PDF Forensic Analysis System using YARA

Suleiman J. Khitan[1], Ali Hadi[1] and Jalal Atoum[2]
[1]Computer Science Dept., Princess Sumaya University for Technology, Amman, Jordan
[2]Mathematics & Computer Science, Southern Arkansas University, Texas, USA

Summary
This this paper presents an important enhanced method to detect suspicious PDF files by applying two scanning methods (structure scan and YARA scan), which depend on extracting and pointing out malicious objects that are often used for attacks. This enhanced method will be a great assistant to forensic analysts in analyzing PDF files and detecting malicious content in them. Testing both scanning methods was carried out through conducting several experiments on a real dataset. The results show an improvement for detecting malicious PDF files when applying both methods. The structure scan achieved an accuracy of 99.91% and the YARA scan achieved an accuracy of 98.05%.

Keywords:
Malware Analysis, PDF Documents, Malicious PDF, Suspicious PDF, Structure Scan, YARA Rules, Learning Machines.

1. Introduction

Recent developments in network technology have become better at showing the importance and expansion of data exchange; that is, data such as video files, images and documents rapidly sent from one machine to another. The widespread importance of the information security has raised awareness about anonymous execution files, and has made hackers to think twice before they propagate their malicious code using most common file types.

Many users have a wrong understanding that document files, like Microsoft documents and PDF files, are the most protected and trusted compared to execution files. In fact, hackers can embed their malicious codes within these document files – and by fooling users to open these files, they are turned into easy targets.

The Portable Document Format (PDF), developed by Adobe Systems in 1993, has become the file format for the distribution of printable documents these days and was released as an open standard by the International Organization for Standardization in 2008 as ISO 32000-1 [1].

In contrast to other document files like Microsoft Office, PDF files are considered the most widespread application, enabling individuals to easily transfer electronic documents in trusted ways without depending on a specific platform. In addition to these reasons, the PDF format supports Application Programming Interfaces (APIs) in filling fields in forms for survey questions, and provides rich elements to the users. The rich elements explained in the PDF structure contain static and dynamic contents. Table 1 displays a brief list of these contents.

Table 1 PDF Content

Static Contents	Dynamic Contents
Text formats &Layout	JavaScript within a file
Encodings used & Font	Dynamic Action
Support for Multimedia	Getting online Data

The API features supported by PDFs may be exploited for cyberattacks [2] [3]. In addition, the dynamic content may lead to several security issues that can be used to hold malicious elements to install malware and steal data. These features may contain codes written in JavaScript and will allow the developer to insert advanced features, such as multimedia files, to connect with outside sites. Unfortunately, the attacker can use the features provided by JavaScript to exploit vulnerabilities in the PDF viewer application itself.

By using JavaScript, the attacker is capable of doing two things: triggering the vulnerable code and then pointing the execution to an arbitrary code of their choice to gain user privileges to run or stop the application; or denying service to the legitimate user through heap-spraying [4] or other memory manipulation techniques.

In addition to the vulnerabilities of the PDF viewer, attackers have also taken advantage of advanced PDF features such as the /Launch option, which executes an embedded script automatically, or the /URI and /GoTo options, which can open external resources from the same computer [5].

Vulnerabilities in PDFs are grouped into two classes [6]: JavaScript-based and non-JavaScript-based. A JavaScript-based exploit is achieved in the PDF because the PDF standards support the JavaScript language, which enables attackers to embed it into an object inside the document. Here, the goal is to exploit the bugs in the implementation of PDF JavaScript API and to use a technique like heap spraying to fill the PDF reader memory with a shell-code which also gives the attacker the opportunity to execute this shell-code. This may involve downloading malware from the internet or extracting it from the PDF files themselves, writing it to the file and executing it [7].

A non-JavaScript-based exploit, which is utilized by the attackers using some PDF features, is rarely encountered

Manuscript received May 5, 2017
Manuscript revised May 20, 2017

compared to the JavaScript-based exploit; CVE-2011-0611-Flash, for example, is a weakness in Flash Player that causes a denial of service [8]. Another example of a non-JavaScript exploit is using the /Launch action, which may be used to run a malicious code automatically as described in CVE-2010-1240 [8].

Analyzing PDF files depends mainly on three forms of technique: static, dynamic and a combination of the two. The static technique [9] depends on inspecting the PDF documents thoroughly and searching for features and content which are important for labeling PDF documents as clean or not. The drawback to this solution is that it is restricted to finding obfuscations that hide malicious code, which leads to a high false positive rate. The dynamic technique [10] [11] depends on running the files in a monitored virtual machine and analyzing them for any vulnerable behavior. The limitations of this technique are the need for a longer analysis time and more expense to create the monitored system. The third technique combines static and dynamic analysis in order to analyze and detect malicious files [12] [13] [14]. The advantage of this over the two other techniques is that it does not need much time for analysis – unlike the dynamic technique – and, in addition, it can provide a high positive rate compared to the static technique.

In this paper, a method for detecting and classifying suspicious PDF files is presented based on a structure scan [15] and a YARA scan, which inspect the PDF documents thoroughly and search for features that are important in labeling PDF documents as suspicious. In addition, the dataset of clean and malicious PDF files was analyzed, to discover the variations between them using machine learning techniques, in order to check the detection accuracy of the method.

2. Background

The PDF format was developed by Adobe with the first version (1.0) in 1993 [16] [17]. Each new version is compatible with earlier versions, so any application viewer that can renders PDF 1.7 can open files from previous versions. Adobe added more features to the PDF format in every new feature, such as compression, encryption or forms.

PDF formats mainly consist of four parts [16] [18]:

- The header. A single line consists of %PDF- and the version number, which specifies the version of the PDF programming language: “%PDF-1.7” means the PDF files are version 1.7.
- The body. This consists of PDF objects which build most of the PDF formats. The basic format of PDF is made up of objects as a type of data. There exist eight different types of object: Boolean values, Numeric objects, Strings, Names, Arrays, Dictionaries, Streams and Null.
- Cross-reference table (xref table). This table lists all the indirect objects in the PDF formats and their locations, and is updated whenever the user updates the file.
- Trailer. This locates the cross-reference table, the end of the file through the mark “%%EOF” and certain objects, like root objects.

YARA [19] [20] is a free tool that helps in identifying and categorizing malicious files. Like any antivirus system, it scans the files based on signatures which are in YARA text or binary strings that specify a malware.

Using simple rules, YARA scans the malicious files looking for strings that exist in the rules; if they are found in the file, the rules are applied.

YARA’s rule starts with the key word rule followed by an identifier. These identifiers should follow the same identifier rules in C language and should not exceed 128 letters.

Rules are composed of two parts. The first is the string section. Each string needs an identifier that starts with $ followed by a sequence of letters; this identifier will be part of the condition section that points to the specified string. Strings are defined as both text strings, which are put between double quotes, and hexadecimal strings, which are put between curly brackets.

The condition section acts as the logic of the rule. It contains identifiers that refer to the strings in the string section. The identifiers are Boolean variables that are true if the strings are found in a file or process, or false if not.

In addition to the string and condition sections, a YARA rule may contain further information regarding any rule, which is called metadata and which is recognized in the metadata part as (meta). The metadata section includes identifiers and values separated by the equals sign. These values can be represented in three forms: integers, strings or Boolean values. Here is a simple example which shows the structure and the main components of any YARA rule:

```
rule any_Rule
{
   Meta:
     Any_Text_1 = “It is an     example”
     Any_Text_2 = 13
     Any_Text_3 = False
   strings:
     $text_string = “ASDFGHJKL"
     $hex_string = {41 53 44 46 47 48 4a 4b
4c}
   condition:
     $text_string or $hex_string
}
```

YARA can be called from a code written in Python through using the yara-python extension, which allows Python users to use the YARA functionality.

3. Related Work

Corona et al. [21] presented a system to detect malicious JavaScript embedded inside PDF files called "Lux 0n discriminant References" (Lux0R). Their approach depends on the lexical properties of the JavaScript code using the references of its API, which are functions, constants, objects, methods and keywords, as well as attributes. They utilized machine learning techniques to obtain a subset of API references which define malicious code.

Laskov and Šrndić [22] proposed a model for detecting malicious PDF files with JavaScript-related malware. They presented a tool, PJScan, which is capable of detecting malicious PDF documents.

The architecture of PJScan consists of the extraction of the JavaScript from the malicious files to obtain the script's lexical properties via the tokenizer. The output, which is the token sequence, is fed as an input to the machine learning algorithm. The learning algorithm will be acquainted with the known malicious PDF files in order to produce a model used for classification of unknown malicious files. In the second phase, every unknown malicious PDF file passes through the same stages, from the extraction of JavaScript, its tokenization and the application of the token sequence to the learning algorithm, in which the detector compares this output with a learned model to measure the deflection from a predefined threshold. So, values that are close to a learned model are considered as malicious; otherwise, they are benign.

Wepawet is a web-based service that implements a static and dynamic analysis of malware in PDF documents; it depends on JavaScript contained within it [23] [24] and utilizes JSAND to detect malicious JavaScript code based on lexical analysis.

Uploading PDF files for analysis gives a report, which contains details about the files that are flagged as malicious, such as the MD5 of the file, and provides information on whether the file is malicious, suspicious or benign and on malwares and shellcodes.

The detection results are identified based on the usage of well-known vulnerabilities to classify a file as a malicious PDF file, while suspicious files are identified based on the existence of shellcode and obfuscated JavaScript.

In 2012, Smutz and Stavrou [25] presented a framework for detecting malicious files by using machine learning. The framework depends on selecting features, in order to distinguish between benign and malicious files, with the use of a classifier which chooses features randomly for each individual classification tree to give a high detection rate.

4. System Architecture

The system architecture is shown in Figure 1. There are two scanning methods: structure scan, which depends on predefined keywords to be scanned that are available within the keywords file; and YARA scan, which requires rules to scan the PDF files with and which is available through the YARA rules file. With regard to the scanning method, the PDF files and the analysis files are read to calculate their hash values.

The system checks the hash value of every PDF file if it is available in the hash value database. If not, the system adds the hash value to this database, then performs the specified scan to add the output to the output database folder and finally displays the output to the analyst. If the hash value of the PDF file is included in database, the system checks if the PDF file has been previously analyzed with the analysis files. If it has, the system displays the output. If not, the system performs the scan using these analysis files, adds the output to the output database and then displays this to the analyst.

4.1 Structure Scan

According to Khitan et al. [15], the objective of this phase is to scan the PDF documents searching for features which are important for labeling PDF documents as suspicious, to give a brief idea about the structure of the document (like number of pages) and to list possibly suspicious objects in it. PDF files contain data represented in ASCII and binary formats, therefore the PDF documents are read as a byte sequence to easily parse these.

The authors added more suspicious keywords in addition to Didier Steven's work [26], which are important features when scanning malicious PDF files. The further set of keywords used in this phase are [15]:

- /FlateDecode
- /LZWDecode
- /RunLengthDecode
- /JBIG2Decode
- /ASCII85Decode
- /ASCIIHexDecode

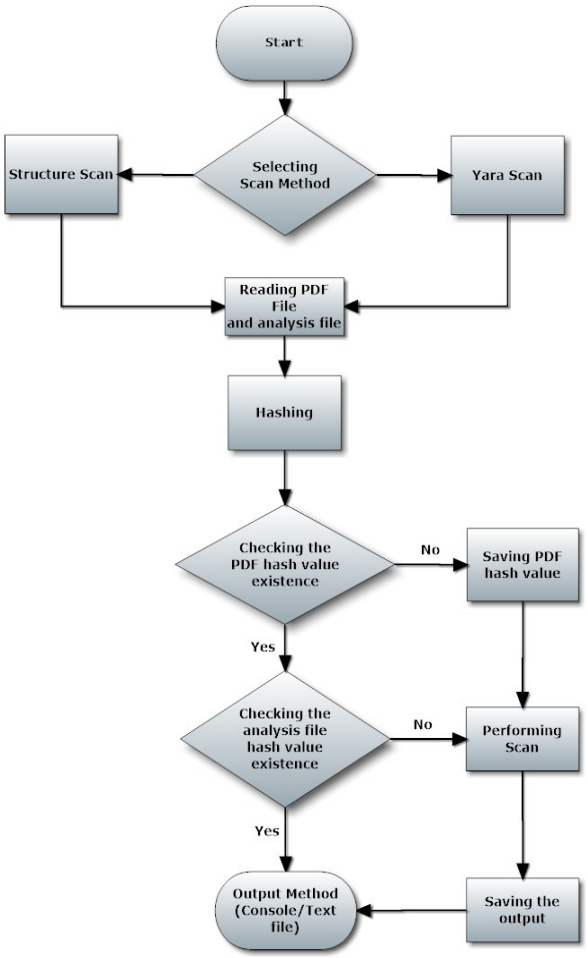

Figure 1 System Architecture

- /CCITTFaxDecode
- /DCTDecode
- /URI
- /GoTo

4.2 YARA scan

The second method for scanning PDF documents uses YARA rules to search for byte sequences and strings in order to spot malware in PDF files. The yara-python module was used to integrate YARA capabilities with the proposed system, in addition to the use of YARA editor, to create the YARA rules for the system.

These rules are used to search and identify PDF characteristics and can accordingly classify these files as suspicious or benign. Each rule consists of a set of strings and a Boolean that identifies its logic and a description for this rule.

Table 2 shows a list of YARA rules that are currently applied with the system. Analysts have the ability to craft their own rules to be used for further or future PDF analysis.

4.3 Calculating hash value

The goal of this phase is to check if the PDF documents have been analyzed already by calculating their hash value in addition to the hash value of the analysis files. As such, this step is considered worthwhile, as it can save the analyst's time if the document has been analyzed before.

4.4 Reporting

In this phase, the result of the scan is displayed to the analyst. There are two ways to view the output; either displayed on the console or copied into a text file carrying the same name of the scanned PDF.

The report contains related information about the analyzed PDF file. If the report is generated from the structure scan, the related information will include the objects defined in the keywords file, with their quantities found in the document and the result of the scan. If it is suspicious, there will be an explanation why it has been marked as suspicious.

If the report is created through the YARA scan, the information will include the YARA rules matched through the scanning process, which includes the problem, the strings to search for and conditions.

5. Experiments

The experiment was implemented using Windows 8 in a Hyper-V virtual machine. The virtual machine was configured to use one virtual processor of 1GB RAM with varied programs like Python 2.7.8, yara-python 3.0 and Microsoft Visual C++ 2010 x86 Redistributable to integrate the YARA rules The experiment was implemented using Windows 8 in a Hyper-V virtual machine. The virtual machine was configured to use one virtual processor of 1GB RAM with varied programs like Python 2.7.8, yara-python 3.0 and Microsoft Visual C++ 2010 x86 Redistributable to integrate the YARA rules within the proposed system. The Hyper-V virtual machine has been used for static analysis of PDF documents and to keep the host operating system safe from malicious datasets.

Malicious and benign PDF files were used to evaluate the proposed system; two experiments were performed using the same dataset. Both were tested with a dataset consisting of 19,593 benign and malicious PDF documents with a total size of 918 MB, downloaded from the site Contagiodump [27], a website containing up-to-date malware samples, threats and tests. Table 3 shows the properties of the dataset used in the experiment.

Table 2 YARA Rules Used

Rule Name	Description
Bad_Header : PDF	Header not within the first 1024 of the file
No_Startxref : PDF	PDF document doesn't have startxref label
Embedded_JavaScript : PDF	The PDF document contains JavaScript code
Suspicious_OpenAction : PDF	The PDF document contains Action performed automatically when a document is opened
Suspicious_OpenAction : PDF	The PDF document contains an action to be performed when the document or page is viewed
Embedded_File : PDF	The PDF document contains embedded files

The correctness of the collected dataset was checked, as the presence of malicious files in benign samples, or contrariwise, will produce negative results on the studied experiments. For that reason, a copy of all documents in the malicious as well as in the benign dataset were scanned using Kaspersky Endpoint Security 10 antivirus in a separate Hyper-V virtual machine, which confirmed that the files were classified correctly (Table 3).

5.1 Structure Scan: Experiment 1

In accordance with Khitan et al. [15], the experiment was performed to search for suspicious features and to compute their frequencies in both the malicious and benign datasets. The results are shown in Table 4, which represents the 30 features with the number of files that contain each feature in the analyzed PDF files. Percentages were calculated by dividing the number of files with a certain feature over the total number of the samples.

Table 3 PDF Documents collected for the experiment

Category	# of files	Size of files
Benign Files	8800	761 MB
Malicious Files	10793	157 MB
Total	19593	918 MB

Table 4 Structure Scan Results – Experiment 1

Features	Mali-cious	Clean	%Mali-cious	%Clean
JavaScript	2766	298	14.12%	1.52%
JS	2758	290	14.08%	1.48%
mismatched objects	58	0	0.30%	0.00%
mismatched streams	29	7	0.15%	0.04%
PDFs with no Cross reference table	647	1560	3.30%	7.96%
PDFs with no Startxref	284	0	1.45%	0.00%
FlateDecode	3067	8597	15.65%	43.88%
LZWDecode	58	359	0.30%	1.83%
ASCII85Decode	205	57	1.05%	0.29%
ASCIIHexDecode	402	408	2.05%	2.08%
RunLengthDecode	53	0	0.27%	0.00%
JBIG2Decode	3	143	0.02%	0.73%
DCTDecode	96	1672	0.49%	8.53%
Encrypt	5	58	0.03%	0.30%
CCITTFaxDecode	1	471	0.01%	2.40%
OpenAction	1762	610	8.99%	3.11%
Launch	68	12	0.35%	0.06%
AA	89	352	0.45%	1.80%
Acroform	1714	2658	8.75%	13.57%
URI	1	1241	0.01%	6.33%
RichMedia	2	0	0.01%	0.00
ObjStm	34	2924	0.17%	14.92%
EmbeddedFile	908	979	4.63%	5.00%
Page = 1	3144	3406	16.05%	17.38%
%EOF missing	6394	0	32.63%	0.00%
Bad Header	718	0	3.66%	0.00%
XFA	906	2	4.62%	0.01%
GoTo	8	485	0.04%	2.48%

Total Dataset = 19593

5.2 YARA Scan: Experiment 2

In this experiment, the PDF files were scanned using RegEx as a feature of YARA to search and identify the presence of suspicious features.

The results of the experiment on benign and malicious files are shown in Table 5, which shows the number of files that contain each feature from the analyzed PDF files.

Table 5 YARA Scan Results – Experiment 2

Features	Malici-ous	Clean	%Malici-ous	%Clean
JavaScript	2797	298	14.29%	1.52%
JS	2795	290	14.28%	1.48%
mismatched objects	92	10	0.47%	0.05%
mismatched streams	6	6	0.03%	0.03%
PDFs with no Cross reference table	285	0	1.46%	0.00%
PDFs with no Startxref	286	0	1.46%	0.00%
FlateDecode	3075	8597	15.71%	43.91%
LZWDecode	9	359	0.05%	1.83%
ASCII85Decode	126	57	0.64%	0.29%
ASCIIHexDecode	336	408	1.72%	2.08%
RunLengthDecode	5	0	0.03%	0.00%
JBIG2Decode	1	143	0.01%	0.73%
DCTDecode	96	1672	0.49%	8.54%
Encrypt	5	58	0.03%	0.30%
CCITTFaxDecode	1	471	0.01%	2.41%
OpenAction	1790	610	9.14%	3.12%
Launch	68	12	0.35%	0.06%
AA	77	101	0.39%	0.52%
Acroform	1721	2658	8.79%	13.58%
URI	1	1204	0.01%	6.15%
RichMedia	2	0	0.01%	0.00%
ObjStm	35	2924	0.18%	14.93%
EmbeddedFile	906	785	4.63%	4.01%
Page = 1	3192	3130	16.30%	15.99%
%EOF missing	6392	0	32.65%	0.00%
Bad Header	661	0	3.38%	0.00%
XFA	902	2	4.61%	0.01%
GoTo	8	485	0.04%	2.48%

Total Dataset = 19593

As can be seen from Tables 4 and 5, there are differences in certain keywords between the two scans and in the number of files that contain them. This is because the structure scan detects by matching the featured characters, compared to the YARA scan, which depends on binary strings that uniquely match the keywords.

6. Results

In the two experiments and as shown in the analysis of the results presented in Tables 4 and 5, six features were selected: Bad Header, %%EOF missing, JavaScript, JS, OpenAction and XFA. These features have the most different values between clean and malicious files and, as such, are important features to be used as indicators for suspicion.
To calculate the number of files according to the proposed hypothesis above, it must be determined how these features are presented in the files, as each PDF file may contain more than one feature. So, the relationship between these features was calculated and the PDF files were required to be rescanned.

6.1 Structure Scan Results

The relationship between the specified features in the hypothesis and how they are positioned in the dataset can be seen in Table 6 [15]. Each feature has a symbol to simplify its representation.
According to the results presented in Table 6, the predicted number of suspicious files can be calculated as:

Predicted no. of suspicious files (P.Fs) = H + E + L + R + Y + Z + V + J + O + X (1)

By applying equation (1) to the results of the malicious files listed in Table 6, the predicted number of suspicious files in the malicious dataset is:

P.Fs = 10783
By applying equation (1) to the results of the clean files listed in Table 6, the predicted number of suspicious files in the clean dataset is:

P.Fs = 932

Table 6 Features Presence in the files – Structure Scan

Features	Symbol	Frequency in Malicious	Frequency in Clean
Bad Header	H	718	0
%%EOF missing	E	6394	0
JavaScript		2766	298
JS		2758	290
OpenAction		1762	610
XFA		906	2
(JavaScript ∩ JS ∩ OpenAction) – XFA	R	1754	7
JavaScript – (JS ∪ OpenAction ∪ XFA)	Y	8	39
JS – (JavaScript ∪ OpenAction ∪ XFA)	Z	0	31
OpenAction – (JavaScript ∪ JS ∪ XFA)	V	5	603
JavaScript ∩ JS ∩ OpenAction ∩ XFA	J	3	0
(JavaScript ∩ JS ∩ XFA) – OpenAction	O	3	2
XFA – (JavaScript ∪ JS ∪ OpenAction)	X	900	0

6.2 YARA scan results

In order to find the relationship between the specified features in the hypothesis and how they are positioned in the dataset, the YARA scan was performed again in order to find the frequencies of each feature and the intersections between these features (Table 7). Each feature has a symbol to simplify its representation.
According to the results presented in Table 7, the predicted number of suspicious files can be calculated using equation (1) on the results of the malicious files listed. The predicted number of suspicious files in the malicious dataset is thus:

P.Fs = 10759
By applying equation (1) to the results of clean files listed in Table 7 the predicted number of suspicious files in the clean dataset is:

P.Fs = 932

Table 7 Features Presence in the files – YARA Scan

Features	Symbol	Frequency in Malicious	Frequency in Clean
Bad Header	H	661	0
%%EOF missing	E	6392	0
JavaScript		2797	298
JS		2795	290
OpenAction		1790	610
XFA		902	2
(JavaScript ∩ JS) – (OpenAction ∪ XFA)	L	1002	250
(JavaScript ∩ JS ∩ OpenAction) – XFA	R	1782	7
JavaScript – (JS ∪ OpenAction ∪ XFA)	Y	8	39
JS – (JavaScript ∪ OpenAction ∪ XFA)	Z	6	31
OpenAction – (JavaScript ∪ JS ∪ XFA)	V	6	603
JavaScript ∩ JS ∩ OpenAction ∩ XFA	J	2	0
(JavaScript ∩ JS ∩ XFA) – OpenAction	O	3	2
XFA – (JavaScript ∪ JS ∪ OpenAction)	X	897	0

7. Detection Accuracy

Before explaining the detection rates that emerged through the experiments, the following terms are presented [28]:

- True Positive (TP). The number of files detected as malicious from malicious samples.
- True Negative (TN). The number of files detected as benign from benign samples.
- False Positive (FP). The number of files detected as malicious from benign samples.
- False Negative (FN). The number of files classified as benign from malicious samples.

In the experiment the performance of the proposed system was assessed with regard to FP and TP rates:

$$True\ Positive\ Rate\ (TPR) = \frac{TP}{TP + FN} * 100 \quad (2)$$

$$False\ Positive\ Rate\ (FPR) = \frac{FP}{TN + FP} * 100 \quad (3)$$

The FP and TP rates of the proposed system were evaluated in terms of the presence of the six features selected in Section 6: JavaScript, JS, OpenAction, XFA keywords and the absence of both the %%EOF keyword and PDF header within the first 1,024 bytes of the file, as shown in Tables 8 and 9.

Table 8 Detection Results for Structure Scan

		Known Samples	
		Benign	**Malicious**
Detected Samples	**Benign**	TN = 7868	FN = 10
	Suspicious	FP = 932	TP = 10783

Table 9 Detection Results for YARA Scan

		Known Samples	
		Benign	**Malicious**
Detected Samples	**Benign**	TN = 7868	FN = 214
	Suspicious	FP = 932	TP = 10759

Using the results from Tables 8 and 9, a comparison in detection accuracy was made between the presented system and two public classifiers: Naïve Bayes and Decision Tree-J48.

The two classifiers were executed on the dataset to detect suspicious PDF files using the features (EOF not present, obj does not equal endobj, ObjStm, JBIG2Decode, DCTDecode, FlateDecode, XFA and URI) which were determined using GeneticSearch algorithm in the WEKA platform. For each system, the TP, FP and accuracy rates are shown in Table 10.

Table 10 Comparison between the presented system and machine learning classifiers

System		**TP (%)**	**FP (%)**	**Accuracy**
Presented System	**Structure Scan**	99.91%	10.59%	95.19%
	YARA Scan	98.05%	10.59%	94.02%
Naïve Bayes		99.4%	3.8%	97.99 %
Decision Tree-J48		99.8%	0.2%	99.81 %

Table 10 outlines the results of the comparison between the proposed system and the machine learning classifiers, where the proposed system in both methods (structure and YARA scans) displayed the highest FP rate, and presented a less accurate detection than the two classifiers. This means that both classifiers have greater classification capabilities compared to our system, which indicates a better features selection.

From the results above, and according to the features selection, our hypothesis – which selected two significant features (i.e. %%EOF missing and XFA) and four irrelevant features compared to the machine learning's features selection – gives a higher detection accuracy.

8. Performance Evaluation

To evaluate the system and how it can detect suspicious PDF files, it was compared with another tool, Wepawet, which analyzes PDF files by using an interpreter to run JavaScript [12] [13].

The comparison was conducted on 5,000 PDF files deemed as FN, which meant the dataset was known to be malicious. To carry out the comparison, the experiment was performed using the keywords which were used in the proposed hypothesis. The relationships between the keywords are shown in Tables 11 and 12; the results of the comparisons are shown in Table 13.

Table 11 Keywords relationship - Structure Scan

Features	**Frequency in Malicious**	**Frequency in Clean**
Bad Header	347	204
%%EOF missing	3010	1811
JavaScript	1236	748
JS	1232	746
OpenAction	755	449
XFA	404	235
(JavaScript ∩ JS) – (OpenAction ∪ XFA)	478	295
JavaScript ∩ JS ∩ OpenAction ∩ XFA	2	0
(JavaScript ∩ JS ∩ OpenAction) – XFA	749	448
(JavaScript ∩ JS ∩ XFA) – OpenAction	3	3
JavaScript – (JS ∪ OpenAction ∪ XFA)	4	2
JS – (JavaScript ∪ OpenAction ∪ XFA)	0	0
OpenAction – (JavaScript ∪ JS ∪ XFA)	4	1
XFA – (JavaScript ∪ JS ∪ OpenAction)	399	232

Table 12 Keywords relationship - YARA Scan

Features	**Frequency in Malicious**	**Frequency in Clean**
Bad Header	312	182
%%EOF missing	3009	1810
JavaScript	1260	764
JS	1258	763
OpenAction	778	463
XFA	402	235
(JavaScript ∩ JS) – (OpenAction ∪ XFA)	481	297
JavaScript ∩ JS ∩ OpenAction ∩ XFA	1	0
(JavaScript ∩ JS ∩ OpenAction) – XFA	771	462
(JavaScript ∩ JS ∩ XFA) – OpenAction	3	3
JavaScript – (JS ∪ OpenAction ∪ XFA)	4	2
JS – (JavaScript ∪ OpenAction ∪ XFA)	2	1
OpenAction – (JavaScript ∪ JS ∪	4	1

XFA)		
XFA – (JavaScript ∪ JS ∪ OpenAction)	398	232

Table 13 Comparison with Wepawet

		Detected Suspicious	Detected Benign	False Negative (FN)
Wetawet	4859	4693	166	3.41%
Structure Scan – Proposed method	5000	4996	4	0.08%
YARA Scan – Proposed system	5000	4985	15	0. 3%

The three scanning methods (Wepawet, YARA and structure scans) were each performed on the 5,000 malicious files. When Wepawet was used, the analysis was successful for 4,859 files (Table 13).

An analysis of the results shows that Wepawet did not have the ability to evaluate all the samples. Specifically, it missed 3.41% of the known malicious PDF files. It is believed there were some analysis problems which influenced the system, since it did not completely execute all the specifications of the PDF files and was only implemented on JavaScripts and executables. It can be observed from Table 13 that the proposed system outperformed Wepawet in terms of the FN rate, which was 0.3% using the YARA scan and 0.08% using the structure scan.

From the method evaluation of Wepawet, it can be seen that the results support the proposed hypothesis of selecting six features from 30 to be used as significant features in detecting suspicious PDF files; these have a positive result on the performance of the classification system.

9. Conclusion

While PDF documents are used by many users as a stable and reliable document exchange technique format, it is also highly used by hackers to run harmful code on computers. This is because the PDF structure provides the ability to embed codes like JavaScript and to communicate with outside sites.

In this paper, the structural format of the PDF has been studied. The research also dealt with the techniques used by hackers to keep their harmful codes hidden from security specialists and security software like antiviruses.

The system that implements static detection has been presented to detect suspicious PDF documents based on the presence of the most significant features that are commonly found in malicious files. As an additional step, an experiment was conducted in order to classify the PDF documents based on these keywords, as they are found in most of the malicious PDF files.

It can be noticed from the results that features selection according to the presented system gave a high rate for detecting suspicious PDF files.

Applying WEKA to check the detection accuracy of the presented system for extracting features, by running two algorithms to detect suspicious PDF files, gave a higher detection rate than the presented system and has two mutual features: XFA and missing %%EOF.

10. Future Work

Future work will focus on utilizing other data mining algorithms and testing them, in addition to combining static and dynamic analysis to extract JavaScript and to detect malicious PDF files whose exploitation techniques do not rely on features embedded within them.

11. Limitations

A limitation of the system presented in this paper is that it is unable to differentiate between malicious and benign PDFs because it detects suspicious PDF files according to the existence of certain features. Therefore, our method cannot detect any malicious PDF files that do not use these features as an attack vector.

References

[1] Adobe, "PDF Reference and Adobe Extensions to the PDF Specification," 2008. [Online]. Available: www.adobe.com/devnet/pdf/pdf_reference.html. [Accessed 23 January 2015].

[2] Sood and R. Enbody, "Targeted Cyberattacks: A Superset of Advanced Persistent Threats," Security & Privacy, IEEE, vol. 11, no. 1, pp. 54 - 61, 2012. doi: 10.1109/MSP.2012.90.

[3] Beuhring and K. Salous, "Beyond Blacklisting: Cyberdefense in the Era of Advanced Persistent Threats," Security & Privacy, IEEE, vol. 12, no. 5, pp. 90 - 93 , 2014. doi: 10.1109/MSP.2014.86.

[4] M. Egele, P. Wurzinger, C. Kruegel and E. Kirda, "Defending browsers against drive-by downloads:Mitigating Heap-Spraying Code Injection Attacks," Springer, pp. 88-106, 2009. doi: 10.1007/978-3-642-02918-9_6.

[5] Blonce, E. Filiol and L. Frayssignes, "Portable Document Format (PDF) Security Analysis and Malware Threats," Proceedings of the 10th European Conference on Information Warfare and Security: ECIW, 2011.

[6] F. Gutierrez and K. Selvaraj, "The Rise of PDF Malware," Symantec, 2010.

[7] D. Stevens, "Anatomy of Malicious PDF Documents," IEEE, 2010.

[8] NIST, "National Vulnerability Database," 2011. [Online]. Available: https://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2011-0611. [Accessed 23 January 2015].

[9] D. Maiorca, G. Giacinto and I. Corona, "A Pattern Recognition System for Malicious PDF Files Detection," Springer, vol. 7376 , pp. 510-524, 2012. doi: 10.1007/978-3-642-31537-4_40.
[10] M. Polychronakis, K. Anagnostakis and E. Markatos, "Comprehensive shellcode detection using runtime heuristics," ACM, pp. 287-296, 2010. doi: 10.1145/1920261.1920305.
Bazzi and Y. Onozato, "IDS for detecting malicious non-executable files using dynamic analysis," IEEE, pp. 1 - 3 , 2013.
[11] F. Schmitt, J. Gassen and E. Gerhards-Padilla, "PDF Scrutinizer: Detecting JavaScript-based attacks in PDF documents," IEEE, pp. 104-111, 2012. Doi: 10.1109/PST.2012.6297926.
[12] T. Dube, R. Raines, M. Grimaila, K. Bauer and S. Rogers, "Malware Target Recognition of Unknown Threats," Systems Journal, IEEE, vol. 7, no. 3, pp. 467 - 477 , 2012. Doi: 10.1109/JSYST.2012.2221913,
[13] Han, B. Chul, H. Geun and K. Sohn, "Toward extracting malware features for classification using static and dynamic analysis," IEEE, pp. 126 - 129 , 2012.
[14] S. Khitan, A. Hadi and J. Atoum, "Enhanced Analysis Method for Suspicious PDF Files," IJCSNS International Journal of Computer Science and Network Security, vol. 15, no. 5, pp. 32-38, 2015.
[15] Adobe, "Portable document format reference manual," Addison-Wesley, 1993.
[16] L. Leurs, "The history of PDF," 9 August 2013. [Online]. Available: http://www.prepressure.com/pdf/basics/history. [Accessed 23 January 2016].
[17] "PDF file structure," [Online]. Available: http://www.simpopdf.com/resource/pdf-file-structure. html..
[18] V. Álvarez, "YARA User's Manual," 2014.
[19] "YARA in a nutshell," [Online]. Available: https://plusvic.github.io/yara/. [Accessed 24 January 2016].
[20] Corona, D. Maiorca, D. Ariu and G. Giacinto, "Lux0R: Detection of Malicious PDF-embedded JavaScript code through Discriminant Analysis of API References," in AISec'14: Proceedings of the 2014 ACM Workshop on Artificial Intelligence and Security, co-located with CCS '14 - See more at: https://pralab.diee.unica.it/en/node/1116#sthash.JBkKevvW.dpuf, 2014.
[21] P. Laskov and N. Šrndić, "Static Detection of Malicious JavaScript-Bearing PDF Documents," in In Annual Computer Security Applications Conference, 2011. doi: 10.1145/2076732.2076785.
[22] M. Cova, C. Kruegel and G. Vigna, "Detection and analysis of drive-by-download attacks and malicious javascript code," in In Proceedings of the 19th international conference on World wide web, 2010. doi: 10.1145/1772690.1772720.
[23] S. Ford, M. Cova, C. Kruegel and a. G. Vigna, "Wepawet," 2008. [Online]. Available: http://wepawet.cs.ucsb.edu. [Accessed 21 Januray 2016].
[24] Smutz and A. Stavrou, "Malicious PDF Detection using Metadata and Structural Features," ACM, pp. 239-248, 2012. doi: 10.1145/2420950.2420987.
[25] Stevens, "Didier Stevens," 2010. [Online]. Available: http://blog.didierstevens.com/2010/03/29/escape-from-pdf. [Accessed 21 January 2016].
[26] Mila, "CVE-2013-0640 samples listing," 24 April 2013. [Online]. Available: http://contagiodump.blogspot.com. [Accessed 21 January 2016].
[27] S. Salem, R. Darwish and S. Sayed, "A Real-Time Approach for Detecting Malicious Executables," Springer, pp. 355-364, 2014. doi: 10.1007/978-3-319-01857-7_34.