Malware Research in PDF Files

A multidisciplinary approach in the identification of malicious PDF files.

Shir Bentabou **Alexey Titov**

Advisor: Ph.D. Amit Dvir and Ph.D. Ran Dubin

Department of Computer Science
Ariel University
Israel

04.09.2019

Malware Research in PDF Files

A multidisciplinary approach in the identification of malicious PDF files.

Shir Bentabou   Alexey Titov

Advisors: Ph.D. Amit Dvir and Ph.D. Ran Dubin

Ariel University,Department of Computer Sciences,40700 Ariel, Israel

Abstract

Cyber is a prefix used in a growing number of terms that describe new actions that are being made possible by the usage of computers and networks. The main terms in those are cyber crime, cyber attack and cyber warfare, all of those can be carried out by malwares.

Malware, or malicious software, is any software intentionally designed to invade, cause damage, or disable computers, mobile devices, server, client,or computer network. Malware does the damage after it is implanted or introduced in some way into a target's computer. Nowadays there are many distribution strategies for malwares and many programs are used as platforms. Some of these programs are in the user's everyday use, and seem pretty innocent. In our project we will focus on PDF files as a platform for malware distribution.

PDF, Portable Document Format, is used for over 20 years worldwide, and has become one of the leading standards for the dissemination of textual documents. A typical user uses this format due to its flexibility and functionality,but it also attracts hackers who exploit various types of vulnerabilities available in this format, causing PDF to be one of the leading vectors of malicious code distribution.

Users normally open PDF files because they have confidence in this format, and thus allow malwares to run due to vulnerabilities found in the readers. Therefore, many threat analysis platforms are trying to identify the main functions that characterize the behavior of malicious PDF files by analyzing their contents, in order to learn how to automatically recognize old and new attacks.

The target of our work is to test and analyze how the combination of three different approaches and the use of machine learning methods can lead to effective recognition of malwares in PDF documents.

Contents

1 Introduction

Cyber is a prefix used in a growing number of terms that describe new actions that are being made possible by the usage of computers and networks. As the evolution of modern computers and networks progressed, a cat-and-mouse game evolved simultaneously, and continues to this day. This cat-and-mouse game is cyber warfare, and is caused by the challenges in information security, as a result of various cyber threats that exist.

The first known cyber-attack is the Morris worm, when a student in Cornell university wanted to know how many devices existed in the internet. He wrote a program that passed between computers and this program asked each device it reached to send a signal to a control server that counted the signals sent to him. Nowadays the growth in number of cyber-attacks is unimaginable; according to published data by Check Point Software Technologies, there were 23,208,628 attacks in February 23rd, 2019 alone.

The growth in cyber-attacks is not only in the numeric value, but also in the different kinds of attacks that exist. There are many different types of threats, and usually they consist of one or more kinds of attacks in the following list:

Advanced Persistent Threats	DDoS	Intellectual Property Theft	Rogue Software
Phishing	Wiper Attacks	Spyware/Malware	Unpatched Software
Trojans	Money Theft	MITM	
Botnets	Data Manipulation	Drive-By Downloads	
Ransomware	Data Destruction	Malvertising	

Table 1: List including different kinds of cyber-attacks.

Phishing is one of the popular distribution methods for malware. It is the fraudulent attempt to obtain sensitive information from a target by disguising as a trustworthy entity in electronic communication. Phishing is typically carried out by e-mail spoofing, and often directs users to enter personal information at a fake website or by sending fake but credible documents, often PDFs.

Threats as these come from various sources. The profile of the attackers does not match one certain type and depends on the source's interests and available technology. The most common sources of cyber threats are: nation states or national governments, terrorists, industrial spies, organized crime groups, hacktivists and hackers, business competitors, and disgruntled insiders.

Malware, or malicious software, is any software designed to serve any kind of cyber-attack. Software is considered malware based on the intent of the creator rather than its actual features. Different kinds of malware serve for different uses, and it does the damage after it is implanted or introduced in some way into a target's computer.

The first recorded malware was named Elk Cloner, created by 15-year-old high school student Rich Skrenta as a prank, and affected Apple II systems in 1982. This virus was disseminated by infected floppy disks and spread to all the disks that were attached to the system by attaching itself to the OS.

What started as a teenage prank in 1982, has evolved into a wide range of variated software that is used today as malwares. The main types of malwares that exist today are:

- Trojan Horse – This type of malware infects a computer and usually runs in the background, sometimes for long periods of time, and gains unauthorized access to the affected computer, gathers information about the user / machine it is installed on. The information gathered by the trojan is then sent to the attacker, normally a server side that stores the data for the attacker.
- Virus - A virus is software usually hidden within another program that can produce copies of itself and insert them into other programs or files, and usually performs a harmful action.
- Worm - Similar to viruses, worms self-replicate in order to spread to other computers over a network, usually causing harm by destroying data and files.
- Spyware - Malware that secretly observes the computer user's activities without permission and reports it to the software's author.
- Exploits - Malware that takes advantage of bugs and vulnerabilities in a system in order to allow the exploit's creator to take control.
- Ransomware - Malware that locks you out of your device and/or encrypts your files, then forces you to pay a ransom to get them back.

And other forms…

Writing malware of the kinds we have seen above alone is not enough to create a cyber-attack. The malware must reach the target's system and operate on it in order to carry out an attack, and for that there are many distribution methods, some of them are explained below:

- Social Engineering - Socially engineered attacks exploit weaknesses of humans rather than weaknesses of software. Users are manipulated into running malicious binaries believing it is safe.
- E-Mail – E-Mail attacks can exploit vulnerabilities in the e-mail software or in the libraries that the e-mail software uses. Moreover, viruses and trojans are often disguised as innocent e-mail attachments in phishing e-mails.

- Network Intrusion - Network intrusion attacks are initiated by the attacker. The attacker finds some vulnerability in the network and takes advantage of it to infect some system in the network.
- Links - Links typically lead to malicious sites that download malware to the victim's device when they load the page.
- Infected Storage Devices - Storage devices can be used as a distribution method when they are infected with malware, and when plugged into a device, they can transfer their contents to the device and infect it.
- Drive-by Downloads – Relates to the unintended download of malware from the internet, either without the user knowing, or with their authorization without understanding the consequences.

All the above methods intend to distribute malware to systems, normally without the users being aware that the process has happened at all. In order to do that, the platforms used in these methods are well known files and programs in the user's daily use. A very popular file type for distributing malware is PDF, Portable Document Format. In our project we will focus on PDF files as a platform for malware distribution.

PDF, Portable Document Format, is a file format used for over 20 years worldwide, and has become one of the leading standards for the dissemination of textual documents. Based on the PostScript language, each PDF file encapsulates a complete description of a fixed-layout flat document, including the text, fonts, vector graphics, raster images and other information needed to display it. PDF files are composed by a set of sections:

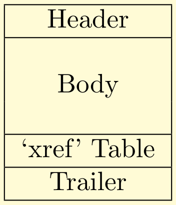


Figure 1: The structure of a PDF file.

1. **PDF Header** - This is the first line of a PDF file and it specifies the version number of the used PDF specification which the document uses (e.g. "%PDF-1.7").
2. **PDF Body** - The body of the PDF document is composed of objects that typically include text streams, fonts, images, multimedia elements, etc. The body section is used to hold all the document's data that is shown to the user. Notice that streams are interesting to us in the security aspect because they can store a large amount of data and thus store executable code that runs after some event.

3. **Cross-Reference Table** – Also called 'xref' table, this table contains the references to all the objects in the document. The purpose of a cross reference table is that it allows random access to objects in the document, so we don't need to read the whole PDF document to locate an object. Each object is represented by one entry in the table, which is always 20 bytes long. If the document changes, the table is updated automatically.

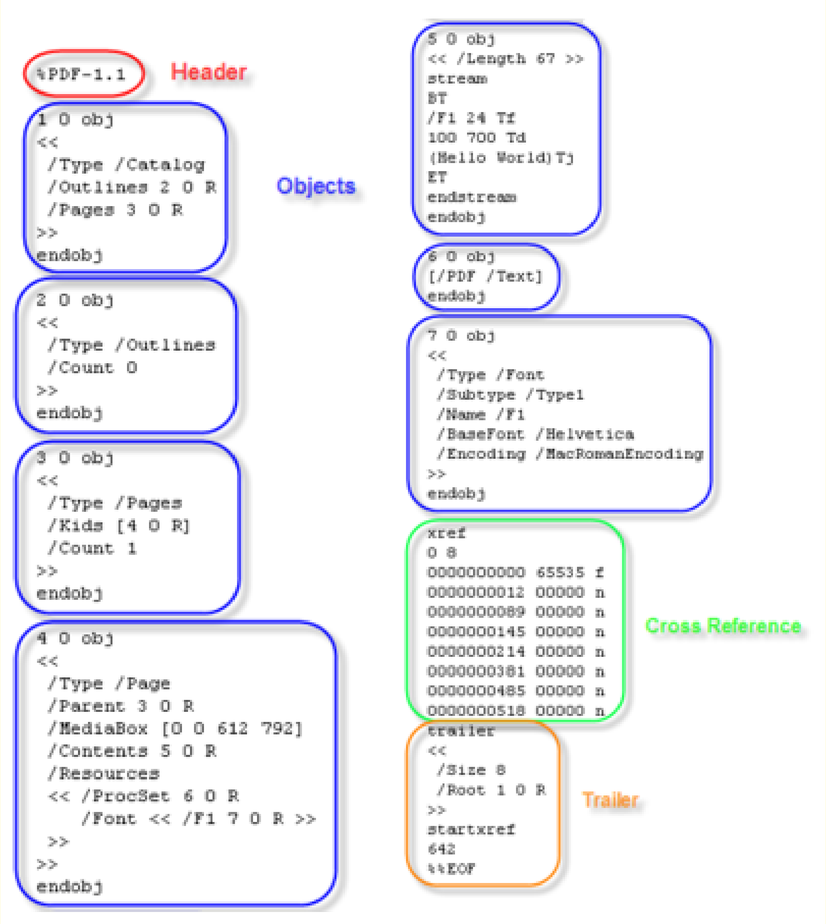
4. **Trailer** – This section specifies how the application that reads the PDF document should find the cross-reference table and other special objects in the document. The trailer section also contains the EOF indicator.

Following is a simple example of a PDF file [3]. This example gives us a notion of how the PDF files look like before they are parsed. These contents can be seen for every PDF file, by opening it with any kind of text editor. In more complex files, it is possible to see different kinds of objects in the body section. Every object resides between /Obj – /Endobj tags and contains different kinds of data. For example, streams can contain large amounts of data (even executable code) and are normally compressed. Due to that they are not readable without using some tool to decompress the data.

PDF is used widely around the world since it's creation, and still is because of two main advantages that it provides compared to other file formats: (1) **PDF files are compatible across multiple platforms** - A PDF reader presents a document independently of the hardware, operating system and application software used to create the original PDF file. The PDF format was designed to create transferable documents that can be shared across multiple computer platforms. (2) **The software for viewing PDF files is free** - Most PDF Readers, including Adobe Reader, are free to the public. This ensures that anyone you send the file to will be able to see the full version of your document.

A typical user uses this format due to its flexibility and functionality, but it also attracts hackers from the same reasons: it enables cross platform attacks and is widely used (including specific targets for attacks) because many readers are free. Moreover, there are various types of vulnerabilities available in this format, causing PDF to be one of the leading vectors of malicious code distribution. The vulnerabilities available in PDF derive from PDF's support of various types of data in addition to text such as JavaScript, Flash, media files, interactive forms or links to external files and URLs. Moreover, a lot of the PDF's content (JS, URLs, etc.) may be invisible to the user opening it.

In addition to that, PDF files are believed to be less suspicious than executable files. It is a common security practice for an IT administrator to define a policy to block executable files from staff e-mail attachments or web downloads, but it is rare to block PDF documents in such a manner. Users normally open PDF files because they have confidence in this format, and thus allow malwares to run due to vulnerabilities found in the readers. Therefore, many threat analysis platforms are trying to identify the main functions that characterize the identity and behavior of malicious PDF files by analyzing their contents, in order to learn how to automatically recognize old and new attacks.



```
%PDF-1.1
1 0 obj
<<
 /Type /Catalog
 /Outlines 2 0 R
 /Pages 3 0 R
>>
endobj
2 0 obj
<<
 /Type /Outlines
 /Count 0
>>
endobj
3 0 obj
<<
 /Type /Pages
 /Kids [4 0 R]
 /Count 1
>>
endobj
4 0 obj
<<
 /Type /Page
 /Parent 3 0 R
 /MediaBox [0 0 612 792]
 /Contents 5 0 R
 /Resources
 << /ProcSet 6 0 R
    /Font << /F1 7 0 R >>
 >>
>>
endobj
5 0 obj
<< /Length 67 >>
stream
BT
/F1 24 Tf
100 700 Td
(Hello World)Tj
ET
endstream
endobj
6 0 obj
[/PDF /Text]
endobj
7 0 obj
<<
 /Type /Font
 /Subtype /Type1
 /Name /F1
 /BaseFont /Helvetica
 /Encoding /MacRomanEncoding
>>
endobj
xref
0 8
0000000000 65535 f
0000000012 00000 n
0000000089 00000 n
0000000145 00000 n
0000000214 00000 n
0000000381 00000 n
0000000485 00000 n
0000000518 00000 n
trailer
<<
 /Size 8
 /Root 1 0 R
>>
startxref
642
%%EOF
```

Figure 2: PDF file ‘Hello World’ example[3].

2 Related Work

In this project we will focus on phishing carried out through PDF files, that means making the user download some file believing it is safe or entering an unsecure website not knowingly. This can be done easily using PDF because PDF files allow hyperlinks to be embedded in them for easy use, and also enables the use of JavaScript and PDF object streams in the body part of the file, and this way code can be executed without the user knowing, or with his knowledge but without the awareness that this could be unsafe for him. URLs are a main method for this kind of phishing, as the user can click or hover over a hyperlink in a PDF file and be directed to a malicious website or to download a malicious file. Moreover, because of the JavaScript and streams that can be part of the PDF file, this can happen without the user knowing, in the background [13] [7].

In general, there are many tools available in the software market for the classification of files and URLs as malicious or benign. Anti-virus products normally work in a static way, producing signatures for the identification of malicious files/URLs. In addition to that, dictionaries are in use in a remote server. Signatures are kept in these dictionaries in order to identify malicious files/URLs, and are frequently updated. Most of the AVs also use blacklists, these lists contain URLs, IPs and more, and every time the AVs identify a communication with some address in the blacklists, they will block it. Apart from the static tools there are dynamic analysis tools using sandbox environments that examine the behavior of a file/URL in a separated environment [1].

In previous works, we have seen projects that were meant to identify malicious PDF files, and in surveys we have read [6] [9], there have been various attempts to identify malicious files by using a range of features of the PDF file (notice that features are also tags used in the objects of the file). These features were combined to create a feature vector for the files and provided as input to machine learning algorithms (from variated kinds) that classified the files benign or malicious [2] [13]. Most of these surveys based their work on Didier Stevens [3] and Otsubo's [12] research and tools. The objective of these researches was to create algorithms that identify malicious files with low FP (False Positive) and FN (False Negative) rates and classify files efficiently with the shortest time and resources needed.

In Torres and De Los Santos research [2], they have combined four different PDF analysis tools, different sets of features (based on 21 features Didier Stevens chose in his research [3]), and three machine learning algorithms (Support Vector Machine, Random Forest, Multilayer Perceptron) to find the most efficient way to classify if a PDF is malicious or benign, using machine learning. The achieved results are shown in the table below. In their research, MLP algorithm showed the best results.

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

Table 2: Table of Torres and De Los Santos [2] test results.

In another aspect of our work, we have read articles and researches about URL analysis, and how to classify if a URL is malicious or benign. From what we have read we have seen that not only static analysis tools and blacklists were used, but also a lexicographic approach was used in most of them, to improve the success rates of classifying URLs.

In D. R. Patil and J. B. Patil's research [1], they have provided an effective hybrid methodology to classify URLs. They have used supervised decision tree learning classification models and performed the experiments on a balanced dataset. The experimental results show that, by including new features, the decision tree learning classifiers worked well on the dataset, achieving 98%-99% detection accuracy with very low FP and FN rates. Moreover, using the majority voting technique, the experiments achieved 99.29% detection accuracy with very low FP and FN rates, which is better than the existing anti-virus and anti-malware solutions for URLs.

In our project we will create an ensemble machine that will focus on three different areas of a PDF file. These three areas are: 1) The image of the first page of the PDF file, 2) The text of the first page of the PDF file, and 3) The features of the PDF file. Although the third area has already been quite researched, the two first areas have not yet been researched, therefore specific researches about it haven't been found. This means we will enter a new field of research, hoping to reach some progress in finding an efficient and effective way to identify these kinds of attacks.

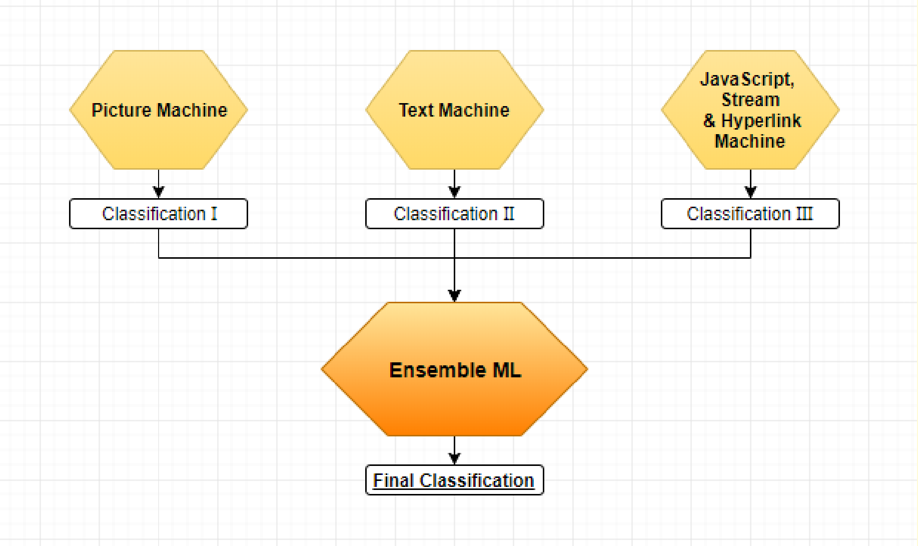


Figure 3: our schema for project

3 PDF Based Attack Techniques [5]

It is known that there are many types of attacks that are based on PDF files. This is a subject that has been researched thoroughly in the past decade. Using a PDF file as an attack vector can be very simple. As mentioned above, AVs do not always have the best solution to deal with malicious PDF files, as they base their signatures on a file's MD5 or hash, and even a small change in the file can change the MD5 or hash, and the vulnerabilities can continue to be used.

There are many ways that PDF files can be used as an attack vector. Readers have many vulnerabilities that ease the use of PDF files, and many of the characteristics of the PDF file make it a great attack vector. In this section we will present the existing approaches of utilizing PDF files in order to conduct an attack [5]. Social engineering takes a big part of most PDF based attacks, as the user-visible content of the PDF file can exist only for social engineering causes, while the not-user-visible content of the file can be extremely malicious.

JavaScript code attacks: PDF files can contain JavaScript code for legitimate purposes, for example multimedia content and form validation. The main indicator for JavaScript code embedded in a PDF file is the presence of the '/JS' tag [3] [7] [13] [14]. Normally, the goal of malicious JavaScript in a PDF file is to exploit a vulnerability in the PDF reader in order to execute embedded malicious JavaScript code. Downloading an executable file can also be carried out using JavaScript. Alternatively, JavaScript code can also open a malicious website that can perform a variety of malicious operations.

JavaScript code obfuscation is legitimately used to prevent reverse engineering for copyright purposes. However, it can also be used by attackers to conceal malicious JavaScript code, prevent it from being recognized by signature based or lexical analysis tools [7], and to reduce readability by a human security analyst. Data in the streams of a PDF can be compressed and this way hide malicious JavaScript code in them.

Embedded file attack: A PDF file can contain other file types in it, including HTML, JavaScript, executables, Microsoft Office files, and even additional PDF files. An attacker can use this functionality in order to embed a malicious file inside a benign file. This way, the attacker can utilize the vulnerabilities of other file types in order to perform malicious activity. The embedded file can be opened when the PDF file is opened using embedded JavaScript code or by other techniques such as PDF tags (such as '/Launch'). Usually, embedded malicious files are obfuscated in order to avoid detection. Adobe Reader PDF viewer versions 9.3.3 and above restrict file formats that can be opened, using a blacklist which is based on file extension.

Form submission and URL / URI attacks: Adobe Reader supports the option of submitting the PDF form from a client to a specific server using the '/SubmitForm' command. Adobe then generates a file from a PDF in order to send the data to a specified URL. If the URL belongs to a remote web server, it is able to respond. An attack can be performed by a simple request to a malicious website that will automatically open on the web browser, and the malicious website can exploit a vulnerability in the user's browser. Security mechanisms such as the protected mode of Adobe Reader can be disabled easily. Moreover, a URI address can be used to refer to any file type located remotely (both executable and non-executable files).

4 Our Work

The attacks mentioned in the previous section can be visible to the user and require some action from the user, or be completely invisible and happen in the background without the user knowing. In our project we will focus on three ways to identify malicious attacks in PDF files:

1. Preview of the files – Anti-viruses work by creating hashes (such as MD5) for malicious files found. For every malicious file they detect, they create a hash for it, and store it in their databases, so that if they encounter these files again, they will be able to block or warn about them. The problem of this approach is that if a single attribute of the file is changed, the hash also changes, and this way a file can be only briefly changed and pass the AVs detection. PDF files can be opened for initial preview, without opening the file itself. The initial preview shows the first page of the PDF file. Please note that 'previewing' the file is not proven to safe.
   As AVs check the files by hash that can be easily changed, we want to create a detection based on the content of the file. That means, we want to be able to extract previews for the PDF file's first page (in the form of images), and by the content of the files be able to detect malicious files. Our aim is to create an efficient image similarity engine, to detect files by their image.

2. Text detection – Text detection, similar to the image detection explained above can also help detect malicious files by their content. Normally, malicious PDF files contain rather innocent content in order to be credible to the user, convincing the user to open it. Therefore, we can learn about the characteristics of malicious PDF file from the text inside them, the same way as spam filters for e-mails work.

3. PDF tags and features – These are the structural tags and features of a PDF file that can give us a lot of information about the file. Normally they are invisible to the simple user that uses a reader in order to view the PDF files. These tags contain all the PDF's content and can give us information about the JavaScript code inside the file, links and URLs, obfuscated code, structure of the PDF file and more.

All the samples we have used in our work were received from our advisor, Ph.D. Ran Dubin, CEO & Co Founder at SNDBOX. SNDBOX is an artificial intelligence malware research platform. The dataset contained 9,557 samples overall, and was made up of 9,158 benign samples and 419 malicious samples.

5 Existing Tools

During our work we have found various existing tools that helped us with our research. Many of them were found while searching for specific solutions to problems we have encountered along the way. In this section we will explain about all the tools we have used.

1. PDFiD – This tool was developed by Didier Stevens [3], and was meant to help differentiating between malicious and benign PDF files. This tool is a simple string scanner written in python. It scans a PDF file for specific tags, to check if they are included in the file, and returns the expressions and the number of times they have been found in the PDF file.

2. JAST – This tool was developed as part of a research by Fass et al [14], in order to detect malicious JavaScript instances. This solution combines the extraction of features from the code's abstract syntax tree with a random forest classifier. It is based on a frequency analysis of specific patterns, which are either predictive of benign or malicious samples. The analysis made by this tool is entirely static, and yields a high detection accuracy of almost 99.5%. This tool also supplies a simple classification of JS code – if there is JS code in a (text) file or not, and if there is – if it is obfuscated or not.

3. AnalyzePDF – This tool analyzes PDF files by looking at their characteristics in order to add some intelligence about the file's nature – if it is malicious or benign. This tool has a module that calculates the entropy in a PDF file. It calculates the overall entropy, entropy within the streams of the PDF file, and the entropy out of the PDF file's streams [15].

4. PeePDF - PeePDF is a python tool, that was made to explore PDF files in order to find out if the file can be harmful or not. This tool aims to provide the researcher with all necessary components in PDF analysis. This tool can extract all the objects in a PDF that contain suspicious elements (such as JS code) and supports object streams (that are compressed) and encrypted files. This tool can easily extract all the JavaScript from a PDF file [16].

6 First phase: Preparations

As mentioned before, the overall idea of the project was to create three machines, each one will be a classifier on its own. Each machine will classify a PDF file as malicious or benign by some kind of information about the file. The first machine will classify the file based on the image of the first page of the file. The second machine will classify the file based on the text from the first page of the file. The third machine will classify the file based on features and metadata of the file. The fourth machine, is an ensemble machine, and will classify a PDF file based on the results of all three previous machines we described.

In order to create the machines, we needed to do a few preparations first. That means, prepare the code that will provide the machines the information they need from a PDF file. The machines will use this information in order to classify the files.

As explained above, in this phase we extracted the information we need for our machines from PDF files. Firstly, we defined what information was needed for every machine, and then we wrote code that extracts each type of information from a PDF file.

First Machine: Image Classifier	**Second Machine: Text Classifier**	**Third Machine: Feature Classifier**
Extract preview of PDF file	Extract text from PDF file	Extract telemetry
	Extract text from image	Extract URLs
		Extract JavaScript

In order to create a machine that classifies the PDF file by image, we have to extract an image of the PDF file. In this stage of the project we have decided to extract the preview of the first page of the file. For that, there are libraries in python that are made specifically for the processing of PDF files, and that eased our work. We have used ‘pdf2image’ to converts a PDF to a ‘PIL’ image object, and ‘PIL’ to add image processing capabilities to our code.

In order to create a machine that classifies the PDF file by text, we have to extract text from a PDF file. In this stage of the project we decided to extract the text from the first page of the file. In order to do that we used ‘PDFMiner’ and ‘pyPDF2’, tools that extract information from PDF files, including text from the file. With that said, if the PDF file holds an image in the first page (or the only page), we will have to extract text from an image, and that is another case we dealt with. For that we used ‘pytesseract’, an optical character recognition (OCR) tool for python. It recognizes and reads text embedded in images. Furthermore, we used ‘cv2’ in order to return an image of a specific page in the file.

In order to create a machine that classifies the PDF file by its features, we have to find a way to extract all the features we are interested in from the PDF file. For that, there are many tools that already exist and we used them to simplify our work. For the extraction of the features we are interested in we used a number of tools: ‘PDFiD’ in order to research and choose the features we want, ‘PeePDF’ in order to extract JS and information for the JS in the file and ‘pyPDF2’ in order to extract URLs from the PDF file.

7 Second phase: Image Based Classification Machine

In this phase, we wanted to create an image-based classification machine. This machine should classify a PDF file only by the preview of the file. The idea of classifying a PDF file by image hasn't appeared in any research we have read, thus making our work a first in some sort of way.

7.1 Creating the vector

When we started researching for the classification of a PDF file by image and saw that no research has been made about it, we looked at much simpler examples of image classification. According to an article by Adrian Rosebrock [17], the best strategy presented to work with a picture is using a histogram, and not working with the pixels of the picture. The histogram made much more sense in our case, since relating to the picture by pixels did not give us information we could work with.

Whilst working on this machine, following a tip from our advisor, we tried to find meaningful characteristics that will indicate if a PDF is malicious or not. While searching, we found a characteristic we have seen in many files, and chose to refer to the blurriness of the image. We have seen many malicious samples that contained blurred images, with a message to the user that they should download some update in order to see the document. In order to calculate the blurriness of the image, we used the Laplacian method with an existing library in python ('cv2'). This method returns a numeric value – if that value is under 150, that means that the image is blurry. Figure 4 shows an example of a blurry PDF document and a non-blurry PDF document and their numeric values calculated by the Laplacian method calculation (7.02 opposed to 4837.3).

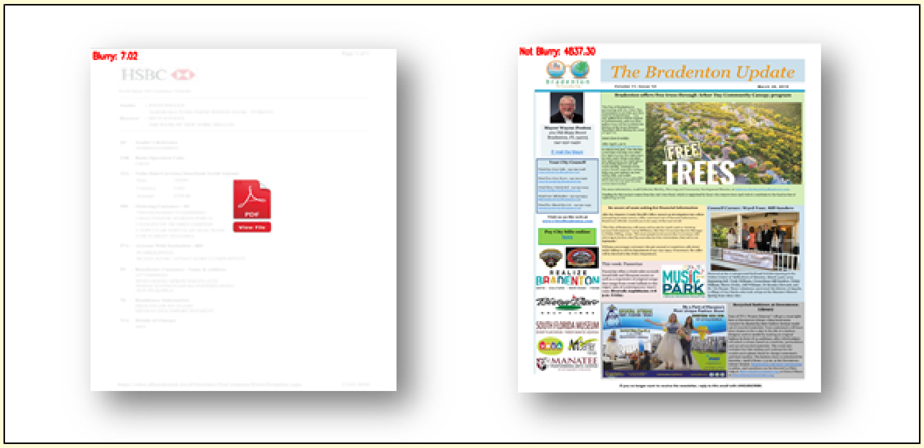


Figure 4: *Left side*: Blurry PDF document, *Right side*: Non-blurry PDF document

In this stage, we had all the information we needed to construct the vector for the images, that the machine will receive as it's input. We decided that the overall size of the image vector is 513. The 512 first indexes relate to the picture's histogram, in other words, to the colors in the image (RGB – 8*8*8

= 512). The last index of the vector holds the numeric value that is returned from the Laplacian method calculation of the blurriness of the image.

7.2 Implementing the machine

When we reached the phase of implementing the image-based classification machine, our advisors had recommended us to use K-Means-Clustering, a popular method for cluster analysis. K-Means-Clustering partitions its observations to a numeric k clusters, each observation belongs to the cluster with the nearest mean.

We have decided to implement the machine using KNN as another example, in order to have something to compare KMC with, and in order to show that what the advisors recommended was a better solution. In more advanced phases of the project we will see that KMC is indeed the better solution.

We chose KNN because it is a simple example of a machine learning algorithm that classifies the samples to one of two groups: benign or malicious. We wanted to check if a simple 'yes-no' classifier will give better results than a clustering algorithm we were recommended to implement. The logic of the clustering preference is that based solely on the image, there are very few cases that a 'yes-no' classifier can determine whether the file is malicious or not. On the other hand, a clustering algorithm can classify a sample in the same category as other similar samples, thus being a better 'judge' of a sample.

7.2.1 KNN Implementation

We will start by explaining the implementations with KNN. We divided our dataset on this machine in the following way: 80% of the samples were used for training, and 20% for testing. At first, we had received 253 malicious samples in our dataset. We completed the dataset by adding 253 benign samples to it, so that the dataset consisted of half malicious samples and half benign samples, containing a total of 506 samples. We ran the machine with default parameters on 1, 3, 5 and 7 neighbors as the K chosen, in order to find the best result.

K	Results
1	90.55%
3	89.76%
5	**91.34%**
7	89.76%

Table 3: Table of results of the different Ks chosen in the KNN implementation.

As can be seen above, the best result we had got was from K=5, with 91.34% success in classifying the test samples. As the project advanced, we had received more malicious samples, and had a dataset that contained 838 samples, consisting of half malicious samples, and half benign samples. We ran the KNN image-based machine again on the whole dataset, and reached 87.5% accuracy of the machine on K=5 neighbors.

7.2.2 KMC Implementation

In order to reach best results with KMC, we wanted to check what would be the best way of defining the number of clusters the algorithm should use. We used the 'Elbow Method' – a method designed to help finding the appropriate number of clusters in a dataset, in such a way that adding another cluster does not give better results. We used two techniques of calculating the ideal number of clusters for our machine: 'MinMaxScaler' and 'StandardScaler', in order to verify that we are reaching the best number of clusters for our machine. The ideal number of clusters depends on the size of the dataset and the samples themselves. When we had 506 samples, the ideal number of clusters was 6, and with 838 samples, the ideal number of clusters was 19 (as seen in the Figure 5). In the clusters we have seen malicious and benign files in the same cluster, according to the basic logic we explained of clustering the images of the files by similarity.

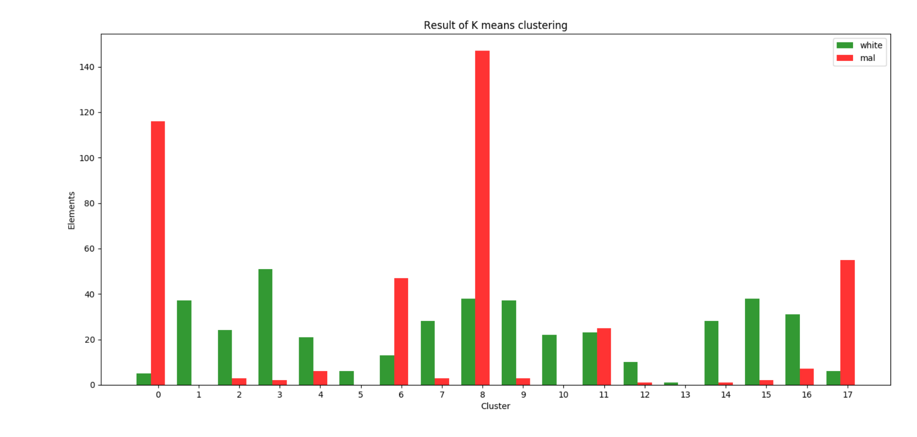


Figure 5: Graph of the different clusters in the KMC implementation.

As seen from the results above, basing our decision on the preview of the PDF file only is obviously not enough. Even the most basic hacker can easily dodge an image-based classification. There are plenty of attacks that do not need to change anything in the PDF's visible content. Examples can be seen in the following paper [4].

8 Third phase: Text Based Classification Machine

In this phase, we wanted to create a text-based classification machine. This machine should classify a PDF file only by the text that the file contains. The idea of classifying a PDF file by the text it contains hasn't been implemented in previous researches we have read, but the overall idea of checking a file by its contents exists.

The first step we made regarding this phase, as part of the first phase, was to extract the text from the first page of the PDF file. In some cases, we have noticed that there was an exaggerated amount of characters, or no characters at all, that were read from a single page of the PDF file. The reason for this was that there can be some stream that represents an image or some graphical content in the file, causing us to read the stream as a whole, and that does not give us the actual content of the file. In these cases, we have decided to deal with this by extracting the text from the image of the first page of the file. This was made as explained in the first phase.

8.1 Creating the vector

The next step is defining the vector that this machine will receive as the input for every sample. We have seen many articles about creating a text vector, and normally this was made by using word embeddings. Word embedding is a collective name for a set of language modeling and feature learning techniques in natural language processing. In word embedding, words or phrases from the vocabulary are mapped to vectors of real numbers. We have decided to try three different word embedding techniques in our project and compare between them.

The word embedding techniques we used were ‘doc2vec’, ‘word2vec’, and ‘TF-IDF’. ‘Word2vec’ uses dictionaries that can be given to it. After some research we have decided to use ‘GoogleNews-vectors-negative300-SLIM.bin.gz’, that uses only words – in order to meet the memory limitations we had. We fixed the size of the vector to 300.

Doc2Vec	Word2Vec	TF-IDF
Default vector made by library.	Default vector with google vocabulary.	Vector is made by counting words and default parameters.

Table 4: Table explains vectors.

8.2 Implementing the machine

When implementing the text-based machine, we followed our advisor’s advice to use deep learning algorithms. We chose logistic regression as the deep learning algorithm we will use. We ran logistic regression on all three word embedding techniques we chose. Specifically, in ‘doc2vec’, we used two different models: Distributed Bag of Words (DBOW) and Distributed Memory with Averaging (DMA), and the combination of both in one model. The best results we have got from ‘doc2vec’ were from the combination of both models.

8.3 Results

In the following image we present the results of the logistic regression algorithm on all three word embedding models. The dataset was split 80% for training and 20% for testing.

Word Embedding Model	**Accuracy**
Doc2Vec (DBOW + DMA)	83.3 %
Word2Vec	**98.4 %**
TF-IDF	**98.4 %**

Table 5: Table of results of the different word embedding with LR.

As can be seen in the results above, ‘word2vec’ and ‘TF-IDF’ gave us the best results (and the same results). We decided to run machine learning algorithms on them in order to try reaching better results. We chose the following machine learning algorithms: Random Forest Classifier, K-Nearest-Neighbors, Support Vector Machine, Multilayer Perceptron, and Naïve-Bayes. We ran these algorithms on a dataset of 506 samples (half malicious and half benign), and the results were as follows.

Word2Vec	**Results**	**TF-IDF**	**Results**
RF Classifier	96.8 %	RF Classifier	96.8 %
KNN	96 %	KNN	63.4 %
SVM	95.2 %	SVM	**99.2 %**
MLP	97.6 %	MLP	97.6 %
NB	88.8 %	NB	96.8 %

Table 6: Table of results of the different machine learning algorithms on ‘word2vec’ and ‘TF-IDF’.

It is important to emphasize that ‘TF-IDF’ has recognized all the malicious samples with the machine learning algorithms – except with Naïve-Bayes. As can be seen above, except for KNN algorithm, we received the best results with ‘TF-IDF’ and therefore decided to continue with this model. We ran the machine again on 838 samples (half malicious and half benign), and compared the results. The dataset was split the same way, 80% for training and 20% for testing.

The best results up to this stage were from the combination of MLP or LR on ‘TF-IDF’, therefore we decided to implement the machine with LR and ‘TF-IDF’. With that said, basing our decision on the text of the PDF file only is obviously not enough. Even the most basic hacker can easily dodge a text-based classification. There are plenty of attacks that do not need to change anything in the PDF’s visible content. Examples can be seen in the following paper [4].

Algorithm	**506 Samples**	**838 Samples**
Logistic Regression	98.41 %	**97.46 %**
KNN	63.49 %	93.03 %
MLP	97.61 %	**97.46 %**
NB	96.82 %	93.03 %
RF Classifier	96.82 %	95.56 %
SVM	**99.20 %**	96.83 %

Table 7: Table with the results of 'TF-IDF' with all the algorithms we tried on two datasets.

9 Fourth phase: Features Based Classification Machine (JS, URL, objects and streams)

In the fourth phase, we created a machine that classifies PDF samples as malicious or benign, based on their features. This type of machine is the most popular amongst researches we have read, and many researches we have read [3] [2] [13] [7] [10] have focused on this approach. Most researches that focused on the file's features have not described in detail the features they have used; we will give some background about the features we have chosen. The features we have chosen come from four different areas: PDF Tags, JavaScript Objects & Streams, URLs, and Entropy.

9.1 Creating the vector

The vector we created held 32 values, each field of the vector contained a numeric value regarding one of the features we chose. The 32 features were made up of 12 PDF tags, 10 features that characterize the URLs in the file, 7 features that characterize the JS, objects and streams in the file, and 3 features regarding the entropy of the file.

9.1.1 PDF Tags

In order to extract the PDF tags we were interested in, as mentioned in the project proposal chapter, there are many existing tools that were used in many researches that were available to us. In this case we have chosen to work with the 'PDFiD' tool by Didier Stevens [3]. This tool helps us count the number of appearances of specific tags in the PDF file. The tags we have chosen were:

1. **Obj** - This tag opens an object in the PDF.

2. **Endobj** - This tag closes an object in the PDF.

3. **Stream** - This tag opens a stream in the PDF.

4. **Endstream** - This tag closes a stream in the PDF.
5. **/ObjStm** - Counts the number of object streams. An object stream is a stream object that can contain other objects, and can therefore be used to obfuscate objects (by using different filters).
6. **/JS**, **/JavaScript** - These tags indicate that the PDF document contains JavaScript. Almost all malicious PDF documents found in the wild (by Didier Stevens) contain JavaScript (to exploit a JavaScript vulnerability and/or to execute a heap spray). JavaScript can also be found in PDFs without malicious intent.
7. **/AA**, **/OpenAction** - These tags indicate an automatic action to be performed when the file is viewed. All malicious PDF documents with JavaScript seen in the wild (by Didier Stevens) had an automatic action to launch the JavaScript without user interaction. The combination of automatic action and JavaScript makes a PDF document very suspicious.
8. **/RichMedia** - This tag can imply presence of flash in the file.
9. **/Launch** - This tag counts launch actions in the file.
10. **/AcroForm** - This tag is defined if a document contains form fields, and is true if it uses XML Forms Architecture.

The first four features were chosen specifically in order to find malformations in the PDF files format. As we have seen in previous researches, many of them have specifically linked malformations in the files to malicious intents.

9.1.2 URLs

In order to extract the URL features we were interested in, we wrote a simple python parser that extracted what we were looking for. Due to the specific features and the format of URLs, that was not too hard to do. The features we chose are:

1. Overall number of URLs in the PDF file.
2. Number of different URLs in the PDF file.
3. Number of URLs with the expression "File://" .
4. Longest length of string after the second slash in the URL.
5. Number of URLs that contain another URL in them.
6. Number of URLs that contain encoded characters in the hostname.
(Example: http://www.%63%6c%69%66%74.com)
7. Number of URLs that contain IPs in them.

8. Number of URLs that contain suspicious expressions (such as: download, php, target, loader, login, =, ?, &, +).

9. Number of URLs that contain unusual ports after the colon (':').

9.1.3 JavaScript, Objects & Streams

In order to extract the JS from the files we have used an existing tool called 'PeePDF'. We used this tool in order to extract all the JavaScript code found in each sample and searched in the code extracted features we were interested in. The JavaScript was extracted from the objects and streams of the files.

Another tool we used was called 'JAST'. This tool classifies whether a string has JavaScript in it or not, and knows how to recognize obfuscated JavaScript code. This helped us extract an important feature about JavaScript in the PDF file.

The features we chose are:

1. Number of objects with JavaScript in them in the PDF file.

2. Number of lines with JavaScript code in the PDF file.

3. Kind of JavaScript in the file: no JS, regular JS, obfuscated JS.

4. Four features for special expressions:

   (a) Number of 'eval' expressions found.

   (b) Number of backslash characters found.

   (c) Number of '/u0' expressions found.

   (d) Number of '/x' expressions found.

9.1.4 Entropy

Entropy is a measurement for the lack of order or predictability of the content of the PDF files. In order to extract this feature from our samples we used an existing tool called 'AnalyzePDF', that calculates the entropy of the files. This feature was suggested to us by our advisors. We have decided to measure the following:

1. The overall entropy of the PDF file's content.

2. Entropy inside the PDF file's streams.

3. Entropy outside the PDF file's streams.

All the features we have chosen were based on decisions we have made and things we have thought were interesting and important, tips from our advisors and previous researches and articles. All the tools that are mentioned in this stage are introduced in the 'existing tools' section of this work.

9.2 Implementing the machine

At first, we ran a few different machine learning algorithms on our 32-sized vector to find the algorithm that will give us the best results. Similar to the second machine, we chose the following algorithms: Logistic Regression, K-Nearest-Neighbors, Multilayer Perceptron, Random Forest Classifier and Support Vector Machine. We ran the machine twice, on 506 samples and on 838 samples, both datasets were made of half malicious samples and half benign samples. We split the two datasets the same way: 80% for training and 20% for testing.

9.3 Results

Algorithm	**506 Samples**	**838 Samples**
Logistic Regression	84.25 %	82.73 %
KNN	59.84 %	70.23 %
MLP	57.48 %	61.90 %
RF Classifier	**92.12 %**	**94.64 %**
SVM	48.81 %	57.73 %

Table 8: Table with the results of the feature-based machine.

Note that these results are lower than most researches we have read that based their machine on the features of the PDF file [2]. After receiving these disappointing results, we have come up with ways to improve this machine, but have not implemented those improvements. In the 'future work' section we will list the ideas we had to improve this machine. A feature-based machine only, similar to previous machines, has also proved to be not enough against evasion attacks [13].

10 Fifth phase: Creating the ensemble classifier

The aim of this final phase was to create the fourth machine, an ensemble machine that will provide us with the final classification of a PDF file. This machine will classify the file based on all three classifications made by the previous machines.

10.1 Creating the vector

To implement that, we decided that there are a few combinations that are worth trying and comparing, before deciding what would be the best strategy for this machine. These strategies were regarding the vector that the ensemble machine would receive as its input. The vector of this machine could be assembled in quite a few different ways, as follows:

- The vector could contain only 3 fields with the decisions of all three previous machines.
- The vector could contain all the vectors from previous stages (513 fields for the image, 300 fields for text, 32 fields for features = 845 fields overall).
- The vector could contain part of the vectors from previous stages and part of the classifications of previous stages, in different combinations.

10.2 Implementing the machine

Furthermore, depending on the vector chosen, we had to apply some machine learning algorithm, in order for this final machine to give its final classification. In order to do that, we chose six different algorithms, and ran them on the different vectors we chose. These algorithms were: AdaBoost Classifier, AdaBoost Regressor, XGB Classifier, XGB Regressor, Random Forest Classifier, Random Forest Regressor.

10.3 Improving Vector

While running the fourth machine, we decided to examine the feature importance of the vectors. As explained above in the previous phase, we chose 32 features for the third machine of this project, and have decided to see if all of these features were being used or if they add value to our vector and machine. When we ran the fourth machine, using all previous machines, we checked for each feature if it was used in the different algorithms of the fourth machine on our samples (838 samples, half malicious and half benign). We discovered that four of the features were not being used at all, therefore we have removed them from the feature vector of the fourth machine. The features removed were: the '/AcroForm' PDF tag, the number of '/x' expressions, the number of '/u0' expressions and the number of URLs that contain encoded characters in the hostname.

Moreover, we have made the feature importance examination for all the vectors of previous machines, and have seen the following:

As the table above shows, only AdaBoost regressor in the fourth machine does not use the blur of the image. Moreover, only the RF classifier and RF regressor use 90%-100% of the text vector. The same can be said about the feature vector.

10.4 Initial Results

In this stage of the project we chose all sorts of possible vectors and all six machine learning algorithms mentioned above, and ran the ensemble machine in order to compare the results and find the best combination. We have tried the following vectors:

	AdaBoost Classifier	**AdaBoost Regressor**	**XGB Classifier**	**XGB Regressor**	**RF Classifier**	**RF Regressor**
Image Histogram (1-512)	V	V	V	V	V	V
Image Blur (513)	V	X	V	V	V	V
Text (514-813)	31/300	138/300	29/300	73/300	300/300	271/300
Features (814-845)	8/32	16/32	7/32	10/32	26/32	21/32

Table 9: Table with the usage of the features in the whole vector in the fourth machine.

- Overall vector (vector that contains all previous stages vectors) with 32 features, and 28 features.
- Vector of machine results combining KNN and KMC for the first machine, and 32 and 28 features in the third machine.
- Combined vector, partly made of previous stages vectors, and partly by previous machine results.

In the table below, the results of all types of vectors we tried on all six machine learning algorithms for the fourth machine are presented. The best result can be seen in the last row of the table.

	AdaBoost Classifier	**AdaBoost Regressor**	**XGB Classifier**	**XGB Regressor**	**RF Classifier**	**RF Regressor**
Overall vector (32 features)	97.62 %	97.02 %	97.02 %	95.24 %	95.38 %	95.24 %
Overall vector (28 features)	97.62 %	97.02 %	97.02 %	95.24 %	96.43 %	95.24 %
Vector of machine results (KNN + SVM + 32 features)	92.26 %	90.48 %	94.05 %	94.05 %	92.26 %	94.05 %

Continued on next page

Vector of machine results (KNN + SVM + 28 features)	92.26 %	91.07 %	92.86 %	94.64 %	94.64 %	94.64 %
Vector of machine results (KMC + SVM + 32 features)	92.26 %	93.45 %	93.45 %	93.45 %	94.05 %	93.45 %
Vector of machine results (KMC + SVM + 28 features)	92.26 %	93.45 %	93.45 %	93.45 %	94.64 %	95.24 %
Combined Vector (KMC + TEXT-VECTOR + 28 features)	98.21 %	97.02 %	97.02 %	95.83 %	95.24 %	96.43 %
Combined Vector (KMC + SVM + 28 features)	95.83 %	97.02 %	97.02 %	97.62 %	97.62 %	97.02 %
Combined Vector (IMAGE-VECTOR + SVM + 28 features)	**98.81 %**	97.62 %	**98.81 %**	98.21 %	98.21 %	98.21 %

Table 12: Table with the results of the fourth machine on combinations of vectors and algorithms.

10.5 Feature Importance

In order to understand the relevance of each machine in the project, we ran the fourth machine on the vector of machine results and retrieved the feature importance of every algorithm. We ran the machine with the next definitions: KMC for the first machine (image), SVM for the second machine (text) and RF for the third machine (PDF features). The results are shown in the table below:

	Image	**Text**	**Features**
AdaBoost Classifier	0.93	0.04	0.03
AdaBoost Regressor	0.18419997	0.60848697	0.20731306
XGB Classifier	0.01018084	0.92208797	0.067731306
XGB Regressor	0.01556311	0.94997084	0.03446609
Random Forest Classifier	0.14533861	0.53994799	0.31471341
Random Forest Regressor	0.06494484	0.90513809	0.02991707

Table 13: Table with the feature importance of all algorithms in fourth machine on vector of machine results. (First machine – KMC, second machine – SVM, third machine – RF).

From the results in the above table 13, it can be seen clearly that the text feature in the vector of machine results is the most significant one in feature importance in most cases. That result awed us, as it was not expected. The text and image machines are expected to be less significant since they judge the visible content of the PDF file only. As mentioned in the first and second phase chapters, visible content in the PDF does not necessarily mean that there is not more content in the file, such as JS code and all that was mentioned in phase four.

In order to improve our results in the fourth machine, we needed to go back and rethink our strategy for all three previous machines. At first, when we built the first three machines, we ran them on a dataset of 506 samples. When we ran the machines again on 838 samples – we saw different results (shown in the first, second and third phase chapters). For example, with 506 samples we saw that for the first machine (image), the best result we received was using KNN, but with 838 samples we got the best results using KMC. Moreover, with 506 samples we saw that for the second machine (text), the best result we received was using SVM, but with 838 samples we got the best results using LR.

In addition to that, during our project we had a meeting with another researcher that focuses on evasion attacks. From his point of view, the first and second machines of our project were very ineffective against an attacker that is acquainted with the machines (that knows that the first page is always taken for image and text extraction). Therefore, we have added to the functionality of these machines, the option of choosing a random page of the file to extract the image and text from. The random selection for the page is done twice, once for the text, and once for the image, meaning that there may be two different pages chosen from the file (unless the length of the file is one page only).

11 Results

In this chapter we will present our results and explain them. We will present the final results of the ensemble machine. The vector chosen from all the options present in section 10.1 was the vector of machine results, with 3 features, each representing a result from a machine (image, text, PDF features). The chosen algorithms for the first three machines are: KMC for the image classification machine, LR for the text classification machine, and RF for the PDF features classification machine.

Result of image machine	Result of text machine	Result of PDF features machine

Table 14: Vector of machine results.

The results are on the following two datasets in two variations for each, depending on the way the text and image pages are selected:

Dataset Index	Number of samples	Dataset content	Page Selection
1	838	50% malicious, 50% benign	First page
2	838	50% malicious, 50% benign	Random
3	9,577	95.625% benign, 4.375% malicious	First page
4	9,577	95.625% benign, 4.375% malicious	Random

Table 15: Table with the groups that we refer to in the final results.

For all four groups presented above, the two datasets were divided in the following way:

- Train first three machines: 70% of the samples (benign and malicious respectively).

- Train ensemble machine: 20% of the samples (benign and malicious respectively).
- Test: 10% of the samples (benign and malicious respectively).

11.1 Results of first group (838 samples; 50% malicious / 50% benign; first page for text & image):

Firstly, we present the accuracy parameters of all six algorithms on the dataset. As can be seen in the table below, here we have received the best results from the XGB classifier, with 96.43% accuracy. An interesting thing that can be seen in the table is that XGB regressor had more than one percent less accuracy than XGB classifier, but the precision on the benign files and the recall on the malicious files was 100%.
Accuracy Parameters:

	Accuracy	**Class**	**Precision**	**Recall**	**F1-Score**
AdaBoost Classifier	95.24%	Benign	98%	93%	95%
		Malicious	93%	98%	95%
AdaBoost Regressor	92.86%	Benign	95%	91%	93%
		Malicious	91%	95%	93%
XGB Classifier	**96.43%**	Benign	95%	98%	97%
		Malicious	97%	95%	96%
XGB Regressor	95.24%	Benign	100%	91%	95%
		Malicious	91%	100%	95%
Random Forest Classifier	94.05%	Benign	95%	93%	94%
		Malicious	93%	95%	94%
Random Forest Regressor	94.05%	Benign	95%	93%	94%
		Malicious	93%	95%	94%

Table 16: Accuracy parameters for all algorithms on first group.

As in the previous chapter, we wanted to examine the feature importance of our vector, made of all three previous machines results for this group:
Feature Importance:

	Image	**Text**	**Features**
AdaBoost Classifier	0.95	0.03	0.02
AdaBoost Regressor	0.23878133	0.28937013	0.47184864
XGB Classifier	0.02476732	0.87546283	0.099769983
XGB Regressor	0.02019422	0.91034234	0.06946351
Random Forest Classifier	0.1556851	0.46782244	0.037649246
Random Forest Regressor	0.10489435	0.58775198	0.30735367

Table 17: Feature importance for all algorithms on first group.

As can be seen above, the results of the feature importance change significantly between the algorithms. The result of the text importance is still relatively high in as to what we expect it to be in most algorithms.

Confusion Matrices: (Positive = Benign, Negative = Malicious) In the following table we present the confusion matrices for all algorithms, on the first group. The columns of the table are as follows:

- Pos. as Pos. – Benign samples classified as benign.
- Pos. as Neg. - Benign samples classified as malicious.
- Neg. as Pos. - Malicious samples classified as benign.
- Neg. as Neg. - Malicious samples classified as malicious.

	Pos. as Pos.	**Neg. as Pos.**	**Pos. as Neg.**	**Neg. as Neg.**
AdaBoost Classifier	40	1	3	40
AdaBoost Regressor	39	3	4	39
XGB Classifier	42	2	1	39
XGB Regressor	39	**0**	4	41
Random Forest Classifier	40	2	3	39
Random Forest Regressor	40	2	3	39

Table 18: Confusion matrices for all algorithms on first group. Note that 84 samples are shown in the table (10% of samples that were used for test).

An interesting fact from the confusion matrices is that the XGB regressor has not classified even a single malicious file as benign.

In addition to the results above, we decided to run the trained machines on the first group (machines that were trained and tested on 838 samples with the first page selected for the extraction of image and text of the PDF file) on a dataset that contained only benign samples. This all-benign dataset contained 8,739 samples that none of them were used in the previous dataset of 838 samples. The results of running the trained machines on the all-benign dataset were as follows:

	Accuracy	**Pos. as Pos.**	**Pos. as Neg.**
AdaBoost Classifier	95.23%	8322	417
AdaBoost Regressor	93.80%	8197	542
XGB Classifier	97.12%	8487	252
XGB Regressor	95.38%	8335	404
Random Forest Classifier	95.28%	8414	325
Random Forest Regressor	95.28%	8414	325

Table 19: Result over remaining benign samples, first group

11.2 Results of second group (838 samples; 50% malicious / 50% benign; random pages for text & image):

We present here the results of the second group. The difference between this group and the previous one is the way the choice of the page for the image and text is made. Here the choice for these is made randomly.

Accuracy Parameters:

In the accuracy parameters of this group we can see that the random choice of page affected the results in all algorithms, causing a decrease in the accuracy.

	Accuracy	**Class**	**Precision**	**Recall**	**F1-Score**
AdaBoost Classifier	90.48%	Benign	95%	86%	90%
		Malicious	87%	95%	91%
AdaBoost Regressor	91.67%	Benign	93%	91%	92%
		Malicious	90%	93%	92%
XGB Classifier	91.67%	Benign	93%	91%	92%
		Malicious	90%	93%	92%
XGB Regressor	92.86%	Benign	93%	93%	93%
		Malicious	93%	93%	93%
Random Forest Classifier	92.86%	Benign	95%	91%	93%
		Malicious	91%	95%	93%
Random Forest Regressor	92.86%	Benign	95%	91%	93%
		Malicious	91%	95%	93%

Table 20: Accuracy parameters for all algorithms on second group.

Feature Importance:

The feature importance was also affected from the random choice of pages for text and image extraction. The table below shows a significant decrease in the importance of the text feature, and a significant increase in the third feature, that contains the classification of the third machine on PDF features.

	Image	**Text**	**Features**
AdaBoost Classifier	0.95	0.02	0.03
AdaBoost Regressor	0.44241618	0.08196731	0.475616652
XGB Classifier	0.01986038	0.04689374	0.9332459
XGB Regressor	0.02112981	0.03368637	0.9451838
Random Forest Classifier	0.30664552	0.22409619	0.46925829
Random Forest Regressor	0.11688741	0.05400321	0.82910937

Table 21: Feature importance for all algorithms on second group.

Confusion Matrices: (Positive = Benign, Negative = Malicious)

Here we present the confusion matrices for all algorithms on the second group. Note that the columns of the table are the same as in presented in the previous group's results. We can see a slight decrease in the classification of the samples.

	Pos. as Pos.	**Neg. as Pos.**	**Pos. as Neg.**	**Neg. as Neg.**
AdaBoost Classifier	37	2	6	39
AdaBoost Regressor	39	3	4	38
XGB Classifier	39	3	4	38
XGB Regressor	40	3	3	38
Random Forest Classifier	39	2	4	39
Random Forest Regressor	39	2	4	39

Table 22: Confusion matrices for all algorithms on second group. Note that 84 samples are shown in the table (10% of samples that were used for test).

As on the first group, here too we ran the machines that were trained on the second group (machines that were trained and tested on 838 samples with random page selection for image and text extraction) on the all-benign dataset. The results of running the trained machines on the second group on the all-benign dataset were as follows:

	Accuracy	**Pos. as Pos.**	**Pos. as Neg.**
AdaBoost Classifier	93.50%	8171	568
AdaBoost Regressor	96.28%	8414	325
XGB Classifier	93.60%	8180	559
XGB Regressor	94.01%	8216	523
Random Forest Classifier	90.03%	7868	871
Random Forest Regressor	94.01%	8216	523

Table 23: Result over remaining benign samples, second group.

11.3 Results of third group (9,577 samples; 95.625% malicious / 4.375% benign; first page for text & image):

In this group, as written above, we reach the larger dataset, with mostly benign samples. This dataset contains all the samples we have, and the difference in the amounts of malicious and benign samples affects the results.

Accuracy Parameters:

In the accuracy parameters of this group we note that the AdaBoost classifier has returned the best result, 0.21% higher than all the other algorithms that returned the same result in the accuracy parameter.

	Accuracy	**Class**	**Precision**	**Recall**	**F1-Score**
AdaBoost Classifier	**99.05%**	Benign	99%	100%	100%
		Malicious	92%	86%	89%
AdaBoost Regressor	98.84%	Benign	99%	99%	99%
		Malicious	88%	86%	87%
XGB Classifier	98.84%	Benign	99%	99%	99%
		Malicious	88%	86%	87%
XGB Regressor	98.84%	Benign	99%	99%	99%
		Malicious	88%	86%	87%
Random Forest Classifier	98.84%	Benign	99%	99%	99%
		Malicious	88%	86%	87%
Random Forest Regressor	98.84%	Benign	99%	99%	99%
		Malicious	88%	86%	87%

Table 24: Accuracy parameters for all algorithms on third group.

Feature Importance:

The feature importance has also changed from previous groups on the smaller dataset. Here we can see a real mix-and-match in the feature importance in the algorithms. All algorithms except our leading algorithm in accuracy, AdaBoost classifier, have given very little importance to the image, and AdaBoost classifier has given almost all the importance to the image feature. This fact is a bit confusing.

	Image	**Text**	**Features**
AdaBoost Classifier	0.95	0.03	0.02
AdaBoost Regressor	0.19335592	0.68552074	0.12112334
XGB Classifier	0.01837745	0.29502392	0.6865986
XGB Regressor	0.00695719	0.44185147	0.5511914
Random Forest Classifier	0.03115066	0.4734846	0.49536474
Random Forest Regressor	0.01641713	0.35635909	0.62722378

Table 25: Feature importance for all algorithms on third group.

Confusion Matrices: (Positive = Benign, Negative = Malicious)

Here we present the confusion matrices for all algorithms on the third group. These results resemble the results of the accuracy parameters in the fact that all the amounts are the same for all algorithms, except for AdaBoost classifier. The difference in the accuracy parameters can be explained by the table below.

	Pos. as Pos.	**Neg. as Pos.**	**Pos. as Neg.**	**Neg. as Neg.**
AdaBoost Classifier	904	6	3	36
AdaBoost Regressor	902	6	5	36
XGB Classifier	902	6	5	36
XGB Regressor	902	6	5	36
Random Forest Classifier	902	6	5	36
Random Forest Regressor	902	6	5	36

Table 26: Confusion matrices for all algorithms on third group. Note that 949 samples are shown in the table (10% of samples that were used for test).

11.4 Results of fourth group (9,577 samples; 95.625% malicious / 4.375% benign; random pages for text & image):

In this group, as written above, the dataset is much bigger, with mostly benign samples. The difference between this group and the previous one is the way the choice of the page for the image and text is made. Here the choice for these is made randomly.

Accuracy Parameters:

As can be seen in the accuracy parameters below, the random choice of pages for the image and text extraction has decreased the performance of the fourth machine. The highest accuracy was received from AdaBoost classifier and RF classifier with 98.63% accuracy. The rest of the algorithms had the same accuracy, 98.52%, only 0.11% less than the two leading algorithms.

	Accuracy	Class	Precision	Recall	F1-Score
AdaBoost Classifier	**98.63%**	Benign	99%	100%	99%
		Malicious	91%	76%	83%
AdaBoost Regressor	98.52%	Benign	99%	100%	99%
		Malicious	89%	76%	82%
XGB Classifier	98.52%	Benign	99%	100%	99%
		Malicious	89%	76%	82%
XGB Regressor	98.52%	Benign	99%	100%	99%
		Malicious	89%	76%	82%
Random Forest Classifier	**98.63%**	Benign	99%	100%	99%
		Malicious	91%	76%	83%
Random Forest Regressor	98.52%	Benign	99%	100%	99%
		Malicious	89%	76%	82%

Table 27: Accuracy parameters for all algorithms on fourth group.

Feature Importance:

The feature importance has also been affected by the random selection of the pages for the image and text extraction. Most algorithms have increased the importance of the third feature - classification of the PDF features machine.

	Image	Text	Features
AdaBoost Classifier	0.92	0.02	0.06
AdaBoost Regressor	0.16147962	0.12584921	0.71267117
XGB Classifier	0.01837745	0.29502392	0.71267117
XGB Regressor	0.00457967	0.29886076	0.69655967
Random Forest Classifier	0.03530152	0.31630768	0.6483908
Random Forest Regressor	0.01775447	0.15577358	0.82647196

Table 28: Feature importance for all algorithms on fourth group.

Confusion Matrices: (Positive = Benign, Negative = Malicious)

Here we present the confusion matrices for all algorithms on the fourth group. As can be seen in the table below, the confusion matrices of all six algorithms are similar.

	Pos. as Pos.	**Neg. as Pos.**	**Pos. as Neg.**	**Neg. as Neg.**
AdaBoost Classifier	904	10	3	32
AdaBoost Regressor	903	10	4	32
XGB Classifier	903	10	4	32
XGB Regressor	903	10	4	32
Random Forest Classifier	904	10	3	32
Random Forest Regressor	903	10	4	32

Table 29: Confusion matrices for all algorithms on fourth group. Note that 949 samples are shown in the table (10% of samples that were used for test).

11.5 Comparing between results from third and fourth groups:

In this section we will show the differences between the performances of the models of the third and fourth groups on the second dataset. As can be seen below, there is a slight decrease when the page choice for the extraction of the image and text is done randomly. Each of these groups (third and fourth) has its own advantage. The third group returns slightly better results, but the fourth group handles evasion attacks in the image and text machines.

	AdaBoost Classifier	**AdaBoost Regressor**	**XGB Classifier**	**XGB Regressor**	**RF Classifier**	**RF Regressor**
First Page	**99.05 %**	98.84 %	98.84 %	98.84 %	98.84 %	98.84 %
Second Page	**98.63 %**	98.52 %	98.52 %	95.24 %	**98.63 %**	98.52 %

Table 30: Overall accuracy of the algorithms on the third and fourth group.

There are plenty of researches that have dealt with evasion attacks in the PDF feature area, therefore we do not focus on this in our work.

11.6 Comparing our results to results from previous work:

In our project we have focused on three areas of interest in a PDF file: image, text and PDF features. Most of the previous researches we have read have focused on one main area of interest. These vary from static lexical analysis, JavaScript analysis, PDF features and more. These previous researches have either supplied a tool that identifies malicious files, or a classification model using machine learning.

In this section we will compare the results we received from our ensemble machine to results from two previous researches. The first research focuses on the PDF feature extraction and machine learning techniques on the vector extracted. This resembles the third machine we have created in our project. The second research focused on the structural properties of PDF files (learning the difference between the structural trees of malicious and benign files). It delves into the relations between the objects of the file in its structural tree.

In Torres and De Los Santos research that was made in 2018 [2], they have combined one set of features and three ML algorithms to classify if a PDF is malicious or benign. The features they chose were not presented, but they explained that they were only related to PDF tags (such as /JS, etc.). The machine learning algorithms they used were SVM, RF, and MLP. Their dataset contained 1,712 samples – half of the samples were malicious and half benign. They divided their dataset in the following way: 995 samples (about 58%) for training, 217 samples for validation (about 12%), 500 samples (30%) for testing. Their results are presented in the table below:

Algorithm	**Accuracy**
SVM	50%
RF	92%
MLP	96%

Table 31: Overall accuracy of the algorithms.

As the table above shows, the highest accuracy reached in this research was 96% with MLP algorithm. We will compare these results with the results presented for the first and second group in the results chapter because of the resemblance between the division of malicious and benign samples in the dataset (made of half malicious and half benign in both). It is important to note that we had less than half the number of samples in the dataset for the first and second group (838 samples) that Torres and De Los Santos had in their research (1,712 samples).

	SVM - Previous work	**First Group**	**Second Group**	**Combined Vector** (IMAGE -VECTOR + SVM + 28 features)
Accuracy	96%	96.43%	92.86%	98.81%

Table 32: Comparison of overall accuracy between our work and Torres & De Los Santos's research.

In the table above, we compare the results of the first two groups in the results chapter and the best result we have gotten on 838 samples from the fifth phase of our project to the results from the research [2]. In the second group we have worse results, but in the first group and the combined vector, we can see that we reach higher results with less samples. Overall we are pretty close to the results they have gotten from PDF tags feature only.

In another research by researchers from Ben Gurion University [5] we have seen a different and interesting approach for the identification of malicious PDF files. They implemented the idea presented in Srndic and Laskov [8]. Srndic and Laskov presented a static analysis approach on the hierarchical structural path feature extraction method. This approach makes use of essential differences in the structural properties of malicious and benign PDF files. The structural paths represent the file's properties and possible actions.

The model presented in [5] receives new samples every day, part of them were used for retraining the machine, and part used for retesting. The initial dataset contained 76% benign samples (446 samples) and 24% malicious (128 samples). Each day they added 160 new samples to the model (adjusting the data to resemble the initial malicious/benign ratio). If an interesting file was located by the model, it was used to retrain the machine. The algorithm they used in their model was SVM-Margin. At the end of 10 days they reached the following results:

	TPR	**FPR**
Third Group (benign = pos.)	99.67%	7.14%
Fourth Group (benign = pos.)	99.67%	7.14%
Third Group (malicious = pos.)	85.71%	0.66%
Fourth Group (malicious = pos.)	76.19%	1.10%
SVM-Margin (10 day)	96%	0.05%

Table 33: Comparison of overall TPR and FPR between our work and researchers from Ben Gurion University.

In the results of [5], they have not shown accuracy, only the TPR (true positive rate) and FPR (false positive rate) measurements of their work. In their work they have related to malicious samples as positive. In the table above these measurements are compared to ours. As can be seen, their research has returned better results. By the measurements above we can see that our model is very good for the identification of benign PDF files, but not as much for malicious PDF files. The fact that there were very few malicious samples in our dataset (4.375% malicious samples in our dataset compared to 24% malicious samples in their dataset) made the machines look for very specific things about the malicious files, therefore reaching not good enough results for our model.

In order to compare the models in a better way (to both researches), we need to increase the number of malicious samples in our datasets, and adjust the datasets in a way that will match the datasets in the two researches shown above in a better way (size and content of our dataset).

11.7 Conclusion

The aim of our project was to create an ensemble machine that will serve as a classifier for PDF files. This ensemble machine focuses on three different areas of the PDF file: image, text and file features. The first machine uses a clustering algorithm on a vector of the image histogram and blur, extracted from the first page of the file. The second machine uses 'TFIDF' as the word embedding and logistic regression as the machine learning algorithm to classify the file by the text extracted from the first page of the file. The third machine uses a vector of features extracted from the whole file, and random forest as the machine learning algorithm to classify the file by its features.

During the different phases of our work, while working on each machine, we had tried different approaches for the way of building the vectors, and different deep learning and machine learning algorithms. In addition to that, during our work we met with our advisors and another researcher and talked about different ways that an attacker can evade our ensemble machine. We built an algorithm that instead of always choosing the first page for our two first machines, chooses a random pages of the file for that.

We have compared our results to two researches that have two different approaches, each focusing on one area of interest of the PDF file. We have indicated what has to be improved in our project and dataset in order to reach more realistic results. The improvements include adding malicious samples to our dataset, and creating a recommended proportion between the malicious and benign files in the dataset as can be seen in [5].

12 Future Work

During the research and execution of our work, we have encountered many approaches, methods and tools that deal with the identification of malicious PDF files. There are many researches that have developed new approaches and tools with good results. When we started our project, we have focused on three specific areas of interest and machine learning techniques for those three areas.

Combining three approaches in one work is relatively not as common, but there are many more methods that can be used and combined in our work.
Overall the biggest issues in our work that we would like to address in the future are:

1. The dataset – We would like to increase the number of malicious samples in the dataset and find the best balance between malicious and benign samples in the dataset [5], relevant for all the machines in the project.

2. Integration of more methods – Another option is increasing the number of methods used to classify the files. That means adding machines with more types of classification and increasing the size of the ensemble machine's vector. By methods we mean the feature selection methods (more types of data regarding the file such as object structure, etc.), and the detection approaches (using not only machine learning as has been done, but adding more approaches such as statistical analysis and clustering) [6] [9].

3. Training method – In our work we have trained the machines once, and tested them directly. Another approach that can be implemented is using a retraining method, that trains the machines periodically or every time there is more samples, this way maybe we can achieve better results.

With all that said, each phase of our project can be improved itself, we present these improvements in the sections below.

12.1 Phase 2 Improvements

12.1.1 Additional methods of picture classification

Histogram is clearly not the only approach existing in the classification of images. Comparing the vector that we created with other approaches may yield better results than the results we achieved.

12.1.2 Near Similar Image Matching

Image matching may be another way to reach image classification in PDF files. It works by detecting special features of the image and comparing to others.

12.2 Phase 3 Improvements

12.2.1 Extraction of text to work for more languages

In our project the extraction of text from image works only for Latin and Cyrillic characters. That means that if there is another language and we have to extract the text from the image, the text machine won't work. Furthermore, if the text is directly extracted from the file or from the image of the file, and is not in English, it will be needed to write a machine suitable for that language (the machine we built works only on English text).

12.2.2 Improving text vector

During our work we considered trying another approach of creating the text vector using ‘Word2vec’ with cosine similarity. Due to time limitations we did not get to do it.

Moreover, we used a slim dictionary due to memory limitations, but there are many advanced dictionaries that also include relations between words and phrases that can be used and may improve results.

12.3 Phase 4 Improvements

12.3.1 Feature selection improvement

During our work, we have read many articles and researches that focus on whole file features, PDF tags, and more. Many of these did not mention specifically what they used, and many did. We could not include everything in our work, but we could try to make a few combinations to see if we can reach better results just by changing the features in our vector.

An example for features we did not know about was ‘SubmitForm’, a PDF tag we discovered while writing the project book [10]. Another example is “/GoTo” [5].

Another approach of improving the vector is combining features, for example instead of creating a feature for the /Obj tag and another for /EndObj tag, creating one feature that indicates if the number of objects opened and the number of the objects closed are the same or not.

Another idea is adding the approach that uses N-Grams in order to distinguish between benign and malicious PDF files. This can be seen in [11].

Lastly, we could create a list of all possible features, and choose randomly from it repeatedly until we find the list that yields us the best classification.

12.4 Phase 5 Improvements

12.4.1 Additional methods

In the ensemble machine we could continue trying to improve the vector and the algorithms used in order to reach better results.

12.5 Missing trailer & malformation in ‘xref’ table

We have encountered numerous problems with files that did not have trailers. Even though in Didier Stevens’s work [3] it is mentioned that this does not necessarily affect the files, it caused us not to be able to open them.

Moreover, we did not delve into the area of malformations in the PDF file. Throughout our work we have seen researches that focused specifically in this area [2] [12], and learnt for example that if there are objects in the file that do not appear in the ‘xref’ table, there is a high chance that a benign file was altered to contain malicious contents.

References

[1] Patil, Dharmaraj R and Patil, JB. *Cybernetics and Information Technologies.* [*Malicious URLs Detection Using Decision Tree Classifiers and Majority Voting Technique*]. pages:11–29, 2018.

[2] Torres, Jose and De Los Santos, Sergio. *Malicious PDF Documents Detection using Machine Learning Techniques.* 2018.

[3] Didier Stevens. *Pdf Tools.* `https://blog.didierstevens.com/programs/pdf-tools/` 2017.

[4] Davide Maiorca, Battista Biggio and Giorgio Giacinto. *Towards Adversarial Malware Detection: Lessons Learned from PDF-based Attacks.* April 2019.

[5] Nir Nissim, Aviad Cohen, Robert Moskovitch, Asaf Shabtai, Matan Edri, Oren BarAd and Yuval Elovici. *Keeping pace with the creation of new malicious PDF files using an active-learning based detection framework.*

[6] Nir Nissim, Aviad Cohen, Chanan Glezer and Yuval Elovici. *Detection of Malicious PDF files and directions for enhancements: A state-of-the-art survey.*

[7] Nedim Srndic and Pavel Laskov. *Static Detection of Malicious JavaScript-Bearing PDF Documents.* 2011.

[8] Nedim Srndic and Pavel Laskov. *Detection of Malicious PDF Files Based on Hierarchical Document Structure.* 2013.

[9] Michele Elingiusti, Leonardo Aniello, Leonardo Querzoni and Roberto Baldoni. *PDF-Malware Detection: A Survey and Taxonomy of Current Techniques.* 2018.

[10] Hamon Valentin. *Malicious URI Resolving in PDF Documents.* 2013.

[11] Joachims Thorsten. *Making Large Scale SVM Learning Practical.* 1999.

[12] Yuhei Otsubo. *O-checker: Detection of Malicious Documents through Deviation from File Format Specification.*

[13] Bonan Cuan, Aliénor Damien, Claire Delaplace, Mathieu Valois. *Malware Detection in PDF Files Using Machine Learning.* 2018.

[14] Aurore Fass, Robert P. Krawczyk, Michael Backes, Ben Stock. *JaSt: Fully Syntactic Detection of Malicious (Obfuscated) JavaScript.* 2018.

[15] Hiddenillusion. *AnalyzePDF.* `https://github.com/hiddenillusion/AnalyzePDF` 2014.

[16] Jesparza. *PeePDF.* `https://github.com/jesparza/peepdf` 2016.

[17] Adrian Rosebrock. *KNN Classifier for Image Classification.* `https://www.pyimagesearch.com/2016/08/08/k-nn-classifier-for-image-classification/` 2016.