A Structural and Content-based Approach for a Precise and Robust Detection of Malicious PDF Files

Davide Maiorca, Davide Ariu, Igino Corona and Giorgio Giacinto
Department of Electrical and Electronic Engineering, University of Cagliari, Cagliari, Italy
{davide.maiorca, davide.ariu, igino.corona, giacinto}@diee.unica.it

Keywords: PDF, Evasion, Adversarial Machine Learning, Malware, Javascript.

Abstract: During the past years, malicious PDF files have become a serious threat for the security of modern computer systems. They are characterized by a complex structure and their variety is considerably high. Several solutions have been academically developed to mitigate such attacks. However, they leveraged on information that were extracted from either only the structure or the content of the PDF file. This creates problems when trying to detect non-Javascript or targeted attacks. In this paper, we present a novel machine learning system for the automatic detection of malicious PDF documents. It extracts information from both the structure and the content of the PDF file, and it features an advanced parsing mechanism. In this way, it is possible to detect a wide variety of attacks, including non-Javascript and parsing-based ones. Moreover, with a careful choice of the learning algorithm, our approach provides a significantly higher accuracy compared to other static analysis techniques, especially in the presence of adversarial malware manipulation.

1 INTRODUCTION

Malicious PDF files have become a well known threat during the past years, and they still constitute a very effective weapon for cybercriminals, as new exploits have recently been released. This is due to the fact that, in spite of the efforts of software vendors such as Adobe, PDF software is often vulnerable to zero-day attacks. In particular, such attacks exploit the integration of the PDF file format with third-party technologies (e.g., Javascript or Flash), thus making the creation of ad-hoc patches a challenging task. Moreover, due to the architectural complexity of PDF files and given the wide variety of code obfuscation techniques employed by miscreants, it is hard for antivirus vendors to provide protection against novel or even known attacks (Symantec, 2014).

Attackers mainly adopt Javascript code to exploit PDF vulnerabilities, by leveraging on well-known techniques, such as Return Oriented Programming and Heap Spraying (Buchanan et al., 2008; Ratanaworabhan et al., 2009). However, there have been cases in which ActionScript code has been employed. For instance, CVE 2010-3654 exploits a vulnerability in Adobe Flash Player using a “Just in Time Spraying” approach (Bania, 2010). In order to evade detection, PDF attacks may also resort to advanced encryption methods for hiding malicious code or malicious embedded files (Adobe, 2008).

Signature-based approaches adopted by current anti-malware systems, based on heuristics or string matching, are not able to cope with novel attacks, and have proved to be weak against polymorphic ones (Esparza, 2011). Aiming at addressing such weaknesses, recent research works mainly focused on two different directions: first, they focused on detecting malicious Javascript code within PDF files, through both static and dynamic (behavioral) analysis (Cova et al., 2010; Laskov and Šrndić, 2011; Tzermias et al., 2011). Then, they leveraged on the external structure of the PDF file (i.e., PDF objects) in order to detect malicious PDF files regardless of the attack or exploit they carried (Maiorca et al., 2012; Smutz and Stavrou, 2012; Šrndić and Laskov, 2013). The latter approaches have proved to be generally more effective than the first ones, since they also allowed for the detection of non-Javascript attacks. However, further research showed that they are extremely vulnerable against deliberate attacks (Maiorca et al., 2013; Šrndic and Laskov, 2014). For this reason, research has focused again on the detection of malicious Javascript code (Corona et al., 2014; Liu et al., 2014) and on hardening security through the adoption of a *sandbox* (Maass et al., 2014).

In this paper, we present a novel machine learning-based approach to the detection of malicious

PDF files that leverages on information extracted both from the *structure* and the *content* of the PDF file. On one hand, we represent the information about the *structure* by analyzing: **a)** general properties of the PDF file structure and **b)** structural properties of the PDF objects in terms of *keywords*. On the other hand, we analyze content-based information such as: **a)** malformed objects, streams and codes, **b)** known vulnerabilities in Javascript code and **c)** embedded contents such as other PDF files. To perform these operations, and to address parsing-related vulnerabilities presented in previous works, we leverage on two well-known tools for PDF analysis, namely, `PeePDF`[1] and `Origami`[2]. Such approach allows for an extremely accurate detection of PDF malware deployed in the wild (including non-Javascript attacks and recent vulnerabilities) with very low false positives. At the same time, with a careful choice of the learning algorithm (whose role will be discussed in detail), it provides a significant improvement on detecting targeted attacks in comparison to the other state of the art approaches.

Contributions

The contributions made by this work can be summarized into four points:

- We develop a novel, machine learning based approach to the detection of malicious PDF files that leverages on information extracted from both the *structure* and the *content* of a PDF file;
- We experimentally evaluate the performances of our system on a dataset that contains a wide number of PDF-related vulnerabilities. We compare such performances to the ones of the most important, publicly available tools;
- We evaluate the robustness of our system against automatic attacks and evasion attempts that have proved to be extremely effective against public available tools;
- We discuss the limits of our system and the role of the *learning algorithm* in assessing its robustness. In relation to that, we provide research guidelines for future work.

Paper Structure

This paper is divided into six Sections beyond this one. Section 2 provides an insight into the structure of the PDF files. Section 3 presents related works on malicious PDF detection. Section 4 describes our methodology to the detection of malicious PDFs. Section 5 provides the experimental results. Section 6 discusses the limits of our approach and provides guidelines for future research work. Section 7 provides conclusive remarks.

2 PDF FILE FORMAT

A PDF file is a hierarchy of objects logically connected to each other. For the sake of the following discussion, we will model the PDF file structure as composed by four parts (Adobe, 2006):

- header: a line which gives information on the PDF version used by the file.
- body: it is the main portion of the file, and contains all the PDF objects.
- cross-reference table: it indicates the position of every *indirect*[3] object in memory.
- trailer: it gives relevant information about the root object and number of revisions made to the document.

Typically, a PDF file presents, in its body, a sequence of *indirect objects*, described by the expression `ObjectNumber 0 obj`. Each object consists of a *dictionary* which defines, through a sequence of coupled *keywords* (also called *name objects*) represented by a /, the characteristics of the data inside the object itself or in one of its references (e.g., in case of an attack, the *presence* of malicious code through the presence of the keyword `/Javascript`). Optionally, an object might also include a *stream*, which contains the object *data* that will be parsed by the reader and visualized by the user (e.g., in case of an attack, the code whose presence is signaled by the dictionary). For more information on the PDF structure, please check the PDF Reference (Adobe, 2006).

3 RELATED WORK

First approaches to malicious PDF detection proposed static analysis on the raw (byte-level) document, by means of *n-gram* analysis (Li et al., 2007; Shafiq et al., 2008) and *decision trees* (Tabish et al., 2009). However, these approaches were not really tailored to PDF files, as they mainly targeted different file formats, such as DOC, EXE, etc. Whereas such a raw analysis may detect many malware "implementations" besides malicious PDFs, it may be quite easy

[1] http://eternal-todo.com/tools/peepdf-pdf-analysis-tool

[2] http://esec-lab.sogeti.com/pages/origami

[3] In the PDF language, *indirect* means that the object can be referenced by other objects.

to evade it either by using modern obfuscation techniques, such as AES encryption (Adobe, 2008), or by resorting to different techniques to exploit vulnerabilities, such as Return Oriented Programming, Heap Spraying or JIT Spraying (Buchanan et al., 2008; Ratanaworabhan et al., 2009; Bania, 2010).

To address the increasing complexity of PDF malware, subsequent works focused on the analysis of embedded Javascript code. A number of solutions for the detection of malicious Javascript code have been proposed in the context of web security. For instance, `Jsand` (Cova et al., 2010), `Cujo` (Rieck et al., 2010), `Zozzle` (Curtsinger et al., 2011), `Prophiler` (Canali et al., 2011) are well-known tools for the dynamic and static analysis of Javascript code. These tools are often employed by systems designed to identify threats embedded in different document formats.

`Wepawet`[4], a framework for the analysis of web-based threats, relies on `JSand` to analyze Javascript code within PDF files. `Jsand` (Cova et al., 2010) adopts `HtmlUnit`[5], a Java-based browser simulator, and Mozilla's `Rhino`[6] to extract behavioral features related to the execution of Javascript code. A statistical classifier is trained on a representative number of samples containing benign code, and malicious code is spotted by detecting anomalous patterns (i.e., codes that are significantly different to the ones the system has been trained with).

A similar approach is adopted by `MalOffice` (Engleberth et al., 2009). `MalOffice` uses `pdftk`[7] to extract Javascript code, and `CWSandbox` (Willems et al., 2007) to analyze the code behavior: Classification is carried out by a set of rules (`CWSandbox` has also been used to classify general malware behavior (Rieck et al., 2008)). `MDScan` (Tzermias et al., 2011) follows a different approach as malicious behavior is detected through `Nemu`, a tool able to intercept memory-injected shellcode. A very similar idea, but with a different implementation, has been developed in `ShellOS` (Snow et al., 2011).

Dynamic detection by executing Javascript code in a virtual environment may be time consuming and computationally expensive, and it is prone to evasion by a clever attacker that leverages on different implementations of the JavaScript engine used by the PDF reader and by the code analyzer (Tzermias et al., 2011). To reduce computational costs, `PJScan` (Laskov and Šrndić, 2011) proposed a fully static lexical analysis of Javascript code by training a statistical classifier on malicious files.

In 2012 and 2013, malicious PDF detectors that base their detection on the *structure* of the PDF file, without analyzing the embedded code, have been developed. To distinguish them from the previous approaches, we call them *structural systems* (Maiorca et al., 2012; Smutz and Stavrou, 2012; Šrndić and Laskov, 2013). Among these, `PDFRate`[8] is the only one *publicly available*. It is based on 202 features extracted from both document metadata and structure and it uses random forests as classification algorithm. All these systems have been motivated by the need to detect PDF malware that also featured different contents than Javascript, such as Flash. The analysis of the PDF structure allowed for a wider detection rate in comparison to previous systems, without having the need of looking for embedded code or trying to de-obfuscate it. However, recent works (Maiorca et al., 2013; Šrndic and Laskov, 2014) showed that such systems suffer from parsing problems and might be easily attackable.

Because of the evident weaknesses of structural systems, research focused again on detecting malicious Javascript code. New approaches either resorted to discriminant API analysis (Corona et al., 2014) or code instrumentation (Liu et al., 2014). An approach to harden security when opening a PDF file by leveraging on *sandboxing* has also been proposed (Maass et al., 2014).

4 PROPOSED DETECTION APPROACH

As stated in Section 3, the vast majority of recent works on malicious PDF detection focused on the analysis of either the Javascript code or the PDF file structure (*structural systems*). Such information is usually processed by a *machine learning* system, i.e., it is converted into a *vector* of numbers (*features*) and sent to a mathematical function (*classifier* or *learner*), whose parameters have been tuned through a process called *training*. Such training is performed by using samples whose classes (benign or malicious) were already known.

However, systems developed until now suffer from several weaknesses: on one hand, systems focused on Javascript cannot address other vulnerabilities, such as the ones based on Flash (Maiorca et al., 2012). On the other hand, structural systems resort to information that can be easily manipulated by an attacker in order to deceive the classifier (Smutz and Stavrou, 2012; Maiorca et al., 2013).

[4] http://wepawet.iseclab.org/index.php

[5] http://htmlunit.sourceforge.net

[6] http://www.mozilla.org/rhino

[7] http://www.pdflabs.com/tools/pdftk-the-pdf-toolkit

[8] http://pdfrate.com/

To overcome these weaknesses, we propose a new machine learning-based approach, which extracts information both from the *structure* of the PDF file and from its *content*. This method is purely *static*, as it does not involve any dynamic simulation performed by a PDF rendering engine.

Figure 1 shows the high-level architecture of our system. To extract information, we created a *parser* that leverages on `PeePDF` and `Origami`.

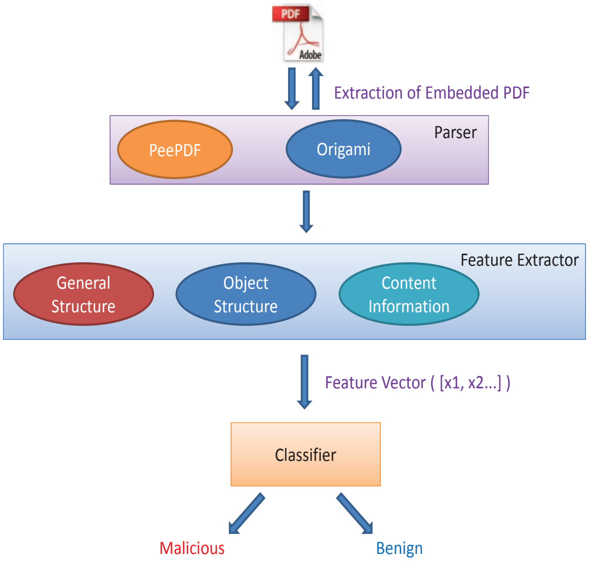

Figure 1: High-level architecture of our system.

These tools perform an in-depth analysis of PDF files to highlight known exploits, suspicious objects, or functions that are known to be potentially malicious (for example, see vulnerability `CVE-2008-2992`). Moreover, they will extract and analyze as a separate sample any *embedded* PDF file. We adopted these two tools as they provide a reliable parsing process compared to other ones, such as `PdfID`, which naively analyzes the PDF file ignoring its structural properties, thus allowing attackers to perform easy manipulations (Maiorca et al., 2013).

Each PDF document will be represented by a vector composed by: **a)** `8` features related to the *general structure* of the file, in terms of number of objects, streams, etc.; **b)** A *variable* number of features (typically not more than `120`) related to the *structure of the PDF objects*, represented by the occurrence of the most *frequent keywords* in the training dataset (its number is variable as it depends on the training data); **c)** `7` features related to the *content* of the PDF objects. In particular, the objects are scanned for *known vulnerabilities*, *malformed objects*, etc.

The remaining of this Section is organized as follows. Section 4.1 provides a detailed description of all the features that we extract to discriminate between benign and malicious PDF files. Section 4.2 describes and motivates the adopted classification algorithm.

4.1 Features

General Structure

We extract `8` structural features, which contain information about: **i)** The *size* of the file; **ii)** The number of *versions* of the file; **iii)** The number of *indirect objects*; **iv)** The number of *streams*; **v)** The number of *compressed objects*; **vi)** The number of *object streams*[9]; **vii)** The number of *X-ref streams*[10]; **viii)** The number of *objects containing Javascript*.

While these features may be considered as weakly discriminant when singularly used, together they provide a good overview of the whole PDF structure. For instance, from our experience, the size of malicious PDFs (and their number of objects/streams) is often smaller than the one of legitimate PDFs. This is somewhat reasonable, as malicious PDFs may not contain text, and the smaller is the file size, the smaller is the time needed to infect new victims. Likewise, *object and X-ref streams* are usually employed to conceal malicious objects inside the file, and *compressed objects* can include embedded contents, such as scripting code or other EXE/PDF files.

Object Structure

We extract the *occurrence* of the most *characteristic* keywords defined in the PDF language. With the term *characteristic*, we refer to keywords that have appeared in our training dataset D with a frequency that is higher of a threshold t. Other works, such as (Šrndić and Laskov, 2013), extracted a similar threshold by arbitrarily deciding a reasonable value for it. Our aim, though, is obtaining t in a more systematic way, so that it better relates to the data in D. In order to do so, we:

1. Split D into D_m and D_l, where the first one only contains *malicious* files and the second one only *legitimate files*. Obviously, $D = D_m \cup D_l$;
2. For each of the two datasets, and for each keyword k_n of the PDF language, we define: $f_n = F(k_n)$, where f_n is *the number of samples* of each dataset in which k_n appears at least once;
3. For each dataset, we find the frequency threshold value t by means of a *k-means clustering* algo-

[9] Streams containing other objects.

[10] A new typology of cross-reference table introduced by recent PDF specification.

rithm (MacQueen, 1967) with *k=2* clusters, computed through an euclidean distance. In order to establish with more precision the sizes of the two clusters, the algorithm has been tested five times with different starting points[11]. In this way, keywords will be split into two groups basing on their f_n value. Thus, for each dataset, we get the set of keywords K defined as: $K = \{(k_n)|f_n > t\}$ Therefore, for D_m we will obtain a set K_m and for D_l a set K_l;

4. We will get the final set of characteristic keywords K_t by: $K_t = K_m \cup K_l$.

The number of keywords in K_t obviously depends on the training data used and on the result of the clustering. The reason why we considered characteristic keywords, along with their occurrence, is that their presence is often associated to specific actions performed by the file. For example */Font* is a characteristic keyword in benign files. This is because it is usually associated to the presence of a specific font in the file. If this keyword occurs a lot inside a sample, it means that the PDF renderer might display different fonts, which is an expected behavior in legitimate samples. Selecting the most characteristic keywords also helps to exclude other ones that do not respect the PDF language specifications. Including the occurrence of non-characteristic or extraneous keywords in the feature set might allow an attacker to easily manipulate the PDF features without altering its functionality, thus increasing the possibility of an evasion of the detection system.

Content-based Properties

We verify if a PDF file is accepted or rejected by either `PeePDF` or `Origami`. There are two features associated to this information, one for `PeePDF` and one for `Origami`. The two tools perform a *non-forced scan*[12]. If one of these tools rejects the files, it means that their might be suspicious elements such as the execution of code, malformed or incorrect x-ref tables, corrupted headers, etc. It is worth noting that such elements might as well be present in legitimate samples. Therefore, `PeePDF` and `Origami` cannot be used as *detectors* of maliciousness, as they would generate too many *false positives*.

There are also `5` features that represent information about *malformed* **a)** objects (for example, when scripting codes are entirely injected inside a PDF dictionary), **b)** streams, **c)** actions (using keywords not proper of the PDF language), **d)** code (for example, using functions related to known vulnerabilities) and **e)** compression filters (the way in which data like images or code are compressed in the file to reduce the file size). This is done as malicious PDFs often include objects that do not strictly follow the PDF language specifications. However, such objects are nevertheless parsed by the reader, which is quite flexible from this respect.

4.2 Classification

We adopted a *supervised* learning approach, i.e., both benign and malicious documents are used for training, and we resorted to decision trees classifiers (Quinlan, 1996). Decision tree are capable of natively handling the heterogeneous nature of features described in Section 4.1, and they have exhibited excellent classification performances in previous works (Maiorca et al., 2012; Smutz and Stavrou, 2012; Corona et al., 2014). In particular, we decided to resort to the *Adaptive Boosting* (`AdaBoost`) algorithm (Freund and Schapire, 1995). Such algorithm linearly combines a set of *weak* learners, each of them with a specific weight, to produce a more accurate classifier. As *weak learner*, we define a low-complexity classification algorithm that usually yields results that are better than random guessing. The weights of each weak learner depend, obviously, on the ones of the training instances with which each learner is trained. Weak learners can be, for example, decision stumps (i.e., decision trees with only one leaf) or simple decision trees (`J48`). Choosing an ensemble of trees also allows for more robustness against *evasion attacks* compared to a single tree, as an attacker has to gather knowledge of multiple tree-models in order to perform an optimal attack (i.e., getting to know which features are most discriminant for *each* tree of the ensemble).

5 EXPERIMENTAL EVALUATION

In this Section, we first describe the dataset employed in our experiments, as well as the training and test methodology for the performance evaluation. Then, we provide *two* experiments. The first one compares the general performances of our approach, in terms of detection rate and false positives, with the ones of the other state of the art research. In particular, we refer to `PJScan`, `Wepawet`, and `PDFRate`, as they can be considered the three most important and *publicly*

[11] The seed value has been set to the default value indicated here: http://weka. sourceforge.net/doc.dev/weka/clusterers/SimpleKMeans.html.

[12] A scan that is stopped if it finds anomalies in the files. This definition is valid for `PeePDF`; in `Origami`, such scan is defined as *standard mode*.

available research tools for malicious PDF files detection. The second experiment focuses on testing our approach against *reverse mimicry attacks* (Maiorca et al., 2013), which have proved to be tremendously effective structural systems. In order to concretely validate our approach, we produce a high number of *real, working* attack samples, and we test them against our approach. Finally, we also provide a performance comparison with the other state of the art tools.

Dataset

We conducted our experiments using real and up-to-date samples of both benign and malicious PDFs in-the-wild. Overall, we collected `11,138` unique malicious samples from `Contagio` [13], a well-known reputable repository which provides information about latest PDF attacks and vulnerabilities. Moreover, we *randomly* collected `9,890` samples of benign PDFs, by means of the public `Yahoo search engine API` (http://search.yahoo.com). We kept a balance between malicious and benign files for the purpose of supervised training.

For the second experiment, we created 500 attack samples variants for each of the three attacks described in our previous work (Maiorca et al., 2013) *Javascript Injection*, *EXE Embedding* and *PDF Embedding*. Hence, in total we generated 1500 real attack samples.

Training and Test Methodology

For the *first experiment*, in order to thoroughly evaluate the generalization capability of our tool, we randomly split our data into two disjunct datasets:

- A training set composed by `11,944` files, subdivided into `5,993` malicious and `5,951` benign files. This set is used to train the classifier.
- A test set composed by `9,084` files, subdivided into `5,145` malicious and `3,939` benign files. This set is used to evaluate the accuracy of the classifier.

This process is repeated *three* times: we compute the mean and the standard deviation of the True Positives (TP) and False Positives (FP), over such three replicas. As a unique measure of the classification quality we also employ the so-called *Matthews Correlation Coefficient* (MCC) (Baldi et al., 2000), defined as:

$$MCC = \frac{TP \cdot TN - FP \cdot FN}{\sqrt{(TP+FP)(TP+FN)(TN+FP)(TN+FN)}}$$

where TN and FN refer to the number of true and false negatives, respectively.

In our experiments, we trained an `AdaBoost` ensemble of J48 trees, whose parameters have been optimized with a 10-fold cross validation (Freund and Schapire, 1995). We chose this kind of classifier ensemble as it showed the best accuracy compared to single classifiers (we experimented with random forest and SVM) or other ensemble techniques on our dataset. For the *second experiment*, we adopted the *same* training sets of the first experiment to train the system but, as a test set, we adopted the `1500` attack samples described before.

5.1 Experiment 1: General Performances

This Section provides a comparison between our approach and three recent research tools for the detection of malicious PDFs: `Wepawet`, `PJScan` and `PDFRate` (see Section 3).

Since `PJScan` adopts a *One Class SVM*, no benign samples have been used for training the system. We trained `PJScan` with the same malicious samples used to train our system. `PDFRate` is trained with a balanced dataset of 5000 benign and 5000 malicious samples, the latter collected from `Contagio`. It is worth noting that `PDFRate` is available with three different instances of the same classifier (*random-forests*), that only differ for the dataset on which they have been trained. For the sake of providing a fair comparison with our system, we considered only the one trained on the `Contagio` dataset, as `Contagio` is the same source from which we collected malware samples. The training size of `Wepawet` is unfortunately unknown[14]. We want to stress that we tried to make the fairest comparison possible, given the fact that it was not possible to train `PDFRate` and `Wepawet`, being them online services, and that the training set of the latter is unknown. Even though a perfect comparison would require the same exact training set for all the systems, we believe that, in this situation, this is the best possible compromise to provide useful information about their performances.

Table 1 shows the results of a direct comparison between our approach and the other tools. For each system, we show the average percentage of true positives (TP), false positives (FP), the related standard deviation within parentheses, and the MCC coefficient computed on mean values for TP and FP. It must be observed that `Wepawet` was not able to analyze all

[13] http://contagiodump.blogspot.it

[14] Being `Wepawet` and `PDFRate` online services, we could not train such systems with our own samples.

Table 1: Experimental comparison between our approach and other academic tools.

System	TP(%)	FP(%)	MCC
Ours	**99.805** (±.089)	**.068** (±.054)	**.997**
PDFRate	99.380 (±.085)	.071 (±.056)	.992
Wepawet	88.921 (±.331)	.032 (±.012)	.881
PJScan	80.165 (±1.979)	.013 (±.012)	.798

the samples. In particular, it analyzed `5,091` malicious files and `3,883` benign files. This is due to some parsing problems that undermine the system, as it does not fully implement all the Adobe specifications and only simulates the execution of embedded Javascript code and executables. We also point out that `PJScan` considered as *benign* all the samples for which it could not find evidence of *Javascript* code usable for the analysis.

Results show that our system definitely outperforms `Wepawet` and `PJScan`. `PJScan` shows the smallest false positive rate, but provides a much lower detection rate compared to the remaining solutions. `Wepawet` performs slightly better than our approach in terms of FP rate, but it exhibits a lower TP detection rate. Our approach performs better than `PDFRate`. In fact, results are better both in terms of TP and FP rate, with an higher MCC coefficient. Moreover, it is worth noting that our approach is superior to `PDFRate` while adopting a significantly lower number of features. In fact, `PDFRate` resorts to `202` features (Smutz and Stavrou, 2012) to perform its analysis, whilst our system has never gone beyond `135` (considering the variable number of object-related features).

5.2 Experiment 2: Evasion Attacks

Differently from current state of the art approaches, the features of our system, as well as its parsing mechanism, have been designed to consider the possibilities of *deliberate* attacks. In particular, an attacker can perform what has been presented as *reverse mimicry*, i.e., embedding malicious content inside PDFs that have been recognized as benign by the targeted system. This is done by only *slightly changing* the overall structure of the file, thus making the attack extremely effective against pure structural systems (Maiorca et al., 2013).

In our experiments, we implemented all the attacks described in our previous work, using the similar vulnerabilities[15]: **a)** *Javascript (JS) Injection* (injecting a Javascript object that exploits a vulnerability), **b)** *EXE Embedding* (injecting an executable that is automatically opened at runtime) and **c)** *PDF Embedding* (injecting a malicious PDF file that is opened after the main file). We then produced, for each attack, 500 attack variants for a total of 1500 samples. We observe that the samples produced in our previous work were only few compared to the ones we have created for this experiment. This was done to better assess the efficiency of the created attacks against the various systems.

Table 2 shows the performances, in terms of *true positives* (`TP`), of the systems tested during the previous experiment (training with the same data, with the same *splits* as before). We notice that `Wepawet` performs really well on *EXE Embedding* and *JS Injection*. That was expected since *reverse mimicry* addresses *static systems*. Interestingly, though, *Wepawet* was not able to scan *PDF Embedding* attacks due to parsing problems. We speculate that this might be related to the fact that `Wepawet` does not fully implement the PDF specifications, and therefore is not able to analyze some elements of the file. Such parsing problems also appeared with *PJScan*, that was not able to analyze *any* of the samples we provided. This is mainly due to the fact that this system is not able to analyze embedded files, i.e., PDFs or other files such as executables, and only focuses on *Javascript* analysis (which also failed, in this case). `PDFRate` performed really poorly, as also shown by our previous work (Maiorca et al., 2013).

As for our approach, the first thing we notice is that our system is able to detect all *PDF Embedding* attacks, thanks to its advanced parsing mechanism. In particular, the system automatically searches for objects that contain, in their dictionary, the keyword `/EmbeddedFiles`. If such object is found, the relative object stream is decompressed, saved as a separate PDF and then analyzed. If this file is found to be malicious, then the container of the original object stream will be considered malicious as well. For the other two attacks, a key role is played by the *learning algorithm parameters* that we chose to train our system. In fact, Table 2 shows that robustness is strongly dependent on two aspects:

- The *weight threshold* (W) parameter of the `AdaBoost` algorithm (Freund and Schapire, 1995) (expressed, in our case, as a *percentage*). Such value allows to select the samples that will be used, for each iteration of the `AdaBoost` algorithm, to train the weak classifiers and tune their weights. In particular, for each iteration, the samples are chosen as follows:

1. The training set samples are ordered by their

[15] For *EXE Embedding* we exploited the `CVE-2010-1240` vulnerability and for *PDF Embedding* and *Javascript Injection* we exploited the `CVE-2009-0927`.

Table 2: Comparison, in terms of true positives (TP), between our approach and research tools with respect to *evasion* attacks (%).

System	PDF E.	EXE E.	JS INJ.
Ours (Opt.)	**100** (±0)	**62.4** (±12.6)	**69.1** (±16.9)
Ours	**100** (±0)	**32.26** (±9.18)	**37.9** (±10.65)
PDFRate	0.8	0.6	5.2
Wepawet	0	99.6	100
PJScan	0	0	0

normalized weights (the lowest weight first). Samples that have been incorrectly classified at the previous iteration get higher weights. The normalized weights sum S_w is set to zero.

2. Starting from the first sample, $S_w = S_w + w_s$ is computed, where w_s is the normalized weight of the sample. If $S_w < W$[16], then the sample will be used for the training. If not, the algorithm stops.

- The *shape* of the decision function. Since these attacks directly address the *shape* of the classifier decision function (Maiorca et al., 2013) (which obviously depends on the weights of each weak classifier), sometimes it is necessary to correct it by using *resampling*, i.e., generating *artificial training data* from the samples set obtained given a specific weight threshold W. Such data will be then used to tune the weights of the samples that will train each weak learner, and therefore of the weak classifiers in the final linear combination. However, such correction might compromise an already robust function, so it must only be used on vulnerable shapes. We call this correction *function optimization*.

Using the default weight threshold, namely, $W = 100$ (the one adopted in experiment 1) with no optimization, performances are already better than `PDFRate` but still not fully satisfactory. With $W = 1$ and an optimized decision function, instead, performances are almost two times better, completely outperforming all the other static approaches. Using $W = 1$ on the test data of experiment 1, we also notice that false positives increase up to 0.2%. We interpret this result with the fact that, by reducing the value of W, we force the algorithm to ignore some samples that might have been incorrectly classified in the previous iterations. This makes the decision boundary less subjected to changes due to samples that might be difficult to be correctly classified. It is a small trade off we have to pay for a higher robustness. The standard deviation values will be discussed in the next section.

6 DISCUSSION

Results attained in the second experiment show that the features we have chosen allow for a significantly higher robustness when compared to the state of the art. However, the high standard deviation attained in Experiment 2 also shows some limits in our approach: in fact, we did not design a *robust* decision function from the beginning, as we mainly focused on defining significant features. As a consequence, our approach is extremely sensitive to the *training data* used, and in order to improve robustness we were forced to introduce some *optimizations* to correct the function parameters. For future work, it will be essential to design a *decision function* that, regardless of the quality of the training data, is able to reliably detect targeted and optimal attacks. This aspect has been often overlooked, especially in computer security applications and has been pointed out, for example, by Biggio et al. (Biggio et al., 2014b; Biggio et al., 2014a; Biggio et al., 2013a; Biggio et al., 2012; Biggio et al., 2010). Moreover, recent works have shown that clustering algorithms can also be vulnerable against evasion and poisoning attacks (Biggio et al., 2014c; Biggio et al., 2013b). Since our method resorts on a clustering phase, possible future works might also address its resilience against such attacks.

7 CONCLUSIONS

Malicious PDF files have become a well-known threat in the past years. PDF documents constitute a very effective attack vector for cyber-criminals. In spite of the efforts of software vendors such as Adobe, PDF software is often vulnerable to zero-day attacks. In this work, we presented a new approach that leveraged on both structural and content-based information to provide a very accurate detection of PDF malware. We also showed that our approach, with a careful choice of the learning algorithm, is also able to cope with evasion attacks. Finally, our work also clearly pointed out the need of secure learning techniques for malware detection, in order to cope with deliberate, adversarial attacks.

ACKNOWLEDGEMENT

This work is supported by the Regional Administration of Sardinia, Italy, within the project “Advanced and secure sharing of multimedia data over social networks in the future Internet“ (CUP

[16] If W is in its percentage form, it must be divided by 100 first.

F71J11000690002). Davide Maiorca gratefully acknowledges Sardinia Regional Government for the financial support of his PhD F.S.E. Operational Programme of the Autonomous Region of Sardinia, European Social Fund 2007-2013 - Axis IV Human Resources, Objective l.3, Line of Activity l.3.1.).

REFERENCES

Adobe (2006). *PDF Reference. Adobe Portable Document Format Version 1.7*. Adobe.

Adobe (2008). *Adobe Supplement to ISO 32000*. Adobe.

Baldi, P., Brunak, S., Chauvin, Y., Andersen, C. A. F., and Nielsen, H. (2000). Assessing the accuracy of prediction algorithms for classification: an overview. *Bioinformatics*, 16(5):412–424.

Bania, P. (2010). Jit spraying and mitigations. *CoRR*, abs/1009.1038.

Biggio, B., Corona, I., Maiorca, D., Nelson, B., Srndic, N., Laskov, P., Giacinto, G., and Roli, F. (2013a). Evasion attacks against machine learning at test time. In *M. Learning and Know. Discovery in Databases - Europ. Conf., ECML PKDD*, pages 387–402.

Biggio, B., Corona, I., Nelson, B., Rubinstein, B., Maiorca, D., Fumera, G., Giacinto, G., and Roli, F. (2014a). Security evaluation of support vector machines in adversarial environments. In Ma, Y. and Guo, G., editors, *Support Vector Machines Applications*, pages 105–153. Springer International Publishing.

Biggio, B., Fumera, G., and Roli, F. (2010). Multiple classifier systems for robust classifier design in adversarial environments. *Int'l J. Mach. Learn. and Cybernetics*, 1(1):27–41.

Biggio, B., Fumera, G., and Roli, F. (2014b). Security evaluation of pattern classifiers under attack. *IEEE Transactions on Knowledge and Data Engineering*, 26(4):984–996.

Biggio, B., Nelson, B., and Laskov, P. (2012). Poisoning attacks against support vector machines. In Langford, J. and Pineau, J., editors, *29th Int'l Conf. on M. Learning (ICML)*. Omnipress, Omnipress.

Biggio, B., Pillai, I., Bulò, S. R., Ariu, D., Pelillo, M., and Roli, F. (2013b). Is data clustering in adversarial settings secure? In *Proceedings of the 2013 ACM Workshop on Artificial Intelligence and Security*, AISec '13, pages 87–98, New York, NY, USA. ACM.

Biggio, B., Rieck, K., Ariu, D., Wressnegger, C., Corona, I., Giacinto, G., and Roli, F. (2014c). Poisoning behavioral malware clustering. In *Proc. 2014 Workshop on Artificial Intelligent and Security Workshop*, AISec '14, pages 27–36, New York, NY, USA. ACM.

Buchanan, E., Roemer, R., Sevage, S., and Shacham, H. (2008). Return-oriented programming: Exploitation without code injection. In *Black Hat '08*.

Canali, D., Cova, M., Vigna, G., and Kruegel, C. (2011). Prophiler: a fast filter for the large-scale detection of malicious web pages. In *Proc. of the 20th Int. Conf. on World Wide Web*.

Corona, I., Maiorca, D., Ariu, D., and Giacinto, G. (2014). Lux0r: Detection of malicious pdf-embedded javascript code through discriminant analysis of api references. In *To appear in the Proc. of the 7th ACM Workshop on Art. Intelligence and Security*.

Cova, M., Kruegel, C., and Vigna, G. (2010). Detection and analysis of drive-by-download attacks and malicious javascript code. In *Proc. of the 19th Int. Conf. on World Wide Web*.

Curtsinger, C., Livshits, B., Zorn, B., and Seifert, C. (2011). Zozzle: fast and precise in-browser javascript malware detection. In *Proc. of the 20th USENIX Conf. on Security*.

Engleberth, M., Willems, C., and Holz, T. (2009). Detecting malicious documents with combined static and dynamic analysis. In *Virus Bulletin*.

Esparza, J. M. (2011). Obfuscation and (non-)detection of malicious pdf files. In *S21Sec e-crime*.

Freund, Y. and Schapire, R. E. (1995). A decision-theoretic generalization of on-line learning and an application to boosting.

Laskov, P. and Šrndić, N. (2011). Static detection of malicious javascript-bearing pdf documents. In *Proc. of the 27th Annual Computer Security Applications Conf.*

Li, W.-J., Stolfo, S., Stavrou, A., Androulaki, E., and Keromytis, A. D. (2007). A study of malcode-bearing documents. In *Proc. of the 4th Int. Conf. on Detect. of Intrus. and Malware, and Vulnerability Assessment*.

Liu, D., Wang, H., and Stavrou, A. (2014). Detecting malicious javascript in pdf through document instrumentation. In *Proc. of the 44th Annual Int. Conf. on Dependable Systems and Networks*.

Maass, M., Scherlis, W. L., and Aldrich, J. (2014). Innimbo sandboxing. In *Proc. of the 2014 Symp. and Bootcamp on the Science of Security*, HotSoS '14, pages 1:1–1:12, New York, NY, USA. ACM.

MacQueen, J. B. (1967). Some methods for classification and analysis of multivariate observations. In Cam, L. M. L. and Neyman, J., editors, *Proc. of the fifth Berkeley Symp. on Mathematical Statistics and Probability*, volume 1, pages 281–297. University of California Press.

Maiorca, D., Corona, I., and Giacinto, G. (2013). Looking at the bag is not enough to find the bomb: An evasion of structural methods for malicious pdf files detection. In *Proc. of the 8th ACM SIGSAC Symp. on Information, Computer and Communications Security*.

Maiorca, D., Giacinto, G., and Corona, I. (2012). A pattern recognition system for malicious pdf files detection. In *Proc. of the 8th Int. Conf. on M. Learning and Data Mining in Pattern Recognition*.

Quinlan, J. R. (1996). Learning decision tree classifiers. *ACM Comput. Surv.*, 28(1):71–72.

Ratanaworabhan, P., Livshits, B., and Zorn, B. (2009). Nozzle: a defense against heap-spraying code injection attacks. In *Proc. of the 18th conf. on USENIX security symp.*

Rieck, K., Holz, T., Willems, C., Düssel, P., and Laskov, P. (2008). Learning and classification of malware behavior. In *Proc. of the 5th Int. Conf. on Detect. of Intrus. and Malware, and Vulnerability Assessment*.

Rieck, K., Krueger, T., and Dewald, A. (2010). Cujo: efficient detection and prevention of drive-by-download attacks. In *Proc. of the 26th Annual Computer Security Appl. Conference*.

Shafiq, M. Z., Khayam, S. A., and Farooq, M. (2008). Embedded malware detection using markov n-grams. In *Proc. of the 5th Int. Conf. on Detect. of Intrus. and Malware, and Vulnerability Assessment*.

Smutz, C. and Stavrou, A. (2012). Malicious pdf detection using metadata and structural features. In *Proc. of the 28th Annual Computer Security Appl. Conference*.

Snow, K. Z., Krishnan, S., Monrose, F., and Provos, N. (2011). Shellos: enabling fast detection and forensic analysis of code injection attacks. In *Proc. of the 20th USENIX conf. on Security*.

Symantec (2014). *Internet Security Threat Reports. 2013 Trends.* Symantec.

Tabish, S. M., Shafiq, M. Z., and Farooq, M. (2009). Malware detection using statistical analysis of byte-level file content. In *Proc. of the ACM SIGKDD Work. on CyberSecurity and Intelligence Informatics*.

Tzermias, Z., Sykiotakis, G., Polychronakis, M., and Markatos, E. P. (2011). Combining static and dynamic analysis for the detection of malicious documents. In *Proc. of the 4th Europ. Work. on System Security*.

Šrndić, N. and Laskov, P. (2013). Detection of malicious pdf files based on hierarchical document structure. In *Proc. of the 20th Annual Network & Distributed System Security Symp.*

Šrndic, N. and Laskov, P. (2014). Practical evasion of a learning-based classifier: A case study. In *Proc. of the 2014 IEEE Symp. on Security and Privacy*, SP '14, pages 197–211, Washington, DC, USA. IEEE Computer Society.

Willems, C., Holz, T., and Freiling, F. (2007). Toward automated dynamic malware analysis using cwsandbox. *IEEE Security and Privacy*, 5(2).