Accepted Manuscript

Machine-Learning based analysis and classification of android malware signatures

Ignacio Martín, José Alberto Hernández, Sergio de los Santos

PII: S0167-739X(18)32515-9
DOI: https://doi.org/10.1016/j.future.2019.03.006
Reference: FUTURE 4830

To appear in: *Future Generation Computer Systems*

Received date : 16 October 2018
Revised date : 10 January 2019
Accepted date : 3 March 2019

Please cite this article as: I. Martín, J.A. Hernández and S.d.l. Santos, Machine-Learning based analysis and classification of android malware signatures, *Future Generation Computer Systems* (2019), https://doi.org/10.1016/j.future.2019.03.006

Machine-Learning based analysis and classification of Android Malware Signatures

Ignacio Martín[a,*], José Alberto Hernández[a], Sergio de los Santos[b]

[a] *Universidad Carlos III de Madrid, Avda de la Universidad 30, Leganés, Madrid, Spain*
[b] *Telefónica Digital, Ronda de la Comunicación s/n Madrid, Spain*

Abstract

Multi-scanner Antivirus (AV) systems are often used for detecting Android malware since the same piece of software can be checked against multiple different AV engines. However, in many cases the same software application is flagged as malware by few AV engines, and often the signatures provided contradict each other, showing a clear lack of consensus between different AV engines. This work analyzes more than 80 thousand Android applications flagged as malware by at least one AV engine, with a total of almost [illegible]00 thousand malware signatures. In the analysis, we identify 41 different malware families, we study their relationships and the relationships between the AV engines involved in such detections, showing that most malware cases belong to either Adware abuse or really dangerous Harmful applications, but some others are unspecified (or Unknown). With the help of Machine Learning and Graph Community Algorithms, we can further combine the different AV detections to classify such Unknown apps into either Adware or Harmful risks, reaching F1-score above 0.84.

Keywords: Multi-Scan Antivirus; Android Malware; Security; Machine Learning; Malware classification; Graph Community Algorithms.

1. Introduction

According to Kaspersky's 2017 Security Report [1], malicious software (aka. *malware*) has become a very powerful and profitable industry, capable of delivering up to 360,000 new or altered malware samples into the Internet daily, making security experts having to deal with identifying and preventing thousands of new undetected threats every day.

AntiVirus (AV) software has been a very powerful tool to fight against malware. There are a large number of AV software tools available in market (e.g. Kaspersky, ESET, AVG, etc), each one has its own set of rules and expertise

*Corresponding author
Email address: ignmarti@it.uc3m.es (Sergio de los Santos)

to identify threats. *Multi-scanner* AV tools have come into play to further improve decision making on whether some piece of software is dangerous or not. Essentially, multi-scanner systems check suspicious software against several AV engines, thus returning the outputs for each engine.

Nevertheless, multi-scanner tools have also shown the so called *lack of consensus* of engines [2], in other words, AV engines do not agree about the "malwarish" nature of certain applications (some AV engines flag an app as dangerous while others do not) and, even when they agree, sometimes they disagree on the type of threat or malware *class*. For instance, adware is usually considered malware although its harm is often limited to annoying users. Oppositely, other types of threats, such as worms, trojans or spyware target their victims' personal data, credentials or resources for theft.

In this light, this article investigates on the lack of consensus of multi-scanner tools and, to the best of our knowledge, we advance beyond the state of the art in the following directions:

- We analyze a large collection of Android applications (more than 80 thousand) tagged as malware by at least one AV from a set of 61 different engines, yielding almost 260 thousand malware signatures.
- Using our open-source tool for mining Android malware signatures, namely *SignatureMiner*, we clean, homogenize and transform this large set of malware signatures into *normalized* malware family names for further data analysis and processing.
- We perform an in-depth analysis of the resulting malware families and their interrelations engine-wide, identifying up to 41 different malware families which belong to three broader categories, namely *Adware*, *Harmful* and *Unknown*.
- With the help of Graph Community Algorithms, we study the relationships between malware families according to their detection patterns and AV engines, showing that some malware types are indeed closely related, besides some AV engines are clearly focused into detecting Adware or Harmful or both, which aligns with the lack of consensus nature of multi-scanner AV tools.
- We make use of Machine Learning classification tools, in particular Logistic Regression with Lasso regularization and Random Forest to classify Unknown applications into Adware or Harmful, showing good performance results (F1-score above 0.84) and shedding light into which AV engines are specialized at each malware category.

2. Previous Work

Antivirus Software solutions provide early detection and prevention of malware infection of devices. AVs have been persistently scrutinized, even in their mobile versions: the authors in [3] review the key points in designing AV engines for mobile devices and how to prevent detection evasion. Similarly, Rastogi et al [4] identify how and when do AV engines fall for obfuscation attacks, finding many to be vulnerable to some kind of transformation attack. The authors in [5] perform data analytics on multi-scanner outputs for Android Applications to find their behavior patterns.

Actually, AV engines sometimes contain flaws and vulnerabilities that must be addressed immediately, such as the ones discovered by the authors in [6]. Other authors even argue the impossibility of fighting against Android Malware through anti-malware system fingerprinting and evasion techniques [7].

Furthermore, AV performance is analyzed in [8], where the authors quantify how devices' performance is affected by AV execution, and [9], a characterization and evaluation of AV overhead. In addition, the authors in [10] compare the design of 30 top AV solutions focusing on their detection and prevention capabilities. In a similar approach, Quarta et al [11] leverage VirusTotal detections to perform a black-box analysis of its AV engines solely based on input samples and the outcome from each AV. Furthermore, the authors in [12] raise concerns regarding malware developers using multi-scanner tools in their loops and propose a methodological approach to detect them from VirusTotal submissions. Lately, in-cloud AV solutions are getting relevance, as they improve performance and availability of resources at smaller on-device cost [13].

Concerning multi-scanner tools, works like [14, 15] have shown the advantages of using more than one AV engine to perform malware detection. AV performance comparison has been studied by the authors in [16], modeling AV confidence through a hyper-exponential curve over a large set of AV engines from the VirusTotal Multi-scanner tool. In [17], AV labels from VirusTotal are subjected to temporal analysis using a collection of malware applications obtained through a honeypot network.

However, the authors in [2] recall the lack of agreement between AV engines in certain applications. To alleviate this, the authors in [18] propose a combination scheme for multi-scanner detections based on a Generative Bayesian model that provides an estimation of the probability that each sample has to be malware. Lately, and inspired by previous authors, Du et al [19] have developed a statistical methodology to infer a Ground Truth dataset when this is not available.

The issue of malware families has been addressed by several authors who have proposed categorization schemes for Android malware applications. In [20] the authors find up to 49 distinct malware families whilst the authors in [21] propose a text mining approach to obtain and classify malware families according to application code. Similarly, Zheng et al propose in [22] a system for the collection and categorization of zero-day malware samples into different families. Additionally, the authors in [23] propose a system to classify malware samples in their families by inspecting their code and API calls.

Adware has been considered a different type of malware to the rest in the literature. In [24]the authors consider malware and adware separately in their Android malware detection system. Also, the authors of [25] acknowledge how adware is not equal to other type of malware and can affect malware detection results. Finally, Yang et al argue in their work [26] that adware should be separated from "truly malicious apps" to provide undisputed malware detection results due to the controversy among AVs on whether to label an adware sample.

Despite these efforts, Maggi et al. [27] extensively review the naming inconsistencies that AV engines incur in when assigning a class to a malware sample. Furthermore, the authors of [28] perform a comprehensive analysis of a large labeled AV dataset and indicate the necessity of considering various AV engines to accurately detect threats. Recently, Wei et al [29] have obtained and analyzed a nearly $25,000$ sample-wide dataset through clustering and manual inspection of each group, providing a large *Ground Truth* dataset following similar procedures to *AVClass* [30].

Sebastián et al [30] proposed *AVClass*, a system to normalize AV labels from different vendors and determine the actual class from the different detection outputs for the same applications. Similarly, the authors in [31] propose Euphony, a system that extracts family names from heuristics and previous knowledge, homogenizes them across engines through graph analysis and provides the most appropriate malware family per sample. Recently, we proposed our lightweight approach to malware family classification: *SignatureMiner* [32].

The rest of this paper is structured as follows: Sections 3 and 4 recap the use of SignatureMiner for data collection along with some insights on the dataset. Section 5 inspects AV engines and signature token correlations to verify and improve the categorization scheme and derive insights and relationships between them. Section 6 details the training and usage of a Machine Learning classifier to determine a more specific category for Unknown samples. Finally, Section 7 concludes this article by summarizing the main findings and most relevant conclusions.

3. Analysis of malware family classes using SignatureMiner

3.1. Dataset

In this work we start from a dataset with $82,866$ different suspicious Android applications provided by TACYT[1] in May 2015. TACYT is a Telefonica's commercial project that collects Android applications from several markets, including Google Play and stores not only the application code itself, but also meta-information related from the Android market (number of downloads, rating stars, descriptions, comments from the users, etc). After this, TACYT uses online multi-scanner systems (like VirusTotal, MetaScan, Jotti, etc) but also internal scanner systems to identify malware and further investigate the output provided by the most popular AV engines (Kaspersky, BitDefender, McAffee, Sophos, Avast, etc), 61 in total. The engines have been anonymized for privacy reasons throughout the paper with a name in the range $AV_1, \ldots, AV_{61}$.

[1] See https://www.elevenpaths.com/es/tecnologia/tacyt/index.html for further details

Although these identifiers typically contain pointers to a reduced subset of malware types, they typically present completely heterogeneous detection identifiers that prevent cross-engine analysis of detections. For instance, as noted in [32], the next list shows the malware signature output by three different AV engines for the same Android app:

- A variant of Android/AdDisplay.Startapp.B
- Adware/Startapp.A
- Adware.AndroidOS.Youmi.Startapp (v)

There seems to be a consensus between the three AV engines regarding the malware type of this app, namely *Adware*. In particular, this app is *Startapp* library-based. In order to homogenize such differences, we use the open-source *SignatureMiner* tool[2] to craft a set of normalization rules from signatures and assign them a *cannonical name*, such that all detections from any family have the same name. The SignatureMiner process used to identify and unify signatures is detailed in our previous article [32].

3.2. Identified Malware Families

After normalization, malware signatures have been inspected, defined and categorized according their end goal based on information provided by AVs:

1. those looking into fast monetary gain through too many ads (in what follows *Adware* type)
2. those looking into more aggressive and intrusive techniques (in what follows *Harmful* type)
3. those which engines are unable to properly identify (in what follows *Unknown*).

Table 1 shows 41 malware classes ($S1, \ldots, S41$) provided by SignatureMiner from raw signatures. The table contains the predicate for each rule (regular expression syntax), family name of the malware class assigned and its associated broader malware category (i.e. Adware, Harmful or Unknown), along with some statistics and numbers for each class. Broad categories have been assigned to each family according to their nature and AV detection signatures. For instance, $S1$ contains all the cases of *AirPush* family class, which belongs to the Adware category. The AirPush class has been found in 12,802 different Android apps and received 35,850 detections from 26 different AV engines.

Next sections overview such three broad malware categories identified in our dataset.

[2]Available at GitHub at: `https://github.com/ignmarti/SignatureMiner`

#	Regexp rule	Family Name	Category	Det. Count	No. Apps	AVs
S1	.*a[ir]*push?.*	Airpush		35,[illegible]	12,802	26
S2	.*leadbolt.*	Leadbolt		1[illegible],414	4,045	21
S3	.*revmob.*	Revmob		38,[illegible]93	13,680	18
S4	.*startapp.*	StartApp		2[illegible],443	11,963	13
S5	[os]*apperhand.* |.*counterclank.*	Apperhand		1,[illegible]06	716	12
S6	.*kuguo.*	Kuguo		[illegible],127	1,893	23
S7	wapsx?	WAPS		1,546	344	6
S8	.*dowgin.*|dogwin	Dogwin		1,098	421	23
S9	.*cauly.*	Cauly	Adware	1,143	626	3
S10	[os]*wooboo	Wooboo		220	120	14
S11	[os]*mobwin	Mobwin		1,284	249	3
S12	.*droidkungfu.*	DroidKungFu		105	54	3
S13	.*plankton.*	Plankton		4,557	741	25
S14	[os]*you?mi	Youmi		1,472	370	22
S15	[osoneclick]*fraud	Fraud		736	382	19
S16	multiads	Multiads		560	555	3
S17	.*adware.*|ad.+	Adware (gen)		33,133	24,515	46
S18	riskware	Riskware		1841	1353	14
S19	spr	SPR		1,789	1,789	2
S20	.*deng.*	Deng		2,926	2,926	1
S21	.*smsreg	SMSreg		649	440	16
S22	[os]*covav?	Cova		1,564	1,296	5
S23	.*denofow.*	Denofow		1,224	610	11
S24	[os]*fakeflash	FakeFlash		1,381	510	15
S25	.*fakeapp.*	FakeApp	Harmful	518	420	14
S26	.*fakeinst.*	FakeInst		493	401	22
S27	.*appinventor.*	Appinventor		4,025	3,113	6
S28	.*swf.*	SWF		4,651	4,566	10
S29	.*troj.*	Trojan (gen)		23,775	16,851	49
S30	.*mobi.*	Mobidash		981	796	16
S31	.*spy.*	Spy		1483	1,221	26
S32	.*gin[ger]*master	Gingermaster		58	36	10
S33	unclassifiedmalware	UnclassifiedMalware		857	855	1
S34	.*virus	Virus		959	896	15
S35	.*heur.	Heur		182	179	15
S36	.*gen.*	GEN		9,827	9,118	25
S37	[osgen]*pua	PUA	Unknown	1,249	1,152	2
S38	[ws]*reputation	Reputation		2,886	2,885	1
S39	.*applicunw[illegible].*	AppUnwanted		4,863	4,860	1
S40	.*artemis.*	Artemis		9,662	6,175	2
S41	.* (Default Case)	Other		10,778	7,880	57
	TOTAL			259,608		

Table 1: Malware classes, their figures and their SignatureMiner rules

3.2.1. *Adware*

This category includes those malware classes that abuse advertisement display for profit. There are in total 60,538 applications tagged with at least one Adware class. The large penetration of the Adware category in this dataset suggests that a majority of malicious applications within Google Play are adware-related apps.

- *Leadbolt*, *Revmob*, *Startapp*, *WAPSX*, *Dowgin/dogwin*, *Cauly*, *Modwin* and *Apperhand/Counterclank* are well-known advertisement networks which are sometimes maliciously misused to perform full screen and invasive advertising.

- *Kuguo* is an advertisement library known for the abuses committed by their developers.

- *Youmi* and *DroidKungFu* are advertising services known for being involved in data ex-filtration episodes.

- *Airpush* is another advertisement network company known for the abuse of adbar pushing notifications committed by its developers.

- *Fraud/osoneclick* refers to a fraudulent malware that attempts to increase number of ad clicks by stealthily retailing advertisements in the background of user interactive applications.

- *Adware (gen)* tag is a generic reference assigned to those samples that do not contain more information in them. In addition, some AVs just mark as *Multiads* applications those which contain different advertisement libraries capable of displaying invasive ads.

It is worth remarking that adware is sometimes a controversial type of malware: advertising is an accepted activity as a way to monetize modern applications (mobile apps or webpages) and therefore not all advertising can be considered malicious. As a result, the borders between legitimate and maliciousness in adware are unclear and different AV engines can have completely different policies, resulting in a clear lack of consensus between AV engines concerning Adware.

3.2.2. *Harmful*

This category includes more dangerous threats, such as enrolling users in premium services or ex-filtrating data through permission overload or exploits. 29,675 applications have been assigned at least one signature to this category.

- *Deng*, *SPR (Security and Privacy Risk)* and *Riskware* are generic names given to flag apps that may unjustifiably require potentially harmful permissions or include malicious code threatening user privacy.

- *Denofow* and *Cova* are generic references to trojan programs which attempt to subscribe users in premium SMS services.

- *SMSReg* is a generic way to flag applications that request SMS-related permissions for data exfiltration or premium subscriptions.
- *FakeFlash*, *FakeInst* or *Fakeapp* are names for applications that replicate the functionalities of other popular apps adding to their malicious code or actions.
- *Appinventor* is a developer platform used to build and generate applications extensively preferred by malware developers.
- *SWF* stands for different versions of Shockwave Flash Player Exploits.
- *Trojan (gen)* is the generic reference of engines to Trojan applications.
- *GingerMaster* is a well-known family of rooting exploits.
- *Spy* is a generic reference to applications incurring in spyware threats.

3.2.3. Unknown

This category includes AV detections which do not include class-related information, either due to generic signatures from AVs or signatures not matching any rule in the dataset. There are 23,915 applications within this group.

- *UnclassifiedMalware*, *Virus*, *Heur* (from heuristics), *GEN* (Generic Malware), *PUA* (Potentially Unwanted Application), *Reputation*, *AppUnwanted* (Application Unwanted) and *Artemis* are generic tags given by different engines in order to flag applications that are detected as unspecified threats.
- *Other* includes the applications which have not been classified due to the lack of signature patterns.

Table 1 clearly displays a certain preeminence of Adware apps over the rest. In particular *Revmob*, *Airpush* and *Adware* are the most popular signatures involving many AV engines. *Trojan* detections are also very popular in the Harmful category, as it is used by up to 49 different engines. Generally, many malware family classes are spotted by more than a single engine, with some exceptions, especially within the Unknown category, where family classes like *Reputation* or *AppUnwanted* are flagged by only one AV engine.

4. Exploratory data analysis

Let A denote the application-AV indicator matrix of size $82,866 \times 61$ whose elements $A_{ij} \in \{0,1\}$ are set to 1 if the i-th Android app has been flagged by the j-th engine or 0 otherwise. This matrix indicates which AVs label each application as malware, i.e. the rows in matrix A are the *detection vectors* of each application sample. Matrix A is very sparse with only 5% of all the entries set to one. On average, each application is detected by 3.1 ± 3.4 engines, suggesting a very large variability.

Indeed Fig. 1 depicts a histogram of application detection counts. As shown, the histogram follows a heavy-tailed like pattern where most malware applications are flagged by only one AV engine whilst some few applications get much higher detection rates. Single-detection applications represent the majority of cases with a total of 38, 933 (46.9% of the total). In fact, no single application has been flagged by all 61 AV engines, being the highest detection count for application no. 78, 692 with 53 different AV hits.

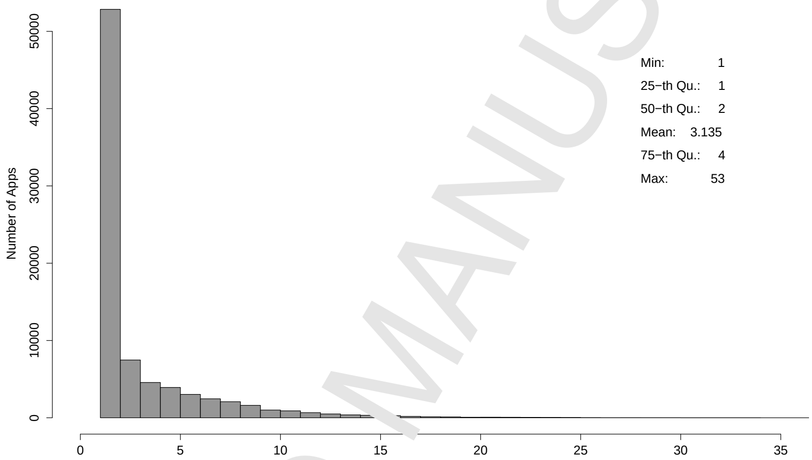


Figure 1: AV detection count per application

Fig. 2 shows the activity of each AV engine. As shown, the most active AV engines are AV27, AV58, AV7, AV2, AV30 and AV32, each one accounting for more than 10, 000 malware app detections.

Fig. 3 represents the popularity of each malware family class (third last column of Table 1). As previously observed, Adware-related signatures are the most common cases of flagged malware, in particular some specific libraries like *Airpush*, *Leadbolt*, *Revmob* and *StartApp* are very popular. Regarding Harmful applications, generic *Trojan* signatures are the most popular ones.

Now let B denote application-Family indicator matrix of size $82,866 \times 41$ whose elements B_{ij} are set to the number of times the i-th Android app has been flagged in the j-th malware category. Scanning matrix B, we observe that single-detection applications account for 38, 933 applications, while the rest (i.e. 43, 933 apps) represent apps with multiple AV detections. From these, 27, 781 apps (i.e. 63.26% of them) are assigned to more than one malware family. In particular, these 27, 781 applications receive between 2 and 12 different family labels (see histogram of Fig. 4). This is another proof for the lack of consensus between AV engines referred in the literature.

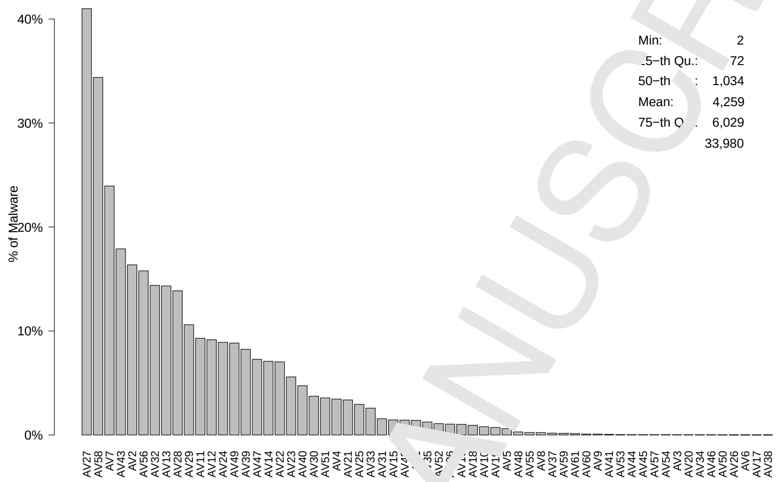


Figure 2: Most active AVs

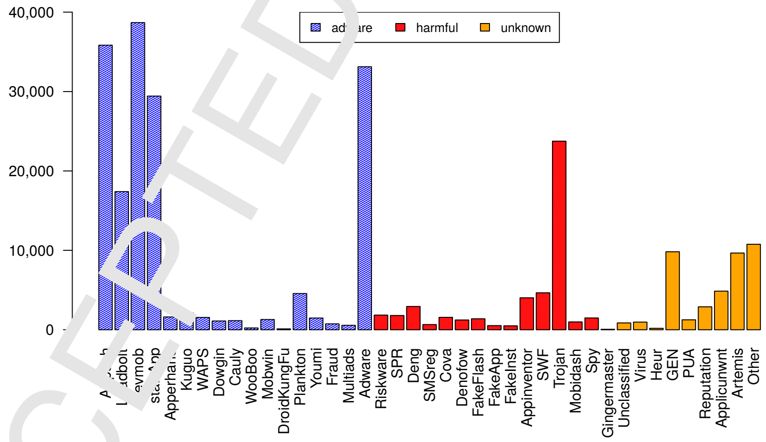


Figure 3: Frequency of detections per malware class

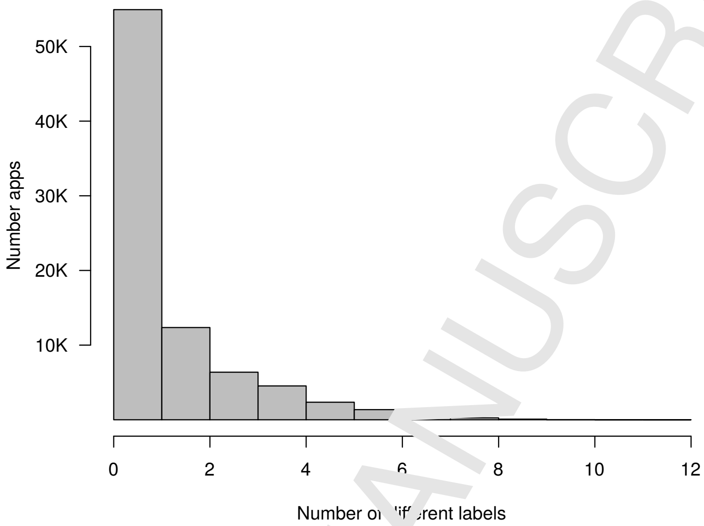


Figure 4: Histogram of different labels per application

5. Analysis and insights of malware family classes and categories

As previously stated, we have detected 38,933 Android apps flagged by a single AV engine. Of the rest (those with two AV detections or more), all AV detections agree on the exact same malware class in 16,152 cases whereas the remaining 27,781 apps show some sort of disagreement between at least two engines. These findings are in line with [27] and [2]: one third of applications have no clearly defined malware class due to uncertain decisions from some engines. Sometimes, two engines use different names for the same malware class but most often it happens that AV engines disagree on the malware type of a piece of software.

5.1. Correlation of malware categories

Recall that each of the 41 malware classes have been assigned to one of the three categories defined above, namely *Adware*, *Harmful* and *Unknown*. To analyze categories, let D, the application category matrix, refer to an $82,866 \times 3$ matrix where D_{ij} accounts for the number of times the i-th application has received a detection in category Adware ($j = 1$), Harmful ($j = 2$) or Unknown ($j = 3$). The correlation of the columns of matrix D specifies how frequently each pair of categories occur on the same application. Table 2 illustrates the correlation matrix of D.

In the table, the Harmful and Adware categories show very weak correlation (0.06) indicating that, in general, AV engines often do not make many contro-

	Adware	Harmful	Unknown
Adware	1	0.06	0.3
Harmful	0.06	1	0.44
Unknown	0.3	0.44	1

Table 2: Correlation of matrix D (Malware Categories)

versial detections involving them both. Hence, it seems that AV engines have a very strong opinion on whether some app is Adware or Harmful.

However, the Unknown category is confirmed to contain samples from the other categories whereby AV engines are unable to specify. Actually, the larger correlation value for Harmful applications (0.44) suggests that these detections are probably more often Harmful than Adware applications (0.3). In the next subsections, we further examine the relationships between malware families and AV engines using graph community algorithms.

5.2. Graph Community Search for Class Redundancies

Graph theory provides algorithms to study member relationships within a network of entities including distances within the network and inference of look-alike communities. When using the correlation matrix of any of the aforementioned matrices A, B or D as the adjacency matrix of a graph, we can leverage existing graph-theory algorithms to gain insights into both malware families and AVs.

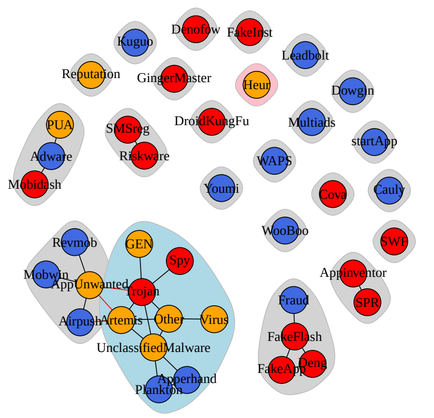
Hence, starting from matrix B defined in Section 4 we compute its correlation matrix, i.e. $Corr(B)$ and define a Graph $G = (N, E)$ whose adjacency matrix is $Corr(B)$. Then, graph G has 4[illegible] nodes (malware classes) and the weights of the edges are equal to the correlation values between malware classes.

Using node edge betweenness [33], we can group together nodes according to their correlation values to find which malware classes are close together. In order to avoid generating communities out of noise, we force all correlation values below some $Corr_{min}$ threshold to be equal to zero.

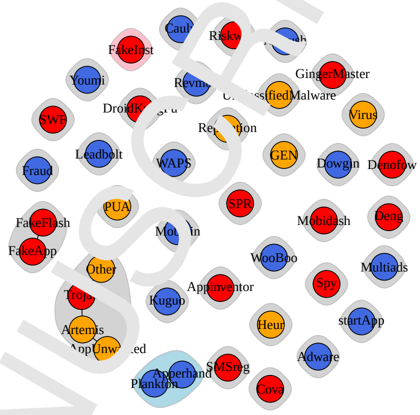
Fig. 5 illustrates two graphs of the communities formed by different malware families and the distance dendrogram used to group nodes in their communities. The case of Fig. 5(a) depicts a noisy graph where the communities displayed are weakly correlated (correlation thresholds of 0.2 and 0.35 respectively). Essentially, most malware families are isolated unless a sufficiently small correlation threshold is allowed.

Then, in Fig. 5(b), with a higher $Corr_{min}$, the previous noise disappears leaving a graph mostly independent, supporting the observations and signature schemes in Section 3. Nonetheless, there are three relevant communities in the graph: one larger community formed by three Unknown signatures (*AppUnwanted*, *Artemis* and *Other*) and one Harmful threat (the generic *token*) and two smaller communities, *FakeFlash-FakeApp* and *Plankton-Apperhand*.

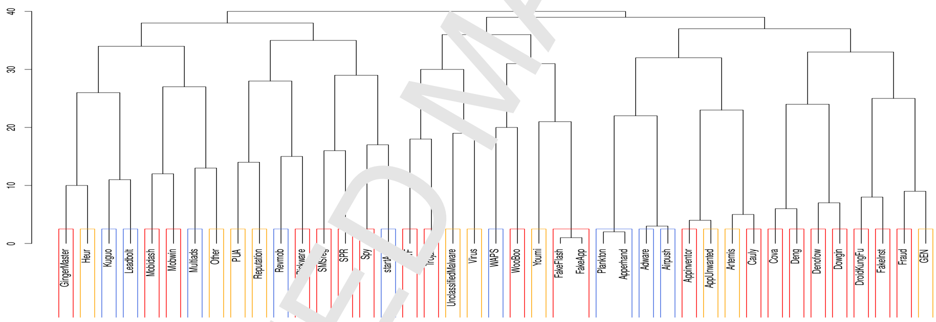
The dendrogram of Fig. 5(c) further illustrates the pairwise relationships between each family class, showing some level of similarity degrees between



(a) $Corr_{min} = 0.2$



(b) $Corr_{min} = 0.35$



(c) Dendrogram $Corr_{min} = 0.5$

Figure 5: Communities of malware classes for different correlation threshold $Corr_{min}$

certain classes of Adware and Harmful with others from the Unknown category. There are two very close communities obtained, namely *FakeFlash-FakeApp* and *Plankton-Apperhand*, having moderate-high correlation values of 0.61 and 0.72 respectively.

In conclusion, we observe that low correlation values exist between most malware family classes except for the above two communities, but still there are some interesting relationships between certain Harmful and Unknown families that can be used by the ML classification algorithms of Section 6.

5.3. Grouping AVs by their detection schemes

We now focus our attention to investigating whether or not some AV engines potentially detect different malware families and categories. Thus, it is interesting to categorize AVs according to the different detections they perform and their frequencies. To do that, we rely again on graph-based clustering, using as adjacency matrix the correlation of B^T, which is the transpose of matrix B.

Fig. 6(a) shows the resulting graph containing nodes colored according to their group (correlation threshold set to 0.35). In general, we observe that most AV engines belong to certain communities, while a few others (in brown) are isolated and not correlated with the rest. These isolated AVs are: AV39, AV40 and AV22.

We observe four main communities of AV engines: an Adware related group (blue), whose most frequent detections are *Revmob*, *Adware*, *Airpush* or *startApp*; a second Harmful-oriented group of AVs (red), whose main family detections are *Trojan*, *Airpush*, *Gen* or *Kuguo* and two broader mixed AV groups with detections in both categories. The first mixed group (green) shows detections mainly in *Plankton*, *Adware* or *Trojan* whereas the second mixed group (orange) show detections in *Other*, *Dewa*, *Airpush* or *Leadbolt*.

These groups illustrate that some engines often incur in similar detection patterns, either by focusing on specific families (the broader categories aforementioned) or, by making more varied detections. It is worth noting that the mixed yellow group includes more families from the Unknown broad category as well as slightly more families from the Harmful categories. On the other side, the darkgreen cluster seems to include detections from both Adware and Harmful categories.

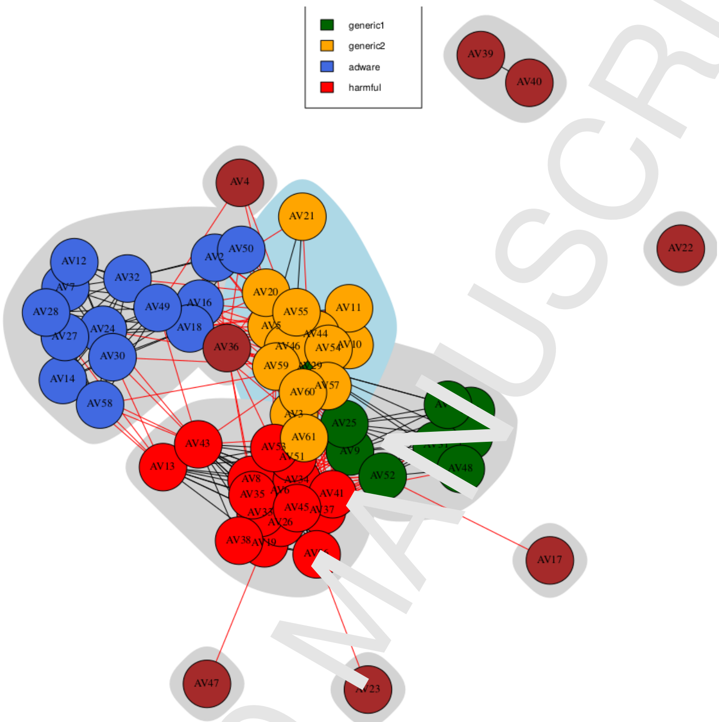
Oppositely, the isolated AV engines tend to aggregate all families from the Unknown category which are more specific to just one or very few AV engines. As a result, these isolated AV engines are unable to produce accurate detection information and have to stick to very generic detection names, such as *Heur*, *GEN* or *PUA*.

Finally, Fig. 6(b) depicts the dendrogram after applying hierarchical clustering to matrix B, showing pairwise comparisons between AVs. Again, the above isolated AVs are indeed separated from the rest while other AVs are very close together, like AV61 and AV60. The colors used in the groups are consistent with the graph communities.

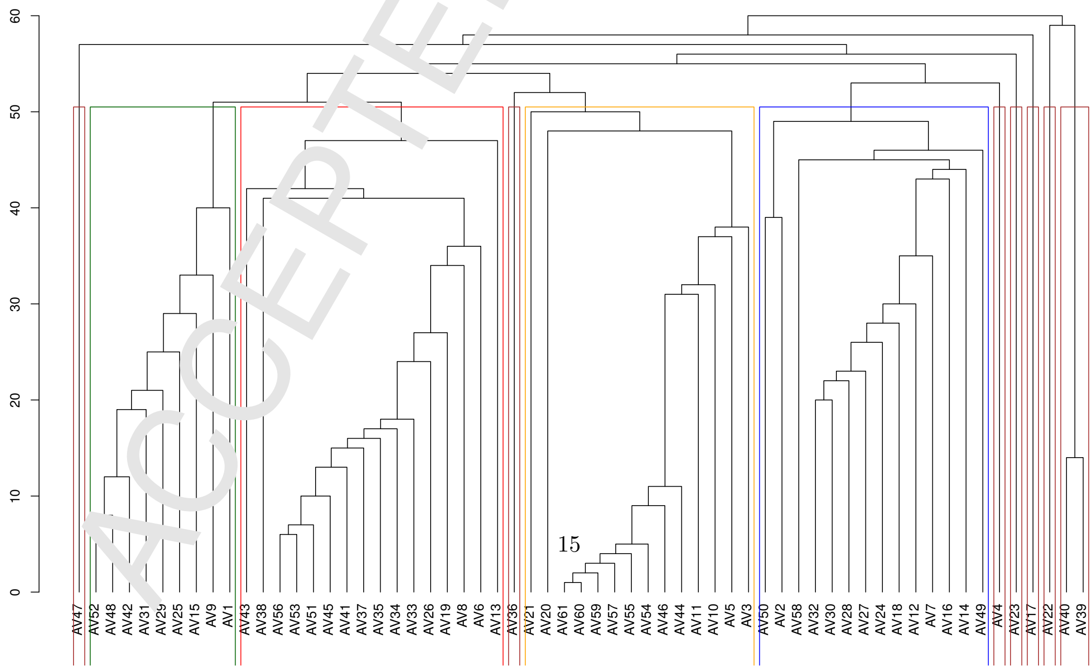
6. Identifying Unknown Category malware

As shown in previous sections, the malware families within the generic *Unknown* category are often closer to *Harmful* than to *Adware* cases. This section aims at further analyzing the malware families within the *Unknown* category, making use of Machine Learning (ML) algorithms.

Essentially, the idea is to train an ML classifier to identify the malware category for each data sample (i.e. each app), and provide a binary decision (Adware or Harmful) using as features only the decisions made by the 61 AV



(a) Graph groups



(b) Dendrogram

Figure 6: Community cluster of AV engines according to the classes they detect

engines. This classifier has a twofold objective: (i) To provide a fast categorical assignment system based on each AV decision, and (ii) to further identify which AV engines are more powerful at detecting each malware family (Adware vs Harmful). To this end, we consider two ML classification algorithms: (i) *Logistic Regression* (LR) due to its ability for probability estimation and interpretability and (ii) *Random Forest* (RF) aiming at maximizing prediction accuracy and F1-score metrics.

Starting from the hypothesis that all apps in our dataset represent some type of danger to the user or malware (soft Adware or dangerous Harmful) since they have been flagged by at least one AV engine, we construct a Ground Truth labeled dataset using the following methodology. First of all, in the vast majority of cases (46.9% apps as shown in Fig. 1), apps are flagged by only one AV engine, so in these cases such apps are directly assigned to either Adware or Harmful class accordingly. In cases with more than one AV engine label, we use majority voting to assign the Adware or Harmful label to each Android app to generate a Ground Truth dataset; for instance, if a given app has been flagged by three AVs where two of them say that the app belongs to Adware while the third one says it is Harmful, we assign the Adware label to it. In case of a draw (for instance, two AVs one indicating Adware and the other suggesting Harmful) we consider this app as Unknown. The idea is to use an ML classifier trained with this Ground Truth labelled dataset to identify the Unknown apps which have very generic and meaningless signatures. Furthermore, the use of AVs as features in the ML model allows to identify which AVs are more accurate at detecting each family class: Adware or Harmful.

The Logistic Regression algorithm has been regularized to improve its performance in the analysis of AV engine contribution: regularization performs embedded feature selection by adding a constraint to the optimization function that forces less-relevant AVs' contribution to be reduced to zero, while other engines are associated with a weight according to their relevance for harmful detection (positive contribution) or adware (negative contribution). There are several regularization schemes in Logistic Regression, being the two most-widely used the *lasso* (ℓ_1) and *ridge* (ℓ_2), which penalize attributes according to the norm and the norm squared of the coefficients respectively. We choose lasso regularization as it typically performs better when applied to binary feature-sets.

To train and validate both algorithms we use the samples in the Adware and Harmful categories first, assuming as label the category they fall into. Hyper-parameter tuning is performed using classical 10-fold cross-validation; this helps to adjust (i) the regularization parameter (C) in Logistic Regression and the number of trees in the case of the Random Forest.

Table 4 displays the performance results for both ML algorithms during training and validation, showing F1-scores above 0.75 for Logistic Regression and 0.84 for Random Forest. The table reports accuracy (Acc) and F1 (F) scores for both training and validation which correspond to the cross-validation and the hold-out validation datasets respectively. After cross-validation, the LR algorithm is optimally configured with a regularization parameter (C) of 53.204 while the RF model is optimally configured to have 166 trees. As expected, RFs outperform LR in terms of accuracy showing outstanding results (0.92 accuracy test).

Table 3: Train and validation scores over the defined categories data

Algorithm	Acc_{train}	Acc_{val}	F_{train}	F_{val}
Logistic Regression	0.895	0.894	0.758	0.757
Random Forest	0.935	0.927	0.859	0.841

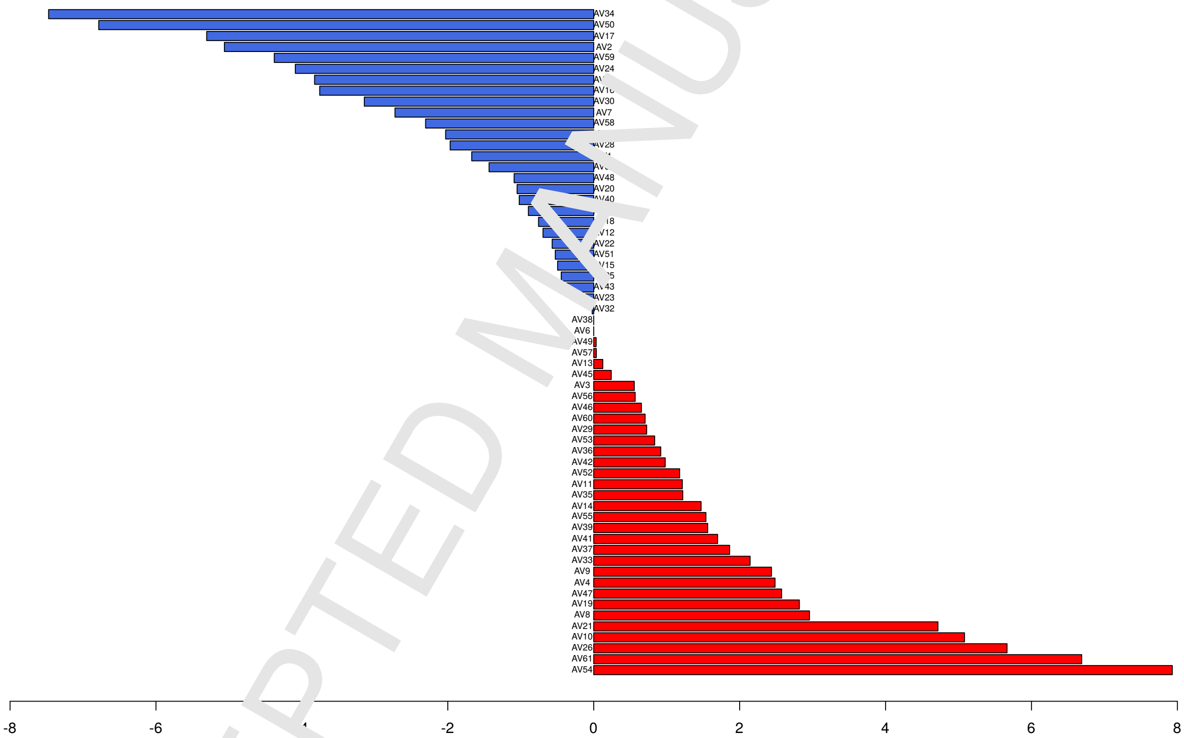


Figure 7: Computed weights after ℓ_1 logistic regression. Positive weights indicate more harmful-aware AV engines and negative ones more adware-aware engines. Zero weight engines are those which detections are irrelevant for classification

Let us now focus on the AV weights provided by LR. Fig. 7 displays the AV weights marked in color to the category they contribute most, namely Adware (blue) vs Harmful (red). At a first glance, we observe that there are slightly more harmful-aware AVs (31) than adware-aware ones (26).

The results of Fig. 7 are consistent with those of Section 5: AV engines like AV50 and AV2 which appear in the adware groups of Fig. 6 have also a low LR coefficient in Fig 7, while those AV engines in harmful groups like AV8 and AV26 have large LR coefficients.

Furthermore, the darkgreen engines in Fig. 6(a) have weights near zero, thus suggesting that they are specialized at both Adware and Harmful. On the contrary, the orange group, that had the larger proportion of Unknown category samples, contains some of the highest weighted engines in the Harmful category, like AV54 and AV61.

Finally, we have used the trained RF model to classify the apps in the Unknown malware category. As a result, we observe that 51.5% of the samples are classified as Harmful while the remaining 48.5% belong to Adware. This result is in-line with the correlation experiments of 5 which already indicated that there is a majority of Unknown samples belonging to Harmful category.

Table 4: Amount of harmful samples detected at each family in the unknown category

Family	Virus	Heur	GEN	PUA	Reputation	Artemis	Other	SINGLETON
Harmful	83.48%	47.87%	34.63%	50%	61.[illegible]%	41.66%	39.98%	38.77%

Furthermore, Table 4 indicates the amount of applications of each malware family in the Unknown category that are identified as Harmful by the classifier. As shown, most apps tagged as *Virus* fall in the Harmful category, while the *Gen* family class is closer to Adware. Within *Reputation*, there is a majority of apps falling within the Harmful class while *Artemis*, *SINGLETON* and *OTHER* are closer to Adware. There are two signatures missing: *applicunwt*, which is not selected as target family in any case by majority voting and *unclassifiedMalware*, which appears only once and is classified as Adware.

7. Summary and conclusions

This work has analyzed 259,608 malware signatures from 82,866 different Android applications flagged as malware by at least one out of 61 AV engines. The signatures have been normalized into a common namespace using the SignatureMiner tool to enable cross-engine analysis. Then, malware signatures have been inspected discovering an adware-dominated ecosystem where families can be summarize into three categories: *Adware*, *Harmful* and *Unknown* according to the risk and nature of each threat.

As shown, Adware and Harmful apps are typically independent and robust, but generic signatures in the Unknown category do not provide further information on the nature of the risk. Using graph-community algorithms and hierarchical clustering we have identified which signatures are close together belonging to a group with similar threats, and also identified which AVs are more focused at detecting each malware category.

Finally, Machine Learning classifiers have been shown to provide not only malware categorization but also AV engine weighting and adware harmful probability estimation. These ML models provide outstanding classification results (F1-score of 0.84 in test) and have shown to further identify some of the Unknown family classes into either Harmful/Adware threats.

Acknowledgments

The authors would like to acknowledge the support of the national project TEXEO (TEC2016-80339-R), funded by the Ministerio de Economía y Competitividad of SPAIN through, and the EU-funded H2020 SMOOTH project (grant no. H2020-786741).

Similarly, the authors would like to remark the support provided by the Tacyt system (`https://www.elevenpaths.com/es/tecnologia/tacyt/index.html`) for the collection and labelling of AV information.

Finally, Ignacio Martín would like to acknowledge the support granted by the Spanish Ministry of education through the FPU scholarship he holds (FPU15/03518).

References

[1] White paper: Kaspersky security network, Tech. rep., Kaspersky Lab (2017).
URL `https://media.kasperskycontenthub.com/wp-content/uploads/sites/43/2017/06/20080315/KESB_Whitepaper_KSN_ENG_final.pdf`

[2] M. Hurier, K. Allix, T. Bissyandé, J. Klein, Y. L. Traon, On the lack of consensus in anti-virus decisions: metrics and insights on building ground truths of android malware, in: Detection of Intrusions and Malware, and Vulnerability Assessment, Springer, 2016, pp. 142–162.

[3] H. Huang, K. Chen, P. Liu, S. Zhu, D. Wu, Uncovering the dilemmas on antivirus software design in modern mobile platforms, in: International Conference on Security and Privacy in Communication Systems, Springer, 2014, pp. 359–366.

[4] V. Rastogi, Y. Chen, X. Jiang, Catch me if you can: Evaluating android anti-malware against transformation attacks, IEEE Transactions on Information Forensics and Security 9 (1) (2014) 99–108.

[5] I. Martín, J. A. Hernández, S. de los Santos, A. Guzmán, Insights of antivirus relationships when detecting android malware: A data analytics approach, in: Proceedings of the ACM Conf. on Computer and Communications Security, CCS '16, ACM, 2016.

[6] H. Huang, K. Chen, C. Ren, P. Liu, S. Zhu, D. Wu, Towards discovering and understanding unexpected hazards in tailoring antivirus software for android, in: Proceedings of the 10th ACM Symposium on Information,

Computer and Communications Security, ASIA CCS '15, ACM, New York, NY, USA, 2015, pp. 7–18.

[7] D. Maier, T. Müller, M. Protsenko, Divide-and-conquer: Why android malware cannot be stopped, in: 2014 Ninth International Conference on Availability, Reliability and Security, 2014, pp. 30–39.

[8] M. Al-Saleh, A. Espinoza, J. Crandall, Antivirus performance characterisation: system-wide view, IET Information Security 7 (2) (2013) 126–133.

[9] D. Uluski, M. Moffie, D. Kaeli, Characterizing antivirus workload execution, SIGARCH Comput. Archit. News 33 (1) (2005) 90–98.

[10] H. Huang, K. Chen, C. Ren, P. Liu, S. Zhu, D. Wu, Towards discovering and understanding unexpected hazards in tailoring antivirus software for android, in: Proceedings of the 10th ACM Symposium on Information, Computer and Communications Security, ASIA CCS '15, ACM, New York, NY, USA, 2015, pp. 7–18.

[11] D. Quarta, F. Salvioni, A. Continella, S. Zanero, Toward systematically exploring antivirus engines, in: Detection of Intrusions and Malware, and Vulnerability Assessment, Springer International Publishing, Cham, 2018, pp. 393–403.

[12] H. Huang, C. Zheng, J. Zeng, W. Zhou, S. Zhu, P. Liu, S. Chari, C. Zhang, Android malware development on public malware scanning platforms: A large-scale data-driven study, in: 2016 IEEE International Conference on Big Data (Big Data), 2016, pp. 1090–1099.

[13] J. Oberheide, E. Cooke, F. Jahanian, Rethinking antivirus: Executable analysis in the network cloud., in: HotSec, 2007.

[14] M. Cukier, I. Gashi, B. Sobesto, V. Stankovic, Does malware detection improve with diverse antivirus products? an empirical study, in: 32nd International Conference on Computer Safety, Reliability and Security. IEEE, 2013.

[15] P. Bishop, R. Bloomfield, I. Gashi, V. Stankovic, Diverse protection systems for improving security: a study with antivirus engines.

[16] P. Bishop, R. Bloomfield, I. Gashi, V. Stankovic, Diversity for security: A study with off-the-shelf antivirus engines, in: 2011 IEEE 22nd International Symposium on Software Reliability Engineering, 2011, pp. 11–19.

[17] I. Gashi, B. Sobesto, S. Mason, V. Stankovic, M. Cukier, A study of the relationship between antivirus regressions and label changes, in: 2013 IEEE 24th International Symposium on Software Reliability Engineering (ISSRE), 2013, pp. 441–450.

[18] A. Kantchelian, M. Tschantz, S. Afroz, B. Miller, V. Shankar, R. Bachwani, A. Joseph, J. Tygar, Better malware ground truth: Techniques for weighting anti-virus vendor labels, in: Proceedings of the 8th ACM Workshop on Artificial Intelligence and Security, AISec '15, ACM, New York, NY, USA, 2015, pp. 45–56.

[19] P. Du, Z. Sun, H. Chen, J. Cho, S. Xu, Statistical estimation of malware detection metrics in the absence of ground truth, IEEE Transactions on Information Forensics and Security 13 (12) (2018) 2965–2980.

[20] Y. Zhou, X. Jiang, Dissecting android malware: Characterization and evolution, in: Proc. of Symp. Security and Privacy, 2012, pp. 95–109.

[21] G. Suarez-Tangil, J. E. Tapiador, P. Peris-Lopez, J. Blasco, Dendroid: A text mining approach to analyzing and classifying code structures in android malware families, Expert Systems with Applications 41 (4, Part 1) (2014) 1104 – 1117.

[22] M. Zheng, M. Sun, J. C. S. Lui, Droid analytics: A signature based analytic system to collect, extract, analyze and associate android malware, in: Trust, Security and Privacy in Computing and Communications (TrustCom), 2013 12th IEEE International Conference on, 2013, pp. 163–171.

[23] L. Deshotels, V. Notani, A. Lakhotia, Droidlegacy: Automated familial classification of android malware, in: Proceedings of ACM SIGPLAN on Program Protection and Reverse Engineering Workshop 2014, PPREW'14, ACM, New York, NY, USA, 2014, pp. 3:1–3:12.

[24] A. H. Lashkari, A. F. A. Kadir, H. Gonzalez, K. F. Mbah, A. A. Ghorbani, Towards a network-based framework for android malware detection and characterization, in: Proceeding of the 15th international conference on privacy, security and trust, 2017.

[25] B. Ren, C. Liu, B. Cheng, J. Guo, J. Chen, Mobisentry: Towards easy and effective detection of android malware on smartphones, Mobile Information Systems 2018.

[26] C. Yang, J. Zhang, G. Gu, Understanding the market-level and network-level behaviors of the android malware ecosystem, in: 2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS), IEEE, 2017, pp. 2452–2457.

[27] F. Maggi, A. Bellini, G. Salvaneschi, S. Zanero, Finding non-trivial malware naming inconsistencies, in: International Conference on Information Systems Security, Springer, 2011, pp. 144–159.

[28] A. Mohaisen, O. Alrawi, Av-meter: An evaluation of antivirus scans and labels, in: S. Dietrich (Ed.), Detection of Intrusions and Malware, and Vulnerability Assessment, Springer International Publishing, Cham, 2014, pp. 112–131.

[29] F. Wei, Y. Li, S. Roy, X. Ou, W. Zhou, Deep ground truth analysis of current android malware, in: M. Polychronakis, M. Meier (Eds.), Detection of Intrusions and Malware, and Vulnerability Assessment, Springer International Publishing, Cham, 2017, pp. 252–276.

[30] M. Sebastián, R. Rivera, P. Kotzias, J. Caballero, Avclass: A tool for massive malware labeling, in: International Symposium on Research in Attacks, Intrusions, and Defenses, Springer, 2016, pp. 230–253.

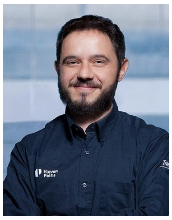
[31] M. Hurier, G. Suarez-Tangil, S. Dash, T. Bissyandé, Y. Le Traon, J. Klein, L. Cavallaro, Euphony: Harmonious unification of cacophonous anti-virus vendor labels for android malware, in: Proceedings of the 14th International Conference on Mining Software Repositories, MSR '17, 2017, pp. 425–435.

[32] I. Martín, J. A. Hernández, S. de los Santos, SignatureMiner: A fast anti-virus signature intelligence tool, in: 2018 IEEE Conference on Communications and Network Security (CNS), IEEE, 2018, pp. 1–2.

[33] M. Girvan, M. Newman, Community structure in social and biological networks, Proceedings of the national academy of sciences 99 (12) (2002) 7821–7826.

Ignacio Martín completed the BEng. in Telematics Engineering and the MSc. Cybersecurity at Universidad Carlos III de Madrid in 2014 and 2015 respectively. At present, he is pursuing the Ph.D. degree at Universidad Carlos III de Madrid in the areas of Data Analytics applied to computer security.

José Alberto Hernández completed the MEng. in Telecommunications Engineering at Universidad Carlos III de Madrid (Madrid, Spain) in 2002, and the Ph.D. degree in Computer Science at Loughborough University (Leics, United Kingdom) in 2005. At present, he is a senior lecturer at Universidad Carlos III de Madrid, where he has published over 80 articles in both journals and conference in-proceedings. His research interests include computer and communication networks, big data technologies, data science and data analytics and machine learning techniques applied to big data environments.



Sergio de los Santos is currently Innovation and Labs Leader at ElevenPaths, Telefónica Digital. Sergio holds a degree in IT Engineering, a M.Sc. in Software Engineering and Artificial Intelligence and has been awarded with Microsoft MVP Consumer Security title from 2013 to 2017. Sergio is also an assistant professor in a master of Technology and Communication Security in the University of Seville.

Analisis and Normalization of more than 250k Android related multi-scanner malware signatures using SignatureMiner

In-depthh analysis and categorization of malware families into three categories: Adware, Harmful and Unknown

Clustering of malware families according to their cross-engine relations

Clustering of AV engines according to their detection schemes

Machine Learnning classifier into Adware/harmful with good performance (0.84 F-score).

Insights on AV engine orientation (Adware/harmful) based on logistic regression classifiers