On Fingerprinting of Public Malware Analysis Services

Álvaro Botas[1], Ricardo J. Rodríguez[2], Vicente Matellán[1], Juan F. García[1], M. T Trobajo[1], and Miguel V. Carriegos[1]

[1] Instituto de Ciencias Aplicadas a la Ciberseguridad, Universidad de León, Spain
[2] Centro Universitario de la Defensa, Academia General Militar, Spain

Abstract. Automatic Public Malware Analysis Services (PMAS, e.g. VirusTotal, Jotti, or ClamAV, to name a few) provide controlled, isolated, and virtual environments to analyse malicious software (*malware*) samples. Unfortunately, malware is currently incorporating techniques to recognize execution onto a virtual or sandbox environment; when an analysis environment is detected, malware behaves as a benign application or even shows no activity. In this work, we present an empirical study and characterization of automatic public malware analysis services, considering 26 different services. We also show a set of features that allow to easily fingerprint these services as analysis environments; the lower the unlikeability of these features, the easier for us (and thus for malware) to fingerprint the analysis service they belong to. Finally, we propose a method for these analysis services to counter or at least mitigate our proposal.

Keywords: malware analysis service, analysis-aware malware, sandbox, characterization, unlikeability

1 Introduction

In the last few years, there has been a steady growth (around a 26% increase a year [15]) in the number and complexity of applications with non-legitimate intentions (known as malicious software, or *malware*).

To detect malicious behavior of a binary, we analyse its interactions with the operating system (OS) and network. Since manual analysis is unfeasible (due to the amount of malware), several methods have been historically developed to automatize the process using isolation (using virtual environments and sandboxes).

An Automatic Public Malware Analysis Service (PMAS, e.g. VirusTotal, Jotti, or ClamAV, to name a few) allows any user to upload files to be analysed, returning a report indicating whether they are malicious or not. Note that we indistinguishably use PMAS as singular and plural acronym.

Cybercriminals enhance malware by adding functions to detect when it is being analysed (this kind of malware is called *analysis-aware* or *split personality* *malware* [1,9]) so that it starts behaving benignly when it thinks that is the case. The longer the malware remains undetected, the more likely cybercriminals will get better revenues from it.

In this work, we present an empirical study and characterization of PMAS. We also show a set of features that allows (both us and analysis-aware malware) for easily fingerprinting these services as analysis environments; the lower the variability (unlikeability) of these features, the easier for us (and thus for malware) to fingerprint the analysis service they belong to. Finally, we propose a method for PMAS to counter or at least mitigate our proposal.

The rest of the paper is organized as follows. Section 2 introduces our characterization of PMAS. Then, in Section 3 we describe our methodology and experiments to obtain data from a PMAS server. In section 4 we discuss our preliminary results. In Section 5, we give some countermeasures for PMAS to be able to partially mitigate our proposal. Section 6 reviews related work. Section 7 concludes the paper and states future work.

2 Characterization of Public Malware Analysis Services

In order to characterize each PMAS, we consider the following features:

– *How the suspicious files are sent to the service.* Specify the method used (External application, web form, email, or others).
– *Where the analysis is done.* The service may perform the analysis on its own infrastructure or use other infrastructures (metaservices).
– *How analysis results are given to the user.* Through a report file, a web page, or by email.
– *Price of the service.* We distinguish between free, paid, or freemium.
– *Accepted file types.* Each service may accept different file types and from different OSes.

Table 1 summarizes the characterization of the different services that we have considered (for a given features, "N/A" means not available, i.e., we do not have enough knowledge to state a conclusion). The Microsoft$^+$ entry encompases Microsoft Defender 10, Microsoft Defender 8, Microsoft Defender 7, Microsoft Other, Microsoft Vista, XP, Essentials and Others.

We had initially considered 100 PMAS, although they were reduced to 26 for different reasons. We have only considered free or freemium services. Furthermore, some well-know PMAS as Anubis have been discontinued and evolved to provide service to just large companies or institutions.

3 Experiments

We developed a probe–software specifically to obtain information about the PMAS running it. The software is written in C# and compiled as a Windows

Service	**App send form**	**metaservice**	**Report**	**Report Data Specified**	**Price**	**Document**		
						Accepted Filetype	**Url's**	**Operating System Platform**
360TotalSecurity	Online form, Mail	Yes	Email	N/A	Free	EXE N/A	No	N/A
Jotti	Online Form	Yes	WebPage	Hash Information	Free	EXE, DMG JPG, PDF, APK, . . .	No	Desktop OS Win, MacOSX, Linux Mobile OS
UploadMalware	Online Form	Yes	Webpage	Hash Information	Free	JPG, PDF, APK, . . .	No	Win, MacOSX, Linux Mobile OS
Virustotal	Online Form Mail Public API External App	Yes	Webpage	Depends of the file (platform)	Free	Executables images,doc,flash Apk, ipa office	Yes	Desktop OS Win, MacOSX, Linux
BleepingComputer	Online Form	N/A	No	No	Free	N/A	No	N/A
Roboscan	External app	N/A	Email	Malware or not	Free	Several	No	N/A
Fortinet	Online Form	N/A	Email	N/A	Free	PEx	No	Windows
Microsoft^{+}	Online Form	N/A	Webpage	Malware or not	Free	PEs	No	Windows
Bitdefender	Online Form	N/A	Email	N/A	Free	PDF, flash, office PEs, DMG APK, Flash, OFFICE	Yes	Multiplatform
Sophos	Online Form	N/A	Email	N/A	Free	Several	Yes	Multiplatform
F-Secure	Online Form	No	Email	Malware or not	Free	PEs, DMG	Yes	Desktop OS
Avira	Online Form	No	Email Webpage	Malware or not	Free	EXE, DMG	Yes	Desktop OS
PayloadSecurity	Online Form	No	Webpage	Created files paths Keychan, File Details Network details	Freemium	PE, Office, PDF APK files and more (e.g. EML)	Yes	Windows Android
McAfee	External app Email	No	N/A	N/A	Free	PEs	No	Windows
Malwr	Online Form	No	Webpage	Created files paths Registry Network activity	Free	PEs PDF, Office, ASCII Flash, Java	No	Windows
ClamAv	Online Form	No	Email	N/A	Free	Several	No	N/A
Valkyrie	Online Form	No	Webpage	Created files paths Registry PE Imports	Freemium	PEs	Yes	Windows
Zoner	Online Form	No	Email	N/A	Free	N/A	No	Multiplatform
ThreatAnalyzer	Online Form	No	WebPage File	Several	Paid	Several	No	Windows (Let you choose versions)
SonicWall	Online Form	No	N/A	N/A	Free	N/A	No	N/A
AVG	Online Form	No	N/A	N/A	Free	Several	No	Multiplatform
Symantec	Online Form External app	No	N/A	N/A	Free	Several	Yes	Multiplatform
Nanoav	Online Form	No	Email	Malware or not	Free	N/A	No	N/A

Table 1. Characterization of Public Malware Analysis Systems.

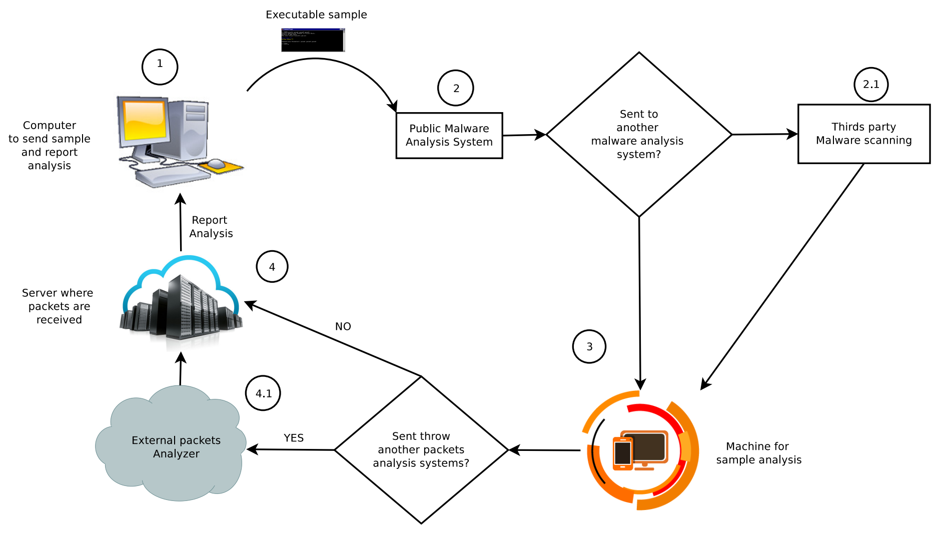

Executable sample
1
Computer to send sample and report analysis
2
Public Malware Analysis System
Sent to another malware analysis system?
2.1
Thirds party Malware scanning
Report Analysis
4
Server where packets are received
NO
3
4.1
External packets Analyzer
YES
Sent throw another packets analysis systems?
Machine for sample analysis

Fig. 1. Workflow diagram to collect PMAS information.

executable file, so we discarded PMAS that focus on runnables from OS other than Windows.

During execution, network packages are sent to a web server under our control to collect the data. The workflow diagram to collect data is shown in Fig. 1. Its steps are:

1. Our probe is specifically crafted for a given PMAS, to which it is sent.
2. The PMAS can either run the probe itself or send it to another external service for analysis.
3. While the sample is being analysed in the PMAS (or in an external service), it is sending network packages to our server. These packages contain information gathered from the PMAS that is analyzing it.
4. Requests are collected in our server. We are continuously receiving network packages sent from different PMAS, so they must be classified by their PMAS of origin.
5. Finally, data is parsed and analysed.

We initially sent our probe to 70 services, but we only received responses from 26 services. Response dataset dates from April 2016 to January 2017. Multiple responses were obtained from a reduced set of services, so some services executed the probe multiple times. Fig. 2 shows the response ratio of received over sent probes: Although in some cases it is uncertain whether the PMAS is a metaservice (a service that uses external engines to analyse malware), we assumed that it might be a metaservice when that response ratio is greater than 5.

Metaservices are shown on the left in Fig. 2 (red bars). Services which do not clearly indicate whether they rely on external engines are shown in the middle (yellow bars). Finally, services using their own engines are shown on the right (blue bars).

3.1 Machine Characterization

We characterize each PMAS considering both physical level (hardware) and logical level (software). Specific features that we collected are depicted in Fig. 3.

Regarding software, we distinguish between *static data* (i.e., data that are not modified at runtime) and *dynamic data*. As dynamic data, we distinguish elements belonging to *Internet connectivity*, such as*open ports*, *Local IP address*, and *external IP address*. Data of interest as *ISP* or *country* can be extracted from these elements. As static part, we consider different elements: *environment variable paths*, *paging file*, and *OS-related data* (such as *Product ID*, i.e., the Windows product identifier; *Build Lab*, i.e., the Windows version; or *Build Lab Ex*, i.e., extended information of Windows version). Respecting the OS's user account, we collected the *registered owner*, *user domain*, *logon server* (that is, network machine name), *Profile Guid*, and *user name*.

In respect of hardware, we consider *hard drive serial number*, *BIOS date*, *RAM size*, *CPU processors number*, *processor model*, and *MAC address* of the machine.

3.2 Unlikebility

We want to classify PMAS according to the variability that they introduce in the series of features we have analysed in isolated environments (read: machines) they use for application analysis.

On the one hand, we consider "good services" those with "concealment" capability, that is, the ones that have a lot of variability in those characteristics for their different analysis machines. On the other hand, those PMAS with low variability will be considered "Bad" given their poor capacity of "concealment".

Consider for instance a PMAS where all the machines they use for analysis have the same user name, or the same IP address (or address pattern), which at the same time is different from the user name or IP of other PMAS: It would be trivial for the malware to detect that it is being executed in a PMAS (the PMAS would have low "concealment" capability, if any at all), so that we would consider that service "bad".

In order to measure the variability within the set of categorical variables, relative frequencies were obtained for the different values obtained from the PMAS of each variable. The full list of variables studied is: MAC, User Name, IP Local, CPU, Profile Guid, Number of Processors, Total of paging file, Total of physical memory, HardDrive Serial Number, Build Lab, Build Lab Ex, Product ID, Registered Owner, System Bios Date, Logon Server, User Domain, User Profile. Finally, variability was computed by the unlikeability index [7]:

$$u = 1 - \sum p_i^2 = \sum p_i(1 - p_i) = 2\sum_{i<j} p_i p_j$$

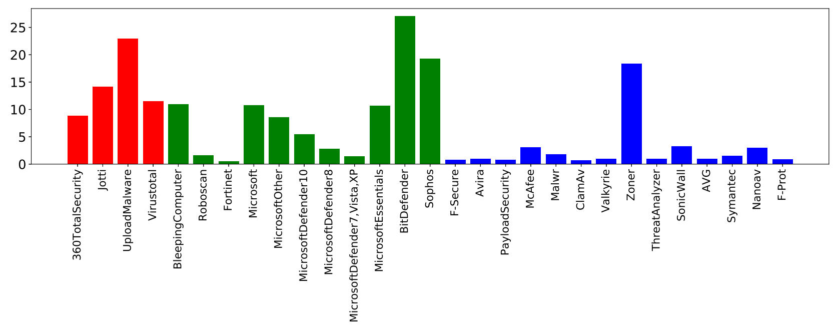


Fig. 2. Ratio of samples received for each probe sent.

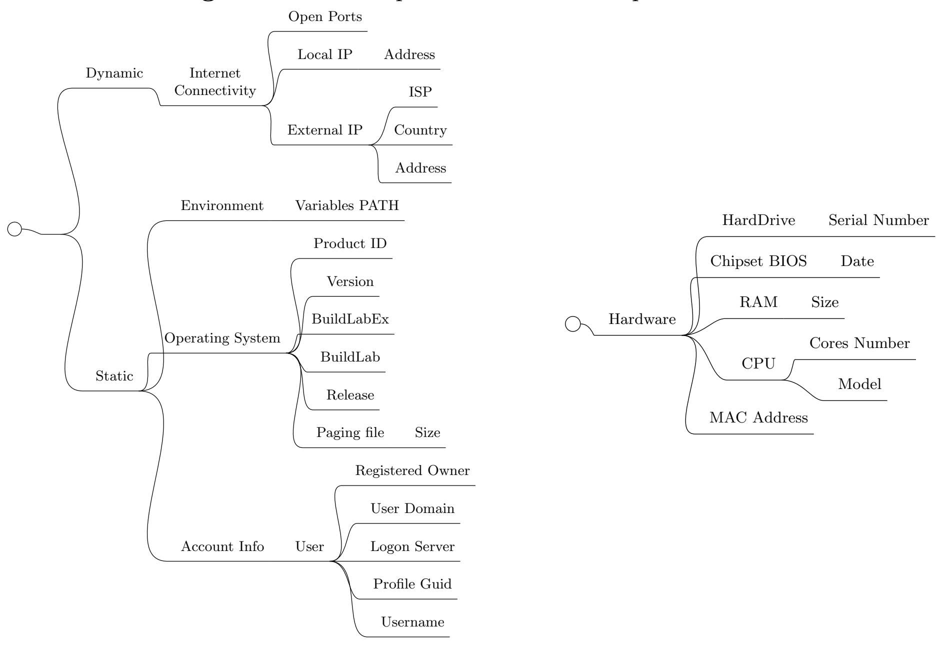
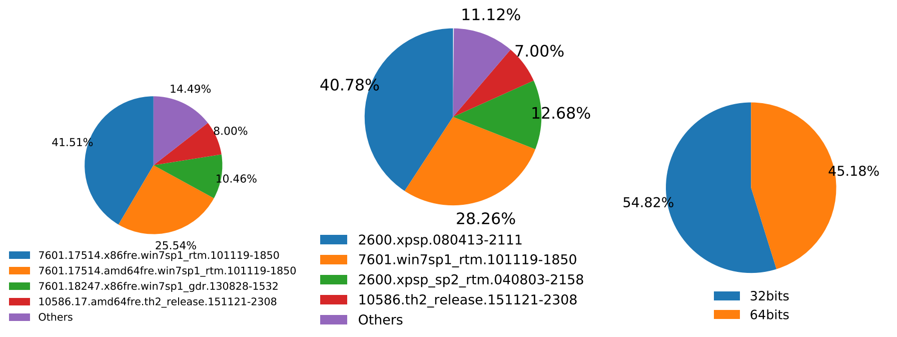


Fig. 3. Software (left-hand) and hardware (right-hand) characterization features.



(a) *BuildLabEx* (b) *Build Lab* (c) OS arquitecture width

Fig. 4. Characterization of (a, b) Windows versions and (c) OS architecture width.

Where $p_i = \frac{n_i}{n}$ is the relative frequency for i-th level of the variable.

We analysed each PMAS individually, obtaining Unlikebility for each variable for that specific PMAS.

We also calculated Unlikebility for each PMAS as a whole, considering the median and mean of all the values obtained for their variables.

We make these calculation both considering and discarding the empty values (NaN). Sometimes the Service does not return a value when queried for it; for those cases, we labeled the query results as "NaN" (from "Not a number").

An Unlikebility value close to 1 represents a high variability or "dissimilarity" and a value close to 0 represents a low variability or "dissimilarity", so the closer to 0 a variable is for an specific PMAS, the worse it is (concealment–wise); the same goes for the PMAS as a whole if we consider (the median and/or mean of) all the values obtained for its variables.

4 Results

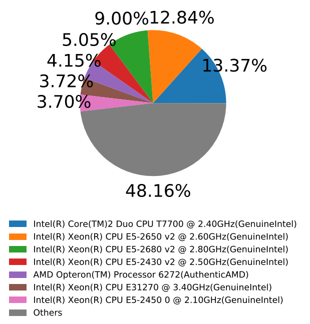
In this studio we have analysed responses from 26 different PMAS. The dataset contains data from 7680 responses to 1500 requests sent. We got more responses than requests due either to some PMAS analyzing the probes several times or to metaservices receiving reports from the external service used.

Responses containing no value regarding features of interest were discarded (see Section 3.1). This happened when: (i) the response was unable to reach our server; (ii) the execution of the file was stopped prior to finishing its activity (i.e., timeout or exception occurs); or (iii) the PMAS does not allow to access to those values (or are unavailable for that particular OS version of the PMAS). Finally, we normalized responses to have an equal response distribution among PMAS.

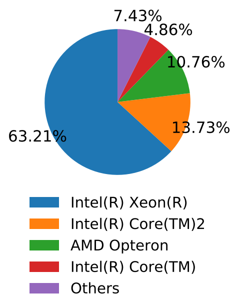
We briefly discuss the most relevant results of our experiments.

Fig 4 shows results of OS-related data. Although 20.22% of responses were obtained (*BuildLabEx*), we observe a prevalence of Windows 7 as OS. Let us remark here that for certain OSes, as Windows XP, this value cannot be obtained. In this regard, the field of *BuildLab* becomes more useful (46.22% of responses obtained). Surprisingly, Windows XP is found in different flavors (with SP1 or SP2 installed) in little more than half of responses, and Windows 7 SP1 is more than 25%. Data coming from *BuildLabEx* was used to infer the OS architecture, with 32-bit and 64-bit almost equally distributed.

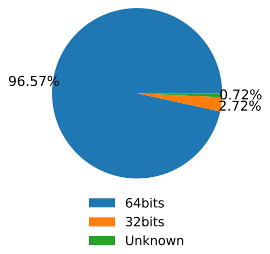
Fig.5 plots the characterization of CPUs used by PMAS (we just received a 16.65% of responses), considering CPU model, family, and architecture width. Results show a prevalence of QEMU-based virtualization systems. For instance, *Intel Core i7 9xx (Nehalem Class Core i7) (GenuineIntel)* is one of the CPU models that can be chosen when configuring a QEMU virtual machine. Regarding CPU family, we observed a high ratio of Intel Xeon technology. In respect of CPU architecture width, more that 95% are 64-bit CPUs, although the OS architecture was just a half when considering data coming from *software*.



(a) CPU model

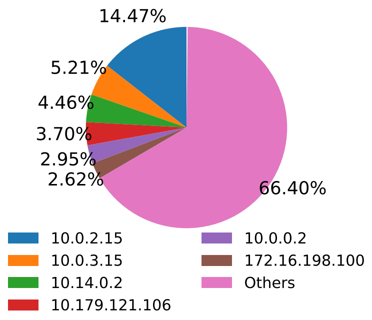


(b) CPU family

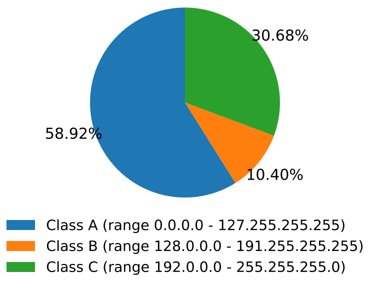


(c) CPU arquitecture width

Fig. 5. Characterization of CPUs of PMAS.

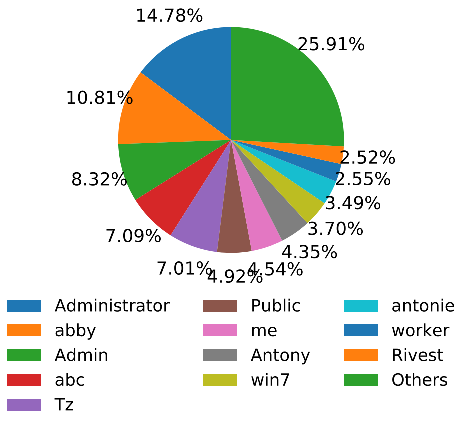


(a) Local IP address



(b) IP address classes

Fig. 6. Characterization of IP addresses of PMAS.



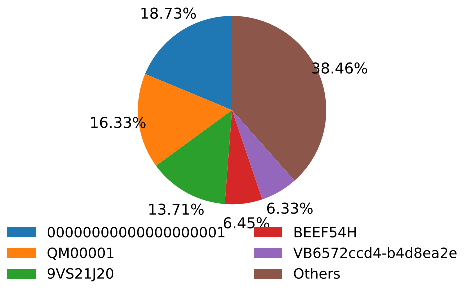


Fig. 8. Distribution of *HardDrive serial number*.

Fig. 7. Distribution of *Username*.

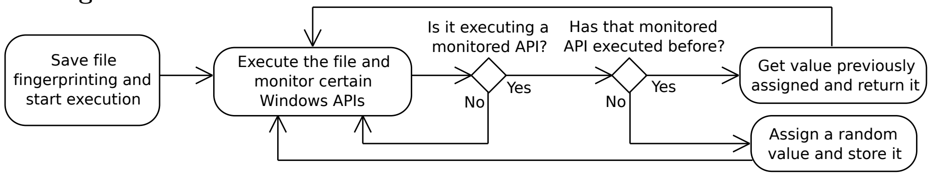

Save file fingerprinting and start execution
Execute the file and monitor certain Windows APIs
Is it executing a monitored API?
Yes
No
Has that monitored API executed before?
Yes
No
Get value previously assigned and return it
Assign a random value and store it

Fig. 9. Proposed countermeasures: Theoretical model to evade PMAS fingerprinting.

Regarding *Internet connectivity*, we had 86.49% responses. Results of characterization of local IP addresses are shown in Fig. 6. The three most common IP addresses accounted for more than a 25%. If we consider IP address classes, class A (usual in business or non-domestic environments) is the most common.

In respect of *username*, we obtained 49.33% valid responses. Fig. 7 shows the distribution of these values. Note that *Administrator*, *abby*, and *Admin* represents more than 50% of collected usernames.

In regard to *HardDrive serial number*, we obtained 57.06% valid responses. Results are plotted in Fig. 8. Note that *00000000000000000001*, *QM00001*, and *9VS21J20* represent almost 50% of collected values.

4.1 Unlikeability results

Variable	Unlikebility NaN	Unlikebility No NaN
MAC	0.848	0.848
User Name	0.076	0.000
IP Local	0.026	0.000
CPU	0.373	0.339
Profile Guid	0.026	0.000
Number of Processors	0.026	0.000
Total of paging file	0.373	0.339
Total of physical memory	0.373	0.339
HardDrive Serial Number	0.963	0.963
Build Lab	0.051	0.000
Build Lab Ex	0.051	0.000
Product ID	0.000	1.000
Registered Owner	0.051	0.000
System Bios Date	0.051	0.000
Logon Server	0.026	0.000
User Domain	0.026	0.000
User Profile	0.026	0.000

Table 2. Unlikebility for variable at Valkyrie PMAS

First, we obtained results for individual analysis of each PMAS. We would like to point out Valkyrie PMAS values (see table 2). This is one of the most relevant cases because it is a not-so-good at concealment Service.

Then, we calculated unlikeability for each PMAS as a whole (see table 3). Despite having yet to establish a threshold for unlikeability in order to classify PMAS as either good and bad, it follows that PMAS like Valkyrie are not good at hiding themselves (its unlikeability is close to zero).

Upon further inspection of the raw data we obtained for Valkyrie, we realized its low unlikeability was due to it having all its hard drives starting with VB... which suggests that they only use Virtual Box. Also, at the time of this

Service	Median (NaN)	Median (no NaN)	Mean (NaN)	Mean (no NaN)
Valkyrie	0.051	0.000	0.199	0.225
Emsisoft	0.000	1.000	0.221	0.659
ClamAv	0.420	0.420	0.442	0.507
MicrosoftDefender	0.361	0.000	0.454	0.197
McAfee	0.402	0.624	0.501	0.536
Bitdefender	0.524	0.463	0.527	0.416
F-Secure	0.538	0.565	0.530	0.423
Malwr	0.681	0.611	0.586	0.620
PayloadSecurity	0.519	0.411	0.591	0.500
Fortinet	0.670	0.548	0.594	0.578
Symantec	0.564	0.891	0.633	0.847
Nanoav	0.613	0.883	0.640	0.843
MicrosoftOther	0.614	0.855	0.648	0.783
Avira	0.680	0.816	0.666	0.783
MicrosoftDefender10	0.659	0.858	0.681	0.776
Roboscan	0.684	0.904	0.682	0.836
360TotalSecurityr	0.669	0.905	0.684	0.865
MicrosoftEssentials	0.719	0.871	0.692	0.809
BleepingComputer	0.683	0.912	0.697	0.878
Microsoft	0.684	0.852	0.701	0.787
AVG	0.680	0.904	0.701	0.856
MicrosoftDefender8	0.687	0.839	0.702	0.765
SonicWall	0.740	0.893	0.712	0.857
UploadMalware	0.733	0.906	0.720	0.863
Virustotal	0.723	0.900	0.722	0.858
Jotti	0.718	0.893	0.723	0.855
Sophos	0.771	0.869	0.770	0.842

Table 3. Median and Mean of Unlikebility of each PMAS as a whole (considering and ignoring NaN values)

analysis, they always use a certain specific model of CPU, RAM, or size of the paging file, among others.

5 Proposed countermeasures

Even if achieving complete evasion of automatic malware analysis systems (and thus never being recognized as malware) is probably an unachievable goal, the aforementioned features have frequently repeated values, hence allowing malware developers to easily fingerprint and evade a PMAS.

A theoretical countermeasure model for automatic malware analysis system to use in order to prevent malware from evading detection using our PMAS fingerprinting method can be seen in Fig. 9.

To collect PMAS features using our method, malware needs to execute certain Windows APIs. Thus, we propose PMAS to monitor those APIs calls and return random values (instead of the real ones) every time these APIs calls are executed.

If a registered API call (like Username or CPU Value) is detected during execution, the system check if it has been executed before: It it hasn't, it returns a random value and stores it in a database; if it has, it retrieves the previously stored value and returns it again.

The value returned (while initially random) has to remain the same for every execution, since otherwise malware can easily detect its randomness and associate this odd behavior with a PMAS – a regular machine wouldn't return random values for the same API call during different executions – which is exactly what the PMAS wants to prevent.

In order for each malware application to always receive the same random value, PMAS should fingerprint each binary file (e.g., using its MD5 hash) before analyzing it. PMAS can then keep track of results given to a specific app and always return the same (and random for the first call) results to it. The registration of this random signature for the malware must be done before beginning its analysis.

If the analysis system implements this model, our method efficacy to recognize a service would be impaired, since many data belonging to the malware developer could not be obtained.

6 Related Work

There are several works in the literature regarding the presence (or absence) of an analysis system. Numerous techniques for virtual machines and emulators detection were provided and qualitatively compared in [6, 12]. Methods to fingerprint anti-virus emulators were introduced in [2].

In [3], static analysis was used to detect anti-analysis techniques. Similarly, dynamic analysis was used in [8] to detect anti-analysis signatures based on sequences of system calls over malware samples.

Detection of anti-analysis malware was performed in [4] comparing the binary execution between real and virtual environments. However, no insights about analysis-aware techniques used by the malware were provided. Different methods to detect Cuckoo Sandbox, an open-source tool to create a sandbox environment for statically and dynamically analyzing binaries, were provided in [5].

A taxonomy and non-exhaustive survey on techniques used for detecting analysis environments were introduced in [13]. Besides, a tool was provided to trick an anti-analysis malware sample running on a virtual environment as running on a real environment. A recent study on detection of anti-analysis malware has been recently provided in [16], in which anti-analysis signals are divided into weak signals, strong signals, and composite signals, depending on those signals are normally used by a benign software or by a malware.

Other techniques for detecting execution on hypervisors using time information were provided in [11, 17]. Detection based on unexpected semantics when executing certain CPU instructions was also proposed in [14].

Recently, the FFRI malware dataset was used in [10] to evaluate the most used analysis detection methods. Their approach was based on analyzing Windows APIs calls and results were used to improve Cuckoo Sandbox system.

Regarding PMAS fingerprinting, it is worth mentioning [18]. They evaluated 15 PMAS, being able to fingerprint them using as feature the IP source address of network responses. Our work is similar to theirs, but we perform a more exhaustive analysis of features to fingerprint PMAS.

7 Conclusions and Future Work

Malware have increased in number and complexity during last decade. PMAS are services that allow users to test for malicious behavior. To avoid being detected, malware introduces techniques that recognize these analysis environments, hide its malicious intention and behave as benign software

In this paper we have performed an empirical study to characterize 26 PMAS with different features (file uploading method, analysis location, report procedure, price, and accepted file types). Then, we have developed an application to fingerprint the PMAS with just a few features (such as OS, CPU, or username). Features having lower unlikeability index are easier to be fingerprinted for users and thus for malware. Finally, we proposed a (theoretical) execution model to evade PMAS fingerprintg.

As future work, we aim at identifying the relationships between different PMAS, as well as fingerprinting each PMAS in detail. Furthermore, we aim at implementing and evaluating our evasion model.

Acknowledgments: A. Botas, V. Matellán, J. F. García, and Miguel V. Carriegos work was partially supported by the Spanish National Cybersecurity Institute (INCIBE) [addendum 18] and by Universidad de León [contract key X43]. R. J. Rodríguez work was partially supported by the Aragonese Government under Programa de Proyectos Estratégicos de Grupos de Investigación [project number T21-17R], in part by the Centro Universitario de la Defensa-Zaragoza [project number CUD-2018-09], and in part by the Spanish MINECO project MEDRESE [project number RTI2018-098543-B-I00].

References

1. Balzarotti, D., Cova, M., Karlberger, C., Kirda, E., Kruegel, C., Vigna, G.: Efficient Detection of Split Personalities in Malware. In: NDSS 2010 (2010)
2. Blackthorne, J., Bulazel, A., Fasano, A., Biernat, P., Yener, B.: AVLeak: Fingerprinting Antivirus Emulators through Black-Box Testing. In: 10th USENIX Workshop on Offensive Technologies. USENIX Association (2016)
3. Chen, P., Huygens, C., Desmet, L., Joosen, W.: Advanced or Not? A Comparative Study of the Use of Anti-debugging and Anti-VM Techniques in Generic and Targeted Malware. In: IFIP SEC 2016. pp. 323–336 (2016)
4. Chen, X., Andersen, J., Mao, Z.M., Bailey, M., Nazario, J.: Towards an understanding of anti-virtualization and anti-debugging behavior in modern malware. In: DSN 2008. pp. 177–186 (June 2008)
5. Ferrand, O.: How to detect the Cuckoo Sandbox and to Strengthen it? Journal of Computer Virology and Hacking Techniques 11(1), 51–58 (2015)
6. Garfinkel, T., Adams, K., Warfield, A., Franklin, J.: Compatibility is Not Transparency: VMM Detection Myths and Realities. In: 11th USENIX Workshop on Hot Topics in Operating Systems. pp. 6:1–6:6. USENIX Association (2007)
7. Kader, G.D., Perry, M.: Variability for categorical variables. Journal of Statistics Education 15(2) (2007)
8. Kirat, D., Vigna, G.: MalGene: Automatic Extraction of Malware Analysis Evasion Signature. In: 22nd ACM SIGSAC Conference on Computer and Communications Security. pp. 769–780. ACM (2015)
9. Kumar, A.V., Vishnani, K., Kumar, K.V.: Split Personality Malware Detection and Defeating in Popular Virtual Machines. In: SIN '12. pp. 20–26. ACM (2012)
10. Oyama, Y.: Trends of anti-analysis operations of malwares observed in API call logs. Journal of Computer Virology and Hacking Techniques pp. 1–17 (2017)
11. Pék, G., Bencsáth, B., Buttyán, L.: nEther: In-guest Detection of Out-of-the-guest Malware Analyzers. In: EUROSEC'11. pp. 3:1–3:6. ACM (2011)
12. Raffetseder, T., Krügel, C., Kirda, E.: Detecting System Emulators. In: 10th International Conference on Information Security. pp. 1–18 (2007)
13. Rodríguez, R.J., Rodríguez-Gastón, I., Alonso, J.: Towards the Detection of Isolation-Aware Malware. IEEE Latin America Transac 14(2), 1024–1036 (2016)
14. Shi, H., Alwabel, A., Mirkovic, J.: Cardinal Pill Testing of System Virtual Machines. In: 23rd USENIX Security Symposium. pp. 271–285 (2014)
15. Symantec: ISTR - Internet Security Threat Report. Tech. rep. (2016)
16. Tan, J.W.J., Yap, R.H.C.: Detecting Malware Through Anti-analysis Signals - A Preliminary Study. In: CANS 2016. pp. 542–551. Springer (2016)
17. Wang, G., Estrada, Z.J., Pham, C., Kalbarczyk, Z., Iyer, R.K.: Hypervisor Introspection: A Technique for Evading Passive Virtual Machine Monitoring. In: 9th USENIX Workshop on Offensive Technologies. USENIX Association (2015)
18. Yoshioka, K., Hosobuchi, Y., Orii, T., Matsumoto, T.: Your Sandbox is Blinded: Impact of Decoy Injection to Public Malware Analysis Systems. Journal of Information Processing 19, 153–168 (2011)