Fast Model Learning for the Detection of Malicious Digital Documents

Daniel Scofield
Assured Information Security
dan@dscofield.com

Craig Miles
Assured Information Security
craig@craigmil.es

Stephen Kuhn
Air Force Research Laboratory
stephen.kuhn@us.af.mil

ABSTRACT

Modern cyber attacks are often conducted by distributing digital documents that contain malware. The approach detailed herein, which consists of a classifier that uses features derived from dynamic analysis of a document viewer as it renders the document in question, is capable of classifying the disposition of digital documents with greater than 98% accuracy even when its model is trained on just small amounts of data. To keep the classification model itself small and thereby to provide scalability, we employ an entity resolution strategy that merges syntactically disparate features that are thought to be semantically equivalent but vary due to programmatic randomness. Entity resolution enables construction of a comprehensive model of benign functionality using relatively few training documents, and the model does not improve significantly with additional training data.

CCS CONCEPTS

• **Security and privacy → Intrusion/anomaly detection and malware mitigation**; **Malware and its mitigation**;

KEYWORDS

malware detection, dynamic analysis, anomaly detection

ACM Reference Format:
Daniel Scofield, Craig Miles, and Stephen Kuhn. 2017. Fast Model Learning for the Detection of Malicious Digital Documents. In *SSPREW-7 : 7th Software Security, Protection, and Reverse Engineering / Software Security and Protection Workshop , December 4–5, 2017, Orlando, FL, USA.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3151137.3151142

1 INTRODUCTION

Attackers often embed malware in digital documents [21], including those in formats such as Adobe PDF and Microsoft Word. When an unsuspecting user opens a malicious document, the malware embedded therein executes and compromises the user's system. Since system compromise is undesirable [1, 13], methodologies and tools for determining the disposition of a document, either malicious or benign, are needed.

Many approaches for classifying documents have been pursued [19]. One approach is to check for anomalies in static features extracted from a document [6, 14, 20]. Another approach, commonly employed by anti-virus scanners, is to test documents against byte-signatures derived from previously seen malicious documents [8]. Yet another approach works by monitoring the run-time behavior of a document viewer for unexpected actions as it renders the document [3, 6].

All of the aforementioned approaches for malicious document detection must be trained on or seeded with characterizations of previously encountered malicious and/or benign documents. For instance, traditional antivirus systems rely on curated databases of byte-signatures to detect malicious documents and machine learning approaches rely on models that are trained using features (weighted byte n-grams, dynamic execution artifacts, etc.) extracted from a corpus containing malicious and/or benign documents.

It is a common failing to assume that the size of one's corpus or feature set is of the utmost importance [16], as is evidenced by the gargantuan malware data sets[1] hoarded by cyber-security researchers and analysts, the prevalence of up-to-the-minute malware feeds hawked[2] by commercial vendors, and the many thousands of results arising from a Google Scholar search for the phrase *large corpus*. Though most researchers are aware of the law of diminishing returns and could even quantify the marginal utility of training on an extra datum if so pressed, it is nonetheless tempting to train on as much data as possible. Given this temptation, it is perhaps unsurprising that relatively little work has been done to determine just how small of a corpus or feature set one can maintain while still attaining high accuracy.

In this work, we show that a model built using only a small set of about 1,250 exemplar features is sufficient to classify the disposition of PDF documents with near perfect accuracy, while just 350 exemplar features are enough to attain better than 90% accuracy. Further, the number of benign PDFs that are needed to learn the classification model is itself very small. As few as 20 benign PDFs are sufficient to build a model that accurately characterizes the class of all benign PDFs.

The features we employ correspond to the runtime interactions that a document viewer makes with its underlying operating system while it is rendering a document. Such interactions include, e.g., opening a file or reading a registry key. For instance, one of our features might record that the document viewer wrote data to the file on path *C:\example.txt*. Concretely, a feature records information about the invocation of a Windows system call by the document viewer.

Our classifier is based on the idea that document viewer / OS interactions arising from the rendering of some benign documents can be aggregated into a whitelist. Thereafter, any other document that induces the document viewer to make non-whitelisted requests

SSPREW-7 , December 4–5, 2017, Orlando, FL, USA

ACM ISBN 978-1-4503-5387-8/17/12.
https://doi.org/10.1145/3151137.3151142

[1] E.g., see https://www.kaggle.com/c/malware-classification/data
[2] Team Cymru's Malware Binary Feed - https://www.team-cymru.com/malware-data.html

to the OS can be deemed malicious. However, a naive strategy based on this whitelisting approach fails outright due to the large degree of programmatic randomness present in system call arguments. E.g., the document viewer might always write logging information to a temporary file with a runtime generated random file name. Since the file paths corresponding to these file writes differ syntactically across runs, no whitelist of finite size (modulo machine limits) will ever be sufficient to characterize all possible future write requests.

To overcome the issue of programmatic randomness in the features, we present a novel approach for distilling a set of features down to a minimal set of exemplars. The distillation process is a form of entity resolution; i.e., features that differ syntactically yet which represent the same semantic interaction between the document viewer and the OS are merged together into a single exemplar. Our entity resolution strategy merges features that only differ due to instances of programmatic randomness, and is facilitated by way of a heuristic threshold over Levenshtein [15] edit distance. In particular, if the string representation of a feature can be transformed into the string representation of another feature using fewer than some threshold of edit operations, then the two features ought to be merged into one.

Once the exemplar whitelist is generated, classification is conducted by again monitoring the document viewer as it renders the new document in question. The observed system call invocations are featurized in the same manner as before and compared to the distilled whitelist. The expectation is that any anomalous features arising from a malicious document will neither match nor merge with any of the whitelisted exemplars. If the number of those unmergeable features exceeds another threshold, then the document is declared to be malicious.

What follows in the ensuing sections is a complete description of the aforementioned approach for document disposition classification, and also an empirical evaluation to determine how well it classifies digital documents of the Adobe PDF format. Our contributions include:

(1) We show that an entity resolution strategy that elides instances of programmatic randomness in a whitelist can greatly reduce the whitelist's size while still allowing for high accuracy digital document classifications.
(2) We show that such a whitelist can be constructed by monitoring the system calls invoked by a document viewer as it renders just a small number of benign documents.
(3) We describe the aforementioned classification system in detail, present the results of an empirical evaluation thereupon, and discuss its strengths and limitations.

The rest of this paper is organized as follows: In §2, we describe our featurization, feature set reduction, and classification strategies in detail. In §3, we describe how the classifier was evaluated and present the results. Limitations of the proposed approach are discussed in §4. Related works are discussed in §5 and conclusions follow in §6.

2 APPROACH

In this section, we describe the features we use for classification (§2.1), our feature merging strategy based on a process of entity resolution (§2.2), and our classifier (§2.3).

2.1 Featurization

Our classification approach utilizes features obtained through dynamic analysis on a document viewer as it renders a document. Each recorded feature embeds information about the invocation of a system call by the document viewer. Invoked system calls serve as useful features for discriminating between malicious and benign documents [5] since any user-space program, like a document viewer, must make use of them to interact with the underlying operating system. We use a custom introspective [7] hypervisor to record the system calls invoked by the document viewer, however other approaches based on dynamic instrumentation [9, 17] or a custom kernel driver could also serve to record system call invocations.

A new feature is recorded each time the document viewer process, or one of its children (e.g., an exploit might start cmd.exe), invokes one of the system calls shown in the first column of Table 1. Explanations for these system calls are available on Microsoft's Dev Center website[3]. A feature is recorded as a 3-tuple of the form

$$image, action, object$$

where *image* is the file name of the disk image of the process that made the system call, *action* is the semantic action associated with the observed system call (i.e., create, open, read, write, query, or delete), and *object* is the name of or the path to the object upon which the specified semantic action is to be performed. Table 1 also provides the semantic action and a description of the object associated with each of the supported system calls. Some examples of recorded features, plus explanations, are shown in Table 2. While the system calls we handle appear to cover only a small set of file system, registry, and process related behaviors, most complex behaviors (e.g., networking) are handled by the kernel as combinations of these simpler behaviors and are therefore captured as well.

2.2 Entity Resolution

The features obtained using the methodology of §2.1 on a document viewer as it renders documents evince a great deal of runtime generated randomness. In particular, the file paths and names encoded in the features' object fields often contain substrings of obviously random characters. For example, the file names embedded in the object fields of example features #1 and #3 in Table 2 appear to be of the form *{6 random hexadecimal characters}.log*. Though those two features appear to be semantically equivalent in the sense that they both reflect the writing to a log file by the document viewer, they differ syntactically.

This prevalence of runtime generated randomness precludes the creation of a comprehensive whitelist of such features that characterizes the class of benign documents. Rather, an entity resolution procedure is needed that elides instances of programmatic randomness in the collected features, and thereby provides a means to recognize that two such features are semantically equivalent even if they are syntactically disparate. To that end, we employ a heuristic entity resolution technique based on Levenshtein edit distance to merge semantically equivalent yet syntactically disparate features

[3] https://msdn.microsoft.com/en-us/library/windows/hardware/ff567122(v=vs.85).aspx

System Call	Action	Object
ZwCreateFile	Create	Path to file to be created.
ZwOpenFile	Open	Path to file to be opened.
ZwReadFile	Read	Path to file to be read.
ZwWriteFile	Write	Path to file to be written.
ZwCreateKey	Create	Path to registry key to be created.
ZwOpenKey	Open	Path to registry key to be opened.
ZwQueryKey	Query	Path to registry key to be queried.
ZwDeleteKey	Delete	Path to registry key to be deleted.
ZwSetValueKey	Write	Path to registry key value to be set.
ZwDeleteValueKey	Delete	Path to registry key value to be deleted.
ZwEnumerateValueKey	Query	Path to registry key to be enumerated.
ZwCreateProcess	Create	Disk image name of the created process.

Table 1: System calls monitored by our dynamic analysis, with their corresponding semantic actions and objects.

#	Recorded Feature	Explanation
1	reader.exe,write, C:\temp\2f358b.log	Denotes that reader.exe process attempted to write a file on the specified path.
2	reader.exe,read, C:\docs\file.pdf	Denotes that reader.exe process attempted to read a file on the specified path.
3	reader.exe,write, C:\temp\a632cf.log	Denotes that reader.exe process attempted to write a file on the specified path.
4	reader.exe,create,firefox.exe	Denotes that reader.exe process spawned a child from the firefox.exe disk image.

Table 2: Example features recorded by our analysis, with explanations.

into a single exemplar feature. The set of such exemplar features extracted from a sufficiently representative corpus of benign documents is then interpreted as comprising a comprehensive exemplar whitelist characterizing the class of benign documents.

We define any two features, a and b, to be α-equivalent if

$$\frac{Lev(a,b)}{n} < \alpha$$

where $Lev(a,b)$ denotes the Levenshtein edit distance between features a and b, and n is the string length of the longer feature. Then, letting C be a set of features obtained from the dynamic analysis on the rendering of a corpus of benign documents, we say that $\mathcal{W}_\alpha$ is an exemplar whitelist for C if and only if

$$\forall c \in C, \exists w \in \mathcal{W}_\alpha : c \text{ is } \alpha\text{-equivalent to } w$$

As an example, assume that the features shown in Table 2 arose from the dynamic analysis of a benign PDF. Also assume $\alpha = 0.2$. The edit distance between the first and second example features is 16 and the length of the longer of the two is 35, and $\frac{16}{35} \not< 0.2$, thus those two features are not α-equivalent and should not be merged. Conversely, the edit distance between the first and third example features is 5 and they both have the same length of 35, and $\frac{5}{35} < 0.2$, thus those two features are α-equivalent and should be merged. It turns out the first and the third features are the only pair among the example features that are α-equivalent. As such, the whitelist corresponding to this initial set of example features contains the second feature, the fourth feature, and either the first or the third feature (the other having been merged).

Naively, one can find a subset of C that meets this definition by starting with an empty whitelist, $\mathcal{W}_\alpha$, and iterating over every feature, $c \in C$. At each step, check c for α-equivalence with everything in $\mathcal{W}_\alpha$ and add c to $\mathcal{W}_\alpha$ only if $\nexists w \in \mathcal{W}_\alpha$ such that c is α-equivalent to w. However, this naive strategy tends to be slow since each successive feature is compared to progressively more features as $\mathcal{W}_\alpha$ grows, leading to an asymptotic worst case bound of $O(|C|^2 L)$ where L is the worst-case cost of calculating Levenshtein edit distance (i.e., the length of the longest $c \in C$ squared).

Since the naive approach for constructing an exemplar whitelist doesn't scale, a more efficient strategy is required. Empirically, features that are semantically equivalent tend to be lexicographically close to one another. I.e., they tend to have long coinciding prefixes. We used this observation to inform a revised strategy that provides for a significant reduction in the number of required comparisons when constructing $\mathcal{W}_\alpha$. The revised strategy is nearly equivalent to the aforementioned, except rather than comparing each successive $c \in C$ to the entirety of the current whitelist, instead c is tested for α-equivalence against just the two features in $\mathcal{W}_\alpha$ that immediately precede or succeed it lexicographically. When $\mathcal{W}_\alpha$ is stored in lexicographical order, this revised strategy has worst case complexity of just $O(|C|L)$.

To show that the revised strategy for exemplar whitelist generation still provides for strong entity resolution as compared to the exhaustive approach, we constructed 2,000 exemplar whitelists, for $\alpha = 0.05$ and $\alpha = 0.35$ (cf. §3.3), from 2,000 pairs of Adobe PDF documents. While doing so, we recorded the amount of feature merging that occurred in terms of the number of features that 1) merged with just their lexicographic neighbor(s), 2) merged with just one or more non-neighbors, 3) merged with both a neighbor and a non-neighbor, or 4) didn't merge with any other features. The results, showing the averaged amount of merging across the 2,000 experiments, are shown in Table 3.

% of features merging with	α = .05	α = .35
No other feature	0.06	0.06
Only neighboring feature(s)	0.05	0.04
Only non-neighboring feature(s)	0.14	0.12
Neighboring and non-neighboring feature(s)	99.7	99.8

Table 3: Experimental results showing the amount of feature merging with both lexicographic neighbors and non-neighbors.

#	Recorded Feature
A	reader.exe,write,C:\temp\9467f2.log
B	reader.exe,write,C:\downloads\payload.exe
C	reader.exe,create,payload.exe

Table 4: Example dynamic features arising from the rendering of a document of unknown disposition.

The results show that on average just 0.06% of features merge with one or more non-neighbors but not with either neighbor. This indicates that testing just lexicographically neighboring features for α-equivalence provides a strong approximation to the exhaustive method.

2.3 Classification

Benign versus malicious classification is implemented with a heuristic rule based classifier. A document is classified as malicious when the number, k, of non-mergeable features observed as the document viewer renders the document exceeds a threshold, β. Given a feature merging threshold, α, this k for a suspect document is determined as follows.

Let $\mathcal{W}_\alpha$ be the exemplar whitelist generated via the method of §2.2 on a corpus of benign documents for some particular prechosen α, and let S be the set of features collected from rendering the suspect document under the dynamic analysis. For any feature $s \in S$, let p_s and n_s respectively represent the two exemplar features that lexicographically precede and succeed s in $\mathcal{W}_\alpha$. Further, let $F : S \rightarrow \{0, 1\}$ be defined

$$F(s) = \begin{cases} 0 & \text{if } s \text{ is } \alpha\text{-equivalent to } p_s \text{ or } n_s, \text{or} \\ 1 & \text{otherwise} \end{cases}$$

Then

$$k = |\{ s \in \mathcal{S} \setminus \mathcal{W}_\alpha \mid F(s) \neq 0 \}|$$

The document in question is declared malicious if and only if $k > \beta$.

Continuing with the example from the preceding section and letting $\beta = 1$, we now assume that a new document of unknown disposition is to be classified. Further, assume that the features arising from the dynamic analysis of the viewer as it renders that document are those shown in Table 4. Of these new features, feature A merges with a feature already in the whitelist (either #1 or #3 from Table 2). However, neither of the other two new features merge with or match any features in the exemplar whitelist, thus $k = 2$ and $k > \beta$, hence the new document is classified malicious.

3 EVALUATION

In this section, we present the results of an evaluation we conducted to determine the efficacy of our approach for document disposition classification as detailed in §2. The evaluation places particular on emphasis on answering three questions:

- §3.1 To what extent does feature merging via entity resolution reduce the size of the retained feature set?
- §3.2 How many benign document renderings must be observed in order to collect a comprehensive set of merged exemplar features?
- §3.3 What level of classification accuracy is attainable using the whitelist comprised of those merged exemplar features?

To evaluate the approach, we sought to classify the disposition of PDF documents by monitoring their renderings under the Adobe Reader v9.0[4] document viewer. This particular version of Adobe Reader was selected for use since it is known to be susceptible to several publicly disclosed exploits.

We used an open-source data set of both benign and malicious PDF documents available[5] from Contagio to conduct the evaluation. From the Contagio set, we used 311 of the benign PDFs for training, and another 33 benign PDFs plus 33 malicious PDFs for testing. The training set size was chosen arbitrarily and our results show that it was more than sufficient; i.e., training on additional benign documents would not have significantly impacted the learned model (cf. Figure 2). The size of the malicious test set was determined by the number of malicious documents inside of the Contagio dataset that are known to target Adobe Reader v9.0. These malicious PDFs include 5 examples of a CVE-2010-2883 exploit, 15 examples of CVE-2010-0188 exploit, and 13 examples of CVE-2011-2462 exploit.

3.1 Feature Set Reduction

As the feature merging threshold α increases, so too does the percentage of features that get merged together. Figure 1 shows the percentage of original features derived from the benign training set that are retained after feature merging for various α. It is interesting to note that at even relatively low α of, say, 0.05, a reduction of more than half is observed. In other words, over half of the features obtained by monitoring Adobe Reader v9.0 while it renders benign documents only vary due to small amounts of programmatic randomness. Further, as will be seen in §3.3, accuracy as high as 90% can even be attained using α up to 0.35, which provides a feature set size reduction of 85%.

3.2 A Comprehensive Whitelist of Exemplar Features

Figure 2 illustrates the growth of the whitelist with and without feature merging as the number of rendered documents is varied. With no feature merging (i.e., $\alpha = 0$), the size of the whitelist monotonically increases with growth proportional to the number of documents rendered. In contrast, feature merging appears to provide an upper bound on the size of the whitelist irrespective of the number of documents rendered. For instance, with $\alpha = 0.05$ (cf.

[4] ftp://ftp.adobe.com/pub/adobe/reader/win/9.x/9.0/enu/AdbeRdr90_en_US.exe - Installer MD5 Sum: f41aa5dec8c9137b2ff4174ec47d8129

[5] http://contagiodump.blogspot.com/2013/03/16800-clean-and-11960-malicious-files.html

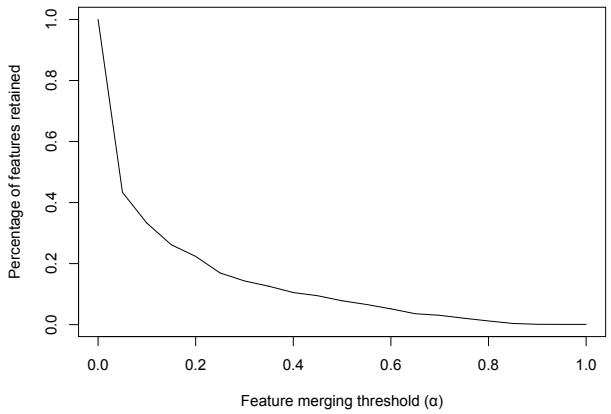

Figure 1: Effect of varying α in terms of the percentage of original features retained after feature merging via entity resolution.

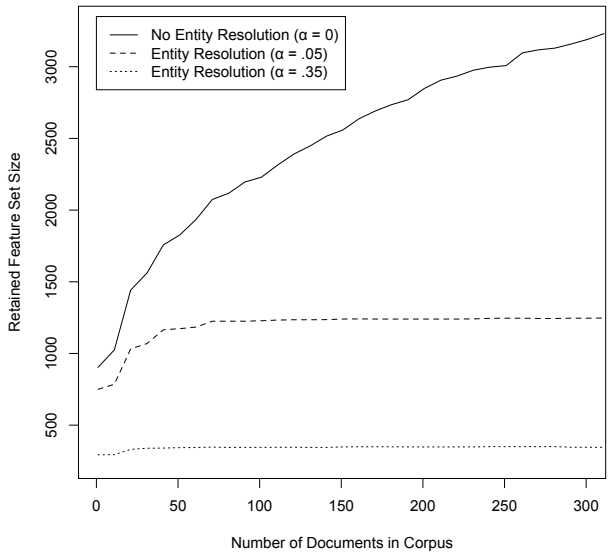

Figure 2: Growth rate of feature whitelist with and without entity resolution based feature merging; $\alpha = 0.05$.

§3.3), the size of the whitelist tops out at 1,247 exemplar features, and with $\alpha = 0.35$ (ibid.), it tops out at 345 exemplar features.

The quickness with which the upper bound on the whitelist's size is reached is also interesting. At $\alpha = 0.05$, rendering just 60 randomly selected documents from the training set is sufficient to obtain 95% of the whitelist that is obtained by analyzing the entire training set of 311 documents, and at $\alpha = 0.35$, just 20 benign documents are necessary to obtain 95% of the whitelist constructed

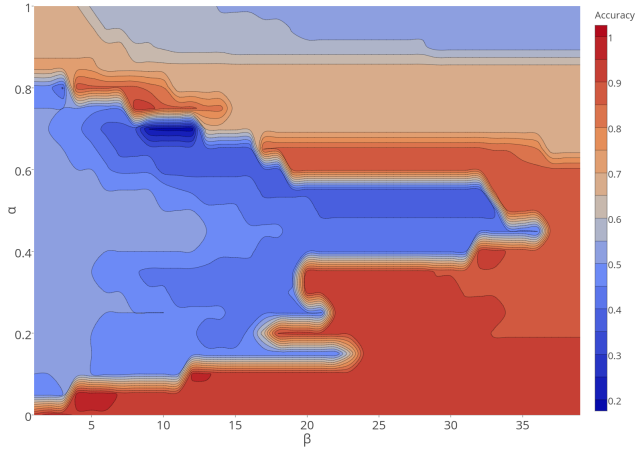

Figure 3: Classification accuracy attained as α and β are varied.

using the entire training corpus. This shows that the vast majority of viewer / OS interactions differ only slightly due to runtime variations, and thus a small corpus of no more than 100 benign documents is more than sufficient to build up a comprehensive whitelist of exemplar features. We leave the question of whether the Contagio data set of benign PDFs is representative of the universe of benign PDFs to future work.

3.3 Classifier Accuracy

A contour map that shows how classification accuracy varies with α and β is presented in Figure 3. Classification accuracy is defined

$$\frac{TP + TN}{P + N}$$

where TP and TN are respectively the count of true positive and of true negative classifications, and P and N are respectively the count of malicious and of benign documents in the testing set.

The contour map shows that nearly perfect classification accuracy (> 98% with precision 1.0 and recall 0.97) is attained with $\alpha = 0.05$ and $\beta = 5$, and that good accuracy (> 90% with precision 1.0 and recall 0.85) is attained with $\alpha = 0.35$ and $\beta = 25$ (and at many other points as well). This means that an analyst can perform nearly perfect classifications using a whitelist containing just 1,247 exemplar features (cf. §3.2), and with minimal trade-off can even use a significantly smaller whitelist containing just 347 exemplar features.

3.4 Discussion

The results of the evaluation we conducted indicate that a classifier using features derived from the system calls that a document viewer makes as it renders documents can attain near perfect accuracy. Since related works in digital document classification have also attained similarly high classification accuracy (cf. §5), this result is pleasing but far from revolutionary. However, the present work nonetheless stands out because we have shown that the benign class of PDF documents can be characterized by a very small set

of viewer / OS interactions that are induced by those documents' collective rendering. Further, these interactions can be learned by monitoring the rendering of just a few documents; i.e., small data is sufficient. These findings lend to both the credence and feasibility of NIST's recommendation (cf. NIST Special Publication 800-53 [12]) that both federal and private organizations should employ so-called detonation chamber based security controls, since they are shown herein to be efficacious and to require only minimal data processing facilities.

4 LIMITATIONS

Since our approach is based on the dynamic analysis of document viewers as they render digital documents, it faces a variation on the same code coverage issues faced by most dynamic analysis strategies. That is, the dynamic analysis component can only observe the interactions between the user space document viewer and the operating system that actually occur at runtime. If a malicious document is structured in such a way that its payload only executes as a result of some human interaction with the document viewer, such as the clicking of a button, then our approach will fail to observe the interactions that arise from execution of that payload unless its triggering mechanism is actually actuated at runtime. In order to overcome this limitation, one would either need to employ the use of software automation technologies like AutoIt[6] to automate the clicking of links and buttons throughout the viewer's interface or rely on a human-in-the-loop to manually actuate interface artifacts like links and buttons. Similarly, if an exploit payload is configured to desist if it detects that it is being executed within a sandboxed or dynamic analyusis environment, then our approach would also fail to observe the malicious actions that the exploit would otherwise perform.

In our evaluation, we tested against Adobe Reader v9.0 and therefore our malicious test set only included PDF documents that we confirmed to work against Adobe Reader v9.0. However, due to the brittle nature of exploits, our evaluation system would likely fail to detect malicious PDFs targeting other version of Adobe Reader. If our approach were to be adopted to detect malicious PDFs en masse, then separate models of expected functionality would need to be generated for each version of the document viewer.

Conversely, though we only tested against three different PDF exploit CVEs (cf. §3) due to a lack of publicly available data, we are confident that our model for classifying the disposition of PDFs with respect to Adobe Reader 9.0 would generalize to other CVEs that target Adobe Reader 9.0 as well. The argument for this is logical: Since we believe our model comprehensively characterizes all the actions Adobe Reader 9.0 might undertake while rendering a benign document and since a PDF exploit CVE would only exist if it induced the application into performing unexpected actions, then our classifier would also treat as anomalous the unexpected actions arising from exploitation by any other PDF exploit CVE targeting Adobe Reader 9.0.

Another potential limitation of our approach is that an attacker with knowledge of the detector's threshold configuration (i.e., α and β) could craft their malicious document in such a way that it performs fewer than β anomalous actions. In so doing, the document would be able to evade detection. While such an attack against our detector is theoretically possible, it seems largely untenable in practice since good values for β tend to be very small (cf. Figure 3). In other words, an attacker would need to achieve their malicious aims while performing no more than a small handful of operations that invoke system calls. Since even a relatively simple action like listing a directory can result in many hundreds of system call invocations, we therefore believe that only the most trivial of attacks, such as the deletion of a single file, could pass undetected.

[6]https://www.autoitscript.com/

5 RELATED WORKS

The distribution of malicious documents has seen a steady rise as Advanced Persistent Threat (APT) and phishing campaigns have become the preferred avenue of attack for cyber-adversaries. As such, there is already a great deal of prior work in the area of malicious document detection.

We differentiate our work from that of our predecessors along two orthogonal dimensions. First, our approach detects malicious documents by actually monitoring the document viewer process for malicious activity. This is in contrast to the majority of prior work, which has generally sought to statically detect anomalies in the document. Such static detection can be evaded by attackers via the application of obfuscations to their malicious documents. In contrast, our method will observe every malicious interaction that a malicious document induces a document viewer to take, irrespective of any applied obfuscations. Second, of the few prior works similar to our own [3, 6] that also monitor the dynamic runtime of the document viewer, none have sufficiently addressed the sheer glut of dynamic features. We have shown that it is not necessary to maintain a gigantic whitelist of system call based features to characterize the set of benign documents.

We now present a brief survey of related works so as to triangulate the present work with respect to that which preceded it. A more comprehensive survey of works related to the classification of digital documents is available in Nissim, et al. [19].

The most similar prior work to our own is Engleberth, et al. [6]. They too monitored dynamic behaviors of document viewers to construct a white list, and they even went so far as to generalize their features in a manner that appears similar to our entity resolution. However, the only record of their work exists in the form of a short slide-deck presentation, with no accompanying paper publication, and thus details are scant as to how their strategy for feature generalization actually functioned. Conversely, herein we have fully defined a principled approach for dynamic feature space reduction via entity resolution. Bazzi and Onozato [3] also employed the use of dynamic document viewer features for malicious document detection, but they employed no feature space reduction technique.

Other approaches for malicious document detection have used static features instead of the dynamic features we use. The prior work using static features for classification can itself be broken down into two groups: those that classify based on a document's metadata and/or structure, and those that classify based on the discerned functionality of code embedded within the document.

Classification approaches that turn on metadata and structural features [18, 20, 22, 23] have often been successful at binary classification of malicious documents. However, such approaches can be evaded if the attacker applies obfuscations. The other style of static analysis seen in prior work, which turns on the code embedded in documents, are feasible because the PDF specification allows for Javascript code to be embedded within a PDF document. It has been noted [24] that many attacks delivered via malicious PDF documents are conducted by embedding malicious Javascript within said documents. As such, approaches [14, 24] which extract, tokenize, and featurize embedded Javascript have also shown some success. However, these approaches can also be evaded when the attacker obfuscates the JavaScript. Further, the presence of obfuscated Javascript is not in itself a sufficient discriminator by which a document can be declared malicious, since JavaScript in benign PDF documents is also often obfuscated for intellectual property protection purposes.

Aside from this work's focus on malicious document detection, approaches for feature set reduction have been proposed in the broader malware detection literature. For instance, after observing that a glut of string-typed features obtained by their analysis was causing poor efficiency in their malware familial classifier, Islam, et al. [10] reduced their retained feature set by throwing away those features that were not strongly discriminative with respect to any particular malware family. Their approach effectively sets a lower-bound threshold on the common term frequency cross inverse document frequency (TFxIDF) weighting scheme such that any features with sufficiently low weight are discarded. Such a feature reduction strategy differs from our own in that it discards features which are not ideal discriminators, whereas our approach instead uses entity resolution to merge syntactically different yet semantically equivalent features.

Another approach for feature set reduction, BitShred [11], collapses the feature space by hashing the individual features. The cost of such an approach is the introduction of hash collisions, whence otherwise disparate features become indistinguishable from one another. This strategy, like our own, can cause two syntactically different features to be merged together, however the criteria by which their features are merged cannot be considered as a form of entity resolution. Rather, whether two features merge under BitShred, i.e., whether they hash to the same value, is merely a random artifact of the employed hash function. In contrast, our approach attains similar reductions in feature set size, but does so with the added benefit that only features thought to be semantically equivalent are merged.

Bailey, et al. [2] studied a dataset in which dynamic analysis features were present for the purposes of familial classification and applied Normalized Compression Distance (NCD)[4] to mitigate the effects of randomness when comparing groups of dynamic features between malware samples. While this technique is similar in effect to the use of entity resolution in that it minimizes the effect of programmatic randomness, it is distinguished from the present work in that they utilized an unsupervised clustering approach using NCD to pairwise compute the similarity of two files. In contrast, our approach is a supervised technique that applies entity resolution to every feature in the training corpus and merges features where appropriate to create a minimal whitelist that characterizes the benign class.

6 CONCLUSION

We developed a classifier for the disposition of digital documents that requires training on only a very small data set of benign documents and which only retains a very small set of exemplar features. This small-data classifier attains similar accuracy to the big-data approaches previously detailed in the literature. In particular, our approach has been shown to attain 98% accuracy in classifying PDFs as either malicious or benign. Further, our classification approach entails so few comparisons that it can easily be performed in an online fashion. As such, the proposed strategy is suitable for use in conjunction with any sandboxing or detonation chamber based technologies that provide for the tracing of system calls.

ACKNOWLEDGMENTS

The authors thank Austin Benincasa and Salvatore Paladino for their contributions. The work is supported by Air Force Research Laboratory contract number FA8750-15-C-0017.

REFERENCES

[1] Ross Anderson, Chris Barton, Rainer Böhme, Richard Clayton, Michel JG Van Eeten, Michael Levi, Tyler Moore, and Stefan Savage. 2013. Measuring the cost of cybercrime. In *The economics of information security and privacy*. Springer, 265–300.

[2] Michael Bailey, Jon Oberheide, Jon Andersen, Z Morley Mao, Farnam Jahanian, and Jose Nazario. 2007. Automated classification and analysis of internet malware. In *International Workshop on Recent Advances in Intrusion Detection*. Springer, 178–197.

[3] Ahmad Bazzi and Yoshikuni Onozato. 2013. IDS for detecting malicious non-executable files using dynamic analysis.. In *APNOMS*. 1–3.

[4] Rudi Cilibrasi and Paul MB Vitányi. 2005. Clustering by compression. *IEEE Transactions on Information theory* 51, 4 (2005), 1523–1545.

[5] Artem Dinaburg, Paul Royal, Monirul Sharif, and Wenke Lee. 2008. Ether: malware analysis via hardware virtualization extensions. In *Proceedings of the 15th ACM conference on Computer and communications security*. ACM, 51–62.

[6] M Engleberth, Carsten Willems, and Thorsten Holz. 2009. Detecting malicious documents with combined static and dynamic analysis (Powerpoint Presentation). *Virus Bulletin* (2009).

[7] Tal Garfinkel, Mendel Rosenblum, et al. 2003. A Virtual Machine Introspection Based Architecture for Intrusion Detection.. In *NDSS*, Vol. 3. 191–206.

[8] Kent Griffin, Scott Schneider, Xin Hu, and Tzi-Cker Chiueh. 2009. Automatic generation of string signatures for malware detection. In *International Workshop on Recent Advances in Intrusion Detection*. Springer, 101–120.

[9] Galen Hunt and Doug Brubacher. 1999. DETOURS: BINARY INTERCEPTION OF WIN 32 FUNCTIONS. In *3rd Usenix Windows NT Symposium*.

[10] Rafiqul Islam, Ronghua Tian, Lynn Batten, and Steve Versteeg. 2010. Classification of malware based on string and function feature selection. In *Cybercrime and Trustworthy Computing Workshop (CTC), 2010 Second*. IEEE, 9–17.

[11] Jiyong Jang, David Brumley, and Shobha Venkataraman. 2011. Bitshred: feature hashing malware for scalable triage and semantic analysis. In *Proceedings of the 18th ACM conference on Computer and communications security*. ACM, 309–320.

[12] Joint Task Force Transformation Initiative Interagency Working Group. 2013. *NIST Special Publication 800-53 Revision 4 - Security and Privacy Controls for Federal Information Systems and Organizations*. Technical Report. National Institute of Science and Technology (NIST).

[13] Suleyman Kondakci. 2009. A concise cost analysis of Internet malware. *Computers & Security* 28, 7 (2009), 648–659.

[14] Pavel Laskov and Nedim Šrndić. 2011. Static detection of malicious JavaScript-bearing PDF documents. In *Proceedings of the 27th Annual Computer Security Applications Conference*. ACM, 373–382.

[15] Vladimir I Levenshtein. 1966. Binary codes capable of correcting deletions, insertions and reversals. In *Soviet physics doklady*, Vol. 10. 707.

[16] Yun Li and Bao-Liang Lu. 2009. Feature selection based on loss-margin of nearest neighbor classification. *Pattern Recognition* 42, 9 (2009), 1914–1921.

[17] Chi-Keung Luk, Robert Cohn, Robert Muth, Harish Patil, Artur Klauser, Geoff Lowney, Steven Wallace, Vijay Janapa Reddi, and Kim Hazelwood. 2005. Pin:

building customized program analysis tools with dynamic instrumentation. In *ACM Sigplan Notices*, Vol. 40. ACM, 190–200.

[18] Davide Maiorca, Giorgio Giacinto, and Igino Corona. 2012. A pattern recognition system for malicious PDF files detection. In *International Workshop on Machine Learning and Data Mining in Pattern Recognition*. Springer, 510–524.

[19] Nir Nissim, Aviad Cohen, Chanan Glezer, and Yuval Elovici. 2015. Detection of malicious PDF files and directions for enhancements: a state-of-the art survey. *Computers & Security* 48 (2015), 246–266.

[20] Himanshu Pareek, P Eswari, N Sarat Chandra Babu, and C Bangalore. 2013. Entropy and n-gram analysis of malicious PDF documents. *Int J Eng Res Tech* 2, 2 (2013).

[21] Karthik Selvaraj and Nino Fred Gutierrez. 2010. The rise of PDF malware. *Symantec Security Response* (2010).

[22] Charles Smutz and Angelos Stavrou. 2012. Malicious PDF detection using metadata and structural features. In *Proceedings of the 28th Annual Computer Security Applications Conference*. ACM, 239–248.

[23] Nedim Šrndic and Pavel Laskov. 2013. Detection of malicious PDF files based on hierarchical document structure. In *Proceedings of the 20th Annual Network & Distributed System Security Symposium*.

[24] Cristina Vatamanu, Dragoş Gavriluţ, and Răzvan Benchea. 2012. A practical approach on clustering malicious PDF documents. *Journal in Computer Virology* 8, 4 (2012), 151–163.