ADAM: Automated Detection and Attribution of Malicious Webpages

Ahmed E. Kosba[1], Aziz Mohaisen[2], Andrew G. West[2], Trevor Tonn[3], Huy Kang Kim[4]

[1] University of Maryland at College Park
[2] Verisign Labs
[3] Amazon.com
[4] Korea University

Abstract. Malicious webpages are a prevalent and severe threat in the Internet security landscape. This fact has motivated numerous static and dynamic techniques to alleviate such threat. Building on this existing literature, this work introduces the design and evaluation of ADAM, a system that uses machine-learning over network metadata derived from the sandboxed execution of webpage content. ADAM aims at detecting malicious webpages and identifying the type of vulnerability using simple set of features as well. Machine-trained models are not novel in this problem space. Instead, it is the dynamic network artifacts (and their subsequent feature representations) collected during rendering that are the greatest contribution of this work. Using a real-world operational dataset that includes different type of malice behavior, our results show that dynamic cheap network artifacts can be used effectively to detect most types of vulnerabilities achieving an accuracy reaching 96%. The system was also able to identify the type of a detected vulnerability with high accuracy achieving an exact match in 91% of the cases. We identify the main vulnerabilities that require improvement, and suggest directions to extend this work to practical contexts.

1 Introduction

The ever increasing online and web threats call for efficient malware analysis, detection, and classification algorithms. To this end, antivirus vendors and intelligence providers strived to develop analysis techniques that use dynamic, static, or hybrid—which use both—techniques for understanding web malware. While static techniques are computationally efficient, they often have the drawback of low accuracy, whereas dynamic techniques come at higher cost and provide higher accuracy. Certain functionalities, such as deep analysis of dynamic features, are more costly than gathering of indicators and labeling of individual pieces of malware. Systems that are costly utilizing dynamic features should be augmented with intelligent techniques for better scalability. Such techniques include machine learning-based components utilizing light-weight features, such as network metadata, for finding the label and type of a given website without using the computationally heavy components. In addressing this problem, we introduce ADAM, an automated detection and attribution of malicious webpages that is inspired by the need for efficient techniques to complement dynamic web malware analysis.

The motivation of this work is twofold. First, iDetermine, a proprietary status quo system for detecting malicious webpages using dynamic analysis is a computationally expensive one. While iDetermine is the basis for our ground-truth and network metadata used for creating features for webpages, it also does a great quantity of other analysis to arrive at accurate labels (*e.g.,* packet inspection, system calls). We envision our efforts could integrate as a tiered classifier that enables greater scalability with minimal performance impact. Second, existing literature on webpage classification [7, 17, 18, 23, 24] provided promising accuracy. Because these approaches rely primarily on static features, we hypothesize that metadata from network dynamics might improve it as well.

There are multiple challenges that ADAM tries to address. First, webpages face different types of vulnerabilities: exploit kits, defacement, malicious redirections, code injections, and server-side backdoors – all with different signatures. This malice may not even be the fault of a webpage owner (*e.g.*, advertisement networks). Moreover, the distribution of behavior is highly imbalanced, with our dataset having 40$\times$ more benign objects than malicious ones. Despite these challenges, our approach is currently broadly capable of 96% accuracy, with injection attacks and server-side backdoors being identified as areas for performance improvement and future attention. The system is also capable of identifying the types of detected vulnerabilities with exact match in 91% of the cases, with a difference of 1 and 2 labels in 6% and 3% of the cases respectively.
Contribution. The contributions of this paper are: 1) Presenting a system that identifies whether a webapge is malicious or not based on simple dynamic network artifacts collected in sandboxed environments, in addition to 2) Evaluating the system using a real dataset that contains multiple variants of malicious activity.
Organization. The rest of this paper is organized as follows: §2 discusses the background and related work. §3 presents some details on iDetermine system, which generates the data we use in ADAM and the ground truth, while §4 presents the architecture of the ADAM system. §5 presents the evaluation of ADAM and discusses the results, and finally §6 presents the conclusions and sheds some light on future work.

2 Related Work

There has been a large body of work in the literature on the problem at hand, although differing from our work in various aspects, including features richness, quality of labels, and their context. Most closely related to our work are the works in [7, 17, 18, 23, 24], although differing in using static analysis-related features in reaching conclusions on a webpage. On the other hand, ADAM relies on utilizing simple features extracted from the dynamic execution of a webpage and loading its contents in a sandboxed environment, with the goal of incorporating that as a tiered classifier in iDetermine.

Related to our work, but using structural properties of URLs in order to predict malice are the works in [10, 17, 25] for email spam, and in [6, 19] for phishing detection. Additionally, using domain registration information and behavior for malware domain classification was explored in [14, 17]. Related to that is the work on using machine learning techniques to infer domains behavior based on DNS traces. Bilge et al. proposed Exposure [5], a system to detect malware domains based on DNS query patterns on a local recursive server. Antonakakis et al. [2] functions similarly but analyzes global DNS resolution patterns and subsequently creates a reputation system for DNS atop this logic [1]. Gu et al. [11–13] studied several botnet detection systems utilizing the same tools of DNS monitoring. Dynamic malware analysis and sandboxed execution of mal-

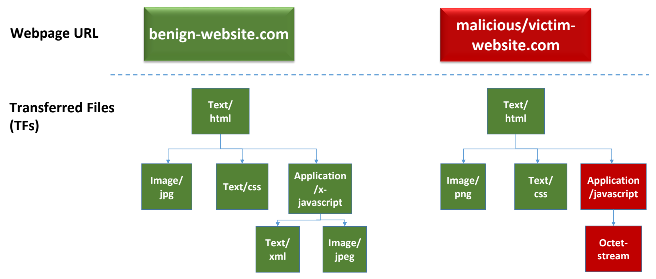

Fig. 1. Two examples of transferred file trees

ware were heavily studied in the literature, including surveys in [8, 9]. Bailey et al. [3] and Bayer et al. [4] have focused on behavior-based event counts. Feature development has since advanced such that malware families can now be reliably identified [16] and dynamic analysis can be deployed on end hosts [15]. Finally, network signature generation for malicious webpages is explored in [21, 22] for drive-by-download detection.

3 iDetermine

iDetermine is a system for classification of webpage URLs. It crawls websites using an orchestrated and virtualized web browser. For each analyzed URL, the system maintains records of each HTTP request-response made while rendering that page. The system applies static and dynamic analysis techniques to inspect each object retrieved while visiting the URL, and monitors any changes that happen to the underlying system to decide whether the retrieved object is malicious or not. We call these objects transferred files (TFs). If any of the retrieved objects was found malicious, iDetermine labels the object based on the type of malice uncovered. The system may label a malicious TF with one or more of the following:

Injection. Occurs when a website is compromised, allowing an attacker to add arbitrary HTML and javascript to the legitimate content of the site with the purpose of invisibly referencing malicious content aimed at silently harming visitors.

Exploit. Implies that an exploit code for a vulnerability in the browser or browser helper was found. Exploit code are the heart of drive-by downloads.

Exploit Kit. A collection of exploits bundled together and usually sold in black market. These kits increase the probability that the browsers of the visiting users are successfully exploited.

Obfuscation. A TF contains obfuscated code with known malicious activity behavior.

Defacement. Occurs when an attacker hacks into a website and replaces some content indicating that the site has been hacked into.

Redirection. A TF redirects to a known malicious content.

Malicious executable or archive. This means that either an executable or an archive file, e.g. zip, rar, jar, that contains malicious code of some sort was detected to be downloaded by visiting the webpage.

Server side backdoor. A TF shows symptoms of being a known server-side backdoor script, like the C99 PHP Shell. Such files allow remote attackers to control various aspects of the server

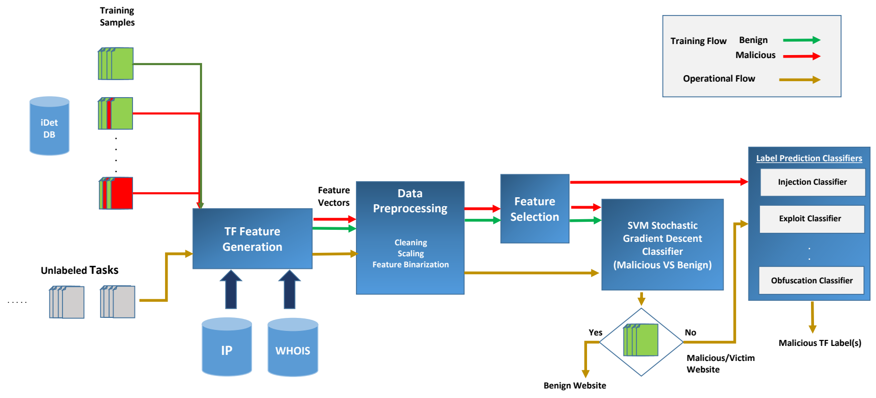

Fig. 2. The workflow for classifying URLs based on TFs

The processing of the data of each URL by iDetermine results in a tree-like structure (see Fig. 1) where each node represents a TF. Each node stores basic file attributes and network information (*e.g.*, HTTP response code, IP address, and Autonomous System (AS) number). These nodes also contain classification data from iDetermine's deep analysis and we use this as ground-truth in training/evaluating our approach.

4 ADAM: System Structure and Overview

Design goals. There are two basic end goals for the proposed system. The main goal is to identify whether a webpage is malicious or not based on the basic metadata maintained by iDetermine, without the requirement to compute any complex and expensive features. If the webpage is classified as malicious, the system also aims at identifying which type of malice this webpage has.

Design. The layout of the system is outlined in Fig. 2. The figure shows the flow of both the training data and the operational data. The system is trained by labeled webpages, in which each individual TF is labeled whether it is benign (green), or malicious (red). The system uses the basic meta-data stored in the system, in addition to a set of simple features generated based on those attributes. This generation is handled by the feature generation module which uses IP and WHOIS databases to acquire information about the IP address and the domain name of the associated TF. After the feature generation stage, the data is preprocessed, and some features may be filtered using a feature selection module, before the data is sent to the classification modules. Then, a two-stage classification procedure is trained based on the preprocessed data.

In the operational mode, for an unlabeled webpage, the system transforms each TF into a feature vector as done by the feature generation module in the training phase, and then the features are pre-processed and filtered based on the feature selection results from the training phase. The TF is then labeled with the label most close to it in the vector space based on a highly accurate ground truth. To this end, in the following two subsections, we provide more details on the generated features, the preprocessing stage, and then we discuss the classification procedure needed to achieve the above two goals.

4.1 Features used for Classification

To achieve the design goals, ADAM relies on a rich set of features, and uses nearly 40 basic features for the classification process. The features fall in the following categories:

- BASIC META-DATA FEATURES: This represents the simple meta-data attributes stored originally by iDetermine, such as the HTTP header information, which includes HTTP method, response code, Is Zipped, .. etc. The meta-data also includes the AS number, and the result of running the libmagic command on the TF file which gives information about the type of the retrieved file.
- URI-BASED FEATURES: These are the features derived from the URI associated with a TF. This includes some basic lexical statistics, e.g. URI components lengths (hostname, path and query), dot count, slash count, special characters ratio and the average path segment length. This also includes binary features to indicate whether the URI contains an explicit IP, or an explicit port number. Furthermore, the features include the top-level domain name in addition to the token words that appear in the different URI components for which we use a bag-of-words representation.
- TF TREE-BASED FEATURES: These are the features we extract from the TF-tree to capture the relationship between different TFs that belong to a single webpage. The TF-tree features capture Parent-child host/IP diversity; TF depth; number of children and the child-parent type relationship.
- DOMAIN NAME-BASED FEATURES: These features are derived from the domain name of the URI of the TF. This includes: the registrar's id and age information, e.g. creation data and expiration date.
- IP-BASED FEATURES: These are a set of features derived from the IP address associated with the TF. This includes the Geo-Location features: country, city and region, in addition to the domain/organization for which the IP is registered. Furthermore, we consider two IP prefixes (/24 and /28) as features to detect networks with malicious activity, instead of considering each IP individually.

It should be noted that the iDetermine system does process and store additional data that could be useful in the classification task. For example, payload and content-based features derived from Javascript as in [7, 24], or flow information features as in [24] can be extracted and utilized. However, we do not integrate these features in order to maintain a content-agnostic and scalable classifier.

4.2 Preprocessing and Feature Selection

After the feature values for each category are inferred, a preprocessing stage is needed before forwarding this data to the classifiers for training and testing purposes. The preprocessing is done based on the feature type. For numeric features, such as the lexical counts, proper scaling is applied to keep the values between 0 and 1. For categorical features such as the top-level domain name or AS number, we apply feature binarization, in which a binary feature is introduced per each possible value, since the feature cannot be encoded numerically due to the absence of order between the values. This approach has been employed before, such as in [17]. This certainly will result in high-dimensional feature vectors that require a scalable classifier suitable for high dimensionality vectors.

Due to the high dimensional feature vectors, it could be beneficial to reduce the dimensionality through a feature selection technique. Therefore, in our experiments, we study the effect of reducing the dimensionality through a chi-square metric.

4.3 Classification

After preprocessing the data, we train a two-stage classification model to detect whether a webpage is malicious, and to identify the type of malice if needed.

The first classification stage includes a binary classifier that is trained with all the TFs from benign and malicious samples. We use an SVM classification algorithm based on Stochastic Gradient Descent using L1-norm for this stage. In the second stage, we build another binary classifier for each type of vulnerability. Each classifier in the second stage is trained using the malicious TF data only, e.g. the injection classifier is trained by the data containing (injection TFs versus No injection but malicious TFs).

The reason we employ this two-stage model is due to the limitations of other possible approaches. For example, a multi-class classifier will not capture the observation that some TFs are labeled with more than one label. Additionally, we found that using multiple binary classifiers directly in a single stage, where each classifier is trained for only one type of attack—versus all the other benign and remaining malicious TFs—will lead to lower accuracy and a higher training time. The low accuracy in this case is due to the higher possibility of false positives because of using multiple classifiers at once. Therefore, we propose this two-stage model to filter out the malicious TFs first using a global classifier, then identify the type of malice separately.

In the operational phase, whenever a webpage is analyzed during operation, the data of each TF retrieved while visiting the URL are used to predict whether it is malicious or not. A URL is labeled as benign if all of its retrieved TFs were classified as benign by the classification algorithm. Then, the type of malice is identified through the second stage if the TF was labeled as malicious.

5 Evaluation

We present the evaluation and analysis of the proposed system. We give an overview and description of the dataset with the evaluation procedure and metrics. Then, we introduce the performance of the binary classification mechanism and malice label prediction, followed by the effect of feature selection on the system accuracy.

5.1 Dataset description and statistics

The dataset we consider for evaluation consists of 20k webpages, 10k each of "malicious" and "benign" types. These URLs were randomly selected from iDetermine's operational history of Internet-scale crawling. As mentioned earlier, iDetermine labels the webpages using sophisticated static and dynamic analysis techniques, and hence we consider such labels as our ground truth labels. Analyzing the URLs of the dataset yields 800k benign TFs and 20k malicious TFs. Each webpage contains about 40 TFs on average. A histogram of the number of TFs per webpage is provided in Fig. 3. For the malicious webpages, a histogram of the percentage of the number of malicious TFs

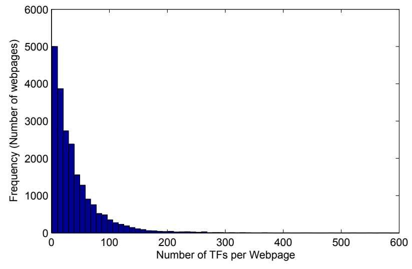

Fig. 3. A histogram of TFs per webpage

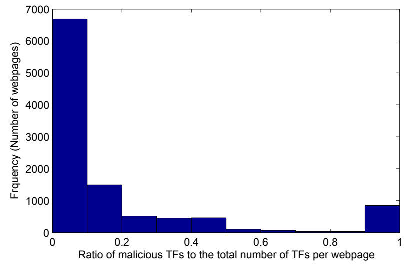

Fig. 4. Malicious TFs per malicious webpages

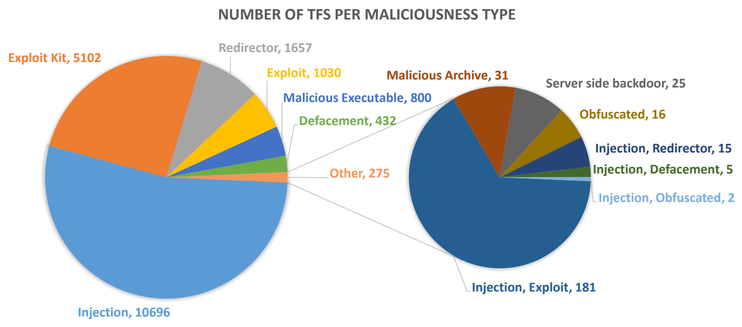

Fig. 5. Distribution of malice among the TFs

per each malicious webpage is shown in Fig. 4. The figure shows that for most malicious webpages, less than 10% of the retrieved TFs are malicious. This confirms the intuition we have for building the classifiers based on individual TFs.

The iDetermine system labels each malicious TF according to any type of malice it uncovered. Note that a malicious TF may be labeled with more than one label at the same time. That is a reason a classifier was built for each malice type in the label prediction module. The distribution of vulnerabilities among the malicious TFs can be illustrated in detail through Fig. 5.

5.2 Evaluation Procedure and Metrics

A prototype of the system was built using Python 2.7, and Scitkit-learn [20] was used for data processing and classification. The evaluation of the system was conducted using 10-fold cross-validation, in which the webpages dataset were divided into 10 distinct partitions, nine of which are used for the training stage while the remaining partition is used as the testing data. For consistency, the dataset was partitioned randomly in a way that guarantees that the distribution of number of TFs per webpage (shown before in Fig.3) is roughly maintained, so that the total number of TFs per partition is almost the same, since the TFs are the main classification data units the system works on.

The performance metrics will be provided at both the TF and the webpage granularity, with more focus on the latter since this is the end system goal. Recall that a webpage is labeled as malicious **if any** of its TFs was labeled by the classifier as malicious. The metrics considered for the evaluation are mainly the false positives rate, which describes the ratio of the benign objects that were labeled as malicious, and the false negatives rate which describes the ratio of the malicious objects that were labeled as benign. We also measure the effectiveness of the detection system through the F1-score, which is calculated based on the harmonic mean of precision and recall. Precision refers to the fraction of the objects that the system labeled as malicious that turned out to be truly malicious, while recall is the ratio of the truly malicious objects that the system was able to label malicious.

5.3 Binary Classification Performance

We start by describing the results of the first classification stage, which aims to identify whether a webpage is malicious or benign, only. Tab. 1 enumerates the performance metrics at both TF and webpage granularity, showing an overall result of 7.6% FN rate and 6.3% FP rate for the webpage results. The reason for having a 14.7% FN rate on the TF-level is that simple metadata may not be indicative for all types of TF malice behavior. Additionally, compared to previous literature, the TF results are consistent with respect to the fact that our TF records dataset is highly imbalanced. Literature studies showed that as the data gets highly imbalanced, the accuracy degrades, e.g. 25% FN rate at a ratio of 100:1 of benign to malicious URLs [18].

To better understand how well the detection mechanism performed, Fig. 7 shows the detection rate per each vulnerability/attack type at the TF-level, which describes the ratio of the TFs labeled as malicious successfully. Note that the “injection” and “server side backdoor cases” were most detrimental to overall performance. This is made clear in Tab. 2 which provides overall performance without those problematic instances, resulting in 2.5% FP rate and 4.8% FN rate overall.

	Prec.	Recall	F-score	FP	FN
TF-level	0.390	0.852	0.530	0.0314	0.147
page-level	**0.935**	**0.924**	**0.930**	**0.063**	**0.076**

Table 1. Binary classification results

	Prec.	Recall	F-score	FP	FN
TF-level	0.527	0.873	0.657	0.0153	0.126
page-level	**0.948**	**0.951**	**0.949**	**0.0257**	**0.048**

Table 2. Binary classification w/o “injection”

5.4 Label Prediction Performance

After a TF is labeled as malicious by the system, the system labels it according to the type of attack/malice it carries by the label prediction module described earlier in Sec. 4. In this section, the results of this module are presented. The main metric we used for the evaluation of the label prediction is the number of different labels between the ground truth and the predicted ones. As an example for illustration, if the ground truth is {Injection}, and the system labeled the malicious TF as {Injection, Exploit}, then this is considered a difference of one. If the predicted label was only {Exploit}, this is considered a difference of two, since two changes are necessary to make the prediction correct. Fig.6 illustrates the CDF of the label difference metric. As the figure clearly shows, the median of the difference in label predictions is zero. In fact in more than 90% of the cases, there was no difference between the predicted labels and the ground truth, and in only about 3% of the cases there was a difference of two labels.

Furthermore, to evaluate the capability of each individual label prediction classifier. Fig.8 shows the performance of each classifier, by providing two quantities: the rate of

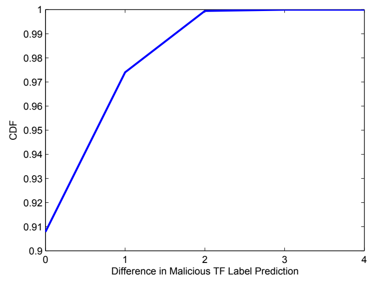

Fig. 6. The CDF of the difference in malice label predictions

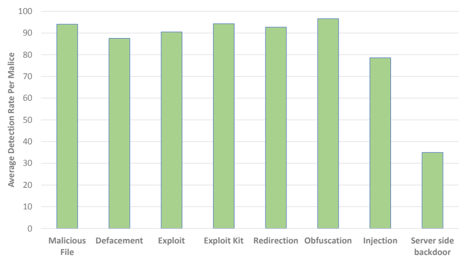

Fig. 7. Detection rate per TF vulnerability type for various malice types

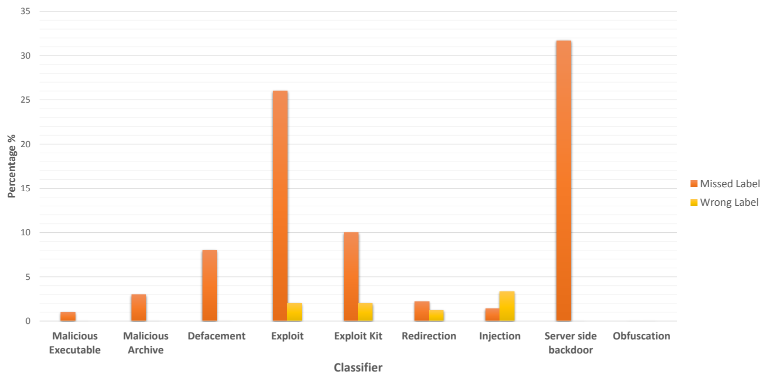

Fig. 8. Performance of individual label prediction classifiers

miss-label, which indicates the ratio of the cases where a classifier was not able to detect a TF that has the type of attack it's concerned with, and the rate of wrong-label which is the ratio of the cases where the classifier gave a positive detection, while the TF does not include such type of malice. As the figure indicates, with respect to the miss-label rate, the server side backdoor classifier had the highest miss-label rate, which could be directly due to the few samples of server side backdoors that the dataset has (recall Fig.5). Then, it can be observed the both the exploit and exploit kit classifiers have high miss-label rates as well, which suggests that new exploit attacks that the system did not specifically learn about may not be directly easy for the system to infer. With respect to the wrong-label rate, one interesting observation is that the injection classifier had the highest wrong-label rate. This could be because most of the malicious TFs in the dataset are Injection attacks (recall Fig.5), which could have resulted tendency towards labeling malicious TFs as injection due to the imbalanced training dataset.

5.5 Feature selection results

Due to the number of categorical features we have, high dimensional vectors result due to feature binarization. This can have a negative effect on the scalability of the system. Additionally, not all features after expansion/binarization can be directly useful in identifying whether a webpage is malicious or not. In this subsection, we provide some observations on feature selection results.

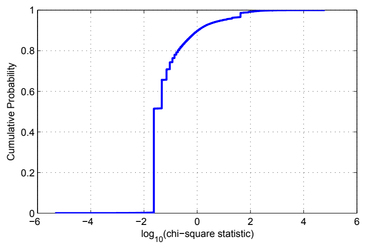

Fig. 9. The CDF of the feature scores. Note the vertical jump in the curve, indicating that half of the features are equally important.

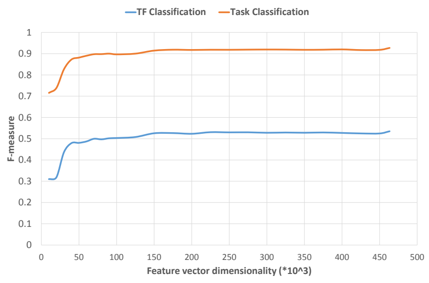

Fig. 10. The effect of the number of features on detection accuracy. The detection accuracy gets stable after using only 50% of the features.

Table 3. Distribution of Generated Features

Feature Category	Number of Features
Meta-data based	18850
URL-based	378740
TF tree-based	157
Domain name-based	419
IP-based	65153

With respect to the chi-square score calculated for each feature by the feature selection module, it can be observed that the feature scores considerably vary, ranging from 10^{-5} to 10^{5}. To illustrate the distribution of the feature scores, Fig.9 provides the CDF of the logarithm of all feature scores over the dataset. The main observation in this figure is that roughly the lowest 50% of the features have the same score (Note the vertical jump after -2). This may suggest that half of the features may not be very important for the classification. This can be confirmed next by studying the effect of the number of features used for classification on the detection accuracy of the system. From another perspective, Fig.10 illustrates the effect of the number of used features on the F1-score. The features are selected based on their scores; when n is the number of features used, the top n features according to the scoring criteria are used. The figures shows the performance increases rapidly till it reaches some point, beyond which the F-score almost gets stable. This is consistent with the score CDF figure provided before.

It is also interesting to identify how important each feature category is. Since many of the features we use are categorical (and hence binarized), it may not be very helpful to solely identify the best feature or group of features, because this would be very specific to the dataset, and may be affected by the distribution of the malice types in the dataset. It could be more useful to see the histogram of the feature scores among the different feature categories that we employ in our classification process. Fig.11 illustrates the histogram of the logarithm of the feature scores among each feature category, while Table 3 shows the number of features generated per each feature category. As shown in the figure, each category has a percentage of its features with feature scores more than -2 (i.e. falling in the top 50% features), providing a motivation for employing all these features for the classification process.

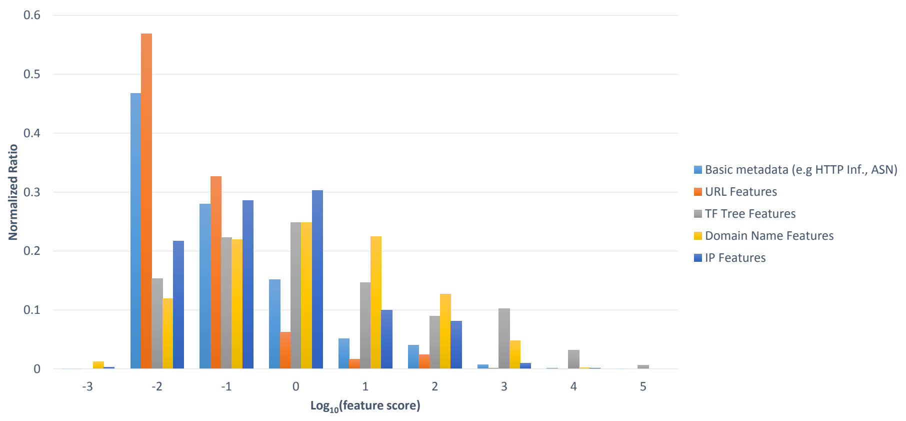

Fig. 11. Histograms of feature scores among feature categories

6 Conclusion and Future Work

This paper presented ADAM a system that uses machine learning over simple network artifacts that are inferred during dynamic webpage execution. ADAM's goal is to detect whether a webpage is malicious or not, and to identify the type of malice if the webpage was found malicious. Under cross-validation and a dataset that spans different types of attack behavior, the system was able to detect malicious webpages with an accuracy of 93% identifying injection and derver-side backdoor vulnerabilities as the main areas requiring detection improvement. Excluding injection samples from the dataset has resulted in an accuracy reaching 96%. Additionally, the malice labeling module was able to detect the label(s) of malicious TFs exactly in about 91% of the cases, with a difference of one and two labels in 6% and 3% of the cases respectively.

Several directions can be explored to extend this work. Since many of the features have a dynamic nature over time, e.g., IP addresses, an adaptive mechanism will be needed to capture such dynamic changes. Furthermore, more studies could be done to enhance the accuracy of the model presented in this paper in order to better detect injection and server side backdoor attacks, in addition to identify exploit attacks.

References

1. M. Antonakakis, R. Perdisci, D. Dagon, W. Lee, and N. Feamster. Building a dynamic reputation system for DNS. In *USENIX Security*, 2010.
2. M. Antonakakis, R. Perdisci, W. Lee, N. V. II, and D. Dagon. Detecting malware domains at the upper DNS hierarchy. In *USENIX Sec. Sym.*, 2011.
3. M. Bailey, J. O. J. Andersen, Z. Mao, F. Jahanian, and J. Nazario. Automated classification and analysis of Internet malware. In *RAID*, 2007.
4. U. Bayer, P. M. Comparetti, C. Hlauschek, C. Krügel, and E. Kirda. Scalable, behavior-based malware clustering. In *NDSS*, 2009.
5. L. Bilge, E. Kirda, C. Kruegel, and M. Balduzzi. EXPOSURE: Finding malicious domains using passive DNS analysis. In *NDSS*, 2011.

6. A. Blum, B. Wardman, T. Solorio, and G. Warner. Lexical feature based phishing URL detection using online learning. In *AISec*, 2010.
7. D. Canali, M. Cova, G. Vigna, and C. Kruegel. Prophiler: a fast filter for the large-scale detection of malicious web pages. In *WWW*, 2011.
8. J. Chang, K. K. Venkatasubramanian, A. G. West, and I. Lee. Analyzing and defending against web-based malware. *ACM Computing Surveys*, 45(4), 2013.
9. M. Egele, T. Scholte, E. Kirda, and C. Kruegel. A survey on automated dynamic malware-analysis techniques and tools. *ACM Computing Surveys*, 44(2), 2008.
10. M. Felegyhazi, C. Kreibich, and V. Paxson. On the potential of proactive domain blacklisting. In *LEET*, 2010.
11. 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.
12. G. Gu, P. Porris, V. Yegneswaran, M. Fong, and W. Lee. Bothunter: Detecting malware infection through IDS-driven dialog correlation. In *USENIX Security*, 2007.
13. G. Gu, J. Zhang, and W. Lee. BotSniffer: Detecting botnet command and control channels in network traffic. In *NDSS*, 2008.
14. S. Hao, M. Thomas, V. Paxson, N. Feamster, C. Kreibich, C. Grier, and S. Hollenbeck. Understanding the domain registration behavior of spammers. In *IMC*, 2013.
15. C. Kolbitsch, P. M. Comparetti, C. Kruegel, E. Kirda, X. Zhou, and X. Wang. Effective and efficient malware detection at the end host. In *USENIX Security Symposium*, 2009.
16. D. Kong and G. Yan. Discriminant malware distance learning on structural information for automated malware classification. In *KDD*, 2013.
17. J. Ma, L. K. Saul, S. Savage, and G. M. Voelker. Beyond blacklists: Learning to detect malicious web sites from suspicious URLs. In *KDD*, 2009.
18. J. Ma, L. K. Saul, S. Savage, and G. M. Voelker. Learning to detect malicious urls. *ACM Trans. Intell. Syst. Technol.*, (3):30:1–30:24, May 2011.
19. D. K. McGrath and M. Gupta. Behind phishing: An examination of phisher modi operandi. In *LEET*, 2008.
20. F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, et al. Scikit-learn: Machine learning in python. *The Journal of Machine Learning Research*, 12:2825–2830, 2011.
21. N. Provos, P. Mavrommatis, M. A. Rajab, and F. Monrose. All your iFRAMEs point to us. In *USENIX Security*, 2008.
22. N. Provos, D. McNamee, P. Mavrommatis, K. Wang, N. Modadugu, et al. The ghost in the browser analysis of web-based malware. In *HotBots*, 2007.
23. K. Thomas, C. Grier, J. Ma, V. Paxson, and D. Song. Design and evaluation of a real-time url spam filtering service. In *IEEE Security and Privacy*, 2011.
24. L. Xu, Z. Zhan, S. Xu, and K. Ye. Cross-layer detection of malicious websites. In *CODASPY*, 2013.
25. T.-F. Yen, A. Oprea, K. Onarlioglu, T. Leetham, W. Robertson, A. Juels, and E. Kirda. Beehive: Large-scale log analysis for detecting suspicious activity in enterprise networks. In *ACSAC*, 2013.