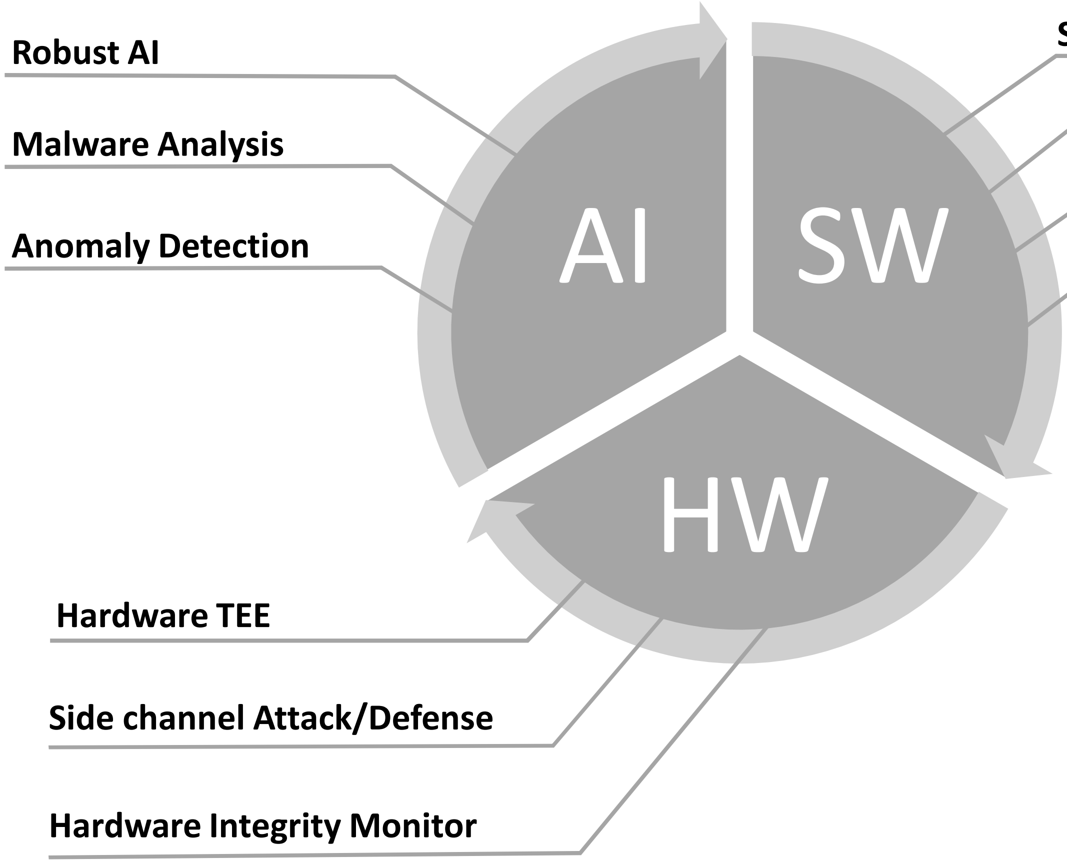
Generative 모델을 활용한 멀웨어 탐지 블랙박스 모델의 취약성 분석

백윤흥
SNU Security Research Group
서울대학교 전기.정보공학부

Topics

- PDF malware
- PDF classifiers
- White/black-box models for classifiers
- Automatic generation of evasive PDF malware
- Our approach using a generative model

SNU Security Research Group

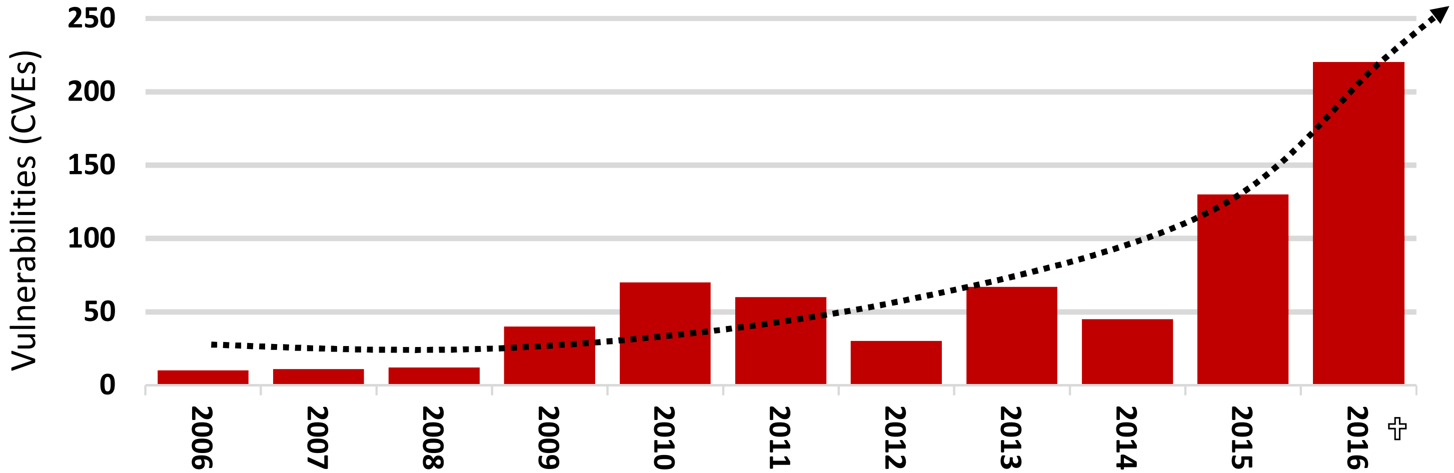


Principal Investigators: 2
Graduate Students: 26

http://sor.snu.ac.kr

PDF malware

- **PDF document can be malicious !**
- **# of detected PDF-based attacks is drastically increasing***
  - **In 2018, >47K *new* PDF-based attacks were discovered**
  - **In 2019, >73K PDF-based attacks were reported in one month, and PDF malware accounts for 17% of newly detected threats**



- **PDF malware is popular as PDF documents can be viewed on any device and are easy to create**

* https://www.sonicwall.com/resources/2020-cyber-threat-report-pdf

✞ http://www.cvedetails.com/vulnerability-list.php?vendor_id=53&product_id=921

PDF malware example

- **PDF consists of multiple objects which are hierarchically connected with each other.**
- **Adversaries can inject their own JavaScript code into the PDF document structure**
- **JavaScript code exploits specific PDF reader's vulnerability to perform malicious actions**

```
%PDF-1.3
1 0 obj
<</Pages 1 0 R /OpenAction 2 0 R>>
2 0 obj
<</S /JavaScript /JS (

var heap_ptr   = 0;
var foxit_base = 0;
var pwn_array  = [];

function prepare_heap(size){
    var arr = new Array(size);
    for(var i = 0; i < size; i++){
        arr[i] = this.addAnnot({type: "Text"});;
        if (typeof arr[i] == "object"){
            arr[i].destroy();
        }
    }
}
function gc() {
    const maxMallocBytes = 128 * 0x100000;
    for (var i = 0; i < 3; i++) {
        var x = new ArrayBuffer(maxMallocBytes);
    }
}
```

Injected Javascript code example

JavaScript encoding

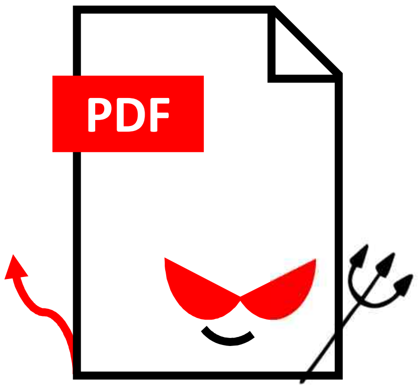
- **First, adversaries encode malicious JavaScript**

```
%PDF-1.3
2 0 obj
<<
/OpenAction << /JS 9 0 R /S /JavaScript >>
/Type /Catalog
/Pages 3 0 R
>>
endobj

9 0 obj
<</Filter /FlateDecode
/Length 2691
>>
stream
x\234\355\_s\3338^N^?\317\247\360\365\341&\2311\33
'V^]5+w\325H\211\345\353\335LVvd[\262\235\313\337\
|^?q\335\212\24\
7\337G\243\321\351\355u\353\240\365\242wx\324\267>
'\323\313h6_\304\311r\265\276\372\317\365\315\355\
\274\337}}\360\342\303\326\345\335*\274\235\257W\2
\337\335\222^F*\217\220c[8\273\336\377\211\376o\26
\311^E\362\277\315\376w\330\377\256\270<
endstream
endobj
```

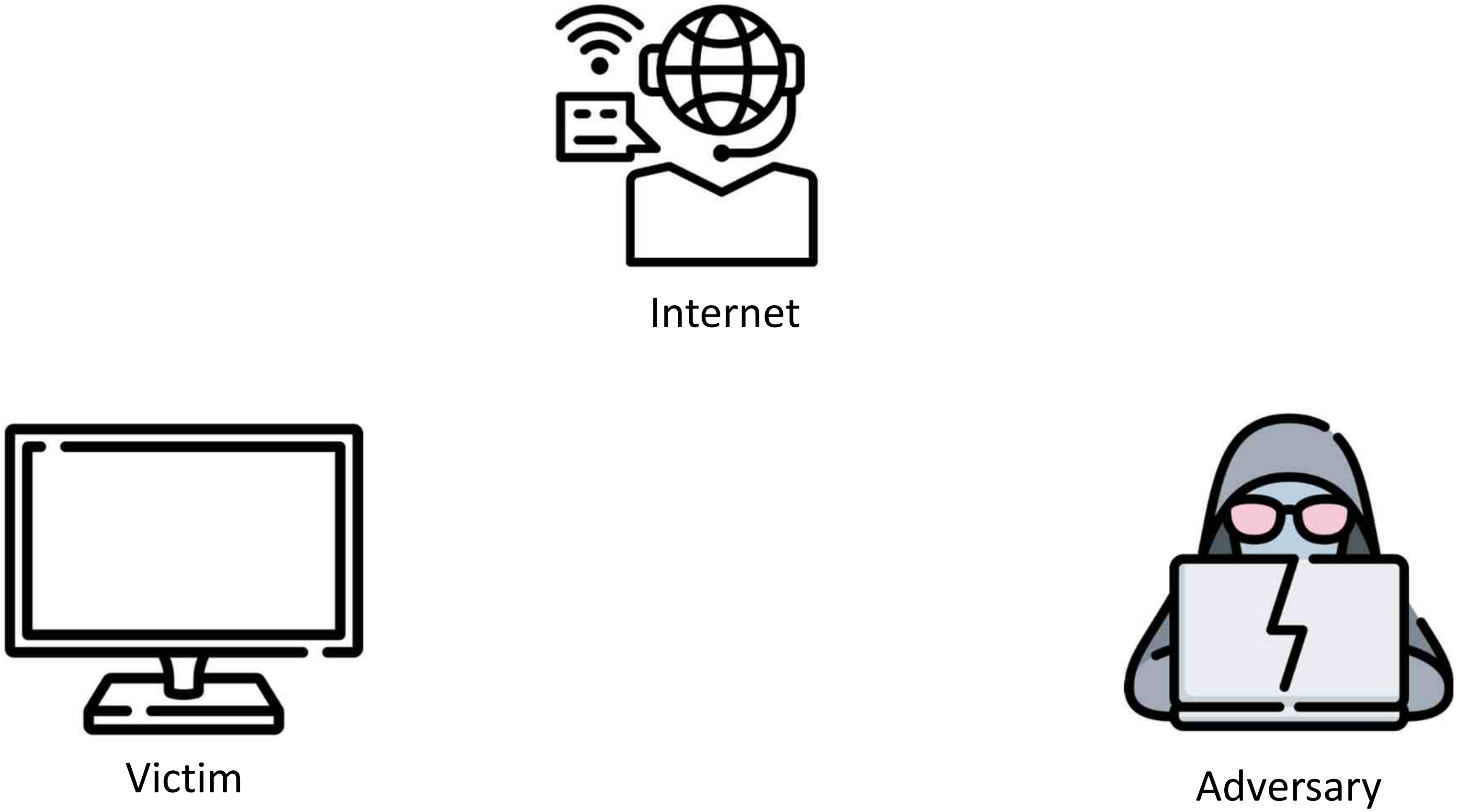
JavaScript injection

- **Then, they inject encoded malicious JavaScript code into PDF structure**



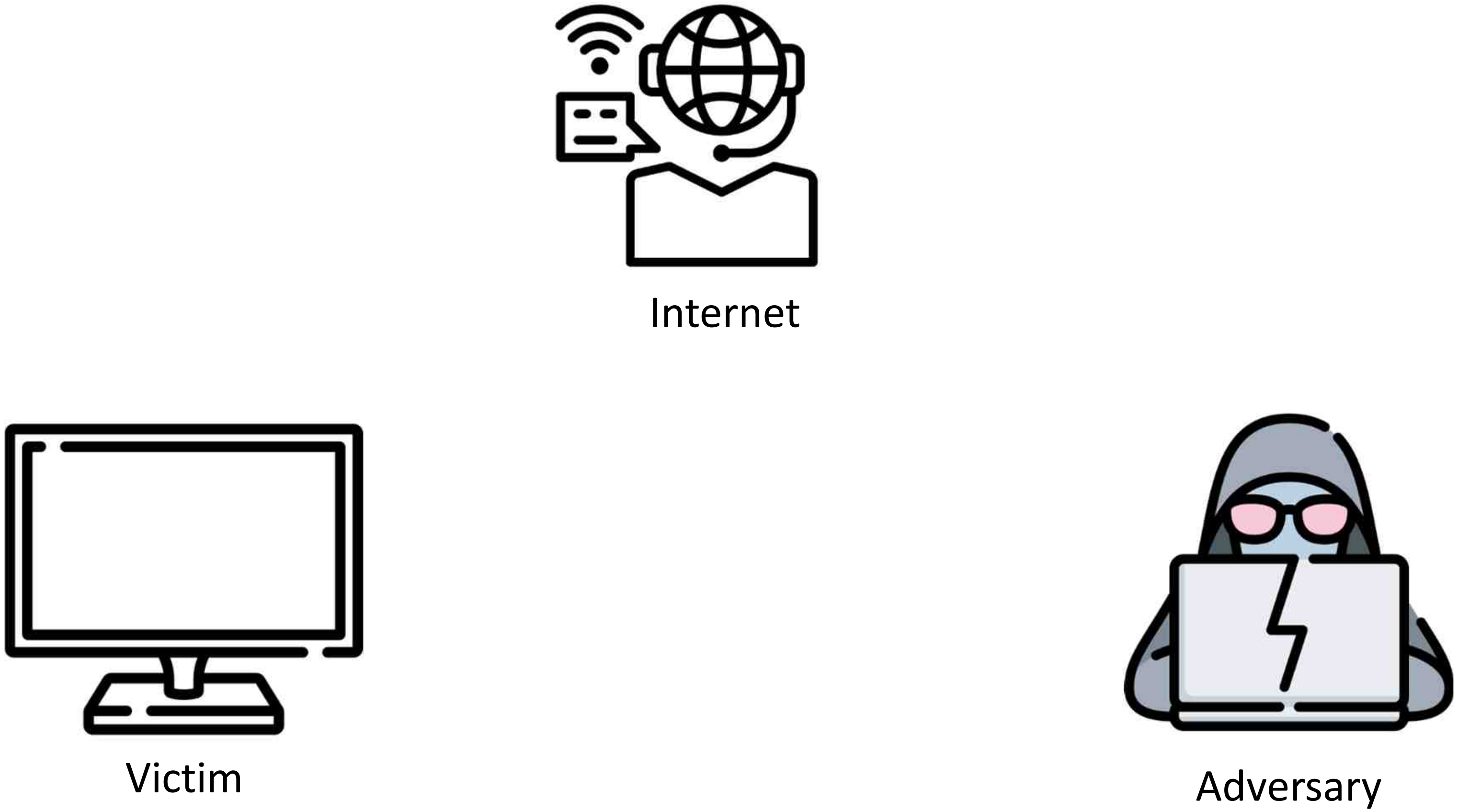
PDF malware circulation

- **Adversaries spread their malicious PDF documents**



PDF malware download

- **Victim downloads the malicious PDF document**

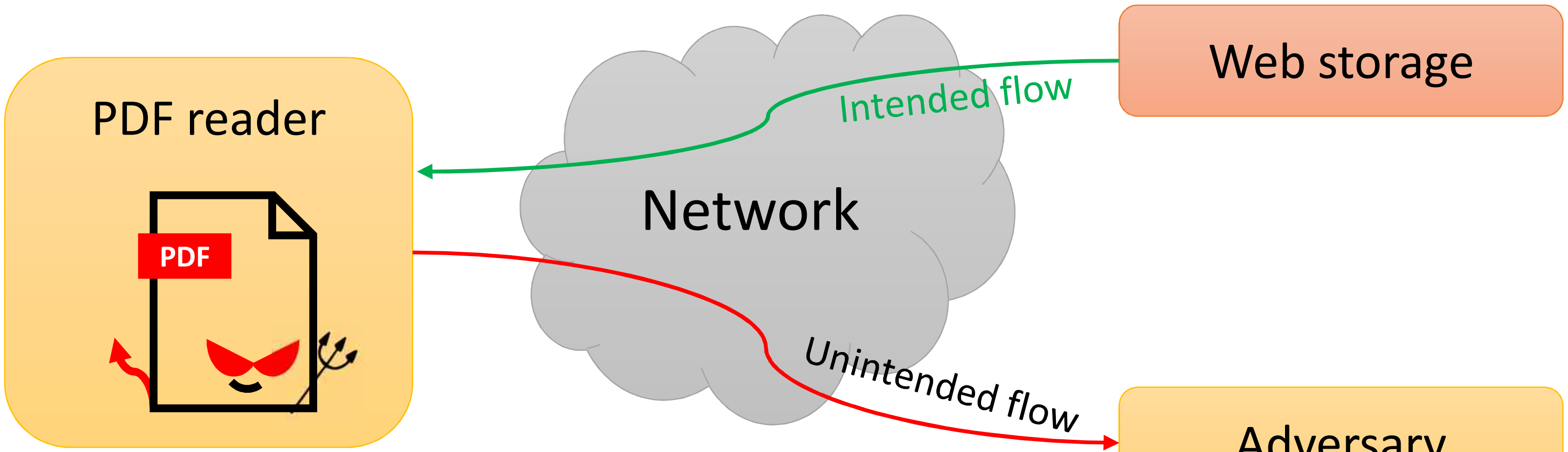


Malware infection

- **When victim opens the malicious PDF document, the system is infected.**
- **PDF reader application may become malicious**

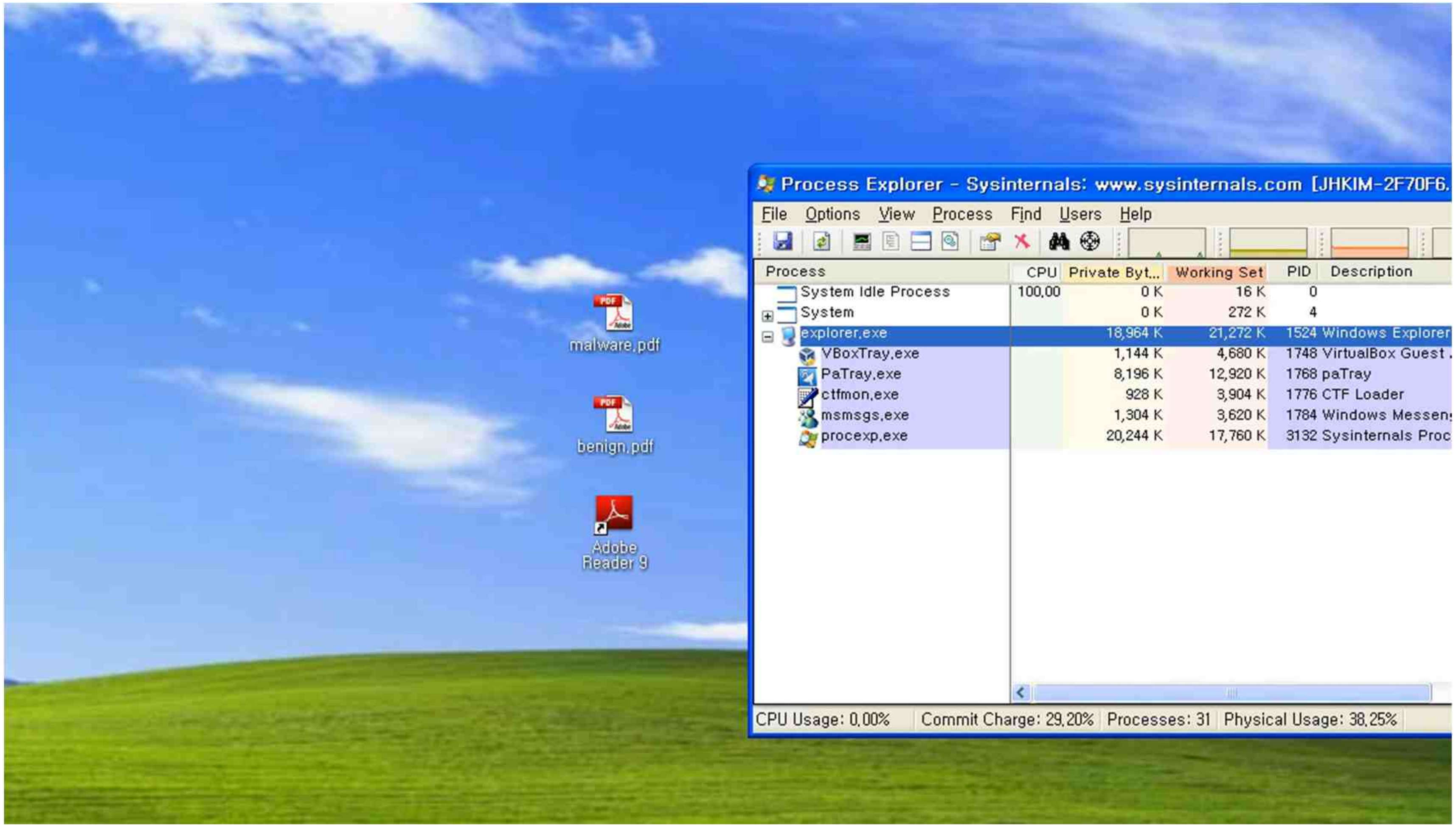
Once infected…

- **Private information may be unintentionally leaked to adversaries**
- **Infected PDF reader application …**
  - **may send your documents in web storage everywhere**
  - **Have access to your web storages to download from them.**
  - **Have permission to send data over the network.**



Once infected…

- **Control may be hijacked to open malicious payload**



PDF malware defense

- **PDF malware classifiers**
- **Rule-based classifiers are easily bypassed**
- **ML technology has been applied to tackle the rapidly increasing zero-day PDF malware**

Content-based Classifier

Metadata of PDF files

PDFrate (ASASC ’12)

Structure-based Classifier

Logical structure of PDF files

Hidost (NDSS ’13, JIS ‘16)

Content-based classifier

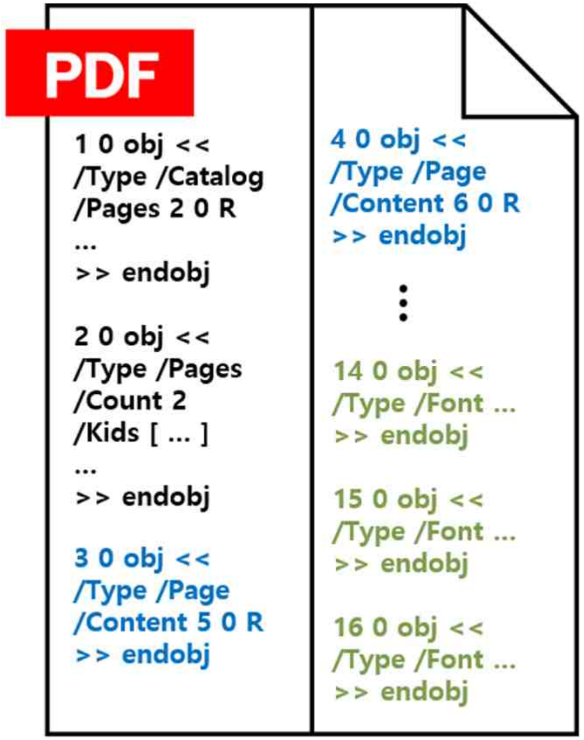
- **Based on features extracted from file document metadata**
- **A classifier, PDFrate, extracts 202 features manually selected**

count_font	title_len
count_javascript	creator_len
count_page	producer_len
count_endobj	createdate_tz
count_stream	ratio_imagepx_size
count_obj	ref_min_id
pos_box_max	count_font_obs
pos_eof_avg	count_image_large
pos_ref_avg	count_image_med
producer_len	count_image_small
len_stream_min	count_image_total
	count_startxref

PDFrate example

- **For example, count of font objects, page objects, JavaScript objects…**
- **The count of font objects is 3, and the count of page objects is 2 in here**
- **No JavaScript object in this example**

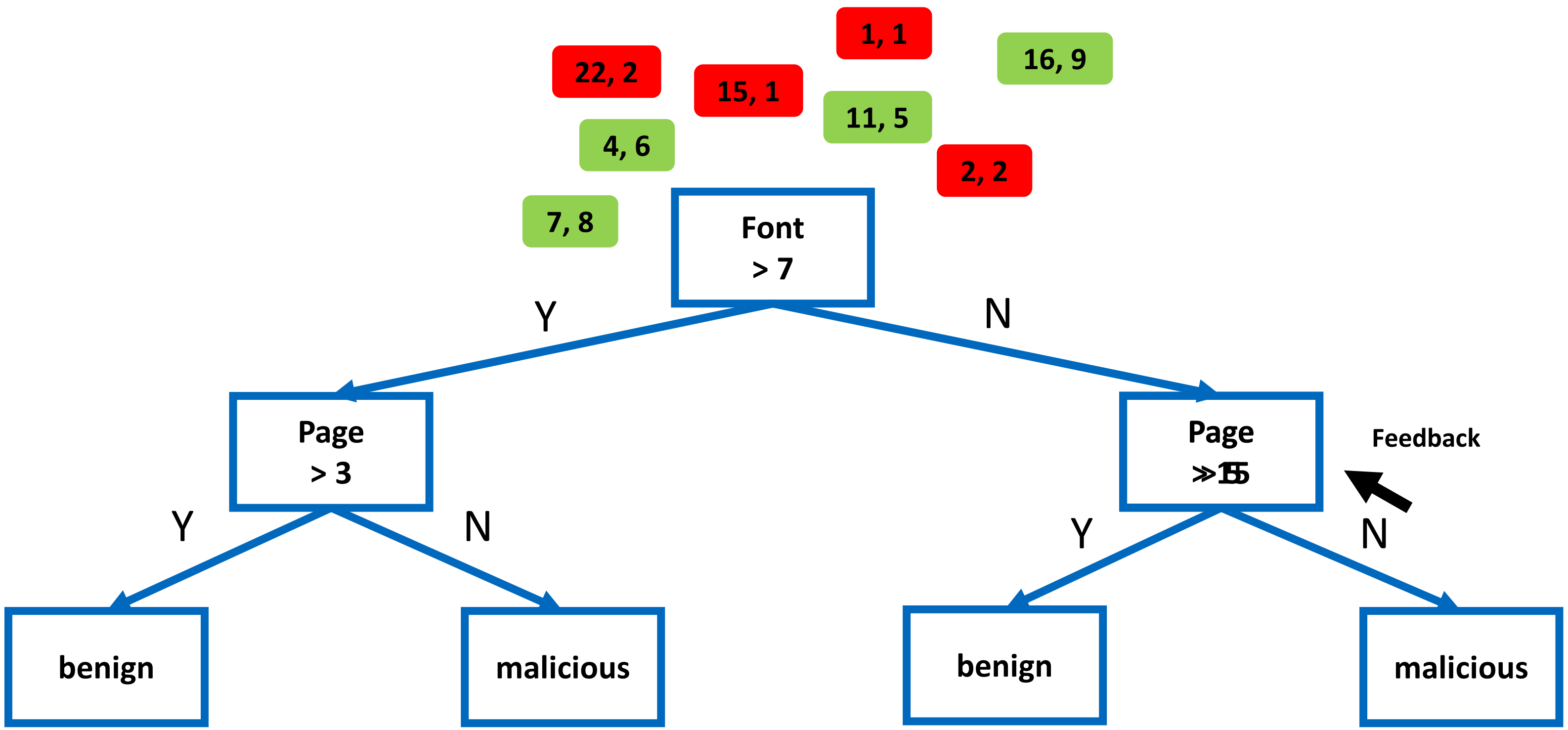
count_font
count_javascript
count_page
count_endobj
count_stream
count_obj
pos_box_max
pos_eof_avg
pos_ref_avg
producer_len
len_stream_min



Constructing a decision tree

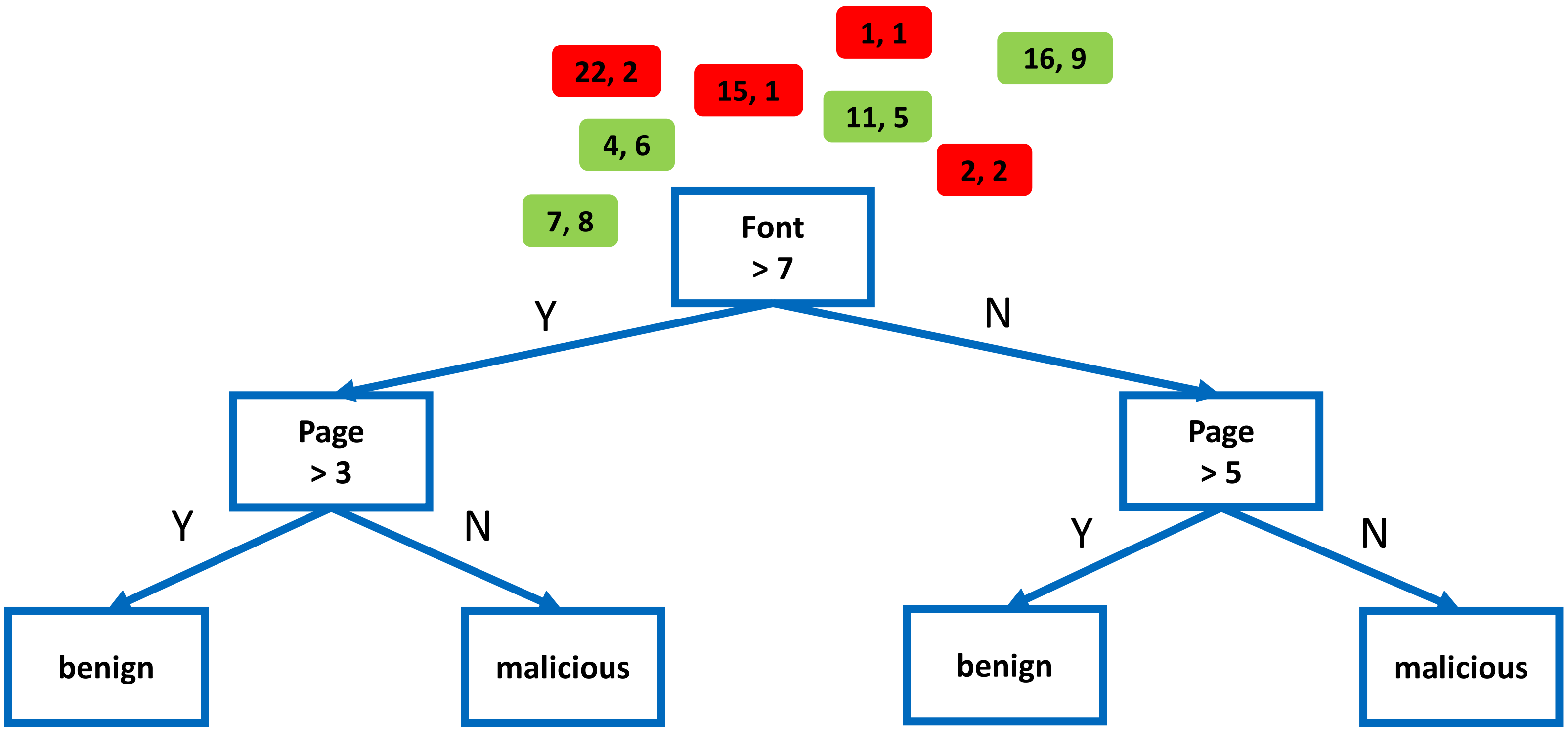
- **The data samples follow down the decision tree**
- **Choose feature boundary randomly**

Left : the count of “Font” objects
Right : the count of “Page” objects



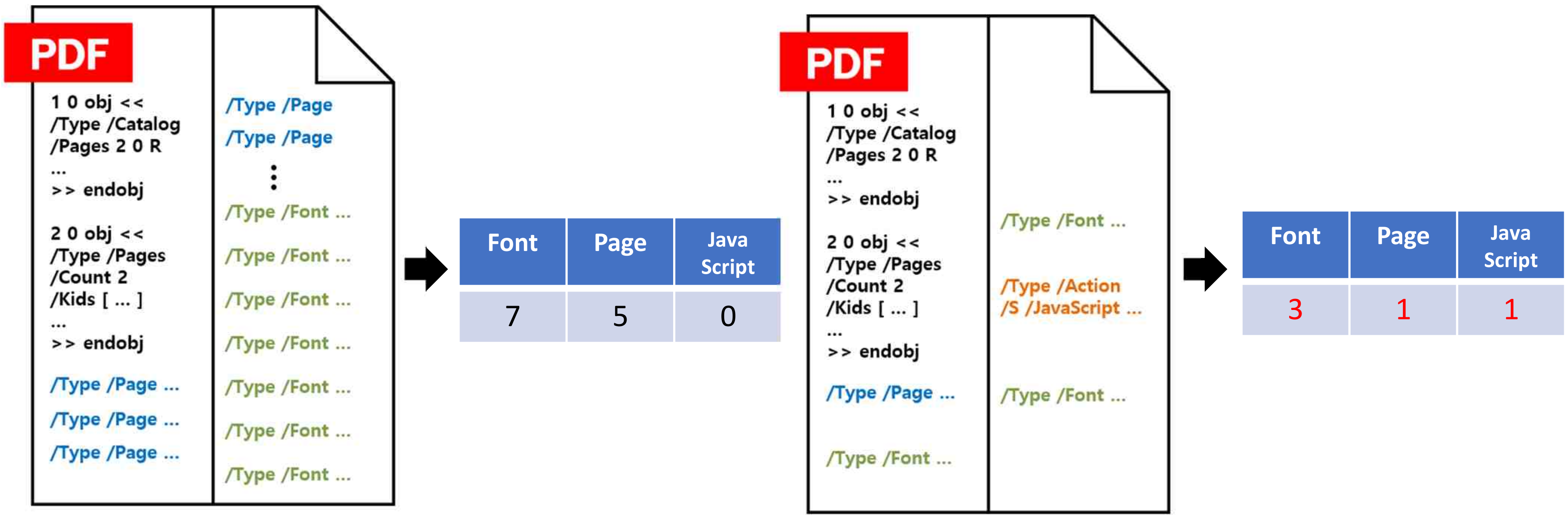
Making decision with the tree

- **After modifying decision boundary, all the test data is correctly classified**



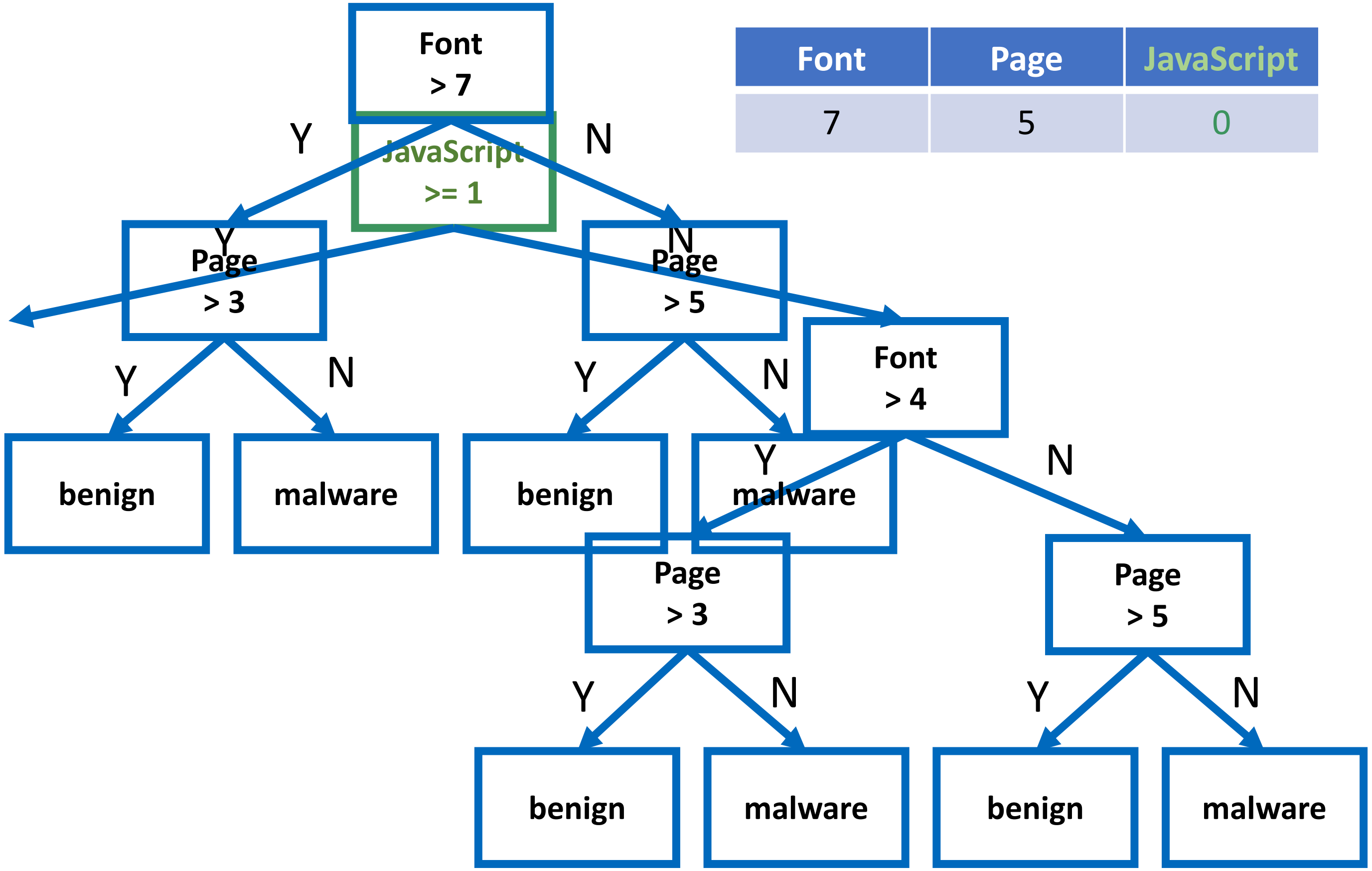
Malware defense with PDFrate

- **The font objects identify the font program and contain additional information about it**
- **A typical PDF malware has a smaller number of font objects than a typical benign PDF because most of PDF malwares do not have any contents .**



Benign PDF　　　　PDF malware

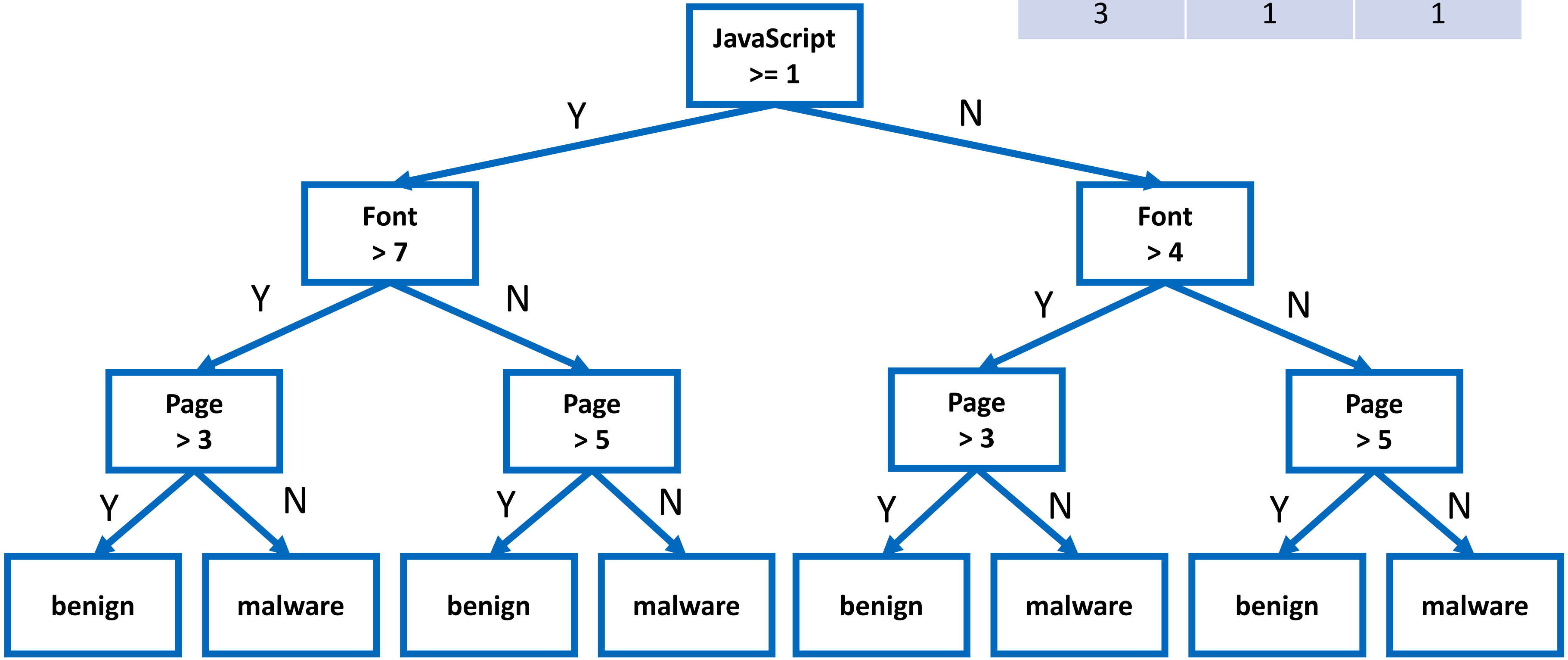
Decision tree for benign PDF



Font	Page	JavaScript
7	5	0

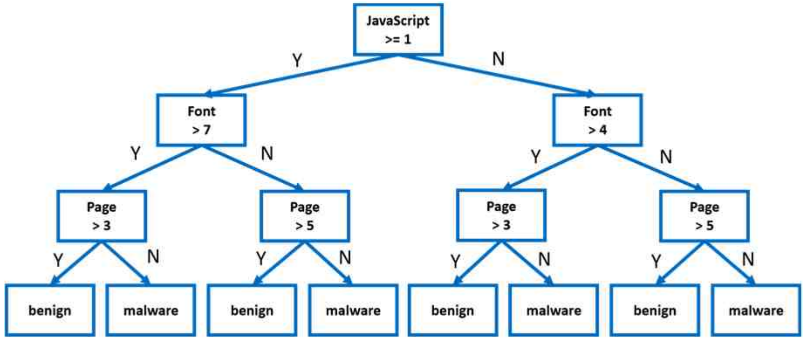
Decision tree for malicious PDF

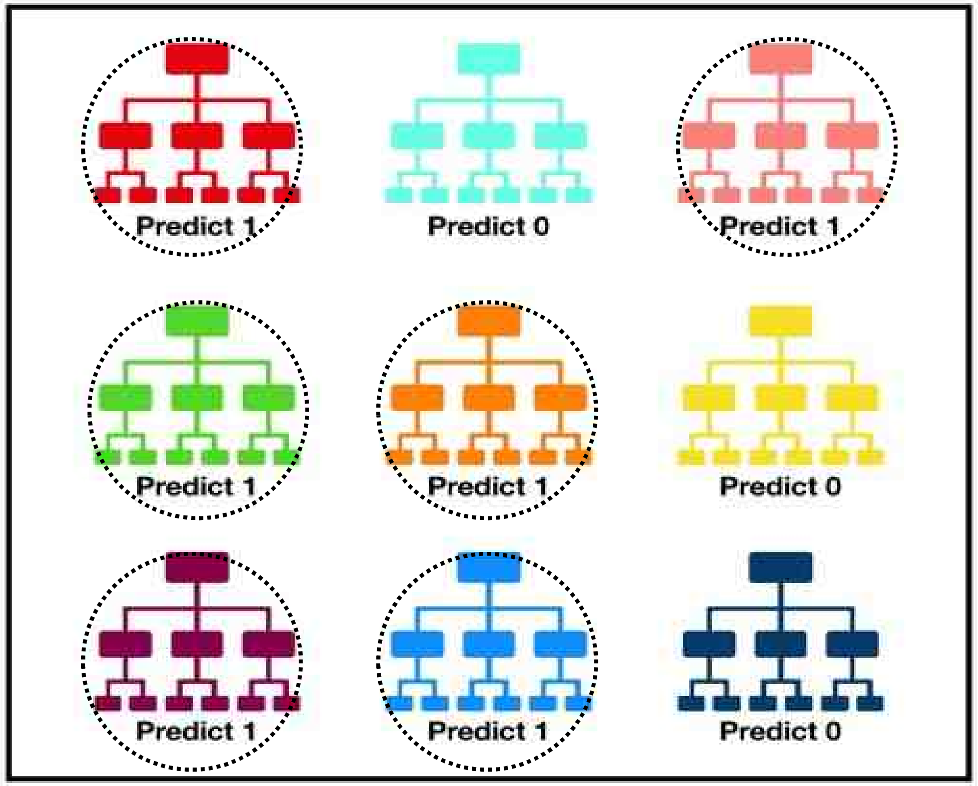
Font	Page	JavaScript
3	1	1



Building RF with decision trees

- **Random Forest (RF) is used by PDFrate for classifying benign/malicious PDFs**
- **RF, as its name ‘forest’ implies, consists of many random individual decision trees independently trained**
- **Through voting process among selected best trees make a final decision**

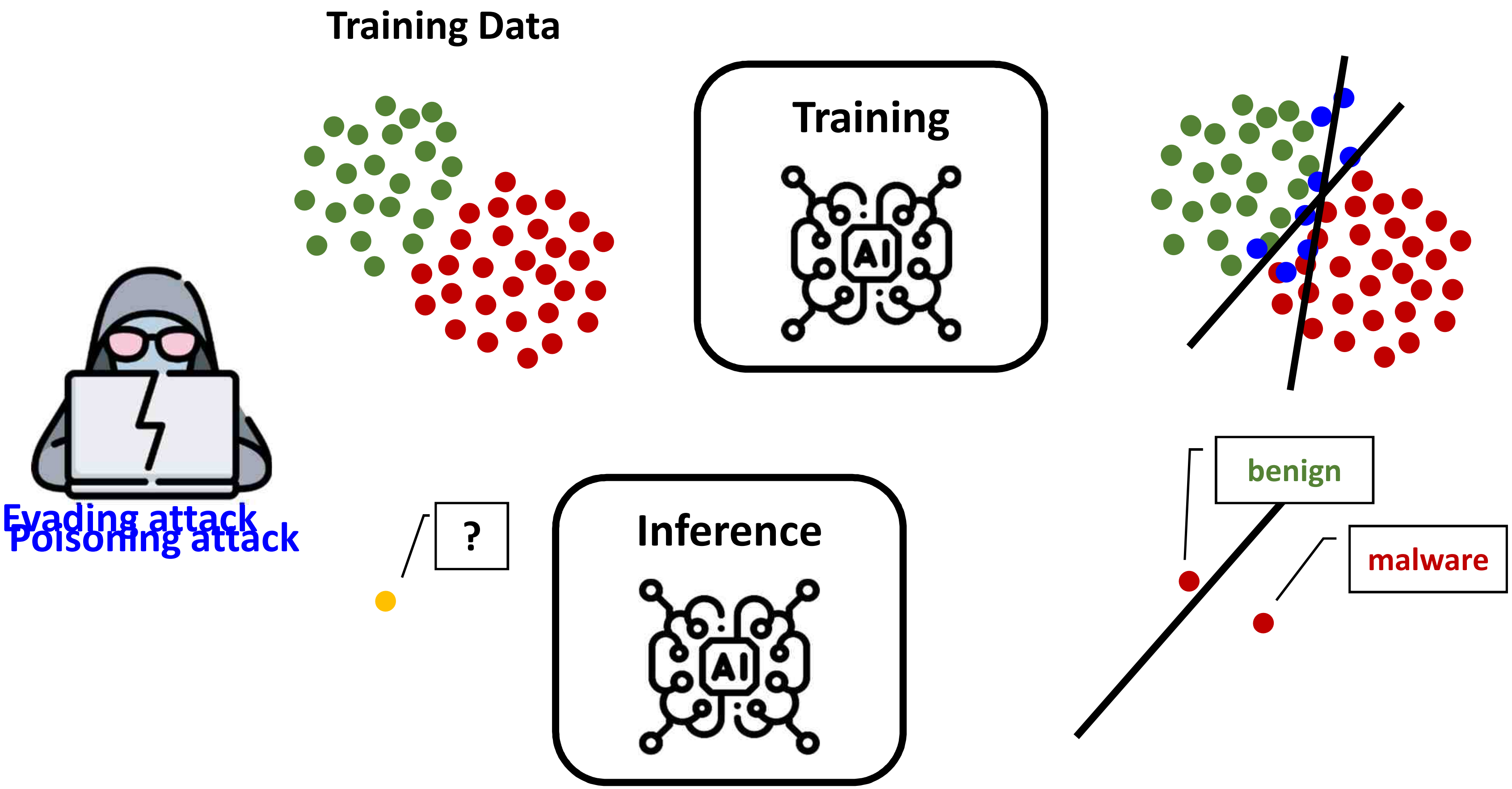




six 1s & three 0s ➔ **predict 1**

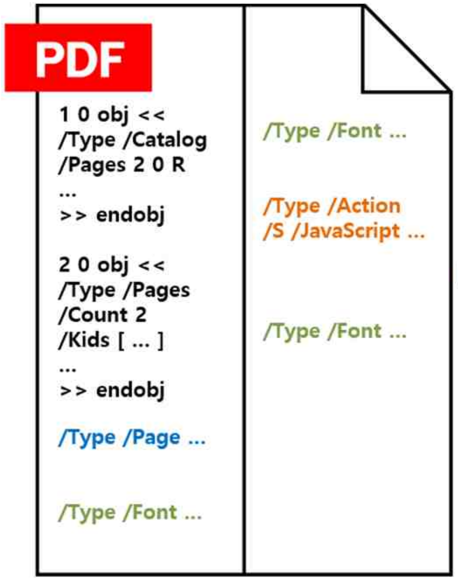
Machine learning does help!

- **PDFrate detection accuracy ➔ 0.997**
- **Unfortunately, the assumption that training data are reprehensive is often abused by adversaries**



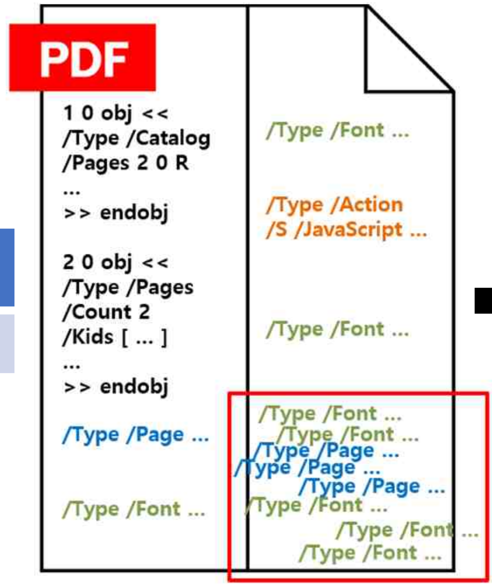
Evading PDFrate

- **The feature set is manually defined for good performance**
- **Also the feature values are determined by running a set of simple regular expressions on raw bytes of the PDF file**
- **Hence, vulnerable to a mimicry attack that crafts feature values.**



Font	Page	Java Script
3	1	1

PDF malware



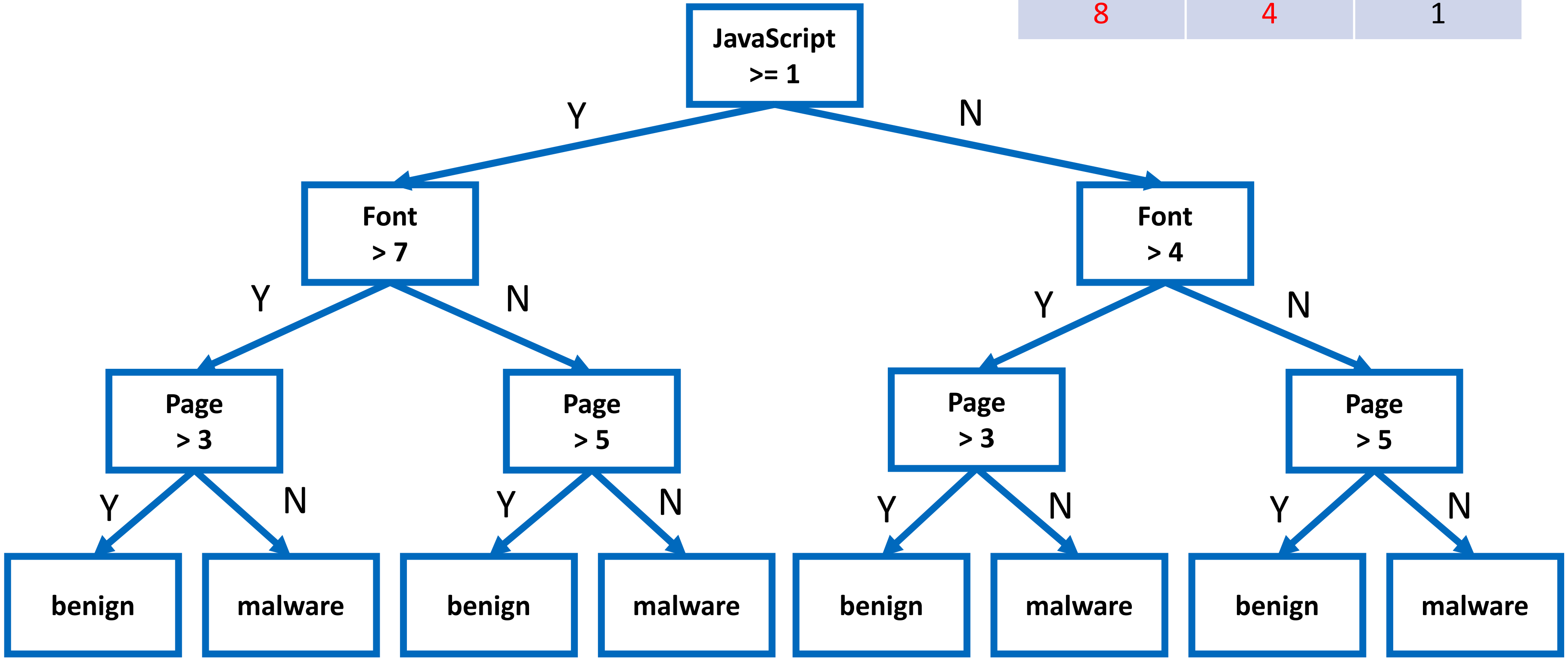
Font	Page	Java Script
8	4	1

PDF malware with mimicry attack

PDFrate under attack

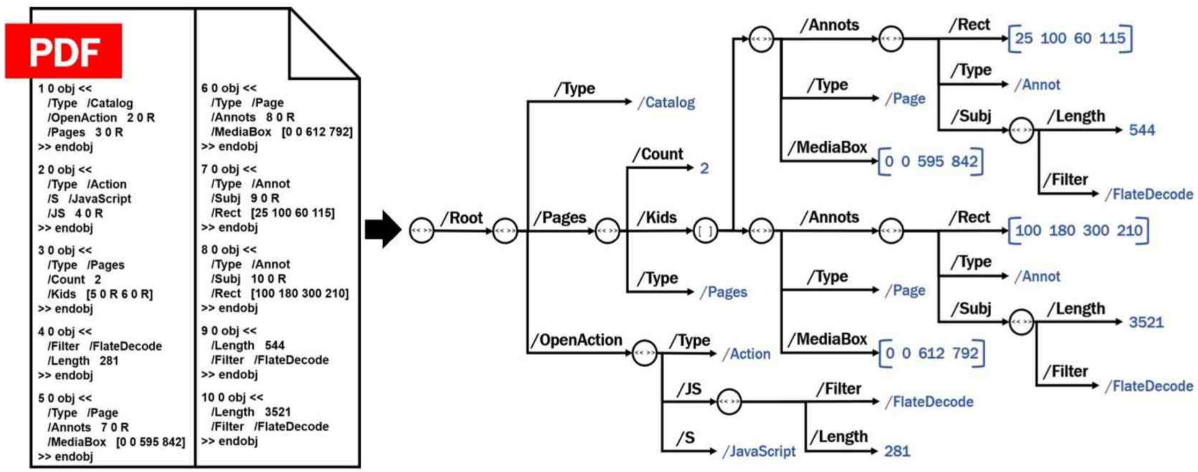
- **Decision tree of PDFrate for PDF malware evading with mimicry attack**

Font	Page	JavaScript
8	4	1



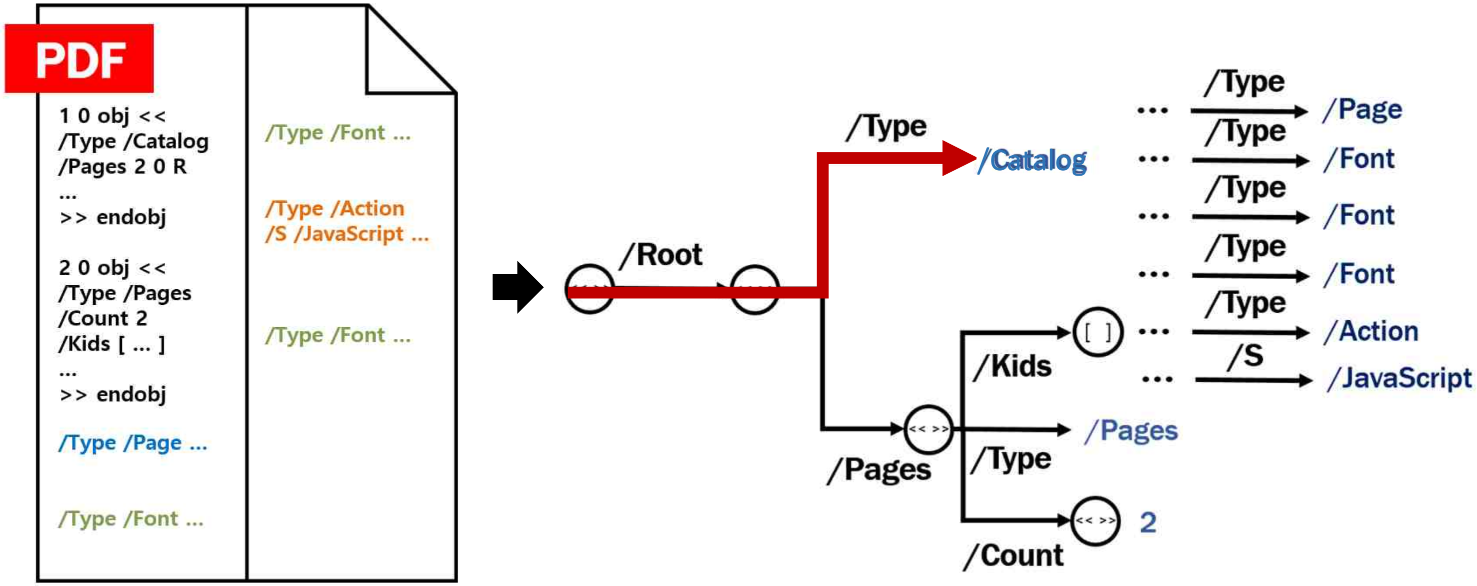
Structure-based classifier

- **A classifier, Hidost, discriminates between malicious and benign files based on the logical structure**
- **Not relying on a collection of individual features and their values, but on their relations in the PDF structures.**
- **Thus, relatively more robust against naïve mimicry attacks that only manipulate feature values ➔ accuracy: 0.999**
- **A total of 6,087 features are used**



Hidost – Feature

- **Parse PDF into a structural representation**
- **The feature set consists of paths from "/Root" to leaf nodes**

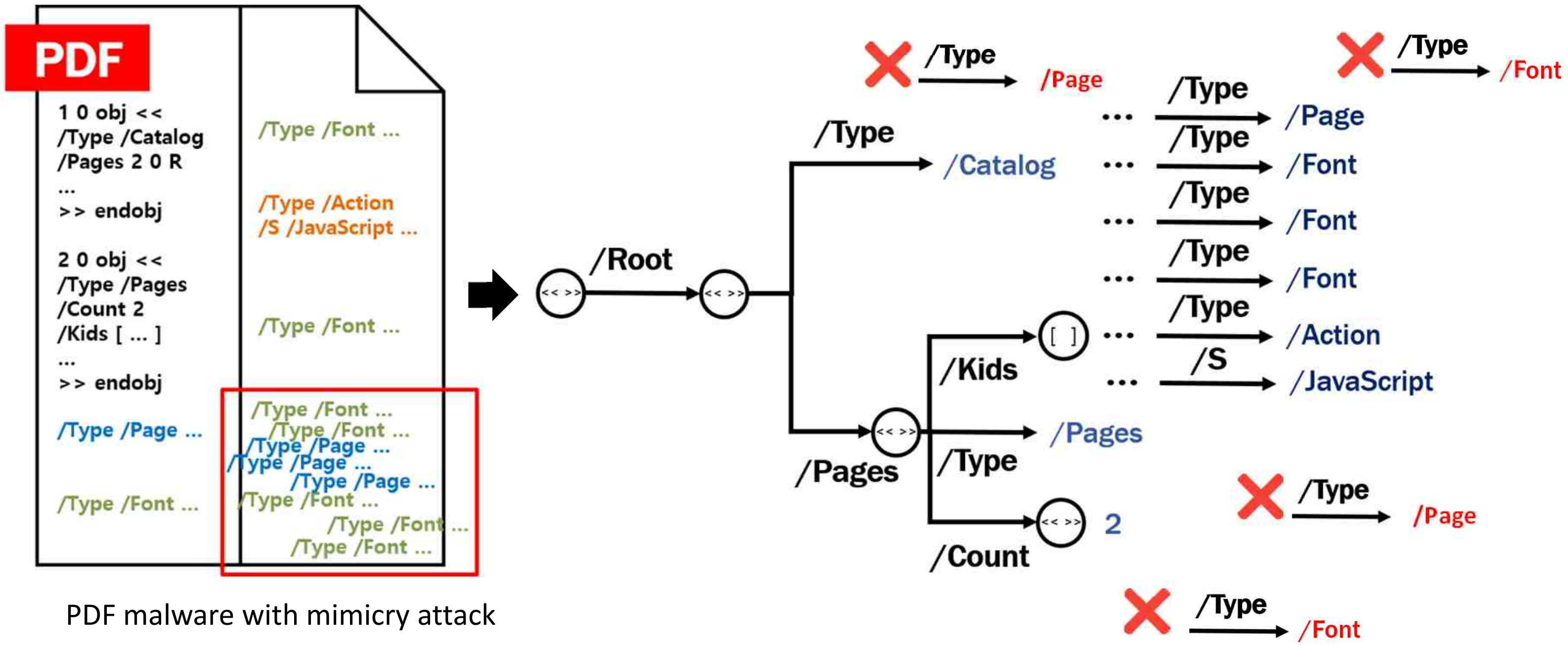


PDF malware

/Root /Type	/Root/Pages /Type	/Root/Pages /Count	/Root/Pages/... /Type	/Root/Pages/... /S	. . .
	/Pages	2	/Font	/JavaScript	. . .

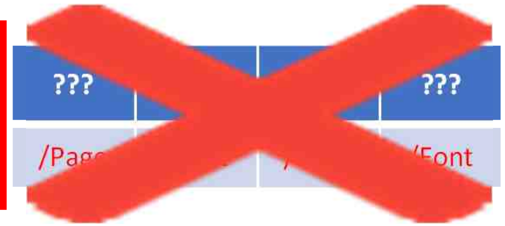
Mimicry defense

- **Mimicry attack that inserts objects of benign PDF into PDF malware without a sense of PDF structure**
- **Hidost will discard those objects in the feature set**

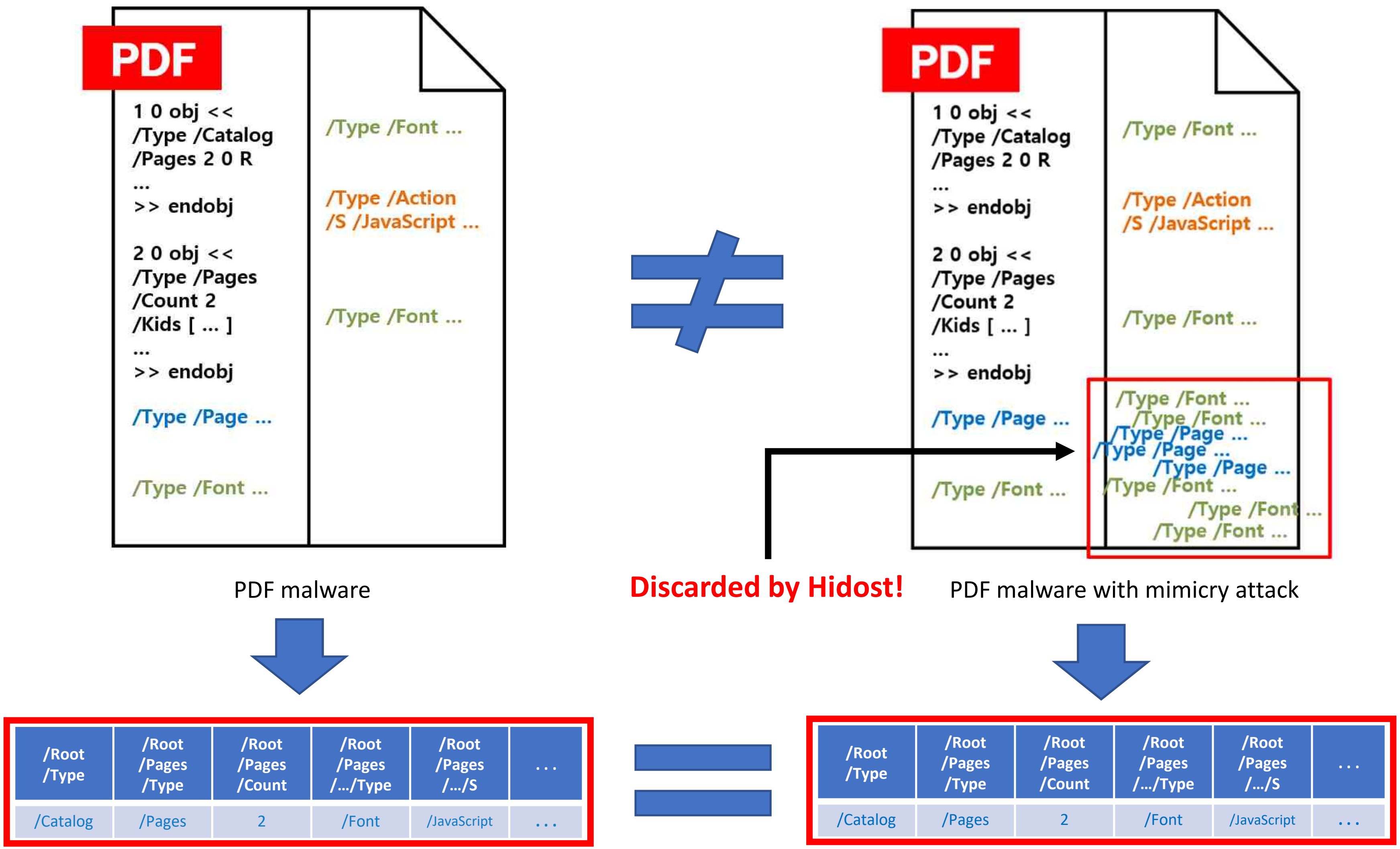


PDF malware with mimicry attack

/Root /Type	/Root/Pages /Type	/Root/Pages /Count	/Root/Pages/... /Type	/Root/Pages/... /S	. . .
/Catalog	/Pages	2	/Font	/JavaScript	. . .

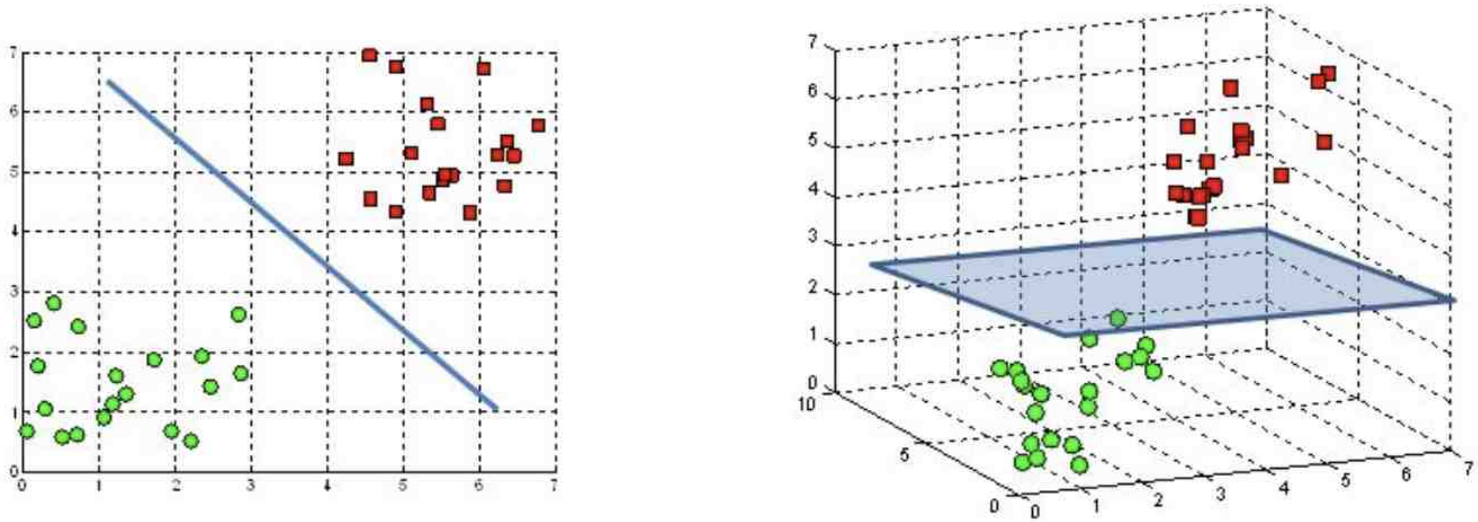


Hidost classification



Training with SVM

- **Hidost used the support vector machine (SVM) as a large set of features are used (a total of 6,087)**
- **SVM can deal with a large set of features**
- **SVM fits a hyperplane to data points in such a way that separates two classes**

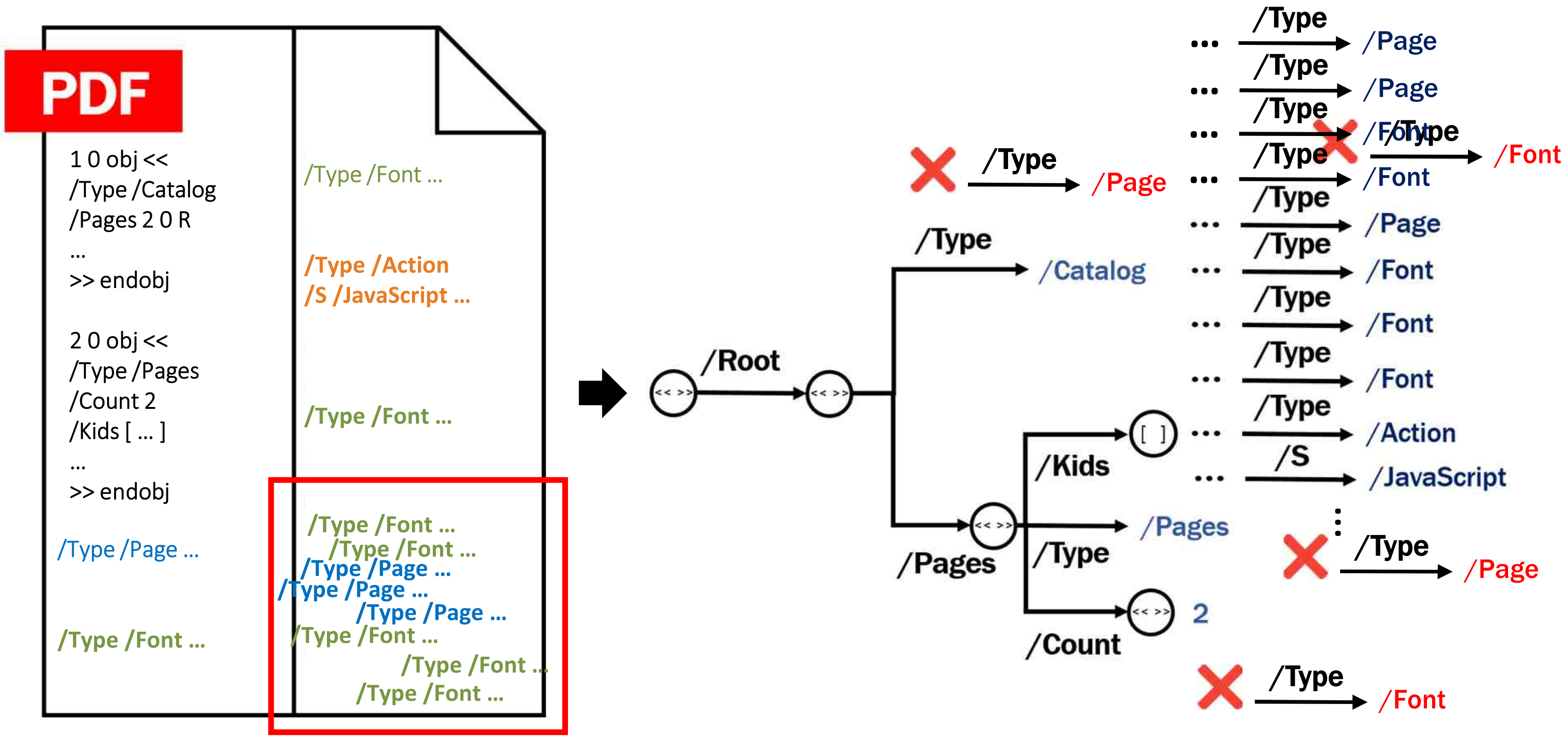


<Support Vector Machine>

Beating malware classifiers

- **A content-based classifier, PDFrate, has been subverted by mimicry attack techniques manipulating feature values.**
- **A structure-based classifier, Hidost, is also vulnerable to a mimicry attack crafted by additional human endeavor.**
- **An adversary may beat Hidost by insert objects from benign PDF into PDF malware to look structurally similar to benign PDF.**

Mimicry attack on Hidost



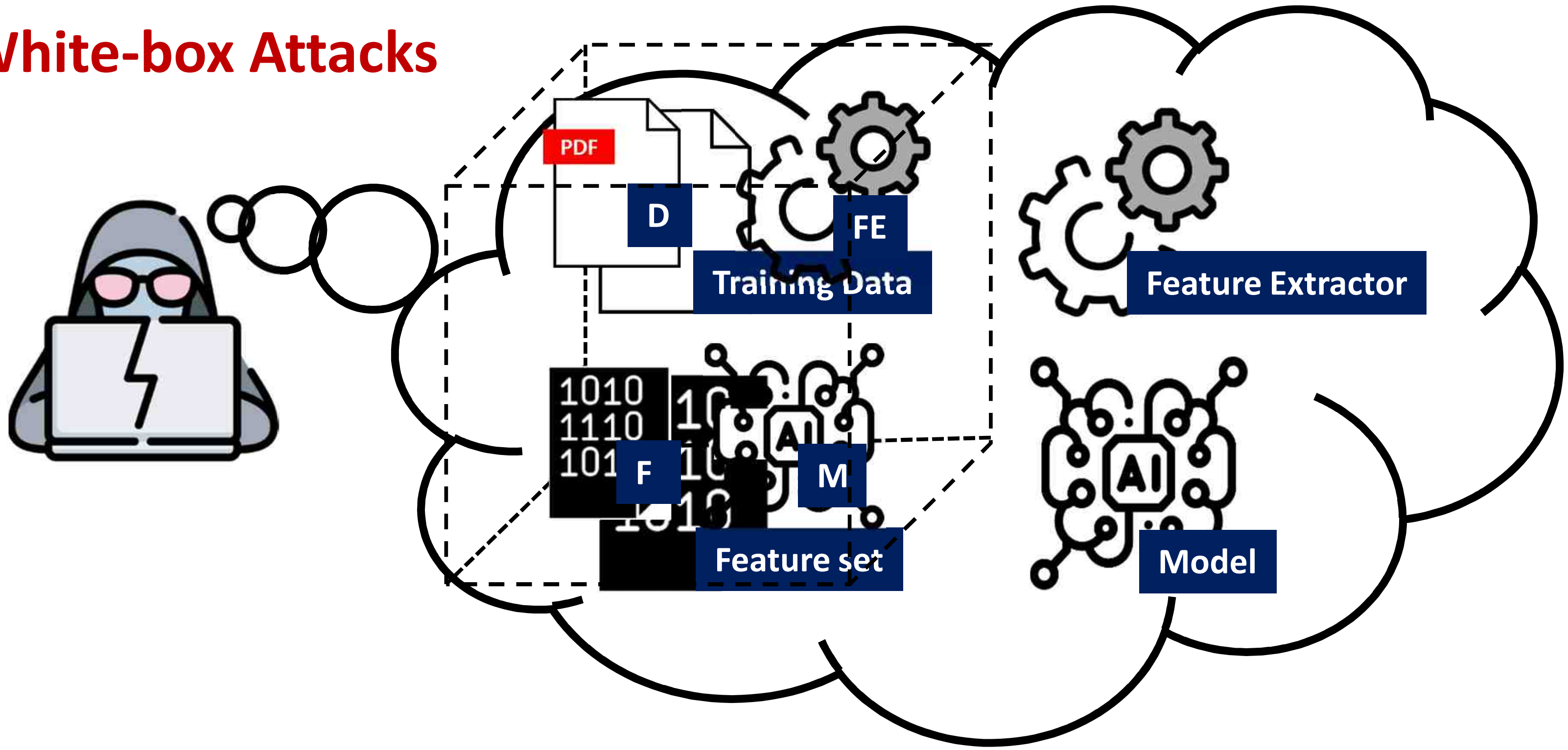
Malicious PDF with a
hand-crafted
mimicry attack

/Root /Type	/Root/Pages /Type	/Root/Pages/ Count	/Root/Pages/... /Type	/Root/Pages/... /S	/Root/Pages/ ??? .../Type	/Root/Pages/ ??? .../Type	/Root/Pages/ ??? .../Type	/Root/Pages/ ??? .../Type
/Catalog	/Pages	2	/Font	/JavaScript	/Font	/Font	/Page	/Page

Manual malware generation

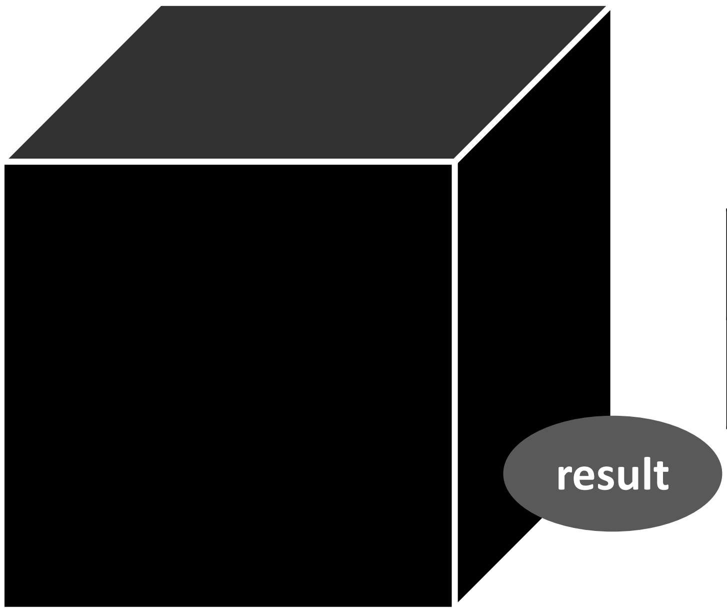
- First of all, too time consuming …
- The human usually need to understand the classifier
  - Must know everything about the classifier's detection process
  - Training data (D), Feature Extractor (FE), Feature set (F), Model (M)

White-box Attacks



Black-box attack

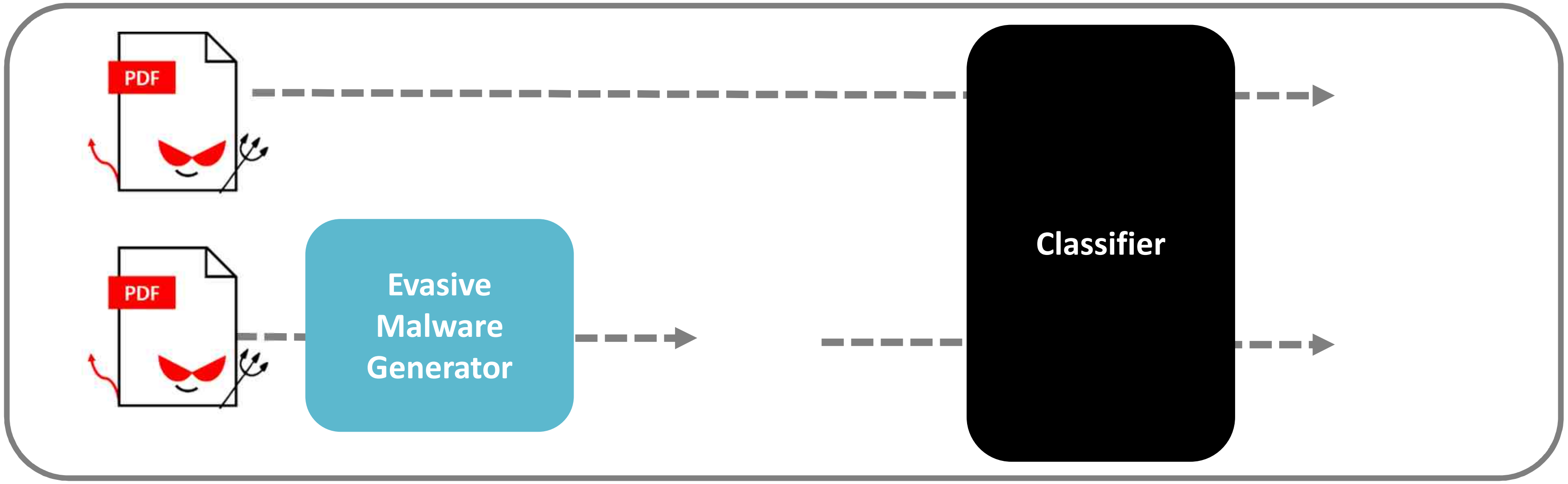
- **White-box attacks are not realistic in practice.**
- **Attackers usually have the lowest level of knowledge about classifier's detection process**
- **They are only allowed to know the final classification result (either benign or malicious) ➔ Black-box attacks**



	D	FE	F	M
White box	√	√	√	√
Black box	X	X	X	X

Automating malware generation

- **Develop an adaptive adversary that automatically generates adversarial example (malware) against black-box classifiers**
- **Goals**
  - **Test the robustness of existing classifiers against advanced attacks**
  - **Try to construct more robust classifiers**



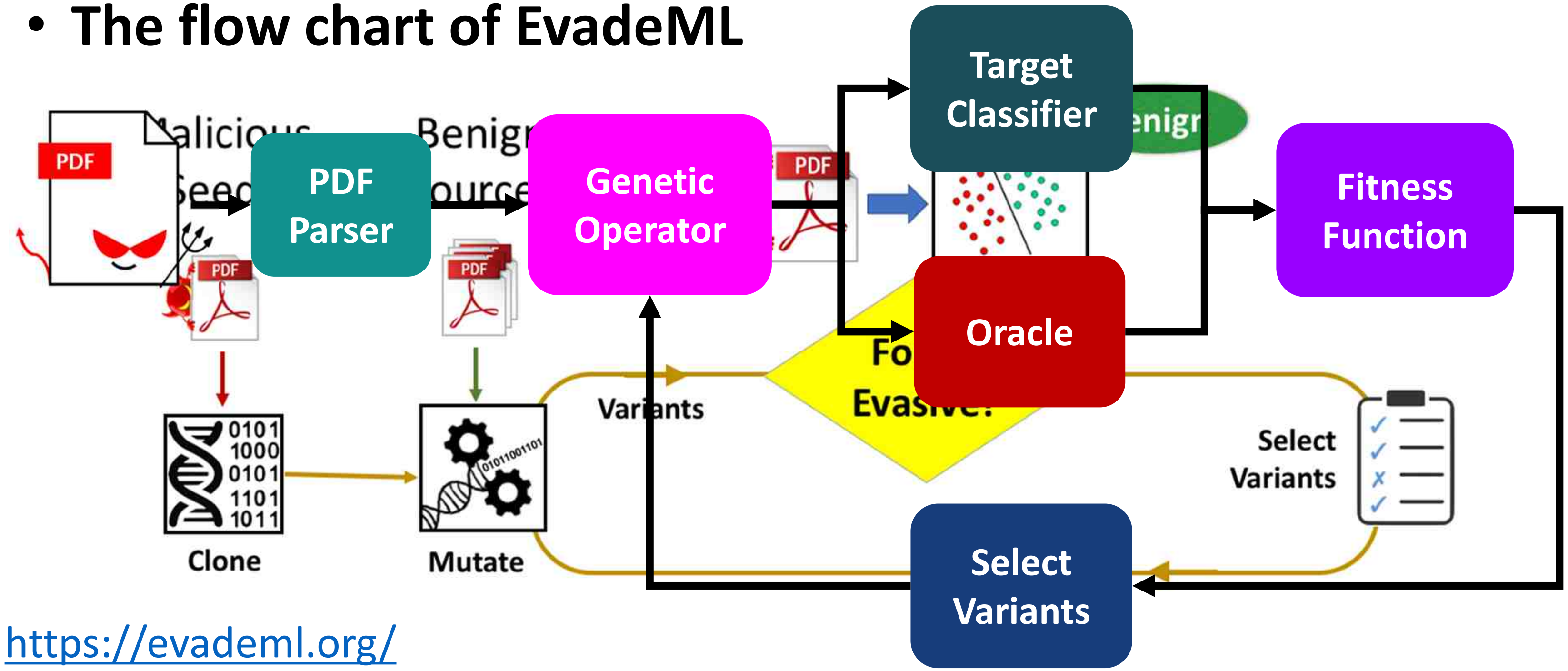
- **Adversarial examples must …**
  - **Maintain the maliciousness of the original malicious file**
  - **Evade the target classifier**

EvadeML

- Automatically generating adversarial example to evade PDF classifier

	Target classifier	Attack scenario	Strategy to evade classifiers	Strategy to maintain maliciousness
EvadeML	PDFrate Hidost	Black-box attack	Genetic programming (Random mutation)	X

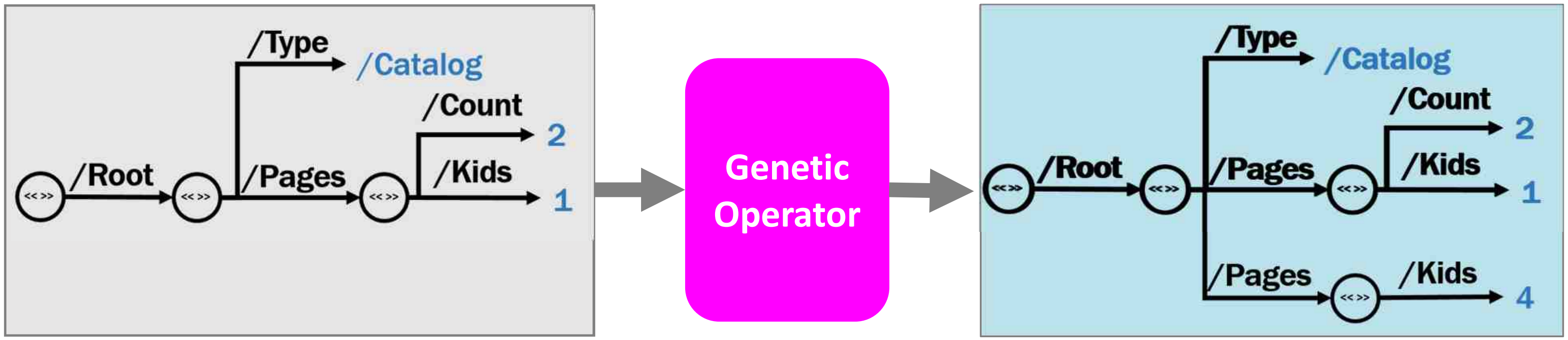
- The flow chart of EvadeML



https://evademl.org/

Genetic operators

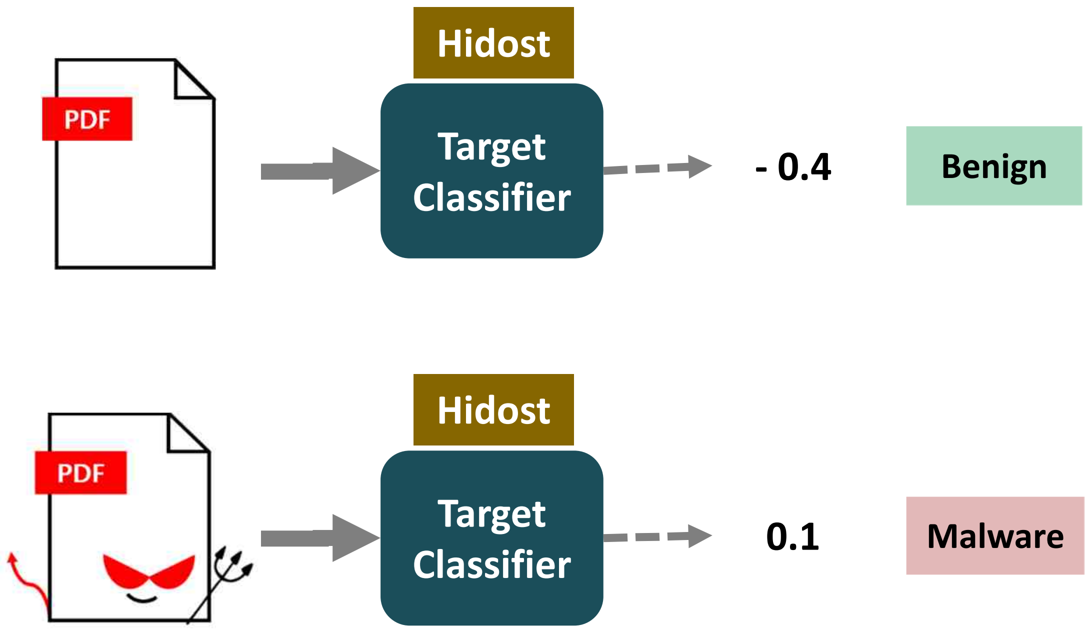
- **Generating variants by mutating the PDF malware**
- **Three operations for random mutation**
  - **Deletion: Object is removed**
  - **Insertion: Object is inserted (from benign file)**
  - **Replacement: Object is replaced (from benign file)**



Example of insertion operation

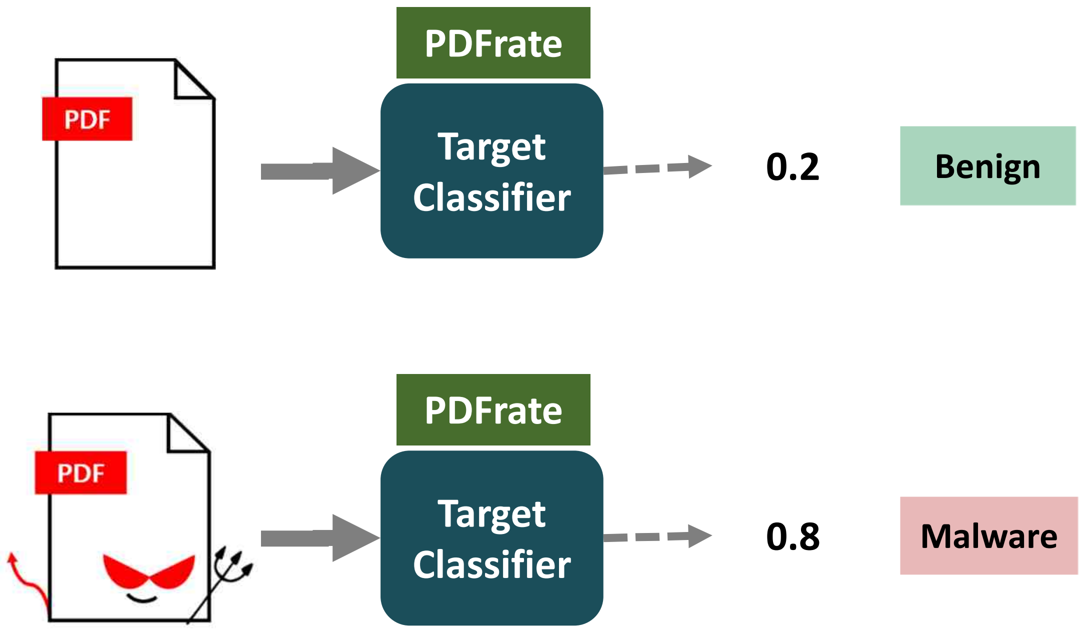
Bypassing Hidost

- **Classification threshold value is zero (0)**
  - **Score ≤ 0 : benign**
  - **Score > 0 : malware**



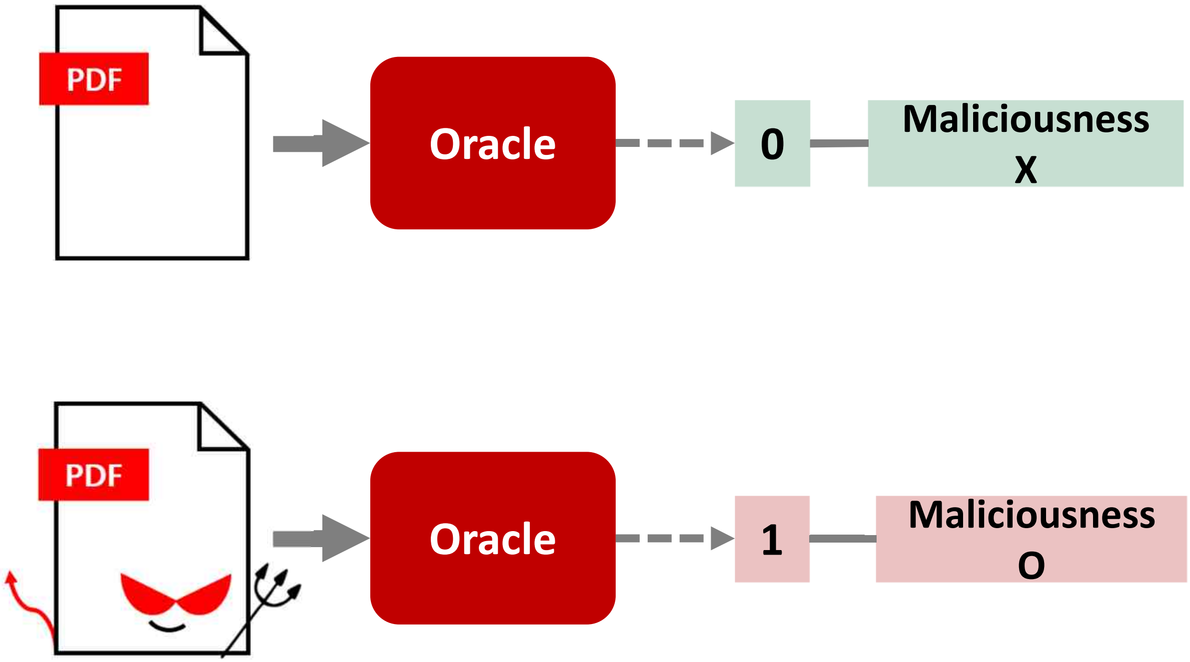
Bypassing PDFrate

- **Classification threshold value is 0.5**
  - **Score ≤ 0.5 : benign**
  - **Score > 0.5 : malware**



Oracle: Cuckoo Sandbox

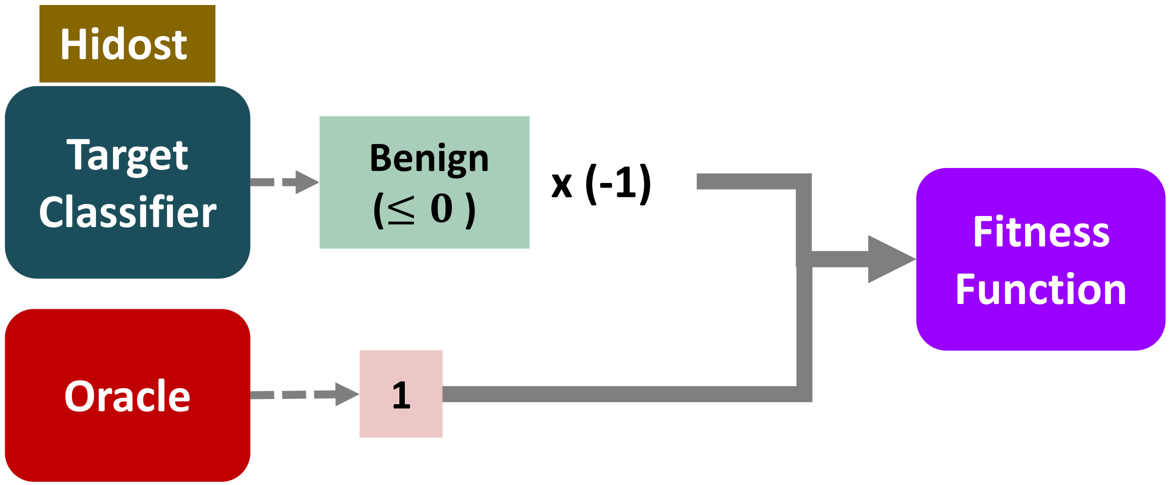
- **Verifying whether variant maintains the original malicious behavior**
- **Cuckoo sandbox runs a submitted sample with several virtual machines in parallel**



Fitness score

- **Fitness score of each generated variant**
- **High scores are better**

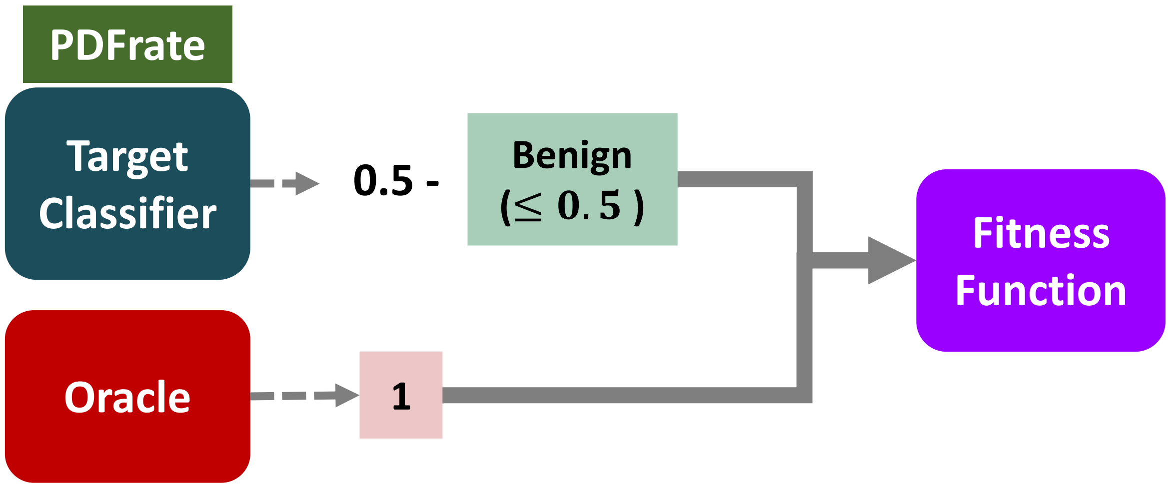
$$fitness_{hidost}(x) = \begin{cases} hidost(x) \times (-1) & oracle(x) = 1 \\ LOW\_SCORE & oracle(x) = 0 \end{cases}$$



Fitness score

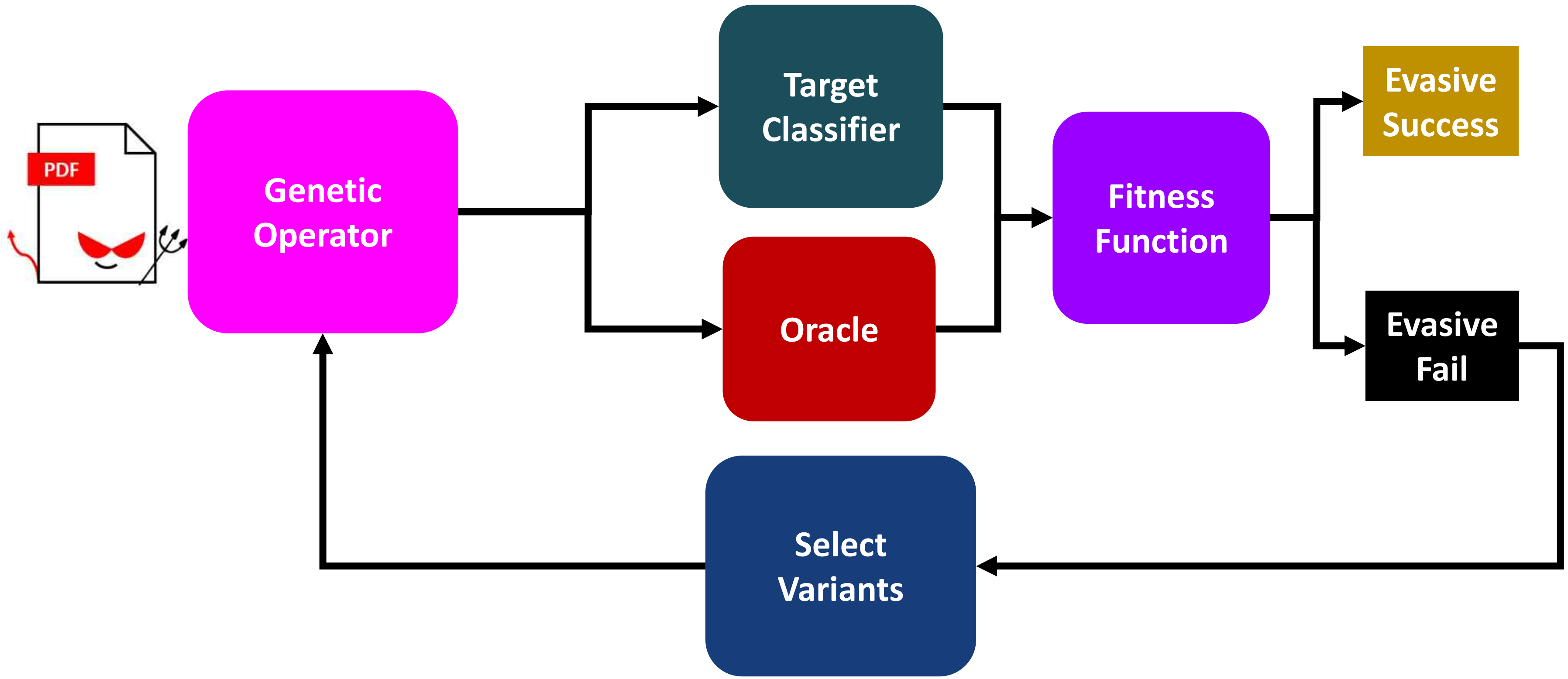
- **Fitness score of each generated variant**
- **High scores are better**

$$fitness_{pdfrate}(x) = \begin{cases} 0.5 - pdfrate(x) & oracle(x) = 1 \\ LOW\_SCORE & oracle(x) = 0 \end{cases}$$

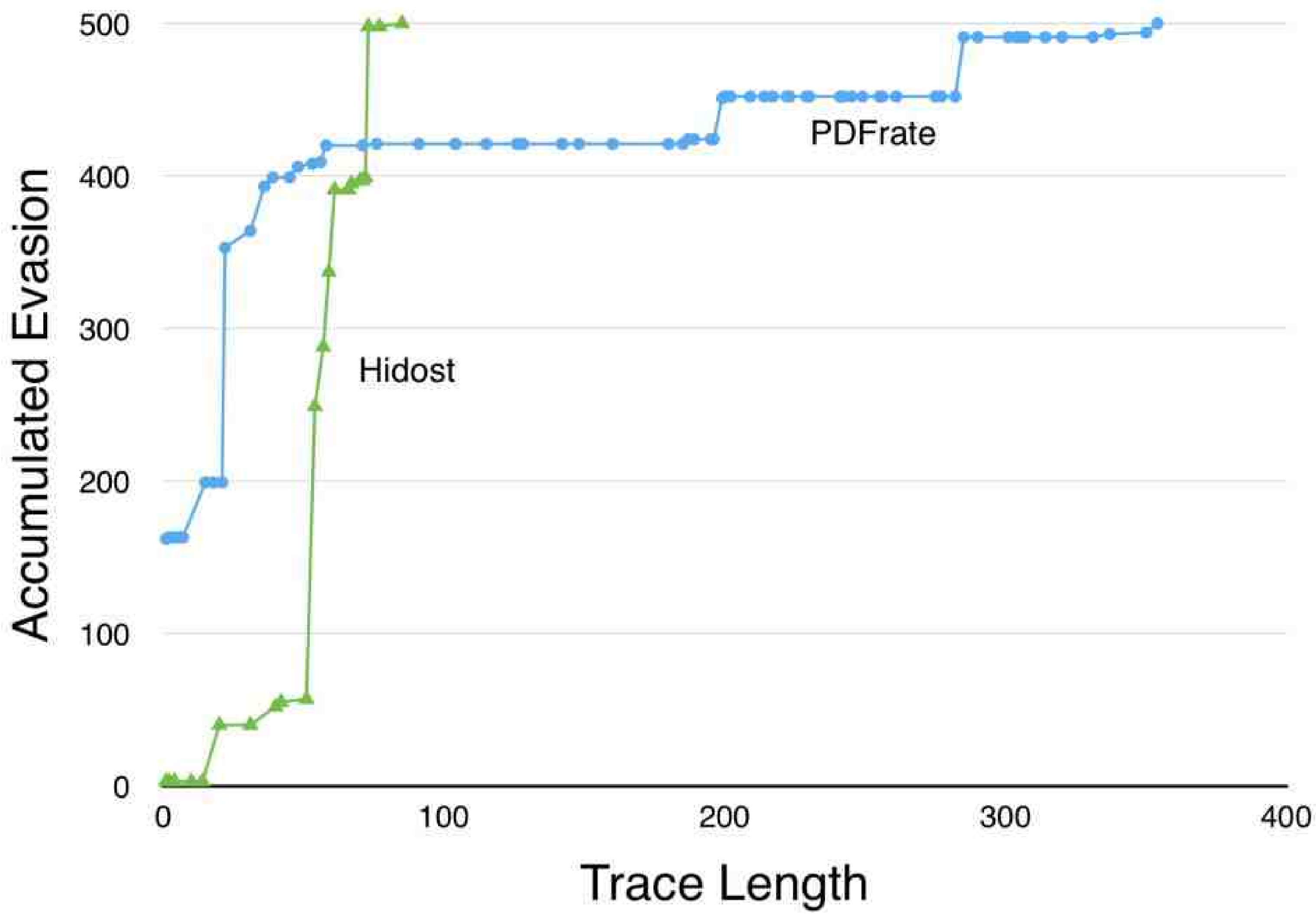


Genetic programming

- **The process continues over multiple generations until the adversarial example is created**
- **No learning-based intelligence in generating variants**



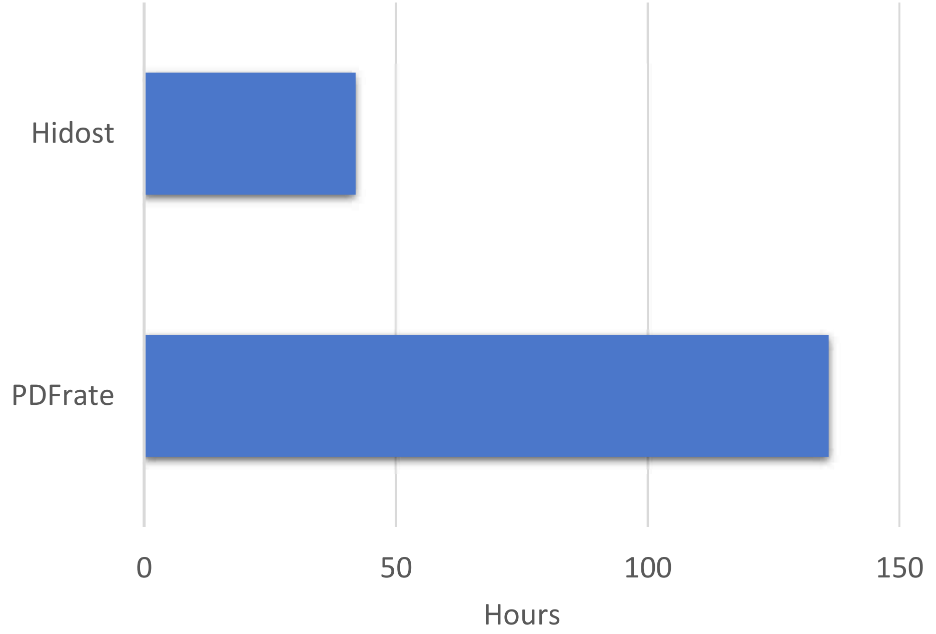
Trials to evade classifiers



Limitations

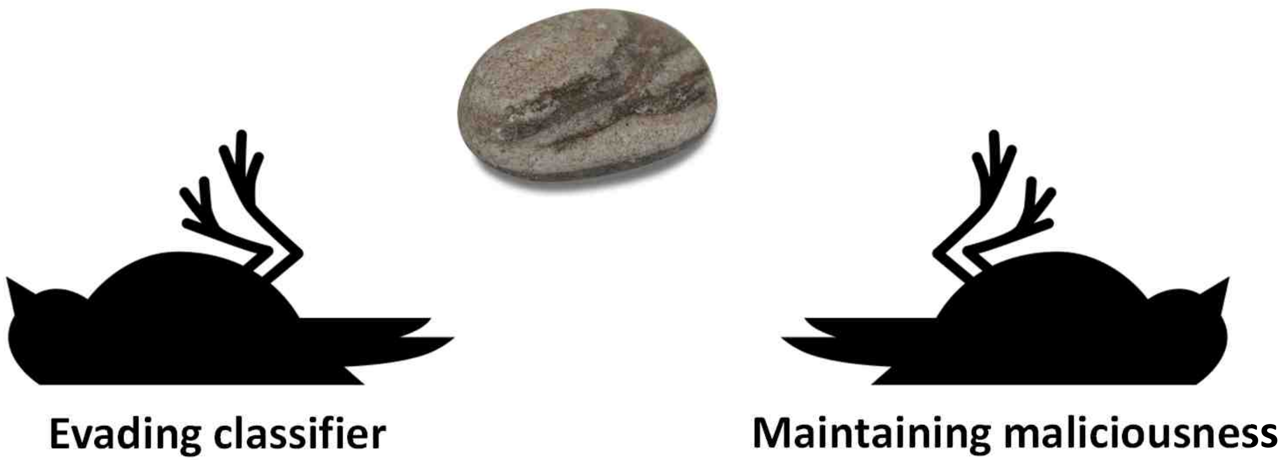
- **All generated variants must go through the oracle**
- **Due to lack of intelligence, most variants are generated randomly, losing the original maliciousness**
- **Hence, the speed to generated evasive malware is high**

➔ > 120 hours are required



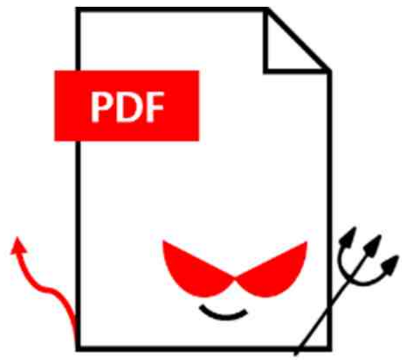
Our approach

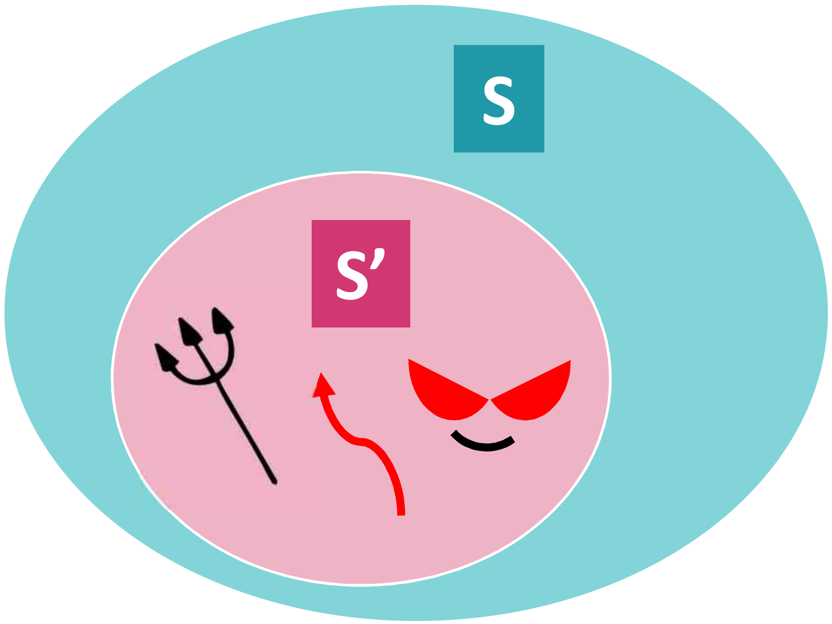
- **To overcome the limitations of EvadeML, we employ a generative ML model that can automatically generate adversarial examples.**
- **By learning the structures of both benign and malicious PDFs, the model aims to simultaneously achieve two goals: evading classifier and maintaining maliciousness.**



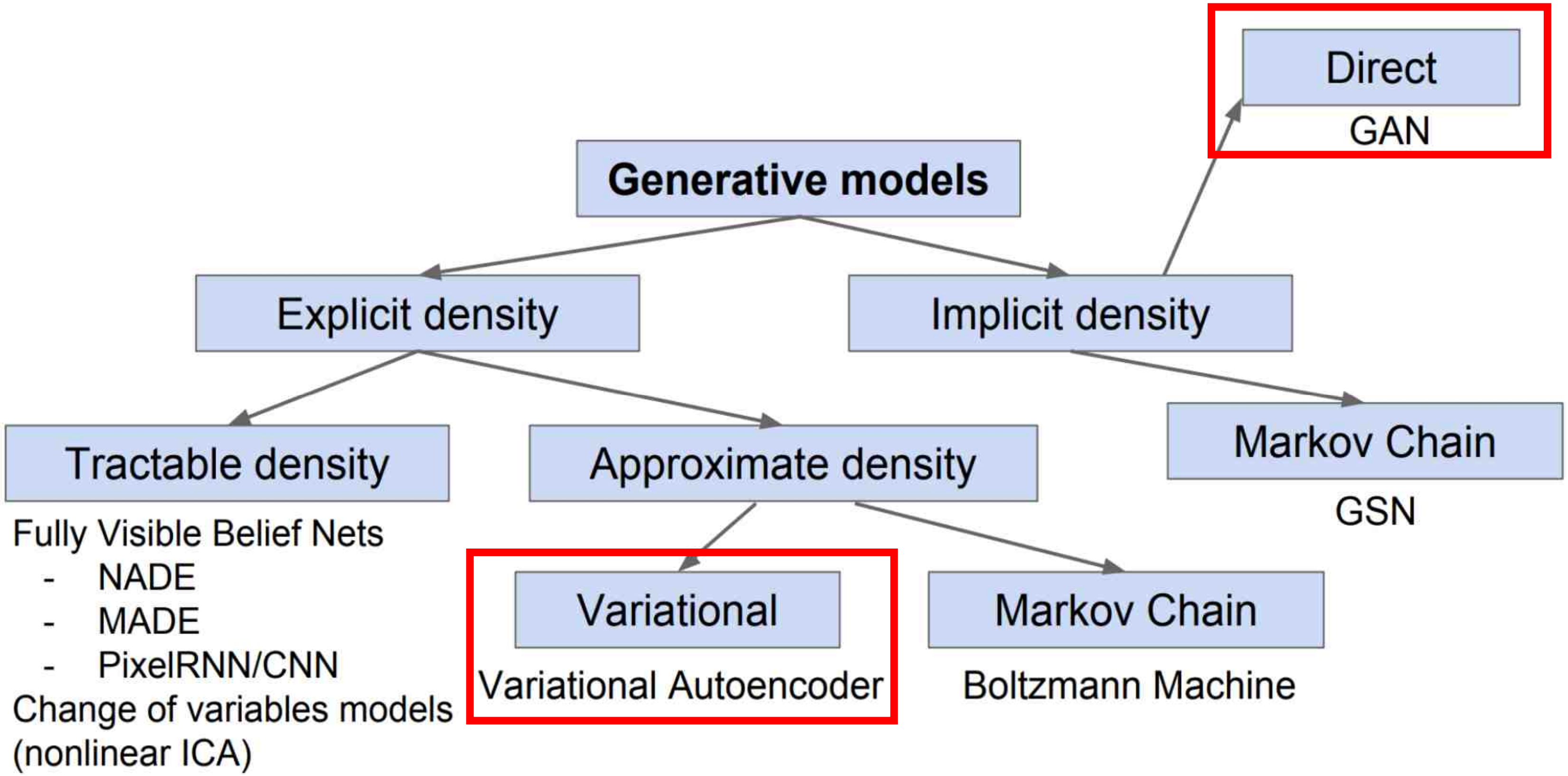
Learning to keep maliciousness

- **The generator model must not modify the features that are related to the malicious behavior**
- **Let S be the entire feature set, S’ be the features related to the malicious behavior**
- **We have another ML model that guides the generator to only modify the features in S-S’.**





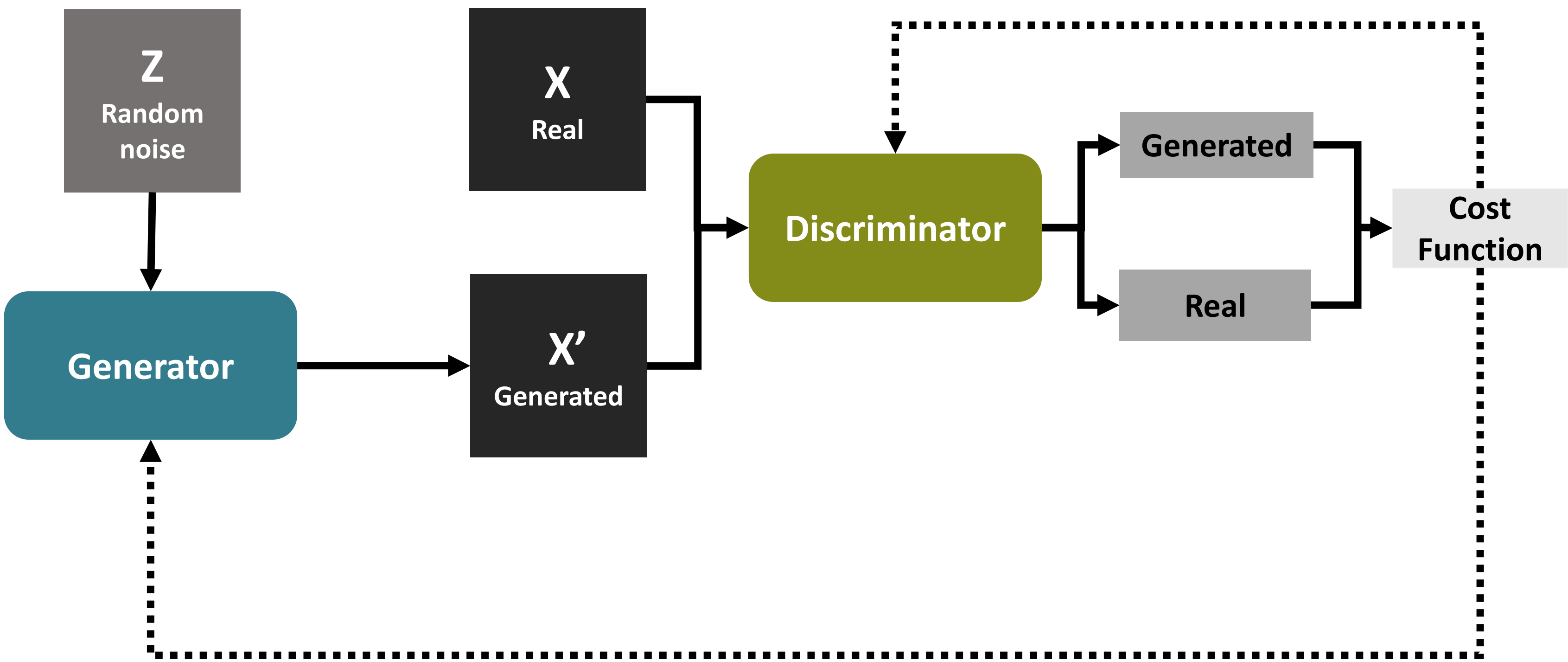
Taxonomy of generative models



<Ian Goodfellow, 2016>

Inspired by GAN

- **Generative Adversarial Network (GAN)**
- **Suitable in generating variants**

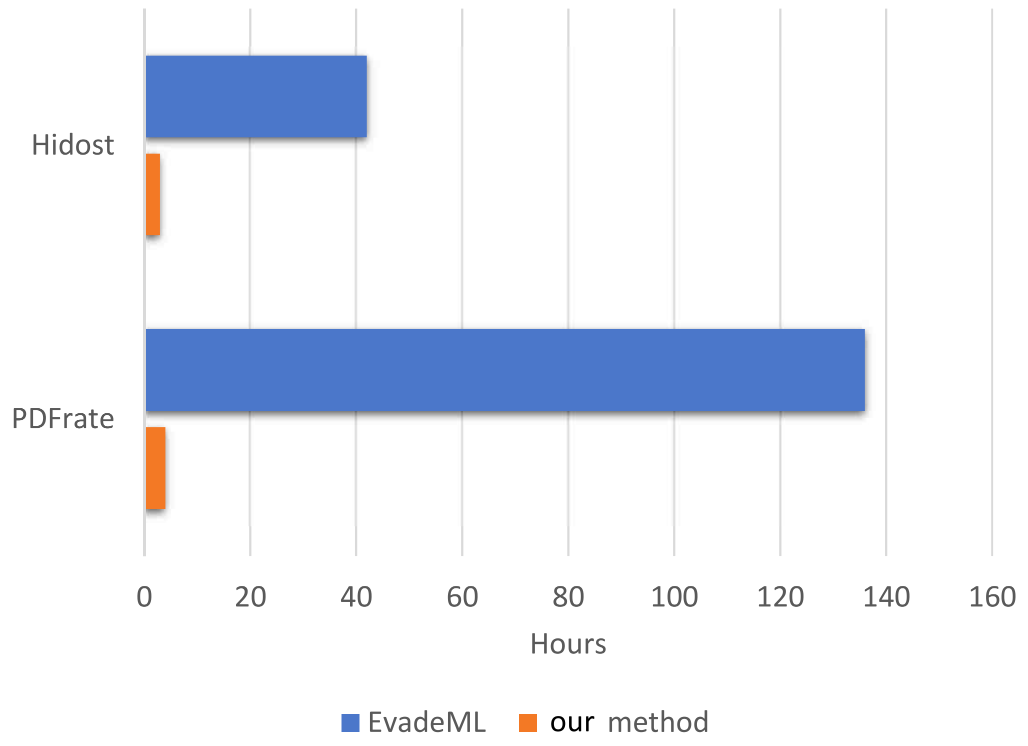


Our way to keep maliciousness

- **Use the discriminator as a assistant tool to find S-S’ and only modify those features**
- **Hence, successfully maintain the original maliciousness**

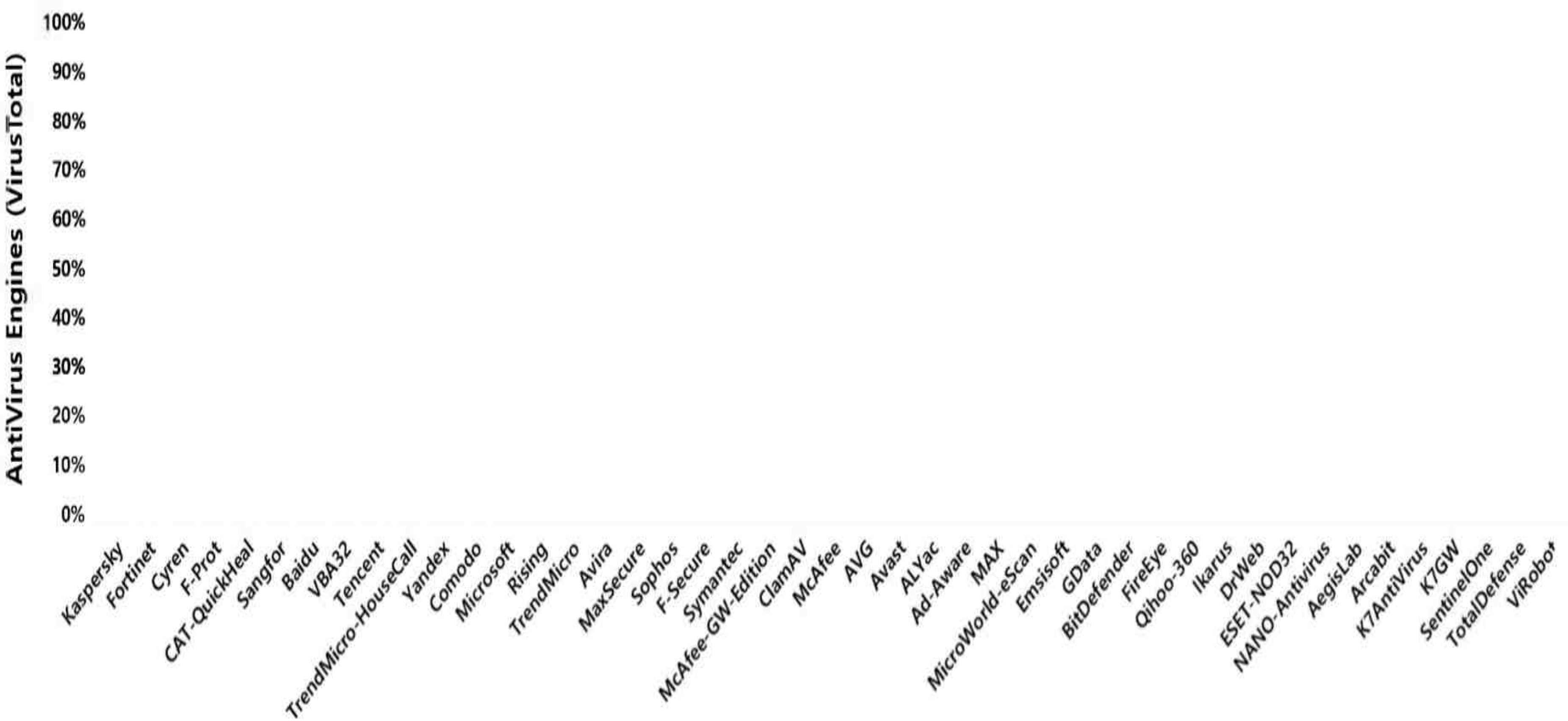
Speed comparison with EvadeML

- **13 times faster than EvadeML (to evade Hidost)**
- **30 times faster than EvadeML (to evade PDFrate)**



Evasion success rate

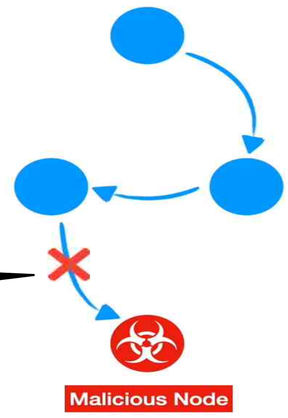
- **Attack against commercial anti-virus engines**
- **Achieved more than 60% evasion success rate in 27 engines**



Arms race is on-going…

- **EvadeML has been subverted**
  - **Usenix Security ’19: Retraining ML PDF classifiers with S’**
  - **Usenix Security ’20: Enhancing robustness of Hidost and PDFrate**
- **Extension to binary malware**
  - **Binary has much more complex structures/semantics than PDF**
  - **The challenge is difficult to retain code semantics which can easily be broken if binary malware is randomly mutated**
  - **Maliciousness will be lost if the code semantics is not retained**
  - **IEEE Security & Privacy ’20 : generate Android malware by selecting appropriate benign features that preserve the original code semantics**

If code semantics is broken,
malicious node is non-reachable
(Lost maliciousness)



Thank you!

Yunheung Paek

SNU Security Research Group

Electrical and Computer Engineering Department

Seoul National University