POLITECNICO
DI TORINO

PhD in Computer and Control Engineering

XXXI cycle

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

HISPASEC

cost
EUROPEAN COOPERATION
IN SCIENCE & TECHNOLOGY

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

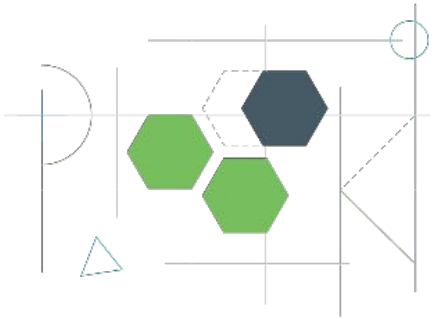
Anatomy of an APK

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

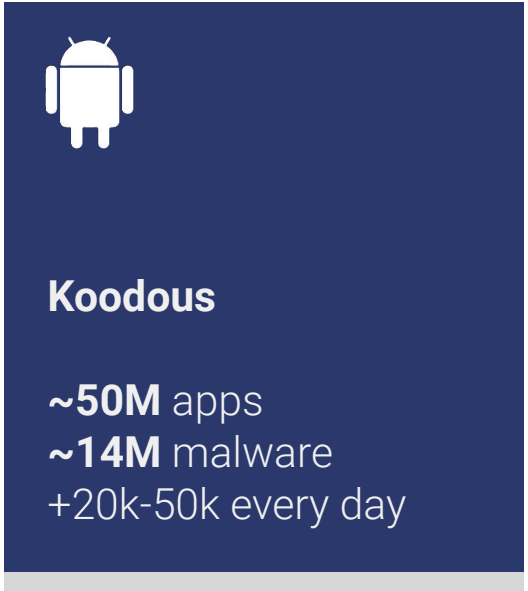
Services
They handle background processing associated with an application

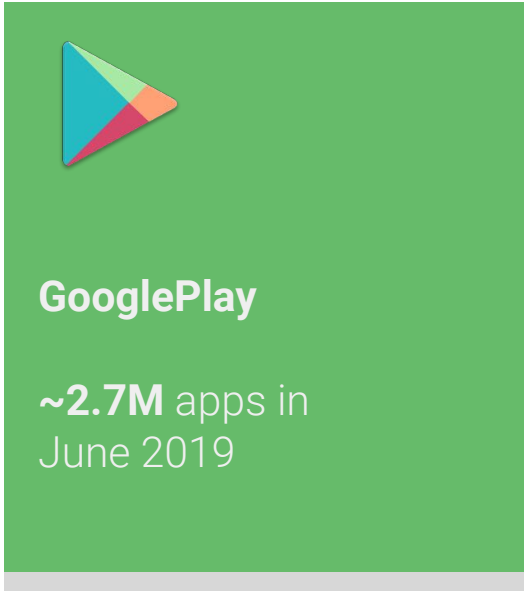
Broadcast Receivers
They handle communication between Android OS and applications

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?





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

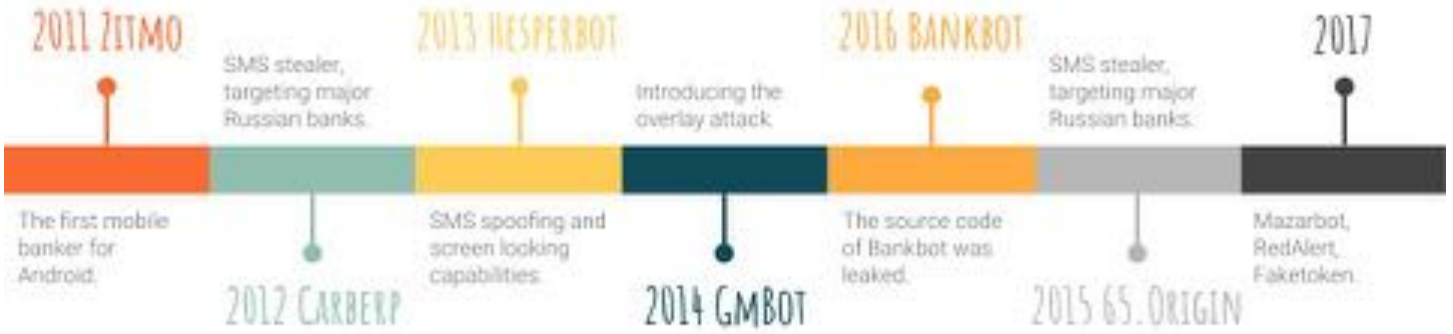
Requires to run the application in a sandbox, but it can be detected
Expensive and it follows one execution path only
The analysis is precise
e.g., Frida-based emulators, Xposed (CuckooDroid).

Both of them are required.

A STUDY OF ANDROID BANKING TROJANS

Andrea Atzeni, Fernando Díaz, Francisco López, Andrea Marcelli, Antonio Sánchez, and Giovanni Squillero. The rise of android banking trojans. IEEE Potentials, 2019.

Timeline


2011 ZITMO
The first mobile banker for Android.
2012 CARBERP
SMS stealer, targeting major Russian banks.
2013 HESPERBOT
SMS spoofing and screen locking capabilities.
2014 GMBOT
Introducing the overlay attack.
2016 BANKBOT
The source code of Bankbot was leaked.
2015 65.ORIGIN
SMS stealer, targeting major Russian banks.
2017
Mazarbot, RedAlert, Faketoken.

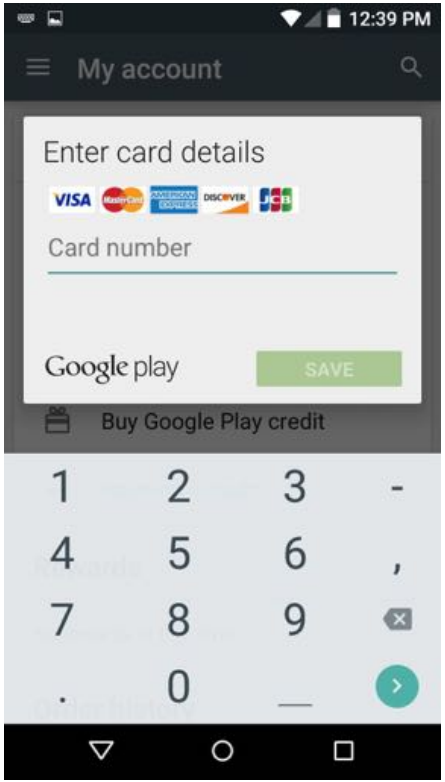
Modus operandi

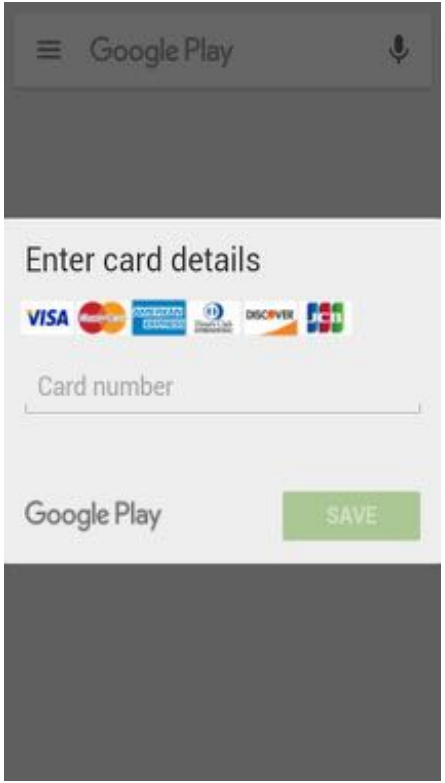
Infection

Persistence
- Anti-analysis techniques
- Privilege escalation

Communication
- C&C

Attack
- The overlay attack
- SMS spoofing
- The social engineering role



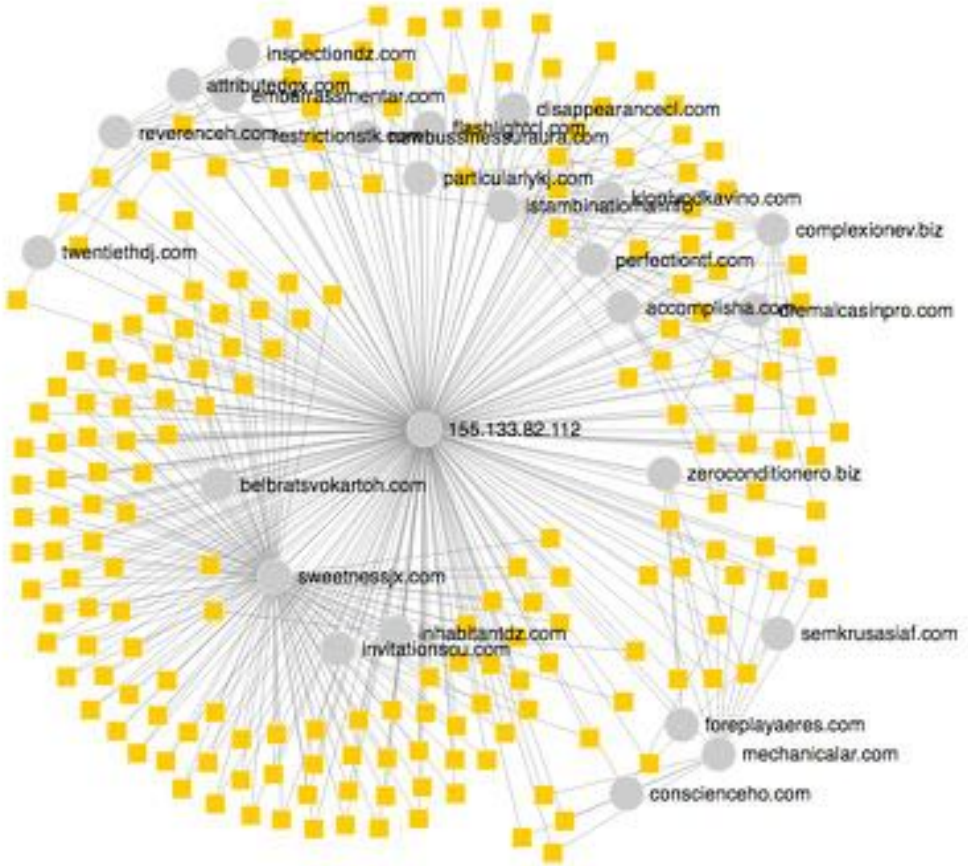


Detection

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

Andrea Atzeni, Fernando Díaz, ***Andrea Marcelli****, Antonio Sánchez, Giovanni Squillero, and Alberto Tonda. Countering android malware: A scalable semi-supervised approach for family-signature generation. IEEE Access*

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.

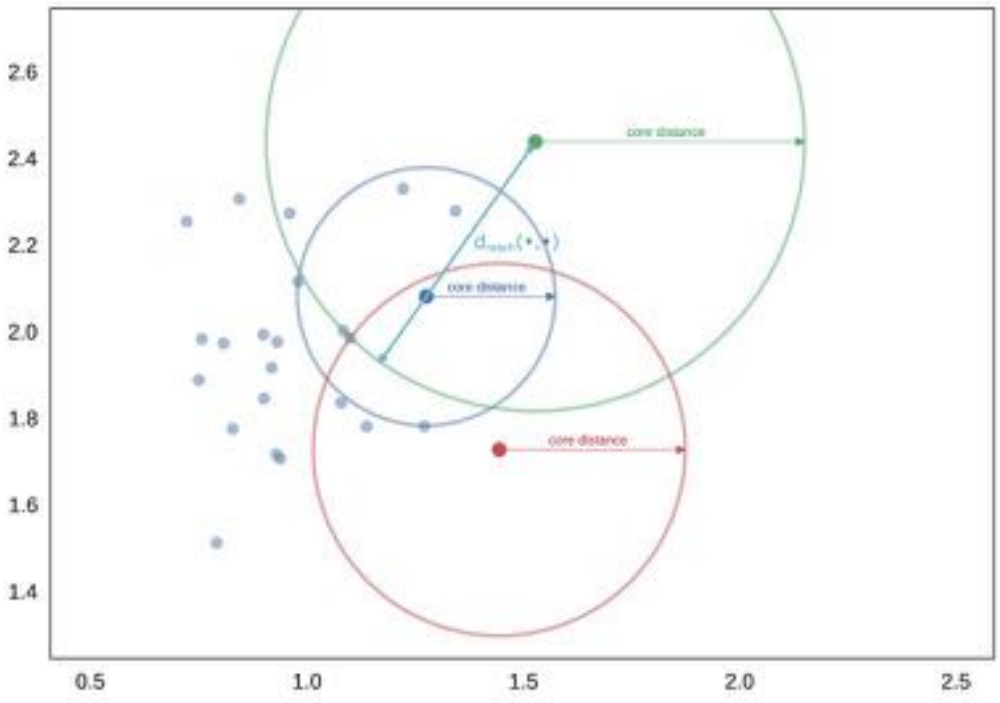
Density based

HDBSCAN
Enhanced version of DBSCAN.

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

Edit distance gives the best results

min_clust_size = 2, 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

Analysis method	Sofware	Statistical property
Parsing Manifest file	Androguard	Filters Activities Receivers Services Permissions
Statically from APK	Androguard	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
Dynamically	DroidBox	Files written Crypto usage Files read Send SMS Send network Recv Network
	CuckooDroid	HTTP request Hosts Domains DNS

Iterative clustering

The dataset **D** is divided into **m** partition

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

Each partition $\mathbf{d_i}$ is clustered individually

O is the union of the outliers $\mathbf{o_i}$ found in each iteration

$$\mathbf{O} = \bigcup_{i=0}^{m-1} \mathbf{o}_i$$

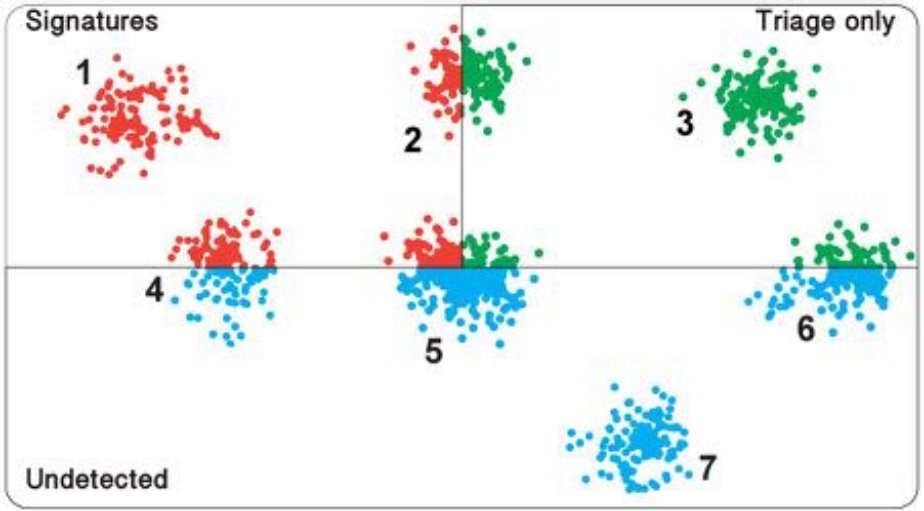
Family type

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

Andrea Atzeni, Fernando Díaz, ***Andrea Marcelli****, Antonio Sánchez, Giovanni Squillero, and Alberto Tonda. Countering android malware: A scalable semi-supervised approach for family-signature generation. IEEE Access*

Eliana Giovannitti, Luca Mannella, ***Andrea Marcelli****, and Giovanni Squillero. Evolutionary antivirus signature optimization. In 2019 IEEE Congress on Evolutionary Computation (CEC), 2019*

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.

About YARA

“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

```
rule example {

   meta:
      author = "Andrea Marcelli"

   strings:
      $a = "IEncrypt.dll"

   condition:
       $a and
       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")
}
```

Requirements

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

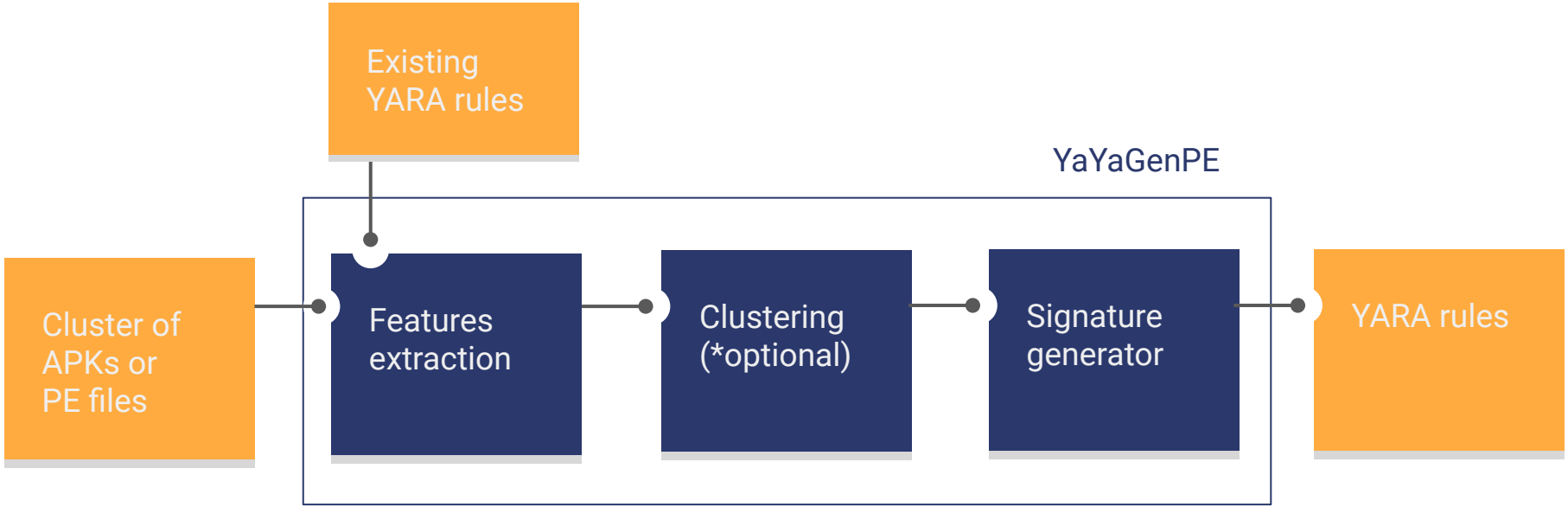
The algorithm should scale up to **few thousands** of input samples

Limit FPs

Avoiding FPs should not be related to number of samples input

The signature should catch other variants too.

The framework workflow



Key point

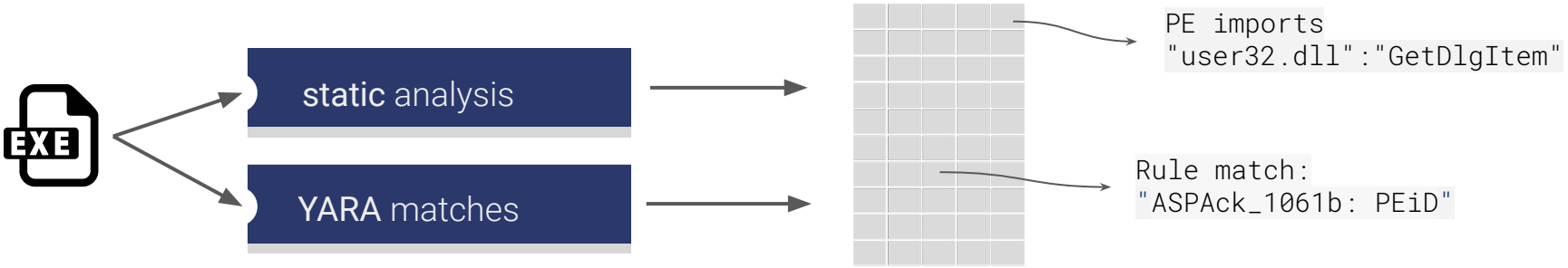
INPUT IS GENERIC. OUTPUT IS A MALWARE SIGNATURE

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

Key point

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

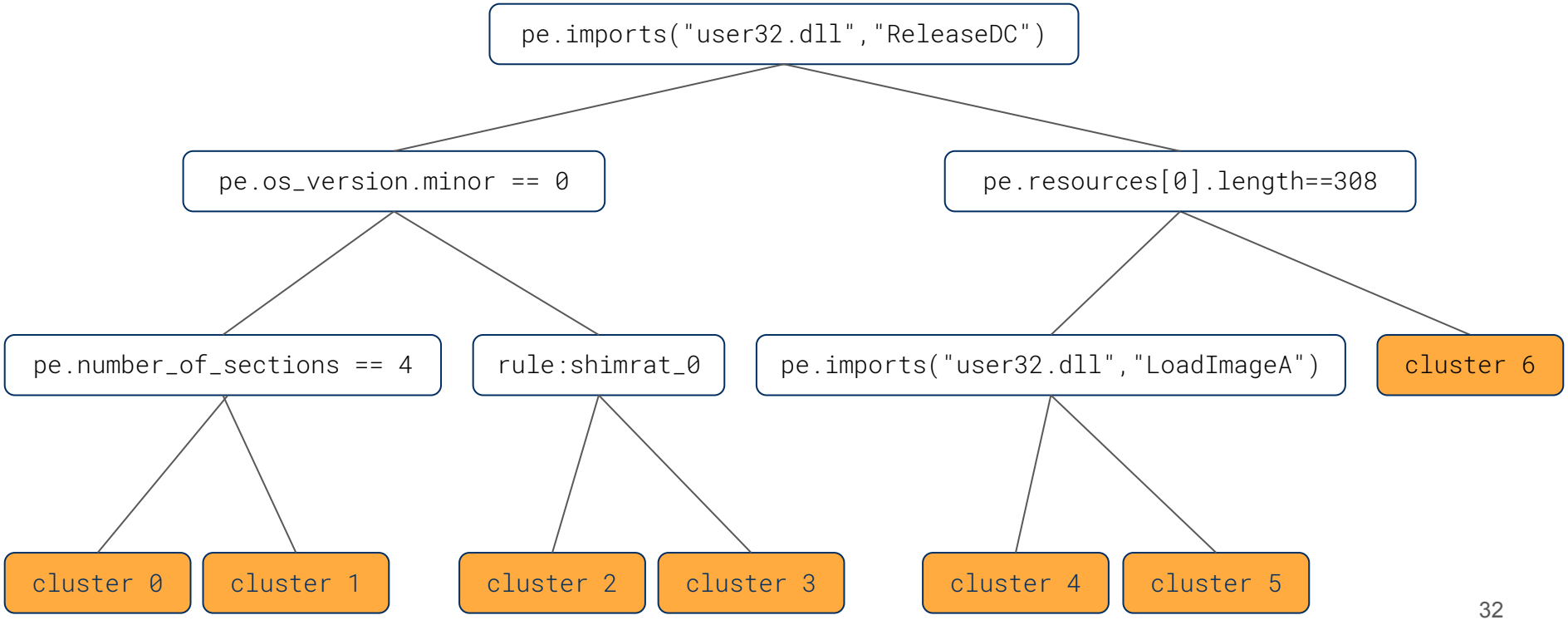
Each **cluster is splitted** into two new ones basing on the value of a single feature

The best **best splitting feature** is the one that maximise the distance among centroids Cluster centroids are approximated, and **Jaccard** distances is used

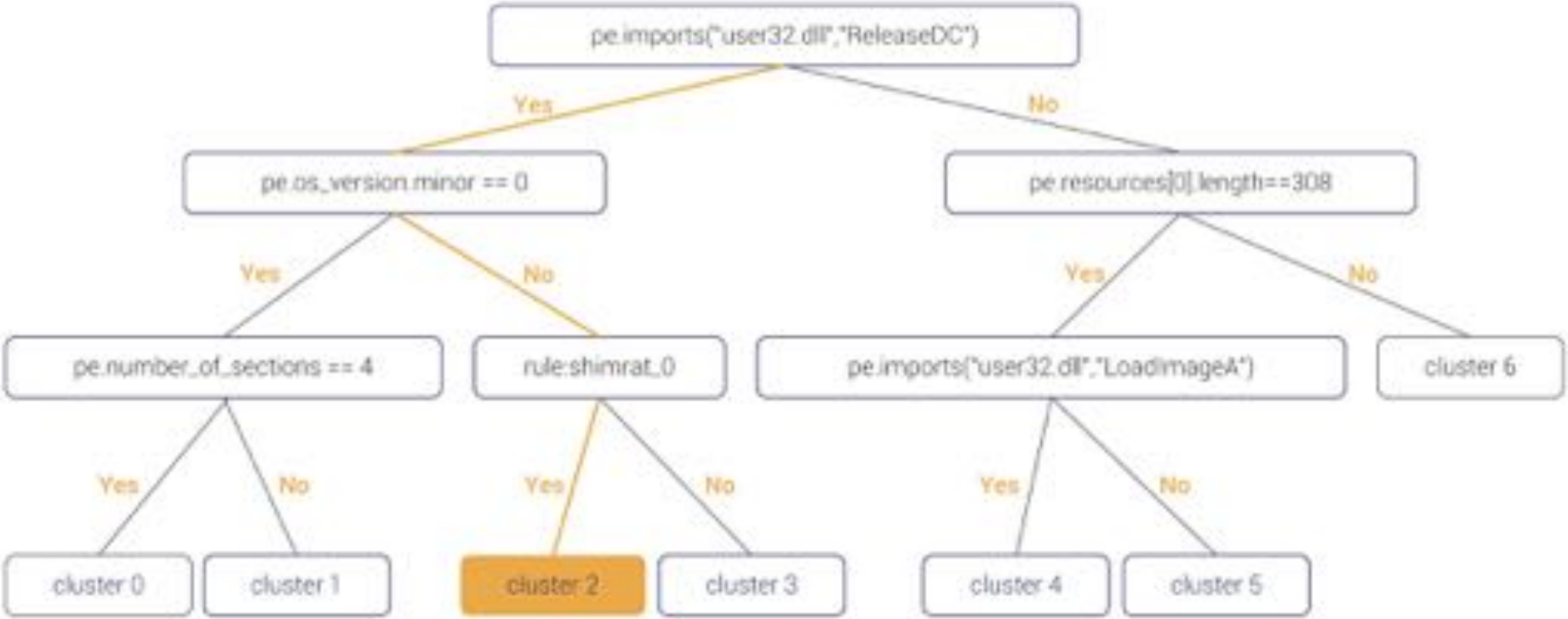
The stopping criterion is the distance between centroids (experimentally set)

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



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.

DNF

$$(l_1 \wedge l_2 \wedge l_3) \vee (l_4 \wedge l_5)$$

clause — 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**.

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

Weights are automatically assigned using the **Simplex Method**.

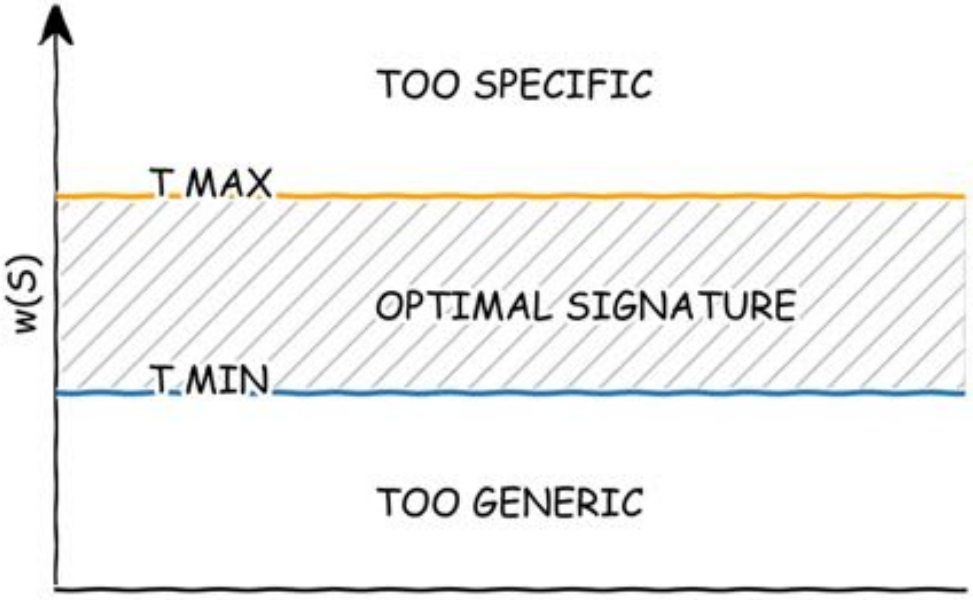
$$w(S) = \min_{\forall i} w(c_i)$$

Generality vs specificity

A **weighting system** evaluates the rules

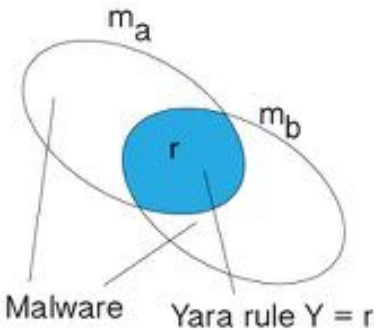
The higher the weight, the less FPs
Possibly more FNs

The lower the weight, the more FPs
Possibly less FNs

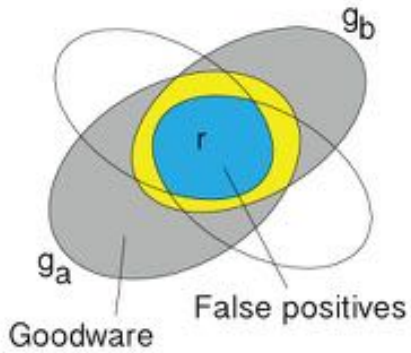


FPs

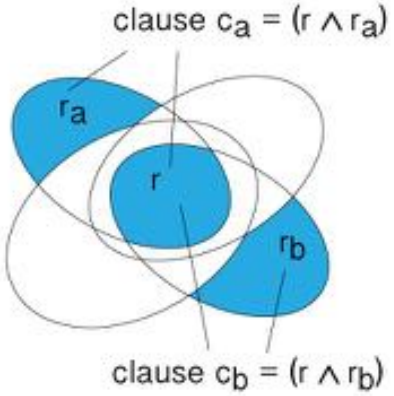
Phase I:
generate a new Yara rule



Phase II:
check false positives

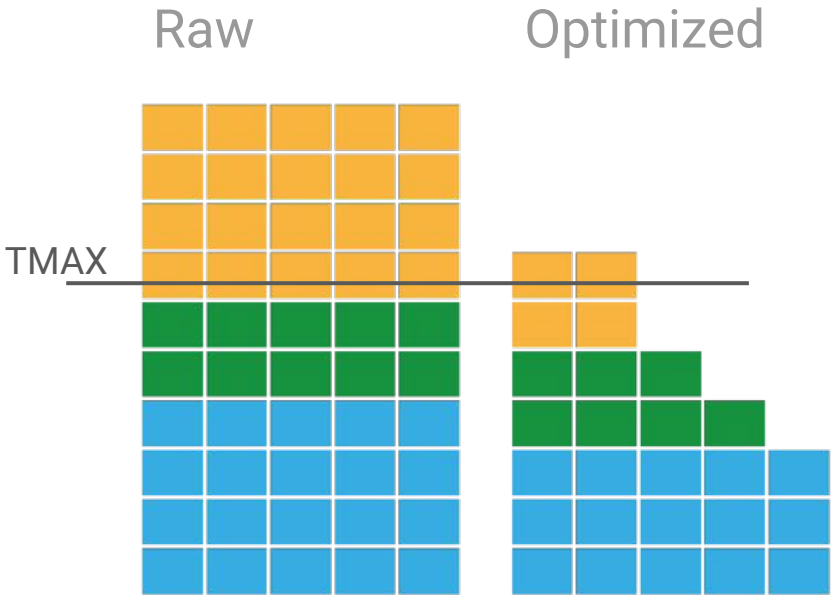


Phase III:
generate a new Yara rule $Y^* = (c_a \vee c_b)$



Signature optimization

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

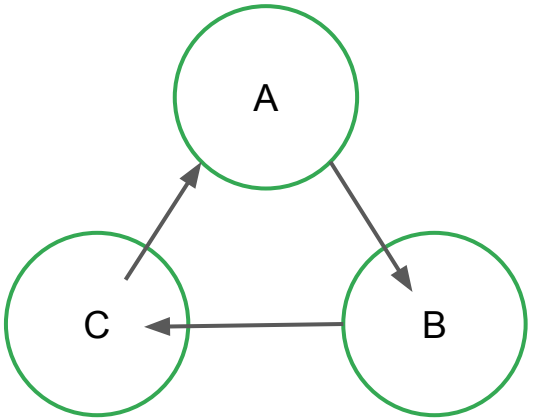
Introduction of heuristic comparisons:

- 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

Yet another YARA rule Generator

*YaYa is grandma in ES

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

EXPERIMENTAL RESULTS

Clustering results

Iterative clustering evaluation

N	Hom	Comp.	Hom	Comp.
50k	0.96	0.36	0.85	0.49
100k	0.96	0.35	0.85	0.49
200k	0.96	0.35	0.85	0.50
non-iterative	0.92	0.36	0.78	0.50

The non iterative version has a lower Hom. value.

Automatic labelling

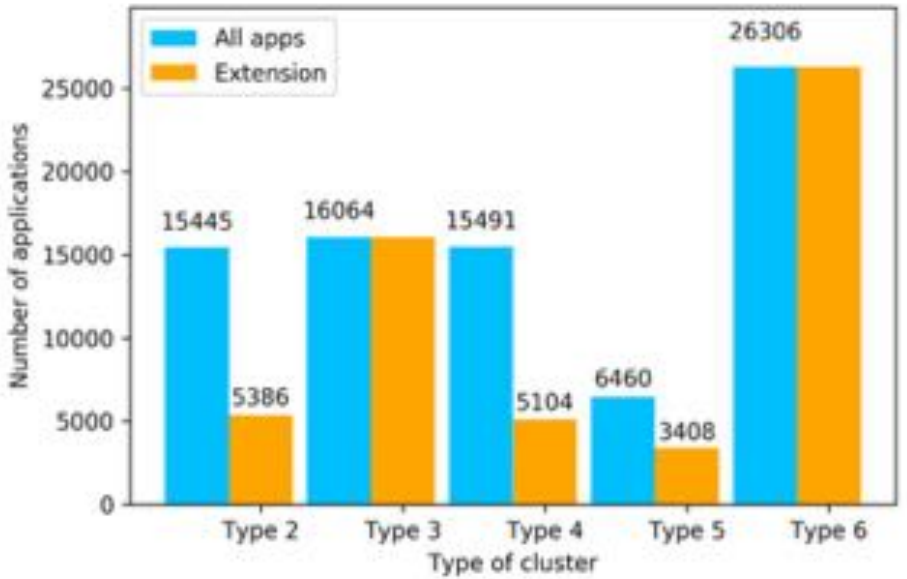
Num. of total applications (blue)

Num. of labelled apps (orange)

Family 1..6

N = 100k

1M dataset



Automatic signature generation results

Comparison state of the art

	YaraGenerator	YarGen	YaBin	BASS	**YaYaGen**
Based on	Strings	Strings	Code	Code	PE header + rules
Algorithm	Common strings	Whitelist strings	Whitelist funcs	BinDiff + LCS	CLUSTERING + SET COV. + EA
Guaranteed input coverage	NO	NO	YES	YES	YES
Packer resistance	NO	NO	GOOD	GOOD	GOOD
Clustering	NO	NO	NO	YES	YES
Scalability	YES	YES	YES	NO	YES

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

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	**23**	0	**80**
YarGen	RULES Z0	**53**	**1**	**130**
	RULES Z0 + OPCODES	53	0	86
YaBin	Yara (-y)	**36**	0	**76**
	YaraHunt (-yh)	36	**10**	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	**497**	**0**	**3349**
	UDT + GREEDY + RULES	493	0	3373
	HD + GREEDY	837	0	3237
YarGen	RULES Z0	**2782**	**2**	**3367**
	RULES Z0 + OPCODES	2760	0	3226
YaBin	Yara (-y)	**1166**	0	**3172**
	YaraHunt (-yh)	1166	**68**	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**
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 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			
UDT + CLOT + RULES	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

YaYaGen rules for UPX packed malware **do not detect** UPX packed goodware

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		)s
Zerber	329	UDT + CLOT + RULES	36		utes
		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

Signature optimization results

Optimization results: AVG num of literals

FAMILY	TPs	Non optimized	HC Optimizer	SGX Optimizer
Cluster 10	4/4	260	9.00	11.50
	4/4	260	9.20	12.60
Cluster 20	3/4	64	20.20	39.20
	3/4	64	18.20	40.90

SGX produces rules with more literals than HC.

Optimization results: AVG rule score

FAMILY	TPs	Non optimized	HC Optimizer	SGX Optimizer
Cluster 10	4/4	17614	625.00	401.70
	4/4	17614	597.00	401.80
Cluster 20	3/4	2073	612.30	555.00
	3/4	2073	620.50	583.90

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.

Talks

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

POLITECNICO
DI TORINO

Thank you

* References

Griffin, Kent, et al. "Automatic generation of string signatures for malware detection." *International workshop on recent advances in intrusion detection*. Springer, Berlin, Heidelberg, 2009.

Preda, Mila Dalla, et al. "A semantics-based approach to malware detection." ACM SIGPLAN Notices 42.1 (2007): 377-388.

Perdisci, Roberto, Wenke Lee, and Nick Feamster. "Behavioral Clustering of HTTP-Based Malware and Signature Generation Using Malicious Network Traces." NSDI. Vol. 10. 2010.

Faruki, Parvez, et al. "AndroSimilar: robust statistical feature signature for Android malware detection." Proceedings of the 6th International Conference on Security of Information and Networks. ACM, 2013.

Zheng, Min, Mingshen Sun, and John CS Lui. "Droid analytics: A signature based analytic system to collect, extract, analyze and associate android malware." Trust, Security and Privacy in Computing and Communications, 2013 12th IEEE International Conference on. IEEE, 2013.

Corno, Fulvio, Matteo Sonza Reorda, and Giovanni Squillero. "The selfish gene algorithm: a new evolutionary optimization strategy." Proceedings of the 1998 ACM symposium on Applied Computing. ACM, 1998.

https://github.com/Xen0ph0n/YaraGenerator

https://github.com/Neo23x0/yarGen

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

https://www.talosintelligence.com/bass