Malware Detection in PDF Files and Evasion Attacks

Bonan Cuan[1] Aliénor Damien[2,3] Claire Delaplace[4,5] Mathieu Valois[6]

Under supervision of

Olivier Bettan[2] Boussad Addad[2] Marius Lombard-Platet[2]

[1]LIRIS, [2]Thales Group, [3]LAAS, [4]IRISA, [5]CRIStAL, [6]GREYC

REDOCS 2017

Context

- A PDF file can contain
  - JavaScript Code
  - Flash objects
  - Binary Programs
  - ...
- All PDF readers have weaknesses
- Many attack vectors used by malwares

Context

- A PDF file can contain
  - JavaScript Code
  - Flash objects
  - Binary Programs
  - ...
- All PDF readers have weaknesses
- Many attack vectors used by malwares

Our Work

- Use machine learning to detect infected PDF
- Modify infected PDF to lure the classifier
- Find efficient counter-measures to this attack

1. Malware Detection using Machine Learning
2. Evasion Attacks
3. Counter-Measures

PDF Structure

In a Nutshell

- PDF: set of objects identified by tags (features)
- Several tools for PDF analysis (e.g. PDFiD)
- 21 features are frequently used by malwares
  - *based on Didier Stevens security expert's work: https://blog.didierstevens.com/programs/pdf-tools/*

Supervised Learning

Definition

- Inferring a function from labeled training data

In our case

Dataset:

- 10 000 clean PDF
- 10 000 PDF with Malware (Contagio)

Feature vector = [Tag1 occ., Tag2 occ., ...]

For a given PDF

Function: $class(X) = y$

- $X \in \mathbb{Z}^n$: feature vector
- y: label

$$\begin{array}{rl} 1 & \text{if the PDF is clean} \\ -1 & \text{if the PDF contains a malware} \end{array}$$

Example

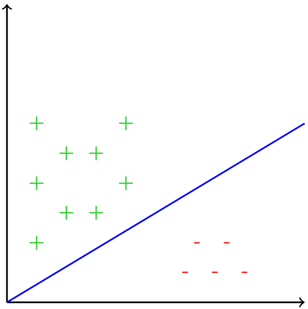
```
PDFiD 0.2.1 CLEAN_PDF_9000_files/rr-07-58.pdf
 PDF Header: %PDF-1.4
 obj                     23
 endobj                  23
 stream                   6
 endstream                6
 xref                     2
 trailer                  2
 startxref                2
 /Page                    4
 /Encrypt                 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
```


```
{'name': 'CLEAN_PDF_9000_files/rr-07-58.pdf',
 'label': 1,
 'features': array([23, 23, 6, 6, 2, 2, 2, 4, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0])}
```


$$f(23, 23, \ldots, 0) = 1$$

SVM (Support Vector Machine)

- One scatterplots per label
- Find a hyperplan to delimit them



Training our SVM

- 60% of our data set used for training
- 40% used for testing

Description

- Get the feature vectors and labels for the training dataset
- Normalize independently each feature
- Create the SVM (use scikit-learn python module)
- Test with the remaining PDF

First Results

- Accuracy: 99.62 %
- Malwares detected as clean: 0,34% (28/8087)
- Clean detected as malware: 0,03% (3/8087)

Model Improvements

Change the Training and Testing Sets

- Modify the splitting ratio
  - 80%/20% $\rightarrow$ better accuracy
- Use X-validation

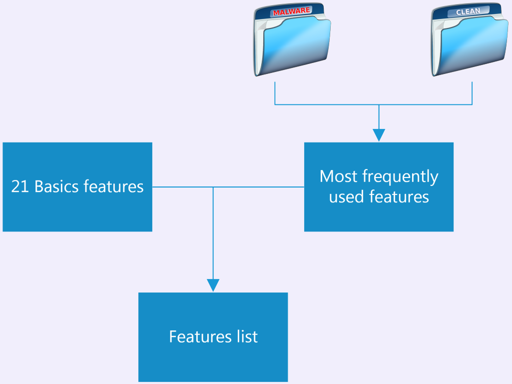
Change the Chosen Features

- Select discriminating feature with respect to our dataset

Features Selection (Frequency)

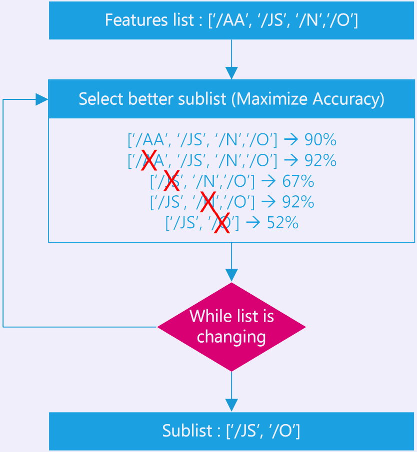
Use every features
⇒ Too many features (computing break)

1st Method : Frequency Selection



Features Selection (Sublist)

2nd Method : Select Best Sublist



Results

Features selection comparison

Features selection	Accuracy (x-validation)	Nb of features
No features selection (21 basics features)	99,48%	21
Sublist from 21 basis features	99,68%	12
Frequency + Sublist from all features	99,59%	18

Other results

- Apparently no overfitting issue

1. Malware Detection using Machine Learning
2. Evasion Attacks
3. Counter-Measures

Adversary Model

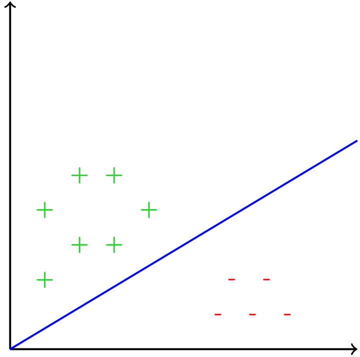
White Box Adversary

- The training dataset
- The used classification algorithm
- PDF files with malware that are detected by the SVM

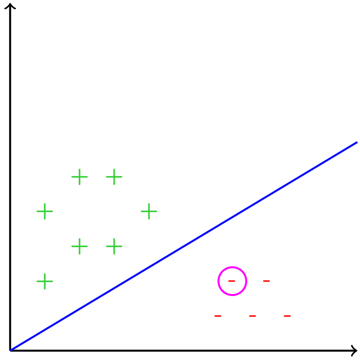
Goal

Append objects to the PDF to evade the detection

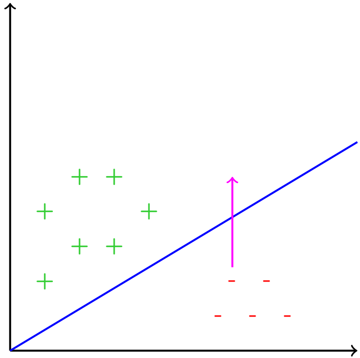
Naive Attack: Increase the Value of One Component



Naive Attack: Increase the Value of One Component

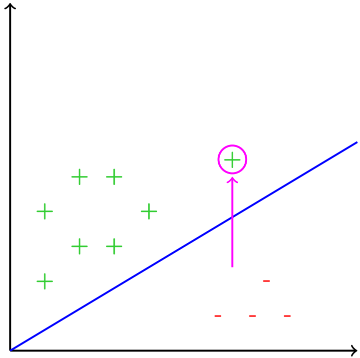


Naive Attack: Increase the Value of One Component



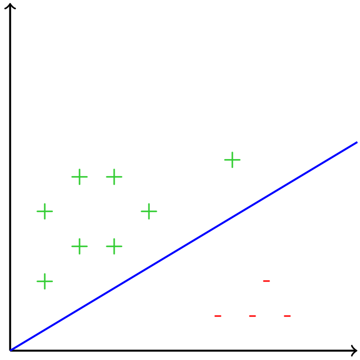
- Increase a well chosen component to cross the border

Naive Attack: Increase the Value of One Component



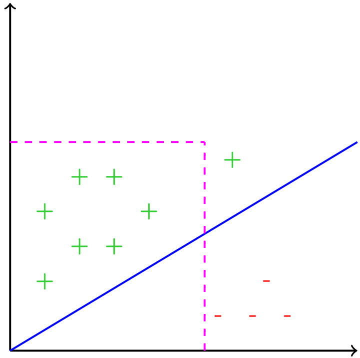
- Increase a well chosen component to cross the border

Naive Attack: Increase the Value of One Component



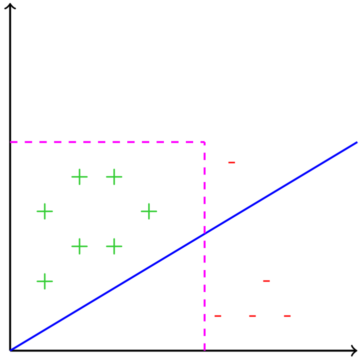
- Increase a well chosen component to cross the border
- Add a lot of “non suspicious” objects (e.g. 50)

Naive Attack: Increase the Value of One Component



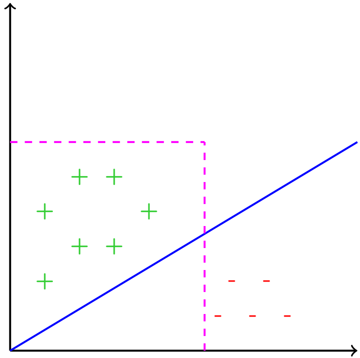
- Increase a well chosen component to cross the border
- Add a lot of "non suspicious" objects (e.g. 50)
- Easy counterattack: Add a threshold to the SVM

Naive Attack: Increase the Value of One Component



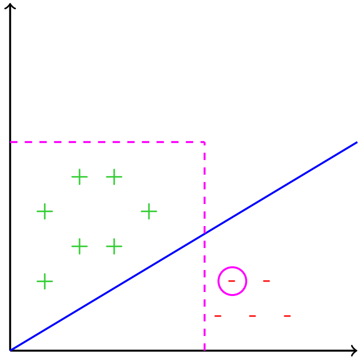
- Increase a well chosen component to cross the border
- Add a lot of “non suspicious” objects (e.g. 50)
- Easy counterattack: Add a threshold to the SVM

Second Attack: Gradient Descent



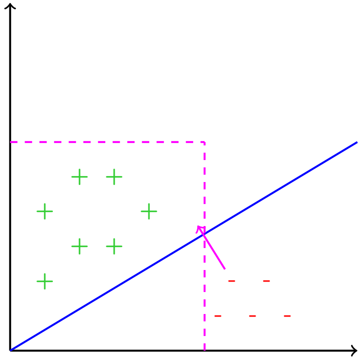
- Step by step approach (iterations)
- More components are modified
- Less objects added on the whole

Second Attack: Gradient Descent



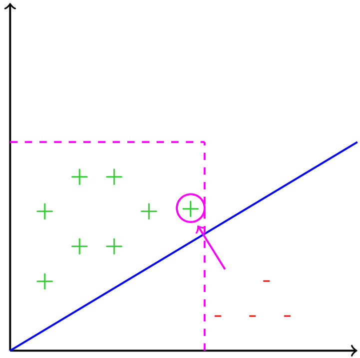
- Step by step approach (iterations)
- More components are modified
- Less objects added on the whole

Second Attack: Gradient Descent



- Step by step approach (iterations)
- More components are modified
- Less objects added on the whole

Second Attack: Gradient Descent



- Step by step approach (iterations)
- More components are modified
- Less objects added on the whole

Test and Result of the Attack

Theoretical Attack

- 100% of the modified malware vectors detected as clean
- Gradient descent computes float vectors

Test and Result of the Attack

Theoretical Attack

- 100% of the modified malware vectors detected as clean
- Gradient descent computes float vectors

In Practice

- Forge new PDF files from gradient-descent-computed vectors
- Rounding is required $\Rightarrow$ precision issues
- 97.5% of the newly forged PDF were detected as clean

Vector Component Threshold

Threshold Computation

Threshold $\in \mathbb{N}^*$ because PDF objects number is discrete

1. Arbitrarily choose a threshold
2. Apply this threshold on each vector component independently
3. Check success rate of gradient descent
4. If success rate not low enough reduce threshold and go to 2)

Vector Component Threshold

Threshold Computation

Threshold $\in \mathbb{N}^*$ because PDF objects number is discrete

1. Arbitrarily choose a threshold
2. Apply this threshold on each vector component independently
3. Check success rate of gradient descent
4. If success rate not low enough reduce threshold and go to 2)

Results

- 5 $\rightarrow$ reduce attacks by 35%
- 4 $\rightarrow$ reduce attacks by 36%
- 3 $\rightarrow$ reduce attacks by 38%
- 2 $\rightarrow$ reduce attacks by 40%
- 1 $\rightarrow$ reduce attacks by 94%

Vector Component Threshold

Threshold Computation

Threshold $\in \mathbb{N}^*$ because PDF objects number is discrete

1. Arbitrarily choose a threshold
2. Apply this threshold on each vector component independently
3. Check success rate of gradient descent
4. If success rate not low enough reduce threshold and go to 2)

Results

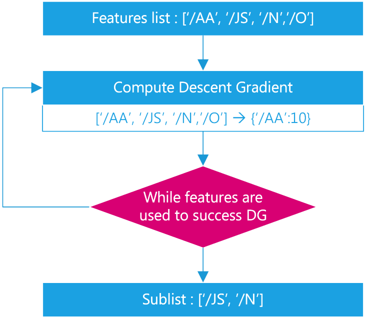
- 5 $\rightarrow$ reduce attacks by 35%
- 4 $\rightarrow$ reduce attacks by 36%
- 3 $\rightarrow$ reduce attacks by 38%
- 2 $\rightarrow$ reduce attacks by 40%
- 1 $\rightarrow$ reduce attacks by 94%

$\Rightarrow$ Cannot perform better only with threshold

Features Selection (Remove GD)

Removing Features

- Gradient descent: only some features used
- Idea: remove features used by GD
- Work with various initial choices of features (not only the 21 from PDFiD)



Features Selection (Remove GD)

Results

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

Adversarial Learning

Principle

Supervised learning:

- Feed SVM by labeling gradient-descent-forged PDFs
- Relaunch the learning step
- Rounds until attack reduction is stable
- No need of feature selection

Adversarial Learning

Principle

Supervised learning:

- Feed SVM by labeling gradient-descent-forged PDFs
- Relaunch the learning step
- Rounds until attack reduction is stable
- No need of feature selection

Results

- labeled forged PDF between each rounds
- Iterations of GD = hardness of the attack

Round	SV	Accuracy (%)	Iterations of GD	Success rate of GD (%)
0	293	99,70	800	100
1	308	99,68	1800	90
2	312	99,67	3000	0

$\Rightarrow$ 3 iterations is enough for SVM to be fully resistant to GD attacks

Conclusion and Perspectives

Conclusion

- Naive SVM: easy to trick with gradient descent
- Usage of threshold: stops almost every GD attack
- Optimal features computation reduces even more the attack surface
- But reduce a bit the accuracy of the SVM

Perspectives

- Change adversary model:
  - Attacker has no knowledge of used classifier
  - Attacker uses another classifier for gradient descent
- Use deep learning like GAN (Generative Adversarial Network)
- Attack classifier with Monte-Carlo Markov Chains (MCMC) techniques

Thank you for your time !
Questions?

bonan.cuan@liris.cnrs.fr
claire.delaplace@irisa.fr
alienor.damien@laas.fr
mathieu.valois@unicaen.fr