Malware Detection in PDF Files using Machine Learning

4 authors, including:

Aliénor Damien
Laboratoire d'Analyse et d'Architecture des Systèmes (LAAS)
8 PUBLICATIONS **65** CITATIONS

Malware Detection in PDF Files Using Machine Learning

Bonan Cuan, Aliénor Damien, Claire Delaplace, Mathieu Valois

HAL Id: hal-01704766
https://hal.archives-ouvertes.fr/hal-01704766v2

Malware Detection in PDF Files Using Machine Learning

Bonan Cuan[1], Aliénor Damien[23], Claire Delaplace[45] and Mathieu Valois[6]

[1]*INSA Lyon, CNRS, LIRIS, Lyon, France*

[2]*Thales Group, Toulouse, France*

[3]*CNRS, LAAS, Toulouse, France*

[4]*Univ Rennes 1, CNRS, IRISA, 35000 Rennes, France*

[5]*Univ. Lille, CRIStAL, 59655 Villeneuve d'Ascq, France*

[6]*Normandie Univ, UNICAEN, ENSICAEN, CNRS, GREYC, 14000 Caen, France*

bonan.cuan@insa-lyon.fr, alienor.damien@laas.fr, claire.delaplace@irisa.fr, mathieu.valois@unicaen.fr

Keywords: Malicious PDF detection, SVM, Evasion attacks, Gradient-Descent, Feature Selections, Adversarial Learning

Abstract: We present how we used machine learning techniques to detect malicious behaviours in PDF files.
At this aim, we first set up a SVM (Support Machine Vector) classifier that was able to detect 99.7% of malware. However, this classifier was easy to lure with malicious PDF files, which we forged to make them look like clean ones. For instance, we implemented a gradient-descent attack to evade this SVM. This attack was almost 100% successful. Next, we provided counter-measures to this attack: a more elaborated features selection and the use of a threshold allowed us to stop up to 99.99% of this attack.
Finally, using adversarial learning techniques, we were able to prevent gradient-descent attacks by iteratively feeding the SVM with malicious forged PDF files. We found that after 3 iterations, every gradient-descent forged PDF file were detected, completely preventing the attack.

1 INTRODUCTION

Billions of Portable Document Format (PDF) files are available on the web. Not all of them are as harmless as one may think. In fact, PDF files may contain various objects, such as JavaScript code or binary code. Sometimes, these objects may be malicious. A malware may try to exploit a flaw in a reader in order to infect the machine.

In 2017, sixty-height vulnerabilities were discovered in Adobe Acrobat Reader (CVEDetails, 2017). More than fifty of them may be exploited to run arbitrary code. Every reader has its own vulnerabilities, and a malicious PDF file may find ways to make use of them.

In this context, several works propose the use *machine learning* to detect malicious PDF files, (e.g. (Kittilsen, 2011; Maiorca et al., 2012; Borg, 2013; Torres and De Los Santos, 2018)). These works rely on the same main idea: select discriminating features (i.e. features that are more likely to appear in malicious PDF files) and build what is called a classifier. For a given PDF file, this classifier will take as input the selected features and, considering the number of occurrences of each of them, will try to deter-

mine if the PDF file contains malware or not. Many approaches may be considered, both for the choice of the features and for the design of the classifier. In fact there are many classification algorithms that can be utilised: Naive Bayes, Decision Tree, Random Forest, SVM... The authors of (Maiorca et al., 2012) first describe their own way to select features, and use a Random Forest algorithm in their classifier, while the author of (Borg, 2013) relies on the choice of features proposed by Didier Steven (Stevens, 2006), and uses a SVM (Support Vector Machine) classifier. Both approaches seem to provide accurate results.

However, it is still possible to bypass such detection algorithms. Several attacks have been proposed (e.g. (Ateniese et al., 2013; Biggio et al., 2013)). In (Biggio et al., 2013), the authors propose to evade SVM and Neural Network classifiers using gradient-descent algorithms. Meanwhile, the authors of (Ateniese et al., 2013) explain how to learn information about the training dataset used by a given target classifier. To do so, they build a meta-classifier, and train it with a set of classifiers that are themselves trained with various datasets. These datasets have different properties. Once their meta-classifier is trained, they run it on the classifier they aim to attack. Doing

so, their goal is to detect interesting properties in the training dataset utilised by this classifier. Hopefully they can take advantage of this knowledge to attack said classifier.

We worked on three aspects of malware detection in PDF files.

First we implemented our own PDF file classifier, using SVM algorithm, as it provides good results. We explored different possibilities for features selection: our initial choice was based on (Stevens, 2006) selection. We refined this choice the following way: from the set of available features, we selected those which appeared to be the most discriminating in our case. We trained and tested our SVM with a dataset of 10 000 clean and 10 000 malicious PDF files from the Contagio database (Contagio Dump, 2013), and we also tuned the SVM to study its behavior. We came up with a classifier that had more than 99% success rate.

In the second part of our work, we study evasion techniques against our classifier: how to edit malicious PDF files such that they are not detected by our classifier. One first naive attack consists in highly increasing the number of occurrences of one arbitrary feature but can be easily countered using a threshold. A more interesting one is the gradient-descent attack. We implemented this attack following the description of (Ateniese et al., 2013).

Finally, we propose ways to prevent this attack. Our first intuition is to set up a threshold value for each feature. We also present a smarter features selection to make our SVM more resistant against gradient-descent attacks. Finally we suggest updating our SVM using *adversarial learning*. We tested these counter-measures and they allowed us to stop almost all gradient-descent attacks.

2 MALWARE CLASSIFIER

In this section, we present the technical tools for PDF file analysis, machine learning, and the different settings we used in our methodology. Then we present the first results for that classifier using different optimization approaches.

2.1 Useful Tools

We first recall the structure of a PDF file, and explain how this knowledge can help us to detect malware.
PDF file Analysis. A PDF file is composed of objects which are identified by one or several tags. These tags stand for features that characterise the file. There are several tools made to analyse PDF files. In this work,

```
PDFiD 0.2.1 CLEAN_PDF_9000_files/rr-07-58.pdf
 PDF Header: %PDF-1.4
 obj                   23
 endobj                23
 stream                 6
 endstream              6
 xref                   2
 trailer                2
 startxref              2
 /Page                  4
 /Encrypt               0
 /ObjStm                0
 /JS                    0
 /JavaScript            0
 /AA                    0
 /OpenAction            0
 /AcroForm              0
 /JBIG2Decode           0
 /RichMedia             0
 /Launch                0
 /EmbeddedFile          0
 /XFA                   0
 /Colors > 2^24         0
```

Figure 1: Output of PDFiD

we used the PDFiD Python script designed by Didier Stevens (Stevens, 2006). Stevens also selected a list of 21 features that are commonly found in malicious files. For instance the feature /JS indicates that a PDF file contains JavaScript and /OpenAction indicates that an automatic action is to be performed. It is quite suspicious to find these features in a file, and sometimes, it can be a sign of malicious behaviour. PDFiD essentially scans through a PDF file, and counts the number of occurrences of each of these 21 features. It can also be utilised to count the number of occurrences of every features (not only the 21) that characterise a file.

Figure 1 shows an output example of PDFiD. For each feature, the corresponding tag is given on the first column, and the number of occurrences on the second one.

We can represent these features by what we call a *feature vector*. Each coordinate represents the number of occurrences of a given feature.
Supervised Learning and PDF file Classification. Machine Learning is a common technique to determine whether a PDF file may or may not contain malware. We consider a classifier function *class* that maps a feature vector to a *label* whose value is 1 if the file is considered as clean and -1 otherwise. To infer the *class* function, we used what is called *Supervised Learning* techniques.

We considered a dataset of 10 000 clean PDF files and 10 000 containing malware from the Contagio database (Contagio Dump, 2013). By knowing which files are clean, and which are not, we were able to label them. We then split our dataset into two parts. The first part has been used as a *training dataset*. In other words, for every feature vector x of every file in the dataset, we set $class(x)$ to 1 if the PDF file was clean and $x = -1$ if it contains malware. We used a classification algorithm to infer the *class* function using this knowledge. The second part of our dataset was then

used to test if the predictions of our classifier were correct.

Usually, between 60% and 80% of the dataset is used for training. Bellow 60%, the training set may be to small, and the classifier will have poor performances when trying to infer *class*. On the other hand, if more than 80% of the dataset is used for training, the risk of overfitting the SVM increases. Overfitting is a phenomena which happens when a classifier learns noise and details specific to the training set. Furthermore, if one uses more than 80% of the dataset for training, one will have to test the classifier with less than 20% of the data, which may not be enough to provide representative results. For these reasons, we first chose to use 60% of our dataset for training, and saw how the success rate of our classifier evolved when we increased the size of the training set up to 80%.

We used a *Support Vector Machine* algorithm (SVM) as the classification algorithm. Basically this algorithm considers one scatterplot per label, and finds a hyperplan (or a set of hyperplans when more than two labels are considered) to delimit them. Usually, it is unlikely that the considered set is linearly separable. For this reason, we consider the problem in a higher-dimensional space that will hopefully make the separation easier. Furthermore, we want the dot-product in this space to be easily computed, with respect to the coordinates of the vectors of the original space. We define this new dot-product in term of the *kernel function $k(x,y)$*, where x and y are two vectors of the original space. This well known trick is due to (Aizerman et al., 1964) and was first applied to a SVM in (Boser et al., 1992). It allows us to work in a higher dimensional space, without having to compute the coordinates of our vectors in this space, but only with dot-products, which is computationally less costly.

We denote by $n_{features}$ the number of features we consider (i.e. the size of our vector), and we choose to use a *Gaussian Radial Basis Function* (RBF) kernel:

$$k(x,y) = \exp(-\gamma \cdot ||x-y||^2),$$

with parameter $\gamma = 1/n_{features}$.

2.2 Experimentations

We implemented our classifier in Python, using the Scikit-learn package. We initially used PDFiD with the 21 default features to create our vectors. Then, we trained our SVM on 60% of our shuffled dataset. We used the remaining 40% data to test our SVM and calculate its accuracy: the ratio (*number of well classified PDF files*)/(*total number of PDF files*). After

having split our dataset, we obtained an accuracy of 99.60%. Out of 2 622 PDF files containing malware, only 29 have been detected as clean (1.11%). Out of 5 465 clean files, only 3 were considered containing malware (0.05%). To compare with related work, the SVM used in (Kittilsen, 2011) has a success rate of 99.56%, out of 7 454 clean PDF files, 18 were detected as containing malware. Out of 16 280 files containing malware, 103 were detected as clean.

Using different settings. To go further in our experimentations, we slightly modify our SVM. For instance, we tested other values of the *gamma* parameter in the RBF kernel. We also tested the other kernels proposed by Scikit-learn package. We figured out that using the RBF kernel with default $\gamma = 1/n_{features}$ parameter yields to the best results, considering all the settings we tried.

Change splitting ratio. We changed how we split the initial dataset into training and testing sets. We saw that if 80% of our dataset is used for training and 20% for testing, then the success rate was slightly higher. We also used cross-validation: we restarted our training/testing process several times with different training sets and testing sets, and combined the results we obtained. We did not notice any overfitting issue (i.e. our SVM does not seem to be affected by the noise of the training set).

Change the default features. Instead of choosing the default 21 features proposed by Didier Stevens (Stevens, 2006), we tried other features selections. In the whole set, we found more than 100 000 different tag types. Considering that it requires 12Gb of memory to compute the vectors of each PDF file in a SVM, using 100 000 tags would be too much. A first strategy implemented was to select features by their frequency in the files (e.g. "90% frequency" means that 90% of the files in the dataset have this feature). In one hand, we chose the most common features in the clean PDF files, in the other, we chose the most frequently used features in the malicious files, and combine them into a sublist of features. Once this selection was made, the resulting list could be merged with the 21 default features. A second strategy, that we call better sublist selection, was to remove non-significant features from the first sublist, by removing features one by one and computing the SVM (with cross-validation) to check if accuracy improved or deteriorated. Note that these two strategies can be combined together. In practice, we selected initial sublist : from 21 default features and/or frequency selection and apply the better sublist selection. Table 1 shows some results found by applying these two strategies, regarding the original result.

Results. The application of the frequency selection

Table 1: Results of features selection using: PDFiD default features (D), frequency selection on all features (F), the merge of these two lists (M), and better sublist selection (BS) applied to each of these feature set.

	Features selection	Accuracy (cross-validation)	Nb of features	Time to compute SVM
(D)	Default 21 features	99,43%	21	21,17s
(F)	Frequency (90%)	99,22%	31	54,49s
(M)	Frequency (95%) + default features	99,40%	39	47,66s
(D+BS)	Sublist from 21 default features	99,68%	11	7,03s
(F+BS)	Frequency (80%) + Sublist	99,63%	13	11,30s
(M+BS)	Frequency (80%) + default features + Sublist	99,64%	10	15,89s

method did not improve the accuracy of our SVM, and increased significantly the number of features, making the training and testing of the SVM much slower. Applying the better sublist selection method improved the SVM's accuracy significantly and kept a reasonable amount of features. We also saw that applying the better sublist selection method to the 21 default features, improved the accuracy (+0.25%). The resulting set contains only 11 features, significantly reducing the time to train and test the SVM.

3 EVASION ATTACKS

In this section, we propose some evasion attacks to trick the trained SVM. The main goal of these attacks is to increase the amount of objects in the infected PDF files so that the SVM considers them as clean. To this end, the modifications performed on the files should not be noticeable by the naked eye. Removing objects is a risky practice that may, in some cases, change the display of the file. On the other hand, adding empty objects seems to be the easiest way to modify a PDF file, without changing its physical appearance.

We consider a white box adversary. In this model, the adversary has access to everything the defender has, namely: the training dataset used to train the classifier, the classifier algorithm (here SVM), the classifier parameters (kernel, used features for vector, threshold, ...), and infected PDF files that are detected by the classifier

This attacker is the most powerful one, since she knows everything about the scheme she's attacking.
Naive attack. The first implemented attack to lure the classifier is the component brutal raise. Given the feature vector of a PDF file, the attacker picks one feature and increments it until the vector is considered as clean by the classifier. The choice of the feature

is either done arbitrarily or with the same process as feature selection in section 2.2.

A quick counter-measure that can be applied is to ignore the surplus number of features when this number is too high, and consider it as the maximum permitted value. To implement this idea, we used a threshold value. A threshold is a value that is considered as the maximum value a feature can take. For example, if the threshold is 5, the original feature vector $x = (15, 10, 2, 3, 9, 1)$ would be cropped to become the vector $x' = (5, 5, 2, 3, 5, 1)$.

We experimented this naive attack with the default features from PDFiD, and figured out that, if we set up a threshold of 1, this naive attack is completely blocked.

Gradient-Descent. The gradient-descent is a widely used algorithm in machine learning to analytically find the minimum of a given function. Among its various applications, we were particularly interested in how it can be utilised to attack SVM classifiers. Given a PDF file of feature vector x, that contains malware and has been correctly classified, the goal is to find a vector x' on the other side of the hyperplan, so that the difference between x and x' is the smallest possible. Usually, to quantify this difference, the L_1 distance is utilised. In other words, given a feature vector x such that $class(x) = -1$, we aim to find a vector x' such that $class(x') = 1$ and

$$||x - x'||_1 = \sum_i |x_i - x'_i|,$$

is minimized. The gradient-descent algorithm tries to converge to this minimum using a step-by-step approach: first, initialise x^0 to x, then at each step $t > 0$, x^t is computed to be equal to:

$$x^t = x^{t-1} - \varepsilon_t \cdot \nabla class(x^{t-1}),$$

with ε_t a well chosen step size and $\nabla class(x^{t-1})$ is the gradient of the *class* at point x^{t-1}. The iteration

terminates when $class(x^t)$ is equal to 1.

An illustration of the result of this attack is shown on Figure 2 where features 1 and 2 are slightly increased to cross the hyperplan.
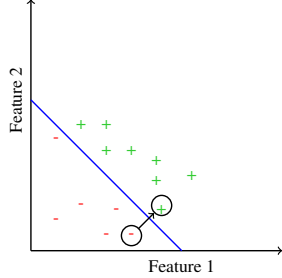

Figure 2: Example of attack using gradient-descent

This attack does not consider components individually but as a whole, allowing the algorithm to find a shorter difference vector than with the naive attack. Hence the L_1 distance between the crafted and the original vector is significantly lower than with the naive attack, it results in a crafted PDF file that has been way less modified.

Using this attack we conducted two experiments: the first was to compute its theoretical success rate and the second its success rate in practice.

Theoretical success rate: to compute the theoretical success rate, we took the feature vector of every infected file and ran the gradient-descent to get the feature vector that lures the classifier. By this experiment, we found that 100% of the forged vectors are detected as clean by the classifier (namely every gradient-descent succeeded).

Practical success rate: to compute the practical success rate of the attack, we ran the gradient-descent on the vectors of every PDF files and then reconstructed new file according to the crafted feature vector.

Remark: If we denote by m the number of selected features considered by the SVM, the gradient-descent computes a vector $x' \in \mathbb{R}^m$, however only an integer number of objects can be added to the PDF file, thus a rounding operation is needed in practice. For this attack we rounded component values to the nearest integer (the even one when tied).

Due to the rounding operation, this practical attack had a 97.5% success rate instead of a 100% theoretical success rate. In previous work (Biggio et al., 2013), the success rate was about the same, even if they made a deeper analysis of it using various settings of their SVM.

4 COUNTER-MEASURES

The gradient-descent attack has an impressively high success rate. This is due to the huge degree of freedom the algorithm has. Every component of the vector can be increased as much as required.

Hence, to counter the gradient-descent attack, one would reduce the degree of freedom of the algorithm. This can be achieved in three different ways: applying a threshold, smartly selecting features that are the hardest to exploit, and finally a mix of both solutions.

Another approach to counter this attack is to restart the training of the SVM with maliciously forged files: it is called _adversarial learning_.

4.1 Vector Component Threshold

Once again, the threshold is defined by $t \in \mathbb{N}^*$ due to the discreteness of the number of PDF file objects. To choose t, we have used algorithm 1. We considered a SVM with the 21 default features of PDFiD.

Algorithm 1 Calculate the best threshold to block as many attacks as possible

---

$t \leftarrow 20$
$s(20) \leftarrow$ success rate of gradient-descent with $t = 20$
while $t > 0$ **do**
  apply t on each forged feature vector x
  compute success rate $s(t)$ of gradient-descent
  if $s(t) > s(t+1)$ **then**
    return $t + 1$
  end if
  $t \leftarrow t - 1$
end while
return t

Algorithm 1 decreases the threshold until a local minimum success rate of the gradient is found (namely when most attacks are blocked).

Remark: Algorithm 1 assumes that the function $s(t)$ is continuous. Hence, by the intermediate value theorem, algorithm 1 can converge. Moreover, we only retrieve a local minimum, not a global one. Despite these imprecisions, our counter-measure is rather efficient in practice (cf. Table 2).

All in all, a threshold of 1 allows the SVM to block almost every attack while keeping a reasonably good accuracy (the difference between no threshold and a threshold of 1 is about 0,10%).

We also applied a threshold of 1 on a SVM constructed from different lists of features (see section 2.2). The corresponding results are presented in Table 3.

Table 2: Percentage of Gradient-Descent attacks stopped depending on the threshold value

Threshold	Attack prevention (theory)	Accuracy of SVM
5	0%	99,55%
4	0%	99,57%
3	29%	99,63%
2	38%	99,74%
1	99,60%	99.48%

Table 3: Features selection global method application results. D means that we used the default features, and F the frequency selection.

Features set	Attacks stopped (theory)	Accuracy	#features
21 D	99,60%	99,37%	7
F (80%)	91,00%	99,45%	9
F (90%)	100,00%	98,00%	30
F (80%) + D	21,00%	99,61%	17
F (90%) + D	96,40%	99,46%	29

Results Threshold usage allows to keep a very good accuracy while reducing the success rate of gradient-descent attack, but this reduction is not optimal and depends a lot on the features list utilised in the SVM.

4.2 Features Selection - Prevent Gradient-Descent

Because the gradient-descent attack uses some vulnerable features, another idea of counter-measure is to select only features that are less vulnerable to this attack. As shown by Figure 3, we selected the features that an adversary has no interest in modifying, in order to make a malicious file pose as a clean one. In other words, increasing the number of occurence of these features will never allow the gradient-descent to get closer to the hyperplan we aim to cross.
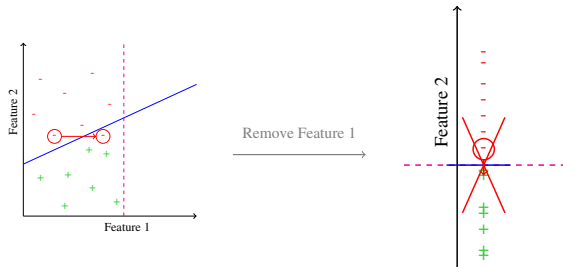

Figure 3: Suppression of features vulnerable to gradient-descent attack (Feature 1 is vulnerable here, and Feature 2 is not)

To detect the vulnerable features from a list, we used algorithm 2. This algorithm computes a SVM with the current feature list and performs the gradient-descent attack on it. At each iteration, the features used by the gradient-descent attack are removed from the current feature list. The algorithm stops when the feature list is stable (or empty).

Algorithm 2 Find the features vulnerable to Gradient-Descent attack and remove them

---

    $features\_list \leftarrow initial\_list$
    $GD\_features\_used \leftarrow initial\_list$
    while $GD\_features\_used \neq \perp$ **do**
      compute SVM for $features\_list$
      apply gradient-descent attack to current SVM
      $GD\_features\_used \leftarrow$ features used by gradient-descent Attack
      $features\_list.remove(GD\_features\_used)$
    end while
    return $features\_list$

Global features selection. To select the final list of features, the following steps are applied: We selected the initial sublist : from 21 default features and or frequency selection (see 2.2), apply the better sublist selection (see 2.2), apply gradient-descent resistant selection (Algorithm 2), apply Better Sublist selection (see 2.2) We used different parameters here (frequency, force 21 default features or not), and the corresponding results are presented in Table 4.

Remark: accuracy is not taking into account the forged file, only the initial dataset of 10 000 clean and 10 000 malicious PDF files.

Results. This features selection method drastically reduces the number of features (between 1 and 3, except without application of the better selection method first), resulting in very bad accuracy results. The application of the global method on the 95% frequently used features + 21 default features only gives quite good results (98,22% of accuracy), but it is far less from the initial SVM.

Combination of Threshold & Features selection counter-measures. The threshold counter-measure previously proposed, significantly reduces the attacks whilst keepingg very good accuracy, but was not sufficient to block every gradient-descent attacks. The features selection counter-measure stopped all gradient-descent attacks, but significantly reduced accuracy and had very different results depending on the initial features list. Hence, we tried to combined both counter-measures to obtain both good accuracy and total blocking of gradient-descent attack. Table 5 presents the results obtained by applying both of these

Table 4: Features selection global method application results

Initial features set	Attack prevention (theory)	Accuracy	Nb of features
21 default features (GD resistant selection only)	100%	98,03%	6
21 default features	100%	55,68%	2
Frequency (80%)	100%	67,64%	1
Frequency (90%)	100%	95,12%	3
Frequency (80%) + default features	100%	55,66%	1
Frequency (90%) + default features	100%	55,79%	3
Frequency (95%) + default features	100%	98,22%	3

techniques.

Results. We obtained in each case a 100% theoretical resistance of gradient-descent, but only the initial features sets including the default 21 features (and with or without the frequency-selected features) had good accuracy (more than 99%).

Practical results. Table 6 summarizes the best results obtained by applying each counter-measure. The conclusion we can make is that the best compromise between accuracy and attack prevention is when both the threshold and the gradient-descent resistant features selection are applied. In this case, we are able to prevent 99,99% of gradient-descent attacks while conserving a reasonably good accuracy of 99,22%.

4.3 Adversarial Learning

One of the major drawbacks concerning the supervised learning that we used so far is that the SVM is only trained once. The decision is then always the same and the classifier does not learn from its mistakes.

The Adversarial Learning solves this issue by allowing the expert to feed the SVM again with vectors it wrongly classified. Hence, the classifier will learn step-by-step the unexplored regions that are used by the attacks to bypass the SVM. Algorithm 3 describes how we iteratively fed our SVM to implement the adversarial learning. Note that we used the 21 default features from PDFiD.

The results of this counter-measure are presented in Table 7. With $n = 10$ we found that 3 rounds are enough for the SVM to completely stop the gradient-descent attack. Furthermore, at each round, we see that the gradient-descent algorithm requires more and more steps to finally converge, and thus the attack becomes more and more costly. The Support Vectors column represents the number of support vectors the SVM has constructed. The accuracy represents the number of correct classifications of the SVM. One interesting fact using this counter-measure is that the

---

Algorithm 3 Adversarial learning

$n \leftarrow$ number of PDF files to give to c at each iteration
$c \leftarrow$ trained SVM
$s_0 \leftarrow$ success rate of gradient-descent
$s_1 \leftarrow -1$
while $s_0 \neq s_1$ **do**
    $s_0 \leftarrow s_1$
    feed c with n gradient-descent-forged files
    relaunch the learning step of c
    $s_1 \leftarrow$ success rate of gradient-descent
end while
return c

---

accuracy of the SVM barely changes. Furthermore, we did not need a more elaborated choice of features.

5 CONCLUSION AND PERSPECTIVES

We implemented a naive SVM, that we easily tricked with a gradient-descent attack. We also implemented counter-measures against this attack: first, we set up a threshold over each considered feature. This alone enabled us to stop almost every gradient-descent attack. Then, we reduced the number of selected features, in order to remove features that were used during the gradient-descent attack. This makes the attack even less practical, at the cost of reducing the accuracy of the SVM. We also proposed another approach to reduce the chances of success of the gradient descent attack, using adversarial learning, by training the SVM with gradient-descent forged PDF files, and re-iterating the process. Our SVM was resistant to gradient-descent attacks after only three iterations of the process.

Obviously more can be said about the topic. For instance, it can be interesting to see what can happen if the adversary does not know the algorithms and fea-

Table 5: Features selection global method with threshold application results

Initial features set	Attack prevention (theory)	Accuracy	Nb of features
21 default features	100%	99,11%	6
Frequency (80%)	100%	94,36%	5
Frequency (90%)	100%	98,00%	30
Frequency (80%) + default features	100%	98,10%	8
Frequency (90%) + default features	100%	99,05%	6

Table 6: Results of Counter-measures Threshold and Features Selection application

	Attack prevention (in practice)	Accuracy	Nb of features
Treshold only	94,00%	99,81%	20
Features selection only	99,97%	98,05%	2 (/JS and /XFA)
Threshold + Features selection	99,99%	99,22%	9

Table 7: Adversarial learning results

Round	# Support Vectors	Accuracy (%)	Steps number of GD	Success rate of GD (%)
0	293	99,70	800	100
1	308	99,68	1 800	90
2	312	99,67	3 000	0

tures that have been used in the classifier. We could also perform a gradient-descent attack using other algorithms (e.g. Naive Bayes, Decision Tree, Random Forest) and see how many files thus forged can bypass our SVM. The adversary could also use other types of attacks, like Monte-Carlo Markov Chains (MCMC) techniques. Other attacks may exploit some properties that are inherent in the training set. To avoid them, it may be interesting to have a look at unsupervised learning techniques, and try to identify a malicious behaviour with clustering. We could also use deep learning algorithms like Generative Adversarial Network (GAN), in order to generate a classifier and test its resistance against various attacks.

ACKNOWLEDGEMENTS

REFERENCES

Aizerman, M. A., Braverman, E. A., and Rozonoer, L. (1964). Theoretical foundations of the potential function method in pattern recognition learning. Automation and Remote Control.

Ateniese, G., Felici, G., Mancini, L. V., Spognardi, A., Villani, A., and Vitali, D. (2013). Hacking smart machines with smarter ones: How to extract meaningful data from machine learning classifiers. *CoRR*.

Biggio, B., Corona, I., Maiorca, D., Nelson, B., Šrndić, N., Laskov, P., Giacinto, G., and Roli, F. (2013). *Evasion Attacks against Machine Learning at Test Time*.

Borg, K. (2013). Real time detection and analysis of pdf-files. Master's thesis.

Boser, B. E., Guyon, I. M., and Vapnik, V. N. (1992). A training algorithm for optimal margin classifiers. COLT '92.

Contagio Dump (2013). Contagio: Malware dump. http://contagiodump.blogspot.fr/2013/03/16800-clean-and-11960-malicious-files.html.

CVEDetails (2017). Adobe vulnerabilities statistics. https://www.cvedetails.com/product/497/Adobe-Acrobat-Reader.html.

Kittilsen, J. (2011). Detecting malicious pdf documents. Master's thesis.

Maiorca, D., Giacinto, G., and Corona, I. (2012). *A Pattern Recognition System for Malicious PDF Files Detection,*.

Stevens, D. (2006). Didier stevens blog. https://blog.didierstevens.com/.

Torres, J. and De Los Santos, J. (2018). Malicious pdf documents detection using machine learning techniques.