# Discriminant Malware Distance Learning on Structural Information for Automated Malware Classification

Deguang Kong
Dept. of Computer Science and Engineering
University of Texas at Arlington
doogkong@gmail.com

Guanhua Yan
Information Sciences Group (CCS-3)
Los Alamos National Laboratory
ghyan@lanl.gov

## ABSTRACT

In this work, we explore techniques that can automatically classify malware variants into their corresponding families. Our framework extracts structural information from malware programs as attributed function call graphs, further learns discriminant malware distance metrics, finally adopts an ensemble of classifiers for automated malware classification. Experimental results show that our method is able to achieve high classification accuracy.

**Categories and Subject Descriptors:** I.2.6 [Artificial Intelligence], D.4.6 [Security and Protection]

**General Terms:** Algorithm, Security, Data Mining

**Keywords:** Malware categorization, Distance learning

## 1. INTRODUCTION

Malware are responsible for a large number of malicious activities in the cyber space, such as spamming, identity theft, and DDoS (Distributed Denial of Service) attacks. Behind the sheer number of malware instances, however, lies the fact that a large number of them came from the same origins. More than 75 percent of malware detected belong to as few as 25 families, based on the 2006 Microsoft Security Intelligence report [5]. Accurate prediction of the evolution trend of a malware family also enables us to deploy effective mitigation methods in advance and thus alleviate the damage caused by this malware family.

Therefore, there is an urgent need of developing methods that can automatically classify malware instances into their corresponding families accurately. The goal of this work is to develop a framework that automatically classifies malware instances according to their inherent rich structural information, such as their function call graphs and basic block graphs.

In this work, we present a new framework for automated malware classification using discriminant distance learning on structural information extracted from malware. This framework extracts the function call graph from each malware program, and collects various types of fine-grained features at the function level, such as what system calls are made and how many I/O read and write operations have been made in each function. For each type of features, our framework evaluates the similarity of two malware programs by iteratively applying the following two basic techniques: (1) *discriminant distance metric learning*, which projects the original feature space into a new one such that malware instances belonging to the same family are closely clustered while clusters formed by different malware families are separated with large margins; (2) *pairwise graph matching*, which aims to find the right pairwise

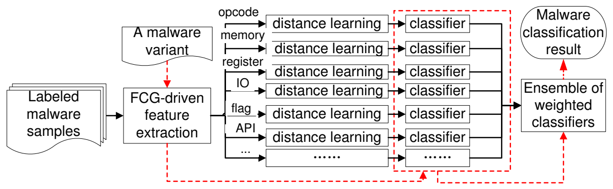

Figure 1: Overview of our automated malware classification framework (solid lines are used for the training process, and dashed line for the process of classifying a new malware variant)

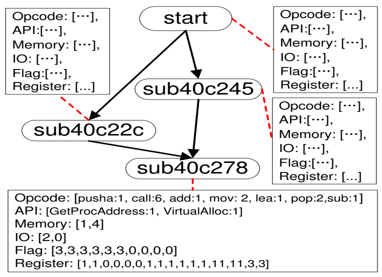
start
sub40c245
sub40c22c
sub40c278
Opcode: [···], API:[···], Memory: [···], IO: [···], Flag:[···], Register: [...]
Opcode: [···], API:[···], Memory: [···], IO: [···], Flag:[···], Register: [...]
Opcode: [···], API:[···], Memory: [···], IO: [···], Flag:[···], Register: [...]
Opcode: [pusha:1, call:6, add:1, mov: 2, lea:1, pop:2,sub:1]
API: [GetProcAddress:1, VirtualAlloc:1]
Memory: [1,4]
IO: [2,0]
Flag: [3,3,3,3,3,3,0,0,0,0]
Register: [1,1,0,0,0,0,1,1,1,1,1,1,11,11,3,3]

Figure 2: Illustration of an attributed FCG

function-level matching between the function call graphs of two malware instances in order to measure their structural similarity. The similarity score estimated between two malware instances for each type of features reflects the likelihood that they should be classified into the same malware family – if observed feature values of that type are used as our *evidence*. We further learn our *confidence* level in each type of evidence and henceforth build a classifier that predicts the family of a new malware instance by combining different types of evidences with their corresponding confidence levels.

## 2. OVERVIEW OF METHODOLOGY

The overview of our automated malware classification framework is depicted in Figure 1.

**Step 1: FCG-driven feature extraction.** To extract structural information from a malware program, we first disassemble the malware program, and build its function call graph. The function call graph is further used to drive the process of feature extraction: for every node (i.e., a function) in the graph, we extract various types of *attributes*, including what library APIs are made and how many I/O read and write operations have been made in this function. Information regarding each type of features is represented as a vector of numerical values. After Step 1, each labeled malware program is abstracted into an attributed function call graph (see Fig. 2).

**Step 2: Discriminant malware distance learning.** The next

Table 1: Different scenarios used in our experiments

| Scenario | **op-n** | **mem-n** | **reg-n** | **io-n** | **flag-n** | **api-n** | **ES-dis** |
|---|---|---|---|---|---|---|---|
| Attribute | Opcode | Memory | Register | I/O | Flag | API | Ensemble |
| Distance learning | No | No | No | No | No | No | Yes |

step concerns how to compute the distance between two malware distances represented as their attributed function call graphs. For each type of attribute, we project the original feature space onto a new one such that malware instances belonging to the same family are closely clustered while clusters formed by different malware families are separated with large margins. Moreover, we perform pairwise graph matching, which aims to find the right pairwise function-level matching between the attributed function call graphs of two malware instances for the purpose of measuring their structural similarity.

**Step 3: Training individual classifiers.** For each type of features, once we have computed the similarity between any two labeled malware instances, we train an individual classifier for it. Our framework is open to any classifier that, in order to classify a new sample, requires only information of a set of *anchor instances*, which are usually the subset of labeled samples in the original dataset. Such classifiers include the kNN classifier, for which the anchor instance set includes the $k$ closest instances from the test instance, and the SVM classifier, whose support vector contains all the anchor instances.

**Step 4: Building ensemble of weighted classifiers.** For each type of features we have considered, the similarity measure between two malware instances reflects the likelihood that they belong to the same family. Given a new malware variant, for each type of features, we form its *evidence* as the distance it is from each of its anchor instances as well as the label information of each anchor instance. The *type* of an evidence is defined to be the type of *attribute* from which it is formed. To learn the confidence level associated with a type of evidence, we use an Adaboost-like approach, which gives an increasingly higher penalty to training samples that are wrongly classified. We henceforth build a classifier that predicts the family of a new malware instance by combining different types of evidences according to their corresponding confidence levels.

The output of the training phase of our automated malware classification framework is an ensemble of classifiers. Given a new unknown malware sample, we first construct its function call graph from the disassembly code, and for each function node in it, we extract different types of attribute. Next, for each type of attribute, we form its evidence that describes the distance between the new sample and the anchor instances as well as how each anchor instance is labeled. We then feed the evidence to the corresponding individual classifier. By combining all the evidences, the ensemble of weighted classifiers makes the final decision on which malware family it should be classified into.

## 3. EXPERIMENTS

We use a malware dataset from Offensive Computing [6], which contains 526,179 unique malware variants collected in the wild. The malware dataset contains both packed and unpacked instances, and in our evaluation, we only use unpacked ones, and disasemble them with IDA pro [1]. We obtain 11 families of malware: Bagle, Bifrose, Ldpinch, Swizzor, Zbot, Koobface, Lmir, Rbot, Sdbot, Vundo, and Zlob.

We use 80% of the malware samples from each family to train our model, and the remaining ones are used for testing the effectiveness of our approach. This process is iterated for five times, and we report the averages as the classification performance.

Table 2: Average F-1 measure in terms of percentage across all families. For SVM, we show the results when $\gamma = 0.3$, $\gamma = 0.7$, and also the average when $\gamma$ is chosen from $[0.1, 0.3, \cdots, 1.9]$; for kNN, we show the results when $k$ = 6, 10, and also the average when $k$ is chosen from [2, 4, ..., 16].

| classifier | $\gamma/k$ | op-n | mem-n | reg-n | io-n | flag-n | api-n | ES-dis |
|---|---|---|---|---|---|---|---|---|
| SVM | 0.3 | 86.06 | 84.20 | **86.14** | 85.07 | 68.59 | 82.99 | **93.88** |
| kNN | 6 | 48.89 | 67.82 | 56.52 | 54.53 | **85.33** | 66.71 | **91.23** |
| SVM | 0.7 | **88.61** | 87.49 | 87.76 | 87.63 | 72.25 | 82.65 | **98.73** |
| kNN | 10 | 62.33 | 66.30 | **87.79** | 58.26 | 66.11 | 68.90 | **95.31** |
| SVM | avg | 84.54 | 84.51 | 83.57 | **85.52** | 68.39 | 84.72 | **93.44** |
| kNN | avg | 54.28 | 65.77 | 69.04 | 55.25 | **73.52** | 66.39 | **90.54** |

**Parameter setting:** For the $k$-nearest neighbor classifier, we choose $k$ between 6 and 10. For the SVM classifier, we use the Gaussian kernel: $k(\mathcal{G}_i, \mathcal{G}_j) = e^{-\gamma \frac{D_{i,j}^2}{t^2}}$,, where $\gamma$ is a tunable parameter, and $t$ is the average distance of the $k$-nearest neighbors for each malware. We choose $\gamma$ between 0.3 and 0.7, and $t$ is computed using the three nearest neighbors. We compare the performances of the methods in different scenarios as shown in Table 1. Our method corresponds to the **ES-dis** scenario.

In Table 2, we show the average performance improvement across all malware families. Clearly, for both classifiers, the $F_1$ measure is significantly improved using our method (i.e., `ES-dis`). For instance, considering both the average cases, our method improves over the best individual method by 9.3% for the SVM classifier, and by 23.2% for the kNN classifier.

## 4. RELATED WORK

Since the seminal works done by Schultz *et al.* [8] and Kolter *et al.* [2], machine learning techniques have been used in a number of efforts to automatically distinguish malware from benign executable programs (e.g., [7, 3, 4]). In contrast to these earlier works on malware detection, this study focuses on malware classification by distinguishing instances from different families. In [9], Yan *et al.* compared the discriminative power of different types of malware features for automated malware classification.

## 5. CONCLUSIONS

We present a generic framework for automated malware classification. We develop methods to compute the similarity of two malware programs based on their attributed function call graphs, and use an ensemble of classifiers that learn from the pairwise malware distances and classify new malware instances automatically.

## 6. REFERENCES

[1] `http://www.hex-rays.com/products/ida/index.shtml`.

[2] J. Z. Kolter and M. A. Maloof. Learning to detect and classify malicious executables in the wild. *Journal of Machine Learning Research*, 7, 2006.

[3] D. Kong, Y. Jhi, T. Gong, S. Zhu, P. Liu, and H. Xi. Sas: Semantics aware signature generation for polymorphic worm detection. In *SecureComm*, pages 1–19, 2010.

[4] D. Kong, D. Tian, P. Liu, and D. Wu. Sa3: Automatic semantic aware attribution analysis of remote exploits. In *SecureComm*, pages 190–208, 2011.

[5] Microsoft security intelligence report, January-June 2006.

[6] `http://www.offensivecomputing.net/`. Accessed in March 2012.

[7] R. Perdisci, A. Lanzi, and W. Lee. Mcboost: Boosting scalability in malware collection and analysis using statistical classification of executables. In *ACSAC*, pages 301–310, 2008.

[8] M. G. Schultz, E. Eskin, E. Zadok, and S. J. Stolfo. Data mining methods for detection of new malicious executables. In *In Proceedings of the IEEE Symposium on Security and Privacy*, pages 38–49, 2001.

[9] G. Yan, N. Brown, and D. Kong. Exploring discriminatory features for automated malware classification. In *Proceedings of DIMVA'13*.