A Close Look at a Daily Dataset of Malware Samples

XABIER UGARTE-PEDRERO, Cisco Systems, Inc.
MARIANO GRAZIANO, Cisco Systems, Inc.
DAVIDE BALZAROTTI, Eurecom, France

The number of unique malware samples is growing out of control. Over the years, security companies have designed and deployed complex infrastructures to collect and analyze this overwhelming number of samples. As a result, a security company can collect more than 1M unique files per day only from its different feeds. These are automatically stored and processed to extract actionable information derived from static and dynamic analysis. However, only a tiny amount of this data is interesting for security researchers and attracts the interest of a human expert.

To the best of our knowledge, nobody has systematically dissected these datasets to precisely understand what they really contain. The security community generally discards the problem because of the alleged prevalence of uninteresting samples.

In this paper, we guide the reader through a step-by-step analysis of the hundreds of thousands Windows executables collected in one day from these feeds. Our goal is to show how a company can employ existing state-of-the-art techniques to automatically process these samples, and then perform manual experiments to understand and document what is the real content of this gigantic dataset. We present the filtering steps and we discuss in detail how samples can be grouped together according to their behavior in order to support manual verification. Finally, we use the results of this measurement experiment to provide a rough estimate of both the human and computer resources that are required to get to the bottom of the *catch of the day*.

CCS Concepts: • **Security and privacy → Malware and its mitigation**; *Software reverse engineering*; *Economics of security and privacy*;

ACM Reference Format:
Xabier Ugarte-Pedrero, Mariano Graziano, and Davide Balzarotti. 2018. A Close Look at a Daily Dataset of Malware Samples. *ACM Trans. Priv. Sec.* 1, 1, Article 1 (January 2018), 30 pages. https://doi.org/10.1145/3291061

1 INTRODUCTION

The annual reports published by security companies show a clear and unstoppable trend in the number of unique malicious executables. According to antivirus companies, this number went from less than one million [2] in 2008 to over one billion [3] in 2014. VirusTotal, which acts as a de-facto central collection point in the field, routinely reports in its daily statistics over two million unique submissions per day [7]. A comparable amount of files is also collected by large security companies through their many different sources and client telemetry data [5]. Many of these files are then shared through malware feeds. Some of these feeds [6] are private, while others are open to the public. Security companies can settle agreements and subscribe to these feeds in order to obtain samples that can afterwards be analyzed in order to obtain actionable information. Some companies may just want to extract Indicators of Compromise (IOC) to help protect their customers. Others

Authors' addresses: Xabier Ugarte-Pedrero, Cisco Systems, Inc. xabipedr@cisco.com; Mariano Graziano, Cisco Systems, Inc. magrazia@cisco.com; Davide Balzarotti, Eurecom, France, davide.balzarotti@eurecom.fr.

2471-2566/2018/1-ART1 $15.00
https://doi.org/10.1145/3291061

might be interested in extracting valuable intelligence about ongoing campaigns or actors. Finally, some companies might just want to make sure that they can properly detect the samples and make sure that their signature databases and heuristic engines are tuned and work correctly.

However, despite the massive infrastructure dedicated to collect, store, and analyze this incredible amount of information, there is still very little understanding of what these datasets really contain. The common belief is that the vast majority of these files are either benign programs infected by a virus or polymorphic variations of a limited number of known malware families. In other words, they are of very limited interest for malware analysts and their overwhelming number is just a burden that greatly complicates the problem of identifying new and more interesting samples. Indeed, a recent study [15] showed that most of the APT-related samples were in fact collected months, or even years before the discovery of the corresponding attacks: while their binaries were available to everyone with a Virus Total key, simply no human analyst had decided to look closely at those samples.

Furthermore, it is unclear which are the challenges that every security company must face on a daily basis in order to dissect the incoming feed of samples and to respond properly and in a timely fashion. One can assume that every company has its own sample ingestion pipeline but, to the best of our knowledge, there is no assesment on how state-of-the-art technology helps to automate the categorization of a *real* feed of samples. Our findings, which are based on systematic measurement, confirm some of the community approximate believes, while they bring some concerns about the current sample feed format. Furthermore, previous approaches have measured the efficiency and efficacy of dynamic and static analysis approaches, but these studies are often evaluated with malware datasets that do not follow the same distribution as a daily malware feed. Our study shows that these results might be affected if evaluated against this type of dataset.

Some days are more busy than others. This phenomenon can be caused by aspects like active malware campaigns, users submitting collections of samples to analysis services, or simple randomness. Even if we accept the fact that polymorphism and file infectors are the root cause of the exponential increase of malicious samples, two fundamental questions remain: *(Q1) Given one day of samples, what is the actual percentage of samples that belongs to known families?* Experts answer this question by simply saying "the vast majority" – but does the majority mean 60%, 90%, or 99.9%? If it would be possible to automatically remove this noise from the dataset, *(Q2) How much effort (in term of computing power and human resources) would be required to manually sift through the remaining samples to identify what they really are?* And finally, even if a large company can dedicate a good amount of resources to process these samples, *(Q3) What are the challenges that a company will encounter when processing sample feeds? How many resources are wasted processing samples with no value at all?*

This paper tries, for the first time, to answer these important questions and put some actual figures on top of our community approximate believes. It does that by looking at a single day in the life of a big security company, and by precisely measuring and classifying the Windows executable binaries that are collected from public and private feeds over a period of 24 hours. This paper tells a story of one day spent fishing for malware samples, and of the months required to sift through the collected files to understand their true nature and their possible value for our community. Even if the answer to question Q1 may differ from one day to another, and from one year to the next, our study provides a real measurement on which we can base our hypotheses and conduct future research. Moreover, we selected the day of the week which showed the highest number of samples. It is important to notice that the process we followed in order to filter out and select the samples that deserve some interest, as well as the analysis done of the collected samples, was designed as a systematic approach and does not necessarily represent the actual workflow of the security

company that provided the data. In fact, different security companies may have different interests – therefore focusing on different subsets of samples, as each security product or service may be specialized on specific types of threats.

This paper makes the following contributions:

- We dissect the samples collected in one day from public and private feeds by a security company and document the composition of this dataset.
- Then, we take a closer look at Windows executables and measure the results provided by state-of-the-art static and dynamic analysis tools.
- We apply clustering techniques in order to group similar sandbox reports together, facilitating sample inspection. This approach also allows us to observe several phenomena about how samples are labeled by AV companies, and the amount of them that lack any characteristic malicious behavior. With this information, we categorize the generated clusters and assign different priorities to them.
- We conduct an experiment with malware analysts in order to understand the effort required to understand what is contained in the dataset.
- Our analysis pipeline allows us to measure the requirements a company has in terms of computational resources and human effort.
- Most importantly, we describe the challenges during the dissection of a sample feed, document some of the limitations of state-of-the-art tools, and the impact this has for a security company.

The rest of this paper is structured as follows. Section 2 collects the most relevant related work on this topic. It is then followed by 2 parts. Part A explains how we collected the data, and reports different useful statistics about its distribution. With this section, the reader will understand the distribution of our feed of samples. Part B, in contrast, details the automated filtering steps we applied to this dataset in order to understand the number of samples that deserve some interest, and to prioritize their analysis. Part C describes 3 manual analysis experiments we conducted to understand the resources (in terms of human effort) required in order to sift through the interesting samples. Finally, Part D discusses the findings of this paper and summarizes its main takeaways.

2 RELATED WORK

To the best of our knowledge, this is the first systematic attempt to shed light on the malware samples collected in a single day through public and private sample feeds on a daily basis by a large security company. However, there are several previous studies that have looked at the analysis of large datasets of malicious samples from different perspectives.

For instance, an interesting line of research have focused on the lineage and evolution of different malware families. Jang et al. [20] proposed a method to identify the software lineage by combining both static and dynamic features extracted from a set of binaries on which the author had a ground truth. On the same direction, Lindorfer at al. [27] studied the evolution of 11 malware families over time and proposes a system to identify the changes introduced in the different versions. More recently, Calleja et al. [11] performed an archaeological tour studying the source code of 151 malware samples from 1975 to 2015. The authors described the software and complexity evolution over the years.

In our study, we do not focus on malware evolution. Instead, we take a snapshot of what a large company receives through its different feeds in a single day, and try to understand the composition of the feed as well as to measure how much effort it takes to process it.

In the last decade, researchers have also frequently studied large malware datasets in order to automatically identify either evasive or packed malware. For instance, Lindorfer at al. [28] proposed a system able to isolate environment sensitive malware. The authors executed each malware sample

multiple times on several sandboxes. Each sandbox had been configured with different monitoring implementations in order to spot behavior discrepancies on the normalized reports. Similarly, Pedrero et al. [34] proposed a system to analyze and classify packed binaries based on the packer properties.

In our case, we do not specifically focus on evasive malware, but instead, we measure how many samples from our feed do not show a malicious behavior in a sandbox. A good percentage of these samples are not malicious, or are non-standalone components part of a larger piece of software, are corrupted files, or have unmet dependencies. This has an impact on the resources of a company, which we tried to measure. More in general, we do not investigate subgroups of our dataset like evasive samples, packed or signed binaries. In the literature, other papers already studied in detail these aspects. For instance, some authors have discussed about signed binaries and the underground ecosystem of certificates for code signing in Windows [23, 24]. Other researchers investigated the problem of both benign and malicious singletons [9]. Even though, these studies go in depth on some parts of our dataset, the contribution of this paper is different. We dissect one day of samples collected from diverse feeds. We document a possible pipeline to cope with this large number of samples. We estimate the necessary resources to dissect this dataset and we provided the final number of samples to analyze manually. Moreover, we discuss the main limitation of the current approaches and we provide directions for future research that have a real impact for the security industry.

Another similar research area covered the analysis of existing datasets collected by online sandboxes. The first study in this direction was conducted by Bayer et al. [10]. In this study, the authors analyzed the dataset of the Anubis sandbox collected in the first two years of operation. They provided interesting insights about malware evolution, trends and about the prevalent types of malware submitted to their sandbox. A similar experiment was later repeated by Lindorfer et al. [29] on a dataset of Android malware samples. This line of research is based on datasets collected by online dynamic analysis systems. Although these studies have similarities with this paper, the goals and the nature of the dataset are different. Our dataset is not coming from an online sandbox but it is composed by several feeds. Most security companies make the effort to integrate as many feeds as possible into their pipelines, and there are even public efforts to list publicly available feeds [6].

Other researchers have tried to study malware on other operating systems. This is the case of Cozzi et al. [13]. In this case, the authors analyzed Linux malware samples collected in one year to show the current threat landscape of ELF malware and compare these threats with what we face daily on Windows. This dataset is composed only by samples collected from VirusTotal and not by several feeds. Also, the goal in this case was different from ours, as it was focused on understanding the ELF malware predominance and its techniques.

Lever et al. [26] recently looked at the network communications of 26.8 million malware samples by combining information extracted from dynamic analysis sandboxes with the DNS queries collected from a large North American internet service provider. In our case we investigate the network behavior of the samples executed in our dynamic analysis environments but we did not focus only on the network traces because it was out of the scope of the paper.

Researchers have also investigated the problem of the collection of large malware datasets. In this direction, Canto et al. [12] discusses the challenges of running public and established online services such as VirusTotal, SGNET and Anubis. While the last two services have unfortunately been taken down, they had been available for years providing a valuable contribution to the entire community.

Our work has clearly a different perspective. First, we do not have the problem of collecting malware binaries. We receive on a daily basis fresh samples from our feeds. Second, these studies

do not discuss the challenges and the corner cases of processing malware feeds and the impact on the resources of a company.

Graziano et al. [15] performed instead a large scale analysis of the samples collected by Anubis with the objective of detecting examples of malware development. This paper shares several aspects with our work, as both had to deal with large datasets in an automated fashion and they had to propose aggressive filtering techniques to remove the noise and focus on the more relevant and interesting samples. Similarly, the study conducted by Huang et al. [17] investigated malware developments. The authors focused their attention on Android and used VirusTotal as a dataset. Similarly to this study, they also used aggressive filtering techniques and combined both static and dynamic analysis as well as analysts for results validation.

One of the steps of our approach required to cluster similar samples. In the literature researchers have proposed different techniques to cluster malware datasets [16, 18, 19, 35, 36]. The main goal of all these approaches is to group together samples belonging to the same family – as this is the typical problem of antivirus companies. In this paper we adopt a state of the art technique to identify samples sharing similar static and behavioral features. This approach is tailored to the data extracted by the company's sandbox solution, and it is based on text similarity and standard algorithms present in an existing clustering library. Although we did not investigate potentially novel clustering algorithms, we consider that the contribution of this paper is different and clustering is just a way to group together similar sandbox reports, and to present them in an organized manner to the analyst, as well as to depict the nature of the contents of the feed.

Finally, one of the goals of this paper is to understand the contents of one day of samples, so that malware analysts know in the future which groups of unknown samples they should focus their effort on. Ultimately, the prioritization depends on the current interest of the company and on the number of allocated resources. For example, a company may be interested in increasing their signature coverage for common malware, while another one can consider irrelevant the known malicious files and focus their attention on the clusters that are more likely to contain new examples of sophisticated threats. Generally speaking, the prioritization of a large malware dataset has not yet been studied in detail. Karant et al. [22] presented an approach to identify suspicious Javascript files that are more likely to exploit zero-day vulnerabilities in a drive-by-download scenario. The proposed technique is based on machine learning algorithms tuned to detect zero days in the analyzed data. In another interesting work, Morales [1] proposed to use clustering for prioritizing manual malware analysis. This framework identifies the most malicious clusters and provides a queue with the top malicious samples for the analysts. Similarly, in our research we also resort to clustering as the base of prioritization, but we let the company decide which prioritization strategy best suits its objectives.

PART A – SETUP AND DATA PREPARATION

A1. The Nature of our Dataset

The scope of this paper is to dissect the feed of samples that a security company might process and store in one day. In our case, this includes both public and private feeds. We work under the assumption that the heterogeneity of these feeds does not allow to know in advance the context of each sample (i.e., how it was collected). Although in a small number of cases it might be possible to obtain this information, in most of the cases it is just not possible. For instance, companies or users submitting samples to VirusTotal do not specify how they collected the sample.

Different companies might have different feeds, resulting in a different number of samples and even a potentially different family distribution. Furthermore, many companies do not share the number of samples they collect or the feeds they import, making difficult to compare our dataset

Table 1. Number of samples for each week in the last 3 months of the year

Week	Nb. of samples	Week	Nb. of samples	Week	Nb. of samples
40	4,487,907	41	8,001,208	42	7,561,698
43	7,324,254	44	8,054,180	45	7,584,566
46	7,786,035	47	8,674,714	48	6,145,345
49	6,398,709	50	4,749,192	51	4,874,549
52	5,057,094	53	2,118,189		

with the ones of other companies. Nevertheless, every company importing data from these sources will confront the same problems and limitations, so the conclusions obtained from this study should generalize to any organization. Also, as we will show, the dataset collected for this study is large enough to understand the implications of processing such a number of samples on a daily basis.

Many security companies (including ours) offer security products such as endpoint scanners, mail filters, gateways, firewalls, web application filters, or any other products that may collect telemetry data as well as suspicious samples during their operation. Moreover, nowadays some vendors conduct the analysis in their own infrastructure (i.e., outside of the infrastructure of the customer) allowing to perform a more powerful analysis and to aggregate statistics and indicators of compromise. Several authors have conducted studies based on this telemetry data [25].

Nonetheless, we have not included this telemetry data in our dataset for two reasons. First, this data is not always collected in order to respect the privacy restrictions configured by each customer (e.g., only hashes might be collected, but not the samples themselves). These samples can belong to any file type (such as documents), and contain confidential information. Even if available, this data has a restricted access and it is not shared with other vendors or entities under any circumstances. Second, many security companies may not have this type of data if they do not offer this type of products. For instance, a company might specialize on analysing an specific type of malware and producing reports with aggregated information about specific ongoing campaigns or actors. Such a company might subscribe to one or several malware feeds, but may not have telemetry data to rely on.

For these reasons, we focus our study on the subset of samples that a big security company obtains through sample sharing feeds from up to 17 different entities, including CERTs, public and private malware tracking services, security vendor partners, customer submissions, and VirusTotal.

A2. A Good Day to Go Fishing

As the goal of this paper is to invest a considerable effort to truly understand a *single day* of data, it is important to carefully choose the experiment date.

One of the aspects that we want to measure in our study is the computational and human resources a company may need to process such a dataset. We started our planning by choosing the week with the highest number of captured samples during the last trimester of 2015 in order to do our measurement under non-favourable conditions.

Table 1 summarizes the number of samples collected every week from 1^{st} of October 2015 until the end of the year. We can observe that the most prolific week was week 47 (16^{th} to 22^{th} of November).

The second, but not less important aspect to consider is the possible fluctuations between the different days of the week. It is reasonable to think that, since companies have a lower activity during weekends, the number of samples shared in feeds might change from day to day.

Table 2. Total number of PE files over one year, and average number of PE files

Day of Week	Total (1 year)	Avg PEs
Monday	39,799,691	298,859
Tuesday	41,374,785	304,719
Wednesday	45,829,468	**344,031**
Thursday	44,725,851	338,893
Friday	43,244,266	324,400
Saturday	40,898,046	327,448
Sunday	38,459,952	320,919

We identified a number of factors that may influence the catch of the day. Over a year, the company collected from their feeds, in average, 8.35% more files during week days than over the weekends. This is probably a common phenomenon, due to the fact that some of these feeds might contain samples collected by sensors deployed in enterprise environments, or correspond to manual submissions of suspicious files found during working hours. Second, because of the high number of samples to be processed every day, it can take several hours to propagate them through the different feeds shared between companies. New files are transmitted over the network and sometimes analysed on the fly before being stored in databases or propagated again, often resulting in delays of up to 24h. In fact, as we detail in Section A3., a significant part of the samples we received on our fishing day were first observed by VirusTotal between few hours and one day in advance.

Table 2 shows the number of samples received each day of the week during one year, starting six months before collection date, and ending 6 months after that day. We can observe that the most prolific day of the week over the year was Wednesday.

It is not surprising given that companies may have a higher activity during the week days, and that on that day, the company may still be receiving samples that were added to the feeds at the beginning of the week.

These factors indicate that Wednesdays are the most appropriate day for sample collection because it ensures the highest possible number of samples. Therefore, we set our collection day for Wednesday, November 18.

Finally, one may wonder why it took so long to perform our study and why we did not repeat our experiment on a more recent date. As it will be more clear in the next sections, this study required us to "borrow" a substantial amount of resources from the company – both in terms of scheduling automated analysis tasks on the company infrastructure and also of assigning files to malware analysts for a manual analysis. As a result, in order to reduce the impact of our experiments on the daily company operation, we were requested to distribute these activities over a period of several months.

A3. A First Look into the Fishnet

On November 18, the company collected a total of **1,261,882** new samples from their sample feeds (i.e., previously unknown to the company according to their SHA2 hashes). Our first objective was to understand how "fresh" these files were by using VirusTotal. In fact, since security companies (including the one that provided the data for this study) and research institutions share much of their data with VirusTotal, 90% of the collected samples were present also in the VT platform. This high percentage can also be explained by the fact that some of the feeds that composed our dataset

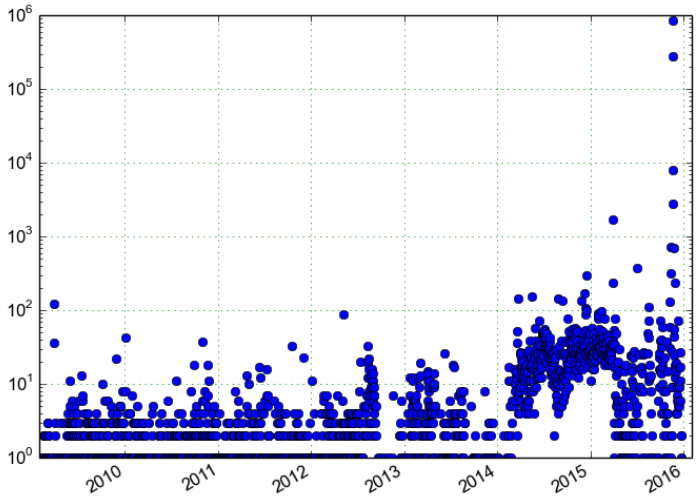

Fig. 1. First seen date in VirusTotal. The X axis represents the date, while the Y axis represents the number of samples first seen on that date, on a logarithmic scale. For clarity, we plot the period from 1^{st} of February 2009 to 1^{st} of February 2016.

were already submitted to VT as part of their collection process. The remaining 10% of the samples were collected from feeds coming from CERTs, universities and research centers. Unfortunately, we were not authorized to share this portion of the samples with external sources.

VirusTotal provides information about the date when a sample was submitted for the first time, which is a good approximation for the "age" of the sample in the wild and how "new" it is for the security community. It is very interesting that 89% of the samples in our catch were first observed either the day before or during the same day. Only 1.6% was already known from previous dates, with few samples dating back as far as June 2006. Figure 1 shows the distribution of first seen dates on a logarithmic scale.

A4. Windows Executables

In order to better understand the content of our dataset, we started by computing the file type of each collected sample using the Linux `libmagic` library. Unfortunately, this information was not accurate enough, especially for PE files that are the main focus of this paper. This required an additional step to parse the PE headers and extract the subsystem information contained in the `IMAGE_OPTIONAL_HEADER`.

The first surprising result was the fact that PE files were less than half of the entire catch of the day. This is because sample feeds include all sort of suspicious documents, ranging from PDF to HTML files, from multimedia files to more generic 'data' blobs. Table 3 shows the variety of PE binaries that have been collected. The second surprise was that the dataset contained a very large number of dynamic libraries (DLLs), and in particular of EFI DLL drivers. When we investigated the origin of these files, we observed that VirusTotal had recently launched a new service to analyze and dissect EFI firmware [4]. We consulted VirusTotal as well, who confirmed the existence of this new service: in the last weeks before our collection date the company massively uploaded firmware images and asked vendors to do the same.

Table 3. PE file subsystems

Subsystem	DLLs	Executables
WINDOWS_GUI	66.327	162.327
EFI_BOOT_SERVICE_DRIVER	214.887	21.201
WINDOWS_CUI	139.246	10.285
EFI_RUNTIME_DRIVER	24.435	3215
NATIVE	92	888
EFI_APPLICATION	781	400
WINDOWS_CE_GUI	113	59
UNKNOWN	28	36
EFI_ROM	17	0
XBOX	3	0
Total	445.929	198.411

Despite the fact that the global threat landscape is in continuous evolution and new forms of malicious files (e.g., ELF binaries for embedded systems [13]) are on the rise, Microsoft Windows PE executables still represent by far the prevalent security threat because of the Microsoft Windows operating system popularity. A modern infection may start by other file types such as PDF documents or web pages, but in most of the cases the last stage of the infection is done by an executable file. For this reason, we limit our analysis to this category and for the rest of the paper (unless specifically stated) we will use the term *sample* or *dataset* to indicate only Microsoft Windows executables of Subsystem 2 (`IMAGE_SUBSYSTEM_WINDOWS_GUI`) and Subsystem 3 (`IMAGE_SUBSYSTEM_WINDOWS_CUI`). Overall, the 172,612 Windows executables accounted for just the 13.7% of the entire catch of the day.

While 60% of the collected samples have a file size between 100K and 1M, there are exceptions at both ends of the spectrum, with the smaller file being only 512 bytes and the largest over 566MB. The vast majority of these files (169,431) are compiled for x86 32 bit systems, 3,156 are compiled for x86-64 bit, three samples for IA64 (Intel Itanium), and 22 for ARM. The dataset contained 89,557 (51%) samples with a section entropy higher than 7, which is a popular indicator of a packed PE file. The number for signed binaries is 31,608 (18.3%) – with 11 of them signed using revoked certificates.

PART B – AUTOMATED FILTERING, ANALYSIS, AND CLASSIFICATION

Even if they may seem few compared to the initial set of 1+ million files, 172K samples are still too many to be analyzed manually. Therefore, the goal of this second part of our study is to discuss how our initial dataset can be processed in a fully-automated pipeline to collect relevant information about each sample.

For this, we first extracted the information provided by VirusTotal reports using the AVClass [33] tool, that assigns a family name (when possible) to each sample. Then, we applied dynamic analysis to extract a behavioral report for every sample, and cluster the reports based on their similarity. Finally, we combined this information. This allows us to understand which groups of similar reports correspond to well defined (and well detected) malware families, which ones contain undetected samples, and which ones contain malware samples that are labeled with generic family names or contradictory names.

There are two important observations about this part. First, it is not our goal to propose new analysis techniques or to improve the state of the art of malware classification. Many papers have

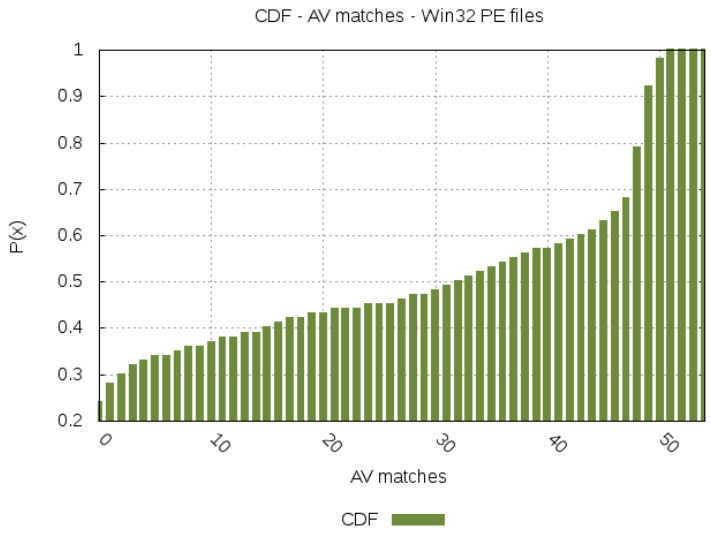

Fig. 2. CDF of AV detection. The X axis represents the number of positives, while the Y axis represents the probability of a PE file of having x positives or less. A 30 % of our dataset has less than 3 positives.

been written on the subject, and our objective is to show what can be achieved by applying state of the art techniques (which are mature enough to be used on a daily basis by a company) on our catch-of-the-day dataset. Similarly, with the designed pipeline we do not intend to propose a new malware classification procedure, but instead, we leverage it to dissect our dataset, and to understand the challenges faced by a company using state-of-the-art technology for the automated processing of its sample feed.

Second, while we are going to rely on proprietary technology (such as the sandbox systems) used by a real company, we are not allowed to discuss what is the current analysis pipeline in use by the company that collected and provided us with the data. So, our goal is to show *a possible* pipeline and discuss the results that can be obtained, and not necessarily to show what the company is doing on a daily basis.

B1. Known Goodware

The problem of identifying known benign files is difficult and prone to mistakes. To be conservative, we leveraged the internal white list used by VirusTotal. Unfortunately, only 25 samples matched the aforementioned white lists, and thus can be considered goodware and removed from further analysis.

B2. Known Malware

After the known goodware have been removed, the second step in our pipeline consists in testing our samples against popular antivirus tools, to understand if existing signatures would detect our files as malicious. The easier way to perform this analysis is by relying on VirusTotal to obtain a complete and aggregated view of the labels assigned by over 50 AV products. Table 4 shows the number of samples grouped according to the number of positive AV matches for our first queries, performed shortly after the collection day. Figure 2 show instead the cumulative distribution of the

Table 4. AV Positives for the collected PE files

	January 2016	July 2016	January 2017 (Re-scan)
0 positives	21,483	21,534	19,221
1 to 4 pos.	15,620	15,582	14,648
5+ positives	126,440	126,598	129,797
Total	163,543	163,714	163,666
Unknown to VT	9,069	8,898	8,946

number of matches. For most of the samples, the scan date corresponds to the same day when the samples were collected in VirusTotal.

We also repeated the queries to VT several months after, on July 2016, and one year later (January 2017), forcing a rescan of the samples. While this would not be possible in a realtime processing of the collected samples, we wanted to see if the number of AV positive matches was considerably different after a long amount of time – as this would indicate that this phase of the analysis should be postponed to take advantage of updated malware signatures.

4,684 samples were initially not detected by any anti-virus solution but were later detected by at least one in our last check. In contrast, 2,281 PE files had at least one positive in VirusTotal in the moment of the first query, and zero positives at the time of the last one. Anecdotically, on our last query there was a small number of samples for which we could not obtain a report. We believe this could be due to a sample removal upon request (VirusTotal might remove some samples or documents if confidential or copyrighted data gets accidentally leaked).

Even if there were some changes in some AV labels, this only affected 3.5% of the total number of samples, and many of these changes were caused by false positives later corrected by AV vendors. This results shows that it is not necessary for a company to wait months in order for the AV detection to stabilize. Therefore, our analysis pipeline can be started immediately after the samples are collected.

AntiVirus labels can differ significantly among the different security products. While some engines provide signatures to detect specific families, others rely on heuristics, indicators, or other features to classify samples. In many cases, the scan results consist of generic names associated to the observed malware behavior (such as *generic trojan*). Also, different AV companies often assign different names to the same family, which further complicates the problem of reliably identifying to which family a sample belongs to. In order to overcome this issue, we leveraged the state-of-the-art `AVClass` tool recently presented by Sebastian et al. [33]. This tool is a best-effort to leverage the aggregated information provided by many AntiVirus products in order to determine the family of a sample. This does not mean that it is a perfect classification system or a sound approach to generate ground-truth, as it is only as good as the labels provided by AntiVirus solutions. In many cases these labels might be inaccurate or too generic. Nevertheless, in this study we measure to which extent this information can be useful to categorize and label groups of samples that are initially not related to each other.

AVClass classified our samples in 1,057 different families, which covered a total of 118,605 PE files (69% of the entire dataset). Table 5 shows the number of samples for the most common 10 families. Interestingly, we can see that there is a significant number of variants of the `Allaple` and `Virut` families which, despite being over ten years old, are still responsible for a large number of malicious samples collected today. They also represent the two most common source of malware variations. `Virut` is a file infector, and therefore any benign executable infected by it would be

Table 5. Number of samples for the top 10 families

Family	Samples
allaple	54,097
virut	16,328
browsefox	7,400
outbrowse	4,600
installcore	2,395
eorezo	1,436
softpulse	1,421
sality	1,253
virlock	1,050
loadmoney	1,047

flagged as malicious. `Allaple` is instead a polymorphic worm that creates multiple different copies of itself on each installed machine.

From the total of 172,612 PEs, 54,007 did not have a clear AVClass label. 8,898 of these samples were unknown to VT and we could not submit them for privacy reasons. For the remaining files, the tool was not able to determine a consistent label, often because there were too few positive matches on VT for those files. Only 8,552 samples with no AVClass had 5 or more positives, but due to the inconsistency or generality of the labels, it was not possible to assign the samples to one specific malware family.

We also run AVClass in order to detect Potentially Unwanted Programs (PUP) on our dataset, using the `-pup` option. The tool flagged 36,146 samples as PUP (22% of the PE files present in VirusTotal), of which 31,618 had an AVClass label assigned. We checked the most common AVClass family names for PUP, and observed that many of these labels correspond to broad family names rather than specific malware families (e.g., browsefox, outbrowse, installcore, eorezo, softpulse, loadmoney...). As we will see in the following sections, this has an impact over sample categorization and prioritization.

B3. Dynamic Analysis

The analysis techniques described so far (file type analysis, and AV labeling) allowed us to narrow down the sample set by automatically identifying known benign and malicious files. The next phase of our analysis pipeline consists in running each sample in a dynamic analysis sandbox to collect information about its runtime behavior. These tools execute samples in an instrumented environment, and provide as a result a behavioral report and a set of indicators of compromise – such as the list of contacted domains, IPs, network connection types, and mutexes created by the sample.

Even if in the previous steps we were already able to filter out a large number of files as malware belonging to well known families, we decided to analyze the entire set of 172K samples in the company private malware analysis sandbox. This allowed us to later refine our classification by finding similarities in the behavioral reports between the set of well-known samples and those which either were not present in VirusTotal, had no positive matches, or no clear label information.

The sandbox provided a behavioral report for 168,817 (97.8%) of the submitted files. The remaining samples were discarded because of errors. As part of the final report, the sandbox also reports an aggregated score of the suspiciousness of the behavior, with 0 being a lack of any malicious indicator, and 100 meaning that there are numerous evidences of malicious behavior (such as thread injection, dropped files, registry manipulation, disk infection, or suspicious network activity).

Table 6. Sandbox score for the analyzed samples

Score	# of PE files	Score	# of PE files	Score	# of PE files
0-9	8,193	40-49	3,022	80-89	668
10-19	9,450	50-59	4,619	90-99	563
20-29	11,479	60-69	1,464	100	125,051
30-39	3,990	70-79	318		

Table 6 summarizes the scores obtained for this set of samples. Interestingly, 75% of the samples score 100 points, and only 4.9% show the lowest possible score.

From the group of samples for which we had no family name (because they had never been submitted to VirusTotal) 4,098 were detected as `Allaple` binaries based on the IOC-based detection mechanism implemented in the sandbox. More interestingly, a stunning 32,532 samples with no family name did not show any malicious behavior during their execution in the sandbox. Some of these samples obtained high scores in the sandbox because the system also considers several static features as potential indicators of maliciousness.

These 32,532 samples are more interesting and deserve a closer look, because there are certain aspects of sandboxes that can be leveraged by malicious samples to evade analysis. First, sandboxes are typically based on virtualization software (unless bare-metal machines are used for analysis). Second, the analysis can only cover one execution path. Malware writers exploit these limitations to avoid sandbox analysis in order to stay undetected as long as possible.

The sandbox system used during these experiments is configured to detect and bypass most sandbox evasion tricks. The solution is based on a top-vendor OEM sandbox product, which has been adapted over time to make sure it can properly analyse samples that leverage anti-sandbox tricks. Maintaining a sandbox is a continuous effort. Malware writers adapt their techniques over time. For this reason, a team in the company regularly verifies the effectiveness of these configurations, and adapt them whenever new techniques are discovered or used by malware authors. This sandbox provides not only methods for mitigating Virtual Machine detection approaches, but also other generic anti-dynamic analysis methods such as checking system properties (number of processors, realistic file system, connectivity), lack of user interaction (it mimics user interaction), bypasses long periods of inactivity (loops, sleeps), and has full internet connectivity.

The scope of this paper is not to identify virtualization detection techniques. For this, the reader may refer to existing previous work [28], or existing tools like Pafish [8] that are leveraged by security professionals for this task.

In any case, given that maintaining sandboxes is a continuous effort for a security company, we model this effort as part of our measurement study. We assume that any company can have a well maintained, up to date set-up, and that a team will need to monitor samples with low activity to understand why samples are not running.

As part of this measurement, we tried to understand why some of the samples did not show a significant activity, and measure how many cases fail because of problems like missing dependencies (e.g., individual components of more complex software applications) or GUI applications without any remarkable malicious behavior. We ran all these samples on a second sandbox that uses a completely different underlying technology to execute the samples, and combined the results with the ones obtained from the first set-up. This sandbox is a proprietary solution developed internally, and is maintained by an independent team.

During these experiments we did not analyze the samples in a bare-metal sandbox set up, because these systems have a limited availability and we did not want to interfere or degrade

the performance in production systems. Nevertheless, as the results will show, a great number of samples lack the necessary requirements to run, are corrupted, or are GUI-based applications that, even with the user-interaction capabilities of our sandbox, still do not trigger any suspicious activity in the system. None of these samples would trigger any different behavior in a bare-metal system.

Table 7. Classification of Samples with No/Low Activity

	No activity	**Low activity**
GUI	599	270
Missing DLLs	3,814	599
Crash	0	723
Corrupted file	9,805	64
Total	14,218	1,656
Still Unexplained	10,159	6,499

We relied on four heuristics to identify the reason why a sample did not run in the sandbox, or run without performing any relevant activity. Some of them are based on the combination of reports obtained from the two sandboxes. The first searched for programs that had a graphical user interface and waited for some user interaction. We isolated these cases based on our sandbox reports that include this information. The second matched programs that did not start (or failed at a later time) because they required some dynamic library that was missing in the system. Again, this situation is detected by the sandbox and documented in the report. The third rule filtered out all samples that crashed, and finally the last rule identified those files that were corrupted or that could not even execute a single instruction (typically because of corrupted binaries that the OS loader was unable to properly start). Table 7 shows the number of samples filtered by each rule for the group of samples that did not show a significant activity, showed a very low activity, or crashed at run-time soon after the start. This last rule combined information from VT reports, and from the two sandboxes. In these cases, the second sandbox allowed us to confirm the cases of samples that did not even start running due to corruption.

Totally, this approach allowed us to identify 14,218 samples with no activity and 1,656 with low activity for which dynamic analysis would never provide any useful results. While this is not so common on a manually collected malware dataset, a real sample feed contains many files with a complex GUI that are benign, or may be corrupted, or have dependencies. This has an impact on the resources a company must dedicate. These samples sum up to 17 GiB of disk space and consumed the equivalent to 55 dedicated sandbox VMs during 24h, considering a 5 minute timeout for each sample.

After this filtering process, there is a number of samples that remain unexplained. Given that we cannot extract further information from these samples, a company would require to resort to manual analysis to understand why these samples did not show a malicious behavior on the sandbox. Part C details in depth the manual analysis experiments we conducted to dissect this set of samples.

B4. Grouping Behavioral Reports

The last step of our analysis pipeline consists of propagating the labels of known families identified using `AVClass` to other samples, by taking advantage of the information extracted from the dynamic behavior of each sample. To perform this task, we implemented a clustering algorithm in order to group together sandbox reports that exhibit very similar indicators.

Our clustering approach first normalizes all the indicators present in the report, such as registry keys created, updated or deleted, mutexes, domains or IP addresses contacted, network protocols used, samples dropped, file paths for files created, modified, or deleted, hooks installed both in user and kernel land, process and thread activity, window activity, memory activity, and other miscellaneous indicators such as anti-reverse engineering or sandbox detection techniques implemented. Once all these information have been normalized and joined in a single report, we compute a locality sensitive hash for each report by using the TLSH [30] algorithm. This approach allows us to measure the text similarity between two normalized reports. Afterwards, we used the single linkage hierarchical clustering algorithm and the distance based flat clustering approach implemented in the `scipy` Python library [21] to aggregate the similar samples. This approach allows us to aggregate groups of similar sandbox reports into clusters. The selected approach requires a threshold to define flat clusters from the hierarchical clusters generated in the first step. Given that we do not have a solid ground truth to determine if two reports are similar to each other, we based our decision on the results reported in the original paper [30] to ensure that our threshold would not produce a high number of false positives, and empirically validated the results by sampling generated clusters and manually checking their correctness. In this way, we selected a conservative threshold of 40.0, that is reported to have a low FP rate (<0.50%), and a reasonable detection rate (>50.00%).

Our system assigned a label to each group of reports according to the nature of the samples it contained: **Majority Clusters** refer to those groups of samples in which a great majority of samples belong to the same well-known family according to `AVClass`. **Mixed clusters** refer instead to groups for which we have contradictory malware family information, and **Noclass clusters** to those for which the majority of the samples do not have any AVClass label. We define the **Majority** and the **NoClass** clusters as containing at least 90% of samples of a certain type, to cope with some possible noise in the data. In fact, 10% of samples in our dataset are not present in VirusTotal and therefore we have no labeling information available. Moreover, although AVClass is a standard and well tested tool, it does not always provide a precise family name because the AV labels it relies upon are often imprecise, inconsistent, and use non-standard terminology [33].

Table 8. Number of samples per cluster type

Cluster type	Not in VT	With AVClass	No AVClass, but in VT
Majority cluster	982	35,105	103
Noclass cluster	824	99	4,655
Mixed cluster	425	13,691	2,134
Singleton sample	519	4,093	2,234

We applied our clustering analysis to all files that exhibited some behavior, with the exception of `Allaple` samples, which can be easily identified given their characteristic sandbox artifacts. As a result 58,018 samples were grouped in 1,853 clusters, while 6,846 PE files were not assigned to any cluster (and therefore were marked as singleton [1] samples). In particular, 65% of the clusters were majority clusters, 23% were flagged as noclass, and 12% as mixed clusters. Table 8 shows cluster distribution in detail.

Our system is not designed to provide a precise malware family classification system. This is outside of the scope of the paper, and only possible with a solid ground truth that is not available

[1]The term singleton is used in mathematics to denote a set of size one. Following this definition, we consider singletons all the samples that are not grouped together in any cluster. This term has been used in the past [9] in a different context, to denote sample hashes that have only been observed in one machine.

for a dataset of this type. Instead, we implemented this approach to measure how useful it is to leverage the information obtained from AVClass and behavioral reports to group samples together. These groupings reduce the effort needed to sift through the dataset. This method allows us to understand how many sandbox reports obtained from a sample feed can be grouped together due to their similarity, and how many of these groups of reports can be labeled automatically.

The prevalence of Majority clusters supports the fact that our report clustering solution was successful at grouping together samples from the same family that show a similar behavior. Moreover, Majority clusters allow us to propagate known labels from samples that had a family to others that did not have one (either because they were not detected by any antivirus software or because the labels were inconsistent). For instance, the 982 samples not present in VirusTotal that were assigned to majority clusters can be considered to belong to these families, despite the fact that we initially had no labeling information for them. In contrast, as we will see in more detail in the following section, a significant number of Mixed or Noclass clusters were composed of samples with generic and unconsistent AVClass labels, or where AVClass was not capable of providing a consistent family name. The inconsistency or inaccuracy of AntiVirus labels has an impact on how we separate **Majority** and **Mixed** clusters, and does not allow to precisely assign a given malware family name to each cluster. This reflects the lack of consensus among AV vendors, as well the fact that many products assign generic names based on behavioral or heuristic indicators. For many of these samples, no proper family name is ever assigned to them. Nevertheless, this does not determine whether these samples are considered relevant or not. On the contrary, this approach allows us to apply different prioritization rules depending on the composition of each cluster type.

B5. Prioritization

Our clustering approach allows not only to propagate labels from known to unknown samples, but also to understand the different types of malware contained in the dataset, allowing to prioritize their analysis. We can think of our dataset as containing three types of files: known malware (class $\mathbf{M}_k$ for short) for which the `AVClass` tool was able to identify the family; unknown malware (class $\mathbf{M}_u$) for which existing antivirus are not able to assign a precise name or family; and benign samples (class **B**).

From a security community perspective, the most interesting group is $\mathbf{M}_u$, which contains both undetected known malware (modifications of existing families), new unknown families, singleton malware samples, or just malware families that may require better signatures to improve AV detection. However, the class is very broad, and not everything in it is equally interesting. For instance, $\mathbf{M}_u$ also contains Potentially Unwanted Programs (PUP) that may have several positives in VirusTotal, but labels too generic to clearly identify them. These samples include applications that inject advertisements on the users machine, or that collect too much data about the user invading her privacy. Although these tools may not be as harmful as banking trojans, ransomware, or RATs, they are still a risk for the users privacy and many AVs label them with generic names such as PUP, or spyware.

On top of this, security companies may have different priorities. For instance, one company may want to improve the quality of their signatures focusing on malicious samples only detected with heuristic-based labels. Another company may want instead to improve their coverage of PUPs, while a third one may be interested in finding emerging unknown families. The categorization and prioritization that follows was conceived from a neutral (i.e., independent of the actual goals of the AV company that provided the dataset) and global point of view, seeking the interest of the security industry globally. Following this principle, we decided to further refine the $\mathbf{M}_u$ class in three sub-categories: $\mathbf{M}_v$ containing variations of well-known families; $\mathbf{M}_g$ containing samples

covered by generic signatures or heuristics, or PUP; and $\mathbf{M}_n$ covering previously unknown families, singleton samples, and any malware which is currently undetected by AV solutions.

The observation behind the prioritization phase is that different clusters contain different distributions of malware classes. Considering this, we designed a set of conservative rules to mark each cluster according to the type of malware ($\mathbf{M}_v$, $\mathbf{M}_g$, or $\mathbf{M}_n$) it most likely contains. The following prioritization rules are based on label propagation and a set of phenomena observed during manual cluster inspection. As we will show in the next Section dedicated to manual analysis, this step can help a company to choose how to better allocate its human analyst resources.

The *first phenomenon* we observed is that some **Majority** and **Mixed** clusters are composed of a majority of samples labeled with a well-known family name and a small number of unlabeled samples that were not present in VirusTotal (and therefore there was no data available to extract a label). In these cases we leverage our clustering results in order to propagate labels to unlabeled samples.

The *second phenomenon* involves samples for which AVClass was not able to provide a family name, but that were present in VirusTotal and had a significant number of positives (and therefore, detection names). In all the cases we observed, this occurred either because the binary was considered a PUP with no consistent or clear family name, or because it was detected by a majority of engines by using heuristics instead of family specific signatures. In a malware analysis pipeline like ours, these samples must be treated with suspicion provided that they are considered malicious by several engines, but from a *detection* perspective, they do not deserve the same priority as undetected or unknown malware.

The *third phenomenon* we observed is related to how clean samples are grouped into clusters. Provided that malware analysis sandboxes are designed to capture behavior typical from malware samples, and that it is not likely to find many variations of the same benign software, this type of samples either show a low activity, are considered singleton by our clustering approach, or end up in clusters of similar benign samples.

Finally, *the fourth (and last) phenomenon* observed is related to how AntiVirus products label the samples. After applying AVClass with its standard parameters and default dictionaries, we observed many clusters in which the samples were present in VirusTotal, had a significant number of positives, but where named using very similar names (e.g. *backupmypc* and *mypcbackup*), or frequent broad names such as *zusy*, *eorezo*, or *icloader*. These types of clusters fall in the **Mixed** cluster category, despite the fact that the samples probably belong to the same family (they show the same behavior).

Based on these phenomena, we assigned the **Majority**, **Noclass** and **Mixed** clusters to different categories according to the following rules:

- **Majority Clusters.** The 1,085 Samples with no AVClass contained in majority clusters are assigned to the family of their corresponding cluster, and therefore belong to class $\mathbf{M}_v$ of variations of known samples.
- **Noclass Clusters.** If the totality of the samples in the cluster are present in VT but have less than 3 positives, and if no single sample in the cluster has a high score, these samples (1,311 in total) are likely to be benign (class **B**). We include in this set a small number of samples with 1 or 2 positives in order to account for occasional false positives and extra paranoid AV set-ups that flag as malicious any sample with a sign of corruption. Although we assign these samples to a low priority group, this prioritization is subject to the interests of the company. For instance, an AntiVirus company may be interested in reducing these false positives, or an AV aggregator may be interested in tuning properly the AV engines to avoid these cases.

If instead, the majority of the samples (at least 90%) in the cluster are present in VirusTotal but are labeled by AVClass as PUP (using its PUP detection option), we assign the cluster to class $\mathbf{M}_g$. These samples (1,280 in total) have too generic detection names so AVClass cannot provide a consistent family name. In this case, again, we left a small margin in our rule (10%) for samples that might not be correctly detected by AVClass as PUP due to a low number of positives.

- **Mixed clusters.** Mixed clusters are harder to handle because they can be the result of different possible cases. First, if the cluster contains a majority of samples that are not present in VirusTotal (and therefore do not have an AVClass label), while the rest of samples belong to one family, we can consider that the samples not present in VirusTotal belong to the same family as well. We can propagate the labels to the 223 samples of this type, that are assigned to $\mathbf{M}_v$.
  Second, if the cluster contains a majority of samples with no family name that are present in VirusTotal and considered PUP by AVClass then we consider that the samples belong to type $\mathbf{M}_g$. Also, we observed a number of clusters with samples present in VirusTotal and a high number of positives (20 or more positives on average). We checked these cases and observed that the labels present in VirusTotal correspond to generic detection names such as *downloader* or *Trojan.Generic*. We included the samples in these clusers into type $\mathbf{M}_g$, totalling 793 samples.
  Finally, many mixed clusters contain samples with conflicting family names. We observed that the majority of these cases correspond to families named differently by several vendors. After checking these cases manually, we concluded that the AV engines provide mostly broad names such as *zusy*, *eorezo*, or *icloader* or when AVClass gets confused by packer names or family synonyms. In any case, the samples show the same behavior and we consider them part of the same family (or part of very similar families). Accordingly, if all the samples in the cluster present in VirusTotal are already detected as malicious by some AntiVirus product we assign them to the class $\mathbf{M}_g$. There are totally 254 samples that meet these conditions. We exclude from this set the clusters in which there are samples present in VT with 0 positives, because we believe those cases deserve a more closer look.

This prioritization process can *tentatively* classify 4,946 previously unknown samples. What remains is 2,754 singleton files that did not belong to any cluster (and therefore fall by definition in class $\mathbf{M}_n$) and 4,177 samples distributed over 204 clusters that were not covered by any of the previous rules (and are therefore also assigned to class $\mathbf{M}_n$).

During our prioritization process, we defined a set of rules based on our observations on the clustering results. These rules occassionally require to set certain thresholds in order to mitigate the imprecision of AV labeling, as well as the percentage of samples that are not present in VirusTotal. In the cases where we needed such a threshold, we adopted a 90% / 10% split in order to divide the clusters. This threshold is based on the number of samples missing in VT or the limitations of AVClass given how AntiViruses label samples [33]. As a matter of fact, this value is rather conservative as it minimizes the number of clusters treated as $\mathbf{M}_g$ or $\mathbf{B}$. Nevertheless, a company implementing a similar analysis pipeline may adjust these thresholds based on the nature of their incoming feeds and the labeling precision of the AV aggregator in use.

Summary of the Analysis Pipeline

Figure 3 provides a visual summary of the incremental refinement process that was performed by our automated analysis pipeline.

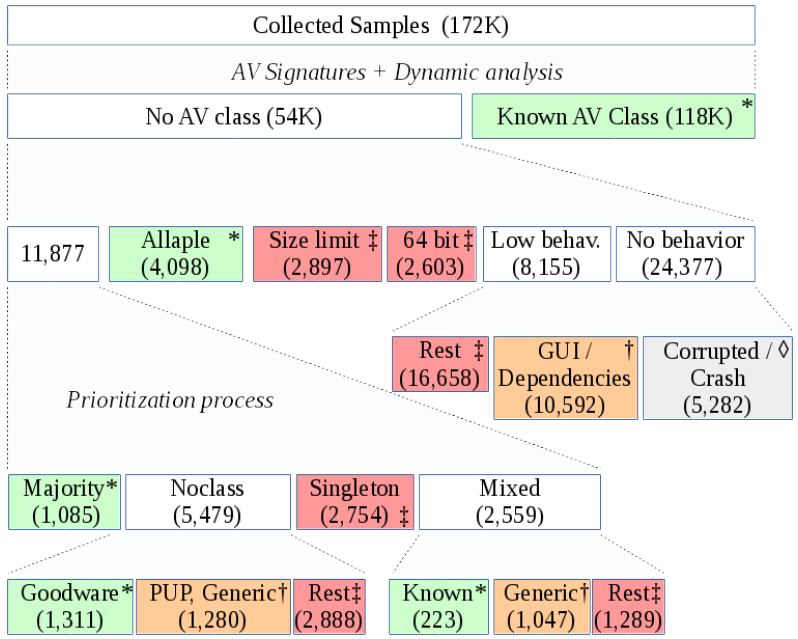

Fig. 3. Automated Analysis Pipeline

On a first step, we leverage VirusTotal and the AVClass tool. This allows us to differentiate the samples with a clear family name and samples that remain unknown. We also apply dynamic analysis to all the PE files.

On a second step, we isolate several sub-sets of binaries: The first isolated subset are 64 bit binaries, that will be treated separately as singleton (high priority) files.

A second subset contains files of a size higher than 10 MiB (mostly installers or bundlers), that also deserve a different treatment. A 92% of these files have either no positives on VT or a very small number (1 or 2). Although this set is composed of mostly clean files, they must still be checked, and are assigned a medium priority.

We also isolate Allaple binaries (i.e., samples that belong to predominant, well-known and easy to detect polymorphic worms). If the sandbox contained specific rules for detecting other similar uninteresting families, those samples would be placed in this third subset.

Finally, the fourth subset is composed of samples that did not show a significant behavior in the sandbox. In this case, we apply a set of rules to identify files that are corrupted or crash, which are discarded from further analysis. The files with missing DLL dependencies (92% have less than 3 positives) or a GUI interface (80% have less than 3 positives) cannot be completely discarded, but have a different priority from the rest, which are considered high priority.

Finally, the third step is to apply our sandbox report clustering approach, that allows us to separate the samples into different *buckets* that contain samples with a very similar behavioral report.

In some of the cases, the great majority of the samples in the bucket belong to the same family, so these labels can be propagated to the rest of samples.

In our picture, green buckets (also marked with the * character on the upper right corner) represent the lowest priority samples that either belong to well-known families ($\mathbf{M}_v$), or are likely

benign. The only gray bucket (marked with ◊), represents samples with the low priority. Orange buckets (marked with †) represent what we consider medium priority samples, given that they contain generic malware samples and PUPs ($\mathbf{M}_g$). Finally, red buckets (marked with ‡) contain the most interesting cases: a combination of singleton samples and unclassified clusters. This bucketing approach allows to identify which samples deserve a closer attention from human analysts. Also, it is very important to stress the fact that since all the samples in each cluster share the same behavior, it is not necessary to analyze them all. In fact, an analyst could randomly sample a given cluster, e.g., by manually reversing only few samples, and this could be sufficient to flag the entire group of samples it contains.

PART C – MANUAL SAMPLING

The analysis pipeline described in Part B allowed us to automatically classify and filter out over 85% of the collected samples, reducing the initial set of 172,612 binaries to 26,192 samples with high priority. We focus this part of the study on this subset of samples in order to understand the implications, in terms of human efforts, of dissecting this dataset.

The vast majority of them (16,658) are programs that antivirus signatures were unable to properly classify, that showed low or no activity in our two sandboxes, and that could not be automatically assigned to a known family by our tag-propagation heuristics. The group then includes 2,754 singleton samples, 2,603 64-bit executables, and 4,177 files (distributed over 204 clusters) that our heuristics assigned to the $\mathbf{M}_n$ category, and thus have a high priority.

26K samples are still too many for an exhaustive manual inspection. Luckily, for the ones for which we have clustering information, our system already grouped similar files together so that it might be sufficient to check only a few samples per cluster.

This section describes three *manual analysis* experiments we conducted in order to unveil the nature of the remaining files. The goal of this manual analysis phase is to understand the nature of the samples, and to estimate the human effort that would be required to look at all the samples in the feed that cannot be processed using fully automated techniques. Moreover, these experiments also document the key factors used for sample classification by experienced reverse engineers.

For this reason, the company that provided the samples also allowed us to assign some manual analysis tasks to six of their malware analysts, with 2 to 6 years of experience in the malware analysis field. The analysts work on the same team and on a daily basis they conduct reversing engineering tasks. They are part of an international team and they all share the same background in the field of binary analysis. Each analyst was provided with a set of files and was asked to answer a set of simple questions: `(i)` Is a sample benign or malicious?, `(ii)` (If malicious) What class of malware does the sample belong to (trojan, botnet, keylogger, adware, ...)? and `(iii)` (If malicious) What malware family does the sample belong to?

In order to overcome this task, the analysts were provided a basic set of information collected during our pipeline, which includes:

- **VirusTotal report.** Labels from AV engines, results for packer scanning engine, submission filenames, number of submissions, whether the sample is signed or not, and information about the certificate, static features about PE headers, section names, sizes, section entropy, imported DLLs and functions, and comments on VirusTotal's community.
- **Behavioral report.** Detected anti-analysis tricks (anti-vm, anti-debug, sleep loops...), static analysis anomalies (uncommon section names, checksum errors, high entropy, etc.), urls found in memory, activities on registry, file system, system services, scheduled tasks, list of mutexes created, files downloaded or dropped to disk, processes created, process injection,

domains queried, network connections (IP, port, protocol), usage of scripting (VBS, BAT, etc.) and proprietary rulesets based on IOCs matched on the behavior of the sample.

We also asked each analyst to mark which type of analysis she performed to reach her decision. If she only relied on the information provided (VirusTotal report and sandbox report), the analyst should mark **blackbox**. We expected this analysis to take less than 10 minutes per sample (and in many cases considerably less). If, on the contrary, she performed some additional static or dynamic analysis (such as unpacking, debugging or process monitoring), the analyst should mark **manual**.

Finally, we asked each analyst to report the time spent on each sample, and the confidence of the final result. A low confidence means that the analyst was not completely sure of her findings and a more in-depth and time-consuming analysis would have been required to confirm the results. All samples were randomly selected and assigned to the six analysts.

C1. High Priority Group

Our manual analysis started by looking at the $\mathbf{M}_n$ class, which consists of a combination of singleton files and samples contained in clusters that did not match any of our rules.

A cursory look at this category allowed us to quickly identify a large group of 997 samples, that consisted of application bundlers and download assistants that in many cases were flagged by AntiVirus software under generic categories such as *downloader* or *adware*. We therefore decided to re-classify these samples as part of the $\mathbf{M}_g$ malware class, which is more suitable to represent the *grayware* nature of these files. The reason why they did not match the rules designed to automatically identify $\mathbf{M}_g$ samples was the inconsistent number of positive matches in VT. This lack of consistency is a sign of the difficulty for AV vendors to draw a clear line between what is considered benign, and what is considered malicious.

After removing these files, we randomly extracted 273 singleton samples, and a fixed number of samples from every cluster containing a well defined behavior: 1 sample from clusters of 2 samples, 2 samples from clusters with less than 30 samples, and 3 samples from larger clusters – for a total of 546 samples.

For the **singleton samples**, 56.8% of the files were labeled by the malware analysts as benign and 43.2% as malicious. For 5.9% of the malware, the analysts were unable to identify the type (e.g, botnet or trojan) and for 50.48% they were not able to assign a precise family name. The margin of error at a 0.95 confidence interval is 5.6% (in other words, there is 95% probability that the percentage of benign files in the singleton group is between 50.2% and 62.4%).

Interestingly, for over 95% of the analyzed samples the analyst did not find necessary to perform a more precise manual analysis. In all these blackbox cases the analysts were able to convict a sample leveraging the available resources. In these quick analyses, the VT reports play an important role. A crucial field is the number of positive engines. In some cases, the analysts convicted the samples only by relying on this field if the number of positives was higher than the 50% of the available AV engines. In the other cases, the combination of this field and some indicators in the behavioral reports solved the problem; the analysts reported that some of the fields in the report were distinguishing of the malware family. In almost all the cases, indicators such as process injection, as well as the anti-debugging and anti-VM techniques made the decision straightforward.

In a number of cases, the analysts found strong indicators to flag the sample as benign. The VT reports show if samples are signed. This field, together with a number of positives equal to zero, and a clean behavioral report indicate that these samples are probably benign.

For the **clustered samples**, 48% of the samples were labeled as benign, and 52% as malicious. In this case, the margin of error for the estimated population mean is 5.7%. Interestingly, in this case the analysts were able to determine the malware type for almost all the samples, and the

precise malware family for 55% of them. In both cases, benign and malicious samples, the number of samples that required manual analysis for their understanding was very low: 3.8% and 2.8% respectively.

We evaluated the consistency of the analysts decisions in the previous experiment. From 86% of the clusters with more than two samples, the analysts provided consistent results and labeling information.

We manually inspected the remaining 14% to better understand the reason behind these mismatches. In some cases, the behavioral reports contained too generic features and as a result the clustering approach grouped together unrelated samples with the same generic sandbox behavior.

C2. Samples with Low or No Activity

This was the most problematic group that survived our automated classification. In fact, since these samples did not show any (or showed an insufficient number of) actions during the sandbox analysis, there was no behavioral profile that could be used for clustering and that could be provided to support the manual analysis.

We started our analysis from the 10,159 samples that did not show activity in both sandboxes, and immediately noticed that a large group of 1,049 programs contained practically no code. After a closer analysis, we concluded that they were residual automatically generated files.

We then randomly sampled and manually checked 349 samples from the 9,110 remaining files. The analysts who checked these samples were asked to conduct a conservative review of the dynamic analysis reports of both sandboxes, including the screenshots or video extracted from the output of the execution, the information available in VirusTotal, and a short inspection of the actual sample if necessary. On top of answering the obvious questions about the malicious nature of the file and the need for an additional manual analysis, we also asked the analyst to mark if the samples showed any sign of failing during their execution or if they required any user interaction (which could explain the lack of behavior).

For the set of 9,110 samples with no activity, 18.3±4% required user interaction and 35.8±4.9% failed to run in the sandbox during runtime either due to programming mistakes or file corruption issues (e.g., some installers may start correctly but detect the corruption of the package to install at runtime and stop accordingly). In this case, the analysts inspected the screenshots and the video recordings to identify specific runtime issues reported in the document we shared with them.

65.6% of the inspected samples were considered legitimate while 34.4% malicious (with an estimated margin of error of 4.9% at 95% confidence according to our sampling rate). 34.9% of the samples considered legitimate were flagged as requiring additional analysis. Most of these cases were binaries for which no malicious indicator was found, but due to the fact that they were not signed, a much deeper inspection would be required to rule out the possibility of some infection or hidden backdoor. In this situation, the analysts tried to identify malicious samples first by checking the number of positives in VT reports. Unfortunately, in many cases this number was not high enough to make a decision. For all these samples, the analysts also checked the strings of the binary, the section names and permissions as well as the entropy, trying to identify the presence of a packer. All this information is available in the online VT report. The lack of information from behavioral reports complicated the work of the analysts. Many benign files were identified by checking the signed field as well as the number of positives, and the name of the binary. In many cases, these binaries are known applications (e.g., music players, PDF viewers, etc) which are not present in the public whitelists. Also, 34.1% of malware was considered to require an extra manual analysis step. In this case, this was the consequence of binaries that did not present any clear malicious behavior, but that had nonetheless some suspicious indicators (e.g.: mimicking legitimate products that one

may expect to be signed, few positives in VirusTotal, or the presence of a packer or non-standard section names).

These numbers indicate that roughly one third of this samples population would require extensive manual analysis to reach a reliable conclusion. Moreover, 80.80±4.07% of the samples were either flagged as benign, required some user interaction, consisted of corrupted installers that crashed, or could not even be loaded by the system loader. This means that only 20% were candidate to contain some malicious functionality even after a highly conservative filtering.

In the second test we looked at the 6,499 samples that showed some limited activity in the two sandboxes (but insufficient to perform clustering). We asked again the analysts to look at 349 sampled PE files, considering again the same sources of information and applying the same classification criteria. For this type of samples, 15.19±3.68% required manual interaction, and 67.91±4.79% failed during execution (note that the samples that generated a crash-dump were already filtered out, so these samples failed to run but showed some warning message instead of exiting abruptly). 73.25% of the samples were considered benign (with a 7% of them being reported to require manual analysis), and 26.8% malicious (with a 59.3% of them requiring manual analysis for a certain answer). In this case, for this population and sample size, the margin of error of the estimated population mean is 4.42%. Overall, 91.4±2.87% of the samples either failed, required interaction, or were considered benign by analysts. In this experiment, the limited amount of information available from behavioral reports hindered the work of the analysts. For this reason, the analysts adopted a step by step approach. As a first step, they quickly marked the failing samples by watching the screenshots and video recordings. Second, they tried to correctly identify the malicious binaries. Again, this was possibile by checking some fields in the VT reports (positives, not signed, strings, imported functions, section names) and the few available fields of the sandbox reports. Considering the scarcity of good information for a correct conviction, more than the 50% of the samples needed a manual analysis. Regarding goodware, many samples were flagged as benign. The analysts made this decision after the inspection of the VT reports. Almost all these samples were signed benign applications that can be dowloaded from legitimate websites. In some cases, the analysts downloaded the binary from the company's website and matched the SHA256 hash to make sure the sample was benign.

C3. 64 bit Binaries

From the beginning of our study we have treated 64 bit binaries separately. Although some malware authors nowadays distribute 64 bit versions of their software, most of them still resort to 32 bit, because these binaries run correctly on both 32 and 64 bit systems. On the contrary, we expect legitimate applications to be widely distributed in their 64 bit version, given that most users today use 64 bit machines.

Given the potentially high proportion of legitimate binaries in this sample set, we divided it in two separate groups: the 2,191 samples with zero positives in VirusTotal, and the remaining 472 with more than zero positives. From the first group we randomly selected 101 binaries and assigned them to the analysts for verification. In all cases they could not find any sign that the sample could potentially contain malicious code. Only 11 samples were flagged as needing some extra manual analysis to confirm the validity of the verdict. These 11 cases were samples that had a copyright, product and original name associated to a real and well known product but that were not signed. This indicator is suspicious as the application could have been tampered to insert malicious code.

The second group is more interesting, also because a stunning 25.8% of the samples did not run correctly in the dynamic analysis sandbox. These samples were identified by checking the screenshots, the video recordings and the messages printed to the standard output. Therefore, in this case we randomly selected one third of the binaries for manual verification. 33.1% of them were

classified as malicious (from which 34% were marked as requiring some extra manual analysis), and 66.9% as benign (from which only 13.9% were considered to need manual inspection). The margin of error of the estimated population mean for the population size and sample size, was 6.27% at a confidence level of 0.95. The majority of the malicious samples had more than a 50% of positives in their VT reports, while many of the benign samples were signed.

Summary of the Manual Analysis Phase

In this section we broke down the final set of unclassified samples in different groups and we conducted a number of manual experiments to understand the nature of each group. If we aggregate all results, we discover that (based on our sampling rate, the size of each group, and at a 95% confidence interval) between 63% and 72% of the 26K we had left for manual inspection, are benign.

And, after removing all samples that crashed or did not work properly, at the end of the day we can expect between 5.4K and 7.6K malicious files – and for over 70% of them the analysts were confident about their final classification and did not believe an in-depth binary analysis phase was necessary.

If we sum up the samples with low or no activity we discarded using automated heuristics and the estimations made from manual analysis results, we can estimate that 27,251 out of 32,532 samples either required user interaction, were corrupted, contained errors, or had missing DLLs. The analysis of these samples required almost 100 VMs for a period of 24h. This has a significant impact on the infrastructure of a company. This is caused by several factors. Some applications (possibly even malicious applications) may require several components to run properly. Other applications are legitimate binaries that have a GUI. Finally, other samples are just corrupted binaries that are treated as unique samples given that they have a different hash.

In terms of effort, our analysts spent between 30 seconds and 90 minutes per sample. In the vast majority of the cases, inspecting the available reports and static information was enough to give a verdict. Sandboxes provide very clear summaries of the activity of a sample in the form of indicators, so understanding if a sample is malicious or clean is often straightforward for a malware analyst. Also, there are cases in which a single sample took up to 90 minutes of inspection to provide a verdict. Nevertheless, this value is far below the time that would be required for a complete reverse engineering and comprehensive understanding of the sample. Even if the group of analysts had a similar experience, the time required for sample analysis highly depends on secondary factors, such as time pressure and workload. In fact, it is important to keep in mind that the six analysts that participated in our experiments were still required to perform their usual work in the company.

Some fields in the reports had a significant impact on the manual classification done by the analysts. Regarding VT reports, we noticed that the analysts heavily relied on some fields, like the number of positives, signed status, names of the sections and entropy. In particular, the presence of a high number of AV positives was used as a strong attribute to convict a sample. Although this is not a good practice, analysts resorted to this indicator because we asked them to decide the nature of a sample after a really quick analysis. Finding stronger indicators requires a tedious effort. Regarding the behavioral reports, the analysts focused their attention on specific entries describing possible injection behaviors as well as evasion techniques like anti-debugging, anti-vm and stalling code. In conclusion, our experiments with analysts highlight that some attributes play a key role for classifying a binary either as malicious or as benign.

Based on their average reported time, it would take exactly **900 hours** to perform a cursory analysis of 24K samples. This time would be considerably reduced if we consider that it is not necessary to analyze manually all the samples in every cluster, specially if all the samples show a well defined behavior. Nevertheless, we should consider that additional time would be required to manually reverse engineer those samples that the analyst were not confident classifying just based

on their behavioral profile and static features. Even if this set likely contains only a few thousands samples, the time required to analyze them could be very high: even a mere 1h per sample (which is probably a very conservative assumption), would result in over 83 person day of work.

PART D – DISCUSSION AND MAIN TAKEAWAYS

In this paper we looked at the samples collected by a large security company over a period of 24h. Although there might be variations from day to day, we made a best effort to focus our study on one of the most *busy* days of the year. This approach allowed us to measure and understand the dimensions of current sample feeds and the implications for the company. This is, to the best of our knowledge, the first study that considers all the samples obtained from a feed, including those for which there is no information available about their procedence, or that do not even run properly on a sandbox.

In this section we distill the takeaways of our experiments, and discuss our conclusions. We believe that these conclusions can be generalized to any day of the year as they point out intrinsic limitations of malware feeds, not a particular sample distribution.

We discovered that, while the catch includes over one million files, only 172K were Windows executable binaries – the focus of this paper.

We then applied several fully-automated refinement steps to this initial set of candidate samples – including collecting antivirus labels, performing a dynamic analysis in an instrumented sandbox, and clustering the results according to their behavior and a number of static features. These steps mimics a realistic pipeline that a security company can put in place to process their daily dataset of collected samples.

The results of these processes can be summarized in the following nine findings:

[F1] We estimate that the entire analysis pipeline can be completed in one day using roughly 600 machines, and a five minute timeout. While this is a considerable amount of processing power, it is still well within the availability of major security companies.

[F2] Antivirus labels play a very important role to pre-filter the initial dataset and identify well-known malware families. Our experiments show that while such labels change and evolve over time, after one year these changes affected only 3.5% of the samples - thus confirming that the analysis process can be performed immediately on the collection day, without considerably affecting the final results.

[F3] AVClass [33] identified the malware family for 69% of the dataset. In other words, roughly 70% of the samples collected every day belong to well-known malware families for which there is a clear consensus among antivirus companies.

[F4] After our fully-automated analysis, we were able to filter out 85% of the samples. This means that only 26K samples are still unknown at the end of our pipeline.

[F5] Human analysts found easier to classify samples that belonged to clusters (even though they did not know this in advance) compared to singleton samples. This suggests that samples that can be grouped based on their behavioral profile are more likely to exhibit distinctive features that can help in their classification. Singleton are instead more often associated to benign files or to less "usual" forms of malware, which require more effort to be classified.

[F6] For over 95% of the samples classified manually, the analysts reached a conclusion by only looking at the antivirus labels and at the sandbox behavioral report. Moreover, in these cases they were often very confident and did not believe any further reverse-engineering effort was necessary to confirm their findings. This opens the possibility, at least in theory, to replace these black-box decisions with expert systems or properly trained machine learning classifiers.

[F7] Although manual classification was often performed in a few minutes, extending this process to the 26K unclassified samples that remained at the end of our pipeline would require over 100 days of manual work (considering 8h of consecutive, uninterrupted work per day). This does not include time-consuming reverse engineering of complex samples for which the analysts were not confident about their classification.

[F8] According to our estimations, 16% of our catch of the day (27,251 samples) did not run properly in a sandbox because they required user interaction, crashed, were corrupted, or had missing dependencies. These samples can consume a considerable fraction (up to a 15.5%) of the computational resources dedicated to our analysis pipeline.

[F9] We measured many of the characteristics of a real malware feed collected by a security company. We discovered that 10% of the samples did not have any labeling information available, 16% did not run properly inside a dynamic analysis sandbox, and at least a 22% fall in the category of PUP or generic malware. The feed also contained a significant number of corrupted samples, missing file inter-dependencies, and families that should no longer pose a real threat. While these numbers may not be surprising for experts in the field, it is the first time they have been precisely measured in a real world scenario. Moreover, based on these observations, we can affirm that a real sample feed is significantly different from manually collected datasets often used for testing and validating malware analysis, classification, and clustering approaches.

Based on these results, and despite the fact that automated techniques can precisely identify over 85% of the collected samples, we can conclude that processing and classifying every single Windows binary that is collected in a single day is only within the reach of large companies, as the required amount of computing and human effort is too high. In the rest of the section we discuss in more detail the main reasons, and present some of the research directions we identified that could bring us closer to a full understanding of the catch of the day.

The Intricacies of Goodware

One of the results of our study is the fact that benign files are in fact the hardest category to analyze. This may sound surprising at first, but it is an unavoidable consequence of two separate phenomena. First, the fact that benign files survive any filtering attempts. While we can try to capture signs of malicious behavior, the only way to identify a benign sample is to whitelist it based on its hash value (which in general is unfeasible given the fact that the vast majority of goodware are singleton files [9] and have a different hash on every machine). As a result, all goodware ends up in the same buckets with the unclassified files and the new malware samples – therefore causing a large increase in the manual effort required to sift through these clusters.

Second, benign files are extremely time-consuming to analyze. In fact, since there is no such a thing as a sign of "*goodware behavior*", the only way to classify a sample as benign is to rule out any possible sign of maliciousness. This is similar to the difference between finding a vulnerability in a program and proving that there are none: the second is a much harder task, which is undecidable for non-trivial programs. Similarly, to verify that a seemingly legitimate application is not modified to contain a backdoor – unless an exact copy of the file is retrieved from a trusted source or the code is signed by a trusted source – the only solution is to completely reverse engineer the application, which is an extremely long and complex task that cannot be completed in a reasonable amount of time.

When in our experiments we concluded that a sample was malicious, it is because we observed a tangible proof of it. We classified instead as benign any file that did not show signs of malicious

behavior during the dynamic analysis phase, and that looked like a legitimate software to a human analyst. However, we have no definitive evidence to support this conclusion.

The importance of dynamic analysis

Having a reliable dynamic analysis system is the first fundamental step in order to properly categorize the collected samples. Also, the expressiveness of the reports and the ability to normalize them (and therefore compare different samples) is a requisite for the success of clustering approaches. Our experiments show that the vast majority of samples that required manual analysis in Part C were those that did not exhibit any behavior in the sandbox. Any improvement in the automatic classification of those samples can have a large impact on the ability to process all the daily samples.

In particular, 16% of the collected PE files show no interesting behavior in the sandbox as they had DLL requirements, waited for user interaction, or were simply corrupted. Also, individual DLL files in the feed represent a challenge because they generally are part of a bigger application, or require a specific installation method, or it is necessary to deliver complex function calls to trigger their functionality. These files may ultimately require manual analysis. Moreover, they also have a considerable impact on the infrastructure of a company, as even a program that crashes at boot still requires more than three minutes of analysis time (e.g., due to the VM setup and report generation). Therefore, it would be interesting to pre-filter the collected samples to identify in advance these problematic files and avoid the computational overhead of dynamic analysis. However, this is an error-prone task, as files that may look corrupted (such as PE files with an incorrect checksum) may still be executed by the loader.

Finally, sample feeds treat files individually, although many of these files may not exist individually in the wild. Some applications (both legitimate or malicious) may have dependencies that if not present on the disk, can cause the application to malfunction. VirusTotal allows to navigate file relationships only if they are submitted to the service in bundles such as zip files. If the files are submitted individually, or are contained in a format that VirusTotal cannot uncompress (such as proprietary installers), it is not possible to track these relationships. The AntiVirus community would benefit enormously if these relationships could be tracked and integrated into sample feeds.

The Role of Machine Learning

As we already discussed above, most of the time it is sufficient for an expert to look at the behavioral report and AV labels of a sample to get a precise idea of the type of malware he is investigating. This suggests that this background knowledge could be captured by a model and performed automatically by a computer. Taking decisions based on non-obvious combinations of multiple variables is a common task for machine learning algorithms. While testing (and validating) the results of a classifier against the human analysts is outside the scope of this paper, several techniques have already been proposed to automatize the analysis of behavioral reports [14, 31, 32]. A properly trained algorithm can save hundreds of hours of tedious manual classification, just for the samples collected in a single day. However, it is important to design these techniques to replace human analysts only for simple decisions, and leave uncertain cases to experts for manual analysis.

The problem of real-world datasets

A sample feed collected by a security company can be very different from the datasets assembled by researchers for malware classification and clustering experiments. First, the distribution of samples is unbalanced. There is no clear boundary between benign and malicious applications, many of the samples do not have clear labels, and some of them show signs of corruption. Second,

many programs (specially benign applications and modular malware) are composed of several related files, but sample feeds rarely provide this sort of information. These unique properties complicate automated processing and consume a non-negligible amount of resources in the form of computing power and human intervention. Added to the fact that the number of unique samples collected every day is overwhelming, it makes state-of-the-art automated analysis unsuitable to properly process daily sample feeds. As a consequence, it is necessary to complement a sample feed with other data sources (such as client telemetry) in order to understand how several files might be related to each other. Finally, AV vendors often label samples with vague or generic terms that rarely help to identify the family of the sample.

In this paper, we documented to which extent an analysis pipeline must deal with inaccuracy and tuning parameters and thresholds to draw the line between what is important and what is not. It is important to note that these thresholds were used not as a way to automatically classify samples, but only to group them based on the similarity of their behavior and therefore reduce the time required by manual analysis. As we already mentioned previously, the actual value of the thresholds depends on the distribution of the collected data, on the amount of available information for each sample (i.e., specific AV labels), and on the accuracy of the adopted tools.

We hope our paper showed that putting together a complete malware analysis pipeline is a complex operation that involves the interaction among many different tools and techniques. Moreover, each tool needs to be tuned by selecting configuration parameters and thresholds to effectively mitigate inaccuracies in the input data. While previous work have focused on designing and evaluating individual steps, more research is still needed to study the integration of existing components for the automated processing of real-world datasets.

The Need for Prioritization

Another take away of our study is that we need more research on prioritization. Existing approaches (such as the one proposed by the US CERT [1]) are designed to prioritize malware-vs-goodware. While this may indeed help to select malicious files from clusters with benign applications, the challenge is to quickly identify the most "interesting" samples that are worth inspecting in more detail. In other words, hiding in the 26K files that remained at the end of our pipeline, there could be the next `Stuxnet`. Random sampling can help to understand the nature of those 26K files by only analyzing a subset of them, but does little to help us finding the needle in a haystack.

For this reason, in our study we divided the remaining 15% of the samples that could not be automatically classified in a number of color-coded buckets. **Green buckets** contained samples that are very likely to be benign or malware samples of well-known families. Since our systems are already protected against these samples, there is little incentive in manually inspecting these files to assign them to a precise family. Sophisticated malware has also nothing to gain in disguising as one of these files – as it would be immediately (even if erroneously) flagged as malicious by exiting antivirus software.

At the other end of the spectrum, the **red buckets** contain files that show signs of suspicious behavior but cannot be tied to an specific family. These buckets contain samples of class $\mathbf{M}_n$ and should therefore be analyzed with high priority. Our sampling at 95% confidence level shows that 47.62±4% of files in this category are malicious. **Red buckets** also contain files for which the automated analysis was unable to provide any useful information. Our sampling shows that 33.09±3.4 are normal benign programs and 67.77±3.4% of these files are programs that require human interaction or crash during execution due to corruption or other incompatibilities (and therefore would not benefit from being tested in a bare-metal machine). Finally, the **orange buckets**

contain generic malware samples and PUPs. Although from an AV industry point of view these samples may deserve a lower priority, a company may decide to invest some efforts in these groups to improve their signatures or to study grayware applications. This prioritization and bucketing can help companies to reduce the number of samples to manually analyze.

3 ACKNOWLDGEMENT

We would like to thank the reviewers for their insightful comments, and Cisco Talos Intelligence and Research Group for their support on this project, as well as the resources provided. More importantly, we would like to express our gratitude to the members of the Malware Research Team who donated their valuable time to conduct the manual analysis experiments.

This project has received funding from the European Research Council (ERC) under grant agreement No 771844 – BitCrumbs.

REFERENCES

[1] 2014. A New Approach to Prioritizing Malware Analysis. https://insights.sei.cmu.edu/sei_blog/2014/04/a-new-approach-to-prioritizing-malware-analysis.html.
[2] 2015. Symantec Global Internet Security Threat Report Trends for 2008. http://eval.symantec.com/mktginfo/enterprise/white_papers/b-whitepaper_internet_security_threat_report_xiv_04-2009.en-us.pdf.
[3] 2015. SymantecâĂŹs 2015 internet security threat report. https://www.symantec.com/security_response/publications/threatreport.jsp.
[4] 2016. Putting the spotlight on firmware malware. http://blog.virustotal.com/2016/01/putting-spotlight-on-firmware-malware_27.html.
[5] 2016. Symante's Internet Security Threat Report 2016. https://www.symantec.com/content/dam/symantec/docs/reports/istr-21-2016-en.pdf.
[6] 2017. Awesome Threat Intelligence. https://github.com/hslatman/awesome-threat-intelligence.
[7] 2017. VirusTotal File Statistics during the last 7 days. https://www.virustotal.com/en/statistics/.
[8] 2018. Pafish - Paranoid Fish. https://github.com/a0rtega/pafish.
[9] C. Gates B. Li, K. Roundy and Y. Vorobeychik. 2017. Large-scale identification of malicious singleton files. In *ACM Conference on Data and Application Security and Privacy (CODASPY)*.
[10] Ulrich Bayer, Imam Habibi, Davide Balzarotti, Engin Kirda, and Christopher Kruegel. 2009. A view on current malware behaviors. In *USENIX workshop on large-scale exploits and emergent threats (LEET) (LEET 09)*.
[11] Alejandro Calleja, Juan Tapiador, and Juan Caballero. 2016. A Look into 30 Years of Malware Development from a Software Metrics Perspective. In *Proceedings of the 19th International Symposium on Research in Attacks, Intrusions and Defenses*. Evry, France.
[12] Julio Canto, Marc Dacier, Engin Kirda, and Corrado Leita. 2008. Large scale malware collection : lessons learned. In *SRDS 2008, 27th International Symposium on Reliable Distributed Systems, October 6-8, 2008, Napoli, Italy*. Napoli, ITALY. http://www.eurecom.fr/publication/2648
[13] Emanuele Cozzi, Mariano Graziano, Yanick Fratantonio, and Davide Balzarotti. 2018. Understanding Linux Malware. In *IEEE Symposium on Security and Privacy*. IEEE Computer Society.
[14] Ekta Gandotra, Divya Bansal, and Sanjeev Sofat. 2014. Malware analysis and classification: A survey. *Journal of Information Security* 2014 (2014).
[15] Mariano Graziano, Davide Canali, Leyla Bilge, Andrea Lanzi, and Davide Balzarotti. 2015. Needles in a Haystack: Mining Information from Public Dynamic Analysis Sandboxes for Malware Intelligence. In *Proceedings of the 24rd USENIX Security Symposium (USENIX Security)*.
[16] Xin Hu, Kang G. Shin, Sandeep Bhatkar, and Kent Griffin. 2013. MutantX-S: Scalable Malware Clustering Based on Static Features. In *Presented as part of the 2013 USENIX Annual Technical Conference (USENIX ATC 13)*. USENIX, San Jose, CA, 187–198.
[17] Heqing Huang, Cong Zheng, Junyuan Zeng, Wu Zhou, Sencun Zhu, Peng Liu, Suresh Chari, and Ce Zhang. 2016. Android Malware Development on Public Malware Scanning Platforms: A Large-scale Data-driven Study. In *Proceedings of the 2016 IEEE International Conference on Big Data (Big Data) (BIG DATA '16)*. IEEE Computer Society, Washington, DC, USA.
[18] Grégoire Jacob, Paolo Milani Comparetti, Matthias Neugschwandtner, Christopher Kruegel, and Giovanni Vigna. 2012. A Static, Packer-Agnostic Filter to Detect Similar Malware Samples. In *DIMVA (Lecture Notes in Computer Science)*, Vol. 7591. Springer, 102–122.

[19] Jiyong Jang, David Brumley, and Shobha Venkataraman. 2011. BitShred: Feature Hashing Malware for Scalable Triage and Semantic Analysis. In *Proceedings of the 18th ACM Conference on Computer and Communications Security (CCS '11)*.

[20] Jiyong Jang, Maverick Woo, and David Brumley. 2013. Towards Automatic Software Lineage Inference. In *Presented as part of the 22nd USENIX Security Symposium (USENIX Security 13)*. USENIX, Washington, D.C., 81–96.

[21] Eric Jones, Travis Oliphant, Pearu Peterson, et al. 2016. SciPy: Open source scientific tools for Python 2001–2012. *URL http://www.scipy.org* (2016).

[22] Sandeep Karanth, Srivatsan Laxman, Prasad Naldurg, Ramarathnam Venkatesan, John Lambert, and Jinwook Shin. 2011. ZDVUE: prioritization of javascript attacks to discover new vulnerabilities.. In *AISec*. ACM, 31–42.

[23] Doowon Kim, Bum Jun Kwon, and Tudor Dumitraş. 2017. Certified Malware: Measuring Breaches of Trust in the Windows Code-Signing PKI. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security (CCS '17)*.

[24] Kristián Kozák, Bum Jun Kwon, Doowon Kim, Christopher Gates, and Tudor Dumitraş. 2018. Issued for Abuse: Measuring the Underground Trade in Code Signing Certificate. *arXiv preprint arXiv:1803.02931* (2018).

[25] Bum Jun Kwon, Jayanta Mondal, Jiyong Jang, Leyla Bilge, and Tudor Dumitraş. 2015. The Dropper Effect: Insights into Malware Distribution with Downloader Graph Analytics. In *Proceedings of the 22Nd ACM SIGSAC Conference on Computer and Communications Security (CCS '15)*.

[26] Chaz Lever, Platon Kotzias, Davide Balzarotti, Juan Caballero, and Manos Antonakakis. 2017. A Lustrum of Malware Network Communication: Evolution and Insights. In *Proceedings of the IEEE Symposium on Security and Privacy*. IEEE Computer Society.

[27] Martina Lindorfer, Alessandro Di Federico, Federico Maggi, Paolo Milani Comparetti, and Stefano Zanero. 2012. Lines of Malicious Code: Insights Into the Malicious Software Industry. In *Proceedings of the 28th Annual Computer Security Applications Conference (ACSAC)*.

[28] Martina Lindorfer, Clemens Kolbitsch, and Paolo Milani Comparetti. 2011. Detecting Environment-Sensitive Malware. In *Proceedings of the 14th International Symposium on Recent Advances in Intrusion Detection (RAID)*.

[29] Martina Lindorfer, Matthias Neugschwandtner, Lukas Weichselbaum, Yanick Fratantonio, Victor Van der Veen, and Christian Platzer. 2014. Andrubis - 1,000,000 Apps Later: A View on Current Android Malware Behaviors. In *Proceedings of the 3rd International Workshop on Building Analysis Datasets and Gathering Experience Returns for Security (BADGERS)*.

[30] Jonathan Oliver, Chun Cheng, and Yanggui Chen. 2013. TLSH–A Locality Sensitive Hash. In *Cybercrime and Trustworthy Computing Workshop (CTC), 2013 Fourth*. IEEE, 7–13.

[31] Konrad Rieck, Thorsten Holz, Carsten Willems, Patrick Düssel, and Pavel Laskov. 2008. Learning and classification of malware behavior. In *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*. Springer, 108–125.

[32] Konrad Rieck, Philipp Trinius, Carsten Willems, and Thorsten Holz. 2011. Automatic analysis of malware behavior using machine learning. *Journal of Computer Security* 19, 4 (2011), 639–668.

[33] Marcos Sebastián, Richard Rivera, Platon Kotzias, and Juan Caballero. 2016. AVclass: A Tool for Massive Malware Labeling. In *International Symposium on Research in Attacks, Intrusions, and Defenses*. Springer, 230–253.

[34] Xabier Ugarte-Pedrero, Davide Balzarotti, Igor Santos, and Pablo G. Bringas. 2015. [SoK] Deep Packer Inspection: A Longitudinal Study of the Complexity of Run-Time Packers. In *Proceedings of the IEEE Symposium on Security and Privacy*. IEEE Computer Society.

[35] George D Webster, Bojan Kolosnjaji, Christian von Pentz, Julian Kirsch, Zachary D Hanif, Apostolis Zarras, and Claudia Eckert. 2017. Finding the Needle: A Study of the PE32 Rich Header and Respective Malware Triage. In *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*. Springer, 119–138.

[36] Georg Wicherski. 2009. peHash: A Novel Approach to Fast Malware Clustering. In *Proceedings of the 2Nd USENIX Conference on Large-scale Exploits and Emergent Threats: Botnets, Spyware, Worms, and More (LEET'09)*. USENIX Association.