

This section presents experimental results achieved using the previously described feature extraction, classification, and evaluation techniques. The results were obtained through 10-fold cross-validation performed on a set of 125,000 malware processes collected from the execution of more than 76,000 distinct malware samples. To characterize the performance of the classifier against new malware families and variants, cross-validation was performed using disjoint sets of malware samples. The malware samples were collected from honeypots, by crawling blacklists, and from publicly available malware collections. The samples were first seen by VirusTotal[2] between January 2012 and June 2015 and identified as malicious by at least 15 AV vendors. The malware samples were executed on a purpose-built malware testbed which consisted of 19 hosts running 32 and 64-bit versions of Microsoft Windows 7, 8.1, and Server 2012 R2. The specific configurations of the hosts varied (*e.g.*, installed software, level of applied patches). To allow malware to communicate over a network, the testbed included a DNS server and routed outgoing traffic to a Dionaea[3] honeypot. The honeypot provided a vulnerable host for malware to infect and a remote server for malware to send data. Data collected by the honeypot included malware payloads, host information, and private data transmitted by the malware samples executed on the testbed. Although presented here serially, the experimental results were obtained through a process of successive refinement of strategies and parameters.

### A. Ground truth comparison

The lack of a universal malware naming scheme and the incompleteness and inconsistency of existing naming schemes complicate the problem of automatic malware classification [18]. These challenges raise the question: What are the relevant malware classes against which a classifier should be evaluated? This section presents a study of the accuracy of the LR classifier against 27 ground truth labeling schemes derived from 16 AV vendors' labels. The ground truth labeling schemes are listed in Table I, identified by the AV labels from which they are derived and whether they are malware category or family labels. Category labels define the general function or

[2]VirusTotal, http://www.virustotal.com

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

TABLE I
LR Classifier $F_1$ scores for various ground truth labeling systems

| vendor | type | classes | micro | macro | weighted |
|---|---|---|---|---|---|
| AntiVir | category | 17 | 0.79 | 0.45 | 0.79 |
| Microsoft | category | 20 | 0.74 | 0.52 | 0.75 |
| DrWeb | category | 12 | 0.72 | 0.3 | 0.75 |
| Microsoft | family | 315 | 0.7 | 0.53 | 0.71 |
| Vipre | category | 47 | 0.68 | 0.43 | 0.71 |
| ESETNOD32 | family | 301 | 0.67 | 0.5 | 0.68 |
| Panda | category | 19 | 0.65 | 0.43 | 0.68 |
| Avast | category | 12 | 0.64 | 0.42 | 0.66 |
| K7AntiVirus | category | 16 | 0.64 | 0.45 | 0.65 |
| DrWeb | family | 241 | 0.57 | 0.46 | 0.59 |
| Kaspersky | category | 34 | 0.57 | 0.45 | 0.58 |
| Symantec | category | 24 | 0.56 | 0.39 | 0.58 |
| FSecure | category | 49 | 0.54 | 0.42 | 0.57 |
| Vipre | family | 220 | 0.54 | 0.41 | 0.57 |
| GData | category | 30 | 0.55 | 0.28 | 0.57 |
| AntiVir | family | 154 | 0.54 | 0.41 | 0.56 |
| Ikarus | category | 46 | 0.54 | 0.37 | 0.56 |
| McAfee | family | 125 | 0.52 | 0.44 | 0.53 |
| Panda | family | 111 | 0.51 | 0.43 | 0.53 |
| Ikarus | family | 442 | 0.5 | 0.38 | 0.5 |
| Kaspersky | family | 290 | 0.49 | 0.4 | 0.49 |
| FSecure | family | 175 | 0.47 | 0.35 | 0.48 |
| Emsisoft | category | 73 | 0.43 | 0.2 | 0.48 |
| Avast | family | 220 | 0.46 | 0.37 | 0.47 |
| TrendMicro | family | 227 | 0.47 | 0.4 | 0.46 |
| GData | family | 261 | 0.42 | 0.35 | 0.43 |
| Emsisoft | family | 293 | 0.41 | 0.29 | 0.43 |

delivery mechanism of a malware sample and include labels such as Trojan horse, worm, and virus. Family labels define specific function, heritage, or shared authorship and include labels such as Zbot, Kelihos, and MyDoom. For each ground truth labeling scheme, Table I lists the number of distinct classes included in the evaluation and the average $F_1$ scores of the classifier. For each labeling scheme, only malware samples positively identified as malicious by the corresponding AV vendor were considered. Furthermore, only classes containing at least 10 distinct malware samples were considered to ensure meaningful cross-validation results. Samples were considered distinct if their SHA1 hashes differed. The classification results were obtained using TF-IDF feature extraction, system call 3-grams, a trace length of 1500 calls, and the LR classifier.

The results in Table I provide two useful insights. First, the macro-averaged $F_1$ scores are consistently low, indicating the classifiers perform poorly against small classes regardless of the chosen ground truth labels. Second, the micro-averaged and weighted $F_1$ scores vary significantly based on the chosen ground truth labels. This indicates that certain ground truth labeling schemes describe malware based on their system call 3-gram frequencies better than others. The highest performing categorical labels were those derived from the AntiVir and Microsoft labels, while the highest performing family labels were those derived from the ESET and Microsoft labels. These four labeling schemes are the focus of the remainder of the presented results.

TABLE II
CLASSIFIER AND FEATURE EXTRACTION STRATEGY COMPARISON

| detector | feature extraction | $F_1$ (micro) | $F_1$ (macro) | $F_1$ (weighted) |
|---|---|---|---|---|
| LR | TF-IDF | 0.71 | 0.54 | 0.70 |
| LR | TF-IDF, SVD | 0.53 | 0.25 | 0.53 |
| LR | TF-IDF, SVD, LDA | 0.56 | 0.34 | 0.56 |
| nearest centroid | TF-IDF, SVD | 0.15 | 0.07 | 0.19 |
| nearest centroid | TF-IDF, SVD, LDA | 0.39 | 0.20 | 0.42 |
| nearest neighbor | TF-IDF, SVD | 0.69 | 0.44 | 0.67 |
| nearest neighbor | TF-IDF, SVD, LDA | 0.69 | 0.45 | 0.67 |
| random forests | TF-IDF, SVD | 0.69 | 0.49 | 0.67 |
| random forests | TF-IDF, SVD, LDA | 0.69 | 0.47 | 0.67 |
| multinomial naïve Bayes | TF-IDF | 0.41 | 0.05 | 0.33 |
| Gaussian naïve Bayes | TF-IDF, SVD | 0.35 | 0.21 | 0.39 |
| Gaussian naïve Bayes | TF-IDF, SVD, LDA | 0.47 | 0.31 | 0.50 |

## B. *Classifier comparison*

This section compares the accuracy of the five classifiers described in Section IV using three feature extraction strategies.

TF-IDF:
: Uses the TF-IDF feature extraction strategy.

TF-IDF, SVD:
: Combines TF-IDF with SVD, projecting the TF-IDF features onto their first 250 singular values.

TF-IDF, SVD, LDA:
: Combines the previous strategy with LDA, projecting the SVD transformed features onto $K - 1$ features.

To facilitate analysis, each of the transformed feature vectors is scaled to unit magnitude using the $L_2$ norm. Only the LR classifier is evaluated using all three feature extraction strategies. The nearest centroid, nearest neighbor, and decision tree classifiers are not evaluated against the TF-IDF features because of the high dimensionality of the data and the computational complexity of the algorithms. The multinomial and Gaussian naïve Bayes classifiers are restricted only to the feature sets that can be described by their models. Therefore, the former uses only the TF-IDF data and the latter uses only the SVD and LDA transformed feature data.

Table II summarizes the performance of each classifier and feature extraction strategy combination in terms of its average $F_1$ scores. The results were obtained using system call 3-grams and traces of length 1500. The worst performing classifiers were naïve Bayes and nearest centroid. The naïve Bayes classifier performed poorly because the assumed conditional independence and probability models were inaccurate and tended to overfit the training data. The nearest centroid classifier performed poorly because the feature data did not form non-overlapping convex sets. The nearest centroid detector's performance improved (but still under-performed the others) when LDA was used because the transformed data had higher separation between classes.

The nearest neighbors and random forest classifiers provided nearly identical performance, regardless of whether LDA was used. This is largely due to the complex decision surfaces these classifiers are able to realize. The high accuracy of the nearest neighbor classifier underscores the weakness of its counterpart, the nearest centroid classifier. Although the two use similar distance metric-based algorithms, the simplistic model and convexity assumption of the nearest centroid classifier causes its poor performance.

The LR classifier performed similarly to the nearest neighbors and random forest classifiers, providing slightly higher $F_1$ scores. However, the LR performance degraded significantly when the feature reduction techniques SVD and LDA were used. The comparatively poor performance of the LR classifier with feature reduction indicates that the feature data are generally linearly separable in the original feature space, but not so in the reduced feature space.

For a production deployment of the classification system, the LR classifier is preferred to the random forest and nearest neighbor algorithms. The LR classifier provides lower training complexity than the random forest classifier and provides an efficient mechanism for re-training the models as new malware samples are discovered. The LR classifier also provides a lower model storage and testing complexity than the nearest neighbor classifier.

## C. *$n$-gram length*

The results presented in the preceding sections assumed an $n$-gram length of $n = 3$. This section considers the LR classifier and a trace length of 1500 system calls for $n \in \{1, 2, 3, 4, 5\}$. Fig. 1 shows the weighted $F_1$ scores achieved by the classifier for the four highest-performing ground truth labeling systems versus the $n$-gram length. The error bars indicate the standard error of the mean over the 10 cross-validation folds. Classifier accuracy generally improves with increasing $n$. In particular, there is a large increase in performance moving from $n = 1$ to $n = 2$. This indicates that frequency information about system calls (*i.e.*, 1-grams) alone is insufficient for classification. The comparatively high accuracy achieved using the AntiVir labels is largely an artifact of the label distribution: The vast majority of the malware samples belong to a single category. The marginal improvement in performance decreases as $n$-increases, with $n = 5$ providing no significant improvement over $n = 4$. In a production deployment, using the smallest $n$-gram length that

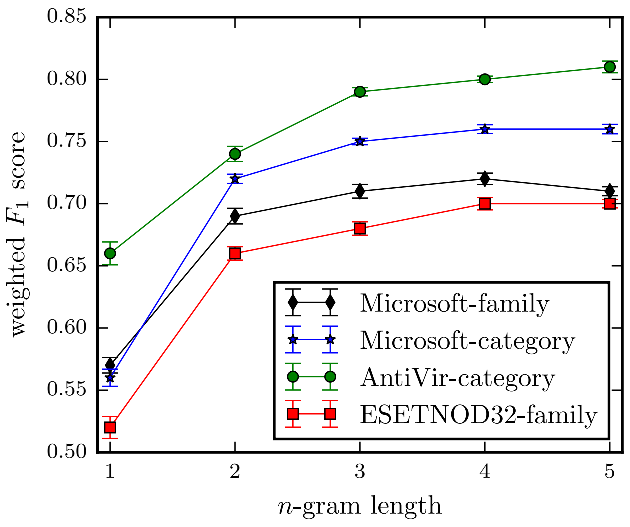

Fig. 1. Weighted $F_1$ score vs. $n$-gram length for LR classifier and trace length of 1500 system calls

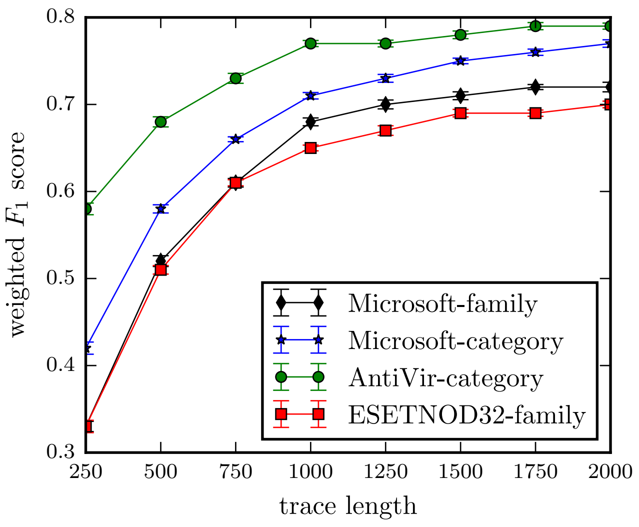

Fig. 2. Weighted $F_1$ score vs. trace length for LR classifier and $n = 3$

provides sufficient classification results is desirable to limit the memory and computational overhead associated with storing and processing the feature vectors.

### *D. Trace length*

This section fixes the $n$-gram length at $n = 3$ and explores the effect of trace length on classification accuracy. The trace length is the number of system calls observed from the beginning of the execution of a process. Fig. 2 shows the weighted $F_1$ score of the LR classifier versus trace length. Performance improves as the trace length increases due to the additional information provided by the longer traces. However, using longer trace lengths is not always desirable or possible. First, a longer trace length requires that the malware execute for longer, causing potentially adverse effects. Second, many of the malware processes considered in this study were short-lived, making analysis at longer trace lengths impossible.

At a trace length of 1500, the average execution time of the studied processes was 205 ms. This result indicates that high classification accuracy was achievable during the initial execution of the malware samples. The behaviors of the malware samples during their initial execution included modification of host configurations to hide the presence of the malware, collection of information about the host, communication with remote servers, creation of new processes, and propagation within a host and over the network. This study primarily considers trace lengths of 1500 because of the diminishing returns in classification accuracy achieved by further increasing the trace length.

### *E. Category-level classification results*

The preceding sections characterized aggregate classifier performance in terms of average $F_1$ scores. This section provides analysis of per-category accuracy of the LR classifier using the Microsoft category labels for $n = 3$ and trace lengths of 1500. The experimental results are presented in terms of the classifier's confusion matrix. The confusion matrix is a $K \times K$ matrix, wherein the rows represent the ground truth labels and the columns represent the classifier outputs. Traditionally, the entries of a confusion matrix are the number of instances of the row class assigned to the column class by the classifier. The diagonal entries indicate correctly classified instances, while the non-diagonal entries indicate incorrectly classified instances. Fig. 3 shows the confusion matrix for the LR classifier using the Microsoft family labels. Since the classes are not balanced, the entries of the matrix display the percentage of the row class assigned to the column class by the classifier. For example, more than 90% of the worms were correctly classified as worms, whereas less than 10% of the spammers were correctly classified spammers. The categories with the highest classification accuracy included backdoors, distributed denial of service (DDoS) tools, dialers, exploits, password stealers (PWS), rogue AV software, viruses, and worms.

The confusion matrix shows two dark vertical lines in the backdoor and Trojan columns, indicating that many malware samples from multiple categories were misclassified as backdoors and Trojans. The backdoor misclassifications were caused primarily by malware samples in other classes providing backdoor-like functionality. Conversely, the Trojan misclassifications were caused primarily by the abundance of Trojans in the training set and the wide variety of functions they exhibited. The Trojan family included many different types of malware grouped together because of their common delivery mechanism, rather than any specific malicious functionality they provided. As such, the Trojans used in this study provided functions that overlapped with the majority of the other malware classes, causing the misclassifications.

The categories with the lowest accuracy included software bundlers, spammers, and monitoring tools. The software

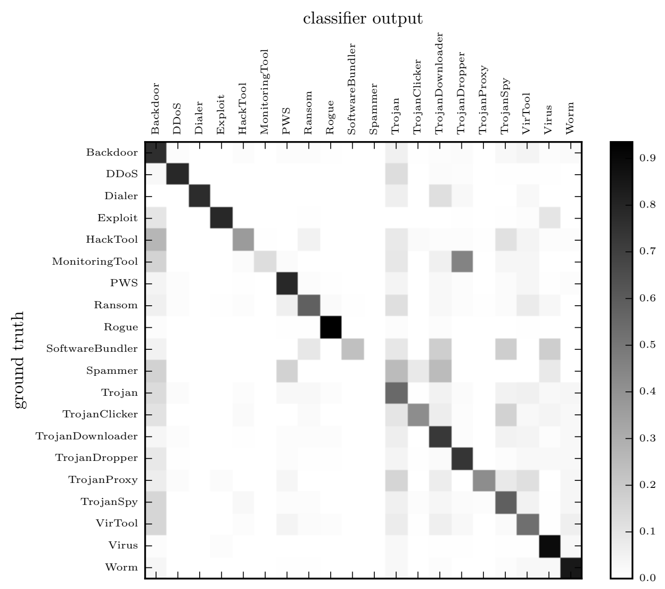

Fig. 3. Classifier confusion matrix for Microsoft category ground truth labels, showing the fraction of samples in each class indicated by the row labels classified as the column labels

bundler samples were most often misclassified as various types of Trojans. This is likely due to the similarity between Trojans and software bundlers, which are legitimate software that also install malware. The misclassifications can also be attributed to the fact that Trojans vastly outnumbered software bundlers in the malware set. Spammers are used primarily to send unwanted email messages. Here, their relatively poor performance is caused by their low representation and the fact that many other malware samples provided spammer-like functions. Monitoring tools are commercial programs that monitor computer usage, and can include keyloggers and password stealers. Monitoring tools are similar to Trojans and Trojan Droppers, because they typically are bundled with and installed alongside other software, and are similar to spyware because they transmit collected data over the network. Overall, the results presented in Fig. 3 indicate that overlapping malware functionality and broadly-defined malware families are the primary cause of category-level misclassifications.

### *F. Family-level classification results*

This section characterizes the performance of the LR classifier using the Microsoft family labels. Here, the experimental results are presented in terms of the classification performance achieved for each of the individual malware categories in the ground truth. The results presented in Table III show the precision, recall, $F_1$ score, and number of instances of the 20 malware families with the highest and lowest $F_1$ scores. Among the best-performing families are those with thousands of instances, whereas the poorest performers have at most 68 instances. These results are consistent with those presented in Section VI-B, which indicated that the classifier performs better for classes with many instances.

TABLE III
PER-FAMILY CLASSIFIER PRECISION, RECALL, AND $F_1$ SCORES FOR THE MALWARE FAMILIES WITH THE HIGHEST AND LOWEST $F_1$ SCORES

| Family | Precision | Recall | $F_1$ score | Instances |
|---|---|---|---|---|
| Trojan.Jeefo | 1.00 | 1.00 | 1.00 | 14 |
| Trojan.Recal | 1.00 | 1.00 | 1.00 | 280 |
| TrojanDownloader.Drstwex | 1.00 | 1.00 | 1.00 | 31 |
| TrojanDropper.Loring | 0.99 | 1.00 | 1.00 | 1028 |
| Virus.Elkern | 1.00 | 1.00 | 1.00 | 18 |
| Virus.Nabucur | 1.00 | 1.00 | 1.00 | 1061 |
| Worm.Fesber | 0.99 | 1.00 | 1.00 | 575 |
| Worm.Klez | 1.00 | 0.99 | 1.00 | 148 |
| Backdoor.Wabot | 0.98 | 1.00 | 0.99 | 272 |
| TrojanDownloader.Ogimant | 0.99 | 0.99 | 0.99 | 1070 |
| Worm.Benjamin | 0.97 | 1.00 | 0.99 | 33 |
| Worm.Mydoom | 0.99 | 1.00 | 0.99 | 1685 |
| Backdoor.Ppdoor | 0.96 | 1.00 | 0.98 | 23 |
| Trojan.Phishbank | 0.98 | 0.99 | 0.98 | 212 |
| TrojanDownloader.Seimon | 0.95 | 1.00 | 0.98 | 79 |
| Virus.Madang | 0.97 | 0.99 | 0.98 | 244 |
| Worm.Soltern | 0.96 | 1.00 | 0.98 | 44 |
| Backdoor.Berbew | 0.97 | 0.97 | 0.97 | 33 |
| Worm.Dumpy | 0.94 | 1.00 | 0.97 | 16 |
| Backdoor.Caphaw | 0.96 | 0.96 | 0.96 | 27 |
| ... | | | | |
| Trojan.Meredrop | 0.02 | 0.03 | 0.02 | 68 |
| TrojanSpy.Crime | 0.02 | 0.03 | 0.02 | 38 |
| HackTool.Keygen | 0.00 | 0.00 | 0.00 | 19 |
| Trojan.Alureon | 0.00 | 0.00 | 0.00 | 11 |
| Trojan.Danglo!gmb | 0.00 | 0.00 | 0.00 | 21 |
| Trojan.Ircbrute!gmb | 0.00 | 0.00 | 0.00 | 22 |
| Trojan.Msposer | 0.00 | 0.00 | 0.00 | 54 |
| Trojan.Orsam!rts | 0.00 | 0.00 | 0.00 | 35 |
| Trojan.Rimod | 0.00 | 0.00 | 0.00 | 16 |
| Trojan.Sisproc | 0.00 | 0.00 | 0.00 | 41 |
| Trojan.Sisron!gmb | 0.00 | 0.00 | 0.00 | 44 |
| Trojan.Trafog!rts | 0.00 | 0.00 | 0.00 | 15 |
| Trojan.Yakad | 0.00 | 0.00 | 0.00 | 20 |
| TrojanDownloader.Delf | 0.00 | 0.00 | 0.00 | 15 |
| TrojanDownloader.Ranos | 0.00 | 0.00 | 0.00 | 26 |
| TrojanProxy.Banker | 0.00 | 0.00 | 0.00 | 13 |
| TrojanSpy.Delf | 0.00 | 0.00 | 0.00 | 16 |
| VirTool.Vtub | 0.00 | 0.00 | 0.00 | 36 |
| Worm.Jenxcus | 0.00 | 0.00 | 0.00 | 21 |
| Worm.Sohanad | 0.00 | 0.00 | 0.00 | 11 |

Generally, the best performing families were those from narrowly defined malware categories, such as viruses, worms, and backdoors. Conversely, the worst performing families were those from more broadly defined categories, particularly Trojans. Furthermore, some of the poorly performing families are also broadly defined. For example, Gandlo!gmb, Ircbrute!gmb, and Sisron!gmb are all generically-defined Trojans. In contrast, the highest performing families are typically very narrowly defined. For example, Klez and MyDoom, are well studied families whose samples perform specific functions and have a shared heritage.

One way to account for varying class sizes when training a classifier is to apply weights to the classes inversely based on their representation in the training set. Rare classes are oversampled to even the class distributions and ensure that the classifier is not biased toward classes with higher representation. For comparison, the weighted LR classifier was compared to the unweighted LR classifier. The two versions provided nearly identical results, indicating that class size was not inherently biasing the classifier. Rather, the low precision and recall of the worst-performing classes in Table III were likely caused by broadly defined malware families and a lack of enough samples to accurately characterize the families.

## VII. Discussions and Conclusions

This study explored the use of multiple classification algorithms, feature extraction strategies, and ground truth labeling schemes for malware classification. Through experimental evaluation, it demonstrated that decision tree, nearest neighbor, and LR classifiers outperformed naïve Bayes and nearest centroid classifiers. Since the goal of this study was to identify those techniques best suited for production deployment, the LR classifier was selected for its comparatively low model and computational complexities. The ESET and Microsoft family labels were selected because they afforded the highest classification accuracy. The classifier performed best against well-defined malware families and worst against broadly defined malware families and categories. The described malware classification system is intended for production systems, where system call trace data collected from executing processes can be used to classify newly discovered malware according to their behavioral similarity to known malware.

## Acknowledgments

Thank you to the KEYSPOT Network, the People's Emergency Center, the Dornsife Center for Neighborhood Partnerships, the City of Philadelphia's Mayor's Commission on Literacy, Office of Innovation and Technology (OIT), and Department of Parks and Recreation (PPR) for their support of this research. This research is sponsored by a Secure and Trustworthy Cyberspace (SaTC) award from the National Science Foundation (NSF), under grant CNS-1228847, and the Isaac L. Auerbach endowed chair for Spiros Mancoridis.

## References

[1] U. Bayer, C. Kruegel, and E. Kirda, "Ttanalyze: A tool for analyzing malware," *EICAR*, 2006.

[2] C. Willems, T. Holz, and F. Freiling, "Toward automated dynamic malware analysis using cwsandbox," *IEEE Security Privacy*, 2007.

[3] C. Castillo, A. Hinchliffe, C. Miller, R. N. KP, F. Paget, E. Peterson, A. Pradeep, C. Schmugar, R. Simon, D. Sommer, B. Sun, and A. Wosotowsky, "Mcafee labs threats report," Intel Security, Tech. Rep., 2015.

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

[5] G. Jacob, H. Debar, and E. Filiol, "Behavioral detection of malware: from a survey towards an established taxonomy," *Journal in Computer Virology*, 2008.

[6] M. Christodorescu and S. Jha, "Testing malware detectors," *SIGSOFT Software Engineering Notes*, 2004.

[7] R. Canzanese, S. Mancoridis, and M. Kam, "System call-based detection of malicious processes," in *International conference on quality, reliability, and security (QRS)*, 2015.

[8] J. Z. Kolter and M. A. Maloof, "Learning to detect and classify malicious executables in the wild," *Journal of Machine Learning Research*, 2006.

[9] L. Nataraj, V. Yegneswaran, P. Porras, and J. Zhang, "A comparative assessment of malware classification using binary texture analysis and dynamic analysis," in *Workshop on security and artificial intelligence*, ser. AISec, 2011.

[10] Y. Zhong, H. Yamaki, and H. Takakura, "A malware classification method based on similarity of function structure," in *International Symposium on Applications and the Internet*, ser. SAINT, 2012.

[11] K. Iwamoto and K. Wasaki, "Malware classification based on extracted api sequences using static analysis," in *Asian Internet Engineeering Conference*, ser. AINTEC, 2012.

[12] S. Cesare, Y. Xiang, and W. Zhou, "Malwise: An effective and efficient classification system for packed and polymorphic malware," *IEEE Tran. Computers*, 2013.

[13] Y. Park, D. Reeves, V. Mulukutla, and B. Sundaravel, "Fast malware classification by automated behavioral graph matching," in *Annual Workshop on Cyber Security and Information Intelligence Research*, ser. CSIIRW, 2010.

[14] Y. Park and D. Reeves, "Deriving common malware behavior through graph clustering," in *Symposium on Information, Computer and Communications Security*, ser. ASIACCS, 2011.

[15] 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 *International Conference on Computational Intelligence and Security*, ser. CIS, 2011.

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

[17] M. Apel, C. Bockermann, and M. Meier, "Measuring similarity of malware behavior," in *Conference on Local Computer Networks (LCN)*, 2009.

[18] M. Bailey, J. Oberheide, J. Andersen, Z. M. Mao, F. Jahanian, and J. Nazario, "Automated classification and analysis of internet malware," in *International Symposium on Recent Advances in Intrusion Detection*, ser. RAID, 2007.

[19] K. Rieck, T. Holz, C. Willems, P. Düssel, and P. Laskov, "Learning and classification of malware behavior," in *International Conference on Detection of Intrusions and Malware, & Vulnerability Assessment*, ser. DIMVA, 2008.

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

[21] T. Lee and J. J. Mody, "Behavioral classification," in *European Institute for Computer Antivirus Research Annual Conference*, ser. EICAR, 2006.

[22] M. Neugschwandtner, P. M. Comparetti, G. Jacob, and C. Kruegel, "Forecast: skimming off the malware cream," in *Annual Computer Security Applications Conference*, ser. ACSAC, 2011.

[23] B. Anderson, C. Storlie, and T. Lane, "Improving malware classification: bridging the static/dynamic gap," in *Workshop on Security and artificial intelligence*, ser. AISec, 2012.

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

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

[26] D. Lobo, P. Watters, and X. Wu, "Rbacs: Rootkit behavioral analysis and classification system," in *Third International Conference on Knowledge Discovery and Data Mining*, 2010.

[27] Y. Liao and V. R. Vemuri, "Using text categorization techniques for intrusion detection." in *USENIX Security Symposium*, 2002.

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

[29] C. D. Manning, P. Raghavan, and H. Schütze, *Introduction to Information Retrieval*. Cambridge University Press, 2008.

[30] A. Genkin, D. D. Lewis, and D. Madigan, "Large-scale bayesian logistic regression for text categorization," *Technometrics*, 2007.

[31] R. Rifkin and A. Klautau, "In defense of one-vs-all classification," *Journal of Machine Learning Research*, 2004.

[32] H. L. VanTrees, *Detection, Estimation, and Modulation Theory*. Wiley-Interscience, 2001.

[33] L. Breiman, "Random forests," *Machine Learning*, 2001.

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

[35] E.-H. S. Han and G. Karypis, "Centroid-based document classification: Analysis and experimental results," 2000.