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

Andrea Marcelli (@_S0nn1_) andrea.marcelli@polito.it
Who I am

**PhD Student** at Politecnico di Torino  
ML, semi-supervised modeling, optimization problem

**Security Researcher** at Hispasec Sistemas  
developing AI algorithms for large-scale malware detection

Twitter  @_S0nn1_  
Email  andrea.marcelli@polito.it  
Web  jimmy-sonny.github.io
Agenda

The signature generation problem

The algorithm

Introducing YaYaGen

Demo
The signature generation problem
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 the most of the existing tools and researches focus on

Semantic signatures provides an abstraction of the program behavior

Our target is Android malware, although the approach is generic

This is not just another ML classifier.
Work motivations

Reduce the malware exposure time

Automate a repetitive task
20k - 50k SUBMISSIONS EVERY DAY

100% recall and high precision requirements

Save a considerable amount of time and resources
“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

```plaintext
rule YaYaSyringe {

    meta:
        author = "DEF CON 26"

    strings:
        $a = "text here"
        $b = { E2 34 A1 C8 23 FB }

    condition:
        $a and $b
        and androguard.filter("action.BATTERYCHECK")
        and androguard.number_of_services == 3

    ...
}
```
The algorithm
The detection workflow

APK submissions → Unsupervised malware families discovery → cluster 1, cluster 2, ..., cluster N → Automatic signature generator
The attributes

Each block is an attribute extracted through the analysis

*Androguard, *Droidbox, Cockoo are used

The quality of the analysis affects the signature generation process.

* They require a custom YARA module
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.
Normal form

\((l_1 \land l_2 \land l_3) \lor (l_4 \land l_5)\)

clause

literal
Signature anatomy

Each signature can be expressed in DNF:

$$S = \bigvee_{i=0}^{n} c_i \quad c_i = \bigwedge_{j=0}^{m(i)} l_{i,j}$$

Each clause can be weighed:

$$w(c_i) = \sum_{j=0}^{m(i)} w(l_{i,j})$$

The weight of a signature is the lowest among its clauses:

$$w(S) = \min_{\forall i} w(c_i)$$
Generality vs specificity

A **weighting system** evaluates the rules.

- **The higher the weight, the less FP**
- **Possibly more FN**

- **The lower the weight, the more FP**
- **Possibly less FN**

![Diagram](image)
Weighting process

Assign correct weights is the key to guarantee high quality results.

The value of **TMIN** and **TMAX** is strongly related to the choice of the weights.

The process is designed to be automated.
Optimization

Experimental results show that automatically generated rules could be over-specific.

Two approaches:

- **Basic optimizer** (< 1 min), a greedy optimizer
- **Evo optimizer** (~ 5 min), based on EA. Encodes the human expert knowledge
Introducing YaYaGen
Yet another YARA rule Generator

*YaYa is grandma in ES

From

a set of application analysis reports

to

a set of YARA rules
Yet another YARA rule Generator

*YaYa is grandma in ES

2 algorithms (*clot*, *greedy*), 2 optimizers (*basic*, *evo*) and some heuristics

Includes a YARA rule parser for attributes weight optimization

Supports FP exclusion from rule generation

Written in Python 3

Ready for *Koodous* (collaborative platform for Android malware research)

* https://koodous.com/
Fork me on GitHub

https://github.com/jimmy-sonny/YaYaGen
Demo
Results
## Results

<table>
<thead>
<tr>
<th>Rule Name</th>
<th>Original</th>
<th>YaYaGen</th>
<th>Improvement</th>
</tr>
</thead>
<tbody>
<tr>
<td>SMSSENDER</td>
<td>539</td>
<td>1,004</td>
<td>+86.3%</td>
</tr>
<tr>
<td>SYRINGE</td>
<td>220</td>
<td>315</td>
<td>+43.2%</td>
</tr>
<tr>
<td>HUMMINGBAD2</td>
<td>136</td>
<td>257</td>
<td>+89.0%</td>
</tr>
<tr>
<td>MARCHER2</td>
<td>559</td>
<td>652</td>
<td>+16.6%</td>
</tr>
<tr>
<td>SMSREG</td>
<td>159</td>
<td>172</td>
<td>+8.2%</td>
</tr>
<tr>
<td>VOLCMANDROPPER</td>
<td>186</td>
<td>430</td>
<td>+131.2%</td>
</tr>
</tbody>
</table>
Takeaways

Automatically generated rules perform better than manual written ones.

Existing rule sets are used to trim the algorithm.

The time required to generate a rule from 100 apps is less than 5 minutes.

However, there is still room for improvement.

Exploit the expert knowledge.

Ready for real-world applications.
Ongoing work

YaYaGenPE
Questions?
Thank you
References


https://github.com/Xen0ph0n/YaraGenerator

https://github.com/Neo23x0/yarGen

https://github.com/AlienVault-OTX/yabin

https://www.talosintelligence.com/bass