Annals of Telecommunications

Malicious Document Detection and Robust ML Model Construction

--Manuscript Draft--

Manuscript Number:	
Full Title:	Malicious Document Detection and Robust ML Model Construction
Article Type:	"CfP: Security and Privacy in IoT Communication"
Corresponding Author:	Yonah Wang Bluedon Information Security Technologies Co. CHINA
Corresponding Author Secondary Information:	
Corresponding Author's Institution:	Bluedon Information Security Technologies Co.
Corresponding Author's Secondary Institution:	
First Author:	Yubin Yang, PH.D.
First Author Secondary Information:	
Order of Authors:	Yubin Yang, PH.D.
	Fengjiao Wang
	Wei Jiang, PH.D.
	Shumin Wei, M.A.
Order of Authors Secondary Information:	
Funding Information:	
Abstract:	With the rapid development of information technology, it has become increasingly more important to perform detection on malicious documents. However, due to the diversity of document structures, attackers have gradually acquired a large attack vector. In this paper, we aim to construct a robust artificial intelligence (AI) document classifier both for industry and academia. Approximately 200,000 samples have been collected, and the AI model has 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 as low as only 0.01%. Moreover, through the study of adversarial machine learning, the model is capable to resist attacks and enjoys good robustness. Finally, we demonstrate that our model can be widely deployed in typical usage scenarios, such as security products or mail servers.

Malicious Document Detection and Robust ML Model Construction

Yubin Yang[1], Wei Jiang[2], Fengjiao Wang[3] ,Shumin Wei [4]

[1]School of Computer Science & Engineering, South China University of Technology, 510641 Guangzhou, China
[2]Bluedon AI Lab, 510641 Guangzhou, China
[3]Bluedon AI Lab, 510641 Guangzhou, China
[4]Bluedon AI Lab, 510641 Guangzhou, China

ronald_yang@126.com, weijiang2009@gmail.com, yonahwang@foxmail.com, weism@chinabluedon.cn

Abstract - With the rapid development of information technology, it has become increasingly more important to perform detection on malicious documents. However, due to the diversity of document structures, attackers have gradually acquired a large attack vector. In this paper, we aim to construct a robust artificial intelligence (AI) document classifier both for industry and academia. Approximately 200,000 samples have been collected, and the AI model has 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 as low as only 0.01%. Moreover, through the study of adversarial machine learning, the model is capable to resist attacks and enjoys good robustness. Finally, we demonstrate that our model can be widely deployed in typical usage scenarios, such as security products or mail servers.

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

1. Introduction

Cyber attackers are turning to document-based malware as suggested by many anti-virus (AV) vendors. Users are increasingly being warned more generally of 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 their ability to connect to the Internet, run processes, and interact with other programs. The growth of content complexity gives attackers more weapons with which to launch powerful attacks and more flexibility to hide malicious and evade detection.

A maldoc usually exploits one or more vulnerabilities in its interpreter to launch an attack. Unfortunately, given the increasing complexity of document readers and the wide library of 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 AAR 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.

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 to known malware samples. Dynamic analysis, or execution-based detection, runs the partial or entire document and traces malicious behaviors, such as vulnerable application programming interface (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 by 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 to aid detection, regardless of whether the data utilized for detection are 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 that 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 ML 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.

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 that 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 173,036 malicious files and 28,332 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.
- Development of an adversarial examples detection framework including adversarial example generation, model hardening, evasion detection and 5 effective defense techniques.

2. Related Work

Existing malicious document detection methods can be classified broadly into two categories: dynamic and static analysis. In dynamic analysis, malicious documents are executed and examined in a specially created environment in order to capture the samples' malicious behavior; in static analysis, the detection is carried out without code execution but with static scanning and examination for the header, N-gram of files, and others. In general, the advantages of static analysis are ease of deployment and good speed (but relatively low accuracy). Compared to static analysis, dynamic analysis, although suffering from low speed and intense resource consumption, exhibits the highest accuracy. Nowadays, both techniques have already had a large number of successful stories. More advanced solutions along this line usually involve the hybrids of dynamic and static detection methods (see Maiorca *et al*. [9] for details). A summary of existing methods is presented in Table1.

Table 1. Taxonomy of malicious PDF document techniques partially based on platform diversity [8] with the addition of works after 2016 as well as summaries parsers, ML, 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 Table 1, we can conclude that the current focus of static analysis is JavaScript or file metadata. Typical detection techniques include Shellcode and OpCode Signature based MPScan [13], and structure and content based classification [9]. However, dynamic analysis mainly focuses on extracting the JavaScript Snippet from the file and running them directly to enable malicious behavior detection. Typical work in this line includes behavior-based analysis [20] and platform diversity-based analysis [21].

We can also see from Table 1 that all but three methods use either open-source or home-grown parsers and assume their capability. However, Carmony *et al*. [20] shows that these parsers are typically incomplete and make oversimplified assumptions about where JavaScript can be embedded. This leads to one of the most important questions in this research: Is the external parser robust enough? This is because the design and implementation of these kinds of external parsers are usually simple without being designed to be secured; In this case, only little effort is needed for the successful evasion of malicious malware. This kind of attack is called 'parser-confusion attacks' by Carmony *et al*. [20].

An important conclusion can be drawn from Table 1: Machine learning (ML), in general, is fit for static rather than dynamic analysis. We are not aware of a paper on dynamic analysis that considers ML, while ML has been the 'default standard' in nearly all static analysis papers. Typical ML work here includes PDFRATE [24] and PDF Malware Slayer [11]. Nearly all of the aforementioned works claim that their classifiers can attain high accuracy in resource-intensive environments, but seldom mention the security of their deployed ML models, much less the need for a comprehensive study of adversarial ML. This raises serious doubts about the effectiveness of classifiers based on superficial features in the presence of adversaries. An attack exploiting this vulnerability is mentioned in Xu *et al*. [14], one that can automatically produce evasive maldoc variants. Here, for each iteration and for every sample, operations like addition, deletion, and replace with respect to the PDF structure tree is performed via genetic programming like operations. During the entire process, the malicious behavior of the sample should remain the same, but the ability to confuse and evade the classifier is stronger at each iteration. This kind of attack is called a 'classifier evasion attack' by Xu *et al*. [14].

An implicit assumption is that behavioral discrepancies exist between benign and malicious documents and such discrepancies can be clearly observed and captured. 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. However, this assumption does not hold if one can answer the following research questions:

- Can we evade the classifier by adding, deleting, or replacing content of the malicious PDF files while keeping the malicious behaviors?
- Can we evade the classifier by gradually adding malicious content to the benign PDF?

The work by Srndic *et al*. [4], started with malicious files, answered the first question, which refers to as 'mimicry attack'. Maiorca *et al*. [10], started with benign files, answered the second question, which refers to as 'reverse mimicry attack'. Both works show how a malicious document can systematically evade the classifier.

In summary, for attacking an external parser, the current technique is called ‘parser-confusion attack’. For attacking ML models, the current technique is called ‘automatic classifier evasion attack’. For the assumed detectable discrepancy, the existing attack is called ‘Mimicry and Reverse Mimicry’.

There is no doubt that the attacks mentioned above attacks have raised significant challenges to the security of ML models, and potentially extends to the entire detection framework. Thus, in this paper, we propose not only some ML techniques that can achieve high accuracy, but also a truly secure, robust model under our effective defense strategies.

3. Design of ML Maldoc Classifier

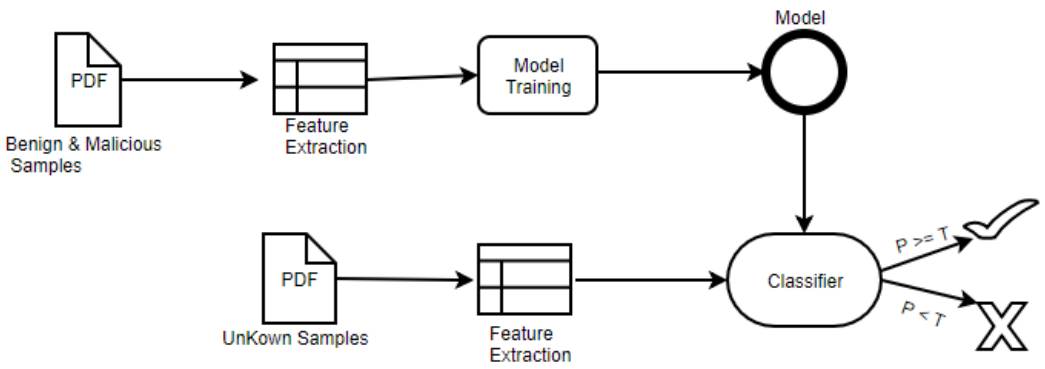
In this session, we focus on designing binary classifiers, a particular kind of learning systems, which classify new data into two predefined categories. Classifiers usually make predictions by computing some numeric or probabilistic score and comparing it with a fixed threshold. We focus on the following aspects of classifier design:

A. General ML Classification Framework
B. Dataset
C. Feature Engineering
D. Classification Alg. Selection
E. Model Evolution

A. General ML Classification Framework

The proposed ML Classification framework are depicted in Figure 1. Our goal is to train a robust model for maldoc detection. Firstly, we need to collect a significant amount of malicious and benign documents during the data collection phase. Secondly, we manually design and extract hundreds of representative features from each document during our feature engineering phase, a process from transforming the documents from raw to feature vectors. Finally, we have trained the ML model so that the model can fit the underlying training data distribution well. The training phase usually performs offline due to heavy time and space footprint while the prediction phase is wrapped as online service. At this point, our model is ready for serving. When a new sample is presented to the model, it can return a confidence score to predict whether the sample is malicious or not. Figure 1 provides a good description of basic ML classification framework.

Figure 1. Basic ML framework.



B. Dataset

Our dataset, a total of 201368 PDF samples, can be divided into two classes: 28332 benign and 173036 malicious samples.

Among those, 156035 malicious samples are downloaded from VirusShare; 9000 malicious samples are from the Contagio dataset, the rest are obtained from two popular search engines. Besides, we obtained the open source dataset, in the present of feature vectors, collected for PDFRate [4] evaluation. This dataset contains 20,000 balanced samples, with 5,000 benign and 5,000 malicious samples from Contagio dataset, and 5,000 benign samples obtained from Google as well as 5,000 malicious samples from VirusTotal. Further, we randomly select around 2000 malicious documents from our dataset to generate around 10,000 adversarial samples targeting deployed ML model for 'model evasion attack' in Session 4.

From Figure 2, we can see that 40% of the samples are compatible with PDF version 1.6 and the rest are mostly distributed from version 1.3 – 1.5. Lower PDF version usually means higher possibility of being exploited. And thus we suggest end users update their PDF Viewer to the latest compatible version in order to eliminate known vulnerabilities.

Figure 2. Dataset divided by PDF Version

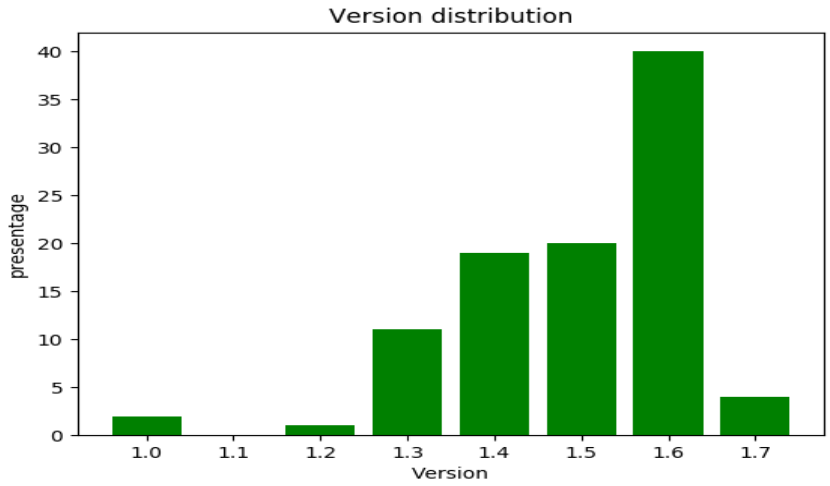


Figure 3. Dataset divided by Diversity

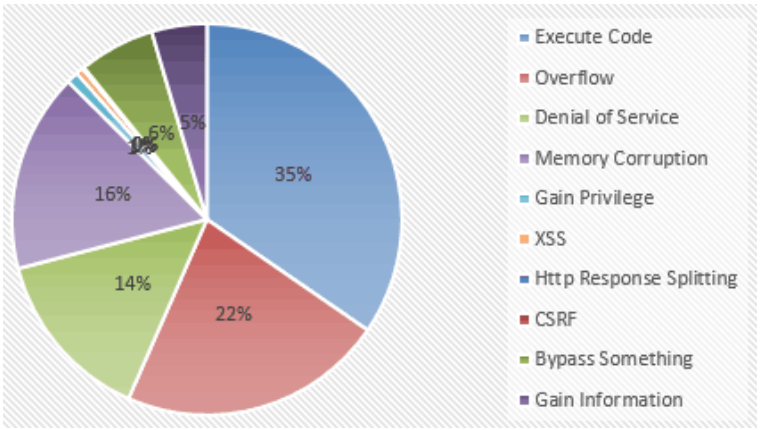
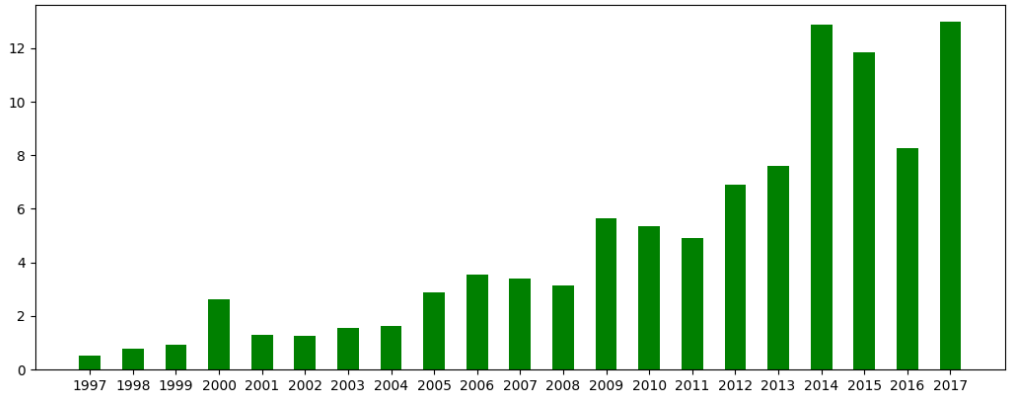


Figure 4. Dataset divided by Time



From Figure 3, we can see that 35% of the samples are Execute Code, followed by 22% of Overflow, 16% of Memory Corruption, 14% Denial of Service. We can see that our ML technique works effectively against a large and diverse dataset, which therefore proves the generality of the technique we propose.

From Figure 4, we plot the whole dataset by time. As shown, we collect samples till 2017, followed by roughly a decreasing collection size as the year get older. Our dataset reflects the latest malicious behavior changes and technology used by the malware authors.

One of the most important and interesting research problem is to evaluate the prediction capability of the ML model, specifically:

- Trained by the data from earlier versions such as 1.0 – 1.5, what is the prediction capability against the version from 1.6 – 1.7;
- Trained by the data from major exploitation type such as Execute Code and Overflow, what is the prediction capability against the other exploitation type? Is the model able to predict the file as malicious?
- Trained by the data till 2010, what is the prediction capability against the most recent years?

We address some of the problems mentioned above in the following sections.

C. Feature Engineering

Because of the limited space of this paper, we intentionally ignore the introduction of the file format of PDF files and refer readers to the latest official open standard for PDF files. In general, PDF file format is diverse and complex. Thanks to the flexibility of the document open standard, we can even run programs in PDFs like the executables. Here, we ONLY focus on parsing the structure, content and metadata of the files and select features manually by our security experts. Significant amount of previous works in feature engineering suggest that the combination of structure and meta-based features performs well enough in terms of model prediction accuracy, even we do NOT directly target to analyze the JavaScript code snippet. This is because analyzing the JavaScript code directly usually involves heavy dynamic analysis, facing with encrypted or obfuscated code and plus significant runtime overhead. While we can easily infer the maliciousness of files by some basic statistics embedded in the file structure and metadata. More, the dynamic approach is not going to be at scale while the static one does.

As shown in Table 2, we apply some basic statistics in calculating the average value of each

feature and find out some representative features which the average values between benign and malicious samples are significantly different:

Features such as count_font and count_box: There are several objects like font, box contained in the benign samples for content description. However, malicious files do not aim at describing information, instead they run 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 many more than those in malicious files. When calculating the number of obj in the same page, that the 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 later;

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 generally on average, 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 typical document being exploited, the value of AcroForm in a malicious sample usually doubles than that of a benign sampl;

It is also worth mentioning that the features exhibit significant interdependence. When one feature's value is modified, many others may be affected because they directly or indirectly depend on each other.

At last, we manually select around 130 features for classification, in great consideration of feature representative, time efficiency and robustness etc.

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

D. Classification Alg. Selection

For classification, we select several algorithms including Decision Tree, Random Forest and SVM for comparison. As shown in Table 3, Random forest is selected for its good efficiency - minute level training time and millisecond level prediction time, good effectiveness - with accuracy as high as 99%, as well as other advantages including excellent robustness (detail in Section 4), easy to interpret and good generalization capability.

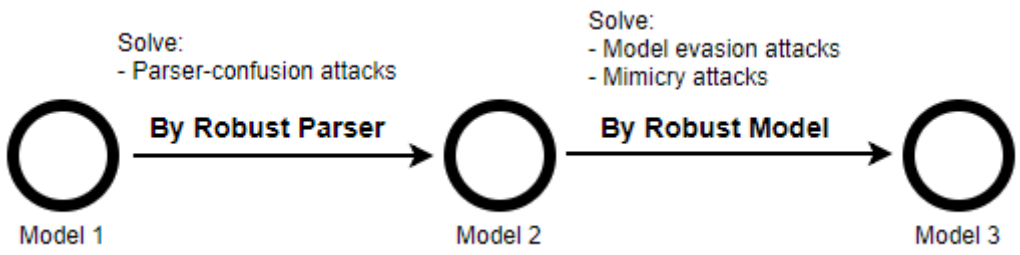
The output of random forest is essentially an ensemble of a multitude of decision trees. That said, Random forest is an ensemble classifier applying the technique of bagging training data. Each node in a decision tree is constructed based on a randomly selected subset of features, as well as the best 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.

Table 3 Comparison of Different ML Models

	SVM	Decision Tree	Random Forest
Accuracy	75.23%	82.41%	**99.64%**
Training Time (the whole dataset)	58m18s	4s	**56s**
Prediction Time (each sample)	1.2ms	0.1ms	**0.1ms**

E. Model Evolution

Figure 5. Model Evolution



We have 2 major updates for our model during experiment and each model provides a probabilistic estimate of the PDF's maliciousness. All 3 classifiers deployed by us produce the output of their decision function, i.e., a real value in the interval [0,1] denoting the percentage of decision that have labeled the submitted file as malicious. We apply the default value of threshold (0.5) when predicting but regard adjusting the threshold as one kind of effective defense strategy.

Model 1: Use peepdf (https://github.com/jesparza/peepdf) as the external parser for feature extraction. After computation and quantization, these features can be used for training and prediction. We extract 133 features which contain static attributes of structure such as count_font, size and count_startxref, content such as title_oth and subject_lc, metadata such as producer_oth and producer_len. But the limitation for peepdf is obvious: Nearly half of the PDF files can NOT be parsed correctly by the external parser. This is due to the defected file structure or intended file obfuscation technique.

Model 2: In this model, with the goal to conquer the major deficiency for model 1, we develop a much more robust external parser on top of the mimicus framework (https://github.com/srndic/mimicus). By using this new external parser, nearly all the PDF files can be properly parsed. For the training of Model 2, we initially use the balanced dataset for training and testing. This balanced dataset includes 20,000 malicious and 20,000 benign examples selected randomly from the whole dataset. Besides, we extract 135 features for Model 2. The main algorithm for model is random forest. After grid search and model parameter tuning, the accuracy of Model 2 increases to 99.99%, with a false positive rate of 0.012%. We then serve our models to major commercial cloud service providers (Model-as-a-Service).

Model 3: The big difference between Model 2 and Model 3 is the robustness. Model 2 has the assumption that during model serving, a benign working environment is provided. But this assumption is largely not held in most situations in the context of malware detection. In Model 3, we assume that adversaries are present and adversarial examples will be submitted to launch the model evasion attacks. We will discuss this kind of attack in detail in Session 4 and propose a few effective defense strategies we have experimented on.

4. Adversarial Machine Learning

Adversarial machine learning is a research field that lies at the intersection of machine learning and computer security [28]. It aims to enable the safe adoption of machine learning techniques in adversarial settings like spam filtering, malware detection. If the ML models are deployed online as a service, attackers can easily launch model evasion attacks or try to steal the high-value model in cloud for good profit. The goal of an adversary launching the evasion attacks is to confuse the model to provide a false negative classification. Our study addresses the case when an attacker attempts to evade detection by modifying the submitted PDF files so that its malicious functionality remains intact but the probabilistic score returned by model is significantly decreased. We focus on evasion attacks in this session and leave the model stealing attacks as future work.

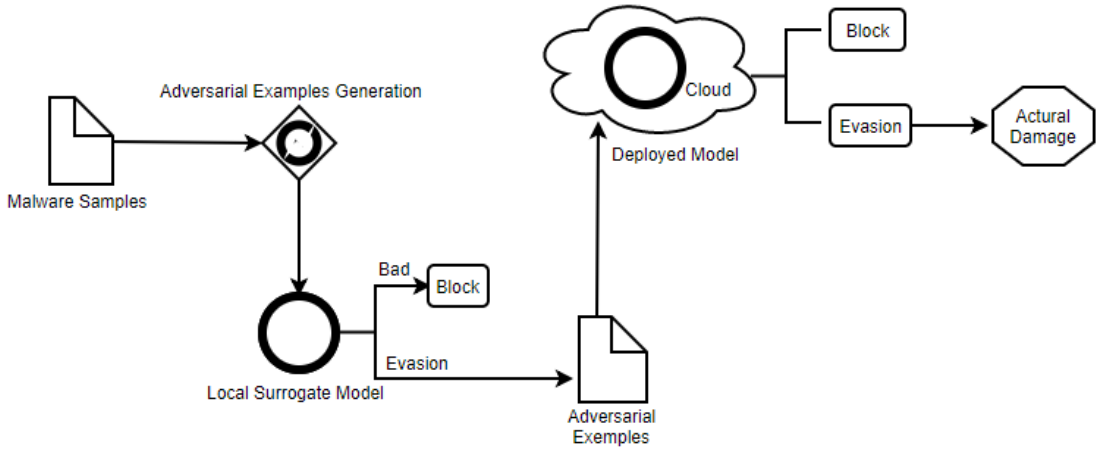
From an adversarial viewpoint, the more information about a learning-based system is available, the higher possibility that the model can be successfully fooled. Such kind of attacks has been studied thoroughly assuming attackers have full knowledge of the deployed classifier. In practice, this assumption is rarely held, especially for the online deployed system. The knowledge about the deployed classifier can be obtained from all kinds of sources. 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. We use a real, deployed ML model as a testing case, to verify the effectiveness of evasion attacks and later propose some effective model defense strategies.

We have built a framework for practical evasion strategies, and adapt several evasion algorithms for different practical application scenarios. Our experiment results reveal that the detection accuracy

of model 2 declines sharply even if it is exposed to the simplest attacks. The experiment results show that 5 potential techniques can strengthen the robustness of model against adversarial data manipulation. The rest of this session includes the following subtopics:

- Adversarial examples generation
- Potential effective defense techniques
- Case Study: How does the adversarial example work
- Model Robustness & Effective Defense Strategies

Figure 6. Model Evasion Framework



A. Adversarial Examples Generation

In this session, we are going to discuss adversarial ML in particular scenarios. To be specific, it is supposed that an attacker has obtained some information of a targeted model, such as the features extracted, the algorithm applied and the training set. It is generally believed that as attackers know more about the model, the adversarial examples generated can perform stronger model evasion attacks. In this session, we refer to the technique proposed by Nedim Smdic [4] in order to conduct evasion attacks against the learning-based model. The general idea of our evasion technique is based on insertion of dummy content into PDF files which is ignored by PDF renderers but affected the computation of features used in Model 2. Once we can influence a subset of model's features, we develop attack algorithms for adversarial examples generation.

To systematically explore the attacker's options, we define an orthogonal set of evasion strategies reflecting various degrees of available knowledge. The letters F, T or C, correspond to the feature set, training dataset and classifier algorithm respectively and are present in the name of a scenario respectively. We describe both high-level algorithms and low-level implementations for staging evasion attacks in following 4 attack scenarios:

- Attack Scenario F

  In scenario F, only the feature set is available to the adversary, to a various extent. The adversary might be aware of some or all features, mistakenly considering obsolete features as being used. He might also be able to read a subset or all features, or modify some or all features to a varying

degree. Manipulation of a sufficient subset of features is, however, required in order to be able to modify samples and proceed with evasion.

- Attack Scenario FT

  This scenario enables the adversary to take advantage of the knowledge of the targeted classifier's training dataset, in addition to the known features. Knowledge of training data enables the attacker to perform the entire attack offline before submitting the final result.

- Attack Scenario FC

  In scenario FC, the adversary knows the feature set and some details about the classifier, such as its type, parameters or the specific implementation. An adversary without any information about the training dataset at all and without a surrogate dataset has little advantage of knowing the classifier. With a surrogate dataset, they can train a surrogate classifier of the right type, yet the accuracy of this approximation depends on the quality of the gathered data. This attack can also be performed offline, similar to other attacks based on surrogate classifiers.

- Attack Scenario FTC

  The adversary has the best chance of evading the targeted classifier if he knows the details of all three classifier components. In that case, he can fully reproduce the online classifier in an offline setting, submitting the attack results only when a sufficiently good evading sample has been found. An offline mimicry attack or an offline classifier-specific attack that defeat the offline classifier have a strong probability of defeating the online one as well.

B. Case Study: How does the adversarial example work

A practical way to interpret attacks is to observe concrete changes in feature values produced by the attacks. Although it does not scale to cases with many features and files, this kind of investigation provides deep insight into the example at hand.

This session presents a concrete example of adversarial example. We select a file with CVE ID 'CVE-2013-0641' as an example. This file has the vulnerability that hackers can execute any code remotely. We apply the methodologies in the above four scenarios to vary selected samples and then check against the VirusTotal anti-virus service. The original sample is scanned by around 60 detection engines, within which 33 engines predict the sample as malicious. While only 22 engines predict the sample variant as malicious, which demonstrates strong model evasion ability of adversarial example.

Table 4 Detection Accuracy for VirusTotal

File_HASH	Origin	F	FC	FT	FTC
00ba5c43b1cec186c634c24ac21982d3 cve-2013-0641	33/61	22/60	23/60	22/60	22/60

Since most PDF detection engines are based on structure and content, once we modify the structure and content, such as adding some objects from benign samples or modifying the file size, the malicious file will have a higher chance to evade the classifiers. We compare the feature space of the file before and after variation. As shown in Table 5, this 'benignization' is evident in the provided

example. By comparing the BEFORE and AFTER columns, we see that the variation for this file includes adding an author (author* features), setting the creation (createdate_ts) and modification (moddate_ts) data into recent past etc. All changes are toward the benign class. After variation, the sample remains the malicious behavior with 10 more models being successfully evaded.

This kind of mimicry attack is well-known in the security literature. Its idea is to transform a malicious sample in such way that it mimics a chosen benign sample as much as possible, making the resulting mimicry sample harder to detect. This attack alg. is simple to implement, and can be applied to any classification algorithm. Besides, it does not necessarily depend on a specific learned classifier model. Thus, it is suitable for evaluation in every evasion scenario.

The results for the whole dataset reveal that even with the smallest amount of available information, say an ability to freely modify two thirds of the feature values, our attacks reduce the accuracy of the model from 99% to the most 2.92% (the FC Scenario).

Table 5 Changes of feature values for a subset of features in an attack. The BEFORE column shows the feature values extracted from a malicious candidate file, the adversarial example generation alg. transformed these values in feature space into a new data point (AFTER).

Table 5. Comparison of generated new malware features

Feature	BEFORE	AFTER
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

C. **Potential effective defense technique**

One of the most important components in our framework is the adversarial examples generation algorithm. The goal is to generate PDF files whose feature vectors are likely to receive low classification scores.

We randomly select 2,000 highly scored malicious samples as the original seeds to generate adversarial examples. We then use these examples to attack against Model 2. The experiments in this section evaluate the effectiveness of evasion techniques presented so far. In our evaluation protocol, we take on the role of an attacker and combine all available means to defeat an up-to-date version of Model 2. As shown in Table 6, this attack causes a great drop to model 2. In the scenario of FC, the detection accuracy of Model 2 is only 2.92%, meaning more than 97% of malicious documents are

managed to evade the classifier. This motivates us the development of model 2.1, model 2.2 and model 2.3, and finally the Model 3.

Model 2.1 is trained by increasing the size of dataset from 100K to 200K samples. Compared to Model 2, Model 2.1 has significant improvement over each of the 4 attack scenarios. But the FC and FTC case is still unacceptable for practical use. The reason for the increased dataset improving the model performance is that more data samples have been seen, the better generality the model will be.

Model 2.2 is trained by including a large of adversarial examples. This technique is called adversarial training in the context of adversarial ML. Around 7000 adversarial examples have been generated and added to our training set. Compared to Model 2, Model 2.2 has large improvement over all of the attack scenarios, especially for the FC and FTC case. The main reason for this is because the model now has experience of differentiating the adversarial samples in training, and thus has a good performance in testing as long as the training and testing sets have the same data distribution.

The major difference between Model 2.3 and Model 2 is the adjusted threshold. It is observed that the probability distribution for the adversarial examples have been significantly decreased and mixed with the benign ones compare to their original malicious seeds. If we still apply the original threshold (0.5), lots of adversarial examples will be unnoticed and thus an increase of false negative rate occurs. By decreasing the detection threshold from 0.5 to 0.43, we increase the sensitivity of the model to embrace more suspicious files for further investigation.

The final Model 3 is the combination of Model 2.1, Model 2.2 and Model 2.3, which has been trained by a much larger dataset including a significant number of adversarial examples, and has adjusted its thresholds for optimized performance. It is not surprised that Model 3 is superior to all the models we have evaluated so far. It is also not surprised that the technique adversarial training contributes the most among the 3 defense techniques.

Table 6. Different attack scenarios and accuracies

Attack Scenarios	Adversarial Examples	Model 2	Model2.1	Model2.2	Model 2.3	Model 3
F	2000	71.18%	96.71%	89.43%	71.03%	98.65%
FC	240*	2.92%	12.50%	81.95%	2.85%	88.89%
FT	2000	84.25%	96.76%	98.58%	85.89%	98.98%
FTC	2000	15.83%	18.71%	86.21%	36.82	92.83%

*This number should be 2000 instead of 240 in an ideal experiment setting, but 240 is also good and can lead to the same conclusions. We are not able to use 2000 for testing because of some technical difficulties when using the mimicus framework.

D. Model Robustness

In our last experiment, we have investigated the robustness of defensive mechanisms against to our evasion technique. We propose to use 5 kinds of techniques to increase the robustness of ML model.

If the feature sets have been exploited by attackers, we can modify the weight of features or delete the related features and retrain the ML models. As shown in Figure 7, the top 30 features are sorted by importance for Model 3 in ascending order. It is not surprised that features such as count_font,

count_javascript, size, count_obj and count_endobj account for most of the weights. In a white box setting of the adversarial environment, we assume the adversary knows the 5 most important features. As an effective defense strategy, we suggest removing those exploited features and retrain the model immediately.

Figure 7. Top 30 features

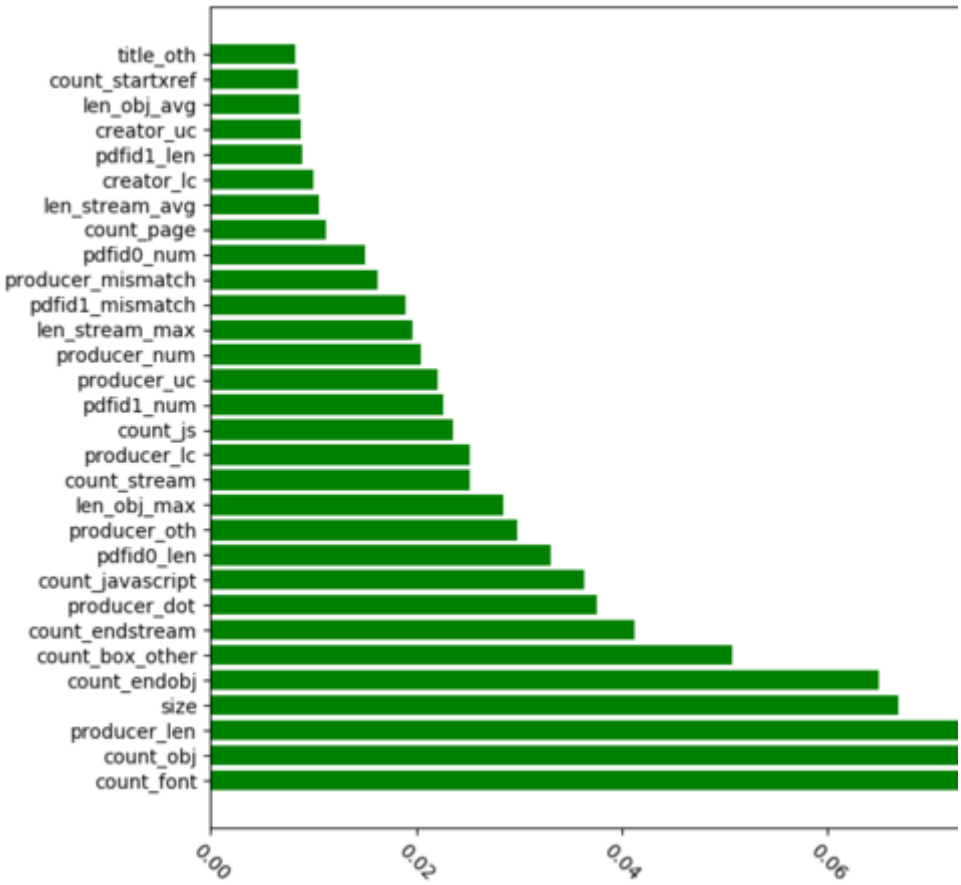


Table 7 shows the detection accuracy of the retrained Model 3 when removing top 5 features sequentially. As shown, when the classifier is trained by using all features above, the accuracy rate of model is around 99.82%. If we delete up to top five features, the accuracy is almost stable. That concludes our model can confront grey box attacks even using the rest of fairly important features for training. The top 5 related features here are (note: with the format to be (index number, feature name)): (1, count_font), (2, count_javascript), (3, size), (4, count_obj) and (5, count_endobj)

Table 7 Accuracy of Model after deleting top 5 related features

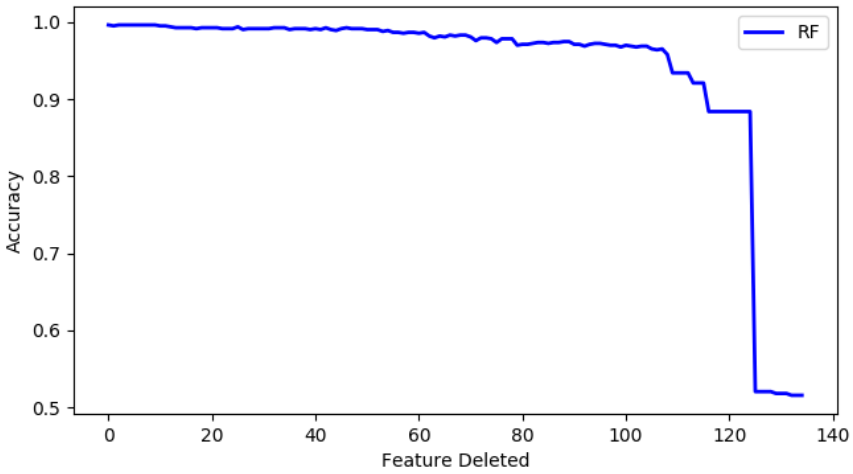
Features Deleted	Accuracy
[]	99.82%
[1]	99.52%
[1,2]	99.52%
[1,2,3]	99.64%
[1,2,3,4]	99.64%

[1,2,3,4,5]	99.64%

More, we assess the model robustness by the effectiveness of features. First, we sort the features by their importance, and then delete the most important features one by one. As shown in Figure 8, the curve represents the accuracy of Model 3 when features are deleted one by one. When the number of features are decreased to 100, the accuracy of the retrained model still maintains at around 90%. This demonstrates:

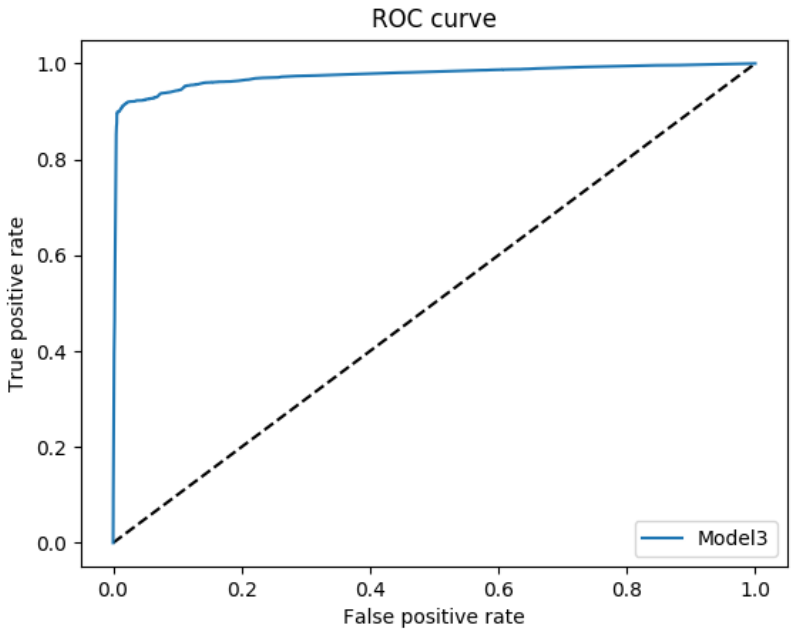
- Despite the high weights of individual features, if they are deleted, the accuracy of model only declines a small margin;
- The interaction of 'Medium Weight' features can make the model robust, and reduce the effect caused by the deletion of individual important features significantly;
- 'Medium Weight' features can help to effectively prevent the 'Model Evasion attacks';

Figure 8 Detection Accuracy as features are deleted



To assess the prediction performance of Model 3, we randomly divide the dataset into two classes: 90% for training and 10% testing; and 10-fold cross validation is applied. As shown in Figure 9, the area below the ROC curve is around 1, indicating good prediction performance.

Figure 9. ROC Curve



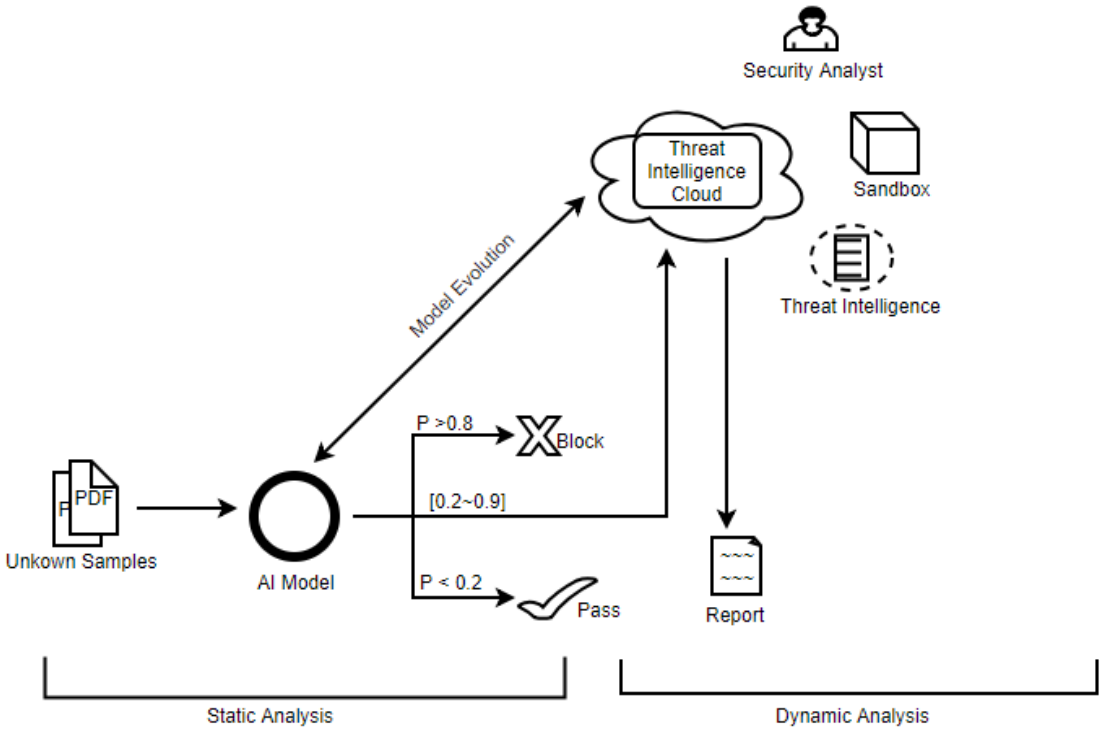
In summary, for the problem of model robustness, we have proposed 5 defense techniques to harden our model. They are:

- Increase the number, the diversity and the timeliness of the training dataset (Model 2.1)
- Adversarial Training (Model 2.2)
- Adjust the model threshold (Model 2.3)
- Remove the features being exploited (Figure 8)
- Hide classifier related info such as probabilities

5. Application: AI Firewall

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

Figure 10. Overview of static & dynamic analysis



In the current development of network and gateway security products, the capability of performing malicious file scanning effectively and efficiently at layer 7 (the network application layer) is the international standard. The industry has strict demands on this product feature. That is, a good detection module should include (1) Millisecond latency for single-file detection, and (2) 99% accuracy while maintaining a false positive rate of less than 0.01%.

The reason behind the demand for low latency is obvious: Since a security module is inline in a working pipeline, high latency will lead to an increased packet drop rate and occasionally data loss. This is strictly forbidden for security devices. In the past few years, with the rapid development of malware, the former industry best practice, the pattern-matching engine has gradually been used less in mainstream security applications. There are two main reasons for this: (1) In order to meet the requirement of high detection accuracy, a large number of security analysts are needed to write patterns, which is a manual process that doesn't scale at all; and (2) Considering the fast-growing size of patterns, the time for core operation - pattern matching grows exponentially. These conditions have inspired us to discover a better engine rooted in AI.

By 2018, we managed to integrate our AI maldoc detector into a firewall, in the hope of replacing the old engine. Although both the old and new engines are static-analysis-based engines, the improvement of shifting from the old to the new AI engine has been tremendous. First, the AI engine does not need to be updated frequently because it can detect previously unseen malware effectively for at least a year. According to our experiment results, the average update frequency for our AI engine is half a year, which is much longer if compared to the 2-week period of the old pattern-matching engine. Moreover, the AI-based engine enjoys low resource consumption in execution. According to our study, during the phase of model prediction, the AI engine can only consume as much as a one-third of CPU and 50% of memory. The CPU portion of consumption is mainly due to computations such as feature extraction and confidence-score computation. The memory portion of consumption is mostly due to the fact that an AI-based model must be sited entirely in main memory when predicting.

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

The truly interesting part arises when the output probability is in the range between 0.1 and 0.9. When this happens, we by default upload the samples to our Threat Intelligence Cloud where multiple dynamic analyses will be performed by a mixed of techniques such as Sandbox, Threat Intelligence, and Security Teams. According to our heuristics in a typical usage scenario, 99% of the files can be processed inline while approximately 1% of the files are uploaded. Dynamic analysis from our cloud plays a great complementary role in the static analysis method inline. By combining the two, we can now completely provide end users with a more advanced, AI-enabled security solution. We have made our cloud service a subscription service and freely open to the research community.

6. Conclusions

In this paper, we introduce the design and implementation of a PDF file classifier based on machine learning (ML). Our experimental results reveal that the classifier can achieve a detection accuracy greater than 99% and has a false-positive rate of less than 0.01%. In addition, compared with a rules-based model, the time and space performance of CPU and memory are improved significantly.

We use a great amount of effort to study the proposed AI-based security application and at the same time give equal weight to the study of securing the ML models. Specifically, we complete the following work: (1) Simulation of a scenario in which attackers generate the adversarial examples in a way to evade the classifiers and (2) Proposing several effective defense techniques for model robustness.

ML based maldoc classifier is an important research topic in the fields such as social engineering and malware analysis. In the future, we plan to focus on the following:

- Malicious PDF file detection based on deep learning.
- Optimization of the static and dynamic analysis engines.
- More file formats supported such as .docx, .pptx, and others.

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.

[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 Threat Report 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

[28]. Adversarial Machine Learning. https://en.wikipedia.org/wiki/Adversarial_machine_learning

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