# Malicious Document Detection and Adversarial Analysis based on Machine Learning

Yubin Yang1, Wei Jiang2, Fengjiao Wang3

1Bluedon, 510665 Guangzhou, China

2Bluedon, 510665 Guangzhou, China

3Bluedon, 510665, Guangzhou, China

[ronald\_yang@126.com](mailto:ronald_yang@126.com), [wei.jiang@nyu.edu](mailto:wei.jiang@nyu.edu), yonahwang@foxmail.com

**Abstract -** Nowadays, with the highly rapid development of information technology, it is becoming more and more important to perform detection on malicious documents (such as PDFs). But due to the diversity of the document structure, attackers can gradually have larger attack vector. This research project aims to construct a robust AI document classifier both for industry and academia. Around 200,000 samples have been collected and the AI model have been trained and optimized. The experimental results show that the Accuracy of the model is as high as 99.82% while the False Positive Rate is only as low as 0.01%. More, through the study of adversarial ML, the model has certain capability to resist attacks and enjoys good robustness. At last, we demonstrate our model can be widely deployed in typical scenarios such as security products or mail servers.

**Key Words：**AI Security; Machine Learning; Maldoc Detection; Adversarial ML

## Introduction

Cyber attackers are turning to document-based malware as users wise up to malicious email attachments and web links, suggested by many anti-virus (AV) vendors. Users are generally warned more on the danger of executable files by browsers, email agents, or AV products, but documents such as PDFs are treated with much less caution and scrutiny because of the impression that they are static files and can do little harm.

However, over time, PDF specifications have changed. The added scripting capability makes it possible for documents to work in almost the same way as executable files, including the ability of connecting to the Internet, running processes, and interacting with other files/programs. The growth of content complexity gives attackers more weapons to launch powerful attacks and more flexibility to hide malicious payload (e.g., encrypted, hidden as images, fonts or Flash contents) and evade detection.

A maldoc usually exploits one or more vulnerabilities in its interpreter/renderer to launch an attack. Unfortunately, given the increasing complexity of document readers and the wide library/system component dependencies, attackers are presented with a large attack surface. New vulnerabilities continue to be found, with 137 published CVEs in 2015 and 227 in 2016 for Adobe Acrobat Reader (AAR) alone. The popularity of Adobe Acrobat Reader and its large attack surface make it among the top targets for attackers. The collected malware samples show that many Adobe components have been exploited, including element parsers and decoders, font managers, and the JavaScript engine etc.

The continued exploitation of AAR along with the ubiquity of the PDF format makes maldoc detection a pressing problem, and many solutions have been proposed in recent years to detect documents bearing malicious payloads. These techniques can be classified into two broad categories: static and dynamic analysis.

Static analysis, or signature-based detection, parses the document and searches for indications of malicious content, such as shellcode or similarity with known malware samples. Dynamic analysis, or execution-based detection, runs partial or the whole document and traces malicious behaviors, such as vulnerable API calls or return-oriented programming (ROP).

In the first half of this paper, we utilize machine learning techniques on document-specific attributes to identify embedded malware. Our approach addresses some of the shortcomings of existing techniques through the use of a broadly applicable mechanism to classify and characterize documents.

As part of our analysis, we show that while the use of documents as an exploitation vector can be an enabling mechanism for the attacker, it also provides additional detection opportunities. All of the data closely associated with malicious activities can be used in aid of detection, regardless of whether the data utilized for detection is inherently malicious or not. The underlying premise and intuition of our study are that malicious documents do have similarities to other malicious documents; they also have dissimilarities to benign documents, regardless of the specific vulnerability exploited or the specific malware embedded in the document. We posit that features based on document structure and metadata are adequate for reliable document classification given appropriate statistical methods are applied to these features. This ensemble classifier is also able to classify previously unseen variants.

Clearly, deployment of learning methods in any security-critical context requires that they can withstand potential attacks. The security of machine learning methods has been previously discussed from conceptual, methodical and practical viewpoints. Typically, the security analysis of proposed learning-based techniques is carried out informally and is occasionally supported by experimental evaluation. From the practical perspective, the success of attacks against learning algorithms crucially depends on the amount of knowledge available to an attacker. Most of the previously reported successful attacks assume that the attacker has full knowledge of the learned model. It can, therefore, be argued that reducing the amount of knowledge leaked about the model, as well as a proactive response to the potential exploitation of such knowledge should provide adequate protection against adversarial data manipulation.

Still, it remains largely unclear what an attacker may learn about a learning-based method deployed “in the wild” and how this information can be exploited. To investigate this problem, we present the results of a case study we performed on a real learning-based model. For any submitted PDF file, the model provides a probabilistic estimate of its maliciousness. Our study addresses the case when an attacker attempts to evade detection by modifying the submitted PDF file so that its malicious functionality remains intact but the probabilistic score returned by the model is decreased.

To systematically explore the attacker’s options, we define an orthogonal set of evasion strategies reflecting various degrees of available knowledge. The general idea of our evasion technique is based on insertion of dummy content into PDF files which is ignored by PDF renderers but affects the computation of features. Once we can influence a subset of features, we develop algorithms for constructing attack instances. In the experiments, we evaluate the effectiveness of our strategies against our model.

In summary, this paper makes the following contributions:

* A new document dataset with 173036 malicious files and 28332 benign files
* Identification of 133 useful and comprehensive static features for detection
* A high accuracy rate of 99.82%, with a false positive rate of less than 0.01% for the learned model
* Prediction time for single file maintains at a millisecond level
* Develop an adversarial examples detection framework including adversarial example generation, robust model generation, and evasion detection
* Present a general model for practical assessment of the security of learning-based detection techniques. This model enables systematic exploration of various kinds of information leaks exploitable by an attacker and is applicable to systems that have a modifiable subset of features.
* We provide an open source software framework for all experiments carried out in our study for independent verification and extension of our results.

## Related work

Existing malicious documents detection methods can be classified broadly into two categories: dynamic and static analysis. Dynamic analysis, in which malicious documents are executed and examined in a specially created environment in order to capture the samples’ malicious behavior; while for the static analysis, the detection is carried out without code execution but with static scanning and examination for the header, binary level N-gram of files, etc. In general, the advantages of static analysis are: easy to deploy, good speed but relatively low accuracy. Compared with that, dynamic analysis, though suffering from low-speed and intense resources-consuming, enjoys the highest accuracy. Both techniques nowadays have already had a large number of successful stories. More advanced solutions in this line usually involve the hybrids of dynamic and static detection methods (Please see Maiorca et al. [9] for detail). A summary of existing methods is presented in Table1.

Table 1: A taxonomy of malicious PDF document techniques. This taxonomy is partially based on Platform Diversity [8] with the addition of works after 2016 as well as summaries parser, machine learning, and pattern dependencies

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Category | Focus | Detection | Work | Year | External Parser? | ML? | Discrepancy? |
| Static | JavaScript | Lexical Analysis [5] | PJScan | 2011 | Y | Y | Y |
| JavaScript | Token Clustering [12] | Vatamanu et al. | 2012 | Y | Y | Y |
| JavaScript | API Reference Classification [7] | Lux0r | 2014 | Y | Y | Y |
| JavaScript | Shellcode and opcode sig [13] | MPScan | 2013 | N | N | N |
| Metadata | Linearized object path [11] | PDF Malware Slayer | 2012 | Y | Y | Y |
| Metadata | Hierarchical Structure [1] | Srndic et al. | 2013 | Y | Y | Y |
| Metadata | Content Meta-features [24] | PDFrate | 2012 | Y | Y | Y |
| Both | Many Heuristics Combined [8] | Maiorca et al. | 2015 | Y | Y | Y |
| Both | Many Heuristics Combined [9] | Maiorca et al. | 2016 | Y | Y | Y |
| Dynamic | JavaScript | Shellcode and opcode sig [15] | MDScan | 2011 | Y | N | N |
| JavaScript | Known Attack Patterns [16] | PDF Scrutinizer | 2012 | Y | N | N |
| JavaScript | Memory Access Patterns [17] | ShellOS | 2011 | Y | N | Y |
| JavaScript | Common Maldoc Behaviors [18] | Liu et al. | 2014 | N | N | Y |
| JavaScript | Platform Independent Tap Point Identification [20] | tap point | 2016 | N | N | Y |
| Memory | Violation of Invariants [19] | CWXDetector | 2012 | N | N | N |
|  | OS | Platform Diversity [21] | PlatPal | 2017 | Y | N | Y |

From Table1, we can conclude that the main focus of static analysis is JavaScript or Metadata from files. Typical detection technique includes Shellcode and Opcode Sig based MPScan[13], Structure and Content-based classification[9]. On the other hand, dynamic analysis mainly focuses on extracting the JavaScript Snippet from the file and running them directly in order for malicious behavior detection. Typical work in this line includes behavior-based analysis[20] and platform diversity-based analysis[21].

We can also see from Table1 that all but three methods use either open-sourced or their home-grown parsers and assume their capability. However, Carmony et al.[20] shows that these parsers are typically incomplete and have oversimplified assumptions in regard to where JavaScript can be embedded. This leads to one of the most important research questions: Whether the external parser is robust? This is because the design and implementation of this kind of external parser are usually simple without being designed to be secured, in this case, the only little effect is needed for the successful evasion of malicious malwares. We call this kind of attack ‘Parser Confusion Attacks’ according to Carmony et. al.[20].

Also from Table1, an important conclusion can be drawn: Machine learning (ML), in general, is fitted for static analysis rather than dynamic analysis, for which we have not seen a dynamic paper with machine learning but ML has been the ‘default standard’ for nearly all static papers because of their ability in classification/clustering without prior knowledge of the pattern. Typical machine learning work here includes PDFrate[24] and PDF Malware Slayer[11] etc. Nearly all their work claim that their classifiers can attain high accuracy under resource intensive environment, but seldom mention the security of their deployed ML models, no need to say a comprehensive study on adversarial machine learning. This raises serious doubts about the effectiveness of classifiers based on superficial features in the presence of adversaries. This kind of attack has been mentioned in Xu et. al.[14], he is capable of automatically producing evasive maldoc variants. Here, for each iteration and for every sample, the operation such as addition, deletion and replace to the PDF structure tree is performed via genetic operation like programming. During the whole process, the malicious behavior of the sample should maintain exactly the same, but the ability to confuse and evade the classifier is stronger at each iteration. We call this kind of attack ‘Classifier Evasion Attack’ according to Xu et. al.[14].

An implicit assumption is that structural/behavioral discrepancies exist between benign and malicious documents and such discrepancies can be observed. Since the document must follow a public format specification, commonalities (structural or behavioral) are expected in benign documents. If a document deviates largely from the specification or the common patterns of benign samples, it is more likely to be a malicious document. In other words, a hyper-plane should always be found and posited in a high dimensional feature space to clearly separate the malicious and benign samples. But this assumption doesn’t hold if we can answer the following research questions:

* Can we evade the classifier by adding, deleting, or replacing content to the malicious PDF files while still keeping the malicious behaviors of files?
* Can we evade the classifier by gradually adding malicious stuff to the benign PDF files ~~and still fly under the radar without being detected~~?

The work by Srndic et. al.[4], starting with malicious files, has answered the 1st question which is called “Mimicry Attack”; While Maiorca et. al.[10], starting with benign files, has answered the 2nd question which is called “Reverse Mimicry Attacks”. Both works show that how a malicious document can systematically evade detection.

In summary, for attacking external parser, the current technique is called ‘Parser-Confusion Attacks’; For attacking ML models, the current technique is called ‘Automatic Classifier Evasion Attacks’; For the assumed detectable discrepancy, the existing attack is called ‘Mimicry and Reverse Mimicry’.

There is no doubt that attacks mentioned above have raised significant challenges on the security of the ML models, extending to the entire framework. Therefore, in this paper, we propose not only some approaches that can achieve high accuracy, but also a truly secured, robust and working model under our effective defense strategies.

## Design and Implementation of ML Model

In this session, we present a machine learning-based framework for detecting malicious documents. The dataset used in our experiment contains 200,000 samples, including all formats of PDF files. We mainly parse the content and structure of these files and select features with good classification performances, and then classify the extracted features by using machine learning method. The experiment results reveal that based on the extracted features and the classification method we used, the accuracy rate of our model is over 99%, with a false positive rate which is less than 0.01%.

### 3.1 Dataset

The PDF dataset, a total of 201368 samples, is divided into two categories: 28332 benign samples and 173036 malicious samples. Among these samples, 156035 are downloaded from VirusShare, with a size of 6.8G; 9000 benign samples of Contagio dataset, 2026 are obtained from Sogou and Baidu (two search engine in China).

We use VirusShare as the source to generate 7000 adversarial samples which are used in testing in the experiment.

Besides, we obtained the open source dataset from mimicus which is used for assessing PDFRATE[4]. This dataset contains 20,000 balanced samples, with 5,000 benign and 5,000 malicious subsamples of Contagio dataset, and 5,000 benign samples obtained from Google as well as 5,000 malicious samples downloaded from VirusTotal.

### 3.2 Feature Extraction

Effective methods for extracting features are based on structure, metadata, content, and Javascript. The experiment results reveal that structure-based features perform well in classification. We calculate the average value of each feature in the dataset and find that the average values of some features in benign and malicious samples are different.

Features such as count\_font,count\_box: There are several objects like font, box contained in the benign samples as PDF file mainly uses these objects for description. However, malicious files do not aim at describing information, instead they run the malicious code embedded in the file to launch the attack.

Features such as count\_page\_obj and count\_obj: Generally, obj in benign files are much more than those in malicious files. When calculating the number of obj in the same page, that in a malicious file is twice as many as that in a benign file. Thus, if the number of obj in the same page increases sharply, the file is likely to be malicious.

Features such as count\_endobj and count\_endstream: In benign files, the endobj refers to the end of an object. Yet a maldoc seldom contains endobj and endstream, for which it aims at confusing the parser to make it fail to obtain the whole object when parsing the malicious file, or fail to parse the malicious documents which can then evade detection successfully. That is the common evasion against the parser.

Features such as count\_js: The main tactic of the malicious document is to embed JS code in the file to execute malicious behaviors. In this way, JS codes contained in a maldoc are much more than those in a benign file.

Features such as count\_acroform\_obs: AcroForm is introduced in PDF Specification 1.2, which is to collect information from users via interaction. The form can display, capture and edit the data, etc. Moreover, it can conduct dynamic interaction from the interactive and editable forms which contain characteristics like dynamic calculation, verification and so on, to the forms generated by servers and filled in by machine. With those characteristics, the form is vulnerable to obscuration and encryption by the attacker. As a result, the value of AcroForm in a malicious sample usually doubles that of a benign sample.

Table 2: Average Value Comparison of Features in Benign and Malicious Files.

|  |  |  |
| --- | --- | --- |
| Feature | Benign File | Malware File |
| **count\_font** | **14.64** | **0.55** |
| **count\_acroform\_obj** | **700** | **1400** |
| **count\_box\_a4** | **12001** | **200** |
| **count\_box\_legal** | **395040** | **0** |
| count\_box\_letter | 7291529 | 866773 |
| count\_box\_other | 32.18 | 1.74 |
| count\_box\_overlap | 1000 | 0 |
| **count\_endobj** | **95.80** | **9.68** |
| **count\_endstream** | **30.43** | **3.78** |
| **count\_page\_obj** | **8001** | **16003** |
| count\_image\_large | 110711 | 400 |
| count\_image\_med | 465247 | 6401 |
| **count\_image\_small** | **915892** | **12002** |
| count\_image\_total | 36.56 | 0.30 |
| count\_image\_xlarge | 300 | 0 |
| count\_image\_xsmall | 21.64 | 0.11 |
| **count\_js** | **0.71** | **1.01** |
| **count\_obj** | **100.96** | **12.01** |
| count\_objstm | 1.57 | 0.15 |

### Classification algorithm

First, we classify the collected files by extracted features which are used as the training dataset. Random Forests Algorithm we used performs well during classification, with high effectiveness and low false positive rate. Moreover, it is easy to use and can classify data rapidly. The output of random forests classification method is a result ensemble of a multitude of decision trees which is constructed from a randomly selected subset of training data. That said, Random Forests is an ensemble classifier applying the technique of bagging training data. Each node in a decision tree is constructed based on a randomly selected subset of features, as well as the best split at each node, which is determined by training data for that node.. Finally, the classification result is determined by the votes of each tree.

One of the important components of AI engine is the algorithm. We select and compare several algorithms with good performance including KNN, NNET, Random forests, and SVM. After multiple times of training and classification experiments, we have found that random forests algorithm demonstrates a high detection accuracy, low false positive rate and latency, and good robustness and resolvability. Therefore, we choose Random Forests as our default algorithm. After more than one hundred experiments, the accuracy rate of Random Forests tends to be a stable value even if the features change.

Table 3 Comparison of Algorithm Accuracy Rate

|  |  |  |  |
| --- | --- | --- | --- |
| SVM | NNET | KNN | RF |
| 75.23% | 82.41% | 97.12% | 99.64% |

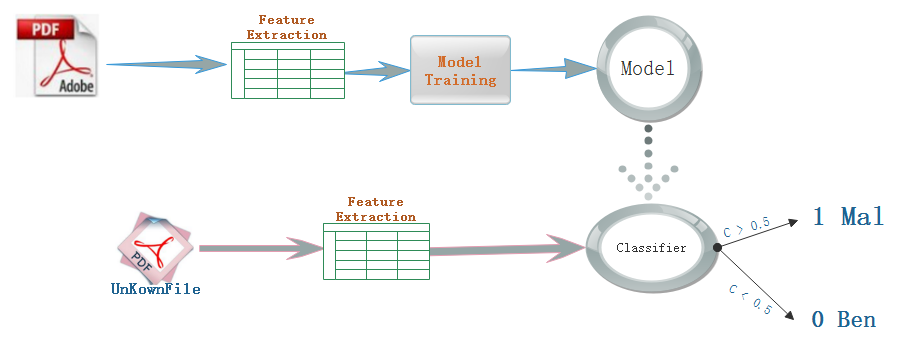
### 3.4 Model Construction

The proposed machine learning-based method of detecting a malicious document includes the following steps, as shown in Figure 1:

1. Feature Extraction. This is a basic preprocessing step, which is to parse the structure, content and metadata in PDF files and then conduct vector computation on these objects, extract them as a two-dimension feature set finally. These features can be trained and classified in the model based on machine learning.

2. Learning and Classification. We select 80% of the data randomly for training and then save the trained model. Later we use 20% of the files for prediction and classification, in order to calculate the information such as accuracy rate, false positive rate, etc.

Figure 1 Basic Framework of Machine Learning



During the experiment, we have updated the model for four times. The initial one (Model 1) uses peepdf as the parser to extract features. After computation and quantization, these features can be used for machine learning-based training and prediction. We have extracted 133 features which contain static attributes of structure (count\_font、size、count\_startxref), content (title\_oth、subject\_lc) and metadata(producer\_oth、producer\_len). We classify these extracted features via Random Forests as it brings higher accuracy. But problems also exist: only half of the files can be parsed if we use peepdf. Therefore, we reselect another parser mimicus[2] which can help us tackle the failure of parsing caused by structure defect or obscuration. By using mimicus, all of the data can be parsed and features can be extracted.

For the training of Model 2, we initially use balanced dataset for training and prediction. This balanced dataset includes 20,000 malicious and 20,000 benign samples which are selected randomly from the whole dataset. Besides, we use mimicus to extract features from the beginning of training of Model 2 and finally we extract 135 features in total. The main algorithm we used remains to be Random Forest. After parameter tuning, the detection rate of Model 2 in multiple times of testing can increase and maintain at 99.99%, with a decreased FP rate of 0.012%.

We use a new sample dataset at a ten-thousand level to retrain the model, with a training time being only 56 seconds by utilizing a quad-core with 4G RAM. Then we test the Model 2 by using 20,000 samples, the accuracy rate still maintains at 99.81% with a false positive rate of 0.086%. From the experiment, we can learn that when the dataset increase to a ten-thousand level, the accuracy drops by 0.18%, because the ways of file classification and code embedding rise along with the increase of dataset. As a result, the robustness and security of the model should be considered and we will discuss this issue in the next session.

## Adversarial Analysis

Machine-learning classifiers are increasingly used in detecting malicious data. However, if the models are deployed online, attackers may evade them by using data manipulation. Such kind of attacks has been studied in the previous works, assuming attackers have full knowledge about the deployed classifier. In practice, this assumption is rarely made, especially for the online-deployed system. The knowledge about the deployed classifier can be obtained from all kinds of sources. In this session, we use a real, deployed Model 2 as a testing case, to verify the effectiveness of classifier evasion.

We have built a system for practical evasion strategies, and adapt several evasion algorithms for practical application scenarios. Our experiment results reveal that the detection accuracy of Model 2 declines sharply even if it is exposed to simple attacks. In addition, we have studied some potential prevention strategies against classifier evasion. The experiment results show that two techniques can improve the robustness of model when facing such attacks. The two techniques include: (1) increasing the amount of dataset used for model training and (2) applying different feature sets to retrain model. In the discussion, we analyze some potential techniques in order to strengthen the robustness of learning-based systems against adversarial data manipulation.

### 4.1 Sample Evasion

In this session, we are going to discuss adversarial learning in particular scenarios. To be specific, it is supposed that an attacker has obtained some information of a known model, such as extracted features, the algorithm applied by the model, etc. If the attacker knows more about the model, the evasion sample designed by him can evade model more easily. In this session, we mainly refer to the methods proposed by Nedim Smdic [4], to conduct evasion attack against the learning-based model. The four scenarios are shown as below:

* F（feature）：In this scenario, only feature set can be used by the adversary；
* FT（feature and training）：In this scenario, the adversary can utilize the knowledge of target classifier training dataset, except for the known features.
* FC（feature and classifier）: The adversary knows feature sets and some details about classifier such as its type, parameters or the specific implementation.
* FTC（all above）: The adversary can evade the target classifier if he knows all information about the classifier components. In that case, the adversary can fully reproduce an online classifier, submitting the attack results only when a sufficiently good evading sample has been found.

We use 2000 malicious samples, which are highly scored by the classifier, as the original samples to generate variants which can evade PDFRATE and still remain malicious. Then we use these variants to attack Model 2. As shown in Table 4, this attack causes a great effect to Model 2; in the scenario of FC, the detection accuracy rate of Model 2 is only 2.92%, which means that more than 90% of malicious documents can evade classifier successfully after variation.

Based on the above scenarios, we construct a Model 3 by modifying features and sample sets, with an increased training data of 200,000 samples. This updated training set includes some new variants, such as some variants generated by Mimicry Attack and Reverse Mimicry Attack methodology. As shown in Table 4, when Model 3 is attacked by the tactics used in the above four scenarios, its detection accuracy rate is higher than that of Model 2. As shown in Table 4:

Table 4: Different attack scenarios and accuracy

|  |  |  |  |
| --- | --- | --- | --- |
| scenario | evasive variants | Model2 accuracy | Model3 accuracy |
| F | 2157 | 71.18% | 96.71% |
| FC | 240 | 2.92% | 12.50% |
| FT | 4196 | 84.25% | 96.76% |
| FTC | 600 | 15.83% | 18.71% |

### 4.2 A case study

For the variation, we select some typical samples to conduct a case study. For example, we select a file with CVE ID (CVE-2013-0641), which can execute any code remotely by exploiting the vulnerability. We apply the methodologies in the above four scenarios to vary selected samples and then check the VT reports of these samples and their variants. The VT reports show that original samples can be detected by 61 detection engines, within which 33 engines can identify the malicious information. Yet after variation, only 60 engines can detect these variants and 22 engines are able to identify their malicious information.

Table 5 VT detection result of generate new malware

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| File\_HASH | Source | F | FC | FT | FTC |
| 00ba5c43b1cec186c634c24ac21982d3 cve-2013-0641 | 33/61 | 22/60 | 23/60 | 22/60 | 22/60 |

As most PDF detection engines are based on structure and content, once we modify the structure and content, such as add some objects of benign samples or modify the file size, the malicious file can evade classifier. We compare the files before and after variation. As shown in Table 6, variation includes modifying the size and content of metadata, adding the number of Count\_javascript and some content of Keywords which are objects in a benign sample, upgrading the file version from 4 to 7. After variation, the sample remains malicious while 10 classifiers have already failed to detect the malicious code of this file.

Table 6 Comparison of generate new malware feature

|  |  |  |
| --- | --- | --- |
| Feature | Source | evasive |
| author\_lc | 0 | 6 |
| author\_len | 0 | 14 |
| author\_uc | 0 | 6 |
| count\_javascript | 1 | 6 |
| createdate\_ts | -1 | 650616173 |
| createdate\_tz | -1 | 10020 |
| moddate\_ts | -1 | 482083775 |
| keywords\_lc | 0 | 4 |
| keywords\_len | 0 | 7 |
| producer\_lc | 0 | 8 |
| producer\_len | 0 | 19 |
| version | 4 | 7 |

### 4.3 Model Update

For the evasion samples, we use two kinds of methods to update the above model. The first method is to increase the number of training samples in order to avoid data overfitting and achieve the local optimum. The second one is to adjust the feature set to improve the detection rate.

If the feature sets are exploited by attackers, we can modify the datasets such as modify the weight of features or delete important features in order to retrain this model. As shown in Figure 2, the top 30 features are sorted by the importance of features after Model 3 is trained. From Figure 2 we can find that features including count\_font, count\_javascript, size，count\_obj，count\_endobj account for more weights in the classification. They are really easy to be exploited by attackers to evade parser and classifier. Therefore, we delete these features and then retrain the Model 3, outputting the prediction result shown in Figure 7.

Figure 2 The Top 30 features Distribution



Table 7 shows the detection accuracy of Model 3 after deleting the above 5 features. As shown in this table, when the classifier is trained by all features, the accuracy rate of model is up to 99.82%. If we delete the first feature even the top five features, the rate is almost stable. That said, our model can confront the attacks based on these features. Despite attackers have the knowledge of features for training model, the accuracy rate can maintain at the level of 99%.

Figure 7 Accuracy Rate of Model after deleting the top 5 features

|  |  |
| --- | --- |
| Feature delete train | 准确率 |
| None | 99.82% |
| count\_font | 99.52% |
| count\_javascript | 99.52% |
| Size | 99.64% |
| count\_obj | 99.64% |
| count\_endobj | 99.64% |

Besides, we assess the model robustness by the effectiveness of classification of features. First, we sequence the features based on their importance, and then delete the most important features one by one to create new feature sets which are used for model retraining. As shown in Figure 3, the curve represents the accuracy rate of the model when features are deleted one by one. When the features are decreased to 100, the accuracy rate of the retrained model still maintains at 90%. This demonstrates:

Despite the high weight of an individual feature, if it is deleted, the accuracy rate of model declines moderately.

The interaction and superposition of “Medium Weight” can make the model robust, and reduce the effect caused by the deletion of individual important features.

“Medium Weight” can help to effectively prevent classification evasion caused by modifying the value of features.

Figure 3 Detection Accuracy Rate when features are deleted



### 4.4 Performance Assessment

In order to assess the prediction performance of the model, we divide dataset randomly into two categories including training (90%) and testing (10%) samples, and apply Cross Validation method to assess model. As shown in Figure 4, the area below ROC curve is about 1, which represents the good prediction performance of Model 3.

图4 ROC Curve



The feature attraction is time-consuming because it needs to load all files from the disk and then parses them one by one. Therefore, we choose stepped processing: parsing files firstly, saving the feature sets extracted from the files, and then using these feature sets for model training. In this way, CPU occupation can be reduced while training time of model can be much shorter. For example, it takes only about 22 minutes to parse the training samples at a ten-thousand level. As shown in Figure 8, we compare training and prediction time as well as accuracy rate of Model 3 by applying different algorithms. The result shows that applying Random Forest Algorithm can achieve higher accuracy rate and short prediction time being 1 second.

表8 训练时间与预测时间

|  |  |  |  |
| --- | --- | --- | --- |
|  | 训练时间 | 预测时间 | 准确率 |
| Random Forest | 56s | 1s | 99% |
| Decision Tree | 4s | 1s | 97% |
| SVM | 58m 18s | 12s | 75% |

## Application：Bluedon AI Firewall

According to the design principle of modulation, we regard the AI-based maldoc detector as one independent detection module that can be easily integrated in our security products, such as the next-generation firewall. An interesting question here is: How Bluedon manages to apply AI technology seamlessly to a 30-years-old security product?

In the current development of Network and Gateway Security products, the capability of performing malicious file scanning effectively and efficiently at layer7 (the network application layer) is the international standard. The industry has strict demand on this product feature. A good detection module should include (1) millisecond latency for single file detection; (2) 99% accuracy while maintaining FP rate which is less than 0.01%.

The reason behind the demand for low-latency is obvious: The module has sequentially been placed into the working pipeline, high-latency will lead to the increase of Packet Drop Rate(PDR) and occasionally data loss. This is strictly forbidden for security devices. In the past few years, as the rapid development of malwares, the formal industry best practice – the pattern matching engine has gradually fade away from the mainstream recently. Two main reasons can be roughly seen: (1) In order to meet the requirement of having high detection accuracy, a large number of security analysts are needed for pattern writing but this manual process is not scaled at all; (2) As the fast-growing size of the core database, the time for core operation – Pattern Matching (PM) grows exponentially. The above two strong pieces of evidence inspire us to discover better engine rooted from AI.

By the year 2018, we have managed to integrate our AI maldoc detector into firewall, in the hope of replacing the old engine. Although both of the engines belong to the category of static analysis-based engine, the improvement of shifting from the old to the new AI engine is tremendous. On one side, the AI engine does not need to update frequently, because it can detect unseen malwares effectively for years. According to our experiment results, the average updating frequency for our AI engine is half a year, which is longer if compared to the 2-week period of the old pattern matching engine; On the other side, AI engine enjoys low resource consumption when executing. According to our study, during the phase of model prediction, AI engine can only take up as much as 1/3 of CPU and 50% of memory consumption. The portion from CPU is mainly caused by the computations such as feature extraction and confidence score computation. The portion from memory is mostly because of the fact that AI model is needed to be sited entirely in main memory when for prediction.

In the context of the firewall, different actions are triggered based on the probability and reasons output from the AI maldoc detection module. For instance, if the output probability is greater than a certain threshold of 0.9, this indicates the AI module has high confidence that this document is malicious; then a blocking operation is triggered, connection is dropped and an alert is raised for further investigation. If the output probability is less than a certain threshold of 0.1, this indicates that the AI module has high confidence that this document is benign, and then we will allow and monitor this connection as normal by default.

The truly interesting part lies when the output probability is in the range of 0.1 to 0.9. If this happens, we will by default upload the samples to our Threat Intelligence Cloud where multiple dynamic analysis will be performed with Sandbox, Threat Intelligence and Security Team. According to our heuristics in a typical usage scenario, 95% of the files are being processed inline while about 5% of the files are uploaded. Dynamic analysis from our TI cloud plays a great complementary role in the static analysis method inline. By means of combination of the two engines, we can now completely provide the end users with the more advanced AI-enabled security solution. We make our cloud service a subscription service and freely open the research community.

## Conclusion

In this paper, we introduce the design and implementation of PDF file classifier based on machine learning. Our experiment results reveal that with the dataset at a hundred level, we can achieve a detection accuracy which is larger than 99% and a false positive rate that is less than 0.01%. Besides, compared with the rule-based model, the Time and Space Performance of CPU and memory are improved significantly when the classifier is running in real time.

We use a great amount of data to study the application security based on artificial intelligence and at the same time give equal weight to the security of artificial intelligence itself. Through a lot of experiments, we have done the following works: 1) simulation of scenario in which attackers change the malicious samples (such as modify the value of features) so as to confuse classifier and then evade it; 2) our classifier conduct self-revision by retraining the model and eliminating the features that have been exploited by attackers in order to maintain the model robustness.

The learning-based classifier is an important research topic in the field of social engineering, malware analysis, etc. In the future, we will move forward and try to focus on the following topics:

* Malicious PDF file detection based on deep learning
* Optimization of the static and dynamic analysis engine
* Analysis of more file formats such as Microsoft Office including docx, pptx, etc.

**References**

1. Nedim ˇ Srndic and Pavel Laskov. Detection of Malicious Pdf Files Based on Hierarchical Document Structure. In 20th Network and Distributed System Security Symposium (NDSS), 2013
2. Nedim ˇ Srndic and Pavel Laskov. Mimicus: A Library for Adversarial Classifier Evasion. [https://github.com/srndic/ mimicus](https://github.com/srndic/mimicus).
3. Nedim Šrndic and Pavel Laskov . Hidost: a static machine-learning-based detector of malicious files, Šrndi′c and Laskov EURASIP Journal on Information Security (2016) 2016
4. Nedim Srndic and Pavel Laskov. Practical Evasion of a Learning- Based Classifier: A Case Study. In Proceedings of the 35th IEEE Symposium on Security and Privacy (Oakland), San Jose, CA, May 2014
5. Pavel Laskov and Nedim Srndic. Static Detection of Malicious JavaScript-Bearing PDF Documents. In Proceedings of the Annual Computer Security Applications Conference (ACSAC), 2011
6. Davide Balzarotti, Marco Cova, Christoph Karlberger, Christopher Kruegel, Engin Kirda, and Giovann Vigna. Efficient Detection of Split Personalities in Malware. In Proceedings of the 17th Annual Network and Distributed System Security Symposium (NDSS), San Diego, CA, February–March 2010
7. Igino Corona, Davide Maiorca, Davide Ariu, and Giorgio Giacinto. Lux0R: Detection of Malicious PDF-embedded JavaScript Code through Discriminant Analysis of API References. In Proceedings of the Artificial Intelligent and Security Workshop (AISec), 2014.PDFrate
8. Davide Maiorca, Davide Ariu, Igino Corona, and Giorgio Giac- into. A Structural and Content-based Approach for a Precise and Robust Detection of Malicious PDF Files. In *Proceedings of the International Conference on Information Systems Security and Privacy (ICISSP)*, 2015.
9. DavideMaiorca, DavideAriu, IginoCorona, andGiorgioGiacinto. An Evasion Resilient Approach to the Detection of Malicious PDF Files. In Proceedings of the International Conference on Information Systems Security and Privacy (ICISSP), 2016.
10. Davide Maiorca,IginoCorona,andGiorgioGiacinto.Lookingat 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 Symposium on Information, Computer and Commu- nications Security (ASIACCS), Hangzhou, China, March 2013.
11. Davide Maiorca,Giorgio Giacinto,and Igino Corona. APattern Recognition System for Malicious PDF Files Detection. In *Pro- ceedings of the 8th International Conference on Machine Learning and Data Mining in Pattern Recognition (MLDM)*, 2012.
12. Cristina Vatamanu, Drago¸s Gavrilu ¸ T, and R˘azvan Benchea. A Practical Approach on Clustering Malicious PDF Documents. Journal in Computer Virology, June 2012.
13. Xun Lu, Jianwei Zhuge, Ruoyu Wang, Yinzhi Cao, and Yan Chen. De-obfuscation and Detection of Malicious PDF Files with High Accuracy. In *Proceedings of the 46th Hawaii International Con- ference on System Sciences (HICSS)*, 2013.
14. Weilin Xu, Yanjun Qi, and David Evans. Automatically Evading Classifiers: A Case Study on PDF Malware Classifiers. In Proceedings of the 2016 Annual Network and Distributed System Security Symposium (NDSS), San Diego, CA, February 2016. http://evademl.org/
15. Zacharias Tzermias, Giorgos Sykiotakis, Michalis Polychronakis, and Evangelos P. Markatos. Combining Static and Dynamic Analysis for the Detection of Malicious Documents. In Proceedings of the 4th European Workshop on System Security (EUROSEC), 2011.
16. Florian Schmitt, Jan Gassen, and Elmar Gerhards-Padilla. PDF Scrutinizer: Detecting JavaScript-based Attacks in PDF Docu- ments. In *Proceedings of the 10th Annual International Confer- ence on Privacy, Security and Trust (PST)*, 2012.
17. Kevin Z. Snow, Srinivas Krishnan, Fabian Monrose, and Niels Provos. ShellOS: Enabling Fast Detection and Forensic Analysis of Code Injection Attacks. In *Proceedings of the 20th USENIX Security Symposium (Security)*, San Francisco, CA, August 2011.
18. DaipingLiu,HainingWang,andAngelosStavrou.DetectingMa- licious Javascript in PDF through Document Instrumentation. In *Proceedings of the 44th International Conference on Dependable Systems and Networks (DSN)*, Atlanta, GA, 2014.
19. Carsten Willems, Felix C. Freiling, and Thorsten Holz. Using Memory Management to Detect and Extract Illegitimate Code for Malware Analysis. In Proceedings of the Annual Computer Security Applications Conference (ACSAC), 2012.
20. Curtis Carmony, Mu Zhang, Xunchao Hu, Abhishek Vasisht Bhaskar, and Heng Yin. Extract Me If You Can: Abusing PDF Parsers in Malware Detectors. In Proceedings of the 2016 Annual Network and Distributed System Security Symposium (NDSS), San Diego, CA, February 2016
21. Meng Xu and Taesoo Kim, Georgia Institute of Technology:PlatPal: Detecting Malicious Documents with Platform Diversity . 26th USENIX Security Symposium 2017
22. VirusTotal. Free Online Virus, Malware and URL Scanner.https://www.virustotal.com/.
23. Stephan Chenette. Malicious Documents Archive for Signature Testing and Research - Contagio Malware Dump. http://contagiodump.blogspot.de/2010/08/ malicious-documents-archive-for.html.
24. Charles Smutz and Angelos Stavrou. Malicious PDF Detection using Metadata and Structural Features. In Proceedings of the Annual Computer Security Applications Conference (ACSAC), 2012.
25. Symantec 2017年安全威胁报告<https://www.symantec.com/content/dam/symantec/docs/reports/istr-22-2017-en.pdf>
26. M.Polychronakis,K.Anagnostakis,andE.Markatos.Com- prehensive shellcode detection using runtime heuristics. In Annual Computer Security Applications Conference (AC- SAC), pages 287–296, 2010.
27. Charles Smutz, Angelos Stavrou . When a Tree Falls: Using Diversity in Ensemble Classifiers to Identify Evasion in Malware Detectors. C Smutz, A Stavrou - NDSS, 2016

Appendix:

|  |  |
| --- | --- |
| feat | imp |
| count\_font | 0.108772 |
| count\_javascript | 0.087662 |
| size | 0.079581 |
| count\_obj | 0.069801 |
| count\_endobj | 0.059954 |
| producer\_oth | 0.05054 |
| producer\_len | 0.04837 |
| pdfid1\_num | 0.039756 |
| producer\_dot | 0.037493 |
| count\_box\_other | 0.03686 |
| count\_stream | 0.036113 |
| count\_endstream | 0.0302 |
| count\_js | 0.020743 |
| pdfid0\_len | 0.019087 |
| producer\_lc | 0.01906 |
| producer\_mismatch | 0.018668 |
| len\_obj\_max | 0.018317 |
| len\_stream\_avg | 0.017994 |
| len\_obj\_avg | 0.016931 |
| pdfid0\_num | 0.016363 |
| producer\_uc | 0.016284 |
| pdfid1\_len | 0.015949 |
| len\_stream\_max | 0.013086 |
| producer\_num | 0.011488 |
| count\_startxref | 0.009243 |
| len\_obj\_min | 0.008178 |
| count\_page | 0.008106 |
| pdfid1\_mismatch | 0.007617 |
| pdfid0\_mismatch | 0.006044 |
| createdate\_version\_ratio | 0.005681 |
| creator\_len | 0.005603 |
| moddate\_version\_ratio | 0.004428 |
| title\_oth | 0.004373 |
| creator\_lc | 0.003601 |
| box\_other\_only | 0.003494 |
| creator\_uc | 0.003382 |
| pdfid\_mismatch | 0.00239 |
| moddate\_mismatch | 0.002291 |
| count\_box\_letter | 0.002237 |
| createdate\_mismatch | 0.001802 |
| moddate\_tz | 0.0017 |
| count\_eof | 0.001628 |
| subject\_lc | 0.001521 |
| title\_num | 0.001473 |
| len\_stream\_min | 0.001459 |
| title\_len | 0.001409 |
| count\_trailer | 0.001283 |
| pdfid1\_uc | 0.001269 |
| createdate\_tz | 0.001172 |
| title\_dot | 0.00116 |
| subject\_len | 0.001011 |
| title\_uc | 0.000923 |
| version | 0.000897 |
| title\_lc | 0.000852 |
| moddate\_ts | 0.000791 |
| box\_nonother\_types | 0.000789 |
| creator\_oth | 0.000768 |
| author\_len | 0.000736 |
| count\_xref | 0.000735 |
| subject\_oth | 0.000664 |
| count\_action | 0.000659 |
| createdate\_ts | 0.00063 |
| pdfid0\_uc | 0.000599 |
| delta\_ts | 0.000594 |
| count\_acroform | 0.000591 |
| author\_uc | 0.000591 |
| image\_totalpx | 0.000526 |
| creator\_dot | 0.000494 |
| delta\_tz | 0.000457 |
| count\_objstm | 0.000434 |
| creator\_num | 0.000433 |
| author\_lc | 0.000378 |
| author\_oth | 0.000319 |
| pdfid0\_oth | 0.000309 |
| count\_stream\_diff | 0.000293 |
| count\_image\_total | 0.000258 |
| creator\_mismatch | 0.000248 |
| author\_mismatch | 0.00024 |
| title\_mismatch | 0.00023 |
| keywords\_len | 0.000219 |
| pdfid1\_oth | 0.000209 |
| author\_num | 0.000203 |
| count\_image\_large | 0.000187 |
| keywords\_lc | 0.000175 |
| pdfid0\_lc | 0.000138 |
| count\_image\_small | 0.000114 |
| keywords\_oth | 0.000109 |
| author\_dot | 8.83E-05 |
| ratio\_imagepx\_size | 8.65E-05 |
| pdfid1\_lc | 6.91E-05 |
| subject\_mismatch | 6.74E-05 |
| subject\_uc | 5.21E-05 |
| keywords\_uc | 5.03E-05 |
| keywords\_mismatch | 4.02E-05 |
| count\_javascript\_obs | 2.06E-05 |
| count\_page\_obs | 1.51E-05 |
| count\_box\_a4 | 1.24E-05 |
| image\_mismatch | 1.02E-05 |
| subject\_dot | 9.83E-06 |
| count\_image\_med | 9.75E-06 |
| count\_font\_obs | 9.20E-06 |
| count\_action\_obs | 8.90E-06 |
| company\_mismatch | 7.60E-06 |
| keywords\_num | 7.54E-06 |
| count\_js\_obs | 6.23E-06 |
| subject\_num | 5.21E-06 |
| count\_image\_xsmall | 3.92E-06 |
| count\_acroform\_obs | 0 |
| count\_box\_legal | 0 |
| count\_box\_overlap | 0 |
| count\_image\_xlarge | 0 |
| count\_objstm\_obs | 0 |
| createdate\_dot | 0 |
| keywords\_dot | 0 |
| moddate\_dot | 0 |
| pdfid0\_dot | 0 |
| pdfid1\_dot | 0 |
| pos\_acroform\_avg | 0 |
| pos\_acroform\_max | 0 |
| pos\_acroform\_min | 0 |
| pos\_box\_avg | 0 |
| pos\_box\_max | 0 |
| pos\_box\_min | 0 |
| pos\_eof\_avg | 0 |
| pos\_eof\_max | 0 |
| pos\_eof\_min | 0 |
| pos\_image\_avg | 0 |
| pos\_image\_max | 0 |
| pos\_image\_min | 0 |
| pos\_page\_avg | 0 |
| pos\_page\_max | 0 |
| pos\_page\_min | 0 |
| ratio\_size\_obj | 0 |
| ratio\_size\_page | 0 |
| ratio\_size\_stream | 0 |