Machine Learning and other Computational-Intelligence Techniques for Security Applications

PhD Candidate: Andrea Marcelli

Supervisor: Prof. Giovanni Squillero
Agenda (40 mins)

Introduction and background

A study of Android banking trojans

Clustering of a 1M applications dataset

Automatic signature generation and optimization

Experimental results

Conclusions
Acknowledgement

- The Hispasec team
- Ivan Zelinka (COST-ACTION CA15140)
- Dario Lombardo (Telecom Italia)
- A. Sánchez (Dekra)
- F. López, F. Díaz, D. García, K. Hiramoto, V. Alvarez, B. Quintero (VT, Google)
- @emdel (Talos)
INTRODUCTION AND BACKGROUND
Introduction

Automation in the AV industry is essential:
  to provide fast coverage
  to scale (> 1M new binaries every day received from an AV company)

Some previous researches oversimplify the problems:
  no ground truth
  no correct labelling
  packing / obfuscation
  same campaign, multiple stages

Automation should aim to assist researchers, replacing them is currently not possible.

I studied some key problems of the AV industry and provided real-world solutions
I spent about 4 months in an AV company supported by COST ACTION CA15140.
About Android
Android apps are APKs

An APK (i.e., the android package) contains the following folders and files:

- META-INF
- res
- AndroidManifest.xml
- classes.dex
- resources.arsc
Android components

Activities
They dictate the UI and handle the user interaction to the smartphone screen

Services
They handle background processing associated with an application

Broadcast Receivers
They handle communication between Android OS and applications

Content Providers
They handle data and database management issues.

An Android app requests permissions to access sensible resources. Intents are used as high level IPC.
How many applications?

Koodous

~50M apps
~14M malware
+20k-50k every day

GooglePlay

~2.7M apps in June 2019

https://koodous.com/

https://www.statista.com
Application analysis

Static analysis
- Does not require to execute the application
- Fast, but vulnerable to obfuscation
- e.g., Analysis of manifest, Java decompiled code, and strings (Androguard).

Dynamic analysis
- Requires to run the application in a sandbox, but it can be detected
- Expensive and it follows one execution path only
- The analysis is precise
- e.g., Frida-based emulators, Xposed (CuckooDroid).

Both of them are required.
A STUDY OF ANDROID BANKING TROJANS

Timeline

2011 ZITMO
SMS stealer, targeting major Russian banks.
The first mobile banker for Android.

2012 CARBERP
SMS spoofing and screen locking capabilities.

2013 HESPERBOT
Introducing the overlay attack.

2014 GMBOT
The source code of Bankbot was leaked.

2016 BANKBOT
SMS stealer, targeting major Russian banks.

2017
Mazarbot, RedAlert, Faketoken.
Modus operandi

Infection

Persistence
  Anti-analysis techniques
  Privilege escalation

Communication
  C&C

Attack
  The overlay attack
  SMS spoofing
  The social engineering role
Detection

Visual Analysis
Hundreds of Android applications contacting tens of different domains, which resolve to the same address.

Uncovering new variants is possible thanks to the graph analysis.

Unpublished algorithm for node ranking based on known detection information.
CLUSTERING A 1M APPLICATIONS DATASET

Requirements

No a priori information about the number of clusters and their composition

Real data contains outliers

Process 1M dataset:
   About 1 day of Windows binaries
   About 1 month of new Android applications.
Density based

**HDBSCAN**
Enhanced version of DBSCAN.

In low dim space has a complexity of $O(n \times \log(n))$ and has a space requirement of $O(n)$.

**Edit distance** gives the best results

$\text{min\_clust\_size} = 2, \text{min\_points} = 2$
Feature selection

35 statistical properties from the static and dynamic analysis of an application

Androguard:
  from the manifest
  from the code analysis

Sandbox:
  I/O file system
  Networking

<table>
<thead>
<tr>
<th>Analysis method</th>
<th>Software</th>
<th>Statistical property</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parsing Manifest file</td>
<td>Androguard</td>
<td>Filters, Activities, Receivers, Services, Permissions</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Accounts, Advertisement, Browser history, Camera, Crypto functions, Dynamic broadcast receiver, Installed applications, Run binary, MCC, ICCID, IMEI, IMSI, SMS, MMS, Phone call, Phone number, Sensor, Serial number, Socket, SSL</td>
</tr>
<tr>
<td>Static All from APK</td>
<td>Androguard</td>
<td>Files written, Crypto usage, Files read, Send SMS, Send network, Recv Network</td>
</tr>
<tr>
<td>Dynamically</td>
<td>DroidBox</td>
<td>HTTP request, Hosts, Domains, DNS</td>
</tr>
<tr>
<td>Dynamically</td>
<td>CuckooDroid</td>
<td>HTTP request, Hosts, Domains, DNS</td>
</tr>
</tbody>
</table>
Iterative clustering

The dataset $D$ is divided into $m$ partitions.

Each partition $d_i$ is clustered individually.

$O$ is the union of the outliers $o_i$ found in each iteration.

$$m = \left\lceil \frac{|D|}{N} \right\rceil$$

$$O = \bigcup_{i=0}^{m-1} o_i$$
Family type

The original dataset is divided in three portions.

Each cluster is one of the 7 types.

This allows to automatically convict new applications and prioritize the work.

It can reduce 1M apps to few thousands interesting samples.
AUTOMATIC SIGNATURE GENERATION AND OPTIMIZATION


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

In this context, malware “signatures” and “rules” have the same meaning.
“YARA is to files what Snort is to network traffic” Victor M. Alvarez

Designed to be fast.

One of the two most-used languages to write malware signatures

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

**Semantic signatures** are defined through custom modules.
An example of YARA signature

```plaintext
rule example {
    meta:
        author = "Andrea Marcelli"

    strings:
        $a = "IEncrypt.dll"

    condition:
        $a and
        pe.image_base == 0x708640768 and
        pe.resources[6].language == 1030 and
        pe.resources[36].type == 10 and
        pe.resources[37].id == 104 and
        pe.imports("user32.dll","GetCursorPos")
}
```
Requirements

The process to generate a signature should be fast (e.g., ~ 5 min for 100 samples)

The algorithm should scale up to few thousands of input samples

Limit FPs

Avoiding FPs should not be related to number of samples input

The signature should catch other variants too.
The framework workflow

Cluster of APKs or PE files

- Features extraction
- Clustering (*optional)
- Signature generator

YaYaGenPE

Existing YARA rules

YARA rules
INPUT IS GENERIC. OUTPUT IS A MALWARE SIGNATURE

The input is a set of Android applications (or Windows files).
It could be a set of malware or goodware, a tight or a loose cluster.

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

Ruleset are converted in YARA, can be directly 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

* For Android, Koodous static and dynamic analysis system provides the features
* For Windows, a custom YARA version extract all the supported features

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

* [https://github.com/erocarrera/pefile](https://github.com/erocarrera/pefile)
**FEATURES ARE GENERIC**

In summary, it’s an approximation algorithm to solve an 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 valid signature.

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 (HDBSCAN) and unsupervised decision tree

Each cluster is the input of the signature generation algorithm.
UDT: Unsupervised decision tree

Each cluster is splitted into two new ones basing on the value of a single feature.

The best best splitting feature is the one that maximise the distance among centroids. Cluster centroids are approximated, and Jaccard distances is 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.
UDT clustering

pe.imports("user32.dll","ReleaseDC")

pe.os_version.minor == 0

pe.resources[0].length==308

pe.number_of_sections == 4

rule:shimrat_0

pe.imports("user32.dll","LoadImageA")

cluster 0

cluster 1

cluster 2

cluster 3

cluster 4

cluster 5

cluster 6
UDT clustering
3. The signature generation

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.
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.
Signature anatomy

Each signature can be expressed in DNF

$$S = \bigvee_{i=0}^{n} c_i$$

Each clause can be weighed

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

The weight of a signature is the lowest among its clauses

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

Weights are automatically assigned using the Simplex Method.
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
FPs

Phase I:
generate a new Yara rule

Phase II:
check false positives

Phase III:
generate a new Yara rule $Y^* = (c_a \lor c_b)$

clause $c_a = (r \land r_a)$

clause $c_b = (r \land r_b)$
Signature optimization
Rules could be over-specific

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

We introduced two optimizers: hill-climber- and EA-based.
Evo - optimizer

Estimation of Distribution Algorithm (EDA)*

Solution representation:
- the individual is a YARA rule
- optimize the attribute subset of the rule

Development of a two **fitness functions**:
- lexicographic fitness
- heuristic fitness

* Loosely inspired by Selfish Gene theory
Lexicographic fitness

Individual comparison based on:
- **Number of reports** matched by the YARA Rule (to maximize)
- **Score** of the YARA rule (to minimize – still greater than Tmin)
- **Number of attributes** inside the YARA rule (to minimize)

Some good results:
- ex1: rules with 3 attributes of weight 150
- ex2: rules with 4 attributes of weight 100 (e.g. 4 URL)

Results improved in respect to Basic Optimizer
but some rules are still unacceptable for human experts
Heuristic fitness

Introduction of heuristic comparisons:
- a better rule has more categories of attributes
- e.g., if a rule contains only URL is worst than the other one

The comparisons are not “hard”, transitive property is lost

EDA (Estimation of distribution)

Soccer like scoring system for the archive
Implementation
YaYaGenPE is an extension of the original YaYaGen framework

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

Include new YARA python bindings to directly extract the features.

Supports FP exclusion from rule generation

Optimization using the Selfish Gene Extended (SGX) library

Written in Python 3.

GitHub repository: [https://github.com/jimmy-sonny/YaYaGen](https://github.com/jimmy-sonny/YaYaGen)
EXPERIMENTAL RESULTS
Clustering results
Iterative clustering evaluation

<table>
<thead>
<tr>
<th>$N$</th>
<th>Hom</th>
<th>Comp.</th>
<th>Hom</th>
<th>Comp.</th>
</tr>
</thead>
<tbody>
<tr>
<td>50k</td>
<td>0.96</td>
<td>0.36</td>
<td>0.85</td>
<td>0.49</td>
</tr>
<tr>
<td>100k</td>
<td>0.96</td>
<td>0.35</td>
<td>0.85</td>
<td>0.49</td>
</tr>
<tr>
<td>200k</td>
<td>0.96</td>
<td>0.35</td>
<td>0.85</td>
<td>0.50</td>
</tr>
<tr>
<td>non-iterative</td>
<td>0.92</td>
<td>0.36</td>
<td>0.78</td>
<td>0.50</td>
</tr>
</tbody>
</table>

The non iterative version has a lower Hom. value.
Automatic labelling

Num. of total applications (blue)
Num. of labelled apps (orange)
Family 1..6

$N = 100k$
1M dataset
Automatic signature generation results
## Comparison state of the art

<table>
<thead>
<tr>
<th></th>
<th>YaraGenerator</th>
<th>YarGen</th>
<th>YaBin</th>
<th>BASS</th>
<th>YaYaGen</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Based on</strong></td>
<td>Strings</td>
<td>Strings</td>
<td>Code</td>
<td>Code</td>
<td>PE header + rules</td>
</tr>
<tr>
<td><strong>Algorithm</strong></td>
<td>Common strings</td>
<td>Whitelist strings</td>
<td>Whitelist funcs</td>
<td>BinDiff + LCS</td>
<td>CLUSTERING + SET COV. + EA</td>
</tr>
<tr>
<td><strong>Guaranteed input coverage</strong></td>
<td>NO</td>
<td>NO</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
</tr>
<tr>
<td><strong>Packer resistance</strong></td>
<td>NO</td>
<td>NO</td>
<td>GOOD</td>
<td>GOOD</td>
<td>GOOD</td>
</tr>
<tr>
<td><strong>Clustering</strong></td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>YES</td>
<td>YES</td>
</tr>
<tr>
<td><strong>Scalability</strong></td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>NO</td>
<td>YES</td>
</tr>
</tbody>
</table>

51
Evaluation criteria

1. **True positives**: the number of malware samples from a specific family covered by the rule

2. **False positives**: the number of goodware samples classified as malicious

3. **Dataset coverage**: the total number of malware samples from the dataset under study that have been covered by the rule

4. **Packer resistance**: the ability of the rule of matching malware samples, even though malware has been packed.
### Comparison input 47 samples

<table>
<thead>
<tr>
<th>TOOL</th>
<th>ALGORITHM</th>
<th># RULES</th>
<th>FPs</th>
<th>TPs</th>
</tr>
</thead>
<tbody>
<tr>
<td>YaYaGenPE</td>
<td>UDT + GREEDY</td>
<td>29</td>
<td>0</td>
<td>76</td>
</tr>
<tr>
<td></td>
<td>UDT + GREEDY + RULES</td>
<td>31</td>
<td>0</td>
<td>75</td>
</tr>
<tr>
<td></td>
<td>HD + GREEDY + RULES</td>
<td>23</td>
<td>0</td>
<td>80</td>
</tr>
<tr>
<td>YarGen</td>
<td>RULES Z0</td>
<td>53</td>
<td>1</td>
<td>130</td>
</tr>
<tr>
<td></td>
<td>RULES Z0 + OPCODES</td>
<td>53</td>
<td>0</td>
<td>86</td>
</tr>
<tr>
<td>YaBin</td>
<td>Yara (-y)</td>
<td>36</td>
<td>0</td>
<td>76</td>
</tr>
<tr>
<td></td>
<td>YaraHunt (-yh)</td>
<td>36</td>
<td>10</td>
<td>194</td>
</tr>
</tbody>
</table>

**CRYPTOWALL FAMILY**

- **MALWARE** 6,881 samples
- **GOODWARE** 3,413 samples
## Comparison on input 533 samples

<table>
<thead>
<tr>
<th>TOOL</th>
<th>ALGORITHM</th>
<th># RULES</th>
<th>FPs</th>
<th>TPs</th>
</tr>
</thead>
<tbody>
<tr>
<td>YaYaGenPE</td>
<td>UDT + GREEDY</td>
<td>65</td>
<td>2</td>
<td>854</td>
</tr>
<tr>
<td></td>
<td>UDT + GREEDY + RULES</td>
<td>64</td>
<td>2</td>
<td>896</td>
</tr>
<tr>
<td></td>
<td>HD + GREEDY</td>
<td>137</td>
<td>0</td>
<td>768</td>
</tr>
<tr>
<td>YarGen</td>
<td>RULES Z0</td>
<td>328</td>
<td>7</td>
<td>705</td>
</tr>
<tr>
<td></td>
<td>RULES Z0 + OPCODES</td>
<td>321</td>
<td>4</td>
<td>687</td>
</tr>
<tr>
<td>YaBin</td>
<td>Yara (-y)</td>
<td>157</td>
<td>0</td>
<td>737</td>
</tr>
<tr>
<td></td>
<td>YaraHunt (-yh)</td>
<td>157</td>
<td>16</td>
<td>937</td>
</tr>
</tbody>
</table>

**CERBER FAMILY**
- **MALWARE** 6,881 samples
- **GOODWARE** 3,413 samples
Comparison input 2478 samples

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

TESLACRYPT FAMILY
MALWARE 6,881 samples
GOODWARE 3,413 samples
Retrohunt evaluate FPs and TPs

<table>
<thead>
<tr>
<th>Family</th>
<th>Algorithm</th>
<th>Input size</th>
<th>Total matches*</th>
<th>TP</th>
<th>FPs</th>
</tr>
</thead>
<tbody>
<tr>
<td>OlympicDestroyer</td>
<td>UDT + GREEDY + RULES</td>
<td>22</td>
<td>143</td>
<td>100%</td>
<td>0</td>
</tr>
<tr>
<td>Sagecrypt</td>
<td>HD + CLOT + RULES</td>
<td>47</td>
<td>136</td>
<td>100%</td>
<td>0</td>
</tr>
<tr>
<td>Crowti</td>
<td>UDT + GREEDY + RULES</td>
<td>75</td>
<td>66</td>
<td>100%</td>
<td>0</td>
</tr>
<tr>
<td>Scatter</td>
<td>UDT + GREEDY</td>
<td>12</td>
<td>57</td>
<td>86%</td>
<td>8</td>
</tr>
<tr>
<td>Scatter</td>
<td>UDT + GREEDY + RULES</td>
<td>12</td>
<td>35</td>
<td>88%</td>
<td>4</td>
</tr>
<tr>
<td>Shiz</td>
<td>UDT + CLOT + RULES</td>
<td>104</td>
<td>12</td>
<td>100%</td>
<td>0</td>
</tr>
</tbody>
</table>

* Using a dataset of ~100 TB
## Packer: UPX malware rules vs. UPX goodware

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>rule:Cerber</th>
<th>rule:Locky</th>
<th>rule:Upatre</th>
<th>rule:Zerber</th>
</tr>
</thead>
<tbody>
<tr>
<td>UDT + GREEDY</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>UDT + GREEDY + RULES</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>UDT + CLOT</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>UDT + CLOT + RULES</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>HD + GREEDY</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>HD + GREEDY + RULES</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>HD + CLOT</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>HD + CLOT + RULES</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
# Packer: UPX malware rules vs. UPX goodware

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>rule:Cerber</th>
<th>rule:Locky</th>
<th>rule:Upatre</th>
<th>rule:Zerber</th>
</tr>
</thead>
<tbody>
<tr>
<td>UDT + GREEDY</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>UDT + GREEDY + RULES</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>UDT + CLOT</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>UDT + CLOT + RULES</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>HD + GREEDY</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>HD + GREEDY + RULES</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>HD + CLOT</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>HD + CLOT + RULES</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

YaYaGen rules for UPX packed malware do not detect UPX packed goodware.
## Signatures stats

<table>
<thead>
<tr>
<th>FAMILY</th>
<th>SIZE</th>
<th>ALGORITHM</th>
<th># RULES</th>
<th># LITERALS (AVG)</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fareit</td>
<td>14</td>
<td>UDT + GREEDY</td>
<td>5</td>
<td>594</td>
<td>~ 30s</td>
</tr>
<tr>
<td>Zerber</td>
<td>329</td>
<td>UDT + CLOT + RULES</td>
<td>36</td>
<td>163</td>
<td>~5 minutes</td>
</tr>
<tr>
<td></td>
<td></td>
<td>HD + CLOT + RULES</td>
<td>86</td>
<td>187</td>
<td>~5 minutes</td>
</tr>
<tr>
<td>Teslacrypt</td>
<td>2478</td>
<td>UDT + GREEDY + RULES</td>
<td>493</td>
<td>381</td>
<td>~3-4 hours</td>
</tr>
<tr>
<td></td>
<td></td>
<td>HD + GREEDY+ RULES</td>
<td>850</td>
<td>336</td>
<td>~3-4 hours</td>
</tr>
</tbody>
</table>
## Signatures stats

<table>
<thead>
<tr>
<th>FAMILY</th>
<th>SIZE</th>
<th>ALGORITHM</th>
<th># RULES</th>
<th># LITERALS (AVG)</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fareit</td>
<td>14</td>
<td>UDT + GREEDY</td>
<td>5</td>
<td>594</td>
<td>~30s</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Zerber</td>
<td>329</td>
<td>UDT + CLOT + RULES</td>
<td>36</td>
<td>163</td>
<td>~5 minutes</td>
</tr>
<tr>
<td></td>
<td></td>
<td>HD + CLOT + RULES</td>
<td>86</td>
<td>187</td>
<td>~5 minutes</td>
</tr>
<tr>
<td>Teslacrypt</td>
<td>2478</td>
<td>UDT + GREEDY + RULES</td>
<td>493</td>
<td>381</td>
<td>~3-4 hours</td>
</tr>
<tr>
<td></td>
<td></td>
<td>HD + GREEDY+ RULES</td>
<td>850</td>
<td>336</td>
<td>~3-4 hours</td>
</tr>
</tbody>
</table>

On average, the UDT approach produces one cluster each 5 applications.
## Signatures stats

<table>
<thead>
<tr>
<th>FAMILY</th>
<th>SIZE</th>
<th>ALGORITHM</th>
<th># RULES</th>
<th># LITERALS (AVG)</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fareit</td>
<td>14</td>
<td>UDT + GREEDY</td>
<td>5</td>
<td>594</td>
<td>~ 30s</td>
</tr>
<tr>
<td></td>
<td></td>
<td>HD + CLOT + RULES</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Zerber</td>
<td>329</td>
<td>UDT + CLOT + RULES</td>
<td>36</td>
<td>163</td>
<td>~5 minutes</td>
</tr>
<tr>
<td></td>
<td></td>
<td>HD + CLOT + RULES</td>
<td>86</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Teslacrypt</td>
<td>2478</td>
<td>UDT + GREEDY + RULES</td>
<td>493</td>
<td>381</td>
<td>~3-4 hours</td>
</tr>
<tr>
<td></td>
<td></td>
<td>HD + GREEDY+ RULES</td>
<td>850</td>
<td>336</td>
<td></td>
</tr>
</tbody>
</table>

On average, HDBSCAN finds clusters of 5 points, but ~20% are outliers.
Signature optimization results
### Optimization results: AVG num of literals

<table>
<thead>
<tr>
<th>FAMILY</th>
<th>TPs</th>
<th>Non optimized</th>
<th>HC Optimizer</th>
<th>SGX Optimizer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cluster 10</td>
<td>4/4</td>
<td>260</td>
<td>9.00</td>
<td>11.50</td>
</tr>
<tr>
<td></td>
<td>4/4</td>
<td>260</td>
<td>9.20</td>
<td>12.60</td>
</tr>
<tr>
<td>Cluster 20</td>
<td>3/4</td>
<td>64</td>
<td>20.20</td>
<td>39.20</td>
</tr>
<tr>
<td></td>
<td>3/4</td>
<td>64</td>
<td>18.20</td>
<td>40.90</td>
</tr>
</tbody>
</table>

SGX produces rules with more literals than HC.
Optimization results: AVG rule score

<table>
<thead>
<tr>
<th>FAMILY</th>
<th>TPs</th>
<th>Non optimized</th>
<th>HC Optimizer</th>
<th>SGX Optimizer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cluster 10</td>
<td>4/4</td>
<td>17614</td>
<td>625.00</td>
<td>401.70</td>
</tr>
<tr>
<td></td>
<td>4/4</td>
<td>17614</td>
<td>597.00</td>
<td>401.80</td>
</tr>
<tr>
<td>Cluster 20</td>
<td>3/4</td>
<td>2073</td>
<td>612.30</td>
<td>555.00</td>
</tr>
<tr>
<td></td>
<td>3/4</td>
<td>2073</td>
<td>620.50</td>
<td>583.90</td>
</tr>
</tbody>
</table>

SGX produces rules with lower scores.
CONCLUSIONS
Conclusions

Studied the Android banking trojans ecosystem

One of the first researchers to study large-scale detection systems in Android

Proposed a new signature generation algorithm for Android and Windows binaries

Implemented two new tools and developed a new extension to YARA to directly extract features from custom modules.
Looking for the perfect signature: an automatic YARA rules generation algorithm in the AI-era
BSidesLV (7-8 Aug 2018)

Looking for the perfect signature: an automatic YARA rules generation algorithm in the AI-era
DEF CON 26 (9-12 Aug 2018)

Inteligencia colectiva con Koodous y YayaGen
Tassi 2018 (13 Sep 2018) - Criptored and BBVA Next Technologies

XII CCN-CERT STIC Conference (12-13 Dec 2018)
Spanish National Government CERT
Thank you


https://github.com/Xen0ph0n/YaraGenerator

https://github.com/Neo23x0/yarGen

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

https://www.talosintelligence.com/bass