Separation of Benign and Malicious Network Events for Accurate Malware Family Classification

Hesham Mekky*, Aziz Mohaisen† and Zhi-Li Zhang*
*University of Minnesota. Email: {hesham,zhzhang}@cs.umn.edu,
†SUNY Buffalo. Email: mohaisen@buffalo.edu

***Abstract*—Labeling malware samples with their appropriate malware family helps understand and track malware evolution and develop mitigation techniques. Current malware analysis techniques that use supervised machine learning rely on classification models that are trained on malware traffic generated from a sandbox environment. These models are then used to classify future unseen observations. In practice, however, malware traffic comes mixed with other legitimate "background" traffic from host machines, such as user browsing and software update traffic. Hence, the classifier's accuracy to predict the correct malware label on unseen (mixed) traffic is low. We propose a novel classification system that uses an Independent Component Analysis (ICA) module that applies distribution decomposition to separate the observed traffic into two components, malware traffic and background traffic. We also use a random forest classifier module to learn a classification model for every malware family, and then use it to predict malware family labels using the output of the ICA module. This system is thus capable of labeling malware traffic after removing background artifacts ("noise"), which makes it more efficient and accurate than current classification methods. Our experiments on three malware family datasets show that the performance of our system improves significantly after removing the background traffic artifacts.**

I. Introduction

Malware analysis, classification and labeling is a well-investigated problem in the cyber security community. Techniques used for malware analysis fall into two categories: static analysis and dynamic analysis. Static techniques utilize meta-data associated with malware binaries, including specific patterns (signatures) in the binary itself, whereas dynamic techniques utilize artifacts generated by the malware at runtime, including memory access patterns, network traces, OS system calls, file system changes, and registry modifications. Static analysis is fast and scalable, since it searches for the precomputed signatures in the given binary (usually as regular expressions). However, it requires costly reverse engineering efforts for obtaining signatures and roles from malware binaries. In addition, attackers can defend against static analysis methods using code obfuscation techniques. On the other hand, dynamic analysis methods are highly accurate, and capable of detecting previously unseen malicious behavior, while addressing code obfuscation, but they cost more time and resources to run the given malware sample in the sandbox environment [1]. In addition, one circumvention mechanism utilized by malware authors is "behavior-poisoning" in which malware generates random noise to disguise its real behavior.

A class of methods leverages machine learning models to produce labels (malware family labels) for network traces based on statistical features extracted from either static binaries [2–4], or from artifacts collected during the execution of the binary [5, 6]. These traces and artifacts come from executing malware in a sandbox environment to avoid any damage to the host machine, network or external resources. Consequently, these statistical features are based on "clean" traces that do not contain any "background" traffic generated by a user machine such as user browsing, OS update or other background software activities. This reduces the classifier's effectiveness in predicting malware labels, when applied to the same features extracted from mixed traffic that is collected on user machines, since these machines generate traffic containing both malware- and background-related traffic.

Our proposed method aims at separating legitimate background traffic attributes generated by user activities, OS updates, background software, etc. from the malware traffic attributes. We utilize Independent Component Analysis (ICA) [7] to separate distributions of features extracted from mixed network traces into two estimated distributions: an estimated malware traffic distribution and an estimated background traffic distribution ("noise") for every feature in our set of features. Then, we use the estimated malware feature distributions as input to our classifier to predict proper malware labels. This classifier learns from the feature distributions extracted from the pure (i.e., not mixed) malware traces collected in a sandbox environment. In other words, we remove the noise in the feature distributions before applying the classifier model. More specifically, our proposed system is composed of two main modules: an ICA separation module and a classifier module. We use these modules in two phases as follows:

- *Learning Phase:* First, we use the network traces (PCAP files) extracted from the execution of malware samples belonging to three different malware families in a sandbox environment. We transform these lower level network traces (packets) into higher level network events to capture the higher level semantics of the malware operation (§II-C). Second, we extract a set of feature distributions that best represents each malware family using n-gram analysis [8] on the higher level network events. The n-gram analysis preserves the order of seen events, which captures the semantics of ordering between events for the given malware family in the feature set (§IV). Third, we use our malware feature distributions to build a random forest classifier model [9] for each mal-

ware family. In addition, using both malware traffic and a background traffic dataset, we construct an ICA unmixing function that separates a mixed feature distribution into two estimated distributions: a malware distribution and a background distribution (§III).

- *Prediction Phase:* We develop a supervised machine learning framework to produce labels for unseen malware samples using the ICA unmixing function and the random forest classifier. First, the feature vector is extracted from the malware sample. Then, the ICA module estimates the malware feature distributions from the mixed feature distributions, and finally the classifier predicts a malware family label using the estimated distributions (§III).

We apply our framework to real network traces collected from executing malware samples in three different families, and show improvements over directly using the network traces without removing the noise in the feature distributions (§V).

In the following sections, we present the necessary preliminaries (§II) and our datasets (§II-C). Then, we present our system overview (§III), followed by our feature analysis and extraction methods (§IV). Then, we show the effectiveness of our method by applying it to three different datasets (§V), followed by related work (§VI), and a discussion (§VII). Finally, we conclude our paper and present future work (§IX).

II. Preliminaries

In this section, we present the preliminaries required throughout the paper. We start by reviewing the basics of behavior-based analysis for malware detection used in this paper in §II-A. We follow that by a review of Independent Component Analysis (ICA) in §II-B, which is the method used for separating the malware traffic from the background traffic in our system. Finally, we briefly present the process used for collecting our malware datasets and transforming this data to higher level network events in §II-C.

A. Behavior-based Analysis

Behavior-based approaches for malware analysis identify and characterize malware by relying on artifacts generated during the malware execution in a sandbox environment. They characterize malware samples based on the way these samples use the host file system, memory, registry, and network. Techniques that utilize these approaches extract features from different behavioral artifacts and footprint of various malware samples based on their malware family association. These approaches are known for their accuracy, since they address the shortcomings of static analysis techniques such as obfuscation and code repacking. They are also agnostic to the underlying code, and thus can address code polymorphism. Moreover, behavior-based approaches and the methods built on top of them rely on features that are easier to understand in relation with the studied malware's context.

While generic behavior-based analysis techniques utilize a large array of behavioral attributes (e.g., memory, file system, registry, and network behavior), they come at a high cost [10] due to the time and resources they require to execute and collect various artifacts. To this end, we focus only on network-related behavioral analysis and features. In particular, we use network events such as connections, their sizes, types, and port attributes as the main attributes for our malware behavioral characterization (as we will explain later in §II-C).

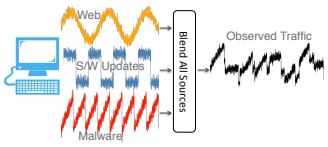


Fig. 1. Observed Traffic from a Host

Ideally, our system should run along with other processes that would generate the background traffic on the same host. However, for testing purposes of our system and techniques explored in it, we run our system building blocks, including the behavior profiling part, in a supervised environment. We use a similar setup to that used in [10] to run various malware samples that belong to a given family, extract representative features to the family, and collect artifacts associated with the background traffic (as we will explain in §IV). We then study the parameters of our system and their effectiveness in identifying malware samples, and isolating background traffic.

B. ICA Primer

The traffic observed from a user host is a blend of different types of traffic, as illustrated in Figure 1, where each "*signal*" represents a different type of traffic, e.g., the *blue signal* represents the software updates traffic. These types are mixed together in the "observed" traffic (*black signal*). Consequently, the accuracy of a method that uses the malware traffic (*red signal*) only to extract features and build a classifier to predict the malware label on unseen, mixed traffic will perform poorly, as we will show in §V. Therefore, we need an algorithm that learns the malware features (*red signal*), and extracts these features from the mixed traffic features (*black signal*). ICA is capable of doing this based on two main assumptions:

(1) Mixed signals are linearly mixed. This fits our application since the distribution of a feature in the observed mixed traffic is going to be a linear mix of the distributions of the same feature in the input traffic sources, e.g., the distribution of the number of successful DNS requests in the mixed traffic is a linear sum of the distributions of the same feature in the input sources.

(2) Mixed signals are statistically independent. Here, we assume that the malware traffic is independent from other types of traffic such as software updates, user activity, etc. For instance, we assume that the number of failed DNS requests for the malware is independent from the number of failed DNS requests in the background traffic.

In a nutshell, ICA decomposes a multivariate signal into additive components assuming statistical independence. Assuming m independent source signals $S = [S_1, ..., S_m]^T$, then we observe the mixture $X = [X_1, ..., X_m]^T$ given by

TABLE I
DATASETS

Family	Samples
Darkness	534
Shady RAT (SRAT)	1,096
Zeus	1,025

$X = A \cdot S$, where A is called the mixing matrix. The goal of ICA is to find an unmixing matrix $W(\approx A^{-1})$ such that $Y = W \cdot X \approx S$ will be the best approximation for S. Therefore, when ICA learns the unmixing matrix for a specific feature (from the malware and background traffic), it will be able to decompose future unseen mixed traffic and extract the feature distribution for the malware component.

ICA algorithms rely on independence to recover the original signals from the mixture. For instance, given two signals X and Y: (i) Entropy $H(X)$ is a measure of uncertainty in X, i.e., the lower the entropy, the more information we have about X, (ii) Conditional entropy $H(X|Y)$ is the amount of uncertainty in X after observing Y, (iii) Mutual information, $I(X;Y) = H(X) - H(X|Y)$, is the reduction of uncertainty in X after observing Y.

Therefore, by having an algorithm that minimizes the mutual information between the estimated components [11], we are looking for latent variables that are maximally independent, i.e., in our application system, we are looking for the two underlying independent distributions (malware and background distributions) in the mixed feature distribution.

C. Datasets

Our datasets are composed of three malware families and a clean dataset. These three malware families cover a wide range of network behavior for our evaluation purposes. Each malware sample is labeled manually using an operational product at VeriSign [1]. These datasets are: (a) the Darkness malware family [12], which infects machines to carry out DDoS attacks, (b) the Shady RAT (SRAT) malware family [13], which infects machines to target high profile organizations (credentials stealing, DDoS attacks, etc), and (c) the Zeus malware family [14], which is a banking Trojan that is used by attackers to run a botnet to steal money, credentials, and system resources from the infected victims and their machines, (d) the clean dataset includes traces from clean hosts generating regular traffic.

Each sample is executed in a controlled environment for a predefined amount of time to collect network artifacts generated solely by each malware family in the form of PCAP traces. Details of each dataset are shown in Table I. In addition to the three malware families, we use the clean dataset to resemble the background traffic behavior generated from regular hosts such as web browsing, OS updates, etc.

We use the PCAP trace generated by each sample to create a profile for each malware sample based on the ordering of events in the packet trace [10]. Profiles are based on network events seen in the packets, e.g., an outbound packet using UDP on port 53 is mapped to "A0A2A5", where A0 refers to the outbound event occurrence, A2 refers to the UDP protocol usage, and A5 refers to the usage of port number 53. Consequently, the *word* "A0A2A5" in a malware sample profile indicates an outbound DNS query over UDP. We apply the same mapping to all packets in the trace. After this process, each malware sample is represented as a sequence of words, where each word is a meaningful network event. This results in good features that ICA can use to learn to unmix the traffic, and the classifier can use to predict the malware family label as shown in §IV and §V. Table II summarizes all network events we consider in our analysis. Request and response quartiles are normalized sizes over all samples, i.e., we compute the largest request/response size and normalize all other request/response sizes into one of the four quartiles (0–25, 26–50, 51–75, and 76–100). This event captures the loudness of a malware family, e.g., if the majority of requests is in the fourth quartile, then this family sends big requests in most of its connections. The rest of the events in the table is self-explanatory.

TABLE II
NETWORK EVENTS

Network Event	Description
Connection	TCP, UDP, RAW
Size	request quartiles, response quartiles
DNS	A, NS, MX, CNAME, SOA, PTR
Request type	GET, POST, HEAD
Response type	200's, ..., 500's
Ports	80, 8080, 53, and others

III. SYSTEM OVERVIEW

Our system consists of two main modules: an ICA separation module and a malware classification module. An illustration of the system is shown in Figure 2. The system takes the traffic artifacts of a malware sample collected from a user host (malware and background traffic) as input, and ultimately determines the label of a malware family association for the sample. Our system marginalizes the noise due to the background traffic and learns the features to be fed into the malware classification module. The classification module uses a knowledge base of known malware families with respect to the studied features, and assigns a probability that the examined malware sample belongs to each family. The decision component then decides the family to which the malware sample belongs based on these probabilities. In the following, we explain what each module does in each of two phases: learning and prediction.

A. ICA Separation Module

Learning Phase: In this phase, we build the ICA unmixing matrix that is capable of separating the two underlying distributions of a given mixed feature distribution as discussed in §II-B. For each feature in the top discriminating features between a malware family and the background traffic, ICA learns the unmixing matrix from the mixed feature distribution. From §II-B, it is clear that ICA can learn many unmixing matrices for the same mixed distribution. We leverage the knowledge

of the malware family distribution (i.e., ground-truth malware samples) to learn a good unmixing matrix. For instance, in Figure 1, we feed the observed traffic feature distribution (blue signal) into ICA and learn the unmixing matrix to find the malware feature distribution (red signal). Then, we compute the Kullback-Leibler Divergence[15] between the estimated distribution and the ground-truth distribution. If this distance is below a certain threshold, then we use this unmixing matrix for that feature, otherwise, we run ICA again to learn a better unmixing matrix. The performance of ICA in estimating the feature distributions is evaluated in §V-A.

Prediction Phase: Given an unseen mixed sample, we extract the feature distributions corresponding to each malware family in our system. Then, we feed these mixed distributions to the learned ICA model. The output of this model is the estimated underlying malware families feature distributions, which we use later in the classification module. In Figure 2, the box labeled ICA contains all the unmixing functions for different malware families learned previously. Thus, the output of the "ICA" box to "Malware1 Classifier" box is the estimated malware 1 feature distributions that are then used to check if this sample belongs to this malware family.

B. Classification Module

Learning Phase: We utilize the ground-truth of the three malware families, and use the top discriminating features to build a random forest classifier [9] for each family using these features. Then, we use the classification models to predict proper labels to future, unseen traffic. Other classification techniques can also be used to generate a probabilistic score instead of a 0/1 label.

Prediction Phase: Here, the classifier uses the output of the ICA module (i.e., estimated features for different malware families), and applies the learned classifier model for malware family *i* on the estimated feature distributions for malware family *i*. The output of each classifier model is a probability that the estimated features of the studied malware sample belongs to a certain family. The decision function determines the family that the malware sample belongs to by selecting the label with the maximum classification probability obtained from the classifier. Notice that we only use a single family association, and our system provides excellent operational results based on this decision. However, this component may also generate a set of labels (e.g., top-k or labels with an over-threshold probability). These labels for a sample can be used to establish similarities among families with respect to the studied features, as well as a behavior-based relationship between different malware families.

While the three families studied in this paper are only for the purpose of demonstrating the concept of our system and highlight the capabilities of ICA, in-lab experiments show the validity of our study on other a wide range of families; our classifier and ICA modules are vertically scalable and extensible to other families that respect the ICA assumptions.

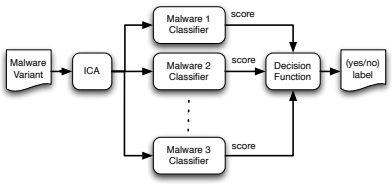


Fig. 2. Malware Labeling Process

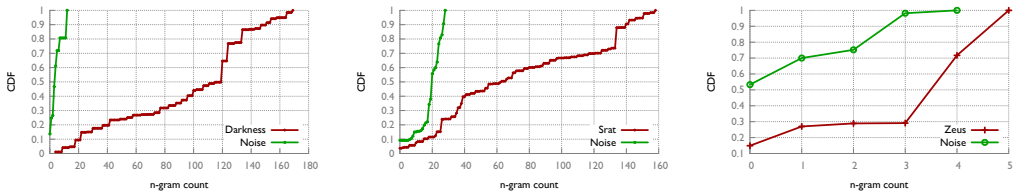
IV. Features Analysis

Based on the discussion in §II-C, each malware or background traffic sample is represented as a sequence of words, where each word indicates the occurrence of a particular network event, e.g., a successful DNS query. Consequently, the words in a malware sample correspond to the two-way communication of the malware with the remote C&C server, or the entity being attacked, e.g., DDoS or botnet infected node. In this section, we present the feature extraction method used to select discriminating features between the malware and background traffic for all malware families (§IV-A), then we present our feature interpretation (§IV-B), and our feature evaluation criteria (§IV-C).

A. Feature Extraction

We use n-gram analysis [8] on the final representation of the data, i.e., words that represent network events, to capture relevant network events in both malware and background traffic. We choose n-gram analysis since it captures the ordering of events presented in the given malware sample, e.g., the distribution of the frequencies for a triple gram *(w1, w2, w3)* indicates the prevalence of these three events in the data, where *(w1, w2, w3)* may correspond to the occurrence of a DNS resolution followed by an HTTP GET request, then an HTTP GET response. Therefore, n-gram analysis can pinpoint interesting network events that can be used as distinguishing features between malware and background traffic. For each malware family, we generate n-gram frequencies for different values of *n* as our initial feature set for that malware family. Then, we eliminate the less discriminating features using recursive feature elimination [16], and we use the top features to learn the ICA unmixing function, and to train the random forest classifier in the *Learning Phase* as discussed in §III. We present performance results for different values of *n* in §V. Our analysis unveiled three classes of discriminating features: malware discriminating, background discriminating, and common features in both malware and background traffic.

Malware Discriminating Features: We observed a set of n-gram distributions, where the malware behavior is governing over the background behavior, i.e., these specific n-grams show up more frequently in the malware data. For instance, Figure 3a shows the CDF distribution for the n-gram frequencies for a specific n-gram feature in the Darkness malware. The x-axis is the frequency of this n-gram in the Darkness data (i.e., the number of occurrences of this n-gram in malware samples belonging to the Darkness family), and the y-axis is



(a) Darkness (b) Srat (c) Zeus

Fig. 3. A selected discriminating feature from each malware family, where the malware behavior is governing the background behavior. Each subfigure is the CDF for a specific n-gram frequencies over all malware and background traffic samples.

the percentage of samples that have less than or equal to that frequency. We can see that 90% of the background data has less than 10 occurrences of this feature, while only 0.05% of the malware data has less than 10 occurrences, which indicates that this n-gram occurs frequently in the Darkness malware family. Figure 3b and Figure 3c show the existence of similar n-gram features in the Srat and Zeus malware datasets, e.g., in Figure 3c 50% of the background data does not contain this n-gram, while only 0.04% of Zeus does not have this n-gram. We present some of the interpretations for these n-gram features later in this section, which give some insights about the malware families behavior.

Background Discriminating Features: Similarly, we observed a set of n-gram distributions, where the background behavior is governing over the malware's, i.e., they show up more frequently in the background data. For instance, more than 90% of the malware traffic has 0 occurrence of the n-gram feature in Figure 4, while 90% of the background traffic has at most 20 occurrences of the same n-gram feature. Similar features exist for Srat and Zeus; omitted for space limitations.

Common Features in Malware and Background: We found features that co-exist in both malware and background, which is due to the nature of TCP/IP stack operation. We leave eliminating these features as a job for the recursive feature elimination [16] algorithm, and therefore these features will be ranked at the bottom during the features ranking process, and thus will not be used by our system, since we use the top discriminant features only, as discussed in §V.

B. Feature Interpretation

So far, our analysis unveiled different classes of features. In this section, we present the interpretation of one feature from the malware discriminating and background discriminating classes for the Darkness malware family, which are shown in Figure 3a and Figure 4, respectively.

Darkness Malware Discriminating Feature: Figure 3a shows a uni-gram feature that is dominant in the Darkness malware family, where more than 50% of the malware samples contain at least 120 occurrences of this feature. This uni-gram feature represents the number of occurrences of outbound HTTP POST requests over port 80, where the outbound size is in quartile 4 (i.e., large requests). We believe that the malware samples could be trying to launch DDoS attacks, which is the nature of the Darkness malware family, or they are trying to send aggregate information regarding the infected host to the C&C server. Currently, we are digging deeper in the payload to confirm our intuition. It is clear that this kind of behavior is not frequent in normal user machines, where users normally send small HTTP POST requests, and most requests are HTTP GET. In addition, software updates normally fetch content from the CDN server using HTTP GET requests. Consequently, this feature performs well in distinguishing between the Darkness traffic and the background traffic.

Background Traffic Discriminating Feature: Figure 4 shows a tri-gram feature that is dominant in the background traffic compared to the Darkness malware samples, where 60% of the Darkness samples does not contain any occurrence for this tri-gram, while 60% of the background traffic samples contain at least 10 occurrences of this tri-gram feature. This tri-gram feature represents the number of occurrences of three consecutive outbound HTTP GET requests over port 80 in the same TCP connection, where the outbound request size is in quartile 1 (i.e., small requests). This perfectly matches the operation of modern web browsers, where HTTP requests are pipelined to the web server, e.g., a web server will send a batch of HTTP requests to different objects in the HTML page instead of sending them one-by-one. In this case, this is not prevalent in the malware family samples. Thus, this feature can discriminate between background traffic and malware samples.

We only presented one discriminating feature for each class of features in the Darkness malware family, while similar features exist for the other two malware families and they are left out due to space constraints. It is worth mentioning that our system uses a set of features to predict the malware family labels since using one feature only would result in a large number of false positives.

C. Feature Evaluation

In our system, relying on recursive feature elimination [16] alone is not enough for the system to perform well, since recursive feature elimination ranks all the features, and we are responsible for selecting the top F features. In addition, the different values for n in selecting the n-grams affect the feature selection process as well as the whole system's performance. Therefore, we vary the number of features and the length of n-grams used, and evaluate the ICA and classifier modules for the given values. Then, we select the parameters that work well for each malware family. More specifically, for each malware

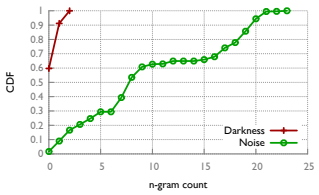


Fig. 4. N-gram feature from the Darkness malware, where background behavior is governing over malware behavior

family, we use the top *F* features using n-grams length up to *n*, and we build the corresponding ICA and classifier models for the learning phase using a subset of the data. Then, we evaluate the performance on the remaining samples. For ICA, we mix the background and malware features using a linear sum of the two distributions, which resulted in the best results for our datasets. We report performance results in §V.

V. Evaluation

We evaluate the performance of our proposed system in recovering the underlying malware feature distribution from the mixed distribution, in addition to predicting the proper malware family labels by testing it on the three datasets described earlier in §II-C. We start by evaluating the performance of ICA in recovering the original malware feature distribution in §V-A, then we show the performance of the overall system under different parameter settings for the number of selected features and the length of n-grams in §V-B.

A. ICA Performance

ICA Estimating Feature Accuracy. As illustrated in Figure 2, ICA is the first stage in our system's operation, where ICA estimates a distribution for each n-gram feature from the input mixed traffic, which contains both malware and background components. There are efficient off-the-shelf implementations for ICA, and we use the FastICA [17] implementation in our system. To evaluate the ICA estimates for the features, we use the ground-truth feature distributions from the malware traffic, and compare them with the ICA approximations. For instance, Figure 5a shows three normalized CDF distributions for an n-gram feature in the Darkness malware family. The red line shows the distribution of the malware feature that does not contain background component (i.e., ground-truth), the blue line shows the distribution of the same feature including the background component (i.e., mixed distribution), and the dotted red line shows the distribution of the same feature after ICA estimates the malware feature distribution from the mixed feature. The figure shows that the dotted red line approximately follows the red line (distribution-wise). The other two plots in Figure 5 show similar distributions for features in Srat and Zeus. This shows the power of ICA in extracting a feature of interest from a mixed feature with a background component. In the following, we evaluate ICA across all n-gram features and malware families empirically.

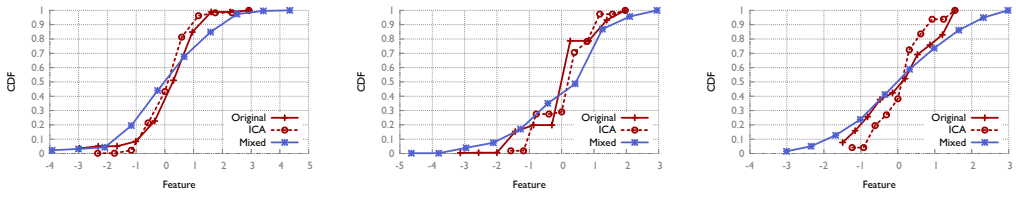
ICA Experimental Results. To evaluate ICA across all n-gram features and malware families, we use Kullback-Leibler (KL) divergence [15] to measure the information lost when using the ICA estimated feature distributions to represent their actual malware distributions. The KL divergence across all features and for the different malware families ranges from a minimum of 0.0665 to a maximum of 1.6744 with an average of 0.4188. An average of 0.4188 means that ICA performs well in estimating the majority of our features, while some of the features are distorted. These features are the lowest ranked features in the top features selected by the recursive feature elimination step. We achieve similar results for different ICA mixing functions, which we discuss next.

ICA Mixing Function. In our ICA experiments, we mix the malware features with their corresponding background features using a linear sum, since ICA relies on a linear mixing function to work properly, i.e., $F(\text{mixed}) = w1 * F(\text{malware}) + w2 * F(\text{background})$, where $F(\text{mixed})$, $F(\text{malware})$, and $F(\text{background})$ are the n-gram distributions in the mixed, malware, and background traffic, respectively. The parameters $w1$ and $w2$ are the weights used in the mixing function. We perform our experiments using different weights and achieve similar results to the linear sum. In practice, this assumption aligns well with the nature of network traffic, since if we see the occurrence of a specific n-gram x times in the malware traffic PCAP and y times in the background traffic PCAP, then we expect the occurrence of that n-gram to be a linear function of x and y in the mixed traffic PCAP. However, the interleaving events and their "timing" in the mixed traffic PCAP might change the order of events we see in the clean malware PCAP, and therefore, change the number of occurrences of that n-gram in the mixed traffic. This issue can be addressed easily using k skip n-gram (refer to §VII for details). A k skip n-gram is a length *n* subsequence from a larger sequence, where the components occur at distance at most k from each other. This resolves the effect of timing in the mixed trace.

B. Classifier Performance

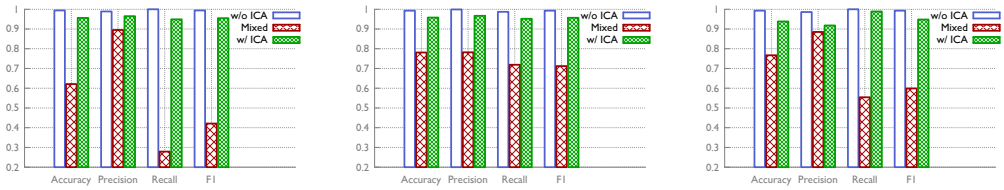
The capabilities of the proposed system are demonstrated in two aspects: its power of separating mixed traffic, and its effect on the accuracy of the classifier built on top of the estimated malware traffic. To this end, we demonstrate the effectiveness of our system in separating mixed signals and classifying malware samples based on the estimated features extracted by the ICA module. We first define the evaluation measures and experimental setup, and follow that with experimental results.

Experimental Setup. We use the Random Forest [9] classifier for the classification stage. For our evaluation, we use 10-fold cross validation to evaluate the performance of ICA and the classifier. In this setting, each dataset is split into 10 folds, where 9 folds are used for training the ICA and the classifier models, and the 10^{th} fold is used for testing (prediction). For the testing fold, we use the mixed feature distributions as input to the ICA model, then the estimated ICA distributions are used as input to the classification stage. We re-run the experiment by alternating the testing fold each time among the 10 folds and compute the evaluation metrics each time.



(a) Darkness Feature (b) Srat Feature (c) Zeus Feature

Fig. 5. Each plot is the normalized CDF distribution comparison between the exact feature distribution without background traffic, feature with background traffic, and ICA approximated distribution



(a) Darkness (b) Srat (c) Zeus

Fig. 6. Each plot shows the classifier results for a different malware, where we compare without using ICA, without using ICA but the traffic is mixed, and using ICA to clean the mixed traffic

Finally, we average our results across all 10 folds. We report the following metrics:

1) Accuracy = (TP + TN) / (P + N), i.e., how well the system predicts both sets of samples (samples containing a certain malware family, and samples that are benign).
2) Precision = TP / (TP + FP), i.e., the percentage of predicted malware samples that are truly malware samples.
3) Recall = TP / (TP + FN), i.e., the percentage of predicted malware samples from all true malware samples.
4) F1 score = geometric mean between precision and recall.

In the above metrics, P represents the positives (malware), and N represents negatives (background), TP represents the true positives, TN represents the true negatives, FP represents the false positives, and FN represents the false negatives. Figure 6 reports these metrics for the three malware families used in this study under three different experimental setups. In the following, we elaborate on these results and findings.

Experimental Results. Figure 6a shows the classification results for the Darkness malware family, where we compare three different setups. First, the blue bars shows the classification results without using ICA or mixing the features, i.e., the input to the classifier is the ground-truth malware and background features, which is the best performance achievable since there is no noise in the malware features. Second, the hashed red bars show the results for the classifier when the input is the noisy features, i.e., after mixing the malware features with their corresponding background features but without using our ICA module. Third, the hashed green bars show the classifier results when we use ICA to clean the mixed features from the background noise, and then use these cleaned features as input to the classifier. From the figures, it is clear that ICA effectively cleans the features so that one can achieve high accuracy in labeling the malware families,

TABLE III
ABSOLUTE NUMBERS FOR RESULTS PRESENTED IN FIGURE 6 FOR THE GREEN HASHED BARS (W/ ICA)

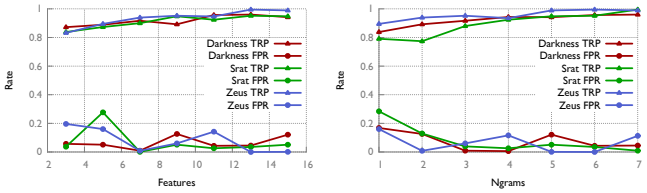
	Darkness	Srat	Zeus
True Positives (TP)	50	88	92
True Negatives (TN)	51	89	83
False Positives (FP)	2	4	10
False Negatives (FN)	3	5	1
False Positive Rate (FPR) $FPR = FP/(FP+TN)$	0.056	0.053	0.012

even outperforming the related literature utilizing the same dataset [10]. The classifier suffers when the mixed features are used without ICA. For example, we note that the accuracy, which summarizes the performance of the classifier is about 0.75 when using the noised features, whereas applying ICA to clean those features, and then applying the classifier results in an accuracy of 0.95, with 20% improvement over using the noised features. Note that these results are based on using the top 15 features and n-grams where n is up to 5. Table III lists the absolute numbers for TP, TN, FP, FN, and the false positive rate (FPR) for one of the 10 test folds in the three malware families. FPR is the percentage of background traffic that does not contain malware that is predicted by our system as malware (i.e., false alarms).

Effect of Changing the Number of Features and Length of n-grams. Next, we evaluate our system against different numbers of features and different values of n for the n-grams. We perform this experiment in order to select the best number of features, and the best value of n for the n-grams for each malware family to achieve the highest effectiveness in labeling a malware family empirically. We report the true positive rate (TPR), i.e., Recall, and the false positive rate (FPR). Basically,

this experiment is a Receiver Operating Characteristic (ROC) analysis [18] to tune our parameters.

Figure 7 reports the results, where Figure 7a shows the results while varying the number of features, and Figure 7b shows the results for different values of n for the extracted n-grams. Each plot reports the TPR and FPR for the three different malware families. From the figures, we can see that after reaching 15 features, where $n = 5$ for the n-grams, the performance starts to stabilize and no gain in performance is achieved. We preferred using $n = 5$ in order to be able to interpret the n-gram features and to avoid overfitting the data.



(a) Features Performance (b) N-grams Performance

Fig. 7. Classifier performance across different number of features in (a), and as the value of n increases for n-grams in (b)

VI. Related Work

Machine learning based algorithms for classifying malware families have a lot of proposals in the literature, which can be categorized as either signature-based, or behavior-based algorithms. Our system belongs to the behavior-based category, since we use features that rely on network events seen while executing the malware sample. Related to our work is CHATTER [10], where the authors classify malware samples using n-gram features based on network events. However, there are two major differences between our system and CHATTER. First, CHATTER relies on fine-grained events, since attributes extracted from one packet are treated as different events, e.g., inbound, DNS, and port number are treated as different network events. On the other hand, our system uses coarse-grained groups, e.g., inbound, DNS, and port number from one packet are treated as one network event. Therefore, our system can capture events like a DNS request followed by an HTTP GET request, while CHATTER cannot do that for the aforementioned operational difference. Second, we use ICA to clean the mixed network traffic from background attributes, while CHATTER assumes clean malware data is in use. In addition, while similar to CHATTER in this aspect, our system is different from other recent related work in an important aspect concerning the order of events. We use n-grams to capture the order of events, which exposes richer information about the operation of the underlying malware. In addition, we use highly accurate labels for our malware samples, since these labels are manually vetted by malware analysts at our institution. Other proposals rely on Anti-Virus labels, which are known to have pitfalls [1].

Beside CHATTER, there have been other methods in the literature related to this work in many aspects. In [2], the authors used behavior graph matching to identify and classify malware samples. However, their technique comes at a high cost for graph generation. In addition, some of those proposals rely on payload analysis, which is a costly operation. Related to our use of network features is the line of research on traffic analysis for malware and botnet detection [19] and for the particular families of malware that use fast flux [20, 21]. Some other methods are related to our use of the DNS features for malware analysis [22–24]. However, none of these studies are concerned with behavior-based analysis and classification of malware beyond the use of remotely collected network features for inferring malicious activities and intent. Thus, although they share some similarity with our work in its purpose, they are different in the utilized techniques.

Our system is different from previous systems that use n-grams and the order of events for malware classification in the following aspects. First, a group of other methods investigated extracting features from executables (e.g., sequence of bytes in the binary [3]), or streams of communication traffic [4], but not the sequence of events happening while executing a malware sample. Examples of such systems include [4, 25–27]. Second, our system is different from others that use network artifacts for identifying malicious activity [19, 28–31], like botnets, in the way we group these events into features using n-grams. Third, using the order of events in characterizing OS processes was first explored by Forrest *et al.* in their seminal work [32], where they show that a process-level intrusion can be detected using the order in which system calls happen as a sequence. Our system is different in the following aspects: we are interested in malware classification not detection, and we use network artifacts not system calls. For a comprehensive study on using machine learning techniques to classify malware, the reader can refer to recent surveys [33].

VII. Discussion

Using n-grams on higher level network events helps understand the underlying operation of the malware, and provides a good feature set for classification. In addition, ICA improves the performance by estimating the malware feature distribution in the mixed traffic. In this section, we present the challenges and limitations of our system due to using n-grams and ICA.

N-grams Challenges: As the length of our n-grams "n" increases, interpreting the features becomes harder. This happens since we have to understand the meaning of longer sequences of network events. With the help of malware analysts at VeriSign, we are exploring other discriminant features in order to gain insight into the main differences in the behavior between the malware and background traffic, as well as the underlying malware operation. This will also help us provide better solutions that detect malware families.

ICA Challenges and Limitations: The two main assumptions that ICA relies on are independence of the two underlying distributions, and the use of a linear mixing function to produce the mixed traffic. In this paper, we assumed that the background traffic such as web browsing, OS updates, or peer-to-peer traffic are independent from the malware traffic

running on the same machine. Our ICA performance analysis shows that in the three malware families we have studied, this assumption holds. ICA also requires only one Gaussian distribution, and we used normality tests to confirm that malware feature distributions are not Gaussian. Consequently, ICA can only work on malware families that do not exhibit any Gaussianity in their features. Our work cannot be applied to malware families that have similar properties to the background traffic (e.g., both are dependent). Exploring the extent to which our system is affected by violation of these properties is a future work.

The second assumption is linearity of mixing, i.e., the mixed distribution is a linear function of the malware and background distributions. Due to the nature of network traffic, this assumption is not violated, which was validated by our results; the distribution of a uni-gram feature such as the number of HTTP GET requests in the mixed traffic is going to be the sum of both the malware and background traffic. To learn the ICA unmixing function, our system mixes the pure malware feature distributions from the ground-truth data, with their corresponding features in the background traffic to train the ICA model to split them as we discussed earlier in §V-A. In practice, some n-grams in the mixed traffic may be missing in the plain malware or background traffic due to timing changes. Therefore, we can use a skipping factor k to skip over words in the mixed data profiles, e.g., considering *"w1 w2 w3 w4"* as a sequence of words, then with a skipping factor $k = 1$, we generate the following bi-grams *{(w1 w2), (w1 w3), (w2 w3), (w2 w4), (w3 w4)}*. This helps improve the features selected by our system, but increases the computation cost to select such features. We faced two challenges when working with ICA: scaling and ordering. The output of ICA is on a different scale from the input. We solved this issue by mapping both the input and output feature distributions to the same scale. Also, ICA outputs are not tagged, i.e., we do not know which of the two estimated distributions is the malware distribution. To fix this issue, we measured the distance between the ground-truth distribution and the two estimated distributions. Then, we used the closer distribution to the ground-truth. We found that these two fixes worked well in our system.

VIII. Acknowledgments

This research was supported in part by NSF grants CNS-1117536, CRI-1305237, CNS-1411636 and DTRA grant HDTRA1-14-1-0040 and DoD ARO MURI Award W911NF-12-1-0385. Part of the research was conducted while the first author was a summer intern and the second author was a researcher at Verisign Labs.

IX. Conclusion

In this paper, we proposed a novel supervised machine learning based classification system that accurately predicts the malware family for a given mixed network traffic for an infected host. Unlike other existing classification methods, our system separates the mixed traffic into malware and background traffic using ICA, which improves the performance of the classifier. We used n-gram frequencies as features in our random forest classifier, which preserve the order of network events. Our system is generic in purpose, and can meet many needs in multiple applications. For example, traffic analysis and identification is a generic problem related to services profiling, and our system can be used.

We are planning to look into extending this system to address problems in other domains with similar contexts. This includes user identification and profiling. We will also look into gaining operational experience and measurements based on a real deployment for applications built using our system.

References

[1] A. Mohaisen and O. Alrawi, "AV–Meter: An Evaluation of Antivirus Scans and Labels," in *DIMVA*, 2014.
[2] Y. Park, D. Reeves, V. Mulukutla, and B. Sundaravel, "Fast malware classification by automated behavioral graph matching," in *ACM CSIIR Workshop*, 2010.
[3] K. Rieck, P. Trinius, C. Willems, and T. Holz, "Automatic analysis of malware behavior using machine learning," *Computer Security*, 2011.
[4] C. Wressnegger, G. Schwenk, D. Arp, and K. Rieck, "A close look on n-grams in intrusion detection: anomaly detection vs. classification," in *ACM workshop on Artificial intelligence and security*, 2013.
[5] K. Rieck, T. Holz, C. Willems, P. Düssel, and P. Laskov, "Learning and Classification of Malware Behavior," in *DIMVA*, 2008.
[6] D. Kong and G. Yan, "Discriminant Malware Distance Learning on Structural Information for Automated Malware Classification," in *ACM SIGKDD*, 2013.
[7] T.-W. Lee, *Independent Component Analysis*. Springer, 1998.
[8] N–gram Models, "http://en.wikipedia.org/wiki/N-gram," Dec 2014.
[9] L. Breiman, "Random Forests," *Machine learning*, 2001.
[10] A. Mohaisen, A. West, A. Mankin, and O. Alrawi, "Chatter: Exploring classification of malware based on the order of events," in *CNS*, 2014.
[11] A. J. Bell and T. J. Sejnowski, "An Information-Maximization Approach to Blind Separation and Blind Deconvolution," *Neural Comp.*, 1995.
[12] A. Mohaisen, O. Alrawi, A. G. West, and A. Mankin, "Babble: Identifying malware by its dialects," in *CNS*, 2013.
[13] D. Alperovitch *et al.*, *Revealed: operation shady RAT*. McAfee, 2011.
[14] A. Mohaisen and O. Alrawi, "Unveiling Zeus: Automated Classification of Malware Samples," in *World Wide Web companion*, 2013.
[15] J. M. Joyce, "Kullback-Leibler Divergence," in *International Encyclopedia of Statistical Science*, 2011.
[16] I. Guyon, J. Weston, S. Barnhill, and V. Vapnik, "Gene Selection for Cancer Classification using Support Vector Machines," *Machine learning*, 2002.
[17] A. Hyvarinen, "Fast and Robust Fixed-point Algorithms for Independent Component Analysis," *IEEE Transactions on Neural Networks*, 1999.
[18] J. A. Hanley and B. J. McNeil, "The meaning and use of the area under a roc curve," *Radiology*, 1982.
[19] G. Gu, J. Zhang, and W. Lee, "Botsniffer: Detecting botnet command and control channels in network traffic," in *NDSS*, 2008.
[20] T. Holz, C. Gorecki, K. Rieck, and F. C. Freiling, "Measuring and detecting fast-flux service networks," in *NDSS*, 2008.
[21] J. Nazario and T. Holz, "As the net churns: Fast-flux botnet observations," in *MALWARE*, 2008.
[22] L. Bilge, E. Kirda, C. Kruegel, and M. Balduzzi, "Exposure: Finding malicious domains using passive dns analysis," in *NDSS*, 2011.
[23] M. Antonakakis, R. Perdisci, D. Dagon, W. Lee, and N. Feamster, "Building a dynamic reputation system for dns," in *USENIX Security*, 2010.
[24] M. Antonakakis, R. Perdisci, W. Lee, N. V. II, and D. Dagon, "Detecting malware domains at the upper dns hierarchy," in *USENIX Security*, 2011.
[25] R. Perdisci, A. Lanzi, and W. Lee, "Mcboost: Boosting scalability in malware collection and analysis using statistical classification of executables," in *ACSAC*, 2008.
[26] J. Z. Kolter and M. A. Maloof, "Learning to detect and classify malicious executables in the wild," *Machine Learning Research*, 2006.
[27] M. G. Schultz, E. Eskin, F. Zadok, and S. J. Stolfo, "Data mining methods for detection of new malicious executables," in *S&P*, 2001.
[28] W. T. Strayer, D. E. Lapsley, R. Walsh, and C. Livadas, "Botnet detection based on network behavior," in *Botnet Detection*, 2008.
[29] G. Gu, P. Porras, V. Yegneswaran, M. Fong, and W. Lee, "Bothunter: Detecting malware infection through ids-driven dialog correlation," in *USENIX Security*, 2007.
[30] G. Gu, R. Perdisci, J. Zhang, and W. Lee, "Botminer: clustering analysis of network traffic for protocol- and structure-independent botnet detection," in *USENIX Security*, 2008.
[31] R. Perdisci, W. Lee, and N. Feamster, "Behavioral clustering of http-based malware and signature generation using malicious network traces," in *NSDI*, 2010.
[32] S. Forrest, S. A. Hofmeyr, A. Somayaji, and T. A. Longstaff, "A sense of self for unix processes," in *S&P*, 1996.
[33] R. Sommer and V. Paxson, "Outside the closed world: On using machine learning for network intrusion detection," in *S&P*, 2010.