Weighted Linear Kernel with Tree Transformed Features For Zero-Day Malware Detection

Prakash Mandayam Comar, Lei Liu, Pang-Ning Tan,
Dept. of Computer Science
Michigan State University
{mandayam, liulei1, ptan}@cse.msu.edu

Sabyasachi Saha, Antonio Nucci
Narus Inc.
Sunnyvale, CA 94085
{ssaha , anucci }@narus.com

ABSTRACT

Malware is one of the most damaging security threats facing the Internet today. A malware-infected machine can be used to launch other cyberattacks such as spamming, phishing, spying, and identity theft. In this paper, we present a novel learning framework capable of detecting both known and newly emerging malware in spite of imperfections in the data. The framework leverages the capability of supervised classification in detecting known classes with the adaptability of unsupervised learning in detecting new classes. A non-linear, tree-based feature transformation is applied to address the data imperfection issues and to construct more informative features for the malware detection task. We empirically demonstrate the effectiveness of our framework using real network data from a large Internet service provider.

1. INTRODUCTION

Malware is a malicious software program stealthily 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. Recent malware programs have been designed with a profit motive in mind. These programs, 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 [2]. Signature based detection employs a set of pre-defined signatures for known malware to determine whether the software program under inspection is malicious. 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 will also keep changing [4]. Similarly, polymorphic malwares are also difficult to identify using this approach as they self-mutate and use encryption to avoid detection. All such approaches that look for payload signatures can be easily defeated by encryption or obfuscation techniques. On the other hand, anomaly-based detection techniques use the knowledge of what is considered as good behavior to identify malicious programs. The key advantage of anomaly-based detection is its ability to detect zero-day attacks. 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.

In this paper, we present a malware detection framework that uses a set of layer-3 and layer-4 network traffic features such as *bytes per second*, *packets per flow*, and *packet inter-arrival times*. Our goal is to classify the specific types of malware based on their network flow characteristics rather than the content of their packets. This enables us to classify the flows even when they are encrypted. Specifically, a two-stage classification approach 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 discriminate the malicious from non-malicious flows. Next, among those flows classified as malicious, a collection of one-class support vector machine (SVM) classifiers will be applied to identify the specific type of malware found in the malicious flows. However, since the classifiers are trained using only the known malware in the training set, they will not be able to detect new or evolving malware. To circumvent the problem, we employ the one-class SVM to determine whether a flow detected as malicious corresponds to a known malware or a new variant that was not available in the training set. Furthermore, the one-class SVM classifiers developed in this study are also unique in that they employ a non-linear, tree-based kernel to address the data imperfection issues such as noise missing values correlated and inapplicable features while constructing more informative features that for accurate detection of various malware classes.

2. PROPOSED FRAMEWORK

As mentioned above, we employ a two stage classification. The rationale for adopting such a framework is as follows. 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. Secondly, 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.

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.

2.1 Tree-based Feature Transformation

Let $\mathcal{D} = \{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\} \in \mathcal{X} \times \mathcal{Y}$ be the network flow data where $\mathcal{Y} = \{1, 2, ...k\}$ represents the 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 supervised learning algorithm for the network flow is the imperfections of the data. As shown in Table 1, some of the extracted features are strongly correlated with each other. Furthermore, a large flow with many packets tend to have more features (size of packet #5, size of packet #6, and so on.) compared to one with smaller number of packets. In order to remove this feature size variation, we fix the number of features by considering all the features extracted from the largest flow and then populating the corresponding values for smaller flows with a token (-1) indicating non-applicability of the feature to the smaller flow. It should be noted that all the numeric features (number of packets, sizes and inter arrival times) are non-negative. The easiest way to deal with missing values due to non-applicable features is to eliminate the data instances having at least one feature with missing value. However, this reduces the amount of training data available for constructing a reliable classifier. In fact, if we had discarded instances with missing values from the ISP data used in our experiment, we would be left with only 7% of the original flows.

The goal of our feature transformation step is to project the data into a metric space so that standard classification techniques can be applied to the transformed data. 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 employ a tree-based approach to construct the new features (see Algorithm 1). An advantage of using a tree-based classifier is that it can automatically handle missing values due to non-applicable features during the tree construction process by treating "inapplicable" as another separate value that can be used as splitting criteria for propagating the data instances to the different children of a node.

For each class c, we build p trees by labeling the instances belonging to class c as -1 and data from other remaining classes as $+1$. The predictions made by each tree on the training instances becomes a new feature, which has the capability to separate instances belonging to class c from the rest of the data. We also need to introduce randomness

Algorithm 1 `Tree-Based Feature Transformation`

```
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 $\mathcal{S}$.
            $h_c^i : \mathcal{S} \to [-1, +1]$
        Apply the tree $h_c^i$ on $\mathcal{X}$ and store
            $F_c(:, i) = h_c^i[\mathcal{X}]$
    end for
end for
return $\{F_c\}_1^k$
```

into the tree construction process in order to avoid generating the same tree for each class. This is accomplished by bootstrapping the instances and subsampling their features instead of using the whole training set for building the trees. This helps reduce the correlations among the newly generated 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

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

Each $\mathcal{F}_c$ represents a subspace of the transformed space $\mathcal{F}$ with the unique property that instances belonging to class c should be well separated from the rest of the data.

2.2 1-Class SVM with Tree-based Kernel for Multiclass Learning

We employ a collection of one class SVM classifiers for multi-class classification and new class detection [3]. Each one class SVM is modeled as 1-versus-all problem where we build a hypersphere for the class i by first labeling the instances belonging to the class i as $+1$ and those that do not belong to the class i 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 is the objective function to be maximized by the 1-class SVM algorithm.

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. Details of the algorithm can be found in [3]

RBF and polynomial kernels are typically used to construct 1-class SVM. However, they may not work well with data that contains missing values and inapplicable features. To circumvent this problem, our framework uses a tree-based feature transformation approach to define a suitable kernel for building the hyperspheres. As noted in Section 2.1, a set of tree-based features $\mathcal{F}_i$ is initially induced for each class i from the original feature space using an ensemble of tree-based classifiers. The RBF and polynomial kernel function can then be applied to the features in $\mathcal{F}_i$ to create the kernel matrix needed for constructing the hypersphere of class i. This approach considers each tree to be equally important in the construction of the kernel matrix for 1-class SVM. If some trees are superior than others, it would be useful to learn a set of weights for the tree-based features when computing the kernel. In this study, we propose a supervised linear kernel for tree-based features, where the weights for the features are estimated by aligning the induced weighted kernel to the ground truth kernel defined as follows.

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

Given the ground truth kernel G, we learn a linear kernel function $K(x, x') = x^T W x'$ that best approximates G by finding the weight matrix W that minimizes the following objective function.

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

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.

$$\frac{\partial \mathcal{L}}{W_{pq}} = -\sum_{ij} G_{ij}(X_{ip}X_{qj}^T) + \lambda W_{pq} = 0 \qquad (3)$$

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

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

Once the enclosing hyperspheres are constructed using the weighted linear kernel for all the classes, the test instances are classified by considering their distance to the center of each hypersphere. The test instance that lies outside all the hyperspheres are tagged as new class.

3. EXPERIMENTAL EVALUATION

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

Table 1: 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
sz	max/min/avg/sdev packet size
py	max/min/avg/sdev payload size
IAT	average inter-arrival time
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

In this section, we present the experimental results comparing the performance of different aspects of the proposed framework.

The proposed framework is evaluated using network flow data from an Internet service provider in Asia. Table 1 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 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 an industry standard IDS/IPS system to generate the class label for each flow by analyzing their corresponding payload. In all, the IDS system has identified 36 different types of malicious flows. The flows that were unlabeled by the IDS system are assigned to "good" (unknown) category. The dataset contains 216,899 flows, out of which only 4,394 of them (2%) were labeled as malicious and categorized into one of the 36 known malware. 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 36 malware classes since the performance on the "good" class may not be reliable.

The overall 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.

3.1 Comparing Tree-Based Feature Transformation against Missing Value Imputation

In this section, we evaluate the effectiveness of applying a tree-based feature transformation approach in dealing with network data that contains missing values and nonapplicable features. We do this by imputing the missing values

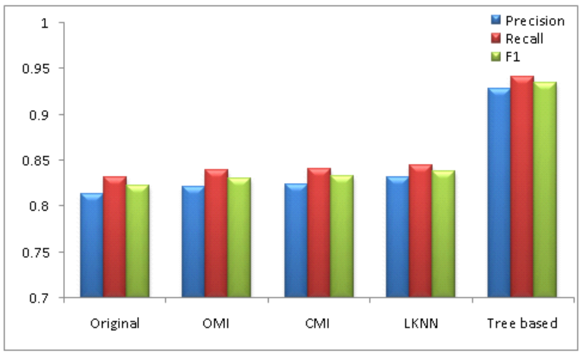

Figure 1: Comparing the classification performance of tree-based features against other baseline approaches for handling data imperfection issues.

Table 2: 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

by different imputation methods and use the 1-NN classifier to classify. The baseline imputation methods are **Original**, where we compute the Euclidean distance without considering the inapplicable/missing features, imputing with overall feature mean(**OMI**), imputing with class specific mean **CMI**, and Local kNN Imputation (**LKNN**) where, the missing values of an instance are imputed based on the value of its closest neighbors computed using non-missing values. We compare these baseline techniques against the proposed tree based feature transformation. Figure 1 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.

3.2 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 [1] 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 2 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.

3.3 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. Here, we consider two types of kernels defined on tree based features: one based on radial basis function (RBF) while the other is based on supervised weighted linear kernel (WL). We compared the performance of the 1-class SVM models against the random forest classifier (trained on all the known malware classes). Table 3 shows the $F1$ measure for each class. 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.

Table 3: Comparison between F1 measure for detecting known and new classes using random forest and the proposed framework with RBF and Weighted Linear Kernel on tree-based features.

Class	# Instances in Train set	# Instances in Test set	RBF	WL	Random Forest
New	0	170	0.38	0.46	0.00
Known Classes					
1	151	151	1	1	0.99
9	68	67	1	1	1.00
13	94	94	0.86	0.85	0.71
16	29	29	0.98	0.98	0.96
17	28	28	0.93	0.93	0.89
19	72	71	0.99	0.88	0.98
21	750	750	0.93	0.9	0.80
22	685	685	0.98	0.97	0.90
23	98	98	0.97	0.9	0.98
26	56	55	0.98	0.96	0.97
30	27	27	0.68	0.54	0.51
37	45	44	0.96	0.99	0.92

4. 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. This paper presents a framework to address these challenges. 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 simple adaptation of 1-class SVM to identify new types of malware.

5. REFERENCES

[1] L. Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001.

[2] M. Christodorescu. Semantics-aware malware detection. In *IEEE Symposium on Security and Privacy*, 2005.

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

[4] A. Lakhotia, A. Kapoor, and E. Kumar. Are metamorphic viruses really invincible. *Virus Bulletin*, 12, 2004.