

# XII STIC CCN-CERT CONFERENCES

Cybersecurity,  
towards effective response and deterrence



Looking for the perfect  
signature: an automatic  
YARA rule generation  
algorithm in the AI era



- Andrea Marcelli (@\_S0nn1\_)
- PhD Candidate at Politecnico di Torino  
Security Researcher at Hispasec Sistemas
- amarcelli@hispasec.com



**POLITECNICO  
DI TORINO**

**HISPASEC**



- Luca Cetro
- MSc in Computer Engineering  
Independent Security Researcher
- [luca.c94.01@gmail.com](mailto:luca.c94.01@gmail.com)



**POLITECNICO  
DI TORINO**



# Agenda (30 mins)

Introduction

The Framework

YaYaGenPe

Results and Demo

Conclusions



# Introduction

# What is a malware signature?

A **unique pattern** that indicates the presence of malicious code

As malware evolves, new signatures need to be generated frequently

**Syntactic signatures** are based on unique sequences of instructions or strings

\* this is where most of the existing tools and researches focus on

**Semantic signatures** provides an abstraction of the program behavior

Our target is **Microsoft Windows Portable Executable malware**, although the approach is generic

This is not just another ML classifier.

# About YARA



“YARA is to files what Snort is to network traffic”

Designed to be fast

The **de-facto standard** language to write malware signatures

Natively supports **syntactic signatures** (strings + regex + hex)

**Semantic signatures** are defined through custom modules.

# An example of YARA rule

```
rule YaYakes {  
    meta:  
        author = "YaYaGenPE -- Yet Another Yara Rule Generator"  
  
    condition:  
        pe.image_base == 708640768 and  
        pe.resources[6].language == 1030 and  
        pe.resources[36].type == 10 and  
        pe.resources[37].id == 104 and  
        pe.imports("user32.dll", "GetCursorPos") and  
        ...  
}
```

# Work motivations



**Scalable**  
1000+ files



**Fast**  
100 file in ~5 min



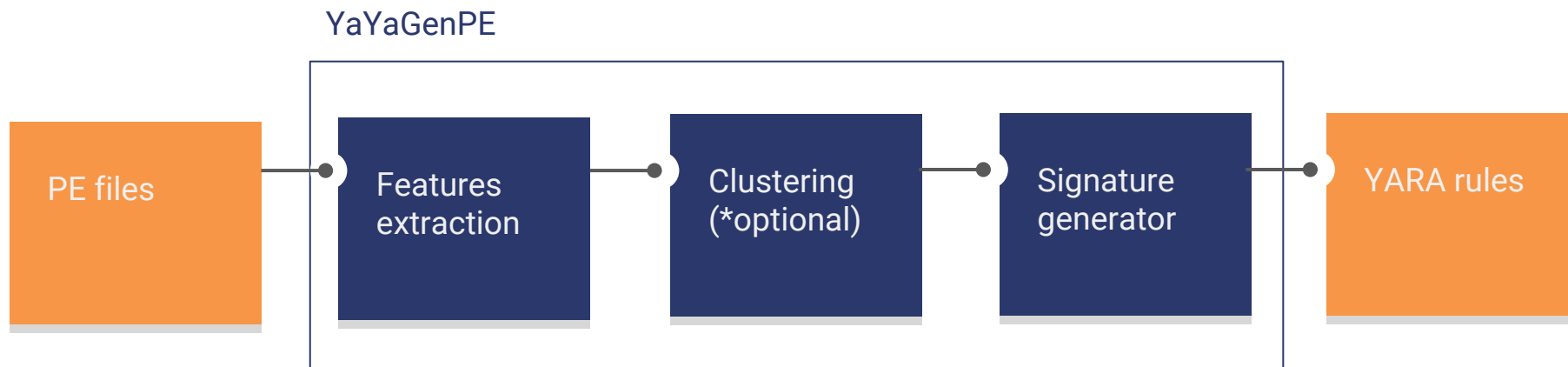
**Limited FP**  
lower unwanted  
detections



**Increase TP**  
detect new  
variants

# The framework

# The framework workflow



# Key point

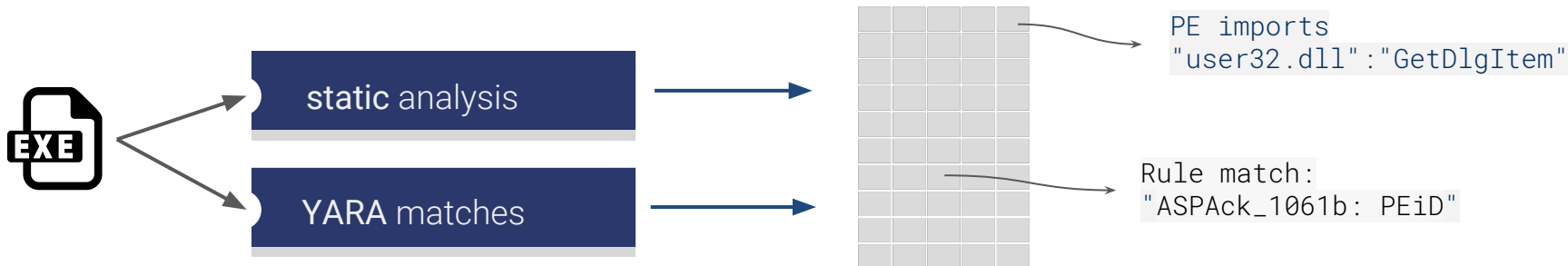
## INPUT IS GENERIC. OUTPUT IS PURE YARA

**The input** is a set of Windows files for which we want to create a rule  
It could be a set of malware or goodware, a tight or a loose cluster

**The output** is a set of pure YARA rules that match all the files in input  
If the files are more similar, **less rules** are generated, and they are **more effective**

\* Ruleset can be uploaded to VT and can be directly used for the retrohunt.

# 1. Feature extraction

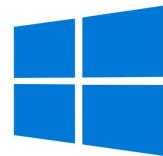


Each **block** is a feature extracted through the analysis, or a YARA rule that matches the file

\*Python module **pefile** is used to extract features accepted by YARA **PE module** sections, resources, rich signature and import/export directories

**Existing YARA rules** (reduced in CNF) add expert knowledge.

\* <https://github.com/erocarrera/pefile>



# Key point

## FEATURES ARE GENERIC

YaYaGen implements an approximation algorithm to solve a hard optimization problem

**Features can be anything.** A set of features simply identify a malware sample  
Anything can be used as far as it produces a **pure YARA rule**

**Strings, binary patterns, regex** can be easily added in the features extraction phase.

## 2. Clustering

It **reduces the complexity** of signature generation process

Allow the framework to scale with 1000+ inputs

2 approaches: **density based** and **unsupervised decision tree**

Each cluster is the input of the signature generation algorithm.

# Density based

## HDBSCAN\*

Enhanced version of DBSCAN

- Real data has outliers
- Scale well

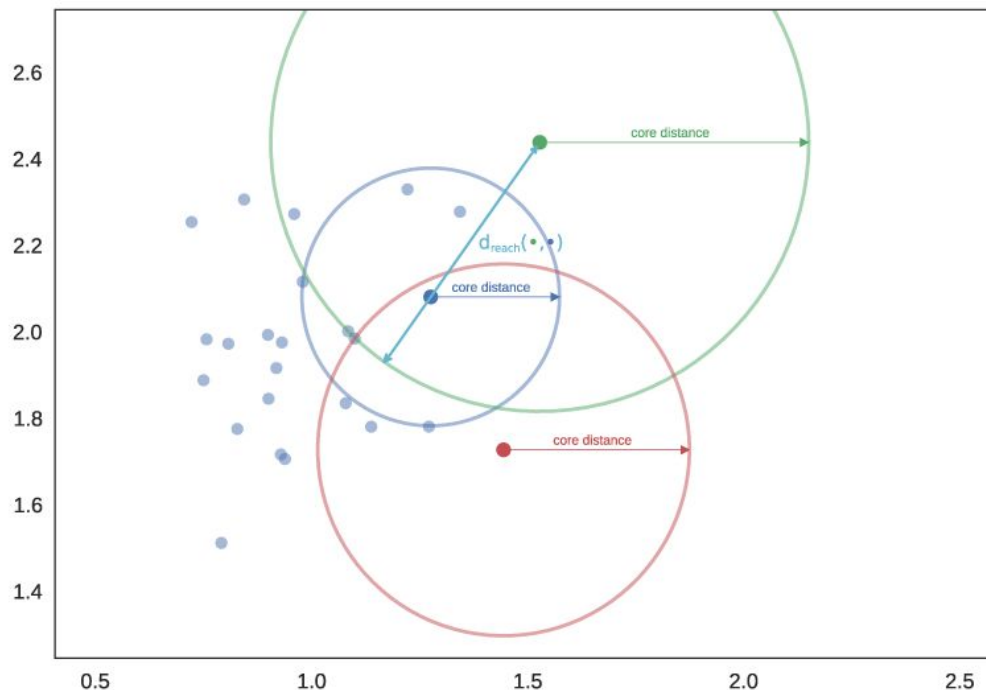
Jaccard distance gives the best results

Clusters are small (~5 points)

`min_clust_size = 2`

`min_points = 2`

\* <https://github.com/scikit-learn-contrib/hdbscan>



# UDT: Unsupervised decision tree

Each **cluster is splitted** into two new ones based on the value of a single feature

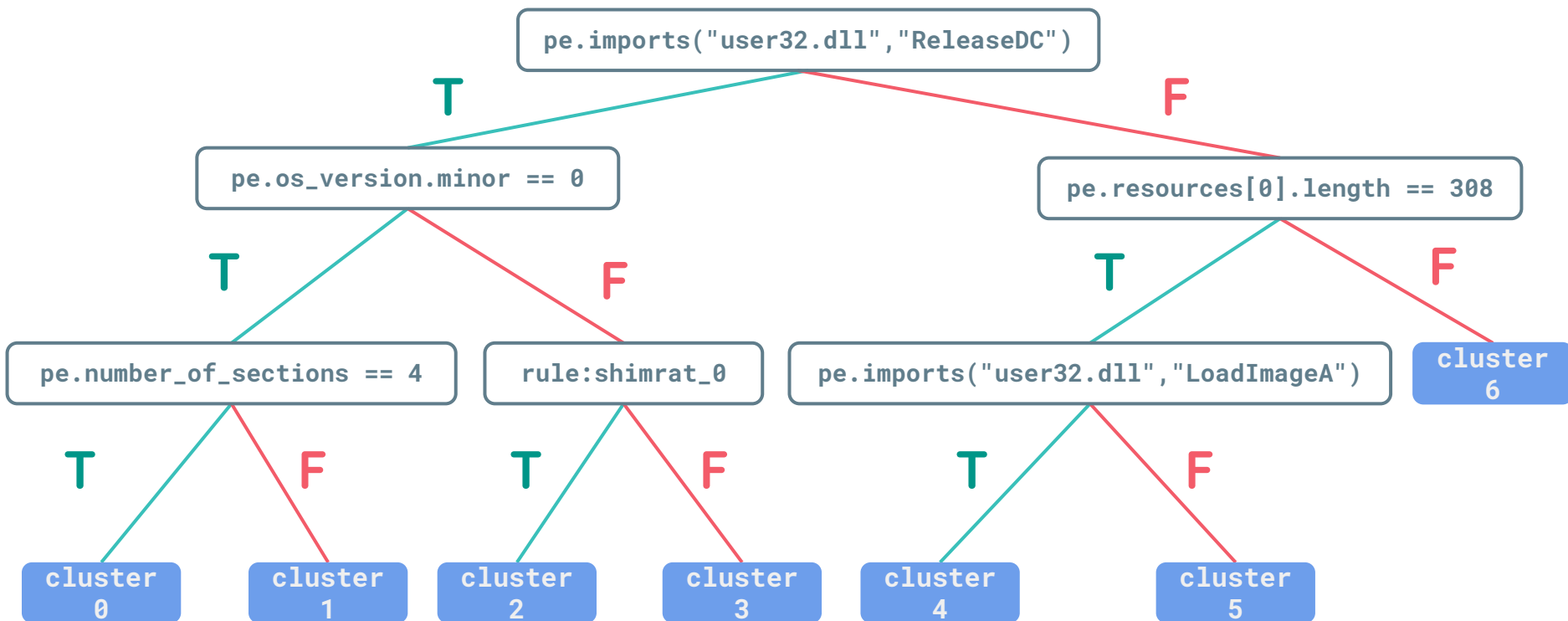
The **best splitting feature** is the one that **maximises** the distance among centroids

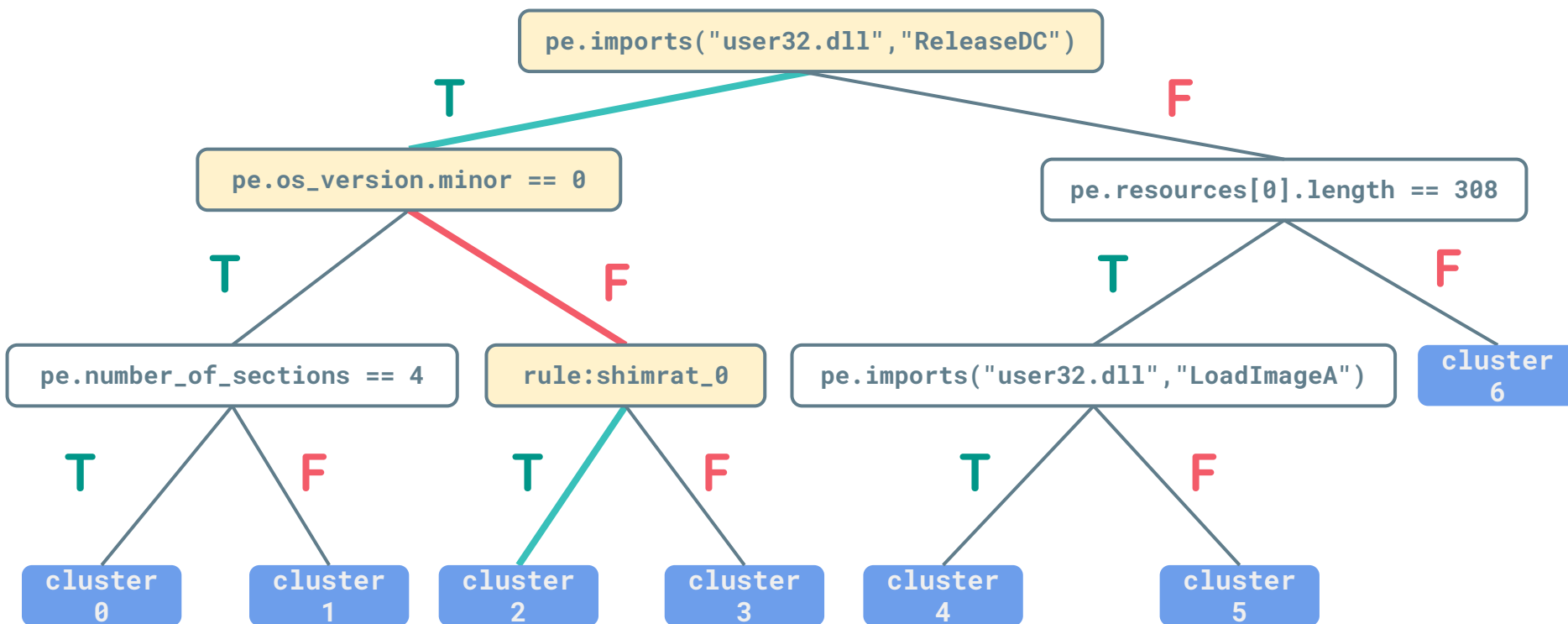
Cluster centroids are approximated, and **Russel-rao and Jaccard** distances are used

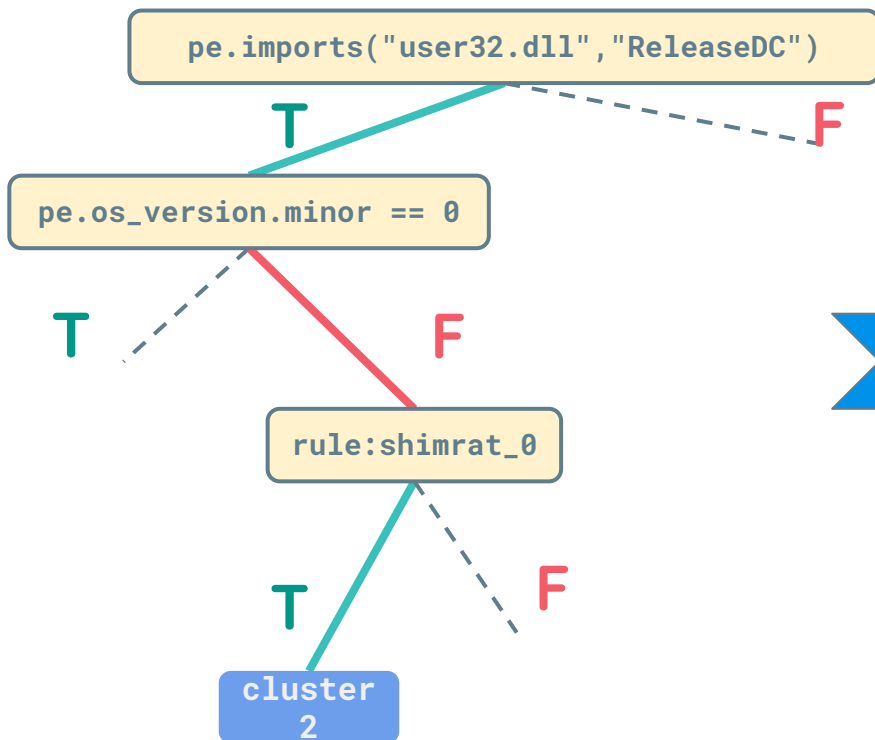
The stopping criterion is the distance between centroids (experimentally set)

The **splitting feature** can be easily added to the generated rules.

\* few features can be included in the YARA rule with the “not” logic operator.





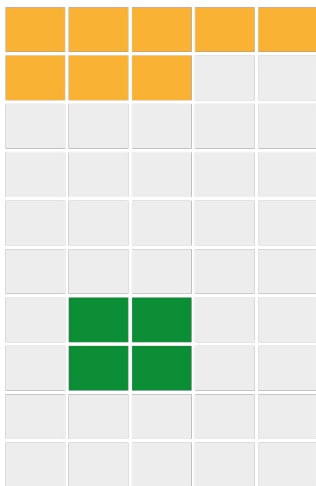


## Cluster 2 FEATURES to YARA literals

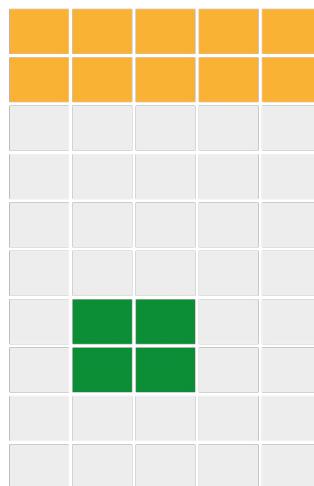
```
pe.imports("user32.dll","ReleaseDC")  
and  
not( pe.os_version.minor == 0 )  
and  
shimrat_0
```

# 3. The signature generation

Sample 1



Sample 2

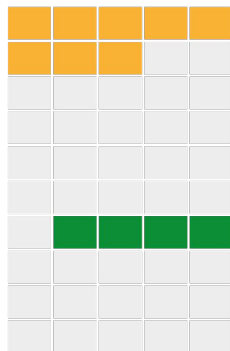


=

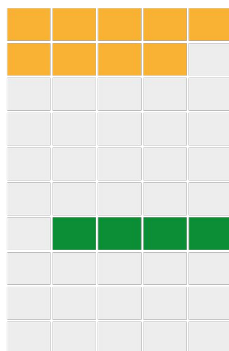
Signature



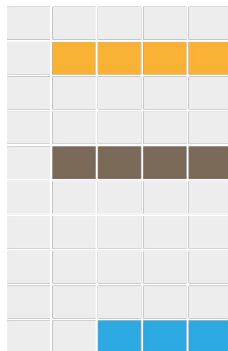
Sample 1



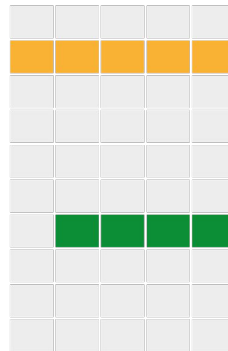
Sample 2



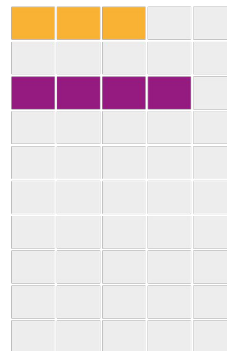
Sample 3



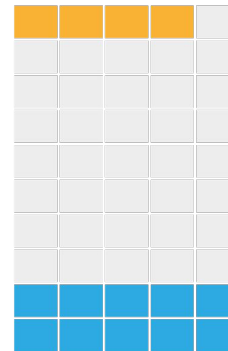
Sample 4



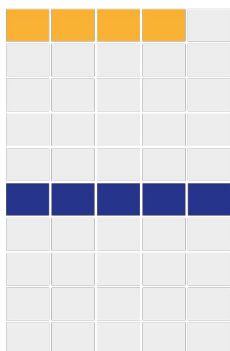
Sample 5



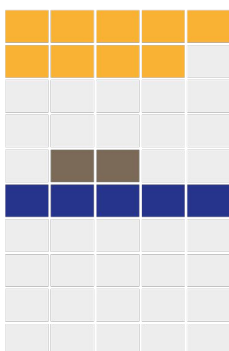
Sample 6



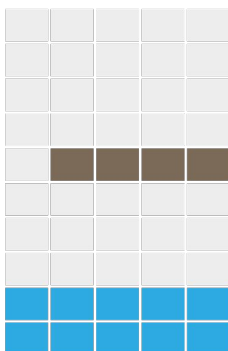
Sample 7



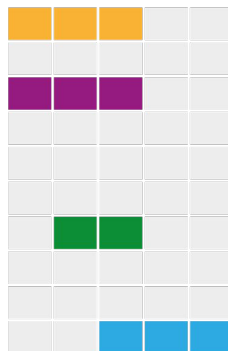
Sample 8



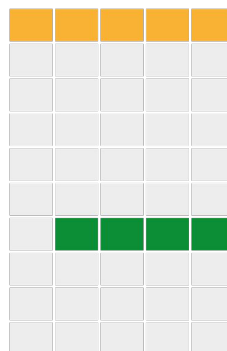
Sample 9



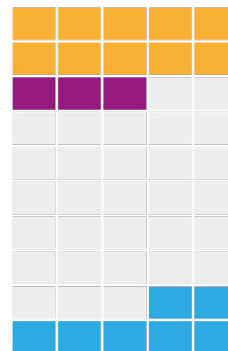
Sample 10



Sample 11



Sample 12



# The solution



Finding the optimal attributes subsets is the goal of the signature generation process

The problem can be reduced to a variant of the set cover problem (NP-complete)

A **dynamic greedy algorithm** builds the signature as a disjunction of clauses.

# DNF

$$\underbrace{(l_1 \wedge l_2 \wedge l_3)}_{\text{clause}} \vee \underbrace{(l_4 \wedge l_5)}_{\text{literal}}$$

Each **clause** is a valid YARA rule

Each **clause** can be **weighed**: a YARA rule can be weighed too

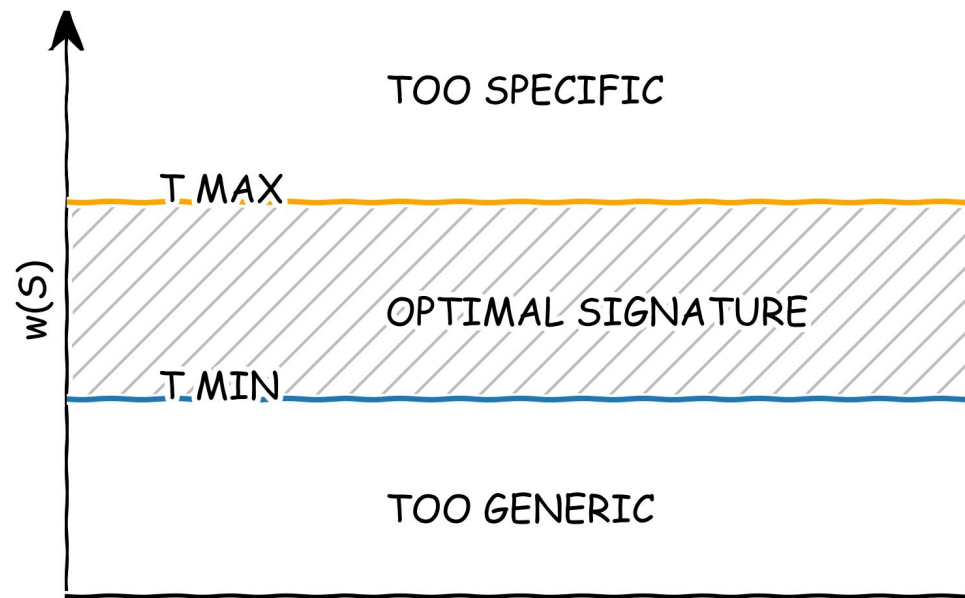
Currently **the weight is the number of features** (i.e., literals).

# Generality vs specificity

A weighting system evaluates the rules

The **higher the weight**, the **less FPs**  
Possibly more FNs

The **lower the weight**, the **more FPs**  
Possibly less FNs



# YaYaGenPE

# Yet another YARA rule Generator

\*YaYa is grandma in ES



2 clustering algorithms (*HDBSCAN*, *UDT*), 2 algorithms for the rule generation (*clot*, *greedy*)

Includes a **YARA rule parser** for the feature extraction

Written in Python 3 (~2 thousand lines of code)

Fork of the original YaYaGen

<https://github.com/jimmy-sonny/yayagen>

# Results

# Tests performed

- 1) **Cluster quality** (V-Measure and Homogeneity)
- 2) **True positives** (malware coverage)
- 3) **False positives** resistance
- 4) **Dataset coverage**
- 5) **Packer** resistance (UPX packer)

# Comparison existing tools

|                       | YaraGenerator  | YarGen            | YaBin           | BASS          | YaYaGenPE    |
|-----------------------|----------------|-------------------|-----------------|---------------|--------------|
| Pure YARA             | YES            | YES               | YES             | NO            | YES          |
| Based on              | Strings        | Strings           | Binary          | Binary        | PE + rules   |
| Algorithm             | Common strings | Whitelist strings | Whitelist funcs | Bindiff + LCS | Set covering |
| Training set coverage | NO             | NO                | NO              | YES           | YES          |
| Packing               | NO             | NO                | YES             | YES           | YES          |
| Clustering            | NO             | NO                | NO              | YES           | YES          |
| Scalable              | YES            | YES               | YES             | NO            | YES          |
| High coverage         | NO             | YES               | YES             | NO            | YES          |

# Comparison input 47 samples

## CRYPTOWALL FAMILY

**MALWARE** 6,881 samples

**GOODWARE** 3,413 samples

| TOOL      | ALGORITHM            | # RULES   | FPS       | TPs        |
|-----------|----------------------|-----------|-----------|------------|
| YaYaGenPE | UDT + GREEDY         | 29        | 0         | 76         |
|           | UDT + GREEDY + RULES | 31        | 0         | 75         |
|           | HD + GREEDY + RULES  | <b>23</b> | 0         | <b>80</b>  |
| YarGen    | RULES Z0             | <b>53</b> | <b>1</b>  | <b>130</b> |
|           | RULES Z0 + OPCODES   | 53        | 0         | 86         |
| YaBin     | Yara (-y)            | <b>36</b> | 0         | <b>76</b>  |
|           | YaraHunt (-yh)       | 36        | <b>10</b> | 194        |

# Comparison input 533 samples

## CERBER FAMILY

**MALWARE** 6,881 samples

**GOODWARE** 3,413 samples

| TOOL      | ALGORITHM            | # RULES | FPS | TPs |
|-----------|----------------------|---------|-----|-----|
| YaYaGenPE | UDT + GREEDY         | 65      | 2   | 854 |
|           | UDT + GREEDY + RULES | 64      | 2   | 896 |
|           | HD + GREEDY          | 137     | 0   | 768 |
| YarGen    | RULES Z0             | 328     | 7   | 705 |
|           | RULES Z0 + OPCODES   | 321     | 4   | 687 |
| YaBin     | Yara (-y)            | 157     | 0   | 737 |
|           | YaraHunt (-yh)       | 157     | 16  | 937 |

# Comparison input 2478 samples

## TESLACRYPT FAMILY

**MALWARE** 6,881 samples**GOODWARE** 3,413 samples

| TOOL      | ALGORITHM            | # RULES     | FPs       | TPs         |
|-----------|----------------------|-------------|-----------|-------------|
| YaYaGenPE | UDT + GREEDY         | <b>497</b>  | <b>0</b>  | <b>3349</b> |
|           | UDT + GREEDY + RULES | 493         | 0         | 3373        |
|           | HD + GREEDY          | 837         | 0         | 3237        |
| YarGen    | RULES Z0             | <b>2782</b> | <b>2</b>  | <b>3367</b> |
|           | RULES Z0 + OPCODES   | 2760        | 0         | 3226        |
| YaBin     | Yara (-y)            | <b>1166</b> | 0         | <b>3172</b> |
|           | YaraHunt (-yh)       | 1166        | <b>68</b> | 4027        |

# Retrohunt evaluate FPs and TPs

| Family           | Algorithm            | Input size | Total matches* | TP   | FPS |
|------------------|----------------------|------------|----------------|------|-----|
| OlympicDestroyer | UDT + GREEDY + RULES | 22         | 143            | 100% | 0   |
| Sagecrypt        | HD + CLOT + RULES    | 47         | 136            | 100% | 0   |
| Crowti           | UDT + GREEDY + RULES | 75         | 66             | 100% | 0   |
| Scatter          | UDT + GREEDY         | 12         | 57             | 86%  | 8   |
| Scatter          | UDT + GREEDY + RULES | 12         | 35             | 88%  | 4   |
| Fareit           | CLOT + RULES         | 6          | 385            | 100% | 0   |
| Shiz             | UDT + CLOT + RULES   | 104        | 12             | 100% | 0   |

\* Using a dataset of ~100 TB

# Packer UPX malware rules vs. UPX goodware

| Algorithm            | rule:Cerber | rule:Locky | rule:Upatre | rule:Zerber |
|----------------------|-------------|------------|-------------|-------------|
| UDT + GREEDY         | 0           | 0          | 0           | 0           |
| UDT + GREEDY + RULES | 0           | 0          | 0           | 0           |
| UDT + CLOT           | 0           | 0          | 0           | 0           |
| UDT + CLOT + RULES   | 0           | 0          | 0           | 0           |
| HD + GREEDY          | 0           | 0          | 0           | 0           |
| HD + GREEDY + RULES  | 0           | 0          | 0           | 0           |
| HD + CLOT            | 0           | 0          | 0           | 0           |
| HD + CLOT + RULES    | 0           | 0          | 0           | 0           |

# Packer UPX malware rules vs. UPX goodwill

| Algorithm            | rule:Cerber | rule:Locky                                                                    | rule:Upatre | rule:Zerber |
|----------------------|-------------|-------------------------------------------------------------------------------|-------------|-------------|
| UDT + GREEDY         | 0           | 0                                                                             | 0           | 0           |
| UDT + GREEDY + RULES | 0           | 0                                                                             | 0           | 0           |
| UDT + CLOT           | 0           | YaYaGen rules for UPX packed malware <b>do not detect</b> UPX packed goodwill |             | 0           |
| UDT + CLOT + RULES   | 0           |                                                                               |             | 0           |
| HD + GREEDY          | 0           | 0                                                                             | 0           | 0           |
| HD + GREEDY + RULES  | 0           | 0                                                                             | 0           | 0           |
| HD + CLOT            | 0           | 0                                                                             | 0           | 0           |
| HD + CLOT + RULES    | 0           | 0                                                                             | 0           | 0           |

# Signatures stats

| FAMILY     | SIZE | ALGORITHM            | # RULES | # LITERALS (AVG) | Time       |
|------------|------|----------------------|---------|------------------|------------|
| Fareit     | 14   | UDT + GREEDY         | 5       | 594              | ~ 30s      |
| Zerber     | 329  | UDT + CLOT + RULES   | 36      | 163              | ~5 minutes |
|            |      | HD + CLOT + RULES    | 86      | 187              | ~5 minutes |
| Teslacrypt | 2478 | UDT + GREEDY + RULES | 493     | 381              | ~3-4 hours |
|            |      | HD + GREEDY+ RULES   | 850     | 336              | ~3-4 hours |

# Signatures stats

| FAMILY     | SIZE | ALGORITHM            | # RULES | # LITERALS (AVG) | Time       |
|------------|------|----------------------|---------|------------------|------------|
| Fareit     | 14   | UDT + GREEDY         | 5       |                  | ~5 minutes |
| Zerber     | 329  | UDT + CLOT + RULES   | 36      |                  | ~5 minutes |
|            |      | HD + CLOT + RULES    | 86      | 187              | ~5 minutes |
| Teslacrypt | 2478 | UDT + GREEDY + RULES | 493     | 381              | ~3-4 hours |
|            |      | HD + GREEDY+ RULES   | 850     | 336              | ~3-4 hours |

On average, the UDT approach produces one cluster each 5 applications

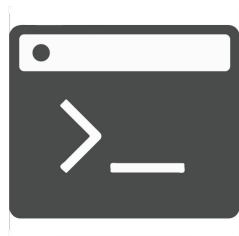
# Signatures stats

| FAMILY     | SIZE | ALGORITHM            | # RULES | # LITERALS (AVG) | Time       |
|------------|------|----------------------|---------|------------------|------------|
| Fareit     | 14   | UDT + GREEDY         | 5       | 594              | ~ 30s      |
| Zerber     | 329  | UDT + CLOT + RULES   | 36      |                  | utes       |
|            |      | HD + CLOT + RULES    | 86      |                  | utes       |
| Teslacrypt | 2478 | UDT + GREEDY + RULES | 493     | 381              | ~3-4 hours |
|            |      | HD + GREEDY+ RULES   | 850     | 336              | ~3-4 hours |

On average, HDBSCAN finds clusters of 5 points, but ~20% are outliers

# Demo

# \$ ./ydg.py



Generate a signature from a set of PE files

```
./ydg.py -dir my_samples -a greedy
```

Include existing ruleset

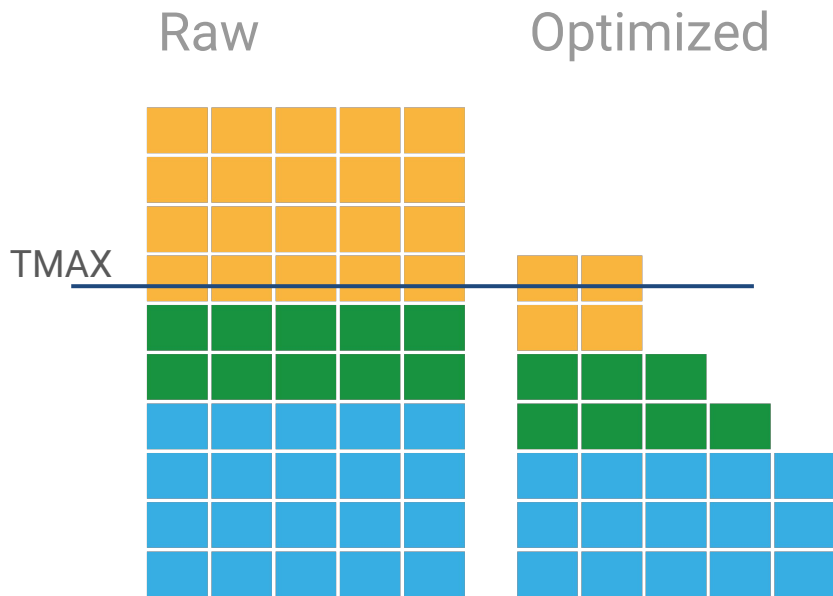
```
./ydg.py -dir my_samples -a greedy -yr yara_rules
```

# Limitations

The combination of files in input may affect the effectiveness of a rule

- \* Python pefile vs YARA PE module
- \* The rule evaluation is oversimplified (currently the sum of features)
- \* A rule can be overspecific (too many features).
- \* technical issue

# Optimization



Rules could be **over-specific**

In the Android YaYaGen version, two optimizers were deployed

We need to study which combinations of attributes create a better rule

# Conclusions

We **did a survey** of the automatic signature generation problem

The work was motivated by the interest from the **research community**

We **developed** YaYaGenPE to improve existing approaches



**POLITECNICO  
DI TORINO**

**HISPASEC**

Thank you

# XII STIC CCN-CERT CONFERENCES

Cybersecurity,  
towards effective response and deterrence



## Email

- [info@ccn-cert.cni.es](mailto:info@ccn-cert.cni.es)
- [ccn@cni.es](mailto:ccn@cni.es)
- [organismo.certificacion@cni.es](mailto:organismo.certificacion@cni.es)

## Websites

- [www.ccn.cni.es](http://www.ccn.cni.es)
- [www.ccn-cert.cni.es](http://www.ccn-cert.cni.es)
- [oc.ccn.cni.es](http://oc.ccn.cni.es)

## Follow us on

