Using File Relationships in Malware Classification

Nikos Karampatziakis[1], Jack W. Stokes[2], Anil Thomas[1], and Mady Marinescu[1]

[1] Microsoft Corporation, One Microsoft Way, Redmond WA, 98052, USA,
just.nikos@live.com,{anilth,mady}@microsoft.com
[2] Microsoft Research, One Microsoft Way, Redmond WA, 98052, USA,
jstokes@microsoft.com

Abstract. Typical malware classification methods analyze unknown files in isolation. However, this ignores valuable relationships between malware files, such as containment in a zip archive, dropping, or downloading. We present a new malware classification system based on a graph induced by file relationships, and, as a proof of concept, analyze containment relationships, for which we have much available data. However our methodology is general, relying only on an initial estimate for some of the files in our data and on propagating information along the edges of the graph. It can thus be applied to other types of file relationships. We show that since malicious files are often included in multiple malware containers, the system's detection accuracy can be significantly improved, particularly at low false positive rates which are the main operating points for automated malware classifiers. For example at a false positive rate of 0.2%, the false negative rate decreases from 42.1% to 15.2%. Finally, the new system is highly scalable; our basic implementation can learn good classifiers from a large, bipartite graph including over 719 thousand containers and 3.4 million files in a total of 16 minutes.

Keywords: Malware detection, Machine Learning, File Relationships

1 Introduction

Symantec recently observed over 903 million files installed on a sample of 47 million computers [3]. While many of these single instance files are benign, a significant percentage are malicious. Malicious single instance files have two sources, polymorphic and metamorphic malware which installs a unique instance of the attack on each new computer, while legitimate software sometimes creates a unique file for each installation. Given the shear volume, human analysts cannot investigate each new file detected in the wild, and the anti-malware (AM) companies cannot solve the problem by hiring more analysts. Since malware authors often rely on automation to avoid detection, commercial anti-malware companies also need to use automation to detect new malware. Ideally, the most automatic and effective way to solve the problem would be to have secure systems that would not allow the execution of malicious code. However this paradigm is

currently far from realistic while malware proliferation is already a sad reality. Since malware is such a significant problem, researchers and industry have devoted much effort towards automated detection [9]. The main issue with these systems is that the false positive (FP) rates (i.e. when a benign file is predicted as malicious) are too high for widespread deployment. An acceptable FP rate for completely automated malware detection would be no more than 0.01%. For AM systems, FPs are much worse than false negatives (FNs) since cleaning (i.e. removing) FPs can prevent a legitimate application, or even the operating system, from running. Given this risk, most AM companies seek first to do no harm.

Current techniques for building malware classifiers lead to prohibitively high FN rates (e.g. $> 99\%$) when operating at an FP rate of 0.01%. To truly combat today's malware, we need to look from new sources of information. For example, a system that uses both the static structure of a file as well as information about its runtime behavior is likely to perform much better than systems that use only one of these sources. In this paper we use a different source of information, namely file relationships, and demonstrate that even a very simple way of incorporating this source into the classifier is very effective.

Typical approaches [12, 20] focus on classifying files in *isolation*. Recently, researchers have proposed using a file's reputation [3] in relationship to the reputations of all computers which report the file to improve the detection rate. Other authors [22] rely on co-occurrences of files on a set of client machines to establish threats such as trojan downloaders. In this work, we take a different approach to improving a malware classifier by learning a file's reputation based on its relationships with other files as we determine them through Microsoft's submission service. We empirically show improved classification accuracy at very low FP rates. Furthermore, we sidestep privacy issues that affect other approaches.

File relationships should contain useful information. Clearly it is enough to show this for a special case. In this paper we only consider containment relationships, the most prevalent type in our data. However, our algorithm is quite general and could be applied to other types of file relationships. Containment arises when malware is distributed in containers such as .zip or .rar files. We also ignore containment relationships among containers (archives containing archives) and only form a bipartite graph of files and containers. After these restrictions, our database has more than four million containers with more than one executable file inside them and exploiting this information may substantially improve over a classifier that ignores it.

Our method starts by training a *baseline classifier* to *individually* predict the probability that a file is malicious. This classifier is trained with over 2.6 million labeled files using logistic regression. This classifier, though simple, uses some very strong features coming from an actual execution of the file in a virtual machine, and is very fast during training and prediction. Our method then propagates this baseline prediction and other information from files to containers using the file relationship graph. It then trains a *container classifier* which assigns a probability to each archive. Here we investigate three ways of integrating information from the neighboring vertices. Finally, we significantly improve

the classification accuracy on individual files by training a *relationship classifier*, again using logistic regression, which considers the malware probabilities associated with all archives containing the file in addition to the file's baseline score. Experiments on a large collection of over 719 thousand archives including more than 3.4 million malicious and benign files show that the relationship classifier significantly outperforms the baseline classifier particularly at very low FP rates. For example, at an FP rate of 0.2%, the FN rate decreases from 42.1% to 15.2%.

In Section 2, we provide some background on using machine learning for malware classification. Our new container classifier and file relationship classifier are described in detail in Section 3. An overview of the system is presented in Section 4, and experimental results are given in Section 5. In Section 6, we discuss the assumptions of our method, and the related work is outlined in Section 7. Finally, we conclude the paper in Section 8. Our main contributions include:

- A large-scale system to classify unknown files based on file relationships.
- A comparison of three methods to classify file containers.
- A new algorithm for significantly improving a file's baseline prediction.
- An evaluation of our system on a large collection of over 719 thousand containers and 3.4 million files, where we demonstrate that it is highly scalable.
- A convincing empirical comparison based on performance at small FP rates.

2 Background and Notation

Most modern anti-malware software follows a rule-based method to detect malware usually referred to as "signatures". Recently, researchers [9] have proposed using machine learning *classifiers* to detect malware. Classifiers built from a set of labeled malicious and benign programs can generalize well to previously unseen but similar programs. Here we focus on linear logistic classifiers because:

- They allow our experimental results to be directly interpretable when the classifier is employed in the real world. In the real world we do not know the proportion of actual malware files. Remarkably, as we show later in the paper, the conclusions we draw from logistic regression remain valid.
- Our system builds on top of a baseline classifier that works on individual files. This classifier is itself based on logistic regression and we exploit this fact to provide an initial hint to our system.
- Finally a logistic classifier can make predictions very fast, which is highly desirable for the scalability of our system.

For each file x_i we would like to classify, we construct a vector of real numbers $\Phi(x_i) \in \mathbb{R}^d$ that contains measurements, also known as *features*, regarding the file x_i. Described in Section 4.2, one of the features we use for our baseline classifier consists of tri-grams of system calls. In general we assume that the mapping $\Phi(\cdot)$, that takes an executable and returns a vector in $\mathbb{R}^d$, is given to us. In typical cases, to get a descriptive enough representation of x_i, d is on the order of hundreds of thousands but for each individual x_i, $\Phi(x_i)$ is sparse, meaning that only few (on the order of a hundreds or a few thousand) of the entries in this vector are non-zero.

2.1 Classification with Logistic Regression

Logistic regression assumes that for each file x_i we can assign a score $s(x_i)$ that is a linear combination of the feature values for x_i, or more formally $s(x_i) = \sum_{j=0}^{d} w_j \cdot \Phi_j(x_i) = w^\top \Phi(x_i)$ for some, yet to be determined, *weights* w where $\Phi_0(x_i) = 1$, w_0 is the bias term, and $a^\top b$ denotes the inner product between vectors a and b. This score is converted into a probability using the *logistic link function* $g(t) = \frac{e^t}{1+e^t}$. Letting y_i be a random variable that is 1 if the file x_i is malware, and 0 otherwise, logistic regression assumes

$$p(y_i = 1|x_i) = g(s(x_i)) = \frac{e^{w^\top \Phi(x_i)}}{1 + e^{w^\top \Phi(x_i)}}. \tag{1}$$

Solving (1) for the score $s(x_i)$ we find that

$$s(x_i) = w^\top \Phi(x_i) = \log \frac{p(y_i = 1|x_i)}{1 - p(y_i = 1|x_i)} = \log \frac{p(y_i = 1|x_i)}{p(y_i = 0|x_i)}.$$

The right hand side is commonly referred to as the *log odds* of x_i being malware. We will return to this relation in Sections 3.5 and 3.6. Finally, finding the optimal weight vector w from a training set of files for which human analysts have provided *determinations* (i.e. whether the file is truly malware or not), is a well studied convex optimization problem for which extremely fast algorithms exist.

3 Beyond Individual Prediction

In this section, we develop new methods whose goal is to improve malware classification particularly at low FP rates. We begin by discussing two problems associated with typical classification algorithms and then propose using a file's relationships to overcome these issues. After considering an ideal, but impractical solution, we propose a new set of algorithms to achieve a similar effect.

3.1 Some Problems of the Standard Approach

First we argue that current approaches to malware classification have a lot of room for improvement if we consider how an analyst would go about determining whether a file is malware or not. Malicious files do not exist in a vacuum. Malware is often distributed in an archive and this reflects a relationship among these files. The exact meaning of the relationship varies from archive to archive. Some typical examples include an executable file and its dependencies (such as dynamically linked libraries), files created by the same author, or components of a large project. In any case, this is precious information that is not captured in the framework of individually predicting on each file, but is routinely leveraged by malware analysts. That is, approaches based on individual prediction ignore the relationships of the file under consideration to other related files and their determinations. This is not just a matter of extending the feature mapping $\Phi(\cdot)$ to include features from the related files. A related file itself may not reveal anything alarming but it may do so if one considers its own related files. Later in Section 5.1, we motivate this idea based on two particular examples.

3.2 Taking Context into Account

We propose to overcome the shortcomings of individual malware classification with a two step procedure which we describe in Sections 3.5 and 3.6. Before that, we introduce some terminology, and describe an ideal but impractical approach. We represent the file relationships with a graph in which the vertices correspond to files and the edges correspond to containment relationships. Even though containers can contain other containers, here we ignore this and focus on a bipartite graph between containers and regular files. We have an additional determination "malware container" which is given to containers that contain at least one file determined to be "malware". Of course, not all vertices have a determination of "malware" or "benign", and we would like to propagate information along the graph so that we can assign each vertex its own probability that it is malicious.

We immediately point out that such context information cannot be easily made available to anti-malware software running on a client, and facilitating this functionality to a client is beyond the scope of this paper. Our focus is to demonstrate the empirical gains we observe when this information is available and utilized on the backend with an approach that is relatively easy to implement and highly scalable. The method we advocate however, is inspired by an impractical solution. We nevertheless detail this impractical solution in the next section to explicate the ideas that lead to our scalable algorithm.

3.3 An Impractical Solution

To assign a malware probability to every file we assume we have a *baseline malware classifier* that employs the approach of Section 2 to assign a score $w^\top \Phi(x_i)$, and hence a probability via the logistic link function, to each executable file in our data. For the files already determined to be malware or benign we can define the maliciousness level to be the score from the baseline malware classifier. For all other files for which we have no determination we can formulate a set of *consistency equations* according to a very simple principle: *we can obtain the maliciousness level of a file by averaging the maliciousness levels, of its related files*, which of course are its neighbors in the graph. This definition treats maliciousness as a fixed point. Formally, letting s_i denote the maliciousness level for file i and $N(i)$ the set of neighbors of i in the graph we have

$$s_i = \begin{cases} w^\top \Phi(x_i) & \text{if } i \text{ is determined} \\ \frac{1}{|N(i)|} \sum_{j \in N(i)} s_j & \text{otherwise} \end{cases}$$

For m files, this defines an $m \times m$ system of linear equations. Typically m is huge; in our case m is greater than 250 million and is growing by two every second. This immediately precludes methods such as Gauss elimination [7] which scale as $O(m^3)$. Furthermore, the graph, and hence each equation, is also evolving because each submission to our service can induce new relationships. Hence the solution of the above linear system is largely of theoretical interest even if one seeks an approximate solution using iterative methods [7].

3.4 A Scalable Solution

The main problem with the previous solution is that it strives to be globally consistent and enforcing this is too time-consuming. Instead we relax the requirement for consistency and, to compensate for this, we make our aggregation rule more flexible. Instead of a simple average, we compute features of the immediate neighborhood of each file and find an optimal way to combine them. Specifically, our solution involves the following steps:

- compute a "malware probability" for each container by aggregating information from all the files it contains and the probabilities assigned to them by our baseline malware classifier.
- compute an improved estimate of the probability of a file being malware by aggregating malware probabilities from the archives that contain it as well as the baseline malware classifier.

3.5 Computing Container Probabilities

A malware analyst does not have to look at the whole graph to get a rough idea about how likely the file is to be malicious; the local neighborhood provides most of the information. Furthermore, we observe that *in order for a container to be malicious it suffices that one of its contained files is malware*. Conversely, a container has a low malware probability only when all the contained files are not malicious. Based on this observation we propose three "container classifier" algorithms for assigning a malware probability to each container: Max neighbor, Union bound, and Biased logistic regression.

The Max neighbor algorithm estimates the probability that an archive is a "malware container" by the maximum of the probabilities (as given by the baseline classifier) of any of the contained files being malware. The Union bound makes a simple assumption: each file is independently providing evidence about the maliciousness of the container. Hence, the probability of the container to be benign is the product over all contained files of their probabilities of being benign. For example, if an archive contains two files, one for which the baseline estimate that it is malware is 0.6 and another whose estimate is 0.5 then we assign a probability of $1 - (1 - 0.6)(1 - 0.5) = 0.8$ to the container being malware.

Biased logistic regression employs a logistic regression classifier with an additional offset motivated by the importance of the file with the maximum baseline probability contained in the archive:

$$\log \frac{p_c(y_i = 1|N(i))}{p_c(y_i = 0|N(i))} = v^\top \Psi(N(i)) + v' \log \frac{p_{b,\max}}{1 - p_{b,\max}} \tag{2}$$

where $N(i)$ is the set of files contained in archive i, v (vector) and v' are the model weights, $p_{b,\max}$ is the maximum of the probabilities assigned to all the files in $N(i)$ by the baseline malware classifier, and $\Psi(N(i))$ is a vector of features calculated from the files contained in archive i. This model is biasing its prediction based on the most malicious file among i's files, in accordance to our

observation that a container is malicious if at least one of the contained files is malicious. Furthermore we adjust this prior belief by a linear combination of features, $v^\top \Psi(N(i))$, computed from the neighborhood of i which captures the *maliciousness levels* of all of the files in the container. Most of these features come from three simple histograms of the baseline probability estimates of the files included in the container. The histograms are separately computed for files which are predicted to be malicious or benign by the baseline classifier. A third histogram is included for files for which the baseline classifier returned a probability but a label of inconclusive. For each type, we split the interval $[0, 1]$ into 20 equally sized bins and create a histogram of the probabilities of the contained files. Then the values of features Ψ_{2j} and Ψ_{2j+1} are respectively the fraction and the logarithm of the number of contained files whose baseline probability estimates fall in the j-th bin. We chose these features to capture both absolute and relative numbers that may affect our decision. These transformations are relatively insensitive to manipulation of the raw numbers from adversaries. We also include three additional features for the container classifier. The first is the biasing feature which is the inverse of the logistic link function. Since the baseline malware classifier is itself based on logistic regression, the biasing feature is $\log(p_{b,\max}/(1 - p_{b,\max}))$ where $p_{b,\max}$ is the contained file with the highest probability. As $(p_{b,\max})$ approaches 1, the biasing term becomes large and hence overshadows any effect from $v^\top \Psi(N(i))$. When $p_{b,\max}$ approaches 0, the biasing term becomes very negative and again overshadows the effect of $v^\top \Psi(N(i))$. Finally when the max probability is 0.5 the biasing term is 0. The second additional feature is the log of the number of files in the container and the third is the product of the first two additional features. The third additional feature captures interactions between the number of files and the maximum file probability in the container. The last two features were important for reducing the number of false negatives for the container classifier. The entire container classifier procedure is summarized in the top part of Table 1.

The main benefit of this approach is that it is extremely fast to make a prediction for a new container. We only need to look at its contained files, and retrieve their probabilities from our database. If a file has not been seen before, we need to obtain its probability from the baseline malware classifier, compute 123 features from three histogram (3*20 bins and two features from each bin plus the three additional features), and take a linear combination with the learned vector v.

3.6 Improving a File's Probability

Our end goal is to improve upon a system that classifies executable files individually. In this sense, the probabilities we obtain from the container classifier is just auxiliary information that summarizes the neighborhood of a file. Therefore we introduce a second step where *we aggregate information across the containers in which a given file participates.* Our final "relationship-based" classifier uses a similar set of features as the container-based classifier as well as a biasing term. However this time our prior belief reflected in the biasing term is that the

baseline classifier is doing well most of the time, and we only want to use the neighborhood information to learn a *correction*. Formally, we model the log odds of file i being malware ($y_i = 1$) as

$$\log \frac{p_r(y_i = 1|N(i))}{p_r(y_i = 0|N(i))} = u^\top \Psi(N(i)) + u' \log \frac{p_b(y_i = 1|x_i)}{p_b(y_i = 0|x_i)} \tag{3}$$

where p_r is the probability according to the relationship-based classifier, p_b is the probability according to the baseline classifier, Ψ is a vector of features we compute from the neighborhood of the file and u is a vector of weights that optimally combines the features according to the maximum likelihood principle and u' is a weight for biasing term. The term $u^\top \Psi(N(i))$ captures the maliciousness level of all of the containers which include the file under consideration. This time we derive the mapping to features Ψ by binning the probability estimates from the container classifier into two histograms (malware, benign) for each of the neighboring containers. If a neighbor is a new container and has not been classified yet we simply ignore it. This is fine since, as before, the neighborhood features will come to the rescue mostly when the baseline classifier's prediction is close to 0.5 and will be overshadowed by the biasing term as the baseline classifier becomes very confident. An additional feature of $\log \frac{p_b(y_i=1|x_i)}{p_b(y_i=0|x_i)}$ helps to bias the model to the baseline probability. The whole procedure is shown in Table 1. Our non-optimized implementation executes it in 16 minutes processing 719 thousand containers and 3.4 million files.

One could argue at this point that we could go back and, based on the improved probability estimates from the relationship classifier, compute new probabilities for archives. We could in fact iterate this procedure until it converges to a fixed point. However it is doubtful that such a fixed point will lead to substantially better *generalization* than our procedure. First, the fixed point integrates information from many potentially unrelated files which are simply too far from the file under consideration. Second, it is well known among machine learning practitioners that treating the output of one classifier as an input to another (aka cascading) is an extremely delicate procedure. Hence it is usually observed that, as a function of the length of the cascade, the generalization performance of the final output rapidly deteriorates.

4 System

This section discusses the system aspects related to training the relationship malware classifier illustrated in Figure 1. The data analyzed in this paper consists of a very large subsample of containers and files used by Microsoft to investigate and create signatures for a number of our commercial anti-virus products including Microsoft Security Essentials and Forefront Endpoint Protection. Microsoft collects suspicious files using a variety of sources including the end user, product support, security organizations (e.g. CERT), and vendor exchange. After submission, each unknown file is automatically scanned by our AM products. Some files are detected by the scanners, and a small subset of the undetected files

Algorithm 1
Notation: $g(z) = \frac{e^z}{1+e^z}$, $\quad t(w,\phi,q) = w^\top \phi + \log \frac{q}{1-q}$.
Let C be the set of labeled containers
Collect Container Data: Let $S_c = \{(\Psi(N(i)), p_i, y_i) \mid i \in C\}$
where $N(i)$ is the set of executables in a container i,
and $\Psi(\cdot)$ is computed from $p_b(y_j = 1 \mid x_j)$, $j \in N(i)$
$p_i = \max_{j \in N(i)} p_b(y_j = 1 \mid x_j)$.
Train Container Classifier:
Find the vector v^* that maximizes
$L_c(v) = \prod_{(\psi,p,y) \in S_c} g(t(v,\psi,p))^{y_i} (1 - g(t(v,\psi,p)))^{1-y_i}$
Assign Container Probabilities:
For each container i: $p_c(y_i = 1 \mid x_i) = g(t(v^*, \Psi(N(i)), p_i))$
Let F be the set of labeled files.
Collect File Data: Let $S_f = \{(\Psi(N(i)), p_b(y_i = 1 \mid x_i), y_i) \mid i \in F\}$
where $N(i)$ is the set of containers containing i and
and $\Psi(\cdot)$ is computed from $p_c(y_j = 1 \mid x_j)$, $j \in N(i)$.
Train Relationship Classifier:
Find the vector u^* that maximizes
$L_r(u) = \prod_{(\psi,p,y) \in S_f} g(t(u,\psi,p))^{y_i} (1 - g(t(u,\psi,p))^{1-y_i}$
Improve File Probabilities:
For each file i: $p_r(y_i = 1 \mid x_i) = g(t(u^*, \Psi(N(i)), p_b(y_i = 1 \mid x_i)))$.

Table 1. Algorithm for Improving File Malware Probabilities.

are investigated manually by analysts for additional signature creation. In addition, we have a large collection of programs known to be legitimate (i.e. clean); many of these programs include one or more containers. Labels (i.e. "malware container", "benign container", "malware", "benign") are assigned to the containers and individual files depending on the source. Some of these labeled files are then used to train the baseline malware classifier. As part of the scanning process, container files (e.g. .zip, .rar) are uncompressed, and the individual files are extracted. A graph is constructed based on the relationships observed in the containers, and in parallel, the trained baseline classifier is used to predict the probability that each individual file is malicious. In another part of the scanning process, a file is run and any files which are dropped (i.e. written to the disk drive) are detected. These "dropped" relationships could also be used in our system. The datasets used to train the baseline and relationship classifiers differ because while all of the individual files used to train the baseline classifier have previously been labeled, most of the files in the containers have not. In the last step, the relationship malware classifier is trained using the baseline file predictions and the relationships from the graph. In the remainder of this section, we further investigate the container details and describe the baseline classifier.

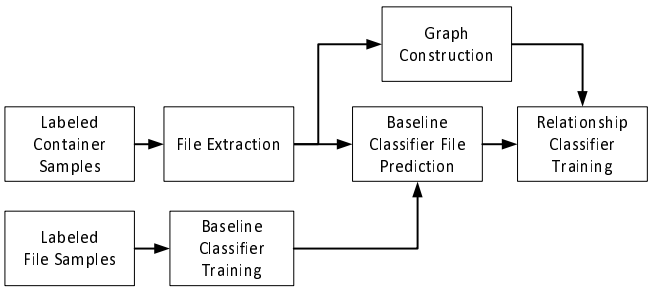

Fig. 1. System Diagram for Training the Relationship Classifier.

4.1 Container Description

To train the classifier, we first obtain labels for the containers by assigning the label "malware container" to containers which were either previously determined as malicious by an analyst, one or more contained files were labeled as malware by an analyst, or an anti-virus engine automatically detected at least one file as malware in the container. Similarly, benign archives are labeled as "benign container" and defined as those previously determined as benign by an analyst, contain no files that were labeled as malware by an analyst or other automated system, and an anti-virus engine did not automatically detect any of its contained files as malware. Next, we constructed a labeled graph containing 4,160,807 nodes and 23,993,309 edges where each node is either an archive or individual file and an edge indicates that an archive contains a file. This graph includes 719,359 total archives including 604,658 malicious and 114,701 benign containers. There are 3,441,448 individual files with 492,443 labeled as malicious and 2,949,005 labeled as benign. Among the individual files, 67,705 of them existed both in malware and in benign containers.

Figures 2 and 3 present the distributions of the number of files included in the malware and clean containers, respectively. The approximately linear relationship of the files in the malware containers on a log-log scale indicates that the number of files in the malware containers roughly follows a power law. On the other hand, many of the benign containers are distinct versions of commercially available software products including multiple versions of the same product written in many different languages. These programs often contain similar, but distinctly different, numbers of files leading to multiple archives with approximately the same number of files. This behavior is also noted for multiple versions of the same program (e.g. Adobe Acrobat Reader versions 7.0 and 7.1).

In Figures 4 and 5 we show the distribution of the number of archives that include each malware and clean file respectively. Again we observe a power law behavior for the malware, with most malware files appearing in very few containers and only a handful of malware files appearing in many containers. On the other hand for the benign files the power law behavior does not span as many orders of magnitude and instead we observe that there are several benign

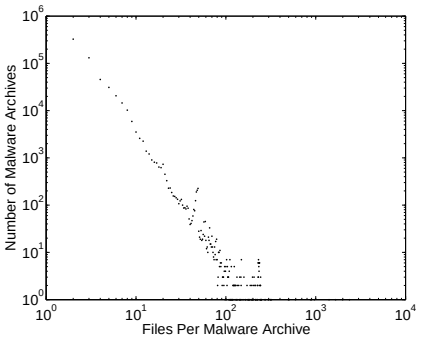

Fig. 2. Distribution of Files in the Malware Containers.

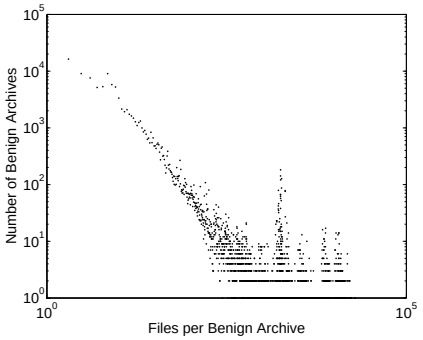

Fig. 3. Distribution of Files in the Benign Containers.

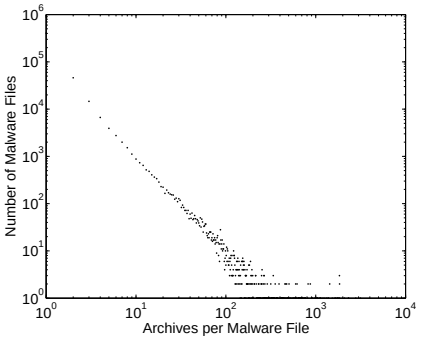

Fig. 4. Distribution of Archives which Contain Malware Files.

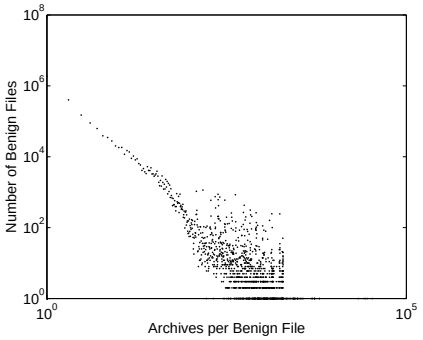

Fig. 5. Distribution of Archives which Contain Benign Files.

files that are contained in many archives. The reason for this discrepancy is that benign software is made with the intent to be reused while malware is created for more opportunistic purposes.

4.2 Baseline Classifier

In this section, we briefly describe the baseline malware classifier which provides the original probability that an unknown file is malicious or benign. We first collected over 2.6 million samples consisting of 1,843,359 malware files and 817,485 samples of files known to be benign. Each of the malicious files was also assigned to a particular malware family. We selected a set of 134 malware families determined by analysts to be important to identify. All malware samples belonging to malware families not in the set were included in a generic malware class, and all samples of legitimate software were assigned to a benign class. Next, we modified the production anti-malware engine used in Microsoft's commercial security products (e.g. Microsoft Security Essentials, Forefront Endpoint Protection) as

well as Windows (i.e. Windows Defender) to produce a set of log files for further analysis. As part of the overall analysis, this AM engine runs each unknown file in a lightweight virtual machine. We then record each system call and its corresponding input parameters. We also extracted the process memory and searched for null-terminated patterns. This collection of patterns often includes strings but sometimes include snippets of code. From these logs, we created four sets of potential features for the baseline classifier. First, we identified each distinct combination of a system call and its parameter values. For example, if the unknown executable calls CreateThread() with a stack size of 1 megabyte, this API/parameter combination would then serve as one potential feature. Next, we constructed tri-grams of the system call sequence. We also included each of the process memory patterns in the pool of potential features. Finally, we included 164 low-fidelity features from analysis of the file such as: is it 64-bit software?, is it an .EXE file?, is it a .DLL file?, and so on. In practice, this last set of features was overwhelmed by the other feature sets and only improved the model's accuracy by 0.01%. It should be noted that our features are constrained by the limitations of the production anti-malware engine. As a result, we cannot use features which require significant time to compute such as the system call graph using dynamic taint analysis. Processing the logs produced a set of over 50 million potential features which needs to be significantly reduced to avoid overfitting. Based on mutual information [13], we then used feature selection of the potential feature set to determine 179,288 features for the classifier. Finally with these selected features, we constructed a labeled training set to train a multi-class classifier with logistic regression using stochastic gradient descent (SGD) to predict if an unknown file belongs to one of the malware families under consideration or to the generic malware or benign class. We chose to train a logistic regression model with SGD because of the large scale nature of our data in terms of both the number of samples and features. Due to our implementation in C# and a .NET memory constraint related to the number of elements in a list, we used SGD to efficiently train the baseline classifier in mini-batches of roughly 450 thousand examples. Training the multi-class classifier produced a false positive of 1.3% and a false negative rate of 0.7% on a separate (i.e. hold-out) test set of over 443 thousand files. Even though the features used in this classifier are relatively simple, training with over 2.6 million files produces a good error rate. Furthermore, malware classification research has produced a number of independent algorithmic improvements to increase malware classification accuracies [18, 17, 15, 2, 11, 16, 6]. As we argue in Section 8, since the baseline classifier and container relationship structure are independent, applying one or more of these algorithmic improvements or additional feature sets to the baseline classifier will help to improve the overall system response.

5 Experimental Results

In this section, we conduct a series of experiments to evaluate the performance of the container and improved relationship malware classifiers.

Name	Determination	# Scanner Detections	# Submissions
. . . _Norton _Antivirus . . . _2007 _.rar	Malware Container	15	2
. . . ba52.bin	Malware Container	15	4
. . . _z0ffzvk _.rar.part	Malware Container	14	2
. . . _dc11.rar	Malware Container	14	2
. . . _regcure _1.0.0.43.1.3a1400.efw	Malware Container	14	2
. . . _Registry _Mechanic . . . .rar	Malware Container	14	2
. . . _CyberLink _PowerDVD _7.0.rar	Malware Container	15	2
. . . _adobe _photoshop _cs2 _.rar	Malware Container	15	2

Table 2. An Example of the File Containers which Include `2.exe`.

5.1 Examples

We first motivate the relationship classifier idea by examining how it affects the baseline probability of two individual files. Upon manual examination of the first file `2.exe`, we found this file to be a variant of a Trojan in the Vundo family. Table 2 indicates the file was included in 8 containers, which were labeled "Malware Container". This table includes the names of the containers, their determinations, the number of scanners which detect them, and the number of submissions. The third column indicates all containers were detected by at least 14 scanners. For this example, the baseline malware classifier failed to correctly identify the file as malicious. The relationship classifier raised the probability of this file from a baseline of 33% to 98.37% which is more indicative of malware. This shows that the malware relationship classifier can help to correctly identify malicious files even when the baseline classifier misclassifies them.

The second example involves a file named `calleng.dll`. The file was manually determined to be benign by an analyst, and the baseline malware classifier assigns a probability of nearly 0% that this file is malware. We scanned it with a set of anti-malware scanners and no scanners detected the file. This file was originally distributed as part of the legitimate `PalTalk` social networking software. Table 3 provides the container relationships. We believe that `(RarSfx)` on row 4 with no detections is the legitimate `PalTalk`. We also have evidence from the scanner detection column that the remainder of the containers in Table 3 are indeed suspicious. In fact, we believe that these are malicious versions of the original `PalTalk` application. While `calleng.dll` itself is not malicious, it clearly appears to be commonly used by malware authors in some manner. In other words, a previously unseen archive containing this file is likely to be malware. After running the malware relationship classifier on `calleng.dll` the malware probability increased to 44.9%. While this is certainly more indicative of being malicious, it is not sufficient to be classified as malware. This shows that even when the new probability estimates of a benign file are increased, this is usually not enough to cause a false positive. This is confirmed by the overall improved results in Figure 7.

Name	Determination	# Scanner Detections	# Submissions
0d...bc.rar	No Determination	13	2
d3...39.rar	No Determination	9	2
ec...da	No Determination	3	2
(RarSfx)	No Determination	0	2
(RarSfx)	No Determination	7	4
(RarSfx)	No Determination	9	4

Table 3. An Example of the File Containers which Include `calleng.dll`.

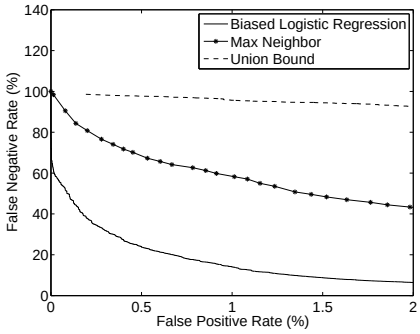

Fig. 6. DET Curves for the Container Classifier Algorithms.

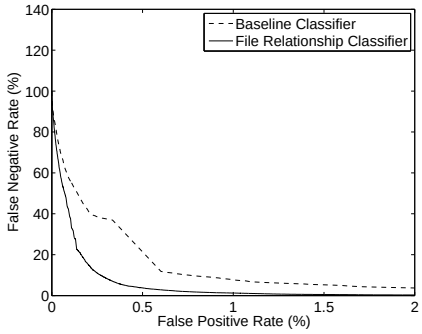

Fig. 7. DET Curves for the Baseline and Relationship File Classifiers.

5.2 Performance Analysis

Detection Error Tradeoff (DET) curves for the three container classification algorithms proposed in Section 3.5 are plotted in Figure 6. To obtain probabilities for all containers with the container classifier in a fair way we used 5-fold cross validation. If the DET curve of a rule is always below the DET curve of another rule then we say that the former *dominates* the latter. It means that for all possible FP rates one classifier is always achieving better FN rates than another; a very strong statement. For malware detection, we care about the region of small FP rates, and in the figures we are zooming in a range of up to 2% FP rate. In this range, the Biased logistic regression algorithm completely dominates the simple approaches of the Max neighbor and Union bound algorithms. In fact, the Union bound method is dominated by the Max neighbor rule which demonstrates the inappropriateness of its assumptions. This leads us to believe that the files in the containers are correlated and reinforces our belief that aggregating information across the files in the container is not trivial. We mention that the Biased logistic regression method dominates the other two rules across all the FP rates, not just across the range shown in Figure 6.

In Figure 7 we present DET curves for individual files. We compare the existing baseline classifier with the relationship classifier of Section 3.6, and we again employ 5-fold cross-validation. As before, for malware classification we are

FP Rate	Label	$p_b \leq t_b$ $p_r \leq t_r$	$p_b > t_b$ $p_r \leq t_r$	$p_b \leq t_b$ $p_r > t_r$	$p_b > t_b$ $p_r > t_r$
1.0%	Malware	6269	161	32170	480548
	Benign	2909583	15561	14590	14959
0.1%	Malware	183454	15406	109043	211245
	Benign	2950180	1556	1546	1411

Table 4. Comparison of Baseline and File Relationship Classifier Statistics for FP Rate = 1% and 0.1%.

interested in small FP rates and therefore plot the curves for FP rates up to 2%. We observe that the relationship classifier convincingly dominates our baseline system: At an FP rate of 0.3%, there is a 77.3% decrease in FN rate (from 37.5% to 8.5%). At our target FP rate of 0.01%, the decrease in the FN rate from 87% to 85% is marginal; there is still more work to be done to achieve widespread detection at these extremely low FP rates. In the rest of the FP range (not shown), the baseline system remains dominated until approximately an FP rate of 60%, when it becomes marginally advantageous to use the baseline system. However, these operating points are uninteresting even for perimeter-based anti-malware systems. Our conclusion is that our approach not only improves upon the baseline system, but it does so for error tradeoffs that are important for our application. In Section 5.3 we explain why (and this is true only for logistic regression) the results of Figures 6 and 7 are invariant to the proportion of malware files we used in the experiments.

In Table 4 we next compare example counts for the baseline and relationship classifiers for different threshold values corresponding to FP rates of 1% and 0.1% . For the row labeled malware in Table 4 with a FP rate of 1%, the sixth ($p_b > t_r, p_r > t_r$) and third ($p_b \leq t_b, p_r \leq t_r$) columns indicate that both classifiers correctly identify 480,548 malicious files but fail to detect 6,269 files. The fourth column indicates that 161 files were incorrectly mispredicted by the relationship classifier as benign (FN) but correctly predicted by the baseline system. The fifth column shows the improvement of the relationship classifier for 32,170 files which were mispredicted by the baseline classifier. For the second row with benign files at 1% FP rate, 15,561 files were correctly predicted to be benign by the relationship classifier but missed by the baseline classifier. Likewise at an FP rate of 1%, the relationship classifier had 14,590 false positives. Similar statistics are noted for a 0.1% FP rate. Note that the baseline classifier threshold (t_b) and relationship classifier threshold (t_r) differ in order to set the operating point to the desired FP rate. Also, the number of FPs for the baseline classifier ($p_b > t_r, p_r \leq t_r$) and for the relationship classifier ($p_b \leq t_b, p_r > t_r$) are approximately equal, but differ slightly due to multiple examples having identical predicted probabilities.

In Figures 8 and 9, we further investigate the 14,590 FPs generated by the relationship classifier at a FP rate of 1%. Figure 8 shows a histogram of the FPs that are found in more than 10 containers. This figure accounts for approximately two thirds of our FPs. The FPs have been binned according to the fraction of associated containers that are malware containers. We see that a large number of

our FPs are associated with containers most of which contain malware. In fact, 31.5% of our FPs are files found in containers in which the majority of files are malware. Figure 9 shows a heatmap for the rest of the FPs, i.e. those found in 10 or less containers. Here we see that the overwhelming majority are associated with only one container which contains no malware.

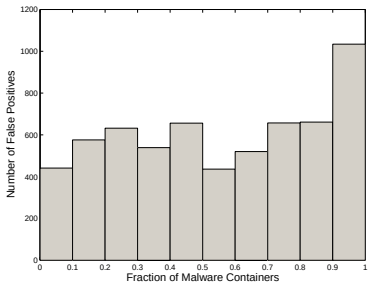

Fig. 8. FPs Binned by the Fraction of Malware Containers Out of All Archives Containing Each.

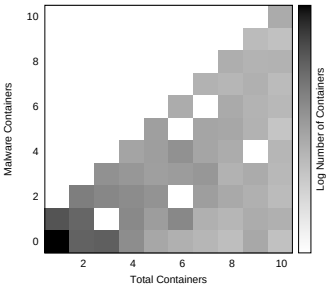

Fig. 9. Heatmap of Containers Associated with False Positives.

5.3 Validity of Results Under Biased Sampling

Can the conclusions drawn from Figures 6 and 7 reflect the real world behavior of our classifier where the proportion of malware files will be different than the one we used for training? As we explain, this is true for logistic regression. First, notice that among each of the two classes, malware and benign, sampling was random. So we can say that by selecting many malware files we have just uniformly increased the probability of malware by a factor $c_1 > 1$ and have uniformly decreased the probability of benign by a factor $c_0 < 1$:

$$p(y_i = 1|x_i, \text{selection}) = c_1 p(y_i = 1|x_i)$$

$$p(y_i = 0|x_i, \text{selection}) = c_0 p(y_i = 0|x_i).$$

Our logistic classifiers model the log odds so we get

$$\log \frac{p(y_i = 1|x_i, \text{selection})}{p(y_i = 0|x_i, \text{selection})} = \log \frac{c_1}{c_0} + \log \frac{p(y_i = 1|x_i)}{p(y_i = 0|x_i)}.$$

In other words, the log odds we compute are indeed inflated by $\log \frac{c_1}{c_0}$. However *the effect of* $\log \frac{c_1}{c_0}$ *is constant across all files.* On the other hand the DET curves of Figures 6 and 7 only depend on the *order of the files*, according to their probabilities. This order is not affected by adding to each file the constant $\log \frac{c_1}{c_0}$. Hence we would have obtained the same figures even if we had trained our logistic classifier with a sample that contained the correct proportion of malware.

6 Discussions

In this section we discuss the assumptions of our system, the constraints under which it needs to work, and investigate some issues related to the notion of suspicious but not necessarily malicious files and containers. Our work is based on two main assumptions. First the relationships we manage to extract from the files contain useful information that is not already captured by our current baseline classifier. The second assumption is that we can extract some of this information with our current representation. To avoid being detected by our approach, a piece of malicious software has to first score low or moderately when scanned with the individual classifier and then has to be associated only with containers that themselves are not suspicious. Assuming that the malicious file bypasses the individual classifier, the best strategies to avoid detection by the relationship classifier is to submit the file by itself or with another previously unseen file that is undoubtedly benign. Currently, our relationship classifier will not do anything to files it cannot directly relate to previously seen files. For such a file we could first find an approximate match (using, say, locality sensitive hashes [1]) and use its relationships. Though such an approach would further reduce opportunities for code reuse and "malware libraries" for malware authors, it is beyond the scope of this paper. Our focus is to demonstrate that even a simple approach has immediate benefits.

A very limiting constraint with respect to the solutions we could employ for our task is the requirement for our system to be highly scalable. We have therefore only considered linear models simply because we cannot afford to train (or even run) more complicated models on the millions of executables in our database. Among the linear models we only presented results for logistic regression, but other methods like SVMs should perform similarly.

Even though the results in Section 5 demonstrate large improvements, our model is not perfect. Table 4 and Figures 8 and 9 indicate our model is still susceptible to FPs which are particularly worrisome to analysts. However while statistical models are subject to false positives in general, we believe that the definition of a false positive is not correct in a portion of these cases. Typically, analysts assign a "benign" label to files which cannot infect a computer. We argue that even if a file cannot be infectious it can still be suspicious if, say, we have only seen it co-occurring (i.e. being part of the same container) with files that have been determined to be malware. To handle this case, we believe that the model can be further improved by adding a third label, "malware related", to any file which cannot infect a computer by itself, has been found in some relationship with malware (contained, dropped), and has never been observed in any files encountered during the installation of a legitimate software package.

Finally, the algorithm we proposed in Table 1 can be thought of as one iteration of a more complicated scheme. Though we already argued that iterating this algorithm is tricky, it could provide useful information that is currently not captured by our model. For example, two iterations of our algorithm would capture information about co-occurence patterns of files in the same archives and would start building archive reputations.

7 Related Work

Malware classification has been a rich area for research, and Idika *et al.* provide a recent survey of the literature [9]. Early efforts on malware classification such as those of Schultz *et al.* [20] and Kolter *et al.* [12] focused on static analysis of the executable files. Features based on n-grams of byte sequences have been used in [18, 19]. In [21] the authors perform sequence analysis of system calls and Christodorescu *et al.* propose detecting malware based on the semantics of the unknown file [5]. Results show improved detection of obfuscated malware compared to commercial anti-virus products. Chouchane *et al.* [4] develop static classifiers for metamorphic malware, by computing probability distributions over metamorphic variants and using them to train a classifier. Perdisci *et al.* [17] proposed using boosting to classify malware.

Recently, a number of authors have proposed behavior-based malware detection systems. For example in [15] the system executes malware in a virtual machine allowing the execution of arbitrary programs at each control flow decision point. This allows the tool to explore, record, and report the complete behavior space of a program. Instead of using a simulated virtual machine for monitoring program behavior, Bayer *et al.* [2] developed a tool where a complete operating system is run in software thereby allowing the identification of malware that terminates after detecting that it is running in a virtual machine environment. Mehdi *et al.* [14] have previously used N-grams of system calls for a malware classification system.

Few papers have explored using graph-based methods for detecting malware. For example, [6, 8, 10] classify or cluster the call-graphs of malware and benign programs. Recently, Chau *et al.* [3] explored building file reputation based on a bipartite graph of applications and machines. Finally, Ye *et al.* [22] built and deployed a system that combines individual predictions with a different definition of file relationships. They consider two files related if they co-occur on a set of client machines. In contrast, we define our file relationships at the time a file is submitted to our service. Besides avoiding thorny privacy issues, the relationships we use are much more localized and reflect pieces of information that the human analysts actually seek when analyzing an unknown sample. In accordance to our experimental results, they also observe a large benefit by moving beyond individual file classification.

8 Conclusions

Automated malware detection is critical given the explosion of new malware in recent years. In this paper we investigate a novel way of improving malware classifiers by going beyond individually classifying a given file. Instead, we take advantage of the information that exists in the *relationships* between the files submitted to our service; information that is already being leveraged by human analysts in their job. Starting from a baseline individual file classifier we proposed three ways to propagate information from files to containers (Max neighbor,

Union bound and Biased logistic regression) and a relationship classifier which uses this propagated information to improve the baseline probabilities.

Our experiments show that on one hand simple approaches like the Union bound fail completely, and hence propagating and aggregating the relationship information is not a trivial task. On the other hand our proposed Biased logistic regression classifier completely dominates Max neighbor. Furthermore, using the container probabilities from Biased logistic regression the relationship classifier substantially reduces the FN rate at small FP rates.

In spite of these encouraging results, more algorithmic improvements are required to rely solely on automated malware classification to block or detect new malware. Improving the baseline classifier, which is relatively simple, can further improve the relationship classifier. In fact we have presented some evidence that large fractions of what appear to be false positives a) cannot be fixed by the relationship classifier and might be due to the baseline system and b) are not exactly false positives because they are files mostly found in malware containers. In any case, we believe that our modular approach of learning a correction to the baseline system nicely decouples the problem in two orthogonal components: one that looks at the file individually and one that looks at the file's relationships. This way, progress in either of these fronts can further improve the overall performance of our system.

Acknowledgment

The authors would like to gratefully thank Dennis Batchelder for helpful discussions and support.

References

1. Andoni, A., Indyk, P.: Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. In: Foundations of Computer Science, 2006. FOCS'06. 47th Annual IEEE Symposium on. pp. 459–468. Ieee (2006)
2. Bayer, U., Kruegel, C., Kirda, E.: TTAnalyze: A tool for analyzing malware. In: Proceedings of 15th Annual Conference of the European Institute for Computer Antivirus Research (EICAR) (2006)
3. Chau, D., Nachenberg, C., Wilhelm, J., Wright, A., Faloutsos, C.: Polonium: Tera-scale graph mining and inference for malware detection. In: Proceedings of SIAM International Conference on Data Mining (SDM) 2011 (2011)
4. Chouchane, M., Walenstein, A., Lakhotia, A.: Statistical signatures for fast filtering of instruction-substituting metamorphic malware. In: Proceedings of the Workshop on Recurring Malcode (WORM'07). pp. 31–37 (2007)
5. Christodorescu, M., Jha, S., Seshia, S., Song, D., Bryant, R.: Semantics-aware malware detection. In: Proceedings of the 2005 IEEE Symposium on Security and Privacy. pp. 32–46 (2005)
6. Fredrikson, M., Jha, S., Christodorescu, M., Sailer, R., Yan, X.: Synthesizing near-optimal malware specifications from suspicious behaviors. In: IEEE Symposium on Security and Privacy. pp. 45–60 (2010)

7. Golub, G.H., Loan, C.F.V.: Matrix Computations. The Johns Hopkins University Press, 3 edn. (1996)
8. Hu, X., Chiueh, T., Shin, K.G.: Large-scale malware indexing using function-call graphs. In: ACM Conference on Computer and Communications Security. p. 611620 (2009)
9. Idika, N., Mathur, A.: A survey of malware detection techniques. Tech. rep., Purdue Univ. (February 2007), http://www.eecs.umich.edu/techreports/cse/2007/CSE-TR-530-07.pdf
10. Kinable, J., Kostakis, O.: Malware classification based on call graph clustering. In: Journal in Computer Virology. pp. 33–45 (2011)
11. Kirda, E., Kruegel, C.: Behavior based spyware detection. In: Proceedings of the 15th USENIX Security Symposium). pp. 273–288 (2006)
12. Kolter, J., Maloof, M.: Learning to detect and classify malicious executables in the wild. In: Journal of Machine Learning Research. pp. 2721–2744 (2006)
13. Manning, C.D., Raghavan, P., Schutze, H.: An Introduction to Information Retrieval. Cambridge University Press (2009)
14. Mehdi, S., Tanwani, A.K., Farooq, M.: Imad: in-execution malware analysis and detection. In: Proceedings of the 11th Annual conference on Genetic and evolutionary computation. pp. 1553–1560 (2009)
15. Moser, A., Kruegel, C., Kirda, E.: Exploring multiple execution paths for malware analysis. In: Proceedings of the IEEE Symposium on Security and Privacy (SP'07). pp. 231–245 (2007)
16. Moser, A., Kruegel, C., Kirda, E.: Limits of static analysis for malware detection. In: Proceedings of the 23rd Annual Computer Security Applications Conference (ACSAC'07). pp. 421–430 (2007)
17. Perdisci, R., Lanzi, A., Lee, W.: Mcboost: Boosting scalability in malware collection and analysis using statistical classification of executables. In: Proceedings of the 2008 Annual Computer Security Applications Conference (ACSAC). pp. 301–310 (2008)
18. Reddy, D., Dash, S., Pujari, A.: New malicious code detection using variable length n-grams. In: Information Systems Security. pp. 276–288 (2006)
19. Reddy, D., Pujari, A.: N-gram analysis for computer virus detection. In: Journal in Computer Virology. pp. 231–239 (2006)
20. Schultz, M., Eskin, E., Zadok, E., Stolfo, S.: Data mining methods of detection of new malicious executables. In: Proceedings of the 2001 IEEE Symposium on Security and Privacy. pp. 38–49 (2001)
21. Sung, A., Xu, J., Chavez, P., Mukkamala, S.: Static analyzer of vicious executables (save). In: Proceedings of the 20th Annual Compter Security Applications Conference (ACSAC'04). pp. 326–334 (2004)
22. Ye, Y., Li, T., Zhu, S., Zhuang, W., Tas, E., Gupta, U., Abdulhayoglu, M.: Combining file content and file relations for cloud based malware detection. In: Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining. pp. 222–230. ACM (2011)