applied sciences

MDPI

Article

Malicious PDF Detection Model against Adversarial Attack Built from Benign PDF Containing JavaScript

Ah Reum Kang [1], Young-Seob Jeong [1], Se Lyeong Kim [2] and Jiyoung Woo [1,*]

[1] SCH Media Labs, Soonchunhyang University, Asan 31538, Korea; armk@arkang.net (A.R.K.); bytecell@sch.ac.kr (Y.-S.J.)
[2] Korea Internet Security Center, Korea Internet & Security Agency (KISA), Seoul 05717, Korea; srkim@kisa.or.kr
* Correspondence: jywoo@sch.ac.kr

Received: 4 October 2019; Accepted: 6 November 2019; Published: 8 November 2019

Abstract: Intelligent attacks using document-based malware that exploit vulnerabilities in document viewing software programs or document file structure are increasing rapidly. There are many cases of using PDF (portable document format) in proportion to its usage. We provide in-depth analysis on PDF structure and JavaScript content embedded in PDFs. Then, we develop the diverse feature set encompassing the structure and metadata such as file size, version, encoding method and keywords, and the content features such as object names, keywords, and readable strings in JavaScript. When features are diverse, it is hard to develop adversarial examples because small changes are robust for machine-learning algorithms. We develop a detection model using black-box type models with the structure and content features to minimize the risk of adversarial attacks. To validate the proposed model, we design the adversarial attack. We collect benign documents containing multiple JavaScript codes for the base of adversarial samples. We build the adversarial samples by injecting the malware codes into base samples. The proposed model is evaluated against a large collection of malicious and benign PDFs. We found that random forest, an ensemble algorithm of a decision tree, exhibits a good performance on malware detection and is robust for adversarial samples.

Keywords: malicious PDF; malware; detection; machine-learning; adversarial attack

1. Introduction

Malware is evolving from an existing form of attaching an executable file to an email and then spreading malicious code using vulnerabilities of document files. Document-type malware is not an executable file itself, so it is easy to bypass existing security programs and the security programs have a high risk of false positives when detecting document-type malware. The most frequent type of malicious document type is PDF (portable document format) which is the most used document in the world. PDF supports flexibility, so attackers would exploit this flexibility. Although Adobe patches vulnerabilities in PDF, new types of malware are emerging. JavaScript is one of the most convenient flexibilities that PDF supports. JavaScript in PDF is used for changing document contents in response to some events and restricting the actions of the reader. For example, to pre-fill some form fields or to validate entered form field values. However, JavaScript in PDF can be exploited by attackers to inject malware into PDF documents.

In order to detect malicious codes embedded in document files, the document structure should be understood first. Previous studies have derived structural and metadata, while previous research has focused on extracting features from malicious document files and constructing algorithms based on the malware-centric feature set (excluding benign features). In most cases, malicious PDFs insert malicious code into stream objects because a stream object has no length constraints while other types

have some. When analysing long bytes, most of the malicious documents contain JavaScript, which is encoded in various forms to reduce the size and the high risk of being detected when the JavaScript is displayed as a bar. Thus, most of the previous studies focus on whether JavaScript is embedded or not. However, we found that not all malware contains JavaScript and a few benign PDFs contain JavaScript. This implies that benign PDFs containing JavaScript can be exploited as an adversarial attack. Furthermore, to improve previous models for higher accuracy, we need to enrich more diverse features than those introduced in previous studies through PDF structure and content analysis.

We aim to build comprehensive features encompassing structural features and content features and then we plan to test benign PDFs embedding JavaScript to determine whether the detection algorithm might be biased in detecting JavaScript. To distinguish JavaScript in malware and JavaScript in benign PDFs, we propose to use text features that can be derived from JavaScript code.

In Section 3, we describe the structure of PDF and its features, and design a different method for adversarial attacks. Section 4 shows the statistical difference of features extracted from benign and malicious PDFs and proposes a malicious PDF detection model. Section 5 contains a discussion about the results. We summarize the findings and introduce future work in Section 6.

2. Related Research

Most of the previous studies mainly depend on JavaScript. This is due to the fact that JavaScript-based exploitation is achieved in PDF. Since the PDF standards support the JavaScript, attackers can hide their attack codes in forms of JavaScript. Detection methods can be divided into static methods and dynamic methods. Static methods extract features from document structure and codes inside the document. Dynamic methods make malicious codes activate in the virtual environment set up for preventing infection and track the working process on the computer. The dynamic process requires a complex working activation process and further tracking of parts of the system affected by the malware.

From previous works, we noticed that static methods work as well with high accuracy. In this work, we reviewed static methods by not running malware hidden in the document, but by analyzing documents to extract signatures. Previous works unfolded into two stages. In the early stage, the detection method depends on metadata and structural features of documents. More advanced methods extract features from JavaScript code itself. Smutz and Stavrou [1] used the attributes of metadata and document structure. Metadata attributes include the number of stream objects, number of lowercase characters in the title, number of characters in each field, each size and position of box and image, data encoding method, object type and the number of encrypted objects. Structure features encompass Count_stream_diff (difference between the number of "stream" and "endstream" appearances), Pos_box_max (relative position of last box display), Count_font_count, Count_font, count_javascript, Count_js, Image_total px (sum of all pixels in all images), Producer_len (number of characters in metadata object), Count_obj (number of instances of "obj" indication), and PDFid0_mismatch(number of unique instances of PDFid0 value). This study proposed to use more comprehensive features than previous studies, but it does not extract features from included scripts. The highest accuracy reached was 99.8%. Šrndic and Laskov [2] developed the structural path features, that are combinations of tags such as OpenAction/JS. Their method works perfectly in detecting malicious documents. The drawback of their method is that path features are not easy to extract because every structural path should be defined manually, even if it uses a regular expression. This implies that hand-crafted features maybe not working when new-types of samples emerge.

The following works constructed features from JavaScript included in documents. Corona et al. [3] presented "Lux 0n discriminant References" (Lux0R). Their approach derives the lexical properties of the JavaScript code in forms of references of its API encompassing functions, constants, objects, methods, keywords, and attributes. After constructing the feature set, they apply machine-learning techniques to detect malicious documents. Laskov and Šrndić [4] proposed PJScan to detect a malicious PDF with JavaScript. They translated JavaScript code into lexical features like operators and extracted the

token from JavaScript code. They used token-sequence as the input of the machine-learning algorithm. Li et al. [5] extracted the PDF header, object information, cross-references, and trailer information, and further developed features such as obfuscation, loops in JavaScript, obsolete information in JavaScript, and trailer-specific patterns. Their work has significance in deciphering obfuscated parts and an in-depth analysis of the contents of JavaScript. However, their work output has an accuracy of 95%, which is relatively lower than previous methods even in smaller samples.

Regarding the adversarial attacks, we reviewed the following key papers. Adversaries subtly alter legitimate inputs to induce the trained model to produce erroneous outputs [6]. Previous research shows that machine-learning algorithms can be vulnerable to adversarial samples. The research on adversarial attacks can be folded into theoretical approaches and practical approaches. The theoretical approaches design a generative model that can defeat the detection model and derive an optimal solution for a generation. The practical approaches directly generate adversarial samples that can detour the detection model.

To defend against the adversarial attacks, adversarial sample detection models are proposed [7]. The method to build additional classifiers that distinguish between clean and adversarial samples [7,8] and the method to reconstruct an existing model to detect adversarial samples, are proposed [9]. Shumailov et al. proposed a method to train the model without adversarial examples and detect unexpected behaviors of the model by feeding adversarial examples [7]. As shown in Papernot and MdDanile's work, black box attacks are proposed as a generative model for various machine-learning models respectively [6] and showed adversarial sample transfer abilities.

Smutz and Stavrou [1] performed adversarial analysis by artificially reducing the influence of the top features derived from the random forest algorithm. It is possible to make it similar to a positive document. If the number of fonts in the document is set to be higher in the malicious document, the method can be bypassed. Šrndic and Laskov [2] modified malicious samples to mimic the most benign samples with the lowest classification accuracy. Maiorca, Corona, and Giacinto [10] proposed a reverse mimicry, which manipulates the benign files to defeat the detection model. They designed EXE Embedding that embeds the simple payload for compression, PDF Embedding which injects PFD file containing malicious codes into files, and JavaScript Injection which directly injects the JavaScript. They found that the detection model focusing on the structural feature could be evaded by the mimicry attack.

Through the literature review, we found some shortcomings of previous works. First, most studies focus on either structure or metadata of document or content feature of JavaScript. Combining the two could improve performance. In addition, previous methods missed semantic features from JavaScript. Corona et al. [3] only used lexical features assuming that semantic features would not be sufficient in detecting malicious documents and they admitted that semantic meaning will disappear when tokening. However, we believe that semantic features would be helpful in discriminating malicious and benign documents both containing JavaScript because words in these two types of documents are different. Last, none of the previous works evaluated proposed methods against benign samples containing multiple JavaScript and adversarial samples originated from them.

In this work, we set aside complex theoretical and methodology issues. Instead, we propose to construct adversarial samples as similar as possible to malware for a practical purpose. It is still an important and challenging problem to implement real-world evasion relaxed in the requirement of an optimal solution of the theoretical formulation [11].

Then, we explore various models varying the feature set and figure out which model is robust for adversarial samples. This work contributes to the practical implementation of the adversarial attacks for malicious PDF detection models based on machine-learning approaches and gives implications for detection model builders.

In this work, we propose a malicious PDF detection method encompassing structural features and semantic features from JavaScript in malware and benign PDF as well. We will build adversarial samples to detour the detection model by inserting malicious codes in a benign PDF containing

JavaScript. We will then evaluate the proposed method against large-scale data collected over a long period until recently and adversarial samples.

3. Malicious Portable Document Format (PDF) Detection Model

3.1. Research Methodology

We perform a PDF structure analysis with the benign and malicious PDF documents downloaded from Contagio malware dump [12] and identify the differences between them. We build a virtual environment to figure out how malicious PDFs work and how JavaScript is exploited. We extract JavaScript code from all malicious PDF documents and figure out the essential functions and keywords that appear most often. Keywords are retrieved based on the number of appearances in JavaScript. Since there is no JavaScript in the benign PDF document obtained by Contagio, we collected additional benign PDF documents containing JavaScript from the Internet. We compare the keywords that appear in the JavaScript code extracted from the benign and malicious PDF documents and identify the differences. This method of deriving text features is improved from previous works and is expected to increase detection accuracy. To validate our proposed model, we design an adversarial attack by injecting malicious codes into benign samples containing JavaScript. To maximize the possibility that adversarial attacks might bypass the detection algorithm, we collect the benign samples containing multiple JavaScript in them and injected malicious codes into these collected samples. We install Windows 7 in VMware Workstation and use Notepad++ editor and OllyDBG for the PDF analysis. We manually extract malicious codes from malicious PDF samples and inject these malicious codes into benign samples.

3.2. Structure of PDF

PDF provides various functions other than those frequently used by normal users, one of which is its ability to add objects. Thus malicious code often exploits them. PDF has the following components.

- Object: PDF consists of data objects.
- File Structure: File structure contains object storage, access, and update information.
- Document Structure: Document structure refers to how various object configurations organize and put the document.
- Content Streams: Content streams contain the appearance of the document, and graphical elements.

PDF file structure is organized with header, Body, Cross-reference table, and Trailer as shown in Figure 1.

Figure 1. Portable document format (PDF) file structure.

- Header: One-line header to inform the PDF specification version.
- Body: A body contains objects that make up the PDF document. It consists of indirect objects that contain the actual content of the document. Body represents document content, font, page, and image elements.
- Cross-reference (xref) table: It is the table used for referencing objects. It usually has a form of the global reference table referencing indirect objects. It has position information of each indirect object and enables random access of a specific object. Each entry is 20 bytes long, including an end-of-line marker.
- Trailer: It indicates which one is the Root object among objects in the body part and where the cross-reference table is located. It starts with the directive trailer and ends with % EOF (End of File). It represents the characteristics of objects wrapped with << >>. In Figure 2,/Size represents the number of cross-reference table entries, and/Root represents i.d. of the Root object. "startxref" tells the location of the cross-reference table.

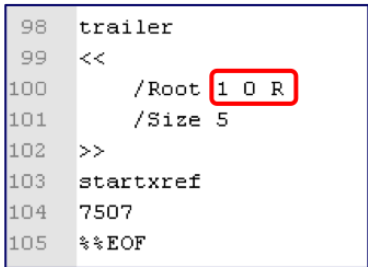

Figure 2. A snapshot of a trailer of PDF.

Figure 2 shows an example of a trailer, located at the end of the file and shows the location of the root object and the cross-reference table. The contents inside the red box refer to the offset 0 of the first object. R stands for reference. Figure 2 shows that the root object is number 1, the number of cross-reference table items is 5, and the starting offset value of the cross-reference table is 7507.

Figure 3 shows an example of a cross-reference table, which shows the position values of objects in the PDF Body part and whether they are used or not. Each entry in the table has a size of 20 bytes, including spaces. The keyword xref indicates the beginning of the cross-reference table. "f" indicates free entry and "n" indicates in-use entry. The number 0 of line 92 represents the object start number, and the number after the object number 5 represents the number of entries. Line 93 indicates object ID 0, offset 0, disabled and line 94 indicates object ID 1, offset 10. Line 95 indicates that object ID 2 is used and offset 120 is used. Line 96 indicates object ID 3 and offset 189. Line 97 is object ID 4, and offset 408.

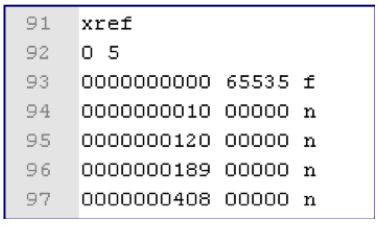

Figure 3. A snapshot of a cross-reference table.

Figure 4 is an example of a body. In the red box, object ID 1 refers to the catalog type, whose page layout refers to a single page, page contents refer to object ID 2, an action refers to object ID 4. In the blue box, object ID 3 indicates that its type has a square picture including page content and media contents, and additionally includes object ID 6 and 8, and object ID 2 as a parent object. Among various types of objects, the stream object is a group of consecutive bytes (binary data), which is a large

size object. This object consists of a Dictionary object followed by several bytes between the keywords 'stream' and 'endstream'.

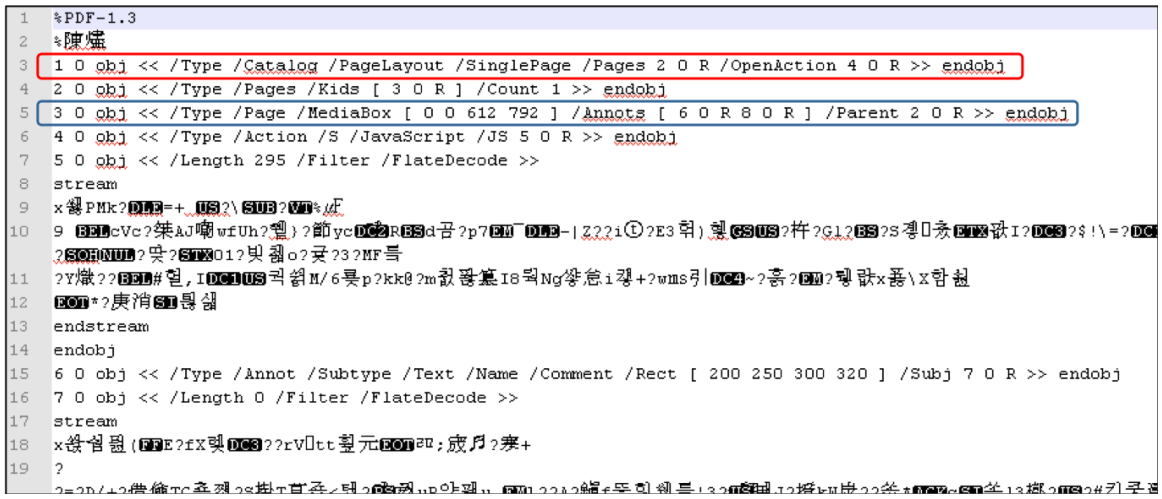

Figure 4. A snapshot of the body.

3.3. Feature

In order to use machine-learning algorithms, it is important to extract features that can characterize the malicious documents and benign document. First, we examined the structural differences between benign PDFs and malicious PDFs using PDFStreamDumper and HxD. PDFStreamDumper is a representative tool for malicious code analysis. It allows users to parse and analyze PDF files in a low format. HxD is a hexadecimal editor that displays both text and hexadecimal numbers simultaneously. The big difference is that benign PDF files contain a much larger number of objects and streams than malicious PDF files as shown in Figures 5 and 6. Malicious document files contain, on average, 90 objects and 31 streams, while benign document files have 11 objects and 3 streams. Second, most malicious PDF files contain JavaScript, while most benign PDF files do not contain JavaScript. Thus, whether or not PDF files contain JavaScript is an important feature for classifying positive and malicious files.

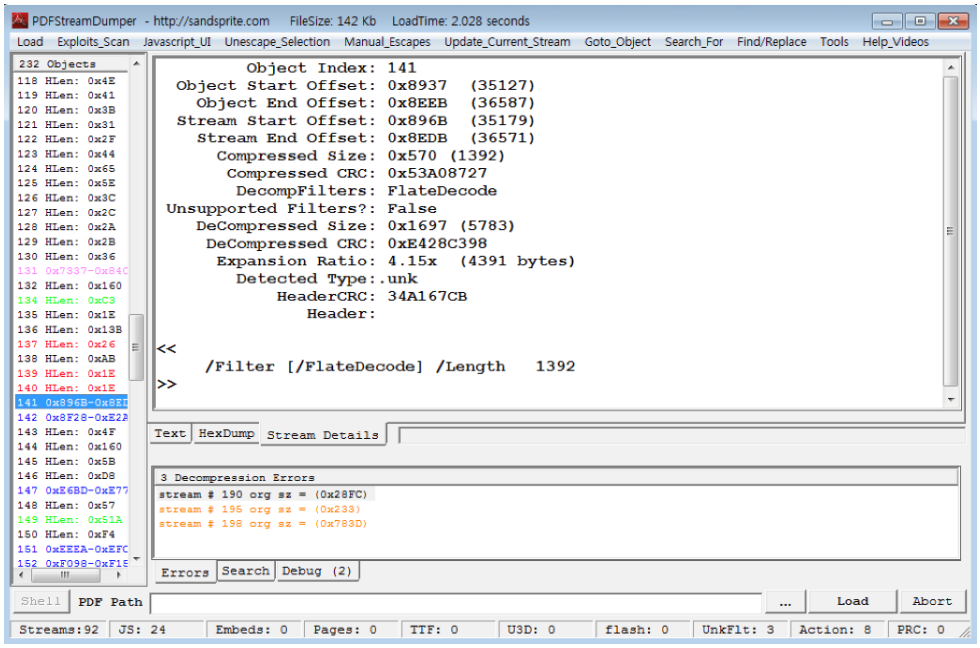

Figure 5. A benign PDF file opened by PDFStreamDumper.

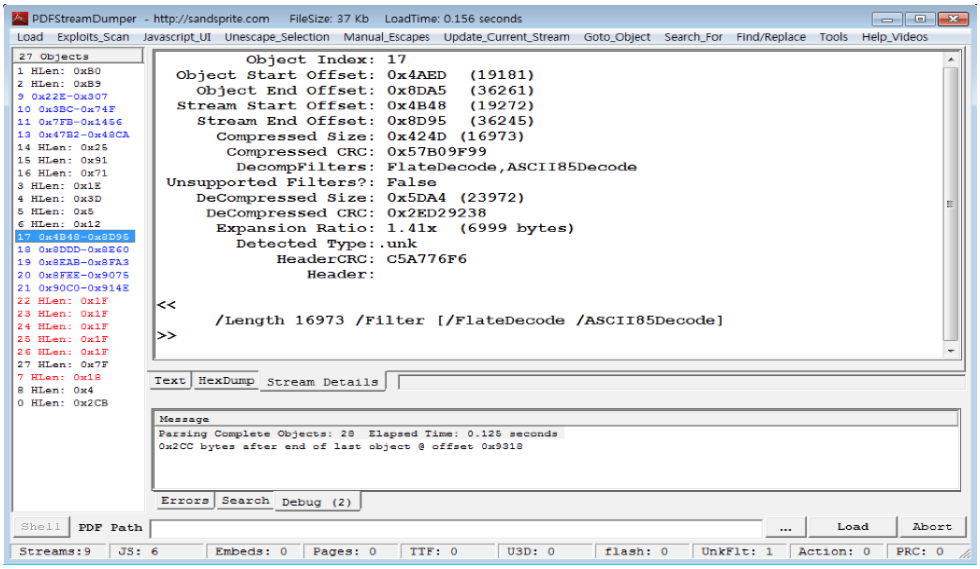

Figure 6. A malicious PDF file opened by PDFStreamDumper.

The JavaScript code is contained in an object of the stream type that is not limited in size, unlike other object types. Most JavaScript codes are obfuscated and encoded by various filters. Encoded JavaScript codes have filters to decode themselves and activate malicious actions. Most JavaScript codes hidden in malware have applied a FlateDecode filter. When the PDF viewer application is executed, it decodes the stream to which the filter is applied. The PDFStreamDumper tool automatically decodes various filters (including FlateDecode, ASCIIHexDecode, and ASCII85Decode) supported by the PDF application to display the plain text of JavaScript. According to the observation of samples, streams of benign PDF files have a single filter, while streams of malicious PDF files often have multiple filters.

Figure 7 shows raw bytes and text strings of a benign PDF file in Figure 5 using the HxD tool. Object 140 specifies that Object 141 is JavaScript code. Object 141 indicates that the encoding type is FlateDecode and the length of the stream is 1392 bytes. Figure 8 shows a JavaScript code, the decoded object of Figure 6 with the FlateDecode filter using the PDFStreamDumper tool.

```
Offset(h)  00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F  Decoded text
00008900   0A 31 34 30 20 30 20 6F 62 6A 0A 3C 3C 2F 53 20  .140 0 obj.<</S
00008910   2F 4A 61 76 61 53 63 72 69 70 74 20 2F 4A 53 20  /JavaScript /JS
00008920   31 34 31 20 30 20 52 3E 3E 0A 65 6E 64 6F 62 6A  141 0 R>>.endobj
00008930   0A 31 34 31 20 30 20 6F 62 6A 0A 3C 3C 2F 46 69  .141 0 obj.<</Fi
00008940   6C 74 65 72 20 5B 2F 46 6C 61 74 65 44 65 63 6F  lter [/FlateDeco
00008950   64 65 5D 20 2F 4C 65 6E 67 74 68 20 20 20 31 33  de] /Length   13
00008960   39 32 3E 3E 73 74 72 65 61 6D 0A 48 89 CC 57 6D  92>>stream.H‰ÌWm
00008970   6F DB 36 10 FE AC 00 FE 0F 9C D0 01 52 EC CA B1  oÛ6.þ¬.þ.œÐ.RìÊ±
00008980   5D A4 83 8D 60 28 E2 25 EB 90 65 41 9C 22 40 93  ]¤ƒ.`(â%ë.eAœ"@"
00008990   6C E0 2C DA 22 26 8B 82 48 C7 4E D3 FC F7 1D 49  là,Ú"&‹‚HÇNÓü÷.I
000089A0   BD DA A2 E2 B4 DD 8B 12 04 0A EF 9E BB E3 F1 EE  ½Ú¢â´Ý‹...ïž»ãñî
```

Figure 7. Raw bytes and text strings of a normal PDF file displayed by HxD.

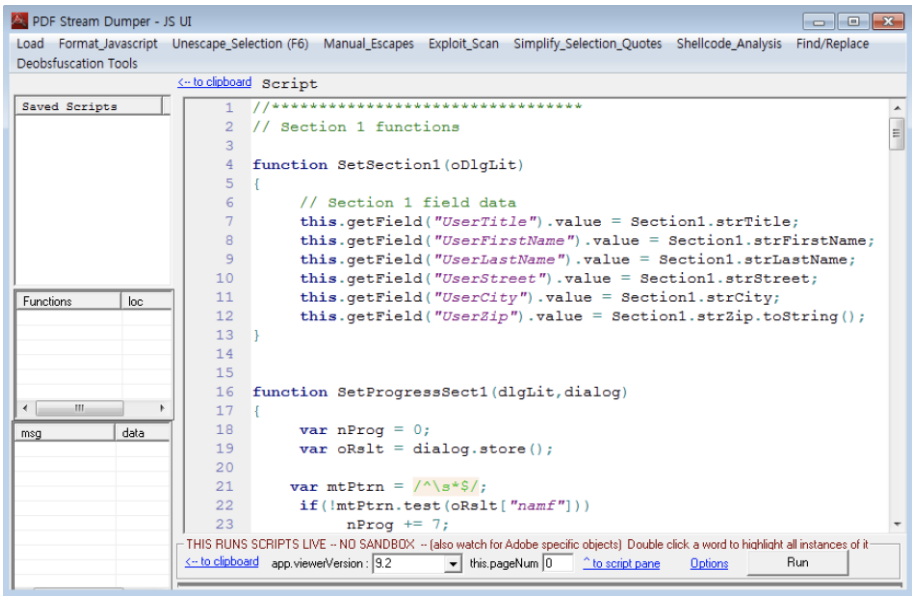

Figure 8. Object 141 decoded by PDFStreamDumper.

In addition, we observed that there were differences between the functions that the benign and malicious documents called. Malicious JavaScript code embedded in PDF documents typically exploits vulnerable PDF viewer applications to execute malicious code payloads or download and run instances of a specific malicious code. For that, malicious JavaScript code uses the eval function, a method related to execution. It also calls the replace or unescape functions that are methods for obfuscation or encoding as shown in Figure 9.

```
var vvv = z(unescape(xxx));
eval(vvv);
var g = 'lace(/lka';
var f = '1/g,prcnt)';
var z = unescape;var prcnt = '%';
var ks = eval;
ks('var filoos = this.in'+z("%66%6f%2e%61%75%74")+'hor;');
var p = 'filoos.rep';
var pgf = p + g + f;
var xxx = ks(pgf);
```

Figure 9. The usage of replace and unescape function in malicious JavaScript content.

In some cases, malicious JavaScript codes contain meaningless text, as shown in Figure 10.

```
var sMu1vDW5=21689;var out = '' + tookNeedYou+
theorySinceExact+
theirDolorHere+
hOnMake+
usedSomeWhen+
withHDummy+
makeButPage+
dummyHasGalley+
…
var jt3iFLWSfM='jt3iFLWSfM';var tookNeedYou = 'var d';var uwyJ1Arp=false;
var theorySinceExact = 'BrdWh';this.j6cf3pENR="j6cf3pENR";
var theirDolorHere = 'xWSu ';function kdMaFEH2P(kdMaFEH2P){return true;}
var hOnMake = '= new';this.rHOQIQPtA="rHOQIQPtA";
var usedSomeWhen = ' Arra';function iOnE9Q6n(iOnE9Q6n){return iOnE9Q6n;}
var withHDummy = 'y();v';this.qoajDB89=4716;
var makeButPage = 'ar yr';var a4Vs0lsG=false;
```

Figure 10. Example of malicious JavaScript content.

Based on what we observed on the samples, we created a keyword list that appears primarily in malicious PDF files. Next, we count the number of matching keywords from text data in each PDF file. We also extract the number of methods and object names and the number of readable strings that appear frequently in malicious PDFs from the decoded JavaScript code.

3.4. Adversarial Attack

As mentioned in the Introduction, signature-based methods are vulnerable when signatures are known. Machine-learning methods have a lower risk than signature-based method even when features adopted in the model are open because feature combination is used and the combination method is hard to be known. White-box algorithms, for instance, tree-based ones, are vulnerable when features and their combination rules are known. However, a complex tree model has a low risk in revealing combination rules and black-box algorithms are more robust in such a case. Adversarial attacks are designed to paralyze the machine-learning approaches. White box-type machine-learning algorithms are vulnerable to adversarial attacks. Even black-box type algorithms are no exception, adversarial examples for a black-box algorithm such as neural network and deep-learning algorithms are proposed [13,14]. Adversarial examples targeting black-box algorithms simplify the check of algorithm performance by deleting a variable. It is impossible that attackers have full access or knowledge to parameters of the machine-learning model. Thus, it is enough to check the detection performance by deleting variables in a gradient way. When features are diverse, it is hard to develop adversarial examples because small changes are robust for the machine-learning algorithm while the detection accuracy will definitely decrease when a small number of features are used. Rich feature sets derived in our work would be working robustly when we delete some important variables.

Previous work [1] focused on developing adversarial attacks to obtain privilege on the detection model by deleting features from malware. They modified the malicious documents deliberately to make them look similar to benign documents, normalizing some features and keeping the malware contained.

In this work, we design a different method for adversarial attacks by creating samples mimicking benign samples the most similar to malware. We collected benign documents containing JavaScript for the base of adversarial samples. These samples are much different from benign documents collected by antivirus software companies. Benign documents collected by antivirus software companies or used in previous studies do not contain JavaScript. There is a risk of misclassifying benign documents with JavaScript as malicious. Therefore, it is necessary to train and test with benign documents containing JavaScript. We injected malware codes into benign samples containing JavaScript. Definitely, these adversarial samples include characteristics of benign samples and malicious JavaScript codes.

We build a model to detect benign samples with JavaScript and adversarial samples as well. For that, we derive diverse feature set encompassing structure/meta feature and content features and make the algorithm learn those features as well.

Adversarial attacks are classified into evasion and positing attacks [15]. Our approach confirms to evasion attack to detour the detection model. We leave the positing attack method that poison the training dataset as future work. We could develop a positing attack model by feeding the adversarial samples into anti-virus software built from machine-learning algorithms. Our approach does not require the attackers' knowledge of the classification algorithm. In this work, we did not consider the game-theoretical model that can change the model interactively reflecting the influence of the attack.

3.5. Machine-Learning Approaches

Representative algorithms among popular machine-learning algorithms used for the classification of different operating methods are selected. We used the naïve Bayes, random forest, and support vector machine. Naïve Bayes is an algorithm that assumes the independence of features, computes the probability of occurrence of two classes in the training data and derives the product of the probability of occurrence obtained for the characteristic of the new data. The support vector machine (SVM) finds a linear hyperplane that divides two classes of data by finding a decision plane that is the furthest

from the two classes of data. SVM can be used for classification and prediction problems at the same time, and its accuracy of prediction is high [16]. It is also known to have the ability to handle large numbers of features. Compared to the neural network technique, however, there is a disadvantage in that the model building time is long. Another disadvantage is that users cannot infer why this black-box algorithm gives this output. The random forest algorithm is an ensemble method that learns randomly several decision trees. The CART (Classification and Regression Tree) method is used to train several decision trees as part of the training data. During the classification, all trees are voted, and a single value with many votes is drawn as a result. The random forest algorithm has strong generalization ability and strong robustness against data noise.

Smutz and Stavrou's work [1] is similar to our study except that we extract text features in the encoding script. In their work, the detection rate was lower than 60% except for the random forest method.

4. Experiment and Results

4.1. Dataset

The PDF files used in this study consist of 11,097 malicious document files, 9000 benign document files collected by Contagio malware dump, from November 2009 to June 2018. Contagio malware dump site provides a collection of the malware samples. Any researcher can download malware samples from the site. Our samples have a long coverage on the period. To design an adversarial attack for the validation, we collected 115 clean files containing JavaScript files in separate. We successfully injected malicious codes into 101 clean files except for encrypted files.

4.2. Feature Extraction

We first performed a dynamic analysis to identify the structural and functional differences between benign and malicious documents in a virtual machine. We found metadata, encoding patterns, and the differences in functions and keywords used in JavaScript between benign and malicious documents. We developed a comprehensive feature set encompassing the structure feature, metadata, and JavaScript content features. Below, we summarized the types of features used in our model and the characteristics of malicious document files, benign document files, and benign files containing JavaScript and adversarial examples. Analysis results of malware and benign samples exhibits differences in the size and the existence of JavaScript. Benign PDFs generally have big size and do not include JavaScript in them.

4.2.1. Structure and Metadata

File size: The malicious document file had an average size of 23 KB. The average size of benign documents was 95 KB, and that of benign documents including JavaScript was 568 KB. Malicious document files were found to be relatively small in size.

File version: We counted the number of the usage of PDF versions from the samples and found that older versions were widely used for malicious documents. PDF 1.3 version was found the most in the malicious document files as shown in Figure 11.

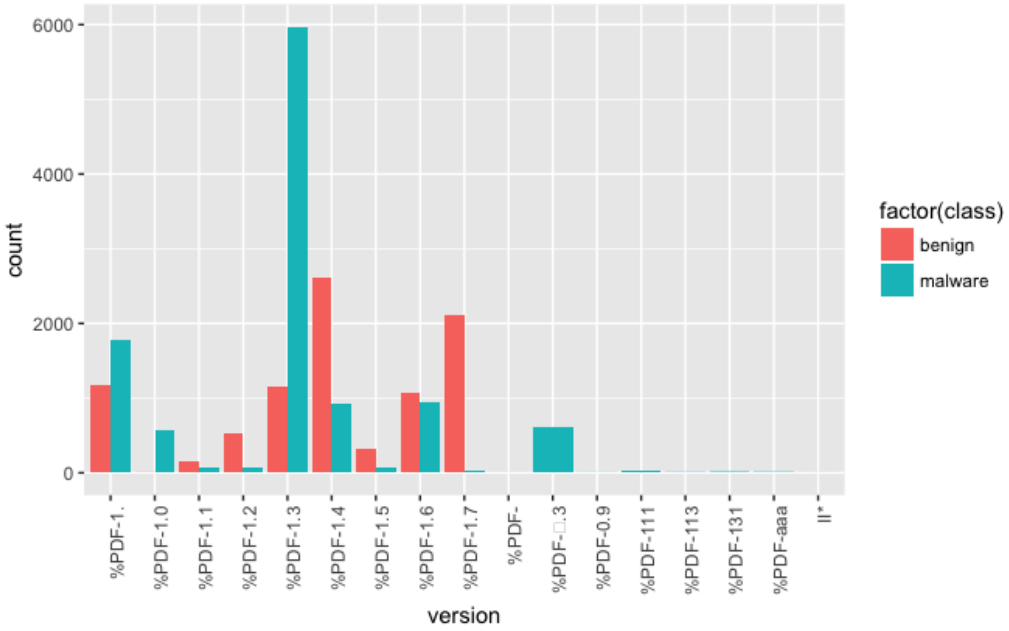

Figure 11. The numbers of usage of PDF versions.

File encoding to embed JavaScript: JavaScript enables attackers to trigger the vulnerability and activate malicious codes hidden in it. To encode JavaScript, keywords such as ASCIIHexDecode, FlateDecode, and ASCII85Decode are used in PDF.

Our samples used mainly three encoding methods. We built a dummy variable that checks if a filter that decodes data encoded in ASCII hexadecimal format was applied. The other two decoders were also checked. The stream object was designed to contain image files and the information on page structure, which are mostly encoded beforehand. A document may include several objects. Each object can be encoded in different formats. Furthermore, several encoding methods could be applied overlay. Table 1 shows the number of objects applied by each encoding method over our dataset.

Table 1. The number of times that an encoding method is applied over all documents per class.

Variables	Malware	Benign	Benign with JavaScript (Base of Adversarial Sample)
ASCIIHexDecode	376	0	0
FlateDecode	8293	0	11
ASCII85Decode	145	0	0
Unencoded JavaScript	9824	0	0

Keywords in PDF: From previous works [17] and also from observations on our samples, we selected 56 features written in PDF. Table 2 shows the average occurrence of keywords per document. All keywords are listed up in the PDF manual [5,17,18]. Kittilsen's work addresses the object type, JavaScript embeddedness, non-JavaScript vulnerability, and non-vulnerability functions such as OpenAction and AA (Additional Action) [19,20]. The non-vulnerability features include functionality to open other documents or hyperlinks, access resources outside the active document, and execute applications [20].

Table 2. The average count of each keyword appearance per a document.

Keywords	Malware	Benign	Benign with JavaScript
Obj: object starts	11	90	180
Stream : stream starts	3	31	96
/Page : pages,# in PDF	0.7	2	1.8
/JS: JavaScript starts	0.7	0.01	1.1
/JavaScript: JavaScript starts	0.9	0.03	1.3
/AA: autorun	0.01	0.1	0.9
/OpenAction: launch something without user interaction	0.4	0.06	0.1
/Type: object type information	3	5.6	29
/Filter: filter pages within a PDF by a word	2	5	26
/FlateDecode : to access the decompressed stream	1	5	33
/Length: page length	3	13	25

4.2.2. JavaScript Features

We applied text-mining techniques to derive keywords from JavaScript embedded in malicious PDF. The decoder filter was adopted to make JavaScript readable. JavaScript is different from plain text. It has numerous special characters like operators, and quotations. Full references on keywords used in JavaScript are referenced in the official JavaScript website [21]. We extracted keywords based on the frequency of randomly selected samples from our entire sample set. Then, we categorized the keywords derived from JavaScript into three categories as follows.

The first category is predefined names including the method name, the object, and the attribute name. Keywords in this category include length, replace, String, fromCharCode, unescape, split, eval and so on. We compared the cumulative sum of occurrences of the name of keywords per class shown in Table 3.

Table 3. The average count of names in JavaScript.

Predefined Names	Malware	Benign	Benign with JavaScript
length: variable length	9.30	0.11	2.20
replace: Replace text in strings with regular expressions or search strings	0.83	0.04	0.18
String: variable type	0.70	0	1.55
fromCharCode: Returns a string from multiple Unicode character values.	0.37	0	0
unescape: Decodes a String object encoded with the escape function escape.	0.47	0	0
split: Splits a string into substrings using the specified delimiter and returns it as an array.	0.32	0.06	0.42
eval: This makes JavaScript source code being executed dynamically.	0.20	0.01	0.67
Array: variable type	0.28	0.01	0.57
substr: Returns the substring at the specified position in the string object.	0.80	0.02	0
charCodeAt: Returns the Unicode value of the character at a specified position.	0.05	0	0
Date: variable type	0.05	1.05	2.09
Object: variable type	0.04	0.02	1.39
charAt: Returns the character value corresponding to the specified index.	0.04	0	0
Push: Adds a new element to the array and returns the new length of the array.	0.03	0.01	0.83
indexOf: Returns the start of a substring in a String object.	0	0	0.33
Number: variable type	0	1.47	2.57

In Table 4, the second category includes the operator, constant, type and reserved words. These reserved words include var, if, this, function, return, for and so on.

Table 4. The average count of each reserved word in JavaScript.

Reserved Words	Malware	Benign	Benign with JavaScript
var	7.72	0.36	12.76
if	6.51	7.50	19.59
this	0.91	9.25	24.90
function	1.56	0.16	5.03
return	1.28	4.39	2.96
for	1.31	30.74	21.35
null	1.96	10.26	36.55
false	0.49	1.00	5.70
new	0.45	1.18	2.40
true	0.61	1.32	7.37
while	0.41	0.27	0.44
try	0.05	0.05	0.77
catch	0.05	0.04	0.41

To enrich lexical features such as app, fnc, plugIns, buf, sum, arr, num, doc, and so on known to be readable with meaning in human language, we extracted readable words frequently shown in JavaScript of documents. Table 5 demonstrates the number of appearances of a few frequent words in JavaScript.

Table 5. The average count of top readable words in JavaScript.

Readable Strings	Malware	Benign	Benign with JavaScript
app	2.40	0.46	7.19
fnc	1.16	0	0
plugIns	0.95	0	0
buf	0.90	0	0.01
sum	0.90	0.32	0.38
arr	0.82	0.02	0.14
num	0.69	0.25	0.24
doc	0.69	0.26	0.21
nPage	0.52	0	0
getAnnots	0.52	0	0
subject	0.38	2.49	2.01
syncAnnotScan	0.38	0	0
proc	0.17	0.13	0

We selected the keywords based on the frequency appeared in randomly chosen malicious documents and manually classified into “names”, “reserved words”, and “readable words” using JavaScript programming skills. In the detection model, it is not necessary to categorize keywords because we consider all keywords equally. In this work, we selected 1% samples from entire samples and selected keywords that appeared five times, resulting in 77 keywords. We only examine around 1% samples from the entire sample set and will figure out the feature set constructed using a few samples that can be applied to the remaining samples.

4.3. Machine-Learning Algorithm

Naïve Bayes and SVM are validated with 10-fold cross-validation. K-fold cross-validation is a re-sampling method used to increase the statistical reliability of the classifier performance measurement when the amount of data is insufficient. The random forest has a cross-validation method in it so that

it can form an ensemble tree with different subsamples and tests the ensemble tree model with the remainder. Thus, there is no need to perform cross-validation separately.

We first built two models. One is built using a structure feature set and the other is built using structure and content feature set. Precision, recall, and F-measure were checked for each class. We then tested the proposed model against the base samples (benign samples containing multiple JavaScript) and the adversarial samples (variants of base samples after injecting malicious codes).

In Tables 6 and 8, 11,097 malicious PDFs and 9000 benign PDFs obtained from Contagio malware dump were used for training and testing. In Tables 7 and 9, 115 benign PDF documents included JavaScript and 101 malicious PDF documents injected malicious code were used for the test.

4.3.1. Structure Feature Model

We firstly built a detection model with the structure and metadata. We then tested the base samples and the baseline model. As shown in Table 6, the random forest achieved the highest accuracy while the naïve Bayes model showed fair performance. Table 7 shows the results of classifying base samples and adversarial samples into this detection model with the structure and metadata. For the base samples, RF and naïve Bayes models showed good performance. However, the machine-learning algorithms using the structure feature set were vulnerable for adversarial samples.

Table 6. The detection performance of machine-learning algorithms with the structure feature model.

Algorithm	Class	Precision	Recall	F-Measure
Random Forest	Benign	0.997	1.000	0.998
	Malware	1.000	0.998	0.999
Naïve Bayes	Benign	0.970	0.678	0.798
	Malware	0.790	0.983	0.876
Support Vector Machine	Benign	0.978	0.996	0.987
	Malware	0.997	0.982	0.989

Table 7. Classification result of base samples and adversarial samples with the structure feature model.

Class	Algorithms	Benign	Malware
Base samples (Benign)	Random Forest	111	4
	Naïve Bayes	111	4
	SVM	74	42
Adversarial samples (Malware)	Random Forest	12	89
	Naïve Bayes	86	15
	SVM	13	88

4.3.2. Full Feature Model

We then developed a model using the full feature set. The naïve Bayes model achieved much improvement as shown in Table 8. Table 9 shows the results of classifying base samples and adversarial samples into the detection model with the full features. However, it is still vulnerable to adversarial attacks. The performance of RF was also improved. RF was found to be robust for an adversarial attack. On the other hand, naïve Bayes and SVM were vulnerable to adversarial attacks. Especially, SVM was not improved at all even using more diverse features. We concluded that the model using the random forest algorithm with structure and content features is the best in malware detection and robust for adversarial samples.

Table 8. The detection performance of machine-learning algorithms with the full feature model.

Algorithm	Class	Precision	Recall	F-Measure
Random Forest	Benign	0.997	1	0.998
	Malware	1	0.997	0.999
Naïve Bayes	Benign	0.878	0.929	0.903
	Malware	0.94	0.895	0.917
SVM	Benign	0.976	0.999	0.987
	Malware	0.999	0.98	0.989

Table 9. Classification result of base samples and adversarial samples with the full feature model.

Class	Algorithms	Benign	Malware
Base samples (Benign)	Radom Forest	113	2
	Naïve Bayes	114	1
	SVM	73	42
Adversarial samples (Malware)	Random Forest	2	99
	Naïve Bayes	58	43
	SVM	13	88

5. Discussion

In this study, we explained the PDF document structure precisely and identified the structural, meta and content characteristics of PDF document. The method and the tool to analyze the document are described in detail to enable readers to duplicate the malicious PDF analysis. Through the analysis of a large collection of PDF documents, two basic PDF file information, four PDF file encoding and four JavaScript insertion information, and 43 keyword information in PDF file are extracted.

To enrich the feature set, we further deciphered the JavaScript contained in the stream object. When JavaScript is inserted, 77 names such as method names, object names, attribute names, reserved words including operators, constants, and types, and readable strings in the script were found in the malicious codes. The number of their occurrences was used as a feature. We categorized these features according to components of JavaScript language and further developed a semantic feature set deriving readable strings. This attempt at categorization would make a thorough derivation of the feature from JavaScript and the novel feature set, semantic features excluded in previous works, could improve the detection performance.

Using a total of 126 features, we constructed a machine-learning model that could distinguish between benign and malicious documents. We tested the black-box type algorithm to minimize the risk of revealing important features. Experimental results showed that three algorithms have good performance for malware and benign samples in both cases of using only the structure feature set and using the content feature set as well. However, the detection model with the structure feature set has a fair performance for base samples that were benign samples containing multiple JavaScript and the adversarial samples that were variants of base samples after the injection of malicious codes.

Regarding the machine-learning algorithm perspective, we found that traditional machine-learning algorithms are working well enough for malware detection, however, the performance decreases for adversarial samples except for the random forest algorithm. The random forest algorithm might have a good performance because of this transferability. Adversarial sample transferability is defined as the property that adversarial samples produced to mislead a specific model that can mislead other models [6]. This property is important in terms of the practical impact of the adversarial attack. In this work, we check the adversarial sample transferability using the random forest model, which trains the model constructing the ensemble trees with the subset of samples and features and validates the built tree with unused samples.

There is still room for improvement for the proposed algorithm. In our study, we only used keywords found in PDF, but there are various malwares that use JavaScript. If we collect additional keywords from other malware that uses JavaScript, we will be able to detect advanced attacks.

6. Conclusions

We develop a malicious PDF detection model using black-box type models with the structure and content features to minimize the risk of adversarial attacks. Since most malicious PDFs use JavaScript, the usage information of JavaScript is an important feature that separates malicious from normal. However, the normal usage of JavaScript used to pre-fill document forms or validate input values should not be classified as malicious. We collected normal documents that used JavaScript to distinguish between normal and malicious usage of JavaScript. We focused on JavaScript code content to improve the previous work that is too heavily focused on whether or not JavaScript is included. We also created adversarial samples of manually inserted malware to make sure that the proposed model distinguished them in case of abuse of a document with normal JavaScript. We found that the model using the random forest algorithm with structure and content features is the best in malware detection and robust for adversarial samples.

As future research, we are planning to advance the adversarial attack for deep validation on the proposed model. We will enlarge the adversarial samples by injecting malicious codes to benign samples without JavaScript and check the robustness of the proposed model. To defend against the advanced adversarial attack, we will develop an advanced semantic feature set to have a comprehensive meaning.

Author Contributions: Conceptualization, A.R.K. and J.W.; Data curation, A.R.K.; Formal analysis, A.R.K.; Investigation, A.R.K. and S.L.K.; Methodology, A.R.K. and J.W.; Project administration, A.R.K.; Supervision, J.W.; Writing – original draft, A.R.K. and J.W.; Writing – review & editing, A.R.K., Y.-S.J. and J.W..

Funding: This work was supported by Institute of Information & communications Technology Planning and Evaluation (IITP) grant funded by the Korea government (MSIT) (No.2019-0-00026, ICT infrastructure protection against intelligent malware threats) and partially supported by the Soonchunhyang University Research Fund under Grant 20180119.

Conflicts of Interest: The authors declare that there is no conflict of interest regarding the publication of this paper.

References

1. Smutz, C.; Stavrou, A. Malicious PDF detection using metadata and structural features. In Proceedings of the 28th Annual Computer Security Applications Conference, Orlando, FL, USA, 3–7 Dcember 2012; ACM: New York, NY, USA, 2012.
2. Šrndic, N.; Laskov, P. Detection of malicious pdf files based on hierarchical document structure. In Proceedings of the 20th Annual Network & Distributed System Security Symposium, San Diego, CA, USA, 24–27 February 2013.
3. Corona, I.; Maiorca, D.; Ariu, D.; Giacinto, G. Lux0r: Detection of malicious pdf-embedded javascript code through discriminant analysis of api references. In Proceedings of the 2014 Workshop on Artificial Intelligent and Security Workshop, Scottsdale, AZ, USA, 7 November 2014; ACM: New York, NY, USA, 2014.
4. Laskov, P.; Šrndić, N. Static detection of malicious JavaScript-bearing PDF documents. In Proceedings of the 27th Annual Computer Security Applications Conference, Orlando, FL, USA, 5–9 Dcember 2011; ACM: New York, NY, USA, 2011.
5. Li, M.; Liu, Y.; Yu, M.; Li, G.; Wang, Y.; Liu, C. FEPDF: A Robust Feature Extractor for Malicious PDF Detection. In Proceedings of the 2017 IEEE Trustcom/BigDataSE/ICESS, Sydney, Australia, 1–4 August 2017; IEEE: Piscataway, NJ, USA, 2017.
6. Papernot, N.; McDaniel, P.; Goodfellow, I. Transferability in machine learning: from phenomena to black-box attacks using adversarial samples. *arXiv* **2016**, arXiv:1605.07277.
7. Shumailov, I.; Zhao, Y.; Mullins, R.; Anderson, R. The Taboo Trap: Behavioural Detection of Adversarial Samples. *arXiv* **2018**, arXiv:1811.07375.
8. Lu, J.; Issaranon, T.; Forsyth, D. Safetynet: Detecting and rejecting adversarial examples robustly. In Proceedings of the IEEE International Conference on Computer Vision, Venice, Italy, 22–29 October 2017.

9. Grosse, K.; Manoharan, P.; Papernot, N.; Backes, M.; McDaniel, P. On the (statistical) detection of adversarial examples. *arXiv* **2017**, arXiv:1702.06280.
10. Maiorca, D.; Corona, I.; Giacinto, G. Looking at the bag is not enough to find the bomb: an evasion of structural methods for malicious pdf files detection. In Proceedings of the 8th ACM SIGSAC Symposium on Information, Computer and Communications Security (ACM), Hangzhou, China, 7–10 May 2013.
11. Huang, L.; Joseph, A.D.; Nelson, B.; Rubinstein, B.I.; Tygar, J.D. Adversarial machine learning. In Proceedings of the 4th ACM workshop on Security and Artificial Intelligence, Chicago, IL, USA, 21 October 2011; ACM: New York, NY, USA, 2011.
12. Mila, P. Contagio Malware Dump. Available online: http://contagiodump.blogspot.com (accessed on 5 June 2018).
13. Hu, W.; Tan, Y. Generating adversarial malware examples for black-box attacks based on GAN. *arXiv* **2017**, arXiv:1702.05983.
14. Goodfellow, I.; Pouget-Abadie, J.; Mirza, M.; Xu, B.; Warde-Farley, D.; Ozair, S.; Courville, A.; Bengio, Y. Generative adversarial nets. In Proceedings of the Advances in Neural Information Processing Systems, Montreal, QC, Canada, 8–13 Dcember 2014.
15. Pitropakisa, N.; Panaousisb, E.; Giannetsosc, T.; Anastasiadisd, E.; Loukase, G. A taxonomy and survey of attacks against machine learning. *Comput. Sci. Rev.* **2019**, *34*, 100199. [CrossRef]
16. Reddy, R.R.; Kavya, B.; Ramadevi, Y. A survey on svm classifiers for intrusion detection. *Int. J. Comput. Appl.* **2014**, *98*.
17. Khitan, S.J.; Hadi, A.; Atoum, J. Atoum, PDF Forensic Analysis system using YARA. *Int. J. Computer Sci. Netw. Secur.* **2017**, *17*, 77–85.
18. Bienz, T.; Cohn, R.; Adobe Systems (Mountain View, Calif.). *Portable Document Format Reference Manual*; Citeseer; Addison-Wesley: Boston, MA, USA, 1993.
19. Kittilsen, J. Detecting Malicious PDF Documents. Master's Thesis, Gjøvik University College, Gjøvik, Noway, 2011.
20. Blonce, A.; Filiol, E.; Frayssignes, L. Frayssignes. Portable document format (pdf) security analysis and malware threats. In Proceedings of the Presentations of Europe BlackHat 2008 Conference, Amsterdam, The Netherlands, 28 March 2008.
21. JavaScript Official Website. Available online: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Grammar_and_types (accessed on 28 October 2019).