Automated Model Learning for Accurate Detection of Malicious Digital Documents

DANIEL SCOFIELD and CRAIG MILES, Assured Information Security
STEPHEN KUHN, Air Force Research Laboratory

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. In particular, we describe and quantitatively evaluate a fully automated, document format–agnostic approach for learning a classification model that provides efficacious malicious document detection.

CCS Concepts: • Security and privacy → Intrusion/anomaly detection and malware mitigation; Malware and its mitigation;

Additional Key Words and Phrases: Malware detection, dynamic analysis, anomaly detection

ACM Reference format:
Daniel Scofield, Craig Miles, and Stephen Kuhn. 2020. Automated Model Learning for Accurate Detection of Malicious Digital Documents. Digit. Threat.: Res. Pract. 1, 3, Article 15 (August 2020), 21 pages.
https://doi.org/10.1145/3379505

1 INTRODUCTION

Attackers often embed malware in digital documents, such as Adobe PDF and Microsoft Word files. This happens so frequently, in fact, that malware-laden digital documents have constituted the most prevalent infection vector seen in the wild from 2010 up to the present [19]. 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, 14], methodologies and tools for determining the disposition of a document, either malicious or benign, are needed.

This is an extended version of our earlier paper, Fast Model Learning for the Detection of Malicious Digital Documents, which was published in SSPREW 2017.
This work has been supported by Air Force Research Laboratory contract numbers FA8750-15-C-0017 and FA8750-18-C-0140.
Authors’ addresses: D. Scofield and C. Miles, Assured Information Security, 1515 SW Fifth Avenue, Suite 1022, Portland, OR, 97201; email: dan@dscarfield.com, craig@craigmiles.com; S. Kuhn, Air Force Research Laboratory, 525 Brooks Road, Rome, NY 13441; email: stephen.kuhn@us.af.mil.
Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page.
Copyrights for third-party components of this work must be honored. For all other uses, contact the owner/author(s).
© 2020 Copyright held by the owner/author(s).
2576-5337/2020/08-ART15
https://doi.org/10.1145/3379505

Digital Threats: Research and Practice, Vol. 1, No. 3, Article 15. Publication date: August 2020.
Many approaches for classifying documents have been pursued [21]. One approach is to check for anomalies in static features extracted from a document [7, 15, 22]. Another approach, commonly employed by anti-virus scanners, is to test documents against heuristics and/or byte-signatures derived from previously seen malicious documents [9]. Yet another approach works by monitoring the runtime behavior of a document viewer for unexpected actions as it renders the document [3, 7].

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 a combination of heuristics and 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.

However, it is a common failing in this area of study to assume that the size of one’s corpus or feature set is of the utmost importance [17], as is evidenced by the gargantuan malware datasets hoarded by cyber-security researchers and analysts, the prevalence of up-to-the-minute malware feeds hawked 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. Likewise, for Microsoft Word documents, we also show that similarly small counts of exemplar features and benign documents serve to provide those same high levels of accuracy.

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 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 that 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 [16] edit distance. In particular, if the string representation of a feature can be transformed

---

1E.g., see https://www.kaggle.com/c/malware-classification/data.

2Team Cymru’s Malware Binary Feed: https://www.team-cymru.com/malware-data.html.

Digital Threats: Research and Practice, Vol. 1, No. 3, Article 15. Publication date: August 2020.
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.

All of the aforementioned was published previously [23], and the present document is an extended journal version of that past conference publication. While our previous publication described a method that provides fast model learning for the detection of malicious digital documents, it still relied on the presence of an expert human in the loop to select satisfactory model parameters that maximize classification accuracy. In this extended version, we present a new method to automate optimal model parameter selection, making the model learning process even faster. Additionally, we present an extended quantitative evaluation that demonstrates that our previously described approach generalizes well to other, previously untested, file format/document viewer application pairs. Finally, this journal version also contains a broader discussion on the internal and external validity of our experiments.

What follows then in the ensuing sections is a complete description of our approach for document disposition classification and an empirical evaluation to determine how well the approach classifies digital documents of the Adobe PDF and Microsoft Word formats. 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, including how it works and how to automatically select satisfactory model parameters.
4. We present the results of an empirical evaluation of our classifier and discuss its strengths and limitations.

The rest of this article is organized as follows: In Section 2, we describe our featurization, feature set reduction, and classification strategies in detail; and in Section 3, we present a case study example that shows actual data and demonstrates the classification approach on a real-world malicious PDF document. In Section 4, we describe how the classifier was evaluated and present the results thereof, and in Section 5, we present a method for automatically selecting model parameters that optimize classification accuracy. In Section 6, we describe the software that was developed in pursuit of the present work. Limitations of the proposed approach and threats to experimental validity are discussed in Section 7. Related works are discussed in Section 8 and conclusions follow in Section 9.

2 APPROACH

In this section, we describe the features we use for classification (Section 2.1), our feature merging strategy based on a process of entity resolution (Section 2.2), and our classifier (Section 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 [6], since any user-space program, like a document viewer, must make use of them to interact with the
Table 1. System Calls Monitored by Our Dynamic Analysis, with Their Corresponding Semantic Actions and Objects

| 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 2. Example Features Recorded by Our Analysis, with Explanations

| #   | 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. |

The underlying operating system. We use a custom introspective [8] hypervisor to record the system calls invoked by the document viewer, however, other approaches based on dynamic instrumentation [10, 18] 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. A feature is recorded as a 3-tuple of the form

\[
\text{image, action, object,} \tag{1}
\]

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.

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

The features obtained using the methodology of Section 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 \(\{6 \text{ random hexadecimal characters}\}.log\). 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 characterize 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 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 \(\alpha\)-equivalent if

\[
\frac{\text{Lev}(a, b)}{n} < \alpha,
\]

where \(\text{Lev}(a, b)\) 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 \(W_\alpha\) is an exemplar whitelist for \(C\) if and only if

\[
\forall c \in C, \exists w \in W_\alpha : c \text{ is } \alpha\text{-equivalent to } w.
\]

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 \(\frac{16}{35} < 0.2\), thus those two features are not 0.2-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 0.2-equivalent and should be merged. It turns out the first and the third features are the only pair among the example features that are \(\alpha\)-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, \(W_\alpha\), and iterating over every feature, \(c \in C\). At each step, check \(c\) for \(\alpha\)-equivalence with everything in \(W_\alpha\) and add \(c\) to \(W_\alpha\) only if \(\exists w \in W_\alpha\) such that \(c\) is \(\alpha\)-equivalent to \(w\). However, this naive strategy tends to be slow, since each successive feature is compared to progressively more features as \(W_\alpha\) grows, leading to an asymptotic worst-case bound of \(O(|C|^2L)\) 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 does not 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 \(W_\alpha\). 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 \(\alpha\)-equivalence against just the two features in \(W_\alpha\) that immediately precede or succeed it lexicographically. When \(W_\alpha\) 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 2K exemplar whitelists, for \( \alpha = 0.05 \) and \( \alpha = 0.35 \) (see Section 4.3), from 2K 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) did not merge with any other features. The results, showing the averaged amount of merging across the 2K experiments, are shown in Table 3.

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 \( \alpha \)-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, \( \beta \). Given a feature merging threshold, \( \alpha \), this \( k \) for a suspect document is determined as follows:

Let \( W_\alpha \) be the exemplar whitelist generated via the method of Section 2.2 on a corpus of benign documents for some particular pre-chosen \( \alpha \), 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 \( W_\alpha \). 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

\[
k = |\{ s \in S \setminus W_\alpha \mid F(s) \neq 0 \}|. 
\]

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
Table 5. Sampling of System Call Features in Benign Whitelist Model for Adobe Reader v9.0

| Recorded Feature                                                                 |
|----------------------------------------------------------------------------------|
| acrord32.exe,write,c:/documents and settings/user/local settings/application data/adobe/updater6/aumlib.log |
| acrord32.exe,read,c:/program files/adobe/reader 9.0/resource/font/minionpro-it.otf  |
| acrord32.exe,read,c:/program files/adobe/reader 9.0/resource/font/courierstd-oblique.otf |

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 CASE STUDY

In this section, we present a case study demonstrating usage of our technique to classify the disposition of a real-world PDF document. The goal of this demonstration is to better motivate our classification approach and the ensuing quantitative evaluation in Section 4 by showing the approach’s usefulness with respect to a real document that one might encounter in the wild.

Since it is a PDF document that is in question, we will need to make use of a model for Adobe Reader obtained by training on a corpus of benign PDFs. To that end, we trained just such a model, using the strategy of Section 2.1 and Section 2.2, on a corpus of a few hundred benign PDFs (the composition of our corpora is described in detail in Section 4). While featureizing the PDFs of the benign corpus, we observed that opening a benign PDF for rendering results on average in 956 features being recorded, and after term merging with $\alpha = 0.05$ (see Section 4 for justification), we were left with a trained whitelist model for PDFs containing 1,247 exemplar features. These exemplar features, a selection of which is presented in Table 5, reflect the things that Adobe Reader normally does when it is opening and rendering a benign PDF.

The particular PDF document under consideration is one that we know a priori exploits the CVE-2010-2883 buffer overflow vulnerability in Adobe Reader v9.0 to trigger execution of a malware payload. To use our approach to automatically classify the disposition of this suspect PDF document, we used our dynamic tracing feature collector to collect the system call trace features the document induced in Adobe Reader, and we used our entity resolution technique to see how many of those features merge with those in the benign PDF model. We observed that opening the malicious PDF resulted in 1,501 features being recorded, of which 929 did not merge with any features in the benign model (i.e., the recorded trace contained 929 anomalous features). Conversely, past experience (see Section 4) tells us that opening a previously unseen benign PDF results in a trace that contains on average just five non-mergable features—and, since 929 is so much greater than 5, we can reasonably conclude that this new PDF is indeed malicious.

By inspecting the features recorded in the trace resulting from opening the suspect document, we can also readily observe the anomalous behavior that resulted in this classification decision. In particular, we know from reverse engineering the PDF document in question that malformed data in an embedded TTF font induces a buffer overflow in Adobe Reader that leads to execution of an embedded malicious payload. The embedded payload is a dropper that drops a JavaScript payload to disk and then uses cmd.exe to run the JavaScript payload under the Windows Script Host cscript.exe. The JavaScript in turn drops a PE32 executable .exe payload to disk and then runs that payload. Finally, the .exe payload opens a persistent backdoor on the infected system. By looking at the PDF’s non-merging features in Table 6, we can see this scenario play out. Because these actions undertaken by both Adobe Reader and the child processes spawned by the exploit payload beneath it were so different from

4MD5: 851D895614645756999BD9F6E002C127.
5https://nvd.nist.gov/vuln/detail/CVE-2010-2883.
Table 6. Sampling of Non-mergeable (i.e., Anomalous) System Call Features Induced in Adobe Reader and Child Processes by Opening Malicious PDF Document

| Recorded Feature                                                                 |
|----------------------------------------------------------------------------------|
| acrord32.exe, writes, c:/docume 1/user/locals 1/temp/winword.js                  |
| acrord32.exe, creates, cmd.exe/operatingSystem-cmd.exe-1240                      |
| cmd.exe, creates, cscript.exe/operatingSystem-cscript.exe-1272                   |
| cscript.exe, reads, c:/docume 1/user/locals 1/temp/winword.js                    |
| cscript.exe, writes, c:/docume 1/user/locals 1/temp/tempq.tmp                    |
| cscript.exe, creates, tempq.tmp/temqp.tmp-1412                                  |
| tempq.tmp, writes, c:/docume 1/user/locals 1/temp/conime.exe                     |
| tempq.tmp, creates, conime.exe/operatingSystem-conime.exe-1452                   |

the actions normally taken by Adobe Reader (cf. Table 5), our classification strategy confidently deemed the file malicious.

4 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 Section 2. The evaluation places particular emphasis on answering four questions:

Section 4.1. To what extent does feature merging via entity resolution reduce the size of the retained feature set?

Section 4.2. How many benign document renderings must be observed to collect a comprehensive set of merged exemplar features?

Section 4.3. What level of classification accuracy is attainable using the whitelist composed of those merged exemplar features?

Section 4.4. Are the classification accuracy and feature space reduction results artifacts of a specific file format, or can they be generalized to other document types?

To evaluate the approach, we sought to classify the disposition of PDF documents by monitoring their renderings under the Adobe Reader v9.0 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 dataset of both benign and malicious PDF documents available 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 (see 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 a CVE-2010-0188 exploit, and 13 examples of a CVE-2011-2462 exploit.

To evaluate the generalizability of our results, we also tested with a Microsoft Word dataset provided to us from VirusShare. From this dataset, we took 30 malicious test documents including 14 examples of CVE-2012-0158,

4 ftp://ftp.adobe.com/pub/adobe/reader/win/9.x/9.0/en-US/AdbeRdr90_en_US.exe - Installer MD5 Sum: f41aa5dec8c9137b2ff4174ec47d8129.
5 http://contagiodump.blogspot.com/2013/03/16800-clean-and-11960-malicious-files.html.
6 https://virusshare.com/.

Digital Threats: Research and Practice, Vol. 1, No. 3, Article 15. Publication date: August 2020.
10 examples of CVE 2010-3970, 4 examples of CVE-2010-3333, and 2 examples of CVE-2011-1982. In addition to these 30 malicious samples, we collected 311 known-benign training documents and 30 known-benign test documents. These sizes were chosen to be commensurate with the above PDF experiment.

4.1 Feature Set Reduction on PDFs

As the feature merging threshold $\alpha$ 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 $\alpha$. It is interesting to note that at even relatively low $\alpha$ 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 Section 4.3, accuracy as high as 90% can even be attained using $\alpha$ up to 0.35, which provides a feature set size reduction of 85%.

4.2 A Comprehensive Whitelist of Exemplar PDF 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 $\alpha = 0.05$ (see Section 4.3), 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 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 dataset of benign PDFs is representative of the universe of benign PDFs to future work.

4.3 Classifier Accuracy

A contour map that shows how classification accuracy varies with \( \alpha \) and \( \beta \) 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. Section 4.2), and with minimal trade-off can even use a significantly smaller whitelist containing just 347 exemplar features.

### 4.4 Generalization of Results to Microsoft Word

To evaluate our approach’s generalizability, we designed a classification experiment like the one above but for Microsoft Word. We monitored Microsoft Word 2010 as it rendered benign training, benign test, and malicious test DOCX files, then we utilized the procedures in Section 2.3 to perform classification. The experiment was designed to match our PDF experimentation as closely as possible. Overall, we found the results to be on par with those reported for PDFs in Section 4.3. Notably, the process of re-configuring our dynamic analysis to support a new file format took only a few hours, and the training process on a single machine took less than 24 hours.

Figure 4 shows the percentage of original features derived from the benign training set that are retained after feature merging for various $\alpha$. As is the case with Adobe PDF, a small $\alpha$ (in this case 0.10) results in a feature set reduction of more than half for Microsoft Word documents. Consistent with the above results, an increase in $\alpha$ does not significantly impact the classification results. Accuracy exceeding 90% can again be attained using $\alpha$ up to 0.35, which in this case provides a feature set size reduction of over 95%.

Figure 5 illustrates the growth of the Microsoft Word 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 has linear growth proportional to the number of documents rendered. In contrast, feature merging with increasing $\alpha$ appears to reduce the effect of growth and eventually (for $\alpha \geq 0.25$) provides an upper bound on the size of the whitelist.
Fig. 4. Effect of varying $\alpha$ in terms of the percentage of original Microsoft Word model features retained after feature merging via entity resolution.

irrespective of the number of documents rendered. For instance, with $\alpha = 0.25$, the size of the whitelist tops out at 990 exemplar features, and with $\alpha = 0.35$ it tops out at 480 exemplar features.

The minimum convergence threshold for Microsoft Word of $\alpha = 0.25$ was significantly higher than the $\alpha = 0.05$ we observed for Adobe Reader. An early investigation of this led to an intuitive explanation based upon the average feature lengths of the two applications. The features are generally a composition of a registry key path or file path plus a random string. Adobe Reader produces features that are on average 83 characters long and uses short four-character-long randomized file names. In contrast, Microsoft Word tends to use deeper registry keys and file paths and longer random strings; the result being features that are on average 152 characters long with the random portions being of length 32. These numbers ($\frac{83}{152} \approx 0.53$ and $\frac{32}{152} \approx 0.197$) very closely match the $\alpha = 0.05$ and $\alpha = 0.25$ thresholds we derived empirically for the two applications. This early investigation served as the basis for the parameter selection work discussed in Section 5.2.

A contour map that shows how Word document classification accuracy varies with $\alpha$ and $\beta$ is presented in Figure 6. Classification accuracy is defined as in Section 4.3 above. The contour map shows that nearly perfect classification accuracy (>96%) is attained with $\alpha = 0.05$ and $\beta = 34$, and that good accuracy (>90%) is attained with $\alpha = 0.35$ and $\beta = 30$. This means that an analyst can perform nearly perfect classifications using a whitelist containing about 7K exemplar features, and with minimal trade-off can even use a significantly smaller whitelist containing just 980 exemplar features.

4.5 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. Section 8), this result is pleasing but far from revolutionary. However, the present work nonetheless stands out, because we have shown that the benign classes of DOCX and 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 (see NIST Special Publication 800-53 [13]) 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. Further, the need for only small amounts of training data and the simplicity of the approach mean that additional file formats can be supported with only a relatively small amount of additional effort.

5 MODEL PARAMETER SELECTION
In the preceding description of our Evaluation (see Section 4), we utilized hand-picked $\alpha$ and $\beta$ that provided the best accuracy as determined by Figures 3 and 6, respectively. However, to fully automate the construction of a suitable classifier for an arbitrary document viewer, a method is also needed to automatically select satisfactory values of $\alpha$ and $\beta$. In this section, we further motivate the need to carefully select $\alpha$ and $\beta$, and we present a method for their automatic selection that approximately recovers the values we selected by hand in the preceding evaluations.

5.1 Motivation
The selection of an $\alpha$-threshold has two effects on the system. First, a good $\alpha$-threshold can reduce the size of the model, since, with merging, we only need to keep a single entry for each set of similar observed viewer/OS interactions (rather than one entry for each individual interaction). Second, and equally important, the accuracy
of detections is directly impacted by the selection of $\alpha$: an $\alpha$-threshold that is too high can spuriously cause anomalous interactions to be merged into benign behaviors and thereby lead to false negatives, while an $\alpha$-threshold that is too low can lead to false positives, because the system will flag as anomalous any benign semantically equivalent interactions (e.g., logs written to randomly named log files) that only differ syntactically in a few positions.

The other model parameter, $\beta$, also has a direct impact on the accuracy of detections. To determine whether a suspect document is malicious or benign, we observe the interactions it induces between the document viewer and the OS. Each such interaction is compared to those in the model (using the $\alpha$-threshold comparison strategy discussed above), and any observed interaction that cannot merge with a term in the model (due to its apparent dissimilarity with all the model’s terms) is deemed anomalous. If the number of observed anomalous system interactions induced by rendering the suspect document within its viewer exceeds the model parameter $\beta$, then the classifier deems the suspect malicious; otherwise, it is deemed benign. Since $\beta$, therefore, controls how many anomalous interactions we need to observe before deeming a document malicious, it follows that a too low $\beta$ will lead to many false positives while a too high $\beta$ will lead to many false negatives. More subtly, since the number of anomalies depends on how lenient the term merging is, the appropriate choice of $\beta$ also depends on one’s prior choice for $\alpha$. A low value of $\alpha$ can lead to a higher number of anomalies, and hence necessitate a higher value for $\beta$ to maintain classification accuracy.

Given that the choices of both $\alpha$ and $\beta$ have such a significant impact on the ultimate detection accuracy of the classifier, it is crucial that optimal values be selected. For example, see Figures 3 and 6, which demonstrate the wide range of detection accuracy achievable by the classifier for Adobe PDFs and Microsoft Word documents, respectively. The red regions reflect high accuracy, where appropriate values of $\alpha$ and $\beta$ parameters were chosen. The blue regions reflect poor accuracy, occurring when the parameter pair selections are not appropriate for the document viewer. This illustrates how crucial it is to select a suitable pair of parameters, since even slight variations in them can lead either to detections approaching 100% accuracy or to terrible performance approaching 0% accuracy.

5.2 Automatic Model Parameter Selection

If an appropriate $\alpha$ parameter is selected, term merging (see Section 2.2) leads to concise models that combine features that varied only due to programmatic randomness. This means that any terms still deemed anomalous
after term merging are likely to be true anomalies rather than just random permutations of common benign system interactions. The optimal value for $\alpha$ can thus be thought of as the minimum value that allows semantically equivalent features to be merged together. Before we describe a mechanism by which to choose such an $\alpha$, let us first describe how programmatic randomness affects edit distance.

Assume there are two features $f$ and $v$, both of length $n$, that differ due to randomness. Assume that this randomness exists in substrings of length $\ell$ in both $f$ and $v$. Further, assume that both these substrings are generated by a random process that uniformly samples an alphabet $\mathcal{Z}$ of size $z$, and that the $\ell$-length substrings begin at the same offset $o$ in both $f$ and $v$. This formalism describes the programmatic randomness shown in the example of Section 2.2, and all of the other programmatic randomness we have encountered so far.

Given this formulation, for integers $i$ such that $0 \leq i < o + \ell$, and $\forall x \in \mathcal{Z}$, we see that $Pr(f_i = x|v_i = x) = \frac{1}{z}$. As such, if $\mathcal{X}$ is a random variable describing the number of matching characters in the random substrings of $f$ and $v$, then $\mathcal{X}$ has a Binomial distribution with $\ell$ trials and a success probability of $\frac{1}{z}$. Conversely, the number of characters expected to vary between $f$ and $v$ (i.e., the number of character substitutions necessary to transform $f$ to $v$ or the Levenshtein distance between $f$ and $v$) is the random variable $L = \ell - \mathcal{X}$.

Given the definition of $\alpha$-equivalence (see Section 2.2), we know that $f$ and $v$ will be deemed $\alpha$-equivalent if and only if $\frac{\ell}{n} < \alpha$. Thus, in expectation, we expect $f$ and $v$ to be $\hat{\alpha}$-equivalent for any $\hat{\alpha}$ such that $E(L) = \hat{\alpha} \ell$. Furthermore, it is clear that $E(L) = \hat{\ell}$ is maximized by setting $\hat{\alpha} = \frac{\hat{\ell}}{n}$. Using the linearity of expectation to rewrite this inequality, $E(L) = E(\ell - E(\mathcal{X}))$, which can be simplified as $\ell - \frac{\ell}{z}$ or $\ell \times \frac{z - 1}{z}$. So, we expect $f$ and $v$ to be $\hat{\alpha}$-equivalent for all $\hat{\alpha} > \frac{\ell}{n} \times \frac{z - 1}{z}$.

If one knows that programmatic randomness occurs in this manner, and knows a priori the appropriate values of $\ell$, $n$, and $z$, then the above derivation can be used to compute a feasible $\alpha$ analytically.

Alternatively, we describe below an algorithm by which an approximately optimal value of $\alpha$ can be recovered. Our approach is to first select a set $\mathcal{S}$ of candidate $\alpha$ values and to simultaneously partition the training data into several groups. Next, a whitelist $\mathcal{W}_\alpha$ is iteratively trained for each candidate $\alpha \in \mathcal{S}$; that is, we perform multiple iterations wherein we train/re-train each $\mathcal{W}_\alpha$ by incorporating an additional group of training data in each round. After each round, the size of each whitelist, $|\mathcal{W}_\alpha|$, is evaluated. The iterations terminate when either every $\mathcal{W}_\alpha$ has converged or the training data has been exhausted. After termination, we return the set of $\alpha$ values from $\mathcal{S}$ for which $\mathcal{W}_\alpha$ was observed to converge.

We have used this $\alpha$ candidate selection algorithm to evaluate experimental data and have found it to be extremely effective. Take as a concrete example the trace data we collected from Microsoft Word while processing documents (see Section 4). By manually inspecting these data, we discovered that Microsoft Word frequently incorporates random Unique Universal Identifiers (UUIDs) into the registry keys to which it writes, with each UUID consisting of 32 random hexadecimal characters. Using the above model, this corresponds to $\ell = 32$ and $z = 16$. Additionally, the average feature length of observed Microsoft Word trace items gives $n = 152$. Using the analytical result above, we would expect for $\hat{\alpha} > 0.197$ (where $0.197 = \frac{32}{152} \times \frac{16 - 1}{16}$), features that differ only due to programmatic randomness would be $\hat{\alpha}$-equivalent. As such, we would expect models with $\alpha \geq 0.197$ to converge and those with $\alpha < 0.197$ to continue growing as additional documents are added. Looking at Figure 5, we see that this result plays out empirically with $\alpha = 0.25$ being the first value on the chart for which the model size converges.\footnote{In this experiment, the value $\alpha = 0.20$ was not tested as candidate $\alpha$, and hence it is not displayed in this chart. However, we have verified that the model converges with $\alpha = 0.20$.}

Given a set of feasible values for $\alpha$, the next step is to compute corresponding optimal values of $\beta$ for each candidate $\alpha$. To do so, a collection of benign documents is first partitioned into three sets: two training sets and a validation set. Then, for each candidate $\alpha$, a corresponding whitelist $\mathcal{W}_\alpha$ is learned from the first training set. Next, we run every document $d$ in the second training set against each $\mathcal{W}_\alpha$ and observe the number of anomalies $k_d$ resulting from the rendering of each document. A satisfactory value of $\beta$ for some $\alpha \in \mathcal{S}$, dubbed $\beta_\alpha$, is chosen.
to be the maximum \( k_d \). This method minimizes the false-positive rate of the classifier over the training data. More specifically, any threshold less than \( \beta_\alpha \) would result in a higher false-positive rate and any value greater than \( \beta_\alpha \) would have an equivalent false-positive rate, but also potentially admit more false negatives.

At this point, our parameter selection algorithm has constructed a set of classifiers with associated tuples \((\alpha, \beta_\alpha, W_\alpha)\). The last step of the parameter-learning algorithm is to choose the optimal classifier from that set of feasible classifiers. To do so, we use the validation set described above to examine the false-positive rate of each classifier and choose the one with the lowest false-positive rate. If a tie exists, preference is given to the classifier with the lowest associated \( \alpha \).

This parameter selection approach was evaluated empirically using our Microsoft Word and Adobe PDF document datasets (see Section 4). For both file formats, we observed that these automatic parameter selection techniques recovered parameter choices that provided accuracy within epsilon of those that we had previously selected by hand. In other words, with these algorithmic techniques for optimally selecting \( \alpha \) and \( \beta \) in place, one may now quickly and automatically construct highly accurate malicious digital document detection models for arbitrary document viewers using just small amounts of benign training data.

6 IMPLEMENTATION

Implementing the approach described above required the development of three software components: a dynamic analyzer, a model learning tool, and a classifier. The dynamic analyzer performed system call monitoring by hooking the x86 SYSENTER/SYSEXIT instructions and decoding the system call arguments. Given a digital document, this component produced the comma-separated trace output described in Section 2.1. It was implemented from scratch as a plugin for the Selective Symbolic Execution platform [4].

The model learning tool implemented the methods described in Section 5.2. As such, this tool consumed a set of benign traces and produced a tuple \((\alpha, \beta_\alpha, W_\alpha)\), corresponding to the term-merging parameter, classification threshold, and whitelist, respectively. The core algorithms were implemented in Python from scratch, but data structures from the sortedcontainers\(^{10}\) and numpy\(^{11}\) libraries were used for convenience. An implementation of Levenshtein edit distance was also borrowed from the edlib\(^{12}\) package.

The final component, the classifier, consumes the comma-separated trace output for a suspect document and a tuple \((\alpha, \beta_\alpha, W_\alpha)\), and produces a conviction decision (i.e., malicious or benign). This component implemented the approach described in Section 2.3. It was written in Python and shared many common data structures with the model learning tool. As such, the same libraries (i.e., edlib, sortedcontainers, and numpy) were used in both tools.

7 DISCUSSION OF LIMITATIONS AND THREATS

In this section, we discuss some limitations and threats to the experimental validity of our evaluation. Where possible, we also seek to demonstrate how these concerns were addressed through the design of our approach or the structure of our quantitative evaluation.

7.1 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

\(^{10}\)https://github.com/grantjenks/python-sortedcontainers.

\(^{11}\)http://numpy.org.

\(^{12}\)https://github.com/Martinsos/edlib.

Digital Threats: Research and Practice, Vol. 1, No. 3, Article 15. Publication date: August 2020.
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. To overcome this limitation, one would either need to employ the use of software automation technologies like AutoIt\textsuperscript{13} 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 such as 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 analysis 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 Microsoft Word 2010, and therefore our malicious test set only included PDF and Word documents that we confirmed to work against those particular document viewer versions. However, due to the brittle nature of exploits, our evaluation system would likely fail to detect malicious documents targeting other versions of those document viewers. If our approach were to be adopted to detect malicious digital documents en masse, then separate models of expected functionality would need to be generated for each version of each document viewer.

Another potential limitation of our approach is that an attacker with knowledge of the detector’s threshold configuration (i.e., $\alpha$ and $\beta$) could craft their malicious document in such a way that it performs fewer than $\beta$ 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 $\beta$ tend to be very small (see 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.

7.2 Experimental Validity

Given the small size of our testing data, the fact that the document malware contained within it is quite old, and that the document viewers we tested against are equally as old, it is reasonable to be concerned about the external validity of our results; i.e., do our results generalize out to the broader universe of document malware that is currently seen in the wild? This section explains why we believe that our results do indeed generalize. There are two real concerns to be addressed here: (1) Does the approach generalize to newer and/or more sophisticated document malware that targets the same viewers we evaluated against and (2) Does it generalize even more broadly to document malware targeting entirely different document viewers. These two concerns are addressed in turn below.

We tested our classifier on a corpus of relatively old document malware that collectively utilized a small handful of different Adobe Reader and Microsoft Word exploits (see Section 4) to trigger execution of the various malware payloads contained therein—and the choice to do so was made for entirely practical reasons. First, we again note that our approach is based on dynamic analysis and hence can only detect a malicious digital document if the malware payload embedded within that document actually executes. In practice, the way a malicious document’s malware payload gains execution is by way of an exploit that leverages a vulnerability like a stack-based buffer overflow in the associated document viewer to effect arbitrary code execution.

This reality unfortunately leads to an experiment design issue for our approach, since it implies that any individual malicious document will only ever have its payload executed if the version of the document viewer opening the document is vulnerable to the exploit contained within the document; i.e., for us to detect some individual malicious document that leverages a particular exploit to execute a malware payload, then we must trace the opening/rendering of that document under a version of the associated viewer that is vulnerable to that exploit. Likewise, the model that those trace features are to be compared against must also have been generated with respect to that same version of the viewer. This same tightly coupled relationship between document

\textsuperscript{13}https://www.autoitscript.com/.
malware instances and specific versions of the associated viewer exists for all types of document malware (i.e., not just PDF and Microsoft Word).

Therefore, to show in our evaluation that our approach could indeed detect known instances of malicious documents, we needed to test against a version of each viewer that is known to be susceptible to the malware in our corpus or, conversely, we needed to include malware in our corpus that targets our selected viewer version. However, since document viewer authors patch vulnerabilities as soon as they become public, it is the case that there are but few unique malware samples to be found in the wild targeting any individual version of a viewer. Thus, to conduct the evaluation presented in Section 4, we selected one particular version of each viewer (Adobe Reader and Microsoft Word) for which we knew a significant number of malicious documents targeting those versions to exist, and then we tested the approach against those malware samples.

The classification results obtained from that experiment were good, so what remains then is to convince the reader that the success of the approach was not tied to those particular old malicious documents and/or those old document viewer versions. The argument for this is logical. Since our models comprehensively characterize all the actions one version of Adobe Reader and one version of Microsoft Word might, respectively, undertake while rendering a benign document, and, since malicious documents are defined by the fact that they induce the viewer application into performing unexpected actions, then it follows that our classifier will catch the unexpected actions arising from any successfully executed payload in those versions and that these anomalous actions will contribute to the malicious document being correctly classified. This implies that our approach will successfully detect any other malicious documents targeting these same viewer versions as well. Likewise, since malware payloads must still make use of system calls to achieve their malicious goals and, since those system calls will invariably differ observably from those of the document viewer, then it is clear that the approach will also extend to new malware targeting newer versions of those viewers so long as each model is generated with respect to the viewer version in question and so long as the opening/rendering of the malicious document is traced under that viewer.

Another potential threat to external validity, already alluded to above, is that the approach described herein might not generalize to other document format/viewer pairs beyond PDFs in Adobe Reader and DOCX documents in Microsoft Word.

While this is a valid concern, we believe it to be rather immaterial in practice due to the particular category of files and applications we have targeted in this work. In particular, we have focused on digital documents and their corresponding viewing applications precisely because document viewers possess rather circumscribed functionality (i.e., reading files, parsing them, and rendering the content to a screen). As such, any observed actions beyond the aforementioned are likely anomalous and possibly malicious. One can also see this play out in the data. Table 7 shows aggregate statistics for the number of anomalous features (previously referred to as $k$) observed in the traces of the documents, both benign and malicious, that comprised our testing dataset in the PDF experiment. From this, we see that opening a benign document induces on average just five document viewer/OS interactions that do not appear similar to anything that was previously observed in the training data. Conversely, the malicious documents induced two orders of magnitude more anomalous document viewer/OS interactions.

Table 7. Number of Anomalous Features ($k$) Using $\alpha = 0.05$ for Malicious PDFs in Evaluation Dataset

|          | Benign | Malicious |
|----------|--------|-----------|
| Mean     | 4.9    | 243       |
| Median   | 5      | 192       |
| Min      | 3      | 1         |
| Max      | 5      | 929       |

$n_{Benign} = 33$ and $n_{Malicious} = 33$.  

Digital Threats: Research and Practice, Vol. 1, No. 3, Article 15. Publication date: August 2020.
Though our study focused on digital document viewers, the approach is theoretically applicable to other categories of application as well. Specifically, the approach should generalize to those applications that (1) perform only a fixed set of operations when parsing input data (i.e., applications that are not general purpose interpreters) and (2) do not make use of operating system interactions that resemble the way that malware generally interacts with the OS. In such cases, it is reasonable to expect the approach to perform well. Since both those caveats are expected to hold for other versions of our particular document viewers (Adobe Reader and Microsoft Word) as well as for other document viewers generally, we therefore conclude the two threats to external validity mentioned above are minimal.

One last threat to external validity is whether our approach can detect the most sophisticated and modern malicious documents. Many “traditional-looking” malicious documents still mirror the paradigm seen in Table 6, where a dropper payload places yet another payload on the disk and installs a persistence mechanism. However, more sophisticated approaches exist; e.g., some malicious documents may execute fileless payloads that only persist in memory, and other malicious documents may install rootkits. Though such malicious documents may be more sophisticated, so long as their payloads must still perform system calls to achieve their objectives (as is the case for all malware to date), they are detectable by our approach.

8 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, 7] 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 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. [21].

The most similar prior work to our own is Engleberth et al. [7]. They, too, monitored dynamic behaviors of document viewers to construct a whitelist, 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.

Supervised classification approaches that turn on metadata and structural features [20, 22, 24, 25] have often been successful at binary classification of malicious documents. However, such approaches can be evaded if the
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 \[^{[26]}\] that many attacks delivered via malicious PDF documents are conducted by embedding malicious JavaScript within said documents. As such, approaches \[^{[15, 26]}\] that extract, tokenize, and featureize 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. \[^{[11]}\] 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 that 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 \[^{[12]}\], 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) \[^{[5]}\] 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.

9 CONCLUSION

We developed a classifier for the disposition of digital documents that requires training on only a very small dataset 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 and 95% accuracy in classifying Microsoft Word documents. 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.
REFERENCES

[1] Ross Anderson, Chris Barton, Rainer Böhme, Richard Clayton, Michel J. G. 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 Proceedings of the 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 Proceedings of the Asia-Pacific Network Operations and Management Symposium (APNOMS’13). 1–3.

[4] Vitaly Chipounov, Volodymyr Kuznetsov, and George Candea. 2012. The SzE platform: Design, implementation, and applications. ACM Trans. Comput. Syst. 30, 1 (2012), 2.

[5] Rudi Cilibrasi and Paul M. B. Vitányi. 2005. Clustering by compression. IEEE Trans. Inf. Theor. 51, 4 (2005), 1523–1545.

[6] Artem Dinaburg, Paul Royal, Monirul Sharif, and Wenke Lee. 2008: Malware analysis via hardware virtualization extensions. In Proceedings of the 15th ACM Conference on Computer and Communications Security. ACM, 51–62.

[7] M. Engelberth, Carsten Willems, and Thorsten Holz. 2009. Detecting malicious documents with combined static and dynamic analysis (PowerPoint presentation). Virus Bull. (2009). https://www.virusbulletin.com/uploads/pdf/conference_slides/2009/Willems-VB2009.pdf

[8] Tal Garfinkel, Mendel Rosenblum et al. 2003. A virtual machine introspection based architecture for intrusion detection. In Proceedings of the Network and Distributed System Security Symposium (NDSS’03), Vol. 3. 191–206.

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

[10] Galen Hunt and Doug Brubacher. 1999. Detours: Binary interception of Win 32 functions. In Proceedings of the 3rd USENIX Windows NT Symposium.

[11] Rafiqul Islam, Ronghua Tian, Lynn Batten, and Steve Versteeg. 2010. Classification of malware based on string and function feature selection. In Proceedings of the 2nd Cybercrime and Trustworthy Computing Workshop (CTC’10). IEEE, 9–17.

[12] 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.

[13] 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).

[14] Suleyman Kondakci. 2009. A concise cost analysis of Internet malware. Comput. Sec. 28, 7 (2009), 648–659.

[15] 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.

[16] Vladimir I. Levenshtein. 1966. Binary codes capable of correcting deletions, insertions and reversals. In Soviet Phys. Dokl., Vol. 10. 707.

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

[18] 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.

[19] Davide Maiorca, Battista Biggio, and Giorgio Giacinto. 2019. Towards adversarial malware detection: Lessons learned from PDF-based attacks. ACM Comput. Surv. 52, 4 (2019), 78.

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

[21] 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. Comput. Sec. 48 (2015), 246–266.

[22] 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).

[23] Daniel Scofield, Craig Miles, and Stephen Kuhn. 2017. Fast model learning for the detection of malicious digital documents. In Proceedings of the 7th Software Security, Protection, and Reverse Engineering/Software Security and Protection Workshop. ACM, 3.

[24] 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.

[25] Nedim Šrndić 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.

[26] Cristina Vatamaniuc, Dragos Gavrilut, and Răzvan Benchea. 2012. A practical approach on clustering malicious PDF documents. J. Comput. Virol. 8, 4 (2012), 151–163.

Received September 2019; revised January 2020; accepted January 2020

Digital Threats: Research and Practice, Vol. 1, No. 3, Article 15. Publication date: August 2020.