# Malicious PDF Documents Detection using Machine Learning Techniques
*A Practical Approach with Cloud Computing Applications*

Jose Torres and Sergio De Los Santos
*Telefónica Digital España, Ronda de la Comunicación S/N, Madrid, Spain*

Keywords: PDF, Malware, JavaScript, Machine Learning, Malware Detection.

Abstract: PDF has been historically used as a popular way to spread malware. The file is opened because of the confidence the user has in this format, and malware executed because of any vulnerability found in the reader that parses the file and gets to execute code. Most of the time, JavaScript is involved in some way in this process, exploiting the vulnerability or tricking the user to get infected. This work aims to verify whether using Machine Learning techniques for malware detection in PDF documents with JavaScript embedded could result in an effective way to reinforce traditional solutions like antivirus, sandboxes, etc. Additionally, we have developed a base framework for malware detection in PDF files, specially designed for cloud computing services, that allows to analyse documents online without needing the document content itself, thus preserving privacy. In this paper we will present the comparison results between different supervised machine learning algorithms in malware detection and a overall description of our classification framework.

## 1 INTRODUCTION

Since the PDF format creation by Adobe in 1994, malformed PDF documents to compromise systems (exploiting vulnerabilities in the reader), has been a very popular attack vector and still is. Specifically, most of significant attacks related with PDF format have been associated with vulnerabilities in Adobe software such as Adobe Reader and Adobe Acrobat. According to several studies, in 2009, approximately 52.6% of targeted attacks used PDF exploits and by 2011, it got to 76%.

PDF is a very popular and trusted extension, while Adobe Reader is the most used program for opening these kind of files. These factors encourage attackers to research and seek for vulnerabilities and new ways of creating exploits that will execute arbitrary code when opened with this specific software. There are some vulnerabilities examples that (even today) are popular as an attack vector, like a vulnerability found in 2008 in Adobe (Adobe, 2008) that would allow the attacker to execute arbitrary code in Adobe Reader and Acrobat 8.1.1 and earlier versions. The extended use of this type of documents, especially for document sharing via email, makes it possible form malware to spread effectively. Users are prone to open received PDF files and interact with the document without noticing any associated danger, as it would occur with executable files.

JavaScript presence into PDF documents provides extra functionalities to the document and makes it interactive. An example of that extra features are forms, which usually include check-boxes, text inputs, action buttons, etc. that provides the document with the typical behavior of an application. Since Acrobat Reader included support for JavaScript (Wikipedia, 2017) (Version 3.02, year 1999) the use of JavaScript in PDF files to execute malicious actions has been very popular, even prior to the inclusion of PDF format as an Open Standard (ISO/IEC 32000-1:2008). Sometimes, the malicious JavaScript code into the document is not the final malware itself, but rather a mechanism to download and execute an instance of certain malware or even a helping code to exploit a certain vulnerability in the reader. Although since at least version 6, JavaScript can be disabled in Adobe Reader, the risks of JavaScript code included in PDF format has been historically advised: In 2006 David Kierznowski provided sample PDF files illustrating JavaScript vulnerabilities. And even (but less frequent) without using JavaScript: In 2010, Didier Stevens was able to execute arbitrary code from a PDF file processed by Adobe Reader without even exploiting anything but the PDF specifications themselves.

Torres, J. and Santos, S.
Malicious PDF Documents Detection using Machine Learning Techniques - A Practical Approach with Cloud Computing Applications.
DOI: 10.5220/0006609503370344
In *Proceedings of the 4th International Conference on Information Systems Security and Privacy (ICISSP 2018)*, pages 337-344
ISBN: 978-989-758-282-0

The number of (in)famous vulnerabilities in Adobe that, still today, are used to execute arbitrary code and spread malware is significant: CVE-2016-4190, CVE-2014-0496, CVE-2013-3346, CVE-2016-6946, CVE-2009-0658, CVE-2009-0927, CVE-2007-5659, CVE-2007-5020, CVE-2010-1297, CVE-2010-2883... Most of them use JavaSCript language inside to execute its payload.

After some really serious security problems during 2007-2011 period, Adobe Reader tried to improve its security adding a sandbox (called Protected View) to Adobe Reader 10.1 and disabling more potential attacks vectors by default.

## 1.1 Paper Organization

The rest of the present article is organized as follows. Section two analyses the technical structure of PDF documents and provides a historical introduction; section three shows the background both in terms of related studies and existing tools, introducing the problem; section four describes and extends the development of the proposal; section five presents the results obtained; section six briefly introduces the analysis framework developed as continuation of the presented classifier; finally, section seven describes the conclusions and possible future work.

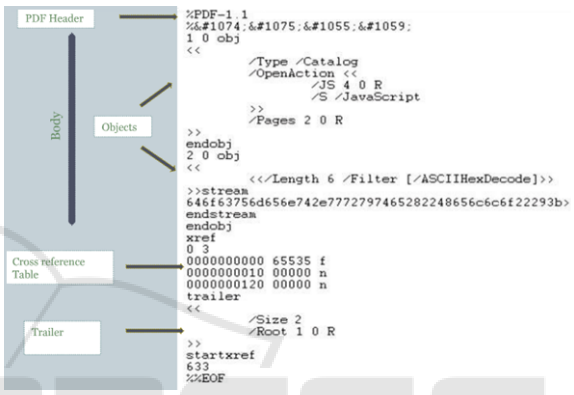
## 2 BACKGROUND

PDF files are plain text files with magic number ”%PDF” (hex 25 50 44 46) and composed by a static set of sections (Figure 1). Sections included into a PDF document are the following (in order of appearance):

- **PDF Header:** Contains the PDF format version, e.g. ”%PDF-1.4”
- **PDF Body:** Composed by a set of objects used to define the contents of the document. There can be of different types, such as images, fonts, etc. This section can also contain another type of objects related with features like animations or security.
- **Cross-Reference Table (xref Table):** The cross-reference table target purpose to link all existing objects into the document. The xref table allows to find and locate content into the document, e.g between different pages. If the document changes, the table is automatically updated.
- **Trailer:** Is the last section into the document and contains the FDF file EOF indicator (%%EOF) and links to the cross-reference that allow users to navigate into the document.

JavaScript code can be included as objects into the document, which implies that (as an object) the code can be found, referenced, etc. The usual attack vector is using JavaScript code as a payload to take advantage of the PDF reader, exploiting some vulnerability.

The use of JavaScript in PDF documents was introduced in the PDF 1.2 format together with Adobe Acrobat Version 3.02 (1999) as part of AcroForms (Appligent, 2013) feature, so all later versions are potentially able to be affected by malicious JavaScript because of this feature enabled by default. As Table 1 shows, this implies more than seven PDF and Adobe Acrobat versions.



```
%PDF-1.1
%&#1074;&#1075;&#1055;&#1059;
1 0 obj
<<
        /Type /Catalog
        /OpenAction <<
                /JS 4 0 R
                /S /JavaScript
        >>
        /Pages 2 0 R
>>
endobj
2 0 obj
<<
        <</Length 6 /Filter [/ASCIIHexDecode]>>
>>stream
646f63756d656e742e77727974652822248656c6c6f22293b>
endstream
endobj
xref
0 3
0000000000 65535 f
0000000010 00000 n
0000000120 00000 n
trailer
<<
        /Size 2
        /Root 1 0 R
>>
startxref
633
%%EOF
```

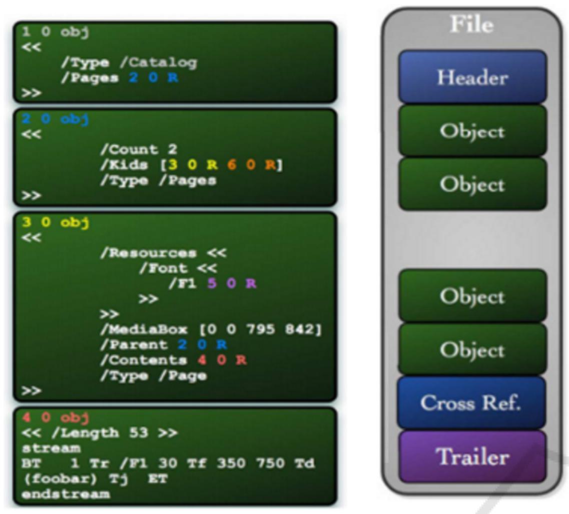
Figure 1: PDF document basic structure. Source: http://infosecinstitute.com.

Table 1: Adobe and PDF format versions.

| Year | PDF Version | Adobe Acrobat Version |
|---|---|---|
| 1993 | PDF 1.0 | Acrobat 1.0 |
| 1994 | PDF 1.1 | Acrobat 2.0 |
| 1996 | PDF 1.2 | Acrobat 3.0 |
| 1999 | PDF 1.3 | Acrobat 4.0 |
| 2001 | PDF 1.4 | Acrobat 5.0 |
| 2003 | PDF 1.5 | Acrobat 6.0 |
| 2005 | PDF 1.6 | Acrobat 7.0 |
| 2006 | PDF 1.7 | Acrobat 8.0 / ISO 32000 |
| 2008 | PDF 1.7, Adobe Extension Level 3 | Acrobat 9.0 |
| 2009 | PDF 1.7, Adobe Extension Level 5 | Acrobat 9.1 |
| 2017 (Exp) | PDF 2.0 | Acrobat XI (or later) |

## 2.1 Code Components

Aside from the sections, the general structure of a PDF file is composed of different code components (types of objects). Components are mostly used into the PDF Body section as part of the different objects disposed as illustrated in Figure 2. The eight basic code components defined by the PDF standard are Boolean values, String, Names, Arrays, Dictionaries, the null object and Streams. From these, Streams are specially interesting from the security standpoint, since they allow to store large amount of data. Usually Stream components are used to store executable code in order to run it after a certain event, such as pressing a button. The way code execution is performed in a PDF document is through launching actions included into objects. Legitimately, launch actions should be used for opening or printing documents, for instance. However it is possible to use them as an attack vector.



```
1 0 obj
<<
    /Type /Catalog
    /Pages 2 0 R
>>
```

```
2 0 obj
<<
        /Count 2
        /Kids [3 0 R 6 0 R]
        /Type /Pages
>>
```

```
3 0 obj
<<
        /Resources <<
            /Font <<
               /F1 5 0 R
            >>
        >>
        /MediaBox [0 0 795 842]
        /Parent 2 0 R
        /Contents 4 0 R
        /Type /Page
>>
```

```
4 0 obj
<< /Length 53 >>
stream
BT   1 Tr /F1 30 Tf 350 750 Td
(foobar) Tj  ET
endstream
```

Figure 2: Example of body section in PDF documents. Source: http://infosecinstitute.com.

## 2.2 PDF Metadata

PDF documents can contain two types of metadata. Firstly, standard document metadata, which contains the expected regular information for documents (e.g. author, access and modification date, etc.); secondly, specific metadata, containing specific info such as versions info (with individualized information), creation application, etc. This specific metadata is stored into the Trailer section of the document.

When researching about malware in PDF documents, a combination of both (standard and specific metadata) allows to extract valuable information in order to detect malicious documents if combined with a deep analysis of the document content by human analysts or an intelligent system in this case.

## 3 STATE OF THE ART

Regarding PDF malware detection, antivirus industry has made great efforts over the last years in this matter. Aside, Adobe Acrobat has improved security in its products significantly, including an internal sandbox (Adobe, 2017) to mitigate the execution of malware. This sandbox drastically reduces the chance to impact the operative system once the vulnerability is exploited. Analysing PDF files searching for malware has been an important topic in both academics and independent studies. Many different approaches and techniques related with malware detection in PDF are easy to find. As a consequence, there are a few quite known tools for PDF documents analysis, that allow to inspect the content of the document and asily check for the presence of potentially dangerous elements such as ULRs, JavaScript code, launch actions, etc. Some of the most relevant tools are the following:

- **PeePDF:** Is one of the most complete tools for PDF documents analysis. It joins together most of the necessary components that a security researcher could need when performing a deep PDF analysis. It parses the entire document showing the suspicious elements, detects obfuscation presence and provides a specific wrapper for JavaScript and shellcode analysis.
- **PDF Tools** (Stevens, 2017): A complete set of tools for analysing PDF files, such as parsers, dumpers, etc. One of the most useful included tool is PDFID, a light Python tool for scanning PDF files seeking for certain PDF keywords that allows to identify (for instance) if the document contains JavaScript or executes some action when opened. The advantage of PDFID over other tools is its simplicity and efficiency. It does not need to completely parse the document.
- **PDF Examiner** (Tylabs, 2017): Is an online solution able to scan a document for several known exploits. It also offers the possibility of inspecting the structure of the file, as well as examining, decoding and dumping PDF object contents.
- **O-checker:** A tool developed by Otsubo et al. as part of an study (Otsubo, 2016) of malware detection in PDF documents, based on the identification of malformed documents.

As related work, different approaches for static malware detection in PDF documents has been described in recent literature. From these, one of the most frequently adopted is the use of machine learning techniques, based in particular features of the document itself to improve malware detection. The main advantage of this approach is a high effectiveness against new attacks or malware families never seen before, in contrast with the traditional solutions like signature based systems, which mostly is effective against known malware.

Regarding to the use of the PDF file structure as a detection vector, (Otsubo, 2016) represents an implementation where this approach has been taken to its extremes. Thus, only features related with how PDF has been crafted are taken into account, such as unreferenced objects, camouflaged streams, camouflaged filters, etc. The basic premise of this approach is that, if the structure of a PDF document has been altered, malware presence in the document is very likely. In this case, the presented system does not rely on machine learning, but just uses predefined rules for determining whether the document is malformed (which implies malware) or not. From our perspective, this solution is very interesting as an additional source of information for a machine learning implementation together with others. In our work, we have included this information as a feature to create the feature vector of our system.

When using machine learning approaches, the success depends on the multiple areas of the documents or strategies for extracting information that will allow the algorithms to be more accurate. For example, in (Laskov and Šrndic, 2011), Laskov et al. purpose the use of lexical analysis of JavaScript code as the input for the machine learning system. However, if the JavaScript code has been designed to look like legitimate code, e.g. blending the malicious one with a big portion of legitimate code, as described in (los Santos, 2016), the lexical analysis of the code may not be effective enough for malware detection. Another popular approach is the use of document metadata as an input for the features vector used for predictions. For example, (Smutz and Stavrou, 2012) uses mainly the properties of documents metadata in combination with what they called ”Structural features” of the document. Those features are closely related with the document content, e.g. number of included images, strings size, etc. The use of metadata is an interesting approach, but usually this metadata is counterfeited by the attacker, so it's necessary take into account not only the typical metadata in malicious documents but rather the counterfeited one. Furthermore, evaluating the content of the document could be an effective solution for detecting a specific set of malware types (such as phishing attacks, where the attacker emulates the appearance of an official document of an specific organization) but may not be effective enough when the attacker attaches the malicious content into a legitimate document.

As a downside of these aforementioned cases, samples for both training and test datasets have been selected from public malware repositories which usually implies that the system will be trained with old and very known samples and malware families. This situation implies losing the focus in the main advantage of machine learning use, that is, get a step ahead to the signature based systems in new malware detection.

## 4 DESCRIPTION OF THE PROPOSAL

The presented proposal is based on the application of machine learning techniques for detecting fresh and real malware, focusing in PDF format and determining if it is possible to improve regular and typical static solutions and become more effective. Specifically, the aim is to get ahead to these solutions in detection of new types/families of malware, developing a solution as a complement for them. This solution may be used as an isolated framework for malware detection that does not need the PDF document content itself to work. This framework can be easily introduced in cloud computing architectures to analyse users documents preserving their privacy.

### 4.1 Samples Identification and Collection

The main goal at this stage of the study is to get an initial dataset which represents (as realistic as possible) the final environment where the classifier will operate. For this purpose we have collected PDF malware samples from different sources and different types (PDF document format versions). We have used a significant percentage of recent samples (about 80% of the PDF samples collected were created in 2016 or 2017). JavaScript presence is a required characteristic for all collected samples.

Regarding to the sources where samples were obtained, we have selected them taking into account we wanted to represent the best possible real world scenario. We have used email accounts that usually receive spam or malicious attached documents, public malware repositories (malwr.com, Contagio, etc) and document repositories in general (P2P networks, search engines, etc.). In addition, in order to collect as much recent malware samples in the wild as possible, most of included samples have been obtained from the Cyber Threat Alliance (CTA, 2017), a dedicated threat data sharing platform based on the cooperation between companies in the industry. From all these sources, we have collected a total of 1712 samples.

For samples identification and classification, we have defined two different classes or sets:

- Goodware: Samples with JavaScript extracted from trusted sites (public document repositories of public entities, universities, etc) without suspicious characteristics detected by heuristics, e.g. obfuscation, suspicious calls to API, etc. As a final layer to ensure that these samples were benign, we used a set of antivirus engines to verify that no sample was detected as malware.

- Malware: Samples downloaded from different sources (email honeypots, public and private malware repositories, etc). To ensure that all collected samples are considered malware (not fully confirmed), aside only the ones detected by more than three reputed antivirus engines have been considered.

## 4.2 Feature Selection and Extraction

As mentioned in section 3, there are different approaches for the extraction of features from a PDF document (metadata features, document structure, content analysis, etc). In our implementation, we have selected a combination of these approaches. We have collected features from the document metadata such as number of versions, edition time, size, etc. Regarding to the document structure, we took into account both presence of possible anomalies in the internal structure of the file (with regard to the PDF format specification) and features related with how the document and its components (objects) are organized. Anomalies in the file format are detected using (Otsubo, 2016) and encoded as a binary predictor. Regarding to the document structure we have selected features from the whole document (number of objects, strings, references, etc.) and a significant number of features related specifically with the JavaScript code embedded in the document. From JavaScript code, we have extracted features usually linked with malware presence that can be considered as ”suspicious”, e.g. IPs, calls to functions related with auto-execution, obfuscation presence, etc.

Extracting and processing features from a heterogeneous set of inputs with a high level of granularity, implies a higher processing time than other approaches, so we have to focus in performance but in balance with getting maximum accuracy, reducing the false negative ratio. The average processing time for the dissection of a document and it classification is less than one second for all collected samples. In exceptional cases, if the sample processing time required is greater than 2 minutes, the sample is automatically rejected.

Once all documents features were selected and extracted, a dimensionality reduction was applied using PCA. As a result, we obtained a final set of 28 final features which compose a predictors vector as the input for the Machine Learning algorithms. Figure 3 shows these features sorted by importance descending in Random Forest model. As observed, some features are significantly valuable in the model (from F1 to F8) with regard to others. Even a subset of features are not being taken into account by the model, however it does not necessarily mean that this same weights distribution persists over time. The presented implementation has been designed to work with a dynamic set of samples sources, allowing to incorporate new sources and retrain the model to adapt it to the typical features of these new samples, which usually implies changes in the model, such as the importance of each feature (e.g. samples from academic sources contains substantially different features with respect to samples collected from the Internet in general). The system also allows to use different prediction algorithms interchangeably.

In order to preserve the documents author privacy, no feature related with the author or the documents private content have been used as a predictor.

## 4.3 Classifier Design and Basic Functioning

For our implementation we have chosen a supervised strategy to address the problem as a binary classification one, since the analysed samples could be only ”malware” or ”goodware” (not-malicious). Therefore, as usual, we have divided the total amount of recovered samples into three datasets: training, test and validation. Those three dataset are structured according to the form $\left\{X_i, y_{iN_v}^{i=1}\right\}$ where $y_i \in \{Goodware, Malware\}$ is the categorical variable and $X_i = [X_{i1}, \dots, X_{iN}]\,|X_i \in \{0,1\}$ represents each of the $N_v$ vectors of N predictors, contained in the dataset.

The collected samples were divided into training set, test set and validation set as follows:

- Training Set: 995 samples
- Validation Set: 217 samples
- Test Set: 500 samples (Collected from private shared malware samples repository)

In order to select the best possible subset of classification algorithms in terms of performance, after an initial pre-selection stage, the selected algorithms were Support Vector Machine, Random Forest and Multilayer Perceptron. These three algorithms were trained with the same training set and evaluated using the aforementioned test and validation sets. As performance estimators we have selected four different metrics to evaluate the strengths and weaknesses of each algorithm.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \qquad (1)$$

$$Recall = \frac{TP}{TP + FN} \qquad (2)$$

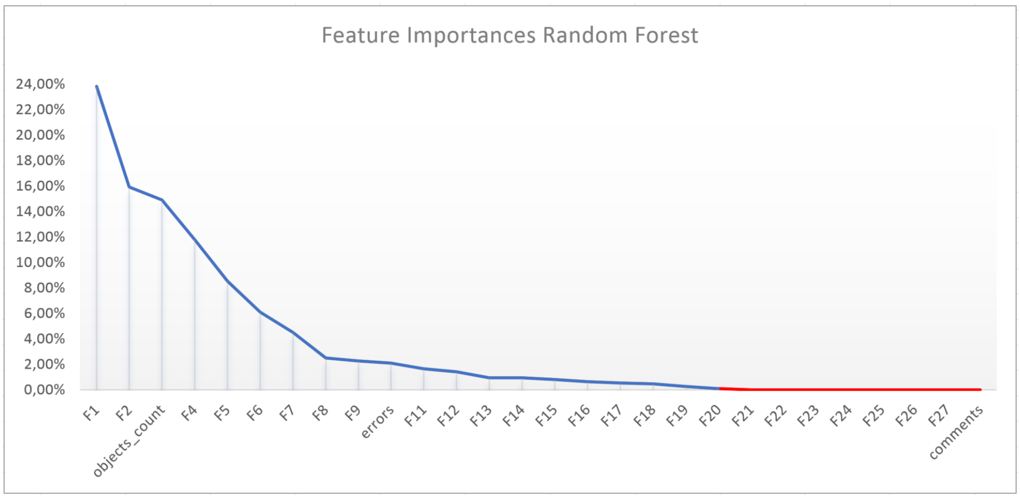


Figure 3: Selected document features sorted by importance descending in the RF model.

$$F1-Score = \frac{precision \cdot recall}{precision + recall} \qquad Precision = \frac{TP}{TP+FP} \tag{3}$$

$$ROC-AUC = \int\int_{-\infty}^{\infty} TPR(T)(-FPR'(T))dT$$

$$TPR = TruePositiveRate \qquad FPR = FalsePositiveRate \qquad T = Thresholdparameter \tag{4}$$

Additionally we use confusion matrices (where Malware is the positive class) to compare the presence of false positives (FP) and false negatives (FN) in all possible cases.

## 5 RESULTS

### 5.1 Validation Results

In order to select a subset of the most promising supervised Machine Learning algorithms and as a first evaluation of our proposal, the system was tested using different algorithms over the aforementioned validation set. After a first selection, particularly taking into account time consumption, the selected algorithms were: Support Vector Machine, Random Forest and Neural Networks (in our case a Multilayer Perceptron).

As Table 2 shows, Random Forest presents the best classification results over the validation set. On the other hand, SVM gets significant worse results than the rest, with an accuracy of 0.75. Despite poor-performance of SVM with respect to Random Forest and MLP, we kept it in order to measure its performance in test phase.

Table 2: Algorithms performance comparison in validation phase.

| Algorithm | Accuracy | Recall | F1-Score | ROC-AUC |
|---|---|---|---|---|
| SVM | 0.75 | 1 | 0.85 | 0.86 |
| RF | 0.92 | 0.95 | 0.94 | 0.95 |
| MLP | 0.91 | 0.95 | 0.94 | 0.95 |

### 5.2 Test Results

For testing, samples have been collected from different sources than training and test validation. Using different sources for this phase allows us to emulate the worst case in malware classification and the real environment where de classifier will operate. To achieve that, we have collected malware samples from private shared repositories and goodware from the eDonkey/Kad networks using eMule, where the presence of malicious PDF is virtually discarted by (eMule, 2016). Goodware samples in this work have been downloaded using the same procedure and after that, examined by different antivirus engines to discard any suspicious sample.

Training the model with the most heterogeneous possible set of samples and testing it with real and recent samples, allows us to ensure that our model will still work fine over time in a real life situation. In addition, the framework were the classifier will operate, allows to easily retrain it using new samples and updating the models according to the new eventual possible malware landscape.

As expected, performance results (Table 3) for the selected algorithms in test phase, show that SVM behaviour has worsened drastically. However, RF and MLP are even better in most cases. Because of that, we will mainly focus our analysis in these two algorithms.

Table 3: Performance comparison in test phase.

| Algorithm | Accuracy | Recall | F1-Score | ROC-AUC |
|---|---|---|---|---|
| SVM | 0.50 | 0 | 0 | 0.70 |
| RF | 0.92 | 0.94 | 0.92 | 0.98 |
| MLP | 0.96 | 0.967 | 0.96 | 0.98 |

Though performance results in both RF and MLP are quite similar, MLP gets betters results in each evaluated metric as Table 3 shows. Even though Area Under Curve ROC (ROC-AUC) is the same for both classifiers, in Figure 5 is possible to appreciate how ROC curve for MLP is closer to an ideal ROC curve.

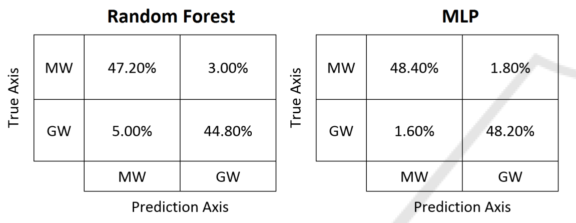

Random Forest
MLP
True Axis
Prediction Axis
MW
GW
47.20%
3.00%
5.00%
44.80%
48.40%
1.80%
1.60%
48.20%

Figure 4: Confusion matrices of RF and MLP in test phase.

In addition to the metrics above, what happens when the classifier fails in prediction needs to be taken into account. In malware detection case, we are specially interested in False Negative cases, because it implies malware going undetected. Nevertheless, it is also necessary to be taken into account that a high False Positives ratio would mean that the classifier will often detect legitimate samples as malware, which could result in an unreliable system.

Regarding to False Positives and Negatives, MLP has shown much better results than RF. As Figure 4 displays, in False Negative terms, MLP a 1.8% with regard to RF (3%). For False Positives, MLP results are even better, where MLP obtains approximately a third party of RF False Positives.

Besides above, in our tests, MLP have also achieved slightly better results in time consumption for prediction, with an average (over 1000 simulations) of 15% less than RF.

## 6 ANALYSIS FRAMEWORK

Implementation details of the final framework where the classification system works, are out of the scope of this paper. However, it is necessary to explain its target and main functioning in order to understand the importance of malware detection using the classifier presented in this paper.

As stated above, our classifier system does not need any feature related with the content of the document nor the author information. Thus, the classifier can be embedded into the framework in server side, using just an anonymous representation (that we called vectorized form) of a document for predicting. In client side, the document analysed by the user is processed, vectorized an then sent to the server.

The advantage of using the vectorized form of the document is that it is possible to uniquely identify a document with a hash transmitted with the vector and after that, recover the prediction result (requesting a hash) without the necessity of transmitting the whole document itself. Of course, the vectorized form of a document, does not allow to uniquely identify a document.

Due to the possibility of using dynamically different classifications algorithms and even versions of these algorithms, all trained models are stored and tagged using and algorithm ID and a creation timestamp. Hence, the developed framework can use different classifiers instances, retraining them periodically with new collected samples and generating new “classifier snapshots” if the new trained model improves prediction results.

Because of the above, when clients request the analysis results of a document, the server will send a prediction together with the classifier and its version.

## 7 CONCLUSION AND FUTURE WORK

The aim of this project is to demonstrate if Machine Learning techniques could be used as a good approach for PDF malware detection, using characteristics from the document that could help to determine when a samples is malicious while respecting document and user privacy. This kind of experiments does not seek to replace traditional solutions, such as antivirus engines, but to complement them and if needed, assists analysts who design and update them.

During the preparation of this work, we have developed tools for PDF document dissection and analysis, also improving some existing others. Using these tools, we have designed a set of document features and built a classifier which uses them for malicious PDF document detection, as the result of a comparison of several previously trained classification algorithms.

As a consequence, we have built a framework that integrates the classifier and adds an extra value to the

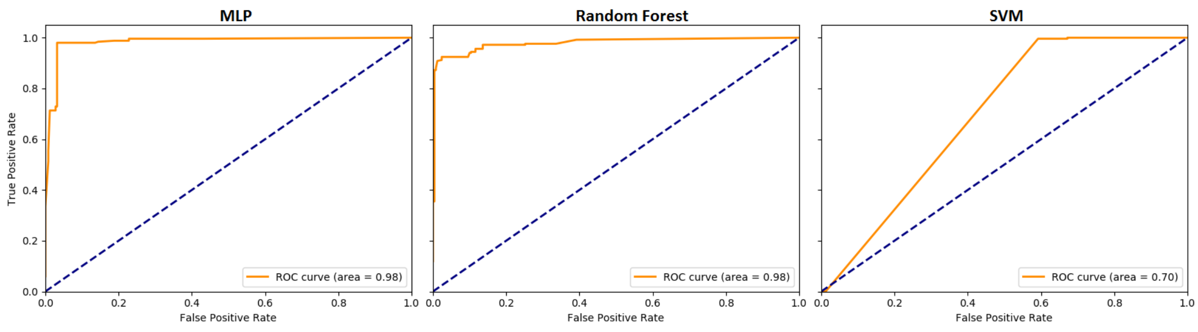


Figure 5: ROC curves for the studied classification algorithms with the test set.

results achieved, allowing any malware analyst or researcher to improve, experiment and research further with new data and algorithms.

However, there is still some room for further improvement. Below is a summary of several of these chances.

We have to remark that using antivirus engines as previous classification systems to train the algorithms, implies that the classifier could inherit their successes and advantages but also their mistakes and disadvantages. In other words, using this approach, the classifier may learn from antivirus false positives and negatives as correct inputs. To minimize the presence of this fact, we have considered as malware samples collected from reputed malware repositories and honeypots, not focusing on antivirus detection level. For goodware samples, aside antivirus validation, we have taken into account another factors as a semi-automated search for malware indicators.

Futhermore, the use of a most detailed syntactic analysis of the JavaScript code included into the document, as described in (Laskov and Šrndic, 2011), could improve the results of our system if combined with the existing solution. However it could affect the system performance in terms of time consumption.

In addition, our system completely parses the document, which implies complex operations, increasing the mean processing time for samples. However, it allow us to extract most of the possible features of the document. This makes it possible to apply a wide variety of existing approaches for malware detection using machine learning.

Regardless of the aforementioned risk factors (already mitigated as much as possible), we has demonstrated that using the presented approach produces promising results in malware detection, reaching accuracy above 96%, with a false positive and negative rate below 1.8%, which is an acceptable rate in antivirus industry.

All of this demonstrates that using anonymous properties from PDF documents mainly focused in how the document have been created could result in an effective approach for malware classification.

## REFERENCES

Adobe (2008). Adobe security bulletin. https://www.adobe.com/support/security/bulletins/apsb08-13.html. [Online; accessed 12-July-2017].

Adobe (2017). Adobe acrobat protected mode. https://www.adobe.com/devnet-docs/acrobatetk/tools/AppSec/ protectedmode.html. [Online; accessed 17-July-2017].

Appligent (2013). Sa quick introduction to acrobat forms technology. http://www.appligent.com/wp-content/uploads/2013/02/Acroforms+WhitePaper.pdf. [Online; accessed 12-July-2017].

CTA (2017). Cyber threat alliance. https://cyberthreatalliance.org/about/. [Online; accessed 05-August-2017].

eMule (2016). emule: is it always a source of malicious files? http://blog.elevenpaths.com/2016/08/nuevo-informe-emule-siempre-una-fuente.html. [Online; accessed 05-August-2017].

Laskov, P. and Šrndic, N. (2011). Static detection of malicious javascript-bearing pdf documents.

los Santos, S. D. (2016). Camufladas, no ofuscadas: Malware de macro creado en espaa? http://blog. elevenpaths.com/2016/05/camufladas-no-ofuscadas-malware-de.html. [Online; accessed 17-July-2017].

Otsubo, Y. (2016). O-checker: Detection of malicious documents through deviation from file format specifications. *Blackhat*, page 16.

Smutz, C. and Stavrou, A. (2012). Malicious pdf detection using metadata and structural features. In *Proceedings of the 28th annual computer security applications conference*, pages 239–248. ACM.

Stevens, D. (2017). Pdf tools. https://blog.didierstevens.com/ programs/pdf-tools/. [Online; accessed 17-July-2017].

Tylabs (2017). Pdf examiner. https://pdfexaminer.com/. [Online; accessed 17-July-2017].

Wikipedia (2017). Adobe acrobat — wikipedia, the free encyclopedia. https [Online; accessed 12-July-2017].