Combining Supervised and Unsupervised Learning for Zero-Day Malware Detection

Prakash Mandayam Comar[+], Lei Liu[+], Sabyasachi Saha[*], Pang-Ning Tan[+], Antonio Nucci[*]
[+]Dept. of Computer Science, Michigan State University, East Lansing, MI 48824
[*]Narus Inc. 570 Maude Court, Sunnyvale, CA 94085
Email: [+]{mandayam, liulei1, ptan}@msu.edu, [*]{ssaha, anucci}@narus.com

***Abstract*—Malware is one of the most damaging security threats facing the Internet today. Despite the burgeoning literature, accurate detection of malware remains an elusive and challenging endeavor due to the increasing usage of payload encryption and sophisticated obfuscation methods. Also, the large variety of malware classes coupled with their rapid proliferation and polymorphic capabilities and imperfections of real-world data (noise, missing values, etc) continue to hinder the use of more sophisticated detection algorithms. This paper presents a novel machine learning based framework to detect known and newly emerging malware at a high precision using layer 3 and layer 4 network traffic features. The framework leverages the accuracy of supervised classification in detecting known classes with the adaptability of unsupervised learning in detecting new classes. It also introduces a tree-based feature transformation to overcome issues due to imperfections of the data and to construct more informative features for the malware detection task. We demonstrate the effectiveness of the framework using real network data from a large Internet service provider.**

I. Introduction

Malware is a malicious software program deployed by attackers to compromise a computer system (including its applications, data, and network infrastructure) by exploiting its security vulnerabilities. Such programs have potential to disrupt the normal operations of the system, while giving the attackers unauthorized access to the system resources and allowing them to gather information covertly from users without their consent. Malware attacks have escalated in recent years with a profit motive in mind. These attacks, which include botnets, spyware, keystroke loggers, and trojans, have been used to commit identity theft, spamming, and phishing scams. Due to their rapid proliferation and increasing sophistication, the need for more advanced techniques to effectively detect malware attacks has become increasingly critical.

Known techniques for malware detection can be broadly classified into two categories, namely, signature-based and anomaly-based detection techniques [1][2]. Signature based detection employs a set of pre-defined signatures for known malware to determine whether the software program under inspection is malicious. For example, Sung et al. [3] created a signature based detection system called SAVE (Static Analyzer of Vicious Executables) that compares API sequences extracted from a program to those stored in a signature database. Programs whose signature matches those found in the database will be flagged as malware. The signatures for malware are often created manually, though automated methods based on n-gram extraction [4][5] have also been developed. A major limitation of the signature based approach is its failure to detect zero-day attacks, which are emerging threats previously unknown to the malware detector system. Furthermore, it fails to detect threats with evolving capabilities such as metamorphic and polymorphic malwares. Metamorphic malwares automatically reprogram themselves every time they are distributed or propagated. So, they will be difficult to capture with signature-based approach as their signatures have to be continuously evolved [6]. Similarly, polymorphic malwares are also difficult to identify as they self-mutate and use encryption to avoid detection. Methods that rely on inspection of payload signatures can be easily defeated by encryption or obfuscation techniques.

A related approach to signature-based detection is supervised classification, which uses instances of known malware to build a classification model that distinguishes the known threats from other programs. Supervised classification approaches also share the same limitation as signature-based detection in that they both perform poorly on new and evolving malware. In addition, building an accurate classification model is a challenge due to the diversity of malware classes, their imbalanced distribution, as well as the data imperfection issues (noise, missing values, and correlated features) that continue to hinder the adoption of more sophisticated learning algorithms.

On the other hand, anomaly-based detection techniques use the knowledge of what is considered as good behavior to identify malicious programs [7][8][9]. A special class of anomaly based detection is specification-based detection, which makes use of certain rule set of what is considered acceptable behavior of programs. Programs violating such rule set are declared as malicious. Statistical and unsupervised learning techniques for anomaly detection have also been extensively investigated in recent years. The key advantage of anomaly-based detection is its ability to detect zero-day attacks [10]. However, it is susceptible to producing a high false alarm rate, which means a large number of good programs will be wrongly identified as malicious.

Our goal is to detect and isolate malicious flows from the network traffic and further classify them as a specific type of the known malwares, variations of the known malwares or as a completely new malware strain. Towards this end, we develop a novel machine-learning based malware detection and classification system using the layer-3 and layer-4 network

978-1-4673-5946-7/13/$31.00 ©2013 IEEE

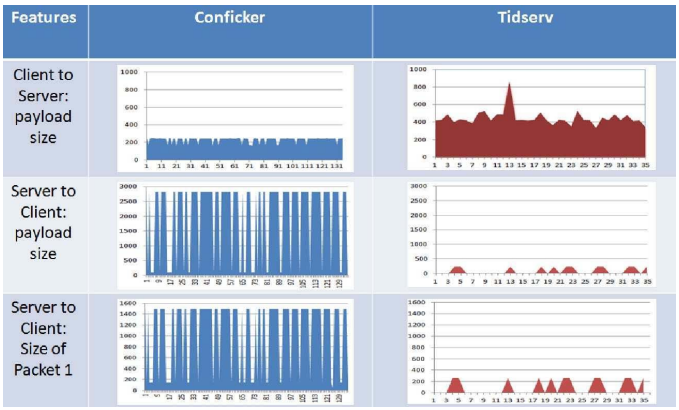

Fig. 1. Comparison of 3 network traffic features from flows associated with the Conficker and Tidserv malwares.

traffic features such as *bytes per second*, *packets per flow*, and *packet inter-arrival times* (see Table I). Our approach is motivated by the observation that threats often exhibit specific behavior in terms of their layer-3/layer-4 statistical flow-level features and these features retain their properties even when the payload is encrypted. Figure 1 demonstrated an example of the behavior for two malwares, Conficker and Tidserv. For Conficker, the payload size of client-to-server communication is either 244 or 169 bytes, whereas for Tidserv it varies widely. However, if we consider the server-to-client communication, Conficker flows always have payload size 2817 or 109 bytes, whereas the payload size for Tidserv is always 0 or 244 bytes. The regularity observed in the layer-3/layer-4 features among the malicious flows suggests the possibility of using such features to effectively detect the known malware classes. This begs the question whether such features can also be effectively used to detect variants of the known malwares and new malwares. This would require sophisticated classifiers that can generalize beyond the classes observed in its training data. Feature selection [11] would also be an important part since the raw features may not be sufficient to detect such malwares.

Our proposed framework leverages the accuracy of supervised classification on known classes with the adaptability of unsupervised learning for new malware detection. Specifically, a two-level classifier is employed to address the issue of imbalanced class distribution due to the disproportionate number of non-malicious and malicious network flows. A macro-level classifier is initially used to isolate the malicious flows from non-malicious ones. Next, among those flows classified as malicious, micro-level classifiers based on one-class support vector machine (SVM) are applied to identify the specific strain of malware. To extend their ability to detect malware classes beyond those in the training set, a probabilistic class-based profiling method is developed. Using this "unsupervised" learning approach, we demonstrate the effectiveness of our framework in determining whether a malicious flow corresponds to a variant of the known malware or a new strain of malware. Furthermore, a tree-based feature transformation is performed to alleviate the data imperfection issues and construct more informative, non-linear features for the accurate detection of various malware classes. The one-class SVM classifiers are constructed using a weighted linear kernel trained on the tree-based features.

In short, the main contributions of our work are as follows:

1) We propose an effective malware detection framework that learns intrinsic statistical flow-level behavioral signature of each malware, unlike most deployed solutions that rely heavily on payload-based or deep packet inspection (DPI) techniques, and thus, fail to classify encrypted traffic while raising concerns about user privacy.
2) We design a tree-based kernel for one-class SVM classifiers to handle the data imperfection issues that arise in the network flow data.
3) We present a two-level malware detection framework that is capable of discriminating new malware from existing ones by applying a class-based profiling approach.

II. Challenges and Motivation

The key challenges in detecting and classifying malware from network flow information include:

- *Imbalanced class representation:* Majority of the flows belong to a few of the most dominant classes. Most classifiers are therefore biased toward detecting the larger classes while completely missing the rare ones.
- *New class discovery:* Not all classes are present at the time the classifier is initially trained. This makes it hard for conventional classifiers to detect new classes.
- *Missing values:* The features used to characterize the network flows may contain missing values. Though numerous strategies for handling missing values are available, our experiments suggest that they are ineffective when applied to the data collected in this domain.
- *Noise in the training data:* The training examples are often labeled using a signature-based intrusion detection system (IDS). Since the signatures do not provide a complete coverage for all malwares, training instances labeled as good are unreliable as they may include malwares whose signatures have not been created yet.

Due to the aforementioned challenges, existing techniques are insufficient to effectively classify both the known and new malwares. This paper seeks to develop a viable solution to overcome these challenges.

III. Proposed Framework

The proposed framework is envisioned as a key component of a larger end-to-end security system that monitors the network flow and deciding whether it is *malicious* or *good* (more precisely, unknown, because they may correspond to malicious flows that do not have a detection signature yet). Figure 2 shows the proposed system architecture, which consists of the following six major components: (i) data capture module, (ii) an intrusion detection/prevention system (IDS/IPS), (iii) information storage, (iv) feature extraction and transformation, (v) supervised classifier, and (vi) a UI portal.

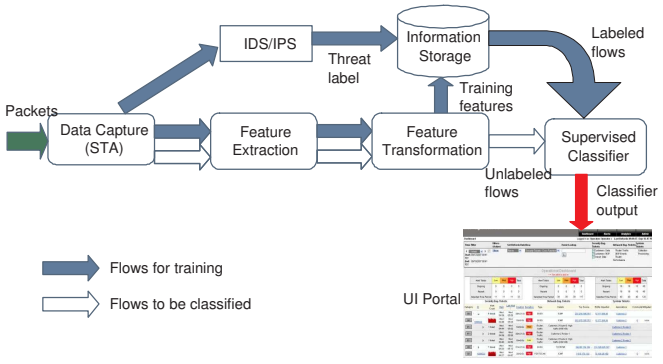

Fig. 2. High level system architecture

The data capture module is a device such as Narus Semantic Traffic Analyzer (STA) which parses packets and collates packets belonging to the same flow. This module is responsible for generating all the flow-level features associated with this flow. The IDS/IPS module performs deep packet inspection and tags the flow whether it belongs to some threat. Otherwise, the flow is labeled as "good/unknown". The *information storage* component stores all the flow features and their associated class labels. The *feature extraction* module extracts statistical features on a per-flow basis while the *feature transformation* module converts them into more robust features that will be used to build classifiers for detecting malicious flows. The classifiers are constructed in an offline fashion and are deployed to incoming network flows. The UI portal is used for reporting the emergence of new *suspicious* flows.

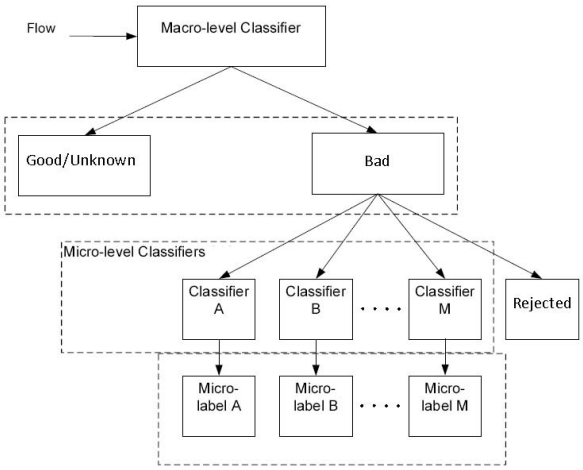

Fig. 3. A schematic illustration of the proposed two-level malware detection framework

Figure 3 illustrates the proposed two-stage classification framework. A macro-level binary classifier is initially used to isolate malicious flows from the non-malicious ones. Next, a multi-class classification technique is applied to further categorize the malicious flows into one of the pre-existing malware classes or as a new malware. The rationale for adopting the two-stage classification framework is as follows.

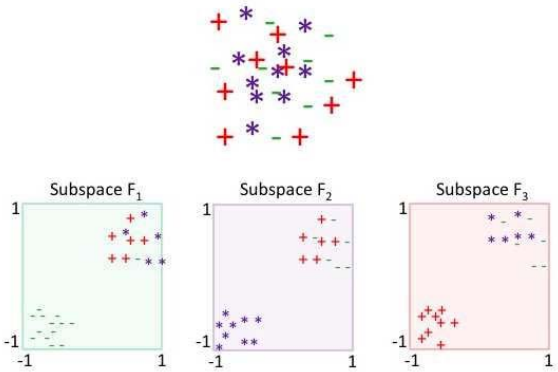

Fig. 4. Example of feature transformation for a 3-class problem. A subspace is generated for each class using an ensemble of tree-based classifiers.

First, as the flows predominantly belong to the "good" class, it would be difficult to build a global model that effectively discriminates all the classes. In fact, the global model is likely to have a strong bias towards predicting most flows as "good", resulting in a high false negative rate for the malware classes. By combining the malicious flows into a single class, this reduces the problem into a binary classification task, where the proportion of the non-malicious to malicious flows is less skewed, thereby helping to reduce the false negative rate for the malware class. Second, to identify the malware type, a large number of classifiers, each designed for a specific type of malware, must be applied. Given that millions of packets may flow through a router every hour, it is necessary to downscale this traffic into smaller processable chunks before applying a large number of sophisticated classifiers to determine the type of malware.

The framework utilizes a collection of 1-class SVM models as its micro-level classifiers, where each model specializes in categorizing instances from a specific malware class. The micro-level classifiers also have the ability to distinguish new malware from those that are already known. It accomplishes this by comparing the distribution of prediction outputs generated by instances associated with the suspected new malware against those generated by the known classes. For example, suppose there are 100 micro-level classifiers. The prediction outputs of the 100 classifiers are expected to be different for each class. Therefore, even if a test instance that belongs to a new class is misclassified as one of the known classes, we expect the prediction outputs generated by other micro-level classifiers in the ensemble will look different than those generated for the class it was confused with. Our framework is therefore designed to exploit this property to facilitate new class discovery.

In addition to the two-stage classification framework, it also uses tree-based features to compute the kernel matrix for one-class SVM in order to deal with the various data quality issues. The construction of such features will be explained in the next section.

A. Tree-based Feature Transformation

In this section, we propose a novel approach that cleverly adapts the decision trees to transform the original data into more informative features. Let $\mathcal{D} = \{(x_1, y_1), (x_2, y_2), \cdots, (x_n, y_n)\}$ be the network flow data, where $y_i \in \{1, 2, \cdots, k\}$ denote the class label chosen from a set of k classes (including the "good" class) and $x_i \in \mathcal{X}$ is the set of features extracted from the layer-3 and layer-4 protocol information of the network flow. One of the key challenges in developing an effective algorithm for classifying the network flows is the imperfections of the data. As shown in Table I, some of the network traffic features are strongly correlated with each other. Furthermore, a flow with small number of packets contains many missing values (e.g., size of packet #10) due to the non-applicability of the feature to the flow. The easiest way to deal with non-applicable features is to eliminate the data instances having at least one such feature. However, this reduces the amount of training data available for constructing a reliable classifier. In fact, if we had discarded instances with inapplicable features from the ISP data used in our experiment, we would be left with only 7% of the original flows. Furthermore, sophisticated classifiers such as 1-class SVM employ a kernel matrix, whose computation assumes no missing values in the feature vector.

The goal of our feature transformation step is to embed the data in a Hilbert space that allows computation of a kernel matrix. Secondly, the data should be transformed in such a way that instances belonging to different classes are projected to different regions of the transformed space. To achieve these goals, we construct a set of decision trees and use the predictions made by each tree as a non-linearly transformed feature (see Algorithm 1). An advantage of using a tree-based classifier is that it handles missing values due to non-applicable features naturally by treating "inapplicable" as a distinct value that does not exist within the domain of possible values (instead of artificially imputing the missing values with the mean values for all flows).

For each class c, we build p trees by labeling the instances belonging to class c as $+1$ and instances from other classes as -1. The predictions made by each tree becomes a new feature, which has a higher predictive power because it can discriminate instances belonging to a class c from the rest of the data. To avoid generating the same tree (feature), we introduce randomness into the tree growing procedure by bootstrapping the instances and subsampling their features instead of using the whole training set. This also helps to reduce the correlations among the new features. Let $\mathcal{F}_c$ be the set of p tree-based features generated to separate class c from rest of data.

$$\mathcal{F}_c = \{f_1^c, f_2^c, \ldots, f_p^c\} \tag{1}$$

where the value for each feature f_i^c is obtained by applying the i-th tree induced from class c (denoted as h_c^i) to the original features of a data instance. The trees are then applied to all the instances in the training and test sets to transform them to a new representation that contains $k \times p$ nonlinear features. The full set of transformed features is denoted by

$$\mathcal{F} = \cup_{c=1}^{k} \mathcal{F}_c = \cup_{c=1}^{k} \cup_{i=1}^{p} h_c^i$$

where each $\mathcal{F}_c$ represents a subspace designed to effectively identify instances from class c. In the next section, we illustrate how to build the 1-class SVM micro-classifiers and their corresponding kernel matrices from the transformed features.

Algorithm 1 Construction of Tree-Based Features

Input: $\mathcal{D} = \{(x_1, y_1), (x_2, y_2), ...(x_n, y_n)\} \in \mathcal{X} \times \mathcal{Y}$
p - number of features per class
Output: $\{F_c\}_1^k$: Collection of $n \times p$ transformed data matrix where each F_c contains p features that separate class c from rest of the data.

for $c = 1$ to k **do**
  Initialize: F_c = $[0]_{n \times p}$
  $D_+ \leftarrow \{(x_i, +1) | (x_i, y_i = c) \in \mathcal{D}\}$
  $D_- \leftarrow \{(x_i, -1) | (x_i, y_i \neq c) \in \mathcal{D}\}$
  Let $D_c = D_+ \cup D_-$.
  for $i = 1$ to p **do**
    $\mathcal{S} \leftarrow$ Sample m instances and f features from D_c.
    Build a regression tree on the sample $\mathcal{S}$.
      $h_c^i : \mathcal{S} \rightarrow [-1, +1]$
    Apply the tree h_c^i on $\mathcal{X}$ and store
      $F_c(:, i) = h_c^i[\mathcal{X}]$
`return` $\{F_c\}_1^k$

B. 1-Class SVM for Known Malware Detection

This paper employs the hypersphere-based 1-class SVM method described in [12] as our micro-classifiers. Specifically, training instances from a given class are encapsulated in a hypersphere constructed in an appropriate feature space. Once the enclosing hyperspheres are constructed for all the classes, the test instances are classified by considering their distance to the center of each hypersphere (the prediction step using multiple hyperspheres is described in Section III-C).

Multiclass learning with one-class SVM has many desirable properties. Firstly, each classification model is a simple hypersphere defined by its center and radius. Thus, the space required to store the models is linear in number of classes. Secondly, the proposed approach mimics the functionality of the nearest neighbor classifier with reduced number of distance comparisons needed, as one have to compare each test instance against the centers of every hypersphere instead of against every training instance. Thirdly, the approach incorporates the rigor of sophisticated discriminative classifiers as regions belonging to different classes are neatly enclosed in hyperspheres with minimal overlap with other spheres.

The hypersphere for class i is constructed by first labeling the instances belonging to class i as $+1$ and those that belong to other classes as -1. We then construct two concentric hyperspheres such that the inner sphere encloses as many

instances from class i as possible. The outer sphere is constructed in such a way that instances that do not belong to class i lie outside of it. The radial distance between the two hyperspheres is called the classifier's margin, which defines the objective function to be optimized by the 1-class SVM learning algorithm. For example, the hypersphere for predicting class k is obtained by solving the following [12]:

$$\begin{aligned}
\min_{R_k,a_k,d_k,\xi_i,\xi_l} \quad & R_k^2 - Md_k^2 + \frac{C}{N_k}\sum_{i:y_i=k}\xi_i + \frac{C}{N_{\bar{k}}}\sum_{l:y_l\neq k}\xi_l \\
\text{subject to} \quad & \parallel x_i - a_k \parallel^2 \leq R_k^2 + \xi_i, \quad \forall i: \ y_i = k \\
& \parallel x_l - a_k \parallel^2 \geq R_k^2 + d_k^2 - \xi_l, \quad \forall l: \ y_l \neq k \\
& \xi_i \geq 0, \quad \forall i: \ y_i = k \\
& \xi_l \geq 0, \quad \forall l: \ y_l \neq k
\end{aligned} \tag{2}$$

where N_k and $N_{\bar{k}}$ are the number of instances that belong to the k^{th} class and its complement, respectively. Here, the positive examples are enumerated by indices i and the negative examples by indices l. ξ_i and ξ_l are the slack variables while $C \geq 0$ controls the penalty for misclassification errors. The variables a_k and R_k represent the center and radius of the hypersphere constructed for the k^{th} class while $\sqrt{(R_k^2 + d_k^2)}$ defines the radius of the outer hypersphere. It can shown that the margin width is given by $\sqrt{(R_k^2 + d_k^2)} - R_k$. To maximize the margin, we need to maximize d_k^2 and minimize R_k^2 simultaneously, and the parameter $M \geq 0$ controls the trade-off between those two terms. The Wolfe dual form of the preceding optimization problem is given below:

$$\begin{aligned}
\max_\alpha \quad & \sum_{i:y_i=k}\alpha_i K(x_i,x_i) - \sum_{l:y_l\neq k}\alpha_l K(x_l,x_l) \\
& - \sum_{i,j;y_i,y_j=k}\alpha_i\alpha_j K(x_i,x_j) \\
& - \sum_{l,m:y_l,y_m\neq k}\alpha_l\alpha_m K(x_l,x_m) \\
& + 2\sum_{i,l:y_i=k,y_l\neq k}\alpha_i\alpha_l K(x_i,x_l) \\
\text{subject to} \quad & \sum_{i:y_i=k}\alpha_i = 1 + M, \quad 0 \leq \alpha_i \leq \frac{C}{N_k} \\
& \sum_{l:y_l\neq k}\alpha_l = M, \quad 0 \leq \alpha_l \leq \frac{C}{N_{\bar{k}}}
\end{aligned} \tag{3}$$

where $K(x, x')$ is a kernel function that measures the similarity between the pair of instances, x and x'. Among the widely used kernel functions include

$$\begin{aligned}
\text{RBF Kernel:} \quad & K(x,x') = \exp\left[-\frac{\parallel x - x' \parallel^2}{2\sigma^2}\right] \\
\text{Polynomial Kernel:} \quad & K(x,x') = (1 + x^T x')^q
\end{aligned}$$

The optimization problem given in (3) can be solved using quadratic programming. One problem with using standard kernel function such as RBF and polynomial is that they assume the feature vectors x do not contain any missing values. In addition, the features are uncorrelated and are equally important when determining the similarity between instances. To circumvent this limitation, our framework uses the tree-based features described in the previous section to learn a weighted kernel matrix for building the hyperspheres. The kernel matrix is induced from a ground truth kernel G, which is defined as follows:

$$G(x_i, x_j) = \begin{cases} +1 & \text{if } y_i = y_j, \\ -1 & \text{otherwise} \end{cases}$$

Intuitively, the goal here is to align the weighted linear kernel $K(x, x') = x^T W x'$ such that it is consistent with the class similarity matrix between training instances G. This is accomplished by minimizing the following objective function:

$$\mathcal{L} = -\sum_{ij} G_{ij}(XWX^T)_{ij} + \frac{\lambda}{2}W_{ij}^2 \tag{4}$$

The first term is minimized when the matrices G and XWX^T are in agreement with each other. The second term W_{ij}^2 is a regularizer term added to keep the model parsimonious[1]. We solve for W by taking the partial derivative of the objective function with respect to W and equating it to zero.

$$\frac{\partial\mathcal{L}}{W_{pq}} = -\sum_{ij} G_{ij}(X_{ip}X_{qj}^T) + \lambda W_{pq} = 0 \tag{5}$$

After rearranging the equation, we have $W = X^T G X$. Thus the similarity between any test instance x^* to a training instance x is given by the following weighted linear kernel

$$\text{WL Kernel}: \qquad K(x, x^*) = \text{sign}(x^T W x^*)$$

The weighted linear kernel will be used to learn the corresponding hyperspheres for each 1-class SVM micro-classifier. The next section explains how the hyperspheres are collectively used to determine whether an incoming flow is a known malware, a variant of the known malware, or a completely new malware strain.

C. Probabilistic Class-based Profiling for New Malware Discovery

The preceding 1-class SVM formulation constructs hyperspheres based on the classes observed in the training set. Such an approach would fail to detect new variants of malware due to the absence of instances from the new malware class in the training set. This section describes our enhancement to the formulation to enable the discovery of new classes. Specifically, new malware will be detected in two ways.

First, consider a new malware whose flow-based characteristics are significantly different than those of the existing malware. In order to detect such malware, we first compute the distance of a test instance to the centers of each hypersphere. If the test instance lies outside the hyperspheres for all the classes, it is predicted to be a new class.

Second, consider a new malware that is a variant of existing ones, cleverly manipulated to avoid detection. Due to the

[1]We set $\lambda = 1$ for our experiments.

inequality constraints imposed when building the hyperspheres (see Equation (2)), we expect instances of the known malware to reside only within a single hypersphere. Thus, if an instance is located near the center of several hyperspheres, this is potentially a new variant of malware evolving from existing ones. Nevertheless, there are also cases where instances of known malware are enclosed within its own hypersphere as well as the hyperspheres for a few other classes (though, they are likely to be located far away from the center of the hyperspheres for other classes). By comparing the distance profile of the instances to all the hyperspheres, we conjecture that it is possible to distinguish a known malware from variants of the malware. However, comparing the distances to the centers of different hyperspheres is not meaningful as each hypersphere is defined in a completely different tree-transformed subspace.

To overcome this problem, we propose a technique that compares the distance distribution of a test instance to the centers of all hyperspheres. Let (a_j, R_j) be the center and radius of the hypersphere constructed for class j and D_j denote the set of radial distances between training instances from class j to the center of their corresponding hypersphere:

$$D_j = \{d_i = \| x_i - a_j \|^2 \, | y_i = j; d_i \leq R_j\}.$$

Let μ_j and σ_j represent the mean and standard deviation of the radial distances in D_j. We assume the radial distances for all training instances located within the hypersphere for class j follow a Gaussian distribution with mean μ_j and standard deviation σ_j. We use the radial distance distribution to represent the profile for class j and apply hypothesis testing to determine whether a test instance belongs to that class. Thus, we do not have to compare the distances of the test instance to the center of each hypersphere, rather we estimate the probability that it belongs to the hypersphere and combine their probability scores. We then set a threshold on the combined probabilities to isolate the new classes from known ones. The details for our new class detection approach are given in Algorithm 2.

The `ComputeScore` function takes the p-values with respect each class (sphere) as input and returns the geometric mean as output. Each p-value signifies to what extent the test point belongs to the class with larger value indicating, higher plausibility of belonging to the same class. The geometric mean will indicate whether the instance have the same profile distribution as any of the known malwares; otherwise it will be classified as a new (variant) of the known malware.

Algorithm 2 Detection of New and Known Malwares

Input: $\mathcal{C} = \{(a_i, R_i)\}_{i=1}^{k}$, where a_i is the center and R_i is the radius of the hypersphere for class i.
$\Pi = \{(\mu_i, \sigma_i)\}_{i=1}^{k}$, set of profile distributions .
$\mathbf{x}$, input test instance.
Output: y, predicted label for instance $\mathbf{x}$.

```
P = ∅
for i = 1 to k do
    d[i] = || x − C_i ||^2
    if d[i] < R_i then
        P = P ∪ {d[i]}
if P = ∅ then
    return (NewClass)
else
    for i = 1 to k do
        pvalues[i] = SignificanceLevel(d[i], μ_i, σ_i)
score = ComputeScore(pvalues)
if score > τ then
    return (NewVariantClass)
else
    y = argmax_i pvalues[i];
    return y
```

IV. Experimental Evaluation

In this section, we present the experimental results comparing the performance of different aspects of the proposed framework. First, we evaluate the effectiveness of applying a tree-based feature transformation approach in dealing with network data that contains missing values and non-applicable features. We then compare the proposed framework against other baseline algorithms in terms of their ability to detect a large number of known malware classes with varying sizes and feature characteristics. Finally we assess the effectiveness of incorporating profile distributions into one-class SVM for new class discovery.

TABLE I
Examples of flow-level features generated by the Narus Semantic Traffic Analyzer (STA).

Name	Feature Description
dir	direction (client to server or server to client)
#pkts	total number of packets
#pkt-p	total number of packets without payload
bytes	total number of bytes transferred
pay_bytes	total number bytes from all payloads
Δt	flow duration
maxsz	maximum packet size
minsz	minimum packet size
avgsz	average packet size
stdsz	standard deviation of packet size
IAT	average inter-arrival time
maxpy	maximum payload size
minpy	minimum payload size
avgpy	average payload size
stdpy	standard deviation of payload size
Flag	TCP flags (acks, fins, resets, pushs, urgs, etc)
sz_X	size of packet X (X= 1,2,$\cdots$,10)
IAT_X	inter arrival time of packet X
pay_X	payload size of packet X

A. Data

The proposed framework is evaluated using network flow data from an Internet service provider in Asia. Table I describes a subset of the 108 flow-level features extracted from the data using Narus Semantic Traffic Analyzer (STA). Some of the challenges in using such features for classification include dealing with (1) heterogeneous features (i.e., mixture

TABLE II
NUMBER OF TRAINING AND TEST INSTANCES FOR EACH MALWARE CLASS.

Class ID (Name)	# Train	# Test
0 (good)	**2103**	210402
1 (MS SQL Server 2000 Attack)	**151**	151
2 (Trojan Asprox)	0	3
3 (Fake App / AV Website)	0	1
4 (Gammima Request Activity)	0	2
5 (MSN Messenger Login Attack)	0	4
6 (Tor Activity)	0	3
7 (Trojan Sasfis)	0	6
8 (Trojan zbot Post Install)	0	18
9 (W32 Downadup Downloader)	**68**	67
10 (W32.Rontokbro)	0	1
11 (W32.Sality Activity 2)	0	4
12 (W32.Sality Activity Download)	0	2
13 (W32.Sality Activity)	**94**	94
14 (Zbot Activity)	0	4
15 (Hotbar Updates)	0	1
16 (Jabber IM Client Connection)	**29**	29
17 (MSN Messenger Login Attempt)	**28**	28
18 (Malicious Site)	0	11
19 (NetBIOS NBStat Query)	**72**	71
20 (P2P Ares Client Connection)	0	13
21 (P2P BitTorrent Activity 1)	**750**	750
22 (P2P BitTorrent Activity 2)	**685**	685
23 (P2P Downadup Activity0	**98**	98
25 (P2P Edonkey Ping Message)	0	22
26 (P2P Gnutella Connection)	**56**	55
27 (Skype Requesting Updates)	0	12
28 (Bot C&C Connection 2)	0	3
29 (Bot C&C Connection 3)	0	2
30 (Bot C&C Connection)	**27**	27
32 (HTTP Tidserv Download Req)	0	1
33 (Tidserv Activity 2)	0	1
34 (Tidserv Activity)	0	35
35 (Trojan SpyEye Activity 1)	0	4
36 (Phoenix Toolkit File Download)	0	1
37 (Yahoo! IM Login Attack)	**45**	44
38 (Unknown 1)	0	38

of continuous and categorical types), (2) correlated features, and (3) missing values due to inapplicable features (e.g., the features sz_9 and sz_{10} are inapplicable to flows that contain less than 9 packets).

We use a reputed commercial IDS/IPS system to generate the class label for each flow by analyzing their corresponding payload. The IDS will label a flow if its payload matches a defined signature of a class. In our data, the IDS system has identified 38 different types of malicious flows listed in Table II. The flows that were unlabeled by the IDS system are assigned to "good" (unknown) category. Majority of these 38 classes are known high-risk malwares, including *Sality, Conficker (Downadup), Tidserv, Trojans, Fake AV Attacks, C & C Bots, etc*. However, this table also includes some mid or low-risk suspicious activities, such as *Failed IM login attempts, Hotbar Updates Activity, etc.* and unwanted user activities, such as, *P2P BitTorrent Activity*. We have included such classes in our data as often System Administrators of Enterprise networks identify such activities and try to block them. So, it will be really beneficial if we can identify such classes along with the more malicious malwares.

The whole dataset contains 216,899 flows, out of which only 4,394 of them (2%) were labeled as malicious and categorized into one of the 38 known classes. It should be noted that some of the "good" flows may actually be malicious as they were not detected by the current IDS system. Thus, our experimental study reports only the classification performance on the 38 classes since the performance on the "good" class may not be reliable. The data is partitioned into two disjoint sets, one for model building (training set) and the other for model evaluation (test set). To simulate new class discovery, some of the malware classes were withheld from the training set and appear only in the test set. More specifically, the training set contains 2,103 malicious flows from the 12 most prevalent malware classes with an equal number of flows belonging to the "good" class. The remainder of the network flows were then assigned to the test set. The test set includes 2,099 malicious flows belonging to the 12 malware classes in the training set as well as 192 flows belonging to 24 "new" malware classes that are not present in the training set. The class distribution is summarized in Table II.

B. Comparing Tree-Based Feature Transformation against Missing Value Imputation

Our framework employs a collection of one-class SVM classifiers to distinguish each type of malware from other types in the data set. Since one-class SVM relies on the similarity values (encoded in the kernel matrix) computed between every pair of instances in the data set, the performance of the classifier can be degraded by poor handling of any imperfections in the data (such as noise, missing values, or collinearity among the features). To address this problem, we perform a nonlinear transformation on the original feature space using the tree-based approach described in Section III-A. To validate the effectiveness of this strategy, we compared our tree-based transformation approach against standard data imputation methods used for handling missing values in the data.

Ideally, a reliable kernel matrix should reflect the class similarity between the data instances, i.e., instances that belong to the same class should be more similar to each other than to those belonging to other classes. One way to evaluate the effectiveness of the various approaches in handling the data imperfection issues is by providing their kernel matrices to the 1-nearest neighbor classifier and compare their classification performance. A summary of the methods used to compute the kernel matrix for the network flow data is given below:

- **Original**: The original feature values generated by Narus Semantic Traffic Analyzer as shown in Table I are non-negative. Missing values can be annotated as -1 or other invalid values that lie outside the range of permissible values for that feature. In this approach, features that contain missing values are ignored when computing the distance or similarity between two data instances. For example, the radial basis function kernel between two instances x and x' are given as follows

$$K(x, x') = \exp\left[-\frac{\sum_{x_i \neq -1, x'_i \neq -1}(x_i - x'_i)^2}{2\sigma^2 N_{xx'}}\right],$$

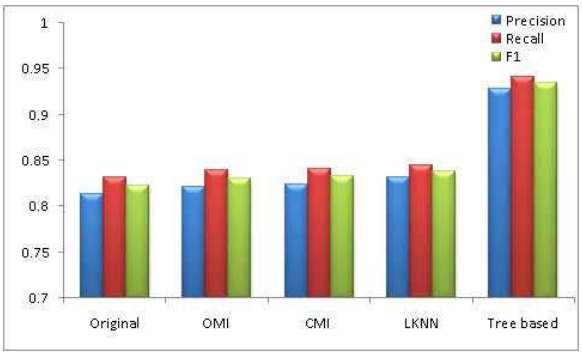

Fig. 5. Comparing the classification performance of tree-based features against other baseline approaches for handling data imperfection issues.

where $N_{xx'}$ is the number of non-missing features in both x and x'.

- Mean Value Imputation. In this approach, the missing values are replaced either by the overall mean computed for that feature (**OMI**) or the mean value of the feature for the given class (**CMI**). The latter approach is only applicable if the class label of an instance is known (i.e., for training instances only). If the missing value is present in a test instance, we first apply the k-nearest neighbor classifier (kNN) on the non-missing features to determine its expected class label, and then impute its missing value by the corresponding mean value for the expected class. The RBF kernel function defined in Section III-B can then be applied to the imputed feature vectors.
- Local kNN Imputation (**LKNN**). In this approach, the missing values of an instance are imputed based on the value of its closest neighbors. Specifically, given a feature vector with missing values, we identified its k nearest neighbors based on the Euclidean distance to all the training examples that do not have any missing values. The missing value in a feature is then replaced by the mean (or median) value of the same feature among its k-nearest neighbors. In our experiment, we set k = 5 as the parameter for the local kNN imputation method, though this parameter can be chosen in a more systematic way (e.g., via cross validation). The RBF kernel function can be computed from the kNN imputed feature vectors.
- **Tree-based**. In this approach, we transform the original data into a new feature representation using the approach described in Section III-A. Specifically, we constructed 25 trees for each class and compute the RBF kernel function on the combined tree-based features.

Figure 5 shows the experimental results obtained when applying the 1-nearest neighbor classifier to the kernel matrices computed using the techniques described above. The classification results are reported in terms of their precision, recall, and F1-measure. As can be seen from this figure, the tree-based approach gave significantly higher precision, recall, and F1 measure compared to other approaches.

TABLE III
CONFUSION MATRIX FOR MACRO-LEVEL CLASSIFICATION USING RANDOM FOREST.

Actual class	Predicted class	
	Good	Malicious
Good (Unknown)	209973	429
Known malware	0	2099
New malware	22	170

C. Macro-level Classification Results

The proposed framework initially applies a binary classifier to filter the suspicious flows from other legitimate ones. In this study, we employ random forest [13] as the macro-level binary classifier. Random forest is an ensemble of decision tree classifiers that predicts its class by combining the outputs produced by the individual tree classifiers. The classification results shown in Table III suggest that the random forest classifier is capable of detecting all the known malware and 88.54% of the new malware classes in the test set for an overall F-1 measure of 90.96% on the malicious classes. Although random forest works well for the binary classification task, as will be shown in the next section, it is incapable of detecting new classes when used as a micro-level classifier.

D. Micro-level Classification Results

This section presents the results of applying the micro-level classifiers to the network flow data. Their performance are evaluated in terms of their ability to distinguish the known from newly emerging malware. As mentioned in Section III-B, we consider two types of kernels: one based on radial basis function (RBF) while the other is based on supervised weighted linear kernel (WL). We also evaluated and compared the kernels generated from tree-based features against kernels computed using the raw features (with mean imputation for the missing values). In addition, we also compare the performance of the framework that uses **profiling** against one without profiling (see Section III-C).

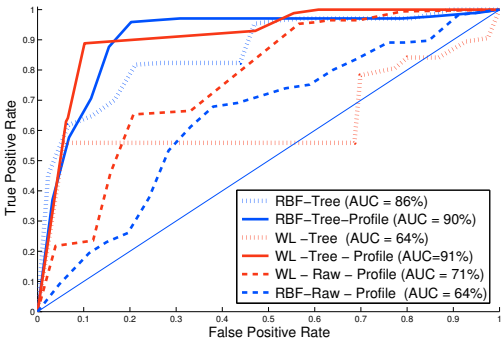

Fig. 6. ROC curves for the detection of new malware class using 1-class SVM with different kernels.

Figure 6 shows the ROC curve for the detection of new classes using 6 different approaches: (1) RBF-raw and WL-raw correspond to 1-class SVM models constructed from the

TABLE IV
COMPARISON BETWEEN F1 MEASURE FOR DETECTING KNOWN AND NEW CLASSES USING RANDOM FORESTS AND 1-CLASS SVM WITH TREE-BASED FEATURES.

	without Profile		with Profile		
Class	RBF	WL	RBF	WL	Random Forest
New	0.38	0.46	0.42	0.50	0.00
Known Classes					
1	1.00	1.00	1.00	1.00	0.99
9	1.00	1.00	0.99	1.00	1.00
13	0.86	0.85	0.80	0.87	0.71
16	0.98	0.98	0.91	0.98	0.96
17	0.93	0.93	0.70	0.93	0.89
19	0.99	0.88	0.93	0.88	0.98
21	0.93	0.90	0.89	0.90	0.80
22	0.98	0.97	0.96	0.97	0.90
23	0.97	0.90	0.98	0.90	0.98
26	0.98	0.96	0.93	0.96	0.97
30	0.68	0.54	0.62	0.26	0.51
37	0.96	0.99	0.48	0.99	0.92

original features (with mean value imputation), (2) RBF-tree and WL-tree correspond to 1-class SVM models constructed from the tree-based features (without profiling), and (3) RBF-tree-profile and WL-tree-profile correspond to 1-class SVM models with tree-based features and profiling (for new class discovery). Clearly, the tree-based features give higher area under the ROC curve (AUC) compared to the mean imputed raw features. The profiling technique also gives significantly higher AUC than those without profiling. These results validated the effectiveness of the proposed tree transformation and profiling techniques for 1-class SVM.

In addition, we also evaluated the performance of the classifier on the known malware. For this experiment, we set the threshold τ for new class detection in such a way that yields a 10% false alarm rate. We compared the performance of the 1-class SVM models against the random forest classifier (trained on all the known malware classes). Table IV shows the precision and recall values for each class along with their corresponding $F1$ measure. The results suggest that the supervised weighted linear kernel gives the highest F1 measure for new class detection without compromising the F1 values of the known classes. In addition, the profiling method increases the F1 measure for new class detection for both RBF and WL kernels.

V. CONCLUSION

This paper presents a malware detection approach based on features derived from the layer 3 and layer 4 network flow characteristics. Though the features are more resilient to payload encryption, they have other issues (missing values and correlated features) that hamper the applicability of more sophisticated learning algorithms. Furthermore, the supervised nature of the algorithm makes it difficult for detecting new malware. Our proposed approach addresses these challenges and identify flows of existing and novel malwares with very high precision. First, a tree-based feature transformation approach is developed to handle the data imperfection issues. A weighted linear kernel based on tree-based features is also proposed to effectively detect the malware classes. Finally, we present a novel adaptation of 1-class SVM to identify new types of malware.

For future work, we plan to extend the formulation to an online learning setting. In addition, when the number of classes becomes extremely large, the prediction step is expensive due to the large number of hyperspheres that must be tested. To address this problem, we plan to develop a hierarchical multi-class learning approach where the hyperspheres are organized in a hierarchical structure, thereby reducing the number of hyperspheres to be tested when predicting the class label of a test instance.

REFERENCES

[1] M. Christodorescu, "Semantics-aware malware detection," in *IEEE Symposium on Security and Privacy*, 2005, pp. 32–46.

[2] N. Idika and A. P. Mathur, "A survey of malware detection techniques," Purdue University, Technical Report SERC-TR-286, 2007.

[3] A. Sung, J. Xu, P. Chavez, and S. Mukkamala, "Static analyzer of vicious executables," in *Annual Computer Security Appl Conf*, 2004, pp. 326–334.

[4] J. Kephart and B. Arnold, "Automatic extraction of computer virus signatures," in *Proceedings of the 4th Virus Bulletin Internationl Congerence*, 1994, pp. 178–184.

[5] T. Abou-Assaleh, N. Cercone, V. Keselj, and R. Sweidan, "Detection of new malicious code using n-grams signatures," in *Proc of the 2nd Annual Conf on Privacy, Security, and Trust*, 2004, pp. 193–196.

[6] A. Lakhotia, A. Kapoor, and E. Kumar, "Are metamorphic viruses really invincible," *Virus Bulletin*, vol. 12, p. 57, 2004.

[7] M. Bailey, J. Andersen, Z. Morleymao, and F. Jahanian, "Automated classification and analysis of internet malware," in *In Proc of Recent Advances in Intrusion Detection*, 2007.

[8] V. Chandola, E. Eilertson, L. Ertoz, G. Simon, and V. Kumar, *Data mining and cyber security*. Springer, 2006, pp. 83–103.

[9] G. Gu, R. Perdisci, J. Zhang, and W. Lee, "Botminer: Clustering analysis of network traffic for protocol and structure independent botnet detection," in *USENIX Security Symposium*, 2008, pp. 139–154.

[10] N. Weaver, V. Paxson, S. Staniford, and R. Cunningham, "A taxonomy of computer worms," in *Proc of the 2003 ACM Workshop on Rapid Malcode*, 2003, pp. 11–18.

[11] A. H. Sung and S. Mukkamala, "Identifying important features for intrusion detection using support vector machines and neural networks," in *Symposium on Applications and the Internet (SAINT)*, 2003.

[12] P.-Y. Hao, J.-H. Chiang, and Y.-H. Lin, "A new maximal margin spherical structured multi class support vector machine," *Applied Intelligence*, vol. 30, pp. 98–111, 2009.

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