MÁSTER UNIVERSITARIO EN CIBERSEGURIDAD

Hunting for New Threats in a Feed of Malicious Samples

TRABAJO FIN DE MÁSTER

Autor: Kevin van Liebergen Ávila

Director: Juan Caballero

Tutor: Javier Junquera Sánchez

Universidad de Alcalá

2022

Escuela de Posgrado

For the unconditional support of my family along this road. . .

“The problem of viruses is temporary and will be solved in two years.”
John McAfee, 1945-2021

Acknowledgements

"Whatever you acknowledgment gains the legitimacy to exist within your world."

Steven Redhead

First, I would like to thank the University of Alcalá, which has seen me enter as an unlearned person, and now sees me leaving as an experienced one; later, thank my supervisor Juan Caballero for allowing me to work with him and to work at IMDEA Software Institute. To my family, who always encouraged me, to the people I have met in these years, also to you, for reading this and giving it more value. And lastly to you, for your advice, your patience, and your faith.

I hope you like it!

Resumen

Hoy en día, las compañias de seguridad recolectan cantidades masivas de malware y otros posibles ficheros benignos. Encontrar amenazas interesantes entre millones de ficheros recolectados es un gran desafío. Una de las plataformas de seguridad más populares, VirusTotal (VT), permite consultar informes de archivos que los usuarios envían. En este proyecto profundizaremos en el feed de ficheros de VT, analizamos 328.3M de reports de archivos escaneados por VT durante un año, que pertenecen a 235.7M de muestras y observamos que 209.6M de muestras son nuevas (89%). Utilizamos los reports de un año para caracterizar el VT Feed, y lo comparamos con la telemetría de uno de los motores de antivirus más grandes del planeta. Utilizamos ambos datasets para responder a reponder a estas preguntas: ¿Cómo de diverso es el feed? ¿Cuál es la distribución de los tipos de ficheros a lo largo del año? ¿Cuál de ambas plataformas detecta antes los archivos maliciosos? ¿Podemos detectar archivos maliciosos detectados por VirusTotal pero no por el motor de antivirus de la telemetría? ¿Cuál es la distribución del malware a lo largo de un año?

A continuación, analizamos 3 estrategias de clustering sobre Windows y APKs ground truths datasets, Hierarchical DBSCAN (HDBSCAN), HAC-T, un HAC mejorado que agrupa sobre TLSH, que reduce la complejidad de $\mathcal{O}(n^2)$ a $\mathcal{O}(n \log n)$, y Feature Value Grouping (FVG). Consideramos que solo HAC-T y FVG producen clustering de alta precisión. Nuestros resultados muestran que FVG es la única estrategia escalable sobre el VT File Feed dataset de un año.

Ademas, hemos desarrollado un técnica novedosa de threat hunting para identificar muestras maliciosas que supuestamente son benignas, por ejemplo, sin detecciones por motores de AV. Cuando lo aplicamos sobre los 235M del VT feed, nuestra encontramos 190K muestras benignas (no detectadas por ninguna empresa de antivirus) que pertenecen a 29K clústers maliciosos, es decir, la mayoría de las muestras de los clústers son maliciosos.

Palabras clave: Threat Hunting, Malware Hunting, Triage, Malware prioritization.

Abstract

Nowadays, security companies collect massive amounts of malware and other possibly benign files. Finding interesting threats among many millions of files collected is a very challenging task. One of the most popular security platforms, VirusTotal (VT), allows querying for reports of files that the users has submitted. VT offers the VT File Feed (i.e., a stream of reports), in this project we deep dive into the VT File Feed, we analyze 328.3M reports scanned by VirusTotal during one year, that belongs to 235.7M samples, we observe that 209.6M samples were new (89%). We use the one-year reports to characterize the VirusTotal Feed, and we compare it with the telemetry of a large antivirus vendor. With both datasets we want to answer the following questions: How diverse is the feed? What is the filetype distribution over a year? Which of both platforms detects earlier malicious files? Could we detect malicious files detected by VirusTotal but not by the large security vendor telemetry? What is the malware distribution over one year?

Then, we evaluate three clustering approaches over windows and apk ground truth datasets, Hierarchical DBSCAN (HDBSCAN), HAC-T, an improved Hierarchical Agglomerative Clustering (HAC) over TLSH that reduces complexity from $\mathcal{O}(n^2)$ to $\mathcal{O}(n \log n)$, and Feature Value Grouping (FVG). We conclude that only HAC-T and FVG produces highly precission clusterings. Our results show that FVG is the only approach that scales the full one-year VT File Feed dataset.

Then, we develop a novel threat hunting approach to identify malicious samples that were supposedly benign, i.e., have zero detections by AV engines. When applied on 235M samples in the VT feed, our approach identifies 190K possibly not-so-benign samples that belong to 29K malicious clusters, i.e., most cluster samples are malicious.

Keywords: Threat Hunting, Malware Hunting, Triage, Malware prioritization.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

There is nothing more difficult to take in hand, more perilous to conduct, or more uncertain in its success, than to take the lead in the introduction of a new order of things.

Niccolo Machiavelli

A malware (malicious software) is a piece of code oriented to perform harmful actions and establish a presence on an infected device (e.g., mobile, PC, server). Nowadays, there are many malware types, e.g., *ransomware*[1], *spyware*[2], *worm*[3], *adware*[4], and *trojan* [5]. The amount of new malware keeps increasing, as well as its complexity [1]. To tackle malware, cyberdefenses in endpoints have been developed such as antivirus (AV) engines. An AV engine is a program responsible for detecting and deleting malware in devices. AV vendors have to analyze in detail new emerging malware. For this purpose, antivirus vendors need to collect, detect, classify, and analyze vast quantities of malware samples. While, the amount of malware samples keeps growing, fortunately many new samples share characteristics with older samples. For example, new malware samples are oftentimes polymorphic variants of the same family. AV engines may differ in the detection and labeling from other AV engines, for example, one AV engine may detect a sample as malicious while another one does not, or one AV engine may label a sample as *vobfus* family while other AV engine labels it as *virut*.

For these reasons, there exist online scanners such as VirusTotal (VT), which receive file submissions from users, and examine them with many AV engines. The results can be available publicly or with a commercial API. VT offers the VT File Feed commercial service, a stream of users submission, VT collects from up to 2M of samples every day [2]. The huge amount of malware makes it challenging to find interesting threats. These new threats could be a sample or a cluster of samples (i.e., group of samples with common features). This challenge is increased if we combine it with the limited number of malware analysts, and that the AV engines detection module are not perfect.

This scenario demands the creation of a new threat hunting approach. Prior threat hunting works define threat hunting depending on their goal. While some articles [3] define threat hunting as the process of seeking proactively indicators of compromise in logs that have not been detected in a network, other

[1] https://www.kaspersky.com/resource-center/definitions/what-is-ransomware
[2] https://www.avast.com/es-es/c-spyware
[3] https://www.kaspersky.es/resource-center/threats/viruses-worms
[4] https://es.malwarebytes.com/adware/
[5] https://www.kaspersky.com/resource-center/threats/trojans

ones define threat hunting as the action of recording all logs in a system during the first ten seconds of execution of an application to detect if it's malicious or not [4]. In this work we define threat hunting as the challenge of finding new threats in the feed of samples collected by security companies.

This work (1) presents a novel threat hunting approach, which finds interesting samples or clusters of samples, to be sent to the malware analysts. We consider two threat hunting applications for the tool:

- **Prioritize interesting clusters (*triage*).** We group similar samples into clusters and rank clusters according to their interests, e.g., prioritizing those where all samples are unlabeled or those that grow quickly.
- **Find interesting samples.**

  We apply our tool to 235M samples in the VT File Feed detecting 190K samples that seem to be benign (zero antivirus detections) that may be potentially malicious because they belong to malicious clusters.

The project also (2) presents a detailed measurement over one-year of data from the VT File Feed. From 2020/12/21 to 2021/12/21, we examine in total 328.3M reports and 235.7M samples. We compare the VT File Feed with the telemetry of an antivirus vendor, which contains metadata of user files observed on real devices, to examine how much malware in the VT File Feed appears on real user devices. We show that despite the telemetry having 17 times more files than the VT File Feed, it contains 9 times fewer malware.

In addition, this work (3) investigates three clustering approaches to group similar samples in the VT File Feed. The clustering approaches we examine are HDBSCAN, a Hierarchical Clustering based on density, HAC-T, a Hierarchical Agglomerative Clustering (HAC) variant based on TLSH that reduces the complexity from quadratic to $\mathcal{O}(n \log n)$, and Feature Value Grouping (FVG) that groups samples base the value of a single feature (e.g., ssdeep, TLSH, vhash, imphash, authentihash, certificate). We show that FVG and HAC-T produce clusters with high precision (90.0-99.9%) over four ground truths datasets: Malicia [5], Malsign [5], Drebin [6], and AMD [7].

1.1. Motivation and Goals

The motivation for this project comes from the need to help the analyst to find new and interesting threats in a feed of millions of file reports and to detect them as early possible.

1. The first goal is to help the analyst identify malware that requires more attention in a feed of millions of daily file reports. For this purpose, we are going to cluster the feed at large-scale for later identify interesting samples and clusters. With this *triage* we can give *early alerts* to an analyst.

   a) Evaluate different clustering approaches in order to know which one scales better in a dataset of millions of samples.

   b) Rank the clusters to prioritize those that deserve more attention.

   c) We build a tool that implements our threat hunting approach. It will work over any PC with Python installed, and will be applicable to any operating system (Windows, Linux, and macOS). Also, it is easy to use and scalable. Finally, we analyze the results obtained for demonstrating its effectiveness.

2. In the process we want to characterize the VT File Feed. We want to answer questions such as How diverse is the Feed? What is the filetype distribution over a year? Could we detect malicious files detected by VT but not by the large security vendor? What is the malware distribution over a yearly collected feed?

1.2. Thesis Organization

This thesis is organized into seven chapters. Chapter 1 is this introduction. Chapter 2 explains the state of the art including the necessary background and terminology, as well as the review of the prior work for malware clustering, detection, and labeling. Chapter 3 presents the overview, how we collect, extract, and filter the data. Chapter 4 analyzes and characterizes the VT File Feed: its volume, distribution, antivirus detection distribution, the labeling of its samples, fully undetected (FUD) samples statistics, certificate statistics, and the comparison with the AV telemetry. Chapter 5 describes how the tool works. Chapter 6 details the threat hunting. Finally, Chapter 7 presents the conclusions and possible future work.

1.3. Contributions

The main properties of this work are:

1. We characterize the VT File Feed over one year. We observe that for the 328M reports belonging to 235M samples, 89% are for new samples (i.e., VT has never scanned it before). Also, 66% of the feed samples are Windows PE, followed by javascript with 9% of feed samples and HTML with 5%. Only 11% of the uploaded samples are signed.

2. We compare the VT File Feed with the telemetry of an antivirus vendor in the same period. We measure that the antivirus vendor telemetry is 17 times larger than the VT File Feed. However, the antivirus dataset has eight times less malicious samples compared with the VT File Feed. The median delay for VT to observe new samples is 4.4 hours later than the AV telemetry.

3. We evaluate three clustering approaches using Windows and Android ground truth datasets (e.g., Malicia, Malsign, Drebin, and AMD), Hierarchical DBSCAN (HDBSCAN), an improved HAC clustering grouping by TLSH called HAC-T, and Feature Value Grouping (FVG). We cluster the VT File Feed with the samples of one year, and we show that only FVG scales to the daily input of feed samples, and HAC-T and FVG produces clusterings with high precision. The only clustering approach that can scale the VT File Feed is Feature Value Grouping (FVG).

4. The implementation of a threat hunting approach, detects malicious samples that have not been detected by any antivirus vendor, but appear in a malicious cluster, we show that 190K supposedly benign samples belong to 29K malicious clusters.

Chapter 2

State of the Art

You can always do more than you think you can

John Wooden

This chapter presents previous works, their characteristics and limitations. The chapter is divided in 2 sections. Section 2.1 presents prior works and Section 2.2 provides a summary of the surveyed research and presents a brief discuss for each paper.

2.1. Background

In this section we provide a brief summary on malware types, continuing by the types of analysis to finalize explaining how the signatures and the fuzzy hashes with his comparison functions works.

Gibert et al. [8] provides a systematic overview of machine learning techniques, it also provides a detailed malware background to know how it works each part. Depending on their target, goal, and propagation we can divide malware into:

- Adware. This type of software shows online advertisements and generate revenue for its developer.
- Backdoor. Typically install a software on an infected host t connect back to it.
- Bot. A program that automatically performs different actions such as Distributed Denial of Service (DDoS). When a group of bots belongs to the same owner, this group is called botnet.
- Downloader. Small program whose goal is to download and launches other malware in the device.
- Ransomware. Encrypts all data on the system and demands money to decrypt the data.
- Rootkit. It is a software oriented to hide other malicious programs. Often operates at kernel level.
- Virus. Malware that can replicate to other devices by human activity.
- Worm. It is a virus that self-replicates to other devices, typically through a vulnerability, which is the main difference between this type of malware and viruses.

- Potentially unwanted program (PUP)/ Potentially Unwanted Application (PUA). A software that can compromise privacy of a user, can include application that shows intrusive ads (adware), or a infostealer, program that spies and recollects information, also allows reading from the keyboard or recording from the webcam.

When we want to analyze those types of malware, to know their goal, functionality, potential and impact we have two types of malware analysis:

- Static Analysis. Consists of examining the code without executing it. This is the safest way to analyze malware, because executing the code could infect your device.
- Dynamic Analysis. It involves executing the software to know step by step what it does, monitoring its activities (e.g., persistence, privilege escalation), the malware should be run in a secure environment (virtual machine or sandbox).

Samples provided by VirusTotal can be signed and unsigned, code signing is a technique that inserts digital signatures into a file and another system validates it.

- Windows files (e.g., peexe) supports this type of technique with Windows Authenticode. An Authenticode signature is a Microsoft code signing designed to verify the integrity, authenticity, and timeliness of a signed application. To generate an Authenticode hash [9] we have to perform the following Signing Process as detailed in Figure 2.1: The publisher acquires a code signing certificate (1,2) by providing his personal information, then the publisher signs the code with his certificate (3,4) and optionally asks the TSA to sign the code timestamped (5,6, optional) to finally distribute it to the clients (7).

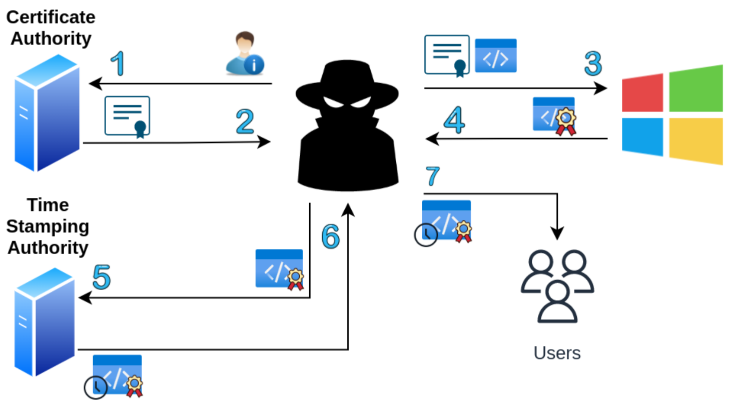

Figure 2.1: Code signing process

- Android requires that all applications be signed with a certificate to get installed in a device. Android also can sign by itself his own certificate and this presents several disadvantages because it is not dependent on a Certificate Authority, so nobody can revoke your certificate and it cannot expire.
- In Linux (elf files) there are not code signing, however, approaches have been developed to sign a file [10].
- Apple files (e.g., ipa, macho, dmg) sign and encrypt. The file is signed with Apple's key and encrypted with user's key, so, always are signed with the same certificate, Apple's certificate in App Store.

- Other signed files (e.g., msp, lnk, msi, cab, zlib, xml) also uses code signing for check the validity of the application. Some PDFs files are signed but actually VirusTotal is unable to extract his certificate.

That is why generic certificates have been appeared, certificates that is easy that anyone can use, it can be of four types (mostly Android):

1. Android Open Source Project (AOSP) keys. Android provides four keys (platform, shared, media, and release) to test the development application and maintain the security of the applications, so, everyone can use it to sign his apps.

2. Platform keys. A hardware vendor (e.g., Samsung, Nokia) signs all the operative system, but it includes third party app, so, some application developed by X company will be signed by a hardware vendor. So, it is usual that many software is signed by the same certificate.

3. Online Application Generator (OAG). Platform that simplify the mobile application creation, everyone can create an application and the platform inserts the same certificate to every app. Kotzias et al. [11] demonstrates that OAGs publish unwanted apps from their clients, so it facilitates the encryption and signed malware creation. So, it is very important to extract certificate to attribute apps to malicious owners [12].

4. Market keys. A market sign all applications with his key (e.g., App store).

In the work we have created a list with generic certificates selected manually, we will use this list to exclude some samples with this certificates to have a balanced clusters if we group by certificate thumbprint. If we cluster with generic certificates we will have unbalanced clusters with many malware families in a group.

Thanks to the certificates we can extract important information, so much that clustering tools have been developed to, for instance, to filter whether which samples are PUP or malware based on their certificates. Kotzias et al. [9] concluded that from 356 K samples analyzed, 88%-95% are PUP and malware the rest of it (5%-12%). Moreover, 17% of malware samples and 15% of PUP have been revoked.

A fuzzy hash /similarity digest is a type of file compression, it pretends to automate the process of cluster similar malwares. Each fuzzy hash has its own comparison function that allow to extract the percentage of similarity between two files to identify similar malwares (e.g., malware variants).

The difference between cryptographic hashes is that if a bit is modified over the file, the fuzzy hash changes only a small number of hash digits, instead the whole hash, as is the case with cryptographic hashes.

However, if we want to make some clustering, we have to attend at some types of fuzzy hashes, those hashes have been analyzed in prior works such as Pagani et al. [13] and we can divide these types depending on the granularity into:

- *Context-Triggered Piecewise Hashing.* The idea is to find collisions in hashed blocks of fixed size (e.g., search same chapters in a book). The most common hash is **ssdeep** [14] (developed in 2006) due to is being used in a lot of applications, however, falls in a lot of tasks [13].

- *Statistically Improbable Features.* Based on the search for the sequence of bytes in common (e.g., words, instructions). The most popular *Statistically Improbable Features* hash is **sdhash** (2010), which split into sequence of 64 bytes (called *features*). For instance, if in the previous hash we were searching similarities on the whole file (e.g., chapter in a book, now we are searching phrases in a text file). **Sdhash** works properly comparing the same files compiled with different compiler options and different toolchains.

- *N-Grams* [15]. It is based on the frequency distribution of substrings of n-bytes in a file (e.g., same words). The most popular hashes are **nilsimsa** [16] (2004) and **tlsh** [17] (2013) because of his reliability if we compare different versions of the same software (binaries changes).

These similarity digests have their own comparison function to calculate the similarity between two files, for instance [17]:

- Ssdeep calculates the string edit distance between two digests and outputs a score from 0 (mismatch) to 100 (perfect match).
- Sdhash uses Bloom filter and hamming distance function normalized to know the similarity score, being 0 a mismatch and 100 score a perfect match.
- Nilsimsa was created as an anti-spam focused locality-sensitive hashing method which uses hamming distance function to know the similarity between two digests, from 0 (dissimilar files) to 128 (identical or very similar objects).
- Finally, TLSH creates its own function similar to hamming distance with some variations (adding the distance of the digest headers and the distance of the digest bodies). It outputs a number between 0 (perfect match) to 100 (mismatch).

Ssdeep is worst because it was not designed to work over a large set, however TLSH outperforms because can handle large digests of file hashes, it also outperforms ssdeep both in scalability and accuracy.

2.2. Related Work

Over time, malware has become more sophisticated, making it increasingly difficult to detect due to the implementation of polymorphism (changing its own code while keeping its original algorithm intact), packing (compressing its code several times) and encryption (decrypting some parts of the code and executing automatically) techniques. These techniques have led to a rapid increase in the growth and difficulty of malware because some solutions were based on comparing malware signatures. These techniques cause software to go undetected as malicious or variants of them, so researches and also antivirus engine have implemented machine learning techniques to speed up the detecting and labeling processes.

However, each AV engine has its own taxonomy and because the different naming conventions such CARO [18] or CME [19] are not widely used, a problem arises when we want to tag large-scale samples.

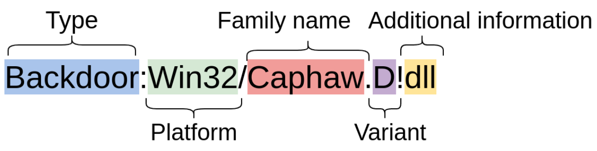

Figure 2.2: CARO Naming convention

Due to the lack of name consistency, AV-meter [20] performed a systematic study and evaluated the performance of antivirus scanners and the reliability of the tags and their detection. It showed that in order to obtain labels that really provide information, many labels are required from antivirus scanners, which is sometimes impossible.

One of the problems is that generic labels (used for labeling the family such as “generic‘, “trojan”, “worm”, “run”, and “start”) take time to populate in antivirus scans and law enforcement efforts that rely on names for attribution of online crimes can be affected by the limitations of antivirus tags. Some remedies are to unify names, make existing names meaningful, greater use of indicators of compromise and more information sharing and to make the name of a class provide more information for the malware family, based on functionality rather than the background history of the family.

Attending to Linux code signing approaches, signatures of ELF files [10] are computed using the MD5 and the RSA signature. The signature is added at ELF as another segment. And when an ELF file is loaded, the ELF manager extract the module and check the referenced segment. If the verification fails the application is aborted.

2.2.1. Triage Works

MAST [21] is a triage framework oriented to malware analysis for mobile applications. In Figure 2.3 we can see MAST architecture. MAST identifies attributes (e.g., permissions, intents, native code information) related with security, it develops through Multiple Correspondence Analysis (MCA), a data analysis technique to determine correlation between multiples data and make multiples polls for generate risk indicators to obtain a MAST scoring of applications suspiciousness. It scores samples according to the suspiciousness they have, samples with higher suspiciousness will have a higher score and then they will send the sample to the analyst. It score regardless of the purpose of the malware, in contrast to Spotlight [22], which depending on its objective, ranks it one score or another.

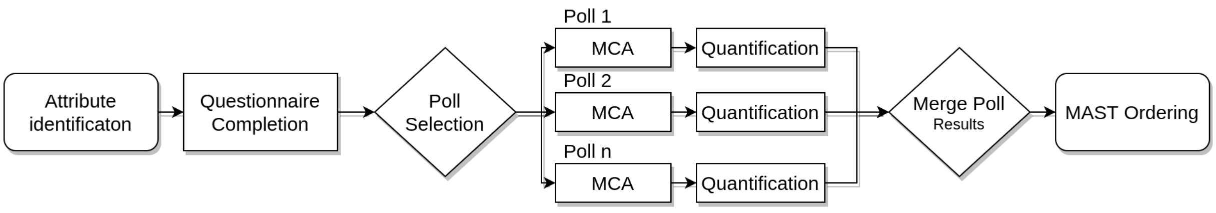

Figure 2.3: MAST overview

Similar to Threat Hunting tools, one project related to our work is Spotlight [22], a large-scale malware lead-generation framework, first identifies samples from well-known families and then cluster the remaining unknown samples and finally ranks the clusters.

While Spotlight and our work uses samples as input, VT extract features that we use for our work. One of the biggest differences between Spotlight and our work is that Spotlight uses dynamic and static malware analysis and uses two scoring functions, a volume based that assign higher rank depending the size cluster and a dynamic score, this function depends on the cluster to analyze (e.g., ad-fraud), and we search for benign samples in malicious clusters.

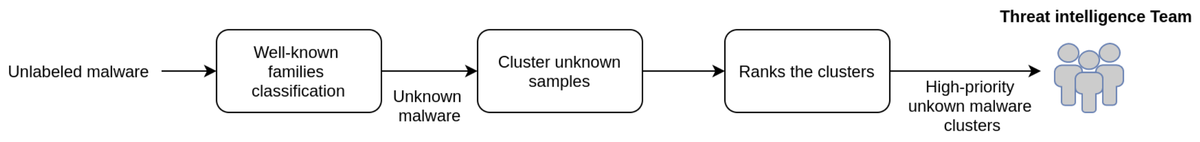

Figure 2.4: Spotlight overview

2.2.2. Clustering Works

When we attend to malware triage and/or clustering works, we have to distinguish if the input have only malicious samples or if we have firstly to filter between benign and malign samples to subsequently apply a cluster and scoring function.

BitShred [23] is a large-scale malware clustering tool, it uses feature hashing (allows to drastically reduce data dimensions by vectorizing features), also BitShred extracts correlated features between family malware and samples using co-clustering techniques (why malware is similar by grouping features simultaneously).

Instead to analyze a malware deeply, BitShred analyzes the existing features to speed up the clustering process and identifies semantic features (co-clustering). BitShred clusterizes a malware set and then performs inter-malware comparison and feature analysis, it measures similarity of a new sample compared to existing clusters to identify the nearest neighbours. BitShred also creates an incremental clustering, although the main difference with our project is that on the research it does not specify the scoring, if it does. We can see in Figure 2.5 the overview tool.

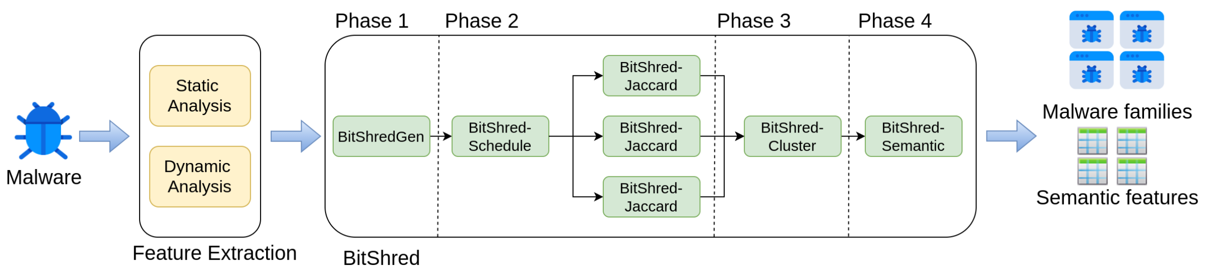

Figure 2.5: BitShred overview

One of the mainly differences between our project is that they does not have any ground truth for clustering accuracy, however the metric used is to know how close the feature hashes version of clustering comes if we compare to a exact Jaccard clustering. Our approach is evaluated with a small dataset over three ground truths, to demonstrate with which features clusters better.

For scalability reasons, it uses Locality Sensitive Hashing and feature hashing, Locality Sensitive Hashing is complementary to feature hashing because it reduces the number of items and feature hashing reduces the number of features.

VILO [24] is a clustering classifier that fit the sample of the nearest neighbors building giving a new malicious sample. First, given a set of malicious samples builds a database of feature vectors.

AVCLASS [25] appeared to tag a large-scale of malware samples, reports the most common family name for each sample given the antivirus engines extracted by VirusTotal. AVCLASS appeared due to the inconsistency of naming conventions explained before.

For Android malware, as AVCLASS [25], there is EUPHONY [26], an automated labeling system that extracts antivirus labels and analyzes associations between labels from different vendors to systematically unify common samples into family malware groups.

With EUPHONY the analyst does not need to know a large amount of malware details in order to analyze new samples, EUPHONY can operate with a minimum lexical knowledge of malware labels with high accuracy. In addition, it can adjust granularity between the malware families.

Zhu et al. [27] evaluate different labels provided by different AV engines, they demonstrate that many AV engines are correlated with others one. For instance, McAfee and McAfee-GW-Edition belongs to the same vendor. Although Microsoft and Cyren owners are different from McAfee, their labels are a 97% match.

With this information it should not consider each AV engine as independent and be considered as “redundant votes” and reduce their weight. Some high reputation AV engine are not usually more accurate, four of them produces a lot of false positives when a benign file is obfuscated (e..g, Symantec).

So, AVCLASS2 [28] appeared, in addition to AvClass [25], it reports also additional information (such as file properties, behaviors, malware class).

2.2.3. Detection Works

SigMal [29] is a malware detection framework based on signals processing techniques (e.g., image, video), it extracts noise-resistant signatures from the samples (extracts packed as well as unpacked). That is, it builds features vectors from bidimensional images to capture section content from a PE file. As we can see in Figure 2.6 given an unknown file, it calculates the nearest neighbors from a feature vector database and produces a ternary decision, malware, benign or unknown sample.

One of SigMal’s limitations is that it cannot detect zero-day malware, whose structure is different from that of the previous malwares analyzed. Also it will be hard to find similarity between encrypted samples with strong cryptographic methods (e.g., AES) that are been used by packers.

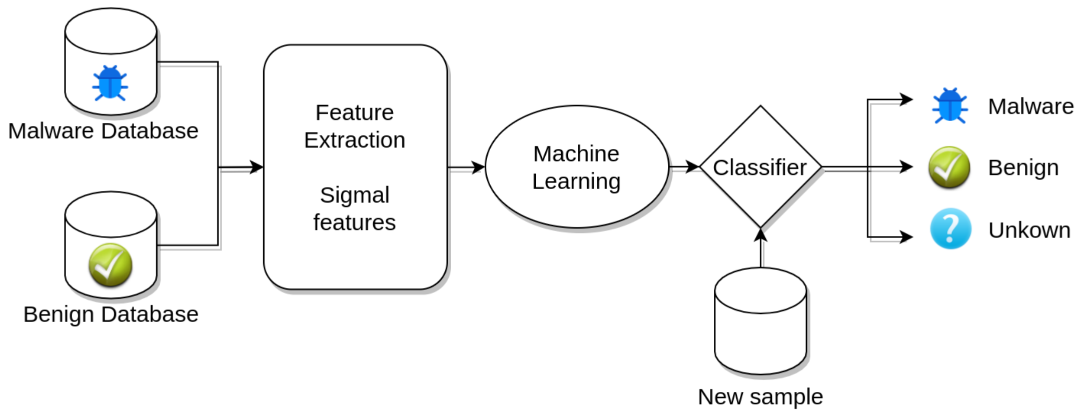

Figure 2.6: SigMal overview

Apart from that, there have been some systematic studies of PUP such as Kotzias et al. [30], they discovered that 54% of hosts have PUP installed and 65% of PUP is being installed by another PUP. Also, they concluded PUP distribution are disjoint from malware distribution.

Buyukkayhan et al. [31] discusses how Windows endpoints that are monitored can be used to detect suspicious software modules that have been overlooked by other defense mechanisms, given a set of a priori samples without knowing whether they are malicious or not, this paper analyzes a heterogeneous environment with a limited ground truth (low training samples). Through different clusterings and outlier detection, they detect modules with suspicious behavior as well as modules impersonating benign programs, and perform an endpoint analysis. The results have shown that algorithms have been developed to identify modules similar to the modules that have blacklists.

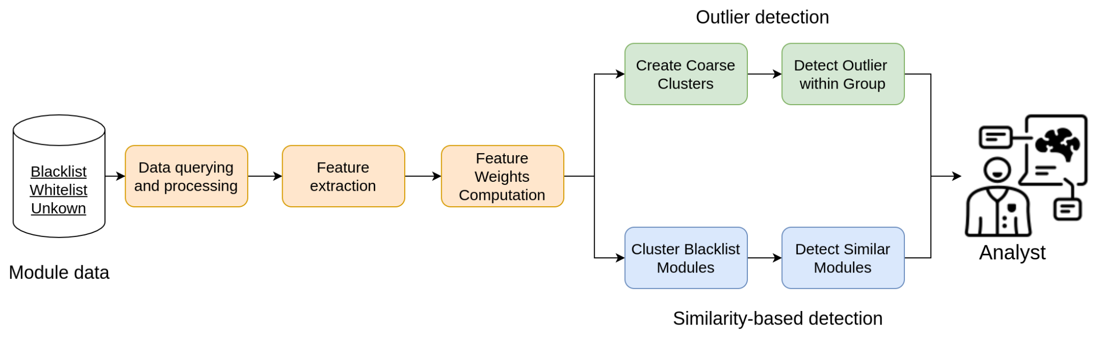
Outlier detection
Blacklist
Whitelist
Unkown
Module data
Data querying and processing
Feature extraction
Feature Weights Computation
Create Coarse Clusters
Detect Outlier within Group
Cluster Blacklist Modules
Detect Similar Modules
Similarity-based detection
Analyst

Figure 2.7: Buyukkayhan et al. overview

Other systematic study try to cluster samples depending on their TLS certificate, for instance, Kim et al. [32] define a threat model to highlight the three types of weaknesses and determine the certificate type when it belongs to a malware and we can see it in Figure 2.8:

- Malformed Certificate. When the malware is not signed properly.
- Stolen Certificate. When the malware is signed properly and there exists legitimate samples with the same certificate.
- Fraudulent or Shell company. When the malware is properly signed and no legitimate samples with that certificate exist.

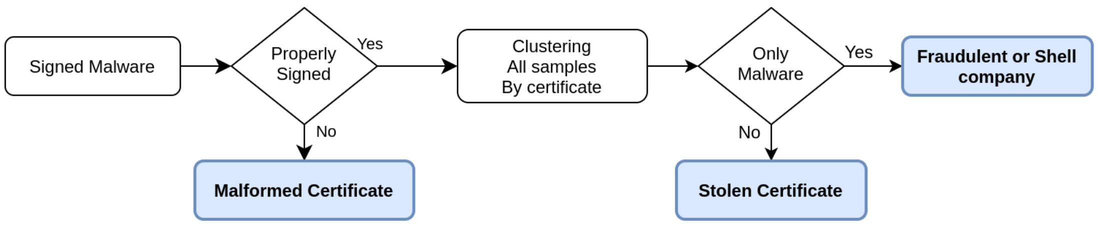

Figure 2.8: Flowchart of the abuse detection algorithm

This is why mitigations such as certificate revocation have been developed in the face of threats against PKI, however, the nature of the infrastructure make hard the revocation effectiveness, that is why Kim et al. [33] claim that effective revocation is based on:

- Discovery of new compromised certificates
- Revoke the certificate correctly
- Disseminating the revoked certificate information to customers

They evaluated that improper revocation due to incorrect dates resulted in signed malware being able to continue to function even with the certificate revoked. On average, a CA revokes a certificate 5.6 months after it has been compromised, and in some cases the dates of the revocation CAs were often incorrect, leading to malware continuing to operate.

Also, if the CAs have successfully revoked a certificate, clients may still be vulnerable to CRL and OCSP mismanagement because in some cases clients have not checked verification status due to: (1) Lack of OCSP and CRLs points, (2) unreachable CRL and OCSP points, (3) CRL no updated, (4) removed revoked certificates from the CRL, (5) inconsistent responses from OCSP and CRLs, and (6) unknown OCSP responses or without authorization.

Chapter 3

Overview

After all, just one virus on a computer is one too many.

Glenn Turner

This structure chapter is divided in four sections, Section 3.1 explains the existing and public malware sources to collect samples, Section 3.2 defines what type of data we collect, the collection date range, some summarized statistics about the reports, samples and clusters, for later understand the large scale that we are facing at, Section 3.3 explains the overview of the approach, the steps involved in the clustering, and finally, Section 3.4 explains the features we collect from VirusTotal.

3.1. Malware Sources

The are many public datasets with periodic updates, which you can ask for new samples and download binaries for analyze them (instead of query to the limited VirusTotal API queries), in fact, the most important datasets are VirusShare, MalShare, and MalwareBazaar, including VirusTotal. To get a global and a general idea about these datasets size, how diverse is it, and the file distribution, in order to select which malware source we are going to use Table 3.1 shows the summary of each malware feed, *Samples* column includes all samples stored on 2022/06/22. *Volume* corresponds to new (i.e., previously unknown) samples added between 2020/12/21 and 2021/12/21. One interesting part is that VirusShare last package was created on March 2021, so it remains 9 months of published samples during our observation period. MalwareBazaar only store malware samples, unlike other datasets that also save PUP.

				Volume	
Feed	**Type**	**Start**	**Samples**	**New**	**Daily**
VT File Feed	File	2004-06	>2,400,000K	209,600K	1,000,000
VirusShare	Malware	2012-06-15	37,683K	1,400K	3,760.3
MalShare	Malware	2017-09-14	4,721K	442K	1,210.1
MalwareBazaar	Malware	2020-02-13	516K	178K	486.0

Table 3.1: Summary of malware feeds.

Table 3.2 shows the differential contribution, how many indicators from one dataset are in another, denoted as |A∩B|/|B|. Being A the rows and B the columns. The intersection between these three datasets are | MalShare ∩ MalwareBazaar |: 81,093, | MalShare ∩ VirusShare |: 30,462, and | MalwareBazaar ∩ VirusShare |: 23,724. Notice that there is more correlation between MalwareBazaar and MalShare than with VirusShare, in spite of VirusShare is larger. So it is possible some samples is sent to another dataset.

	VirusShare	**Malshare**	**MalwareBazaar**
VirusShare	100.00%	6.9%	13.34%
Malshare	2.21%	100.00%	45.59%
MalwareBazaar	1.72%	18.36%	100.00%

Table 3.2: Intersection pairwise for datasets with periodic updates during observation range.

Given the previous results, we can infer that VirusTotal have 2 order of magnitude larger from VirusShare, 3 order from MalShare, and 4 from MalwareBazaar, so VirusTotal es the largest dataset to collect samples. In order of collecting samples we will ask to VirusTotal the file feed, instead of VirusShare, MalShare, and MalwareBazaar. We can check that VirusTotal File Feed is a good resource to find samples, however, if you also want the binaries in a free way, the other datasets are a good resource (VirusTotal allow to download samples with the premium API).

3.2. VT File Feed Collection

We define a Report as the number of times a file appears in the Feed. Therefore, if the same sample appears five times, there will be five reports. Also, we define a Sample as the number of unique reports (filtered by cryptographic hashes). For instance, if a Report with the same certificate appears five times, there will be one sample for that certificate. We define VT reports as the VT Data provided by Virus Total in JSON format, and for that, we define VT Features the formatted VT reports in TSV format, both VT Feature and VT reports include signed and unsigned samples. We define the cluster summary as the created cluster using the VT File Feed that includes the number of reports, samples, AVCLASS labels, properties (e.g., filetype) and certificates information used only by signed samples.

The reports provided by the VT File Feed have been detected as malicious by at least 1 one antivirus engine (1 VT score) from 2020/12/21 to 2021/11/18, the last two months, from 2021/11/19/21 to 2021/12/21 the VT File Feed include reports with also 0 VT score. Notice that 60% of total reports has been detected by at least 1 antivirus engine, the other 40% of reports has no detections. For this project, the VT File Feed covers from 2020/12/21 to 2021/12/21.

We collected 328.3M reports for 235.7M samples (by unique file SHA256), as summarized in Table 3.3. Therefore, a file is analyzed 1.4 times on average over the selected time period.

Data	**All**	**peexe**	**apk**	**other**
Reports	328.3M	220.3M	15.9M	92.0M
Samples	235.7M	155.5M	8.2M	72.0M
New samples	209.6M	134.6M	5.6M	69.3M
Signed samples	13.3M	5.8M	7.5M	94.8K

Table 3.3: Summary of dataset collected from VT File Feed between 2020/12/21 and 2021/12/21.

One exception appears in our VT File Feed for some reports, on July 13th, 2021 the feed had some incomplete scans, likely due some VirusTotal infrastructure error, however on July, 17th, 2021 they reanalyzed the reports with correct information, it appear the updated information in the web page but not in the Feed because they did not correspond to a user request.

Notice that we store features extracted by VirusTotal, we could have also the samples for extracting extra information, but it wouldn't be maneagable at scale because we would require large storage access to store 600K new samples daily, that is the reason that anyone with VirusTotal API access could replicate our approach, so, it is replicable.

3.3. Approach Overview

Our approach is shown in Figure 3.1 and is deeply explained in Subsection 5.2.1.1. The clustering process takes as input our VT File Feed processed in TSV format, and outputs the VT File Feed clustered (cluster summaries) by a specific feature we choose, and the detected threats. The clusters can be made by in batch mode, that allows to quickly generated a clusters files with the information provided, and also in incremental clustering, that can give you *early alerts* to send to the analyst.

A generated cluster comprises information such as the number of samples that belongs to a cluster, number of samples with at least a 1 and 4 VT Score, among other, we can see all columns in Table 5.14.

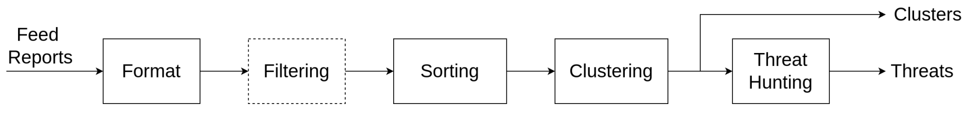

Figure 3.1: Overview

Our approach consists of 4 steps: (1) Format and Filter VT File Feed, (2) Sort VT File Feed, (3) Clusterize, and (4) Threat Hunting, that will be explained in the next subsections.

Format and Filter VT File Feed

The first step include filter reports by a specific filetype, scan date range, and reports with null feature. We also, add a *filetype* column and we escape special characters like tabulators, new lines, and carriage return.

Sort VT File Feed

Once we have our VT File Feed formatted and filtered, we sort our VT File Feed by the feature id (i.e., vhash, thumbprint), sample hash and scan date. Optionally we can also split samples by a scan date range. With this we can easily get the latest report for each sample in VirusTotal, that we suppose it will have the most accuracy results, and also it will be easier to handle the system memory while clustering

Clusterize

Taking as input the previous step output, we proceed to cluster our VirusTotal File Feed using Feature Value Grouping (FVG), the cluster can scale in time and memory at large-scale because it is sorted, by feature id, a change in the feature id indicates a new cluster. Also, we can choose to select a batch or incremental clustering.

Threat Hunting

This part will select interesting samples taking as input the cluster summaries, we define interesting samples as supposedly benign samples that belongs to a malicious clusters, for this we search samples with samples with 0 VT Score and search if the samples reach the threshold (50%) to define a malicious cluster, however, this threshold (50%) can be modified.

3.4. VT Features

We are going to work with VirusTotal [1] data, VirusTotal is an online platform for analyzing suspicious files and URLs to detect malware. VirusTotal generates the VT File Feed, a stream of analyzed files that they have collected and NortonLifeLock Research Group has provided us to work on this project. VT offers an API to connect and download the reports that people has uploaded to its platform, these reports has specific information and it is provided in JSON format, however NortonLifeLock Research Group provide us the VT File Feed in TSV format for legibility.

The VirusTotal File Feed collects several features we can see in Table 3.4, excepting the AVCLASS columns, we have also divided the columns depending whether are for peexe or android samples.

We can see that the VirusTotal File Feed contains cryptographic hashes such as MD5, SHA128, SHA256, or the MD5 hash of an icon, it also contains TrID (tool to identify filetypes based on their binary signatures) information, VirusTotal information such as vt_score (can be modified over time), first seen date VirusTotal has seen a report, the scan date on which a user has uploaded the report and vt_tags, tags that Virus Total has extracted for that report (can be modified over time). Notice that we restrict our analysis to features VirusTotal provide us, and for that, other analysts with access to VirusTotal File Feed can replicate our approach.

- **Cryptographic hashes.** These hash types can be applied over the whole file. The main and most famous cryptographic hashes are MD5, SHA1, SHA256. The importance of these types of hashes is for identify files, if you change one bit the whole cryptographic hash changes, so it has to be strong to collisions. Also, these hash have a fixed length. Attending to our VirusTotal File Feed, some features such as *imphash* are the MD5 of the import table in Windows executables. *Rich PE header hash* is the MD5 of the Header in PE, that includes information about the compilation chain. *cert_thumbprint* is the SHA256 of a sample in PEM format, *authentihash* is the SHA256 of the Windows executable excluding the code signing fields. *icon_md5* is the MD5 of the icon embedded on the files.

- **Fuzzy hashes.** As I have explained in Section 2.1, fuzzy hashes has other advantages facing cryptographic hashes. This fuzzy hashes can be applied over the whole file (e.g., *tlsh*, *ssdeep*) or over specific parts of the file, such as the image icon (*dhash*).

- **Structural hashes.** *vhash* is VT proprietary hash, every VirusTotal cluster new samples and clusters by this hash. The samples are clustered by the same vhash, so we use a equality comparison to cluster samples.

- **Dates.** VT extracts three dates, *lsubm_date*, last time when a user request to analyze the file, *scan_date*, the time when VirusTotal analyzed the file, in our VT File Feed, the *scan_date* must be in our observation range, and *fseen_date*, the first time VirusTotal analyzed the sample.

- **Engine scans.** The *vt_score* is the number of antivirus engines that has detected as malicious a file. Also, the output of the engine scans are tags extracted for one sample, this is processed by AVCLASS tool to extract the AvClass columns.

- **Program name.** The features used to capture the program name information are *auth_code_orig_name*, optionally in Windows executable files in the Authenticode signature, *exiftool_orig_name*, name extracted by the exiftool tool, *package_name*, identifier for Android apps, and *vt_meaningful_name*, extracted by VirusTotal.

[1] https://www.virustotal.com/

Derived Features

- **Filetype.** The way to get the filetype is not so simple, VirusTotal does not extract a single filetype field, this has some disadvantage because we have manually to extract the filetype but we also can apply the granularity we want, for instance, we join EXE, DLL, OCX, CPL as Windows peexe samples, and DOC, DOCX under doc, that is, word files.

  For the filetype columns (i.e., *trid_file_type*, extracted by TrID tool (highly granular), *vt_tags*, list of tags extracted by VirusTotal, and *vt_meaningful_name*, assigned by VirusTotal to the sample) of each report We make a scoring to determine the most filetype collected.

- **AVCLASS2.** This columns belongs to the output of AVCLASS2 tool given a list of tags provided by VirusTotal, these columns are: *avc2_family*, that belongs to the most appearing tag with FAM or UNK type, *avc2_tags* are the tags extracted by AVCLASS using all antivirus, these categories could be behaviour, family, file properties, and unknown, the *avc2_is_pup* determine if the sample is a potentially unwanted program or malware, *symc_avc2_tags* are tags extracted by Symantec antivirus vendor with AVCLASS2, and *avira_avc2_tags* are tags extracted by Avira antivirus vendor with AVCLASS2.

- **Antivirus vendors.** *symc_vt_count* and *avira_vt_count* give us information whether samples have been detected as malicious (1) or not (0).

General information	**Columns**	**Win.**	**And.**
Cryptographic hashes	sha2	✓	✓
	sha1	✓	✓
	md5	✓	✓
	leaf_cert_thumbprint	✓	✓
	icon_md5	✓	✓
	authentihash	✓	✗
	imphash	✓	✗
	rich_pe_header_hash	✓	✗
Fuzzy hashes	ssdeep	✓	✓
	tlsh	✓	✓
	dhash	✓	✓
Structural hashes	vhash	✓	✓
Dates	lsubm_date	✓	✓
	scan_date	✓	✓
	fseen_date	✓	✓
Engine scans	vt_score	✓	✓
Filetype	vt_tags	✓	✓
	trid_file_type	✓	✓
	exiftoo_file_type	✓	✓
	exiftool_original_fname	✓	✓
Program names	package_name	✗	✓
	authcode_orig_name	✓	✗
	vt_meaningful_name	✓	✓
Certificate	leaf_cert_subject	✓	✓
	leaf_cert_issuer	✓	✓
	leaf_cert_valid_from	✓	✓
	leaf_cert_valid_to	✓	✓
	sig_verification_res	✓	✗
	signing_date	✓	✗
AvClass2	avc2_family	✓	✓
	avc2_tags	✓	✓
	avc2_is_pup	✓	✓
	sym_avc2_tags	✓	✓
	avira_avc2_tags	✓	✓
Antivirus vendors	symc_vt_count	✓	✓
	avira_vt_count	✓	✓

Table 3.4: Feed columns

Chapter 4

VT File Feed Analysis

Research is creating new knowledge.

Neil Armstrong

Once we have collected, extracted and filtered the VT File Feed, we characterize and answer the questions such as How diverse is the Feed? What is the filetype distribution over a year? Who of both platforms detects earlier malicious files? Could we detect malicious files detected by VirusTotal but not by the large security vendor? What is the malware distribution over one year?

With the previous data analyzed, I have extracted some statistics about the filetype samples and the features extracted, also split by platform type. As one of the questions was the clusters evaluation (e.g., precision, recall, F1) if we group by leaf certificate thumbprint, we have also extracted Figures with information as mean, median, and standard deviation with the certificates collected in our VT File Feed.

4.1. Samples Volume

Figure 4.1 shows the VT reports, samples, and new samples that appear each day, during our observation period. The volume increase appeared from november is because we begin to include also samples with no detections.

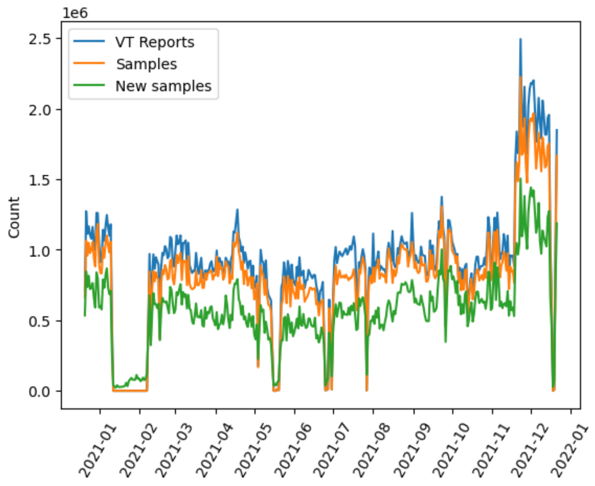

Figure 4.1: Number of daily VT reports and samples collected.

When we collect only samples with at least one detection, we got an average of 0.8M of reports, 0.7M unique reports (samples), and 0.5M of new samples, as we can see in Table 4.1.

	Mean	Median	Stdev	Max
Reports	811,584	916,914	331,508	1,374,242
Samples	731,864	821,116	300,686	1,305,521
New samples	515,455	580,984	207,324	1,001,720

Table 4.1: Daily statistics when collecting reports with at least one detection (from 2020/12/21 to 2021/11/18).

We extract the daily stats since 2021/12/21, when we collect all samples, VirusTotal collects nearly an average of 1.7M reports per day, 1.5M samples, and 1.0M of new samples, as we can see in Table 4.2. Thus, we can estimate that half of the reports (52%), samples (51%), and new samples (50%) are benign (no detections).

	Mean	Median	Stdev	Max
Reports	1,681,470	1,879,952	632,366	2,492,454
Samples	1,493,410	1,680,520	558,895	2,223,638
New samples	1,028,370	1,120,242	382,196	1,504,174

Table 4.2: Daily statistics when collecting all reports (from 2021/11/19 to 2021/12/21).

There are nearly 17.5M vhashes collected, which means that the are about 17.5M vhash clusters collected, and from those, 12.5M vhashes are new (71.4%). We can see that there is a significant difference in peexe clusters respect to apk, both in all vhashes and in the new vhashes, as we can see in Figure 4.2

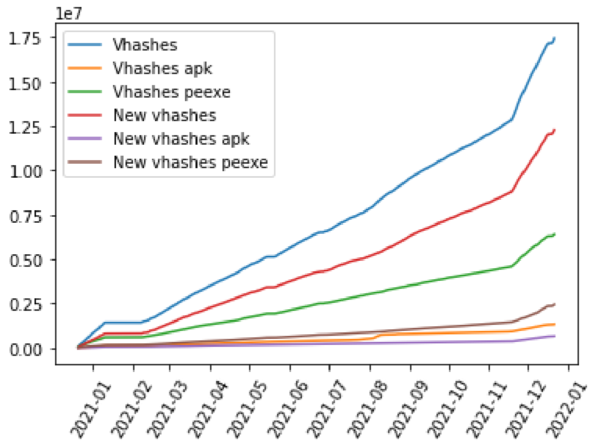

Figure 4.2: Number of accumulative vhashes over VT File Feed.

In Figure 4.3 we see the proportion of the number of scans for all collected samples by VirusTotal, almost 100% of the samples have been analyzed less than 10K times during our observation range. There is at least one sample that has been scanned more than 50K times.

Year	New samples
2006	20.6K
2007	45.0K
2008	89.3K
2009	231.9K
2010	210.1K
2011	281.9K
2012	393.0K
2013	832.2K
2014	924.2K
2015	820.5K
2016	1.8M
2017	4.1M
2018	11.1M
2019	1.8M
2020	11.5M
2021	201.5M

Table 4.3: First seen date in VirusTotal

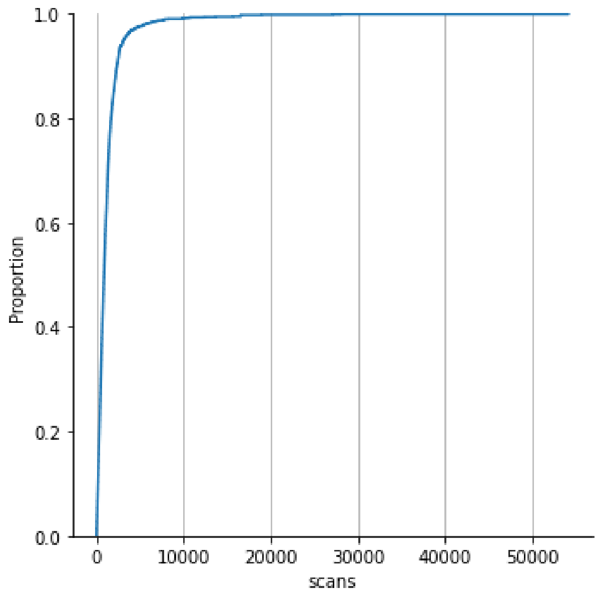

Figure 4.3: Proportion of number of scans per samples.

Table 4.3 shows the first seen a sample has been seen by VirusTotal during our observation period, splitted by year. The first collected malware by VirusTotal during our observation period was in May 22nd, 2006. The number of new samples increases in an exponential way, We have calculated the data splitting in Figure 4.4 by days, in Figure 4.5 by months and in Figure 4.6 by year, the y-axis are in logarithmic scale. Notice that in 2019 and 2020 there are a slightly decrease in the number of new samples, this could be due to antivirus vendor reduce their sharing in 2019.

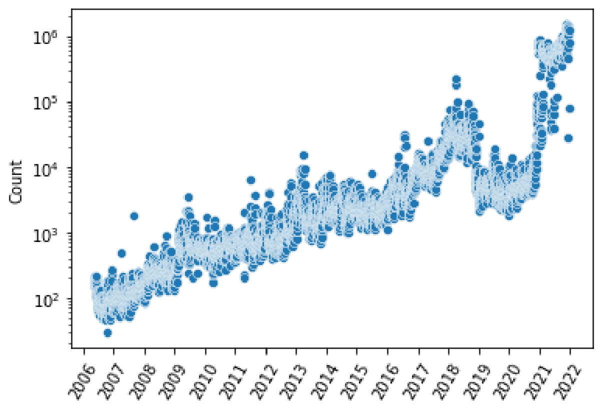

Figure 4.4: Number of samples first seen date by VirusTotal on each day.

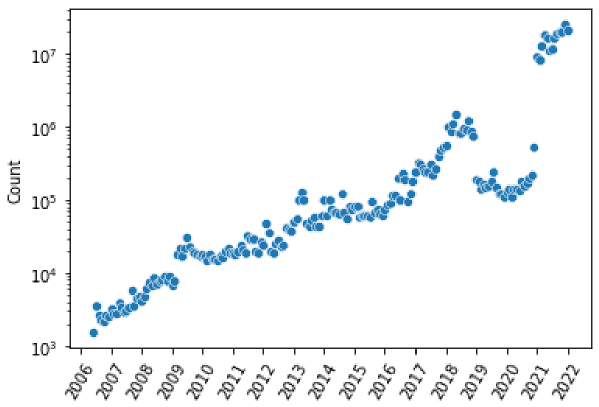

Figure 4.5: Number of samples first seen by VirusTotal on each month.

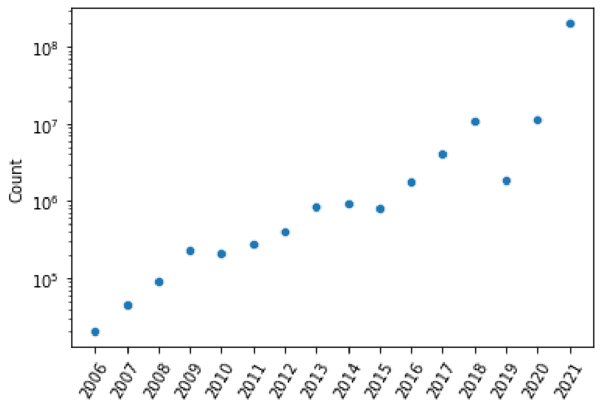

Figure 4.6: Number of samples first seen by VirusTotal on each year.

4.2. Filetype

We can see in Table 4.4 the summarized filetype data, we see how peexe files predominates in the VirusTotal platform with a 66%, peexe includes all Windows PE files (e.g., EXE, DLL, CPL, OCX) followed by other filetypes such as javascript with 8.9%, HTML with 5.3% and PDF 4.8%. The top 4 filetypes comprises the 85% of the whole VT File Feed. NULL filetype corresponds to 1.7% of the samples we can not recognize the filetype because it has not any VT tags, TrID information, and has not a meaningful filename with extension. *doc* and *xls* include also *docx* and *xlsx*, respectively. One of the possible reasons why peexe samples are significantly larger than other filetypes is because you can not inspect the file content or the private data.

4.3. Antivirus Detections

We measure the distribution of the number of antivirus detections in Figures 4.7-4.8, for all reports collected on the last month (also samples with no detections). Y-axis of Figure 4.7 is in log-scale. We study that 85% of the reports in the last month does not have any detection, and a 7% of reports have 1 detection, however, there are 9.6M samples with at least 40 detections.

Filetype	**Samples**	**Perc**
peexe	155,526,594	65.97%
javascript	21,048,404	8.93%
html	12,540,571	5.32%
pdf	11,346,815	4.81%
apk	7,992,206	3.40%
text	5,149,050	2.18%
NULL	4,128,183	1.75%
zip	3,934,987	1.67%
dex	3,015,650	1.28%
gzip	2,926,739	1.24%
lnk	2,718,635	1.15%
elf	942,148	0.40%
rar	516,514	0.22%
jar	448,324	0.19%
doc	429,794	0.18%
xls	428,057	0.18%
macho	409,399	0.17%
php	352,143	0.15%
xml	335,962	0.14%
powershell	321,178	0.14%
Other	1,233,754	0.52 %
ALL	235,745,107	100.0%

Table 4.4: Top 20 filetypes for all samples in VT File Feed.

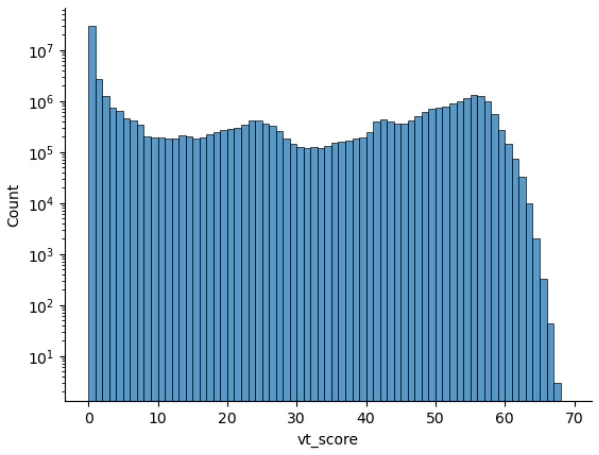

Figure 4.7: Number of detections distribution for all reports since 2021/11/19.

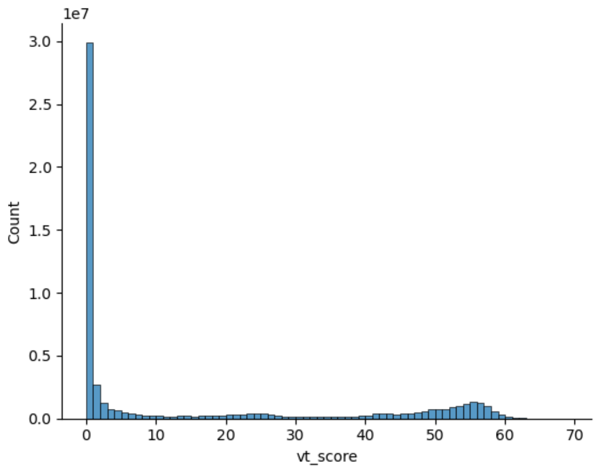

Figure 4.8: Number of detections distribution for all reports since 2021/11/19.

Also, we measure the number of detections on the first report from a sample since 2021/11/19, Figure 4.9 shows the complementary Cumulative Distribution Function, which it represents the fraction of malicious samples depending where we set the detection threshold, 53% of the samples have zero antivirus detections during his first scan, that means, if we set the malicious threshold at one detection, 47% of the samples will be malicious, if we set the malicious threshold at four detections, 41% of samples will be considered malicious.

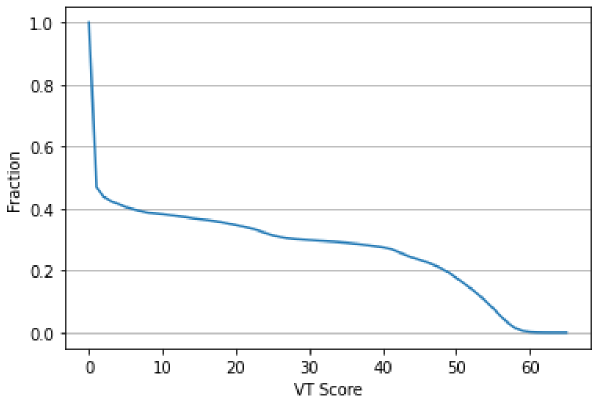

Figure 4.9: Reverse ECDF for the first report of each new samples since 2021/11/19.

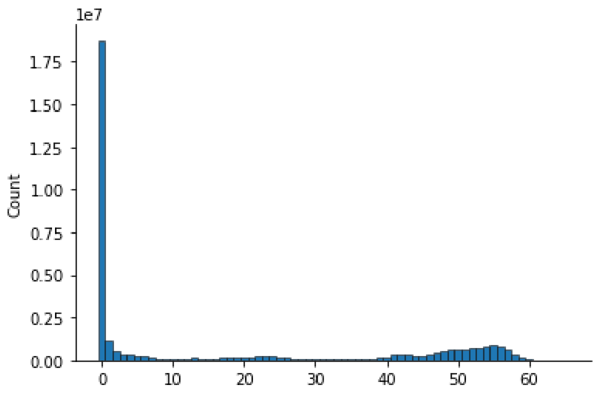

Figure 4.10: Distribution of the number of detections for the first report since 2021/11/19.

If we collect the antivirus detections for all reports (not only the first one), and we have nearly double the volume size.

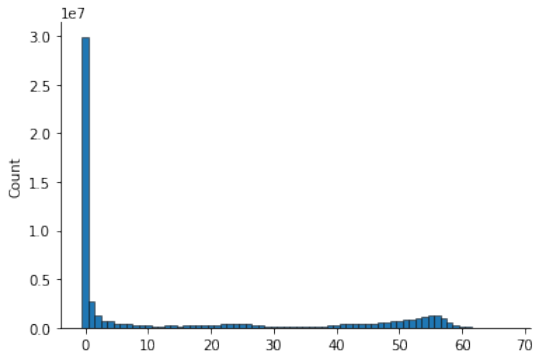

Figure 4.11: VT detections for all reports since 2021/11/19.

If we focus in the whole Vt File Feed, the distribution for the antivirus detections for all reports demonstrates that the number of one detections, 28.7M, is similar than zero detections, 30M, (95.7%).

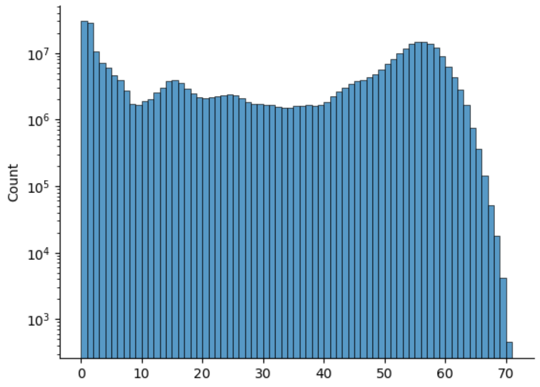

Figure 4.12: All detections for all reports, y-axis in log-scale

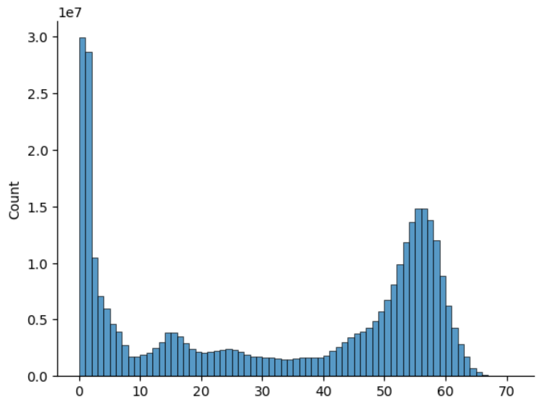

Figure 4.13: All detections for all reports

4.4. Labeling

We get the last report of each sample because we assume it will have the most updated information, From all the Vt File Feed samples (235.7M), 151.7M (64.4%) of those samples have a malware family labeled, being a total of 74.4K malware families. However, 56% family clusters are singletons (i.e., it have only one sample), 19% have at least 10 samples, 7% have at least 100 samples, and just 2% have at least 1,000 samples. That means that the feed is not dominated by a large polymorphic family malware, so the feed is quite diverse.

By filetype, the number of malware families with at least 100 samples (4.9K) is led by *peexe* with 3.8K (77.6%) families, largely followed by *apk* with 447 (9.1%) families. Other filetypes, such as *html* (129, 2.6%), *javascript* (116, 2.4%) have a less presence in the dataset.

We can build datasets for *peexe* and *apk* due to his large size, however we can not create and datasets for other filetypes (e.g., *elf*, *macho*) because is by far smaller and we do not have enough information.

If we analyze the labeled ratio, the first report for a sample label a total of 62.3%, slightly less than with the last scanned report, 64.3%. Notice that the detection ratio increases over time, Malicia dataset from 2012 [34] label a 97.8% of the samples, and AMD dataset from 2016 [7] label 98.9%.

So, the 62% corresponds to a variants of known families well-detected, in the side of an analyst, he can focus on the other 38% unlabeled samples.

Tables 4.5-4.6 shows the Top 10 family malware splitted by filetype. For *peexe*, the top 10 families are dominated by 4 worms and 3 virus,due to their highly polymorphism, for *apk*, there are 10 PUP and 8 of them are adware, for *linux* malware families, are dominated by backdoors including *mirai* derivatives, for *macOS*, seven families are PUP, which five of them are adware, for Word and Excel macros, the top 10 families are loaders.

4.5. Fully UnDetected Malware

The Fully UnDetected Malware (FUD) are samples that at the beginning was supposedly label as a benign sample, but later it was relabeled to malicious (i.e., it reaches a threshold of number of antivirus detections).

The manner we identify FUD samples is if: (1) is new (i.e., the VT First seen is in our observation range), (2) at the beginning (first report) has no detections by any antivirus, and (3) the last report is considered malicious (i.e., has at least four detections).

As the last month we collect samples with zero detections, we also collect samples that his (1) VT First Seen date falls in our observation range, and (2) his first scan date is greater than the VT first seen. The exception belongs to the samples that was seen during our gaps, which a delayed scan date respect the VT first seen does not mean zero detections on the first scan.

With the previous filters we explain, I extract 637K FUD samples, from those, 37K flipped to four detections in less than five minutes, however, in less than 15 minutes just 38K samples flipped. That means that in five minutes the detection score of a sample stabilizes.

Excluding FUD samples that flipped in less than five minutes, 600K samples (0.3% of all new samples), the median of samples that flipped is 7 days (mean 23.8 days), and the 12% of FUD samples flipped in one day or less.

The filetype distribution of FUD samples is 60% *peexe*, following by 11% *pdf*, and 8% *javascript*. The percentage of *pdf* samples in all Vt File Feed is 4.8%, and the number of FUD samples double the volume size, this is because is harder to detect a malicious pdf.

If we attend to the last report, AVCLASS2 detects a 62.5% family malware of the samples, some families have larger fractions of originally FUD samples, and thus are harder to detect, Table 4.7 shows the top 10 families sorted by ratio of originally FUD samples over all family samples with at least 10K samples.

			FUD	
Family	**Class**	**Filetype**	**Samp.**	**Ratio**
FAM:pcacceleratepro	pup	peexe	1,749	9.5%
FAM:sagent	down.	macro	2,141	9.3%
FAM:dstudio	down.	peexe	1,255	6.2%
FAM:pasnaino	down.	peexe	613	5.9%
FAM:opensupdater	pup	peexe	2,051	4.8%
FAM:mobtes	down.	apk	967	4.6%
FAM:hesv	pup	peexe	849	4.4%
FAM:asacub	infosteal	apk	833	4.1%
UNK:agentino	down.	peexe	649	4.0%
FAM:fakecop	pup	apk	672	3.6%

Table 4.7: Top 10 families (with over 10K samples) sorted by ratio of originally FUD samples.

4.6. Certificates

For self-signed certificates and fraudulent or malformed non self-signed certificate you can edit the validity from date, that is the reason why anyone can not know with accuracy the creation of a certificate, we have defined a simple step to guess the estimated creation of a certificate, we have concluded that the best approximation is to extract the minimum date between VT first seen and the *leaf certification valid from* of a report.

We can see in Figure 4.14 the number of certificates that appears in the VT File Feed everyday, the empty gaps belongs the days when the VT File Feed collection tool was not working. The longest place took between January 11th 2021 and February 7th. The huge certificates increase belongs when collecting benign samples since 2021/11/19. we can count in the same day same certificates if has been uploaded more than once in the same day. We can see that in the observed range, from 2020/12/21 until 2021/12/21, the daily mean of certificates is 20,767, the median 14,927, and the standard deviation 15,800.

We can also estimate the new certificates observed in the VT File Feed in Figure 4.15, we define a certificate is new if the estimated creation date falls within the monitored period, and is the first time it appears in the feed (scan date is the same as the VT First seen). The mean of the new certificates observed is 1,379, the median 698, and the standard deviation 1,510.

We can see a huge hike in the number of certificates collected in August, this could be because in summer it is possible that more cyberattacks may be detected. We can see that 1.4K daily unique certificates are news, we can also conclude that of the 1.7M of thumbprints that appear in the VT File Feed, 1.5M (88.2%) belongs to apk files, according to Table 5.1.

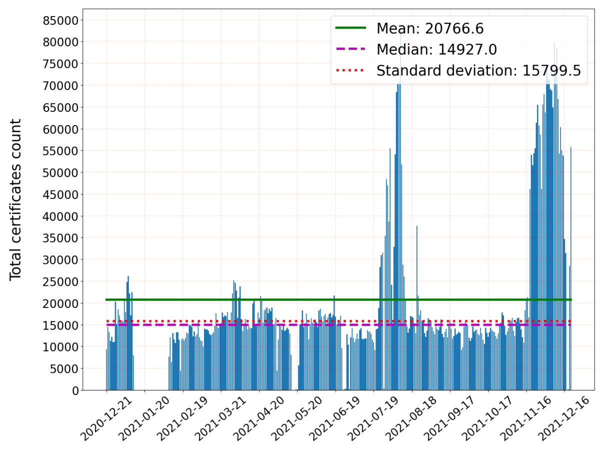

Figure 4.14: Total certificates observed in Feed.

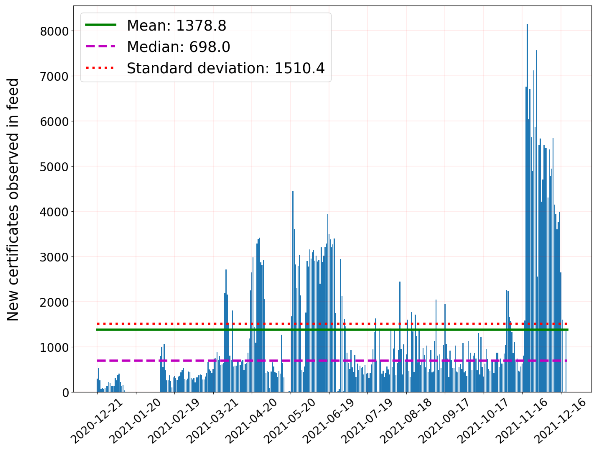

Figure 4.15: New certificates observed in Feed.

4.7. Telemetry

For this Section we have intersected the Vt File Feed with the telemetry feed for an antivirus vendor, and we have filtered 3.8M samples with at least one detection and 2.2M with at least four detections. Previous works demonstrated that public and commercial antivirus tools have small overlap [35, 36]. In Table 4.8 you can see the top 10 family malware distribution for the telemetry, all of those are PUP, in contrast of Vt File Feed, that there were viruses and worms. VT malware distribution are largely biased for highly polymorphic malware families, thus does not capture the ecosystem in real devices. Is interesting to notice that the Vt File Feed is quite distinct from the real files based on the telemetry, for works that involves to study malware ecosystem in VirusTotal, that does not resemble reality. The top 1 family in the telemetry is FAM:winactivator, a cracker to avoid paying Windows license.

There are 11.9M malicious samples not detected by Vt File Feed but detected as malicious by the security vendor, we try to request to VT those samples but we could not for by scalability problems, however, we randomly chose 1M of those samples and request it to VirusTotal. Just 10.9% are known by VT, so, 89.1% samples have never been submitted and shared to VirusTotal. This means the antivirus vendors just share a little bit of their reports to VirusTotal. The reason belongs to another department and we will not cover it.

Family	**Class**	**Devices**	**Samples**
FAM:winactivator	pup	2.0M	10,871
FAM:tool:utorrent	pup	1.6M	1,366
FAM:installcore	pup	1.5M	46,758
FAM:webcompanion	pup	1.4M	2,569
FAM:dotsetupio	pup	1.1M	198
FAM:iobit	pup	898K	4,321
FAM:opensupdater	pup	692K	14,918
FAM:opencandy	pup	579K	9,346
UNK:offercore	pup	555K	363
FAM:driverreviver	pup	545K	615

Table 4.8: Top 10 families for feed samples in the telemetry.

We also measured the detection delay time between VT and the antivirus vendor, and the median delay for VT to find a sample in contrast to the antivirus vendor is 4.4 hours later. The mean delay is 21 days because 12% of the VT first seen appear at least 3 months after it appears in the antivirus vendor, in contrast to the 3% of samples that VT find 3 months earlier than the telemetry.

4.8. Lifetime

Figures 4.16-4.17 shows the lifetime samples in days, the lifetime is the difference between the Virus-Total first seen date and the scan date. We can see that VT registers new samples relatively soon, however, nearly more than 10% is scanned by VT over 1,000 days (2.7 years)

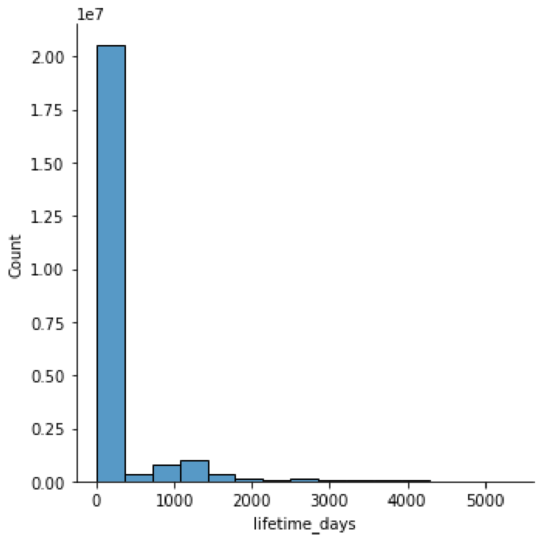

Figure 4.16: Distribution of samples VT lifetime.

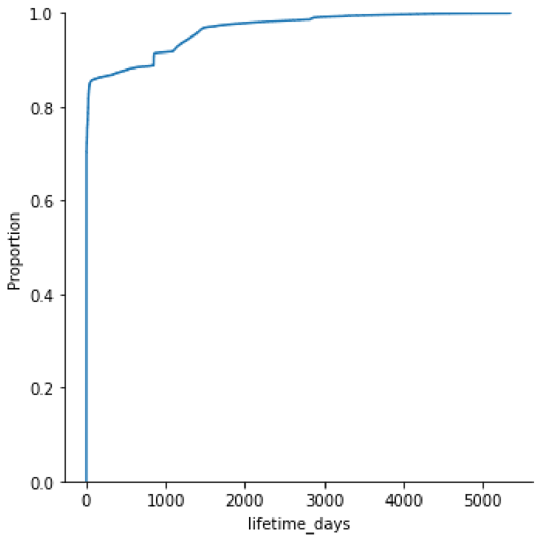

Figure 4.17: Empirical Cumulative Distribution Function of lifetime.

Filetype	Family	Class	Samples
peexe	FAM:berbew	backdoor	19,371,273
	FAM:dinwod	downloader	9,398,314
	FAM:virlock	virus	7,921,534
	FAM:pajetbin	worm	7,164,373
	FAM:sivis	virus	6,222,693
	FAM:lamer	virus	4,074,441
	FAM:salgorea	downloader	3,737,865
	FAM:vobfus	worm	3,415,996
	FAM:drolnux	worm	2,858,975
	FAM:griptolo	worm	2,407,104
apk	FAM:smsreg	pup	616,406
	FAM:ewind	pup:adware	430,531
	FAM:hiddad	pup:adware	219,577
	FAM:fakeadblocker	pup:adware	82,715
	FAM:adlibrary:airpush	pup:adware	80,704
	FAM:adlibrary:revmob	pup:adware	78,495
	FAM:dowgin	pup:adware	68,522
	FAM:dnotua	pup	65,330
	FAM:kuguo	pup:adware	63,262
	FAM:mobidash	pup:adware	40,016
elf	FAM:xorddos	ddos	287,631
	FAM:mirai	backoor	163,525
	FAM:mirai:gafgyt	backoor	59,348
	FAM:tsunami	backoor	3,381
	FAM:mirai:hajime	downloader	2,499
	FAM:mirai:mozi	backdoor	1,996
	FAM:setag	backdoor	1,454
	FAM:dofloo	backdoor	890
	FAM:fakecop	pup	805
	FAM:ladvix	virus	580
macho	FAM:flashback	downloader	33,087
	FAM:mackontrol	backdoor	15,459
	FAM:mackeeper	pup	15,017
	FAM:evilquest	ransomware	7,070
	FAM:cimpli	pup:adware	5,444
	FAM:gt32supportgeeks	pup	3,453
	FAM:genieo	pup:adware	3,339
	FAM:bundlore	pup:adware	3,142
	FAM:installcore	pup:adware	1,543
	UNK:fplayer	pup:adware	905
doc	FAM:emotet	infosteal	24,643
	FAM:valyria	downloader	10,182
	FAM:thus	virus	4,917
	FAM:sagent	downloader	4,717
	FAM:donoff	downloader	2,437
	FAM:sload	downloader	1,856
	FAM:marker	virus	1,326
	FAM:logan	downloader	1,135
	FAM:sdrop	downloader	906
	FAM:alien	downloader	846
xls	UNK:sneaky	downloader	23,521
	FAM:qbot	backdoor	22,416
	FAM:squirrelwaffle	downloader	18,230
	FAM:zloader	downloader	12,371
	FAM:sload	downloader	9,067
	FAM:sagent	downloader	8,581
	FAM:valyria	downloader	6,074
	UNK:encdoc	downloader	5,703
	FAM:icedid	downloader	4,237
	FAM:laroux	virus	2,983

Table 4.5: Top 10 families for executable and macro filetypes.

Filetype	**Family**	**Class**	**Samples**
javascript	FAM:faceliker	clicker	2,288,894
	FAM:facelike	pup	952,180
	FAM:coinhive	miner	766,087
	FAM:cryxos	-	744,894
	FAM:smsreg	pup	415,669
	UNK:gnaeus	-	400,570
	FAM:fakejquery	downloader	330,792
	UNK:hidelink	-	210,306
	UNK:expkit	-	96,433
	UNK:agentwdcr	-	87,101
html	UNK:refresh	-	882,026
	FAM:cryxos	-	363,821
	FAM:faceliker	clicker	312,563
	FAM:smsreg	pup	201,253
	UNK:redir	downloader	200,926
	FAM:coinhive	miner	152,968
	UNK:generickdz	pup	121,975
	UNK:pushnotif	-	120,085
	FAM:ramnit	virus	80,044
	UNK:fklr	rogueware	79,353
pdf	UNK:fakeauthent	phishing	194,963
	UNK:minerva	phishing	15,527
	FAM:pdfka	exploit	13,618
	UNK:pidief	-	6,319
	FAM:alien	downloader	6,137
	UNK:gorilla	phishing	4,749
	UNK:talu	phishing	2,379
	UNK:gerphish	phishing	1,558
	UNK:urlmal	phishing	1,469
	FAM:rozena	pup	839

Table 4.6: Other Top 10 families.

Chapter 5

Clustering

I think computer viruses should count as life.

Stephen Hawking

This chapter explains the clustering. The clustering features required for grouping the samples is described in Section 5.1, the batch clustering is shown in Section 5.2, the cluster summaries are presented in Section 5.3, and finally, the incremental clustering is described in Section 5.4.

5.1. Clustering Features

The features given by VirusTotal limits the VT File Feed clustering, from the existing features we examine ten that identify similar samples, mostly fuzzy hashes, shown in Table 5.1, we can also see the summarized statistics for the huge amount of reports we have collected during one year. The number of samples that does not have a non-NULL feature ID (i.e., unique features), appear in Table 5.1, we can also know the number of clusters if we group by a single feature. *avc2_family* feature belongs to samples with family/unknown tag, (i.e., we exclude samples with generic, behaviour, or file tag). The samples are non-exclusive, reason why all features are larger than peexe + apk + other.

tlsh feature has a 99.9% of non-NULL feature ID, following by vhash that does not extract value for samples without filetype or image filetype (93.4%). Three specific feature belongs to *peexe* samples, *authentihash*, with 99.9% of availability, *imphash*, (96.3%) and *Rich PE Header* (41.9%) Also, the *package name* is a specific feature for *apk* samples, being available in 92.6% of the samples.

		Values			
Feature	**Samples**	**All**	**peexe**	**apk**	**other**
tlsh	235.6M	204.3M	134.0M	7.6M	62.7M
vhash	220.1M	17.4M	6.4M	1.3M	9.7M
authentihash	155.3M	152.9M	152.9M	0	0
avclass_family	151.7M	74.4K	62.4K	2.8K	22.4K
imphash	149.7M	8.3M	8.3M	0	0
richpe_hash	65.2M	2.1M	2.1M	0	0
dhash	60.5M	5.3M	1.2M	2.0M	2.0M
icon_md5	60.5M	15.6M	1.9M	2.2M	11.5M
thumbprint	13.3M	1.7M	168.7K	1.5M	10.6K
package_name	7.4M	2.8M	0	2.8M	0

Table 5.1: Unique feature values in dataset.

5.2. Batch Clustering

Batch clustering takes as input collected samples and clusterizes it once, we can not give *early alerts* because we can not extract delthas or samples variations, nor can we run the clustering every day with the previous created clusters; in fact, if we run the cluster every day with the whole data, each day will take more time than the previous one to clusterize.

5.2.1. Clustering Approaches

We are going to evaluate three clustering approaches that does not need to define the number of clusters before running it. Later, we are going to evaluate the clustering results (precision, recall, and F1) with the scalability issues.

5.2.1.1. Feature Value Grouping

The first approach, Feature Value Grouping (FVG) consists of grouping all samples by an equality comparison on the single values feature (samples that have the same imphash belongs to the same cluster).

1. **Format VT File Feed.** Given a file with Feed Reports, the first step includes many tasks, (1) first we can filter by filetype, in this case, as we want all samples, we do not filter by any specific filetype, we also (2) can filter by scan date and (3) reports with null feature, this value can appear because (3.1) Samples has not that feature, Some features only appear on specific filetypes, like *imphash* that only appear in peexe samples, or *package_name* feature, that only appear in apk samples, and (3.2) VT infrastructure have had a problem and could not extract the feature, if this occurs, it usually appears on the first scan, the following reanalyzed files VT extracts the features correctly. The filtering parts is optional and can be avoided. Then, we (4) add a *filetype* column to our VT File Feed, and finally, (5) we escape special characters (e.g., \n, \r, \t) in specific strings fields (e.g., *leaf_cert_issuer*, *leaf_cert_subject*, *vt_meaningful_name*, *exiftool_original_fname*, and *sig_verification_res*) that corrupts the file.

2. **Sort by feature.** At this point, we sort all VT File Feed reports by the feature (e.g., vhash, thumbprint) we want to cluster, sample hash (e.g., SHA256), and scan date, so, a change in the feature means a new cluster. We will use the latest report for each sample (we assume it will have the more up-to-date and accurate information), also, it accepts as parameter a date range to filter reports we do not need. The reason to sort with the shell command is the efficiency, it makes a merge sort and controls the memory used to maximize the performance [37].

3. **Generate cluster summaries.** We iterate the whole sorted file generated previously, because it is sorted by the feature. Each feature belongs to a cluster, and for each cluster, we calculate the cluster fields. We process the VT File Feed report by report, attending to his feature ID. At the moment a report change his feature ID, we create the cluster with his features. With this manner the VT File Feed can scale in a simple but effective way, regardless the feed size. The generated cluster summaries will be useful to compare what antivirus engine detect more PUP or malware, interconnect clusters by distinct features, search supposedly benign samples in malicious clusters and so on. Our cluster summaries can be exported in different formats, as JSON, TSV, and MongoDB. With MongoDB we save the clusters in a *features* and *samples* collection.

We can cluster the VT File Feed in two ways:

- **Batch mode (absolute).** When the VT File Feed data collection has finished, we begin to cluster, so our input is the whole VT File Feed. We do not cluster new data given precomputed clusters. The advantages are that we process the reports and we do not need to rerun the cluster, but the disadvantage is that we can not receive *early alerts*, so we can not extract variations between different weeks or days.

- **Incremental mode (relative).** Our goal is to create an incremental cluster to detect appearing spikes, the problem with batch mode is that it loses granularity, so with this clustering type we can extract deltas and know which clusters are hot/on arise, thus, it detect *early alerts*. Our input is the VT File Feed splitted by days and we cluster by FVG (i.e., single feature). Notice that we can make the incremental clustering only if we have our data saved in a MongoDB.

We have created a class in the generated cluster summaries step to encapsulate the cluster state, allowing to collect the memory avoiding to clean by hand each attribute separately, it also increases the reusability and structure the code to make it easier to separate between what needs to be done.

Notice that we are extracting statistics per samples, instead of per reports, with this we have more accurate results, and we make the opposite as Spotlight [22], we firstly clusterize the reports and later we check if they are labeled.

In Table 5.2 we can see a part of an example of the cluster summary grouped by the leaf certificate thumbprint feature.

thumbprint	**reports**	**samples**	...
61ed377e85d386a8dfee6b864bd85b0bfaa5af81	932,381	482,941	...
a719dac021e9e077fb75bcbad7e06dbc37cee58c	332,409	241,346	...
27196e386b875e76adf700e7ea84e4c6eee33dfa	254,815	86,725	...
f36580fc751d59c0f59b8a0c869ca58bd0eae97e	181,184	105,821	...
...	...	...	...

Table 5.2: Summary of VT File Feed clustered by thumbprints.

5.2.1.2. Hierarchical Clustering

The next approach is to cluster by Hierarchical clustering, there are multiples Hierarchical approaches but for this we select Hierarchical Density-Based Spatial Clustering of Applications with Noise (HDBSCAN). and HDBSCAN is a DBSCAN Hierarchical extension that give best stability of the clusters by a bottom-up approach.

With this approach we can cluster combining a single or multiple features, we have to create a weighted average matrix distance, with the distance function for the features we want to cluster the boolean distance for cryptographic hashes and *vhash*, and the Hamming distance for *tlsh*, for later cluster by HDBSCAN.

This clustering process is summarized hereafter:

1. **Matrix distance creation for each feature.**

   Every feature has its own comparison function between 2 samples, as described in Section 2.1. However, the range results differs between different features comparison functions, it can be from 0 (mismatch) to 100 (perfect match) as *ssdeep*, or 0 (perfect match) to 100 (mismatch), or uses a hamming distance function with no maximum limit, as *tlsh*. Remind that *vhash* (Virus Total propietary hash) is an equality feature, so the result is not a range between 0 and 1, only outputs 0 (equal) or 1 (different). We have to normalize all results to have the same range, for this we have selected that the output will be 0 if the samples are a perfect match or 1 if the samples are completely different. The normalize methods for the different features are:

- *ssdeep.* We normalize a *ssdeep* score x to $y = 1 - x/100$.
- *tlsh.* We normalize *tlsh* scores to [0-100] range adopting [38] approach [1] and then we limited to [0-1] range.
- *dhash.* Uses as comparison function a hamming distance function, we normalize it dividing it by the maximum length of two hashes in bits.
- *vhash.* Uses a string comparison, so the 0 is a perfect match and 1 is a different sample.

We create a distance matrix for each feature using all VT File Feed samples. Notice that each distance matrix (grouped by different feature) has a different size, depending the non-NULL features for all samples, Table 5.3 shows a *ssdeep* distance matrix example. For instance, to generate the *ssdeep* distance matrix we apply the comparison function for each sample with an *ssdeep* non-NULL value to the other samples, we save the result in a matrix distance (same values for the rows as for the columns).

	00313e588a5161220	**000ee4a7a0a5d2ba4**	**...**	**0012a7e859df562c2**
00313e588a5161220	0.0	1.0	...	0.16
000ee4a7a0a5d2ba4	1.0	0.0	...	1.0
...	...	...	...	...
0012a7e859df562c2	0.16	1.0	...	0.0

Table 5.3: *ssdeep* distance matrix example.

2. **Combine distance matrices into one.**

   Now we combine multiple distance matrices (for each feature we want to cluster) into one, we calculate the mean of the distance for the existing features, we sum the distance between each pair of samples and we divide it by the number of features. When we can not calculate the distance between two features (e.g., a samples does not have feature or VT could not extract that feature) we fix this by two approaches:

   - We can assign a constant for distance of the non existing feature. We have assigned 0.5 and 0.8 constants (used in prior works [14]).
   - Change the weight for each feature, only calculate the mean for the existing features.

   Once we have the total average distance matrix, we cluster by HDBSCAN.

The main problem for this approach is the quadratic complexity ($\mathcal{O}(n^2)$), that restrict us to cluster at large-scale.

5.2.1.3. HAC-T

The last approach is a threshold-based Hierarchical Agglomerative Clustering (HAC-T), it can cluster samples by *tlsh* feature reducing the complexity from $\mathcal{O}(n^2)$ to $\mathcal{O}(n \log n)$; it can cluster 10 million samples in 2.2 h and it produces highly quality clusters (0.97-0-98).

[1] $y = max\{0,\ (300 - x)/3\}$

Dataset	Plat.	Samples	Fam.	Collection
Malsign [41]	Win	142,513	127	06/2012 - 02/2015
AMD [7]	And	24,551	71	11/2010 - 03/2016
Malicia [34]	Win	9,908	52	03/2012 - 02/2013
Drebin [42]	And	5,560	179	08/2010 - 10/2012

Table 5.4: Ground truth datasets used to evaluate clustering.

We modify the open sourced version [2] to print the cluster id after the sample hash given the VT File Feed with just the samples hashes and the *tlsh* features, for later evaluate the clustering over a Ground Truth. We cluster HAC-T and HAC-T-opt (optimized version with some performance cost) with the 30 recommended *CDist* value [39].

5.2.2. Accuracy Evaluation

We measure the accuracy of the generated clustering of the above approaches using ground truths (GT). A ground truth is a file with information that is know to be real, provided by an empirical evidence of direct observation. Our ground truths, links the cryptographic sample hash (e.g., *md5*, *sha1*, *sha256*) with his malware family name.

For this examples we have selected four GT (both public and private):

- **Malicia** [5]. Dataset with 10K Windows malware samples collected by drive-by downloads, it has dominant families so it is a quite unbalanced clustering.
- **Malsign** [9]. This dataset contains 142K Windows samples, it has more installers, and therefore PUP (Potentially Unwanted Program), so precision is worse.
- **Drebin** [6]. Android dataset with 7K samples and 179 different malware families.
- **AMD** [40]. Android dataset that classifies 135 varieties within 71 families in 24K samples.

Table 5.4 summarizes the platform, number of samples, families, and period collection of the dataset explained above. For Malicia we requested VT re-scan of all samples to include *tlsh* hash value, for the other datasets we collected the latest submissions reports. For FVG, on AMD and Drebin we exclude Windows specific features (*imphash*, *authentihash*, *richpe_hash*).

I evaluate the created clustering extracting the precision, recall and F1-measure, comparing it with the four ground truths, the results are in the following Tables. We provide a list of generic certificates (certificates commonly used in malware or PUP, so the malware family name of the certificate differs a lot) for Android datasets to exclude them for a better evaluation.

As the following Tables shown, the best feature for clustering if we only cluster by a single feature is avc2_family, with F1-score from 84.8% to 95.7%, with a notable difference with other features, that can reach a 1.1% F1-score (for *authentihash* feature). The VT File Feed grouped by multiple features extract a low F1 results, going from 9.8% to 34.0%, but with *avc2_family* also increases the F1-score in all clusterings. This seems a bit trick because *avc2_family* is itself a kind of clustering clustered by AVCLASS, so for extracting *avc2_family* feature we need to run AVCLASS before over our file feed. As shown in the below Tables, after *avc2_family* feature, *thumbprint* and *vhash* features achieve better results.

[2] https://github.com/trendmicro/tlsh/tree/master/tlshCluster

Notice that Malicia is from 2013, Drebin 2014, Malsign is from 2015, and AMD from 2016. AVCLASS2 labels may update over time, therefore, the accuracy may improve on newer samples. For the evaluation, notice that singleton clusters does not affect to precision, however, it affects to the recall.

By examining the results, with FVG, HAC-T, and HDBSCAN we can conclude that none of the approaches tested can help us with our triage tool due to scalability and F1 reasons.

AMD

Table 5.5 shows FVG and HAC-T results, despite having a good precision it has a low recall. The highest precision means that in a same cluster (e.g., *vhash*), different labels (e.g., malware family in the GT) do not mix with each other, The more recall means that the same malware family in the GT does not split in many clusters (e.g., *vhash*).

Feature	**Algor.**	**Clust.**	**Prec.**	**Recall**	**F1**
avc2_family	fvg	407	95.4%	89.0%	92.1%
cert_thumb. (gen. certs)	fvg	8,575	92.5%	18.6%	31.0%
cert_thumb.	fvg	8,567	90.1%	15.9%	27.0%
icon_dhash	fvg	17,663	98.5%	9.7%	17.7%
icon_hash	fvg	17,812	98.7%	9.7%	17.6%
pkg_name	fvg	17,430	99.6%	9.2%	16.8%
vhash	fvg	7,478	94.5%	20.6%	33.8%
tlsh	hact-opt	17,515	98.3%	7.6%	14.1%
tlsh	hact	17,884	98.4%	6.9%	12.9%

Table 5.5: AMD GT Clustering.

As HDBSCAN involves a new parameter (default distance), it has been separated in Table 5.6. Default distance is a constant value that is assigned when it can not calculate a distance between two samples while creating the total distance matrix for the clustering.

The features used has been selected because: (1) *icon MD5* has appeared on most of the *apk* samples, (2) *package name* is a specific feature for android apps, (3) *vhash* has been proven in the previous Table that extract better results, (4) *thumbprint* also following the previous Table it cluster much better than most of the other features, (5) *avc2_family* is one of the strongest features for cluster, and (6) *tlsh* outperforms ssdeep, sdhash and other fuzzy hashes features.

Feature	**Def. dist.**	**Clusters**	**Prec.**	**Recall**	**F1**
icon_md5 + pkg_name + vhash + thumbprint (non gen. cert.)	0.5	3,054	76.8%	14.0%	23.6%
	0.8	3,624	78.9%	14.1%	23.9%
first + avc2_fam	0.5	3,705	87.9%	20.8%	33.6%
	0.8	3,705	87.9%	20.8%	33.6%
first + tlsh	0.5	4,648	81.4%	10.4%	18.5%
	0.8	4,716	81.8%	10.7%	19.0%
first + avc2_fam + tlsh	0.5	4,430	88.3%	17.2%	28.8%
	0.8	4,430	88.3%	17.2%	28.8%

Table 5.6: AMD GT for HDBSCAN Clustering.

This results show a low accuracy in the performed clusterings, we will try in another datasets to know if the problem is the dataset or the features/clustering approaches.

Drebin

Table 5.7 shows the clustering results for the features selected previously, it has quite better results due to this dataset has more balanced samples, there is no predominant family.

Feature	**Algor.**	**Clust.**	**Prec.**	**Recall**	**F1**
avc2_family	fvg	199	95.0%	95.3%	95.1%
cert_thumb. (gen. certs)	fvg	1,198	93.4%	48.7%	64.0%
cert_thumb.	fvg	1,203	91.1%	44.0%	59.3%
icon_dhash	fvg	3,037	96.8%	19.0%	31.8%
icon_hash	fvg	3,066	97.0%	17.9%	30.2%
pkg_name	fvg	2,719	99.2%	19.6%	32.7%
vhash	fvg	1,524	92.8%	25.0%	39.4%
tlsh	hact-opt	4,138	99.7%	11.0%	19.8%
tlsh	hact	4,209	99.7%	8.5%	15.7%

Table 5.7: Drebin GT Clustering (7,534 samples).

Table 5.8 shows the HDBSCAN results, and as Table 5.7, it has better results than AMD dataset.

Feature	**Def. dist.**	**Clusters**	**Singletons Samples**	**Prec.**	**Recall**	**F1**
icon_md5 + pkg_name + vhash + thumbprint (non gen. cert.)	0.5	863	10.5%	77.5%	29.4%	42.6%
	0.8	794	9.3%	78.5%	30.4%	43.8%
first + avc2_fam	0.5	830	10.8%	87.3%	43.1%	57.7%
	0.8	830	10.8%	87.3%	43.1%	57.7%
first + tlsh	0.5	876	10.1%	75.9%	21.9%	34.0%
	0.8	876	10.1%	75.9%	21.9%	34.0%
first + avc2_fam + tlsh	0.5	1,240	16.4%	90.1%	27.7%	42.4%
	0.8	1,240	16.4%	90.1%	27.7%	42.4%

Table 5.8: Drebin GT for HDBSCAN Clustering.

Concluding with these *apk* datasets, both of them has a low F1 and also it takes 20 hour average to cluster AMD for HDBSCAN, so it is not worth clustering our VT File Feed by HAC-T and HDBSCAN due to scalability reasons in terms of time and accuracy.

Malicia

For *windows* datasets such as Malicia and Malsign, we include specific *windows* features such as *authentihash imphash*, and *richpe_hash*, as shown in Table 5.9.

Feature	Algor.	Clust.	Prec.	Recall	F1
authentihash	fvg	9,909	100%	0.5%	1.1%
avc2_family	fvg	284	97.0%	75.4%	84.8%
cert_thumb. (gen. certs)	fvg	9,410	100%	1.8%	3.5%
icon_dhash	fvg	9,407	99.9%	2.2%	4.3%
icon_hash	fvg	9,766	99.9%	1.0%	1.9%
imphash	fvg	1,843	99.7%	5.7%	10.7%
richpe_hash	fvg	9,899	100%	0.6%	1.2%
vhash	fvg	900	98.8%	12.8%	22.7%
tlsh	hact-opt	3,772	99.9%	6.2%	11.8%
tlsh	hact	3,899	99.9%	3.8%	7.3%

Table 5.9: Clustering accuracy on Malicia dataset. Samples is the percentage of samples with a non-NULL feature value.

Table 5.10 shows that the best feature to cluster is *avc2_family*, with a 83.8% F1 score when it is included in our clustering, but it keeps being less than if it is FVG clustered alone.

Feature	Def. dist.	Clusters	Singletons	Prec.	Recall	F1
icon_md5 + vhash + thumbprint (non gen.cert.)	0.5	435	1.6%	93.2%	13.8%	24.0%
	0.8	448	1.7%	93.4%	13.8%	24.0%
first + avc2_fam	0.5	103	0.1%	95.6%	74.5%	83.8%
	0.8	103	0.1%	95.6%	74.5%	83.8%
first + tlsh	0.5	455	1.8%	93.3%	13.7%	23.9%
	0.8	485	2.1%	93.7%	13.7%	23.8%
first + avc2_fam + tlsh	0.5	114	0.8%	95.7%	74.6%	83.8%
	0.8	114	0.8%	95.7%	74.6%	83.8%

Table 5.10: Malicia GT for HDBSCAN Clustering.

As Malicia could be the most similar dataset, I clustered trying additional features shown in Table 5.11 such as *authentihash*, *rich PE header hash*, *imphash*, *dhash*, and *ssdeep*. The results show a F1 decrease using Windows native features with respect to Table 5.10.

Feature	Def. dist.	Clusters	Singletons	Prec.	Recall	F1
authentihash + leaf_cert + icon_md5 + imphash + rich_pe_hash + dhash + vhash + tlsh + ssdeep	0.5	809	4.1%	95.6%	6.4%	12.0%
	0.8	809	4.1%	95.6%	6.4%	12.0%
authentihash + leaf_cert + dhash + rich_pe_hash + ssdeep + tlsh + vhash	0.5	1,179	7.9%	95.3%	6.3%	11.8%
	0.8	1,029	6.5%	94.4%	7.6%	14.0%
dhash + vhash + tlsh + ssdeep	0.5	1,267	7.9%	95.3%	5.1%	9.8%
	0.8	1,189	7.0%	95.6%	5.8%	10.9%

Table 5.11: Malicia GT for HDBSCAN Clustering with additional features.

Malsign

Table 5.12 shows the accuracy dataset, *authentihash* feature groups better than in Malicia dataset because it is a more balanced dataset.

Feature	Algor.	Clust.	Prec.	Recall	F1
authentihash	fvg	115,409	100.0%	5.5%	10.4%
avc2_family	fvg	142,509	96.8%	94.5%	95.7%
cert_thumb. (gen.cert.)	fvg	14,560	99.7%	37.4%	54.4%
icon_dhash	fvg	135,508	99.5%	3.9%	7.5%
icon_hash	fvg	135,786	99.5%	3.6%	7.0%
imphash	fvg	23,475	95.9%	36.8%	53.2%
rich_pe_hash	fvg	140,734	99.7%	2.2%	4.4%
vhash	fvg	6,380	96.7%	36.4%	52.9%

Table 5.12: Malsign GT Clustering (142,510 samples).

Due to scalability reasons, is infeasible to run HDBSCAN over Malicia dataset with over 142K samples so we abstain to cluster. We also do not cluster with HAC-T for scalability reasons.

We can conclude with these *windows* datasets that as in *apk* datasets, it is not worth clustering our VT File Feed with these approaches, firstly for time scalability reasons, and secondly for the bad results it extracts. One of the possible approaches could be cluster by FVG with *vhash* or *thumbprint* feature, that achieves better results.

When we cluster by one single feature, the precision is pretty high (97%-100% almost all features) but the recall is pretty low. The best feature is AVCLASS2 family with 84.9% F1 score on Malicia and 92.1% on AMD, then vhash (22.7%-33.8%), leaf certificate thumbprint (27.0%) and TLSH HAC-T-opt (11.8%-14.1%).

5.2.3. Scalability

We evaluate the scalability of the three clustering approaches we have discussed, focusing on the time restrictions, because our server, an Intel(R) Xeon(R) CPU E5-2660 v3 @ 2.60GHz with 40 cores and 128 GB RAM have enough memory to be worried about. We evaluate the approaches over the day with minimum, mean, and maximum samples influx. We define a time threshold of 24 hours to know if it scales or not.

The day with less samples, 2020/12/21, we evaluate 620K samples, the HDBSCAN approach takes more than 24 hours in cluster, HAC-T takes 9.7 hours, and FVG-vhash take less than one hour (49.6 minutes sorting and filtering by that day and 2.6 minutes grouping by vhash) The day with an average number of samples, 2021/11/19, we evaluate 1.4M samples, the HAC-T approach takes 22.4 hours in clustering and FVG-vhash less than one hour (49.7 minutes sorting and 4.3 minutes grouping). The day with maximum samples, 2021/11/23, we evaluate 2.2M samples, HAC-T does not finish in less than one day, however, FVG-vhash takes as usual less than one hour (46.5 minutes sorting and 6.3 minutes grouping).

HAC-T paper mentions that they can cluster 10M samples in 2.2 hours, one possible explanation of this difference is that they use for the results a more efficient C version implementation, while we use the Python version.

5.2.4. Clustering whole Dataset

We have processed 328.3M reports, 235.7M samples, including signed as unsigned files, as we can see in Table 3.3. We tried running HAC-T on the whole dataset but stopped it after 7 days, for FVG we grouped by the best four features we achieved better results, *avc2_family*, *vhash*, *leaf_cert*, and *imphash* for peexe samples.

FVG-imphash is the feature it take less for grouping, due to only sort peexe samples, it takes 5.9 hours sorting and 2.3 hours grouping. Then FVG-leaf_cert takes 5 hours sorting and 3.5 hours grouping (groups Non-NULL leaf certificate thumbprint samples), FVG-avc2_fam. takes 3.9 hours sorting and 10.3 hours grouping, and FVG-vhash takes 4.2 hours sorting and 11.1 hours grouping.

Table 5.13 summarizes the scalability results, it also show the number of clusters, as the number of clusters excluding samples with NULL feature id. It also shows the number of singletons clusters. We can see that the median of all clusters are one, that means that more than the 50% of Non-NULL clusters are singletons. FVG-cert_thumbprint produces 224.1M clusters, of which 223.7M (99.82%) are singletons and 0.5M (0.18%) contain multiple samples. FVG-vhash produces 33.2M clusters, of which 29.2M (88%) are singletons and 4M (12%) contain multiple samples.

		Clusters		Singletons		Non-NULL clusters				Runtime (h)	
Feature	**Filetype**	**All**	**Non-NULL**	**All**	**Non-NULL**	**Max.**	**Mean**	**Med.**	**Std**	**Sort**	**Group**
cert_thumb.	All	224.1M	1.7M	223.7M	1.2M	1.1M	7.8	1.0	959.4	5.0	3.5
avc2_family	All	84.1M	74.4K	84.1M	41.5K	19.4M	2.0	1.0	100,287.4	3.9	10.3
vhash	All	33.2M	17.4M	29.2M	13.4M	3.3M	12.7	1.0	2,138.0	4.2	11.1
imphash	Peexe	14.1M	8.3M	11.6M	5.7M	6.0M	18.1	1.0	5,537.1	5.9	2.3

Table 5.13: Clustering results for FVG with different features on the whole dataset of 235.7M samples (except for imphash).

5.3. Cluster Summaries

The output of the cluster summaries created are given by the following columns listed in Table 5.14.

Type	Columns	Derived	Description
General	cluster_id	✗	Id of a feature.
	reports	✗	Number of times a file appears in the cluster.
	samples_all	✓	Number of unique reports.
	samples_vt1	✓	Number of unique reports with at least 1 VT score.
	samples_vt4	✓	Number of samples with at least 4 VT Score.
AVCLASS2	sym_detected	✗	Samples that Symantec has detected as malicious.
	sym_detected/samples_vt1	✓	Percentage of Symantec samples detected in the cluster samples.
	sym_labeled	✗	Samples that Symantec has detected as malicious and has assigned a FAM (family) or UNK (unknown) AvClass2 tag.

	sym_labeled/sym_detected	✓	Percentage of sym_labeled samples in Symantec samples detected.
	avira_detected	✗	Is the number of samples that Avira has detected as malicious.
	avira_detected/samples_vt1	✓	Percentage of Avira samples detected in the cluster samples.
	avira_labeled	✗	Is the number of samples that Avira has detected as malicious and has assigned a FAM (family) or UNK (unknown) AvClass2 tag.
	avira_labeled/avira_detected	✓	Percentage of avira_labeled samples in avira samples detected.
	samples_pup	✗	Samples considered as PUP. We check if a sample is PUP or not checking if the highest value of the CLASS tag contains 'grayware'.
	allav_tags	✓	AvClass2 tags scanned by all antivirus engines.
	sym_tags	✗	AvClass2 tags scanned by Symantec engine.
	family_sym	✓	Most commonly family name scanned by Symantec.
	avira_tags	✗	AvClass2 tags scanned by Avira engine.
	family_avira	✓	Most commonly family name scanned by Avira.
	family	✓	Most commonly family name scanned by all antivirus engine.
Lifetime	fresh_type	✓	Freshness type: It can be fresh or old.
	freshness	✓	Fresh samples fraction.
	filetype_ctr	✗	Filetype counter, filter by peexe or APK.
	filetype	✓	Highest value of filetype ctr.
	vt_first_seen	✗	Minimum VT first seen in a cluster.
	last_vt_first_seen	✗	Maximum VT first seen in a cluster.
	lifetime_days	✓	Number of days between VT_first_seen and last_vt_seen.
	first_observation	✓	First observation date in our Feed.
	last_observation	✓	Last observation date in our Feed.
	days_observed_ctr	✗	Counter of days observed.
	num_observation_days	✓	Number of days observed.
	detection_delay	✓	Number of days when samples in the cluster where not detected.
Type	samples_fresh	✓	Samples where VT first seen date is greater than our collection start date.

	cluster_type	✓	Cluster class, it can be singleton (the number samples of samples of the cluster is 1), small (the number of samples is less or equal than 5), benign (percentage of samples VT4 in samples VT1 are less than a threshold, in this 0.1), untagged (does not have FAM or UNK tag) or tagged (none of the above).
Features	vt_score_ctr	✗	VT Score counter.
	min_vt_score	✓	Minimum VT score.
	max_vt_score	✓	Maximum VT score.
	mean_vt_score	✓	Mean VT score.
	median_vt_score	✓	Median VT score.
	icon_md5_ctr	✗	Counter of the MD5 hashes of a cluster.
	num_icon_md5	✓	Number of distinct MD5 hash icons of a cluster.
	dhash_ctr	✗	Counter of de dhashes of a cluster.
	num_dhash	✓	Number of distinct dhashes icons of a cluster.
	packages_ctr	✗	Counter of name packages in android samples.
	num_packages	✓	Number of unique name packages in android samples.
	vhashes_ctr	✗	Counter of vhashes in the cluster.
	num_vhashes	✓	Number of unique vhashes.
	auth_names_ctr	✗	Counter of authenticode names.
	num_auth_names	✓	Number of unique authenticode names in peexe samples.
Certificates	certs	✗	Counter with num_certs, num_generic_certs and num_revoked_certs.
	samples_signed	✓	Samples with certificate.
	num_certs	✓	Numbers of unique certificates a cluster have.
	num_generic_certs	✓	Number of generic certificates (list of generic certicates is provided).
	num_revoked_certs	✓	Attending the verification signature, number of revoked certificates.
	cert_subjects	✗	Certificate subjects counter.
	num_cert_subjects	✓	Number of unique certificate subjects.
	min_cert_validity_start	✓	Minimum validity from date from a certificate sample.

	max_cert_validity_end	✓	Maximum validity to date from a certificate sample.
Telemetry	telem_num_samples	✗	Number of samples from the cluster that also appears in the telemetry file.
	telem_sum_devices	✗	Number of devices that have samples from the telemetry and are in the cluster.
	telem_sum_uniq_file_names	✗	Number of unique filenames for those samples appear in the interesction between VT File Feed and telemetry file.
	telem_sum_unique_dirs	✗	Number of unique directories names the samples appear on.
	telem_sum_unique_parents	✗	Number of unique parent directories names the samples appear on.
	telem_sum_unique_urls	✗	Number of unique URLs samples contact to.
	telem_first_seen_ctr	✗	Counter of the first seen samples have been seen.
	telem_first_seen	✓	First timestamp a samples from the telemetry have been seen.
	telem_last_seen_ctr	✗	Counter of the last seen samples have been seen.
	telem_last_seen	✓	Last telemetry sample seen in timestamp.

Table 5.14: Cluster columns

5.4. Incremental Clustering

We have also implemented an incremental clustering approach using FVG. It accepts previously generated clusters and new daily samples to update the new clusters. With this incremental clustering we can detect early alerts given an input of old clusters.

One of the advantages of using incremental clustering is that samples does not change the clusters, the clusters does not reorganize when a new sample appear in the clustering. The incremental clusterings add a temporal component, it accepts for example new daily, weekly or monthly VT samples and the previous generated clusters to create new one. The incremental clustering allows to alert clusters on arise extracting delthas, for instance, a cluster duplicate its size to 2,000 samples in one day. It also check if samples with zero detections inserted in a malicious cluster giving *early alerts*.

For the incremental clustering, each time we run the tool we provide a list of clusters and new samples to update the clusters saved. The generated data is collected in a MongoDB where we store 2 MongoDB collections, the (1) Cluster Summaries collection and the (2) Samples collection, given a file feed sorted by feature id, sample hash and scan date, the tool process the file report by report, for each new cluster (i.e., the cluster-id differs from the previous report) it search if there is a cluster generated in the MongoDB, if not, it creates an empty cluster and it will be updating fields until the file process a new cluster. For each report in the cluster, the column *reports* updates and and check if the *leaf cert. thumbprint* changes to generic certificate. For each sample (i.e., the hash sample differs from the previous one), we check if that sample has already been processed before (i.e., other day appeared and is saved in the Features Collection), (1) if has been processed, we check if the sample has changed the cluster, if has changed (1.1), we remove the clustering features from the old sample cluster and we add the new sample information into the new cluster. If the sample has not changed cluster (1.2), we remove the sample information from the cluster to update the cluster with the new data. (2) If has been processed, we just insert the sample in the cluster, for this we check the VT First Seen Date, if the sample has a less First Seen date, the cluster will update the *First seen Date* with the new sample value. The cluster will also update the following counters: (1) VT Score Counter, (2) Vhash Counter, (3) Imphash Counter, (4) Icon MD5 Counter, (5) Icon dhash Counter, (6) Windows Authenticode, (7) Filetype Counter, (8) Package name Counter, (9) AVCLASS Tags, (10) PUP Counter, and (11) Certificate Thumbprint Counter.

For the Telemetry intersection, we search if the sample appears in the telemetry to update the following columns: (1) First report Counter, the first date the sample appeared in the telemetry, (2) Last report Counter, last time the sample the sample appeared in the telemetry, (3) number of samples, (4) devices sum, number of devices the sample was detected, (5) Sum unique filenames, number of the filenames samples, (6) sum unique dirs, number of directories the samples is observed, (7) Sum unique parents, are the parents (hashes from other samples) responsible for downloading this sample, (8) Sum unique URLs, URLs from which has been downloaded the sample.

When we update a cluster, the info is passed to the Alert module, that can alert us when some defined rule has been completed. The new generated cluster will be stored in the cluster Collection and optionally in TSV format, and the sample will be saved in the Samples Collection.

Chapter 6

Threat Hunting

The best protection is early detection

The Clustering chapter describes how to detect interesting samples, clustering by incremental or batch mode, Section 6.1 describes the Alert module for alerting us, and Section 6.2 presents some practical examples of potentially malicious samples.

6.1. Alert

We perform threat hunting at large scale without applying aggressive filtering, the Alert module seeks for predefined and modifiable rules, it compares the old cluster with the new one with the sample added, so it must be done with the incremental clustering.

It alerts in a separated file the clusters that reach specific conditions, for instance, when a cluster find supposedly benign samples that belong to malicious clusters (most samples are malicious), with this manner we can make *early detection*.

Some applications are:

1. Alert when there are benign samples in malicious clusters (more than half of the samples have more than four detections). That means, search for true negatives.
2. Warn with the same exmaple but in reverse, that is, malicious samples in benign clusters (more than half of the samples have zero detections). That means, search for false positives.
3. Identify new clusters on arise, it compares the previous generated cluster size with the new cluster size, and if the cluster reachs any thresholds alerts.
4. Detect labels by other antivirus engines.
5. Compare what type of clusters, Symantec and Avira does not detect but have more than 4 AV detections.
6. Find connections beyond clusters, for family or by thumbprint.
7. Alert when a PUP cluster samples reach a specific threshold.

8. Switch cluster type from fresh to old. Searching the cluster_type between the previous and the new generated cluster.
9. Alert when the threshold of a specific AV engine detection samples / Samples VT1 is greater than a specific threshold.
10. Warn when a cluster of certificates changes to revoked and its certificate changes to generic.

Queries

We can also search (in batch mode) some specific rules when the clustering is done, make queries when the clustering is completed, the main difference between the incremental clustering is because it searches over the new generated clusters, instead of comparing the old and new cluster. We could ask the same rules we defined above and we can search all clusters that reach some condition regardless when it was created. Also, the malware analyst can modify the rules and ask the clusters every time he wants.

6.2. Not-so-benign

To detect supposedly benign samples we define two ratios in a cluster, (1) r1 is the number of samples with at least one detection over all cluster samples and (2) r4 the number of samples with at least four detections over all cluster samples.

Table 6.1 shows the percentages of the *Fully malicious* clusters (cluster that all samples have at least x VT Score, where x is 1 or 4), *Fully benign* clusters (cluster that all samples have up to x VT Score, where x is 1 or 4), *Mixed* clusters (cluster that contain malicious and benign samples), from those clusters, we extract *malicious majority*, that are clusters with at least half of the samples are malicious. This is computed over the whole FVG-Vhash, FVG-Imphash, and FVG-thumbprint clusterings. For instance, 46% of non-singletons FVG-vhash clusters are fully malicious (i.e., all samples have at least 4 VT Score), 44% are fully benign (i.e., all samples have less than 4 VT Score), 10% of samples combine samples with more and less 4 VT Score, of those, 6% have at least more than half of the samples 4 VT Score.

	Vhash		**Imphash**		**Thumbprint**	
Cluster Type	**r1**	**r4**	**r1**	**r4**	**r1**	**r4**
Fully malic.	73%	46%	89%	81%	50%	19%
Fully benign	19%	44%	6%	14%	23%	65%
Mixed	8%	10%	5%	5%	27%	15%
Mal. majority	5%	6%	3%	3%	17%	8%

Table 6.1: Fraction non-singleton FVG clusters classified as fully malicious, fully benign, mixed, or malicious majority.

We propose some examples of the interesting found samples:

vhash:7a41a7dd4a319c77194ec2a5f6c78aa1. This cluster contains 99.96% samples with at least 1 VT Score and 99.85% with at least 4 VT Score. The cluster is clearly malicious but there are 5 samples that any antivirus vendor has detected it as malicious, so this samples can be reported to an analyst. In fact, there are 15 samples with less than 4 VT Score that could also be interesting for the analyst.

thumb:0069be5da75d35577a5b1b02810571fc72d43162. This *peexe* cluster has 12 samples, the subject name is “ZIRPER\zirpe” and all of them has an invalid self-signed certificate. 11 of the samples are detected by at least 4 AV vendors, however, one of them are not detected by any AV company. The samples have 4 vhash values and the non-detected do not share the vhash value with the other cluster samples. Also, no imphash value was extracted by any of the cluster samples by VT, and the 12 samples belongs to the three filenames, AntiCheatS.exe (7), AntiCheat.exe (4), and AntiCheatUpdater.exe (1).

Chapter 7

Conclusions and Future Work

People do not like to think. If one thinks, one must reach conclusions. Conclusions are not always pleasant.

Helen Keller

This section summarizes the conclusions extracted, proposes future research lines derived from the work, and describes upcoming improvements of the tool.

7.1. Conclusions

The conclusions extracted from this work are:

- We have collected and processed 328M reports for 235M samples during our one-year observation range. VT collects everyday on average 1.7M of reports for 1.5M samples being 1M new.
- Over the whole year, 89% of the VirusTotal samples (328.3M out of 209.6M) are new (have never been submitted to VirusTotal before we started collecting).
- The feed lacks a unified filetype detection, so we develop it in order to extract statistics. We discover that 66.0% are peexe samples, followed by javascript (8.9%), HTML (5.3%), and PDF (4.8%). The feed is really diverse and filetype identification is challenging for 0.52% of the samples (1.2M).
- The VT File Feed is not a malware feed. In fact, half of its volume is for samples with no antivirus detections. Still, it contains more malware than the long AV telemetry.
- The VT File Feed contains 4.9K families with at least 100 samples. So the feed is diverse and a good source to create datasets. Especially for Windows and Android malware.
- At least 0.3% of fresh samples are originally FUD, i.e., in their first scan they have zero detections but later are detected by at least 4 antivirus. Also, PDF files are more likely to be FUD than other filetypes.
- Just 5.6% of the feed and 3.7% of the *peexe* samples are signed.
- We have compared the VT File Feed with the telemetry of a large security vendor. In spite of the telemetry being 17 times larger than the VT File Feed volume, the telemetry has 8 times less malware.

- New samples get detected a median of 4.4 hours early in the telemetry than in the VT File Feed. The telemetry samples are largely disjoint from the VT File Feed samples, that implies that VirusTotal may not be a good source to extract the impact on real devices.
- We have also compared the VT File Feed with popular malware datasets, namely VirusShare, MalShare, and MalwareBazaar. VT File Feed is larger by more than x200 during our observation period compared with the other datasets. We also notice that there is a delay in VirusShare dataset between when the binaries packages are created and when it is released online. In fact, the 425 package was created 22nd March 2021 and it was published 17th June 2022 (almost 15 months later)
- We have evaluated three clustering approaches in four ground truths. Our results show that Feature Value Grouping can produce clusters with 90.0%-99.9% precision, and can cluster the VT File Feed by vhash in 15 hours.
- We have proposed a threat hunting technique that reveals 29K of malicious clusters (with more than half of the samples have at least 4 antivirus detections) contain 190K samples with zero detections (supposedly benign samples).

7.2. Future Work

For future work, we plan to implement a comparative analysis over different AV engines (e.g., Symantec and Avira), to know what type of samples each does not recognize as malicious and why, also to improve the detection and labeling module for those vendors. Other proposal for future work is to investigate other clustering approaches that can scale over millions of samples in a reasonable time and also could have good results in terms of F1 score.

We would like to add the differential contribution for the Avping and Telemetry datasets. With the differential contribution we would know how different is a dataset from the other ones. Also, it could be a good idea to extract the exclusive contribution for all datasets analyzed in this project (VT File Feed, Telemetry, Avping, VirusShare, MalShare, and MalwarBazaar) to know how much difference exists between the small datasets versus the large ones.

If we had unlimited VirusTotal queries, we could download all 190K not-so-benign samples and automate the execution in a sandbox to infer if they are really malicious or not. Finally, we plan to request VirusTotal to reanalyze those binaries to get the latest labels to recalculate the number of not-so-benign samples.

Bibliography

[1] A. Calleja, J. Tapiador, and J. Caballero, “The malsource dataset: Quantifying complexity and code reuse in malware development,” *IEEE Transactions on Information Forensics and Security*, vol. 14, no. 12, pp. 3175–3190, 2018.

[2] “Virustotal last 7 days statistics,” 2022, https://www.virustotal.com/gui/stats/.

[3] P. Gao, F. Shao, X. Liu, X. Xiao, Z. Qin, F. Xu, P. Mittal, S. R. Kulkarni, and D. Song, “Enabling efficient cyber threat hunting with cyber threat intelligence,” in *2021 IEEE 37th International Conference on Data Engineering (ICDE)*. IEEE, 2021, pp. 193–204.

[4] S. Homayoun, A. Dehghantanha, M. Ahmadzadeh, S. Hashemi, R. Khayami, K.-K. R. Choo, and D. E. Newton, “Drthis: Deep ransomware threat hunting and intelligence system at the fog layer,” *Future Generation Computer Systems*, vol. 90, pp. 94–104, 2019. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167739X17328467

[5] A. Nappa, M. Z. Rafique, and J. Caballero, “The malicia dataset: identification and analysis of drive-by download operations,” *International Journal of Information Security*, vol. 14, no. 1, pp. 15–33, 2015.

[6] D. Arp, M. Spreitzenbarth, M. Hubner, H. Gascon, K. Rieck, and C. Siemens, “Drebin: Effective and explainable detection of android malware in your pocket.” in *Ndss*, vol. 14, 2014, pp. 23–26.

[7] F. Wei, Y. Li, S. Roy, X. Ou, and W. Zhou, “Deep Ground Truth Analysis of Current Android Malware,” in *Conference on Detection of Intrusions and Malware & Vulnerability Assessment*, 2017.

[8] D. Gibert, C. Mateu, and J. Planes, “The rise of machine learning for detection and classification of malware: Research developments, trends and challenges,” *Journal of Network and Computer Applications*, vol. 153, p. 102526, 2020.

[9] P. Kotzias, S. Matic, R. Rivera, and J. Caballero, “Certified pup: abuse in authenticode code signing,” in *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, 2015, pp. 465–478.

[10] L. Catuogno and I. Visconti, “An architecture for kernel-level verification of executables at run time,” *The Computer Journal*, vol. 47, no. 5, pp. 511–526, 2004.

[11] P. Kotzias, J. Caballero, and L. Bilge, “How did that get in my phone? unwanted app distribution on android devices,” in *2021 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2021, pp. 53–69.

[12] S. Sebastian and J. Caballero, “Towards attribution in mobile markets: Identifying developer account polymorphism,” in *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, 2020, pp. 771–785.

[13] F. Pagani, M. Dell'Amico, and D. Balzarotti, "Beyond precision and recall: understanding uses (and misuses) of similarity hashes in binary analysis," in *Proceedings of the Eighth ACM Conference on Data and Application Security and Privacy*, 2018, pp. 354–365.

[14] J. Kornblum, "Identifying almost identical files using context triggered piecewise hashing," *Digital investigation*, vol. 3, pp. 91–97, 2006.

[15] A. M. Robertson and P. Willett, "Applications of n-grams in textual information systems," *Journal of Documentation*, 1998.

[16] E. Damiani, S. D. C. di Vimercati, S. Paraboschi, and P. Samarati, "An open digest-based technique for spam detection." in *PDCS*. Citeseer, 2004, pp. 559–564.

[17] J. Oliver, C. Cheng, and Y. Chen, "Tlsh–a locality sensitive hash," in *2013 Fourth Cybercrime and Trustworthy Computing Workshop*. IEEE, 2013, pp. 7–13.

[18] CARO Virus Naming Convention. [Online]. Available: http://www.caro.org/articles/naming.html

[19] D. Beck and J. Connolly, "The common malware enumeration initiative," in *Proceedings of the Virus Bulletin Conference*, 2006.

[20] A. Mohaisen and O. Alrawi, "Av-meter: An evaluation of antivirus scans and labels," in *International conference on detection of intrusions and malware, and vulnerability assessment*. Springer, 2014, pp. 112–131.

[21] S. Chakradeo, B. Reaves, P. Traynor, and W. Enck, "Mast: Triage for market-scale mobile malware analysis," in *Proceedings of the sixth ACM conference on Security and privacy in wireless and mobile networks*, 2013, pp. 13–24.

[22] F. Kaczmarczyck, B. Grill, L. Invernizzi, J. Pullman, C. M. Procopiuc, D. Tao, B. Benko, and E. Bursztein, "Spotlight: Malware lead generation at scale," in *Annual Computer Security Applications Conference*, 2020, pp. 17–27.

[23] J. Jang, D. Brumley, and S. Venkataraman, "Bitshred: feature hashing malware for scalable triage and semantic analysis," in *Proceedings of the 18th ACM conference on Computer and communications security*, 2011, pp. 309–320.

[24] A. Lakhotia, A. Walenstein, C. Miles, and A. Singh, "Vilo: a rapid learning nearest-neighbor classifier for malware triage," *Journal of Computer Virology and Hacking Techniques*, vol. 9, no. 3, pp. 109–123, 2013.

[25] M. Sebastián, R. Rivera, P. Kotzias, and J. Caballero, "Avclass: A tool for massive malware labeling," in *International symposium on research in attacks, intrusions, and defenses*. Springer, 2016, pp. 230–253.

[26] M. Hurier, G. Suarez-Tangil, S. K. Dash, T. F. Bissyandé, Y. Le Traon, J. Klein, and L. Cavallaro, "Euphony: Harmonious unification of cacophonous anti-virus vendor labels for android malware," in *2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR)*. IEEE, 2017, pp. 425–435.

[27] S. Zhu, J. Shi, L. Yang, B. Qin, Z. Zhang, L. Song, and G. Wang, "Measuring and modeling the label dynamics of online anti-malware engines," in *29th {USENIX} Security Symposium ({USENIX} Security 20)*, 2020, pp. 2361–2378.

[28] S. Sebastián and J. Caballero, "Avclass2: Massive malware tag extraction from av labels," in *Annual Computer Security Applications Conference*, 2020, pp. 42–53.

[29] D. Kirat, L. Nataraj, G. Vigna, and B. Manjunath, "Sigmal: A static signal processing based malware triage," in *Proceedings of the 29th Annual Computer Security Applications Conference*, 2013, pp. 89–98.

[30] P. Kotzias, L. Bilge, and J. Caballero, "Measuring {PUP} prevalence and {PUP} distribution through pay-per-install services," in *25th {USENIX} Security Symposium ({USENIX} Security 16)*, 2016, pp. 739–756.

[31] A. S. Buyukkayhan, A. Oprea, Z. Li, and W. Robertson, "Lens on the endpoint: Hunting for malicious software through endpoint data analysis," in *International Symposium on Research in Attacks, Intrusions, and Defenses.* Springer, 2017, pp. 73–97.

[32] D. Kim, B. J. Kwon, and T. Dumitraş, "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*, 2017, pp. 1435–1448.

[33] D. Kim, B. J. Kwon, K. Kozák, C. Gates, and T. Dumitras, "The broken shield: Measuring revocation effectiveness in the windows code-signing PKI," in *27th USENIX Security Symposium (USENIX Security 18).* Baltimore, MD: USENIX Association, Aug. 2018, pp. 851–868. [Online]. Available: https://www.usenix.org/conference/usenixsecurity18/presentation/kim

[34] A. Nappa, M. Z. Rafique, and J. Caballero, "The MALICIA Dataset: Identification and Analysis of Drive-by Download Operations," *International Journal of Information Security*, vol. 14, no. 1, pp. 15–33, February 2015.

[35] Vector Guo Li and Matthew Dunn and Paul Pearce and Damon McCoy and Geoffrey M. Voelker and Stefan Savage, "Reading the Tea leaves: A Comparative Analysis of Threat Intelligence," 2019.

[36] X. Bouwman, H. Griffioen, J. Egbers, C. Doerr, B. Klievink, and M. Van Eeten, "A different cup of {TI}? the added value of commercial threat intelligence," in *29th USENIX Security Symposium (USENIX Security 20)*, 2020, pp. 433–450.

[37] V. Kundeti. My journey towards emptiness!!: [TECH] algorithmic details of UNIX sort command. [Online]. Available: http://vkundeti.blogspot.com/2008/03/tech-algorithmic-details-of-unix-sort.html

[38] J. Upchurch and X. Zhou, "Variant: a malware similarity testing framework," in *2015 10th International Conference on Malicious and Unwanted Software (MALWARE).* IEEE, 2015, pp. 31–39.

[39] J. Oliver, M. Ali, H. Liu, and J. Hagen, "Fast clustering of high dimensional data clustering the malware bazaar dataset," Tech. Rep., 2021.

[40] F. Wei, Y. Li, S. Roy, X. Ou, and W. Zhou, "Deep ground truth analysis of current android malware," in *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment.* Springer, 2017, pp. 252–276.

[41] P. Kotzias, S. Matic, R. Rivera, and J. Caballero, "Certified PUP: Abuse in Authenticode Code Signing," in *ACM Conference on Computer and Communication Security*, 2015.

[42] D. Arp, M. Spreitzenbarth, M. Hübner, H. Gascon, and K. Rieck, "Technical report ifi-tb-2013-02 drebin: Efficient and explainable detection of android malware in your pocket," 2013.

Appendix A

Tools and resources

The tools to develop the desired work are the following one:

- VirusTotal File Feed with 255 GB of daily files provided by NortonLifeLock [1]
- Laptop
- GNU/Linux Operating System
- Integrated Development Environment (IDE) PyCharm
- LaTeXText processor
- Git Version Control

[1] https://nortonlifelock.com