Measuring and Modeling the Label Dynamics of Online Anti-Malware Engines

Shuofei Zhu[1], Jianjun Shi[1,2], Limin Yang[3], Boqin Qin[1,4], Ziyi Zhang[1,5], Linhai Song[1], Gang Wang[3]

[1]*The Pennsylvania State University*
[2]*Beijing Institute of Technology*
[3]*University of Illinois at Urbana-Champaign*
[4]*Beijing University of Posts and Telecommunications*
[5]*University of Science and Technology of China*

Abstract

VirusTotal provides malware labels from a large set of anti-malware engines, and is heavily used by researchers for malware annotation and system evaluation. Since different engines often disagree with each other, researchers have used various methods to aggregate their labels. In this paper, we take a data-driven approach to categorize, reason, and validate common labeling methods used by researchers. We first survey 115 academic papers that use VirusTotal, and identify common methodologies. Then we collect the daily snapshots of VirusTotal labels for 14,000 files (including a subset of manually verified ground-truth) from 65 VirusTotal engines over a year. Our analysis validates the benefits of threshold-based label aggregation in stabilizing files' labels, and also points out the impact of poorly-chosen thresholds. We show that hand-picked "trusted" engines do not always perform well, and certain groups of engines are strongly correlated and should not be treated independently. Finally, we empirically show certain engines fail to perform in-depth analysis on submitted files and can easily produce false positives. Based on our findings, we offer suggestions for future usage of VirusTotal for data annotation.

1 Introduction

Online anti-malware scanning services such as VirusTotal [11] have been widely used by researchers and industrial practitioners. VirusTotal connects with more than 70 security vendors to provide malware scanning. Users (*e.g.*, researchers) can submit a file and obtain 70+ labels from different engines to indicate whether the file is malicious. This capability has been heavily used to annotate malware datasets and provide system evaluation benchmarks [23, 31, 35, 41, 49, 66, 69].

A common challenge of using VirusTotal is that different security engines often disagree with each other on whether a given file is malicious. This requires researchers to come up with a strategy to aggregate the labels to assign a single label to the file. In addition, recent works show that the labels of a given file could change over time [18, 41], which makes it even more difficult to infer the true label of the file.

As such, researchers have tried various methods to handle the label dynamics (*e.g.*, monitoring the labels for a few days) and aggregate the labels across engines (*e.g.*, setting a voting threshold). However, most of these approaches are based on intuitions and researchers' experiences, but lack quantifiable evidence and justifications. Recent efforts that try to measure the label dynamics of VirusTotal are often limited in measurement scale [57] or simply lack "ground-truth" [40], making it difficult to draw a complete picture.

In this paper, we take a data-driven approach to categorize, reason and validate the methodologies that researchers adopted to use VirusTotal for data annotation. Our efforts include (1) analyzing more than 100 research papers published in the past eleven years to categorize their data labeling methods using VirusTotal, and (2) running a measurement over a year to collect daily snapshots of VirusTotal labels for a large set of files from 65 engines. Our goal is to provide data-driven justifications for some of the existing labeling methods (if they are reasonable), and more importantly, identify questionable approaches and suggest better alternatives.

Our measurement follows two key principles. First, we use "fresh" files that are submitted to VirusTotal for the first time. This allows us to observe the label dynamics from the very beginning without being distracted by the files' previous history. Second, we track the fine-grained label dynamics by re-scanning the files daily. We construct a *main* dataset that contains 14,423 files and their daily labels of 65 engines for more than a year. This dataset is used to measure the label dynamics and the relationships between engines. Then to inspect the label "correctness" of engines, we construct smaller *ground-truth* datasets that contain manually-crafted and manually-verified malware and benign files (356 files). In total, we collected over 300 million data points.

First, we measure how often individual engines flip their labels on a given file over time. We find over 50% of the label flips are extremely short-lived, and will flip back quickly the next day (*i.e*, "hazard" flips). Label flips widely exist across

files and engines, and they do not necessarily disappear even after a year. Instead, we show that threshold-based label aggregation (*i.e.*, a file is malicious if $\geq t$ engines give malicious labels) is surprisingly effective in tolerating label dynamics if the threshold t is set properly. However, the most-commonly used $t = 1$ is not a good threshold.

Second, we model the relationships between different engines' labels, to examine the "independence" assumption made by existing works. By clustering engines based on their label sequences for the same files, we identify groups of engines with highly correlated or even identical labeling decisions. In addition, through a "causality" model, we identify engines whose labels are very likely to be influenced by other engines. Our results indicate that engines should not be weighted equally when aggregating their labels.

Third, we use the ground-truth data to inspect the labeling accuracy of engines, and find very uneven performance from different engines. Interestingly, the "perceived" high-reputation engines by existing works are not necessarily more accurate. A subset of engines (including certain high-reputation engines) tend to produce many false positives when the files are obfuscated. This indicates a lack of in-depth analysis from certain engines, and also poses a challenge to find a universally good method (and threshold) to aggregate labels.

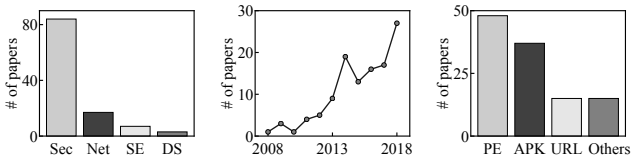
Our contributions are summarized as the following:

- We surveyed 115 academic papers to categorize their methods to use VirusTotal for data labeling.
- We collected the daily snapshots of labels from 65 anti-malware engines for more than 14,000 files over a year. We used the dataset to reason and validate common methodologies for label aggregation. We released the dataset to benefit future research.
- We measured the potential impact introduced by the unstable and inconsistent labels. We identified questionable methodologies and offered suggestions to future researchers on the usage of VirusTotal.

2 Literature Survey: VirusTotal Usage

We start by surveying how researchers use VirusTotal for data annotation. We collect recent papers published in Security, Networking, Software Engineering, and Data Mining, and then categorize their data labeling methods. The goal is to set up the contexts for our measurements.

Collecting Research Papers. We collect conference papers by searching in Google Scholar with the keyword "VirusTotal". We only consider high-quality conference papers in peer-reviewed venues. In total, we identify 115 relevant papers published in the last eleven years (2008 – 2018). The authors either use VirusTotal to label their datasets [23, 31, 41, 49] or leverage the querying/scanning API of VirusTotal as a building block of their proposed systems [35, 66, 69].



(a) Conference (b) Publication Year (c) File Type

Figure 1: Characteristics of related papers. *Sec: Security, Net: Networking, SE: Software Engineering, and DS: Data Science.*

As shown in Figure 1(a), the vast majority of papers (84 out of 115) are published in security conferences (42 papers from the "big-four": S&P, CCS, USENIX Security, and NDSS). Figure 1(b) shows the upward trend of VirusTotal usage among researchers over time. As such, it is increasingly important to formulate a reliable method to use VirusTotal. In Figure 1(c), we categorize the papers based on the type of files/programs that the researchers are scanning, including Portable Executable (PE) files [16, 42, 70], Android APK [15, 23, 36], URL [73, 82], and others (Flash file, PDF) [20, 31, 74, 79]. We find that PE is the most popular file type.

VirusTotal Scanning APIs. VirusTotal provides *file scanning* and *URL scanning* services. Its scanning interface connects with more than 70 security vendors. These security vendors either share their scanning engines for VirusTotal to deploy (as a software package) or provide the online scanning APIs that accept file submissions. To use the VirusTotal API to label a file, users can submit the file to VirusTotal, and VirusTotal returns the scanning results from the 70 vendors (the returned labels could be "malicious" or "benign", indicated by the "detected" field in VirusTotal responses).

It is known that VirusTotal and its third-party vendors keep updating their anti-malware engines, and thus the labels of a given file may change over time. Given a file will receive labels from multiple engines, it is not uncommon for the engines to disagree with each other. For these reasons, researchers often need to aggregate/process the results to generate a single label for the given file (*i.e.*, labeling method).

How Researchers Aggregate the Labels. By manually analyzing these 115 papers[1], we find that 22 papers have used VirusTotal but did not clearly describe their data processing method. As such, we use the rest 93 papers to categorize the mainstream data labeling approaches. A summary is shown in Table 1. Note that different categories may overlap.

First, threshold-based method. Most papers (82 out of 93) use a threshold t to determine whether a file is malicious or benign. If t or more engines return a "malicious" label, then the file is labeled as malicious. Here t can be an absolute number or a ratio of engines. 50 papers set $t = 1$: a file is malicious if at least one engine thinks it is malicious [51, 60, 72, 73, 78, 81]. For 24 papers, t is set to be larger than one. Another eight papers set t as a ratio [14, 29, 30, 48, 58, 74,

[1]The full paper list is available under the following link: https://sfzhu93.github.io/projects/vt/paper-list.html.

Table 1: Summary of related papers. *We select 5 representative papers for each category. Different categories may overlap.*

Data labeling		# Papers	Representative Papers
Threshold	$t = 1$	50	[60, 62, 72, 73, 78]
	$1 < t < 5$	9	[39, 44, 47, 64, 76]
	$t \geq 5$	15	[24, 42, 43, 46, 75]
	$t < 50\%$	4	[48, 58, 74, 87]
	$t \geq 50\%$	4	[14, 29, 30, 84]
Reputable Subset		10	[15, 22, 27, 84, 85]
No Aggregation		10	[34, 50, 53, 54, 80]
Dynamic Label Analysis		11	[34, 41, 58, 67, 73]

84, 87]. We only found a few papers that set an aggressive threshold. For example, two papers set $t = 40$ [18, 46]. Four papers [14, 29, 30, 84] sets the threshold as 50% of the engines.

Second, selecting high-reputation engines. In ten papers, the authors think that different engines are not equally trustworthy. As such, the authors hand-picked a small set of engines that are believed to have a good reputation. However, the high-reputation set is picked without a clear criterion, and the set is different in different papers. For example, Chandramohan et al. [22] only consider five engines' results. Only two of the five engines appear in Arp et al. [15]'s trusted set.

Third, no aggregation. Ten papers directly use VirusTotal's results to build their own system or as their comparison baselines. For example, Graziano et al. [34] use VirusTotal's detection rate as one of their features to train their system. For the rest nine papers, the authors submit samples to VirusTotal, to show their detection techniques outperform VirusTotal engines [53, 80], or to confirm that they have identified important security issues [19, 50, 56], or to demonstrate the effectiveness of malware obfuscations [25, 52, 54, 86].

How Researchers Handle the Label Changes. Surprisingly, the vast majority of the papers (104/115) only take one single snapshot of the scanning results without considering the dynamic changes of labels. A small number of papers considered the potential label changes, and decided to wait for some time before using the labels [18, 58, 73]. The waiting time varies from ten days [58] to more than two years [18]. Others submit the files multiple times to see the differences [25, 41, 67, 79, 83].

Our Goals Our literature survey has two takeaways. First, most researchers use a simple threshold or a trusted set of vendors to determine if a file is malicious. The threshold and trusted set are usually hand-picked without validating the rationality of choices. Second, most researchers only take one snapshot of VirusTotal results, failing to consider possible result changes. In this paper, we seek to run empirical measurements on label dynamics to provide justifications for some of the existing labeling methods. More importantly, we want to identify potentially questionable approaches and propose better alternatives. Several works are related to ours. We briefly discuss the differences.

Closely Related Works. Peng et al. [57] examined the URL scanning engines of VirusTotal for phishing URL detection (data over a month). Our work focuses on anti-malware engines (*i.e.*, file scanning) over a long time (a year). We show anti-malware engines have different/contradicting characteristics compared to URL engines (details provided later).

Kantchelian et al. [40] proposed a machine learning model to aggregate VirusTotal labels. However, they assumed VirusTotal engines are independent of each other, for which we show contradicting evidence in this paper. In addition, the data collection method of [40] is different (*e.g.*, file re-scanning frequency is not controlled), which lost the opportunity to observe fine-grained label dynamics. Finally, [40] did not have real ground-truth for the malware dataset, and assumed VirusTotal labels become stable after a file is submitted to VirusTotal for four weeks (for which we have different observations). Compared with [40], our unique contribution is that we identify previously unknown dynamic patterns (*e.g.*, hazard label flips), measure the "influence" among vendors, and provide simpler suggestions for researchers.

Other Related Works. In addition to malware scanning, VirusTotal also provides malware family information for known malware samples. Researchers found that different VirusTotal engines may attribute the same malware to different malware families [37, 55, 63]. One existing work has measured the correctness and inconsistency of malware family names based on manually labeled datasets [55]. Other works aim to assign a family name to a malware sample by aggregating the family names reported by different VirusTotal engines [37, 63]. These works look into a different aspect of VirusTotal engines compared to ours. More importantly, their analysis was based on a single snapshot of VirusTotal scan, and did not consider the possible changes of VirusTotal labels and malware family names over time.

3 Data Collection

To achieve the above goal, we need to capture the dynamic label changes of different engines over a long period of time. Our measurement follows two main principles. First, we choose "fresh" files for our study, *i.e.*, files that are submitted to VirusTotal for the first time. This allows us to observe the label dynamics from the very beginning without being distracted by the files' previous histories. Second, to observe the fine-grained changes, we leverage the `rescan` API to trigger the VirusTotal engines to analyze our files every day. Then we use the `report` API to query the latest scanning results every day. Table 2 summarizes all the datasets we collected.

Table 2: Dataset summary. *U: Unlabeled, M: Malware, B: Benign.*

Dataset	# Files	Type	Observation Period	# Days
Main	14,423	U	08/2018 – 09/2019	396
Malware-I	60	M	06/2019 – 09/2019	105
Malware-II	60	M	06/2019 – 09/2019	97
Benign-I	80	B	06/2019 – 09/2019	93
Benign-II	156	B	07/2019 – 09/2019	70

3.1 Main Dataset

To obtain a large set of "fresh" files, we use VirusTotal's `distribute` API. VirusTotal receives new file submissions from users all over the world on a daily basis. The API returns information of the latest submissions from users. The information includes a submission's different hash values, whether the file has been submitted to VirusTotal before, the file type, and all the engines' scanning results. We randomly sampled 14,423 PE files that were submitted to VirusTotal on August 31, 2018 *for the first time*. We focus on PE files since it is the most popular submitted file type on VirusTotal [10, 65]. In addition, we hope to include both malicious and benign files in our collection. We purposely selected samples so that: (1) about half of the samples (7,197) had a "malicious label" from at least one engine on August 31, 2018 (*i.e.*, day-1); (2) the other half of the samples (7,226) had "benign" labels from all engines. After August 31, 2018, we leverage the `rescan` API to let VirusTotal engines scan the 14,423 files every day and use the `report` API to query the latest scanning results. As we will discuss later, VirusTotal updates its engines on a daily basis, so that using *day* as the crawling granularity allows us to monitor the fine-grained label dynamics, while making good use of our VirusTotal API quota. We do not treat these files as "ground-truth" data, because files submitted to VirusTotal are suspicious files at best. There is an unknown number of true malware samples mixed with benign files, which remain to be detected by VirusTotal engines.

From August 31, 2018 to September 30, 2019, we invoke VirusTotal's `rescan` API for these 14,423 files every day. All the files are in 32-bit. 5,798 files are Win32 DLL files, and the rest are Win32 EXE files. Regarding file size, more than 95% of the PE files are within the range of 4KB to 4MB. During 396 days' data collection period, we successfully collected data on 378 days (95.45%). Due to technical issues (*e.g.*, power-outage, server failures), we missed data on 18 days (4.5%). We argue that the missing data only account for a very small portion, and should not impact our overall conclusions.

3.2 Ground-truth Dataset

The main dataset is large and diversified, but these files do not have ground-truth labels. As such, we create another set of "ground-truth" files to assess the "correctness" of engines.

Creating ground-truth for this study is especially challenging because our goal is to examine the reliability of existing malware engines. This means we could not use any engine's labels as the ground-truth. In addition, we need "fresh" files for our experiment. This means, any well-known malware discovered by existing efforts are not suitable since they were usually scanned by VirusTotal engines in the past. If we use well-known malware, we can only capture the "tails" of the label change sequences. For these reasons, we need to manually craft the ground-truth sets.

Ground-truth Malware. To craft fresh malware sets, our approach is to obfuscate well-known malware to create new binaries. Obfuscation can help create "fresh" binaries that have never been scanned by VirusTotal before. Meanwhile, obfuscation is not necessarily a determining feature of malware — it is also often used by legitimate software to protect their intellectual property (copyright) or protect sensitive functions (*e.g.*, for payments and security) from reverse-engineering [26, 61, 77].

We apply two obfuscation tools CodeVirtualizer [8] and Themida [9] on four existing ransomware (Table 4 in the Appendix) and create two malware sets respectively. Each malware set contains 60 new malware samples obfuscated from the seeds. We choose ransomware as the seeds due to two reasons. First, it is easy to manually verify the malicious actions (*i.e.*, encrypting files, showing the ransomware notes). Second, we manually confirmed that the majority of engines (57/65) advertise that they are capable of detecting ransomware[2], and thus ransomware is a relatively fair benchmark to compare different engines.

For each newly generated sample, we manually run the binary in a virtual machine to confirm the malicious actions are preserved. We also manually confirm that these samples are indeed "fresh" to VirusTotal. These samples have different hash values (*e.g.*, SHA256, ssdeep) from the seeds and can trigger VirusTotal's file scanning after being submitted. We perform daily VirusTotal scanning from June 18, 2019 for Malware-I and from June 25, 2019 for Malware-II. We monitored the files over a shorter period of time (compared to the main dataset), because we have already observed the two malware datasets have similar patterns of label dynamics as the main dataset.

Ground-truth Benign. We have a mixed approach to get two benign sets, with 236 files in total. First, we apply the same obfuscation tools to two benign programs (a sorting algorithm written by ourselves in C and a text editor in Windows). We generate 80 obfuscated goodware to examine potential false positives of VirusTotal engines. These goodware are directly comparable with the ground-truth malware since they are generated in the same way (with different seeds). We call this set as *Benign-I*. We conduct daily VirusTotal scanning on this dataset for 93 days.

[2]The advertisement list is available at `https://sfzhu93.github.io/projects/vt/advertise.html`.

Second, to test real-world goodware, we manually build 156 PE programs (without obfuscation). Among them, 150 PE programs are built using the source code of GNU Core Utilities (*coreutils*) [4] and the *Mono* project [6]. *coreutils* contains a suite of Unix commands (*e.g.*, `cat`, `ls`, `rm`) and we use cygwin [5] to build them into PE files. *Mono* is an open-source .NET development framework, which contains a C# compiler, development environment and various libraries. We build the *Mono* project on an Ubuntu machine. To increase the diversity, we also select six built PE programs from binutils [1], notepad++ [7] and fleck [2] projects. We call this set as *Benign-II*. Just as before, we perform daily VirusTotal scanning on these 156 benign files for 70 days.

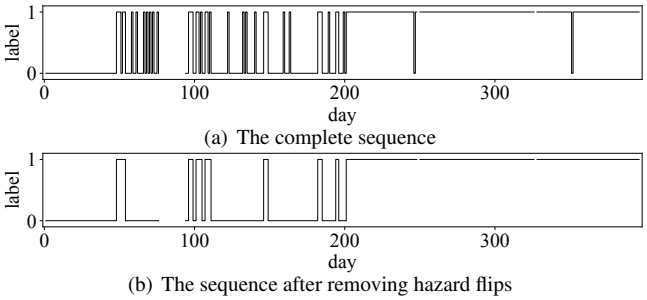
Limitations. We understand that the above ground-truth sets are limited in scale and diversity: the samples are biased towards obfuscated files and the malicious files are seeded with ransomware. This is primarily due to (1) we don't have access to a large number of files (including both benign and malicious files) that have no prior history on VirusTotal; and (2) it takes heavy manual efforts to validate the malicious functionality still exists after obfuscation. Considering the rate limit of VirusTotal, the size of the ground-truth set is already the best effort. As such, this small ground-truth set is only used to complement the main dataset (which is a large sample of *real-world* suspicious files). We use the main dataset to measure the fine-grained label dynamics of VirusTotal over a long period of time. Then we use the ground-truth set to validate some of the observations drawn from the main dataset and cross-examine the correctness of VirusTotal engines.

3.3 Data Summary and Preprocessing

Across the five datasets in Table 2, we collected in total 375,520,749 measurement points. Each measurement point is characterized by a file-ID, a timestamp, an engine name, and a label. These measurement points are generated by 78 different engines in total. However, nine engines were newly added to VirusTotal *after* we started the data collection for the main dataset. We cannot observe these engines' behaviors when the main dataset was firstly submitted to VirusTotal. There are another four engines, which were removed from VirusTotal's engine list during our data collection. The analysis results for these four engines will not help VirusTotal users anymore. As such, we do not consider these 13 engines in our analysis and only focus on the remaining 65 engines in the following sections. After filtering out irrelevant engines, we still have 343,585,060 measurement points. Among them, the *main dataset* contributes 341,668,521 data points.

4 Measuring Label Dynamics

In this section, we formally model the label changes on VirusTotal. We first characterize the different types of temporal label changes and reason the possible causes. We then try to estimate how long one should wait before a file's labels become stable across engines. In the end, we assess the impact of temporal label dynamics on the labeling outcome (given most researchers only submitted files to VirusTotal to scan for once and aggregate VirusTotal labels using a threshold t, as discussed in Section 2). In this section, we focus on the *main dataset* for our analysis.



(a) The complete sequence

(b) The sequence after removing hazard flips

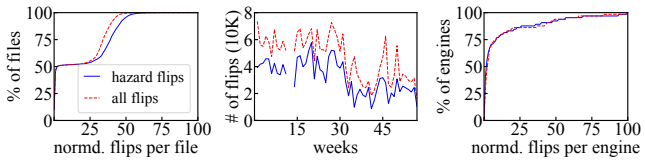
Figure 2: An example file's label sequence from AegisLab.

4.1 Hazard Flips and Non-Hazard Flips

In our context, we model the label dynamics in a form similar to logic signals as a sequence of "0" and "1". More specifically, given a file f and a VirusTotal engine i, the label we obtained daily can be formulated as $S_{i,f} = [l_1, l_2, ..., l_t, ..., l_N]$ where N is the total number of days of data collection, l_t= 0 (benign) or 1 (malicious). A flip refers to a change between two consecutive labels, namely "01" or "10". In total, we have 2,571,809 flips in the main dataset.

We observed an interesting phenomenon, which we call "hazard flip". Given a file, a VirusTotal engine would sometimes *flip its label and then quickly change it back the next day*. We take the term "hazard" from Digital Circuit, which originally represents the temporary fluctuation in the output of the circuit [17]. In our case, hazard refers to a temporary glitch or flip in the labels, namely "010" or "101". More formally, we define a label flip as either "0 $\rightarrow$ 1" or "1 $\rightarrow$ 0". Thus a hazard would contain two flips. We call the two flips in a hazard "hazard flips". Any other flips are referred to as non-hazard flips. In total, we have 1,760,484 hazard flips and 811,325 non-hazard flips.

Figure 2 shows an example. Given a specific file (MD5: `e8799a459bdea599d1bc1615f4b746de`), the original label sequence we obtained from AegisLab is shown in Figure 2(a). After removing hazard flips, the label sequence only containing non-hazard flips is shown in Figure 2(b). We can see that the original label sequence contains many hazards, and some of them last for multiple days. To capture such consecutive hazards, we search each label sequence chronologically and always try to extend an identified hazard. For example, from November 1st, 2018 to November 14th, 2018, the label sequence of AegisLab is "00010101010010". We identify two hazards from it, one is "010101010" and the



(a) CDF of a file's flips and hazard flips (b) # of flips per week over time (c) CDF of an engine's flips and hazard flips

Figure 3: Characteristics of flips and hazard flips.

other one is "010". In total, we find 737,338 hazards with length three ("010" and "101"), 54,801 hazards with length five ("01010" and "10101"), and 8,297 hazards with length seven. The longest hazard lasts for 19 days.

Observation 1: *More than half of the flips on VirusTotal are hazard flips. Hazard flips can be identified by submitting the same file to VirusTotal in three consecutive days.*

4.2 Characteristics of Flips

Next, we analyze the characteristics of flips (including hazard flips) across files, dates, and engines.

Distribution Across Files. In total, 1,352 files (9%) in the main dataset do not contain any flip. For these files, all engines always label them as "benign" throughout our data collection period. Recall that 7,234 files are labeled as benign by all engines on the first day. As such, if a file is labeled as benign by all vendors when firstly submitted to VirusTotal, the probability that there will be no flip on this file is 19%. If a file is labeled as "malicious" by any engine on day-1, at least one flip happens later on the file. For files with flips, on average each file contains 196 flips.

Figure 3(a) shows the CDF of a file's *normalized number of flips*. We normalize a file's flips using the maximum number of flips on a single file (which is 1,054). We find 6,723 (46.61%) files have less than 10 flips (1% of the maximum number of flips), and thus the drawn CDF is close to the y-axis in the beginning. We also draw the CDF for a file's hazard flips (the blue line) which has a similar trend. It turns out that hazard flips and non-hazard flips are highly correlated. We rank files based on their hazard flips and non-hazard flips and compute the Spearman's correlation coefficient [28] between the two rankings. The coefficient is 0.87 with p-value less than 0.01, indicating that files with more hazard flips are more likely to have more non-hazard flips.

Distribution over Time. Figure 3(b) shows the number of flips each week during our data collection time window. We have flips in all 57 weeks. We encountered technical issues in week-12 and week-13 (with some data loss), so that there are fewer flips in these two weeks. On average, each week has 45,119 flips, and the first week has the highest number of flips. Similarly, we also have hazard flips every week.

Distribution Across Engines. Flips are generated by 64 out of the 65 engines. Avast-Mobile labels all samples as benign, and it is the only engine not having flips. Figure 3(c) shows the CDF of an engine's *normalized number of flips*. We normalize each engine's flips using the maximum number of flips from a single engine. The curve is skewed to the left, indicating that a small group of engines contributes to the majority of the flips. For the 64 engines with flips, on average each of them contributes 40,184 (1.56%) flips. However, AegisLab reports 359,221 (13.96%) flips by itself, and it is the engine with the most flips. F-Secure is ranked as the 2nd (297,973 flips), and VIPRE is ranked as the 3rd (233,875 flips). Again, the CDF of hazard flips has a similar trend. We compute the Spearman's correlation coefficient to examine if engines with more hazard flips are likely to have more non-hazard flips. The computed coefficient is 0.75 with a p-value less than 0.01, confirming a strong correlation.

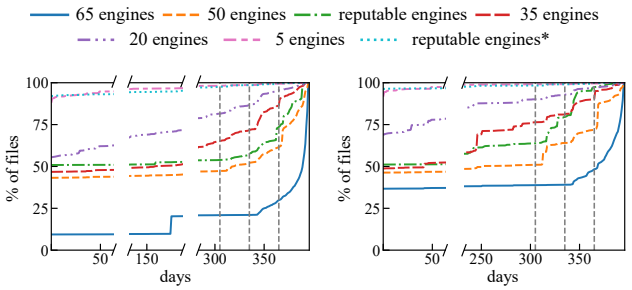
Observation 2: *Both flips and hazard flips widely exist across files, scan dates and engines.*

4.3 Inferring Root Causes of Flips

We tested whether querying VirusTotal API multiple times can resolve (hazard) flips. We found that repeated queries can only address very limited flips, and confirmed that flips are more likely to be caused by internal problems of VirusTotal. To categorize detailed root causes for flips, we mainly use the "update date" and "version information" of each engine used in a scan, provided in VirusTotal responses. Given a flip (l_1, l_2) $(l_1 \neq l_2)$ generated by engine i, we use (u_1, u_2) to represent the engine's last update dates and use (v_1, v_2) to represent the engine versions when i scanned the file. The causes of flips are categorized as follows.

Most commonly, a flip happens when the engine made a model update, representing a decision-change of the engine. 1,710,565 (67%) flips belong to this category where $u_1 < u_2$. For 213,159 (8.3%) flips, their engine version numbers also increase chronologically ($v_1 < v_2$). However, for 1,497,115 (58%) flips, the version numbers are the same for the two consecutive scans ($v_1 = v_2$), meaning the model update is made under the same engine version number.

83,164 (3.2%) flips are likely caused by the inconsistency during engine updates. VirusTotal is backed up by a large (distributed) cloud service to handle the large volume of queries [12]. Multiple instances of an engine are deployed on multiple host machines to handle incoming requests concurrently. Ideally, when there is a new version of an engine, all its instances are upgraded to the new version instantly. However, in reality, we can observe some inconsistency among multiple engine instances (some instances are updated but others did not on the same day). There are 57,450 (2.2%) flips due to using an engine with an older update date to scan a more recent request ($u_1 > u_2$). For example, CrowdStrike has hazards on 3,739 files on day-175. After inspecting the update information on the corresponding three days (u_1, u_2, u_3), we find that for most files, u_1 is equal to u_3, but u_2 is



(a) The original dataset (b) The dataset without hazards

Figure 4: The percentage of files whose labels do not change after day-x. *Reputable engines: the nine high-reputation engines mentioned by previous literature; reputable engines*: the two high-reputation engines mentioned twice by previous literature.*

much larger (both u_1 and u_3 are in 2018, but u_2 is in 2019). Sometimes, not all cloud instances are upgraded to the same engine version even if they updated on the same day. There are 25,714 (1.0%) flips caused by handling two consecutive scanning requests using an engine updated on the same day but with two different version numbers ($u_1 = u_2$ and $v_1 \neq v_2$).

380,807 (15%) flips are likely caused by the non-determinism of engines. In this case, an engine is used in two consecutive scans with the same update date ($u_1 = u_2$) and the same version ($v_1 = v_2$), but reports two different labels ($l_1 \neq l_2$). We do not have a good explanation for the non-determinism based on the current data. We cannot use desktop engines to validate the non-determinism since VirusTotal engines are different from their desktop versions [13].

For the other 397,273 (15%) flips, the data fields for update date or version information are "null" in their VirusTotal responses, and we cannot categorize their root causes. Note that the "detected" values (*i.e.*, label information) are still available in these responses, and thus the missing information does not impact our analysis in other sections.

Observation 3: *Engines' model update is the major reason of flips. However, the inconsistency during engine updates and engines' non-determinism have contributed a non-trivial portion of the flips.*

4.4 Label Stabilization

So far, we observe that label flips are quite prevalent. A practical question is how long a user should wait before a file's labels become stable. In this subsection, we characterize the label stabilization patterns over time and its predictability.

Considering All Engines. Figure 4(a) shows the percentage of files whose VirusTotal labels do not change since day-x until the end of our data collection (the blue line, all 65 engines). For example, when $x = 50$, only 9.37% of the files are stable, meaning these files' labels from all vendors did not change since day-50. The percentage increases very slowly for most of the time, but it suddenly jumps from 9.74% to 20.22% on day-176. This is an anomaly because CrowdStrike has hazards on 3,739 files on day-175 (reasons discussed in Section 4.3). The percentage starts to increase very quickly around day-350, mainly because the time period between x and the end of data collection is too small. Indeed, it is possible that flips can still happen after our data collection period.

Excluding Highly Dynamic Vendors. We expect a file to stabilize quickly if we exclude highly dynamic engines. We rank engines based on their total number of flips. We gradually remove engines with more flips and compute the percentage. As shown in Figure 4(a), removing engines can immediately increase the percentage of stable files. For example, removing 15 engines (50 engines left) can increase the percentage of stable files on day-1 from 9.37% to 43.19%. However, to stabilize most files quickly, we need to remove many engines. In the extreme case, if we remove most engines and only consider the five engines[3] with fewest flips, the initial percentage of stable files is very high (88.05%) on day-1. The percentage increases to 95% on day-77. This, to some extent, confirms that flips widely exist across engines. We cannot remove a small number of engines to make files stabilize.

Only Considering Reputable Engines. As discussed in Section 2, we find ten papers that hand-picked "high-reputation" engines for data labeling. Among them, five papers are related to PE malware, and only three out of the five papers provide detailed lists of their high-reputation engines. This produces a set of nine "reputable engines" for our analysis (Table 5 in the Appendix). In Figure 4(a), we show the percentage of stabled files when we only consider reputable engines. We show that files do not stabilize quickly — it is very similar to the 35-engine line. The reason is some of the reputable engines (*e.g.*, F-Secure) have a large number of flips. Note that among the nine engines, there are two engines (Kaspersky and Symantec) that are mentioned by more than one paper. We refer to these two engines as "highly reputable engines". If we only consider these two engines (the "reputable engines*" line), we observe that most files are stabilized very quickly.

Excluding Hazards Since it is easy to identify and remove hazards (by submitting a file to VirusTotal in three consecutive days), we re-examine the results after removing hazards. As shown in Figure 4(b), removing hazards can help increase the percentage of stabilized files. The initial percentage of stabilized files (considering all engines) changes from 9.37% to 36.69% on day-1. However, removing hazards does not necessarily significantly speed up the file stabilization.

Observation 4: *Waiting for a longer period of time does not guarantee to have more stable labels from individual engines, unless we only consider a small set of engines.*

[3] NANO-Antivirus, K7AntiVirus, Zoner, Ikarus, and Avast-Mobile.

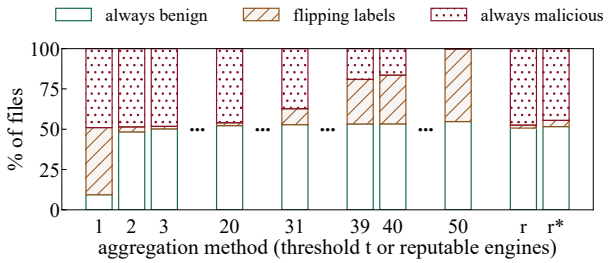


Figure 5: Aggregation method vs. file labels. "always malicious": *files with only malicious aggregated labels throughout all the days*; "always benign": *files with only benign aggregated labels throughout all the days*; "flipping labels": *files with both benign and malicious aggregated labels*. "r": *reputable engines*; "r*": *the two highly reputable engines*.

4.5 Impact of Flips

In Section 2, we find that most researchers only submit a file to VirusTotal once and simply use a threshold t to aggregate VirusTotal labels. If t or more engines have labeled the file as malicious, the file's aggregated label is malicious. We estimate the potential impact of (hazard) flips on this label aggregation policy for different t values by measuring how many files have different *aggregated labels* (0 and 1) on different days during our data collection time window (396 days).

Setting $t = 1$. This means a file is malicious as long as one engine gives a malicious label. As shown in Figure 5 (the left-most bar), 1,352 (9.4%) files only have benign aggregated labels and 7,067 (49.0%) files only have malicious aggregated labels throughout the 396 days. The rest 6,004 (41.6%) files have both benign and malicious aggregated labels, and they are the files influenced by flips. If these files are submitted to VirusTotal for the second time, there is a chance that a VirusTotal user will draw a different conclusion on these files. This suggests $t = 1$ is not a good threshold (and yet 50 out of 93 papers use $t = 1$, see Table 1).

After removing hazards, the number of files with only benign aggregated labels increases to 5,289 (36.7%). The number of files with only malicious aggregated labels is almost unchanged (7,074). The number of files influenced by non-hazard flips is 2060 (14.3%).

Setting $2 \leq t \leq 39$. The second bar of Figure 5 shows the result for $t = 2$. The number of files only having benign aggregated labels increases to 6,975 (48.4%). There are 7,006 (48.6%) files only having malicious labels. The number of files influenced by flips significantly decreases to 442 (3.1%). Flips have no impact on the majority of files. Although flips can happen on these files and actually a different set of engines report malicious labels over time, the flips somehow cancel each other's effect and not influence the aggregated labels. There are very few files influenced by flips. If we remove hazards, the number of files influenced by flips further decreases to 253. When we choose t from 2 to 39, the ratio of files influenced by flips is always less than 30%. If we want to maintain the ratio of "flip-influenced" files below 10%, we need to pick t between 2 to 31.

Setting $t \geq 40$. As shown in Figure 5, when t is equal to 40, there are more files having label changes (*i.e.*, influenced by flips). There are 7,690 (53.3%) files only with benign aggregated labels and 2,376 (16.4%) files only containing malicious aggregated labels. Thus, 4,357 (30.2%) files are influenced by flips. When we choose a larger t like t =50, we can see a more obvious increase of files influenced by flips (compared to $t = 40$), and there are 6,499 (45.0%) files influenced.

Reputable Engines Only ($t = 1$). If we only consider the nine reputable engines, there are 263 (1.8%) files influenced by flips (bar "r" in Figure 5) and 220 (1.5%) files influenced by non-hazard flips. If we only consider the two highly reputable engines, the numbers of files that are influenced by flips and non-hazard flips become 554 (3.8%) (bar "r*" in Figure 5) and 401 (2.7%), respectively. The numbers of influenced files are significantly smaller compared with all engines.

We want to emphasize that even though the threshold-method helps stabilize the aggregated labels, it does not necessarily mean the aggregated labels are correct. Label correctness will be discussed in Section 6.

Observation 5: *Flips can heavily influence labeling aggregation results when threshold t is too small or too large. When selecting t from a reasonable range (2– 39), the aggregated labels are likely to be stable.*

5 Relationships Between VirusTotal Engines

While using a simple threshold helps tolerate label dynamic changes, it makes an implicit assumption that each engine is equally important and relatively independent. Even an early work that aims to predict the label dynamics [40] makes the assumption about the independence between engines. In this section, we seek to examine whether this assumption is true.

First, we measure the *correlations* between different engines' labeling decisions. We apply hierarchical clustering to group engines with a strong labeling similarity. Second, we further examine the potential *causalities* (*e.g.*, how one engine's labels influence another engine). We adopt an existing method [33] to model the influence between different engines. If we can observe correlations or causalities between certain engines, then independence assumption would be questionable. We use the *main dataset* for our analysis.

5.1 Label Correlations Between Engines

To measure the correlation between two engines, we examine how likely the two engines give the same labels to the same files around the same time. More specifically, given a pair of engines (A, B), we compare their label sequences on *the same file* to measure the similarity (or distance). Then we compute

the average similarity score over all the files between A and B. The average similarity score can be used to group similar engines together. In the following, we first discuss our engine clustering algorithm and then discuss our findings.

5.1.1 Engine Clustering

The key to the engine clustering is to define the similarity metric between two label sequences. *Edit distance* is a straightforward metric but is not good at capturing fine-grained temporal similarities. For example, the edit distance between "01000000000000" and "00000000010000" is 2, which is the same as the edit distance between "01000000000000" and "00100000000000". Obviously, the second pair is more correlated since the "timing" between malicious labels is closer.

To encode fine-grained temporal similarities, we divide each label sequence into fixed-sized bins. In each bin, we count the number of 0→1 flips, the number of 1→0 flips, the maximum length of "all-0" sub-sequences, and the maximum length of "all-1" sub-sequences. This forms a feature vector of four values for each bin. Let L be the length of a sequence and S be the size of the bin. Then the feature vector for the sequence has $4 * \lceil L/S \rceil$ dimensions. We do not compute feature vectors using sliding bin to avoid counting the same flip multiple times. Given two sequences, we now can compute a *cosine similarity* between the two feature vectors as the two sequences' similarity score.

For example, assume we choose bin size $S = 7$. Then A's label sequence "01000000000000" can be divided into two bins "0100000" and "0000000". The corresponding feature vector for each bin is [1, 1, 1, 5] and [0, 0, 0, 7] respectively. The entire sequence's feature vector is [1, 1, 1, 5, 0, 0, 0, 7]. Similarly, suppose B's label sequence is "00000000010000", and the feature vector is [0, 0, 0, 7, 1, 1, 1, 4]. The cosine similarity between the two vectors is 0.871. For the two engines A and B, their similarity score is the average sequence-level similarity score over all the files.

Based on the similarity metric, we leverage the agglomerative clustering algorithm to group similar engines. We can easily convert the similarity score ss into a distance function for the clustering ($d = 1 - ss$). We choose this hierarchical clustering method because it is easy to visualize and we don't need to pre-define the number of clusters. We tried bin sizes S as 7, 14 and 28 and observed similar results. Below, we only present the result with $S = 7$.

5.1.2 Clustering Result Analysis

When running a hierarchical clustering algorithm, a threshold t_d needs to be specified. If the distance between two clusters is smaller than t_d, the two clusters will be merged. We visualize the clustering results with different t_d values using a dendrogram in Figure 18 in the Appendix. Intuitively, as we increase t_d, more clusters will be merged together. Figure 6 shows

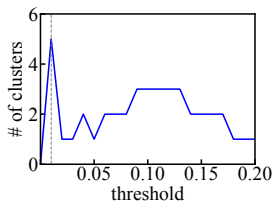


Figure 6: Number of clusters with more than one engine vs. threshold t_d. *The dash line* $t_d = 0.01$.

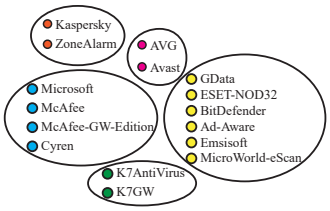


Figure 7: Clustering results with $t_d = 0.01$. *Only clusters with more than one engine are shown.*

the number of clusters as we increase t_d. Note that we only count the number of clusters that have more than one engine (singletons are ignored). When $t_d = 0.01$, we have the largest number of clusters (5 clusters), which are then visualized in Figure 7. The five clusters contain 16 engines. The rest 49 engines (not shown in the figure) could not form meaningful clusters under $t_d = 0.01$. This suggests the 16 engines are highly similar. Among the five clusters, one cluster has six engines (Cluster-I), one cluster has four engines (Cluster-II), and the other three clusters have two engines each.

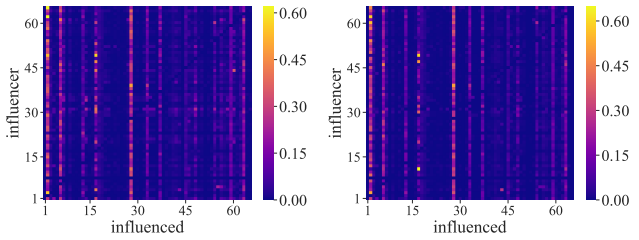
Cluster-I contains GData, ESET-NOD32, BitDefender, Ad-Aware, Emsisoft, and MicroWorld-eScan. We confirm that their label sequences are highly similar. For example, for each pair of the six engines, there are 14,147 files on average where the sequence similarity is higher than 0.99. Note that 14,147 files count for 98% of all the files in the main dataset. A similarity score of 0.99 means the label sequences are nearly identical. We show an example file and its label sequences from the five engines in the cluster[4] (Figure 16 in the Appendix). The flip patterns and timing are exactly the same. This result confirms that there exist groups of vendors whose labels are not independent but are highly synchronized.

Cluster-II contains Microsoft, McAfee-GW-Edition, McAfee, and Cyren. Among them, McAfee-GW-Edition and McAfee are from the same vendor (company), which could be the reason why their labels are highly similar. However, Microsoft and Cyren are operated by different companies from McAfee. For each pair of the four engines, there are 13,922 files on average with label-sequence similarity higher than 0.99. These 13,922 count for 97% of all files in the main dataset. We again show an example in the Appendix (Figure 19) where Microsoft and McAfee report identical label sequences for the file.

For the other three clusters, the engines are backed up by the same company. For example, ZoneAlarm uses Kaspersky's anti-virus engine [59], and AVG and Avast merged into one company in 2016 [68]. These three clusters confirm the effectiveness of our clustering algorithm.

As shown in Figure 6, when t_d is increased to 0.2, all the clusters (with more than one engine) are merged into one big cluster. This big cluster contains 28 engines and the rest 37 engines are not yet able to form a meaningful cluster. These 28

[4] ESET-NOD32 is different on this file.



(a) 0→1 (b) 1→0

Figure 8: Heatmaps for the active model. *All engines are sorted alphabetically. The value of each cell (i, j) indicates the influence from the engine at row-i to the engine at column-j.*

engines represent a group of engines that are highly correlated but have their differences. It is worth further analyzing their influences on each other's labels.

Observation 6: *There are groups of engines whose labeling decisions have strong correlations. These engines' results should not be treated independently.*

5.2 Influence Modeling

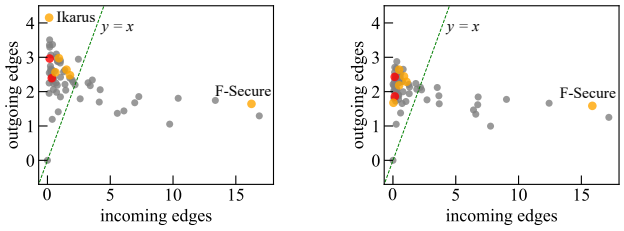
We further examine the potential causalities between the labels reported by two engines, using the *happens-before* relationship. We adapt popular social network influence models [21, 32, 33] to our problem contexts. More specifically, when engine j changes its label on a file to be the same as the label of engine i, we consider i's label is the causality of the label change made by j, or j is influenced by i.

There are two types of influence: active and passive influence. First, suppose vendor j flips its label on a file from "0" to "1" because vendor i also *made a 0→1 flip very recently*. We call such influence as active influence since i's action actively impact j's action. Second, suppose vendor j flips its label from "0" to "1" because vendor i has *stayed on label "1" over a period of time*. We call this relationship as passive influence since j is influenced by i's state, not action.

5.2.1 Active Influence Model

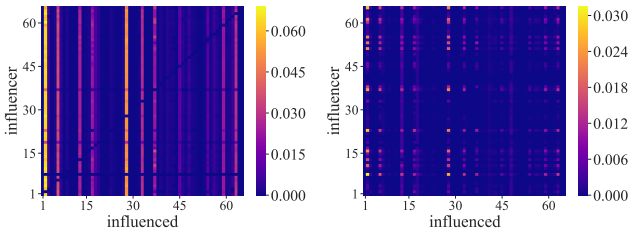
Active model is used to model the causalities between flips from different engines within a short period of time. Given a file f, if engine j flips its label at time t and engine i flips its label on the same direction but slightly earlier than t, we consider j's flip is influenced by i's flip. We set a time window w: an active influence event is established only when i's flip happens within $[t-w, t)$. For our analysis, we separate 0→1 flips from 1→0 flips, because they have different contextual meanings in malware detection. We use A_{i2j} to represent all active influence events from i to j across all files. A_i represents the total number of flips in engine i. The active influence score from i to j is measured as the probability $p_{i,j} = |A_{i2j}|/|A_i|$.

Engine Pair Analysis. We choose the window size $w = 7$ and compute the active influence score between each pair of



(a) 0→1 (b) 1→0

Figure 9: Active model: scatter plot of engines. *x: weighted sum of incoming edges, y: weighted sum of outgoing edges. reputable engines are in orange color, and reputable engines* are in red color.*



(a) 0→1 (b) 1→0

Figure 10: Heatmaps for the passive model. *All engines are sorted alphabetically. The value of each cell (i, j) indicates the influence from the engine at row-i to the engine at column-j.*

engines[5]. In Figure 8, we visualize the active influence score $p_{i,j}$ in a heatmap where i (influencer) is the row number and j (influenced) is the column number. The engines are ordered alphabetically. We observe that there are a number of vertical "bright" lines in both heat maps. This indicates that there are some engines that are easily influenced by all other engines. Examples include AegisLab in the 2nd column, Arcabit in the 6th column, Comodo in the 17th column, and F-Secure in the 28th column. Users have to carefully inspect labeling results from these engines before aggregation. We also observe that there are no clear horizontal lines, indicating that no engine can strongly influence all the other engines.

Active Influence Graph. To better understand the joint influence from all engines, we construct an active influence graph. In this *directed* graph, engines are nodes, and the directed edges are weighted by the influence score between the two engines. For each node, we sum its outgoing edges' weights as an indicator of its active influence to other nodes.

We visualize one influence graph in Figure 17 in the Appendix. AegisLab, F-Secure, Comodo, and Arcabit are mostly influenced by other engines, but their main influencer sets are not identical (with some overlaps). For example, only Comodo is highly influenced by Cylance and Qihoo-360; only AegisLab is highly influenced by Microsoft.

To understand the influence of each engine, we plot Figure 9, where x and y represent the weighted sum of incoming and outgoing edges. We use yellow color to mark the reputable engines mentioned in previous literature (Table 5) and

[5]Window sizes 14 and 28 generate similar results.

use the red color to mark the two highly reputable engines (Kaspersky and Symantec). For 0→1 flips (Figure 9(a)), there are 49 engines above the $x = y$ line and 16 engines below the line. One high-reputation engine (Ikarus) has the largest outgoing edge weight, indicating that it has the biggest active influence. Interestingly, one high-reputation engine (F-Secure) is more easily influenced compared to other reputable engines. For 1→0 flips (Figure 9(b)), the patterns are very similar.

Observation 7: *Active influence widely exists between VirusTotal engines. There are engines that are highly influenced by many other engines at both flip directions.*

5.2.2 Passive Model

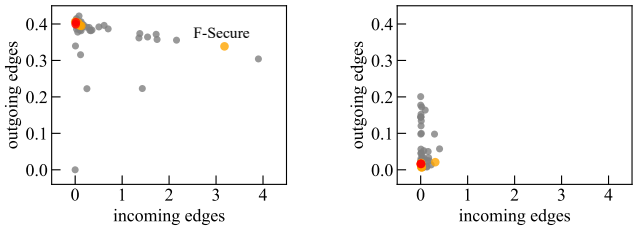
While active influence model captures highly synchronized actions (*e.g.*, j flips right after i's flip), passive influence provides a different angle by showing j flips towards i's current state. Given a file, if engine j flips its label to l (*i.e.*, "0" or "1") at time t, and engine i has already stayed on label l for w days at t, we consider there is a passive influence from i to j. Note that w represents the size of the time window within which i keeps a stable label to influence j. For this analysis, we use $w = 7$ as the window size[6]. We use A_{i2j} to represent the number of passive influence events from i to j and use A_i to represent the number of subsequences with 7 consecutive label l. For example, if i's sequence is "11111111100011111" and l="1", then $|A_i| = 4$. We compute the passive influence score from i to j as $p_{i,j} = |A_{i2j}|/|A_i|$.

Engine Pair Analysis. We again use heatmaps to visualize passive influence scores in Figure 10. Interestingly, the 1→0 flip heatmap looks different from that of 0→1 flip. Figure 10(a) shows engines that are highly influenced by all other engines under the active model are still highly influenced under the passive model (the red vertical lines). However, the result of 1→0 (Figure 10(b)) becomes less obvious under passive influence and there is no vertical line or horizontal line in red. Combined with the active model's result, it shows that engines flipping from "malicious" to "benign" are more likely influenced by other engines making the same flip recently, rather than engines that always stick to the "benign" label.

Passive Influence Graph. Figure 11 is the scatter plot for passive model, where x and y represent the weighted sum of incoming and outgoing edges. For 0→1 flips (Figure 11(a)), all high-reputation engines are around the top left corner, indicating a strong influence to other engines. The exception is again F-Secure, which is more likely to be influenced by others. For 1→0 flips (Figure 11(b)), the high-reputation engines do not have a strong passive influence to others.

Observation 8: *The passive influence is weak in general. The passive influence is relatively stronger when a benign label is flipped to malicious.*

[6]Changing the window size to 14 or 28 returns similar conclusions.



(a) 0→1 (b) 1→0

Figure 11: Passive Model: scatter plot of engines. *x: weighted sum of incoming edges, y: weighted sum of outgoing edges. reputable engines are in orange color, and reputable engines* are in red color.*

6 Analyzing the Ground-Truth Dataset

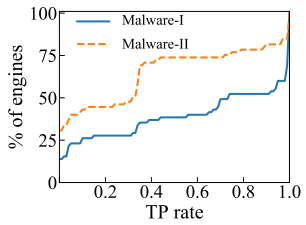
So far, we use the *main dataset* to understand the temporal dynamics of labels and the relationship between engines. While the analysis is benefited from the file diversity and longitudinal data, it says little about the "correctness" of engines. In this section, we use the smaller ground-truth sets to examine the threshold choices for label aggregation and quantify the different detection capabilities of engines.

6.1 Individual Engine Accuracy

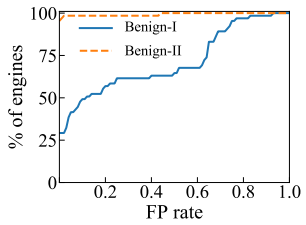
We start by looking at how well individual engines classify malware and benignware. Recall that we have four ground-truth sets, where Malware-I and Malware-II are generated by obfuscating real-world malware, Benign-I is generated by obfuscating goodware, and Benign-II is generated by recompiling goodware. All the files have a clean signature (means they are never scanned by VirusTotal before, no prior history). Note that the ground-truth sets are related to ransomware — they allow us to examine questions about VirusTotal's detection correctness, but the results should mostly reflect the engines' capability of analyzing ransomware.

In Figure 12, we compare the detection result on the first-day of VirusTotal's scanning and the last day of the scanning. Note that, for malware sets, we show the true positive (TP) rate (*i.e*, ratio of correctly identified malware). For benign sets, we show the false positive (FP) rate (*i.e.*, ratio of misclassified benignware). A high TP rate and a low FP rate mean the engine is more accurate. For malware sets, we show that the engines on the last day (Figure 12(c)) are indeed more accurate than that on the first day (Figure 12(a)). In particular, on the last day, 75.4% of the engines have a TP rate of nearly 100% on Malware-I dataset. Files in Malware-II are harder to detect — only 24.6% of the engines have a near 100% TP rate. Overall, the detection capability of different engines varies significantly. A large portion (20%–50%) of engines only detects less than 50% of malware files.

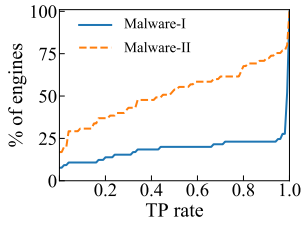
Figure 12(b) and Figure 12(d) show that performance on *benign files* is more stable when comparing the first day and the last day. Benign-II (files are not obfuscated) has almost no false positive. However, engines produce a high FP rate



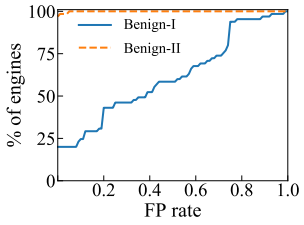
(a) Malware sets on the first day



(b) Benign sets on the first day



(c) Malware sets on the last day



(d) Benign sets on the last day

Figure 12: CDF plots of the true positive (TP) rate and false positive (FP) rate of each engine.

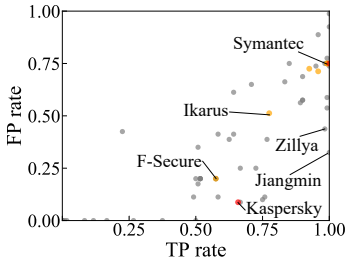


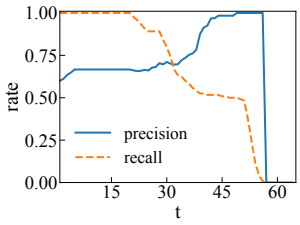
Figure 13: Engines' TP rate and FP rate when analyzing Malware-I, Malware-II and Benign-I.

on Benign-I where obfuscation is applied. Only about 25% of the engines have zero false positive on Benign-I. These engines either have actually analyzed the files or report most files as benign (*i.e.*, having zero true positive when analyzing the two malware sets).

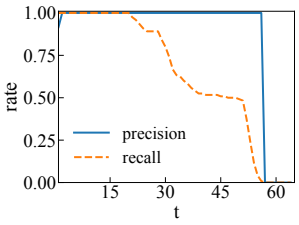
About 75% of the engines have false positives on Benign-I. A possible explanation is that those engines use "obfuscation" as a feature for their malware detection without carefully analyzing the files. About 25% of engines have a high FP rate (>70%), indicating that they have put a heavy weight on the "obfuscation" feature.

To understand which engines are easily misled by obfuscation, we present Figure 13. We combine Malware-I, Malware-II and Benign-I to calculate the FP rate and TP rate for each engine (on the last day). All three datasets are obfuscated — if an engine has a high TP rate and a low FP rate on these datasets, it means this engine has truly analyzed the files' behavior rather than relying on "obfuscation" to make a decision. In practice, benign files may also use obfuscation to prevent copyright infringements and reverse-engineering.

In Figure 13, engines at the bottom left corner has a near-zero FP rate, but also cannot detect most of the malware (low

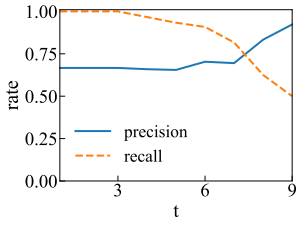


(a) Malware sets and Benign-I

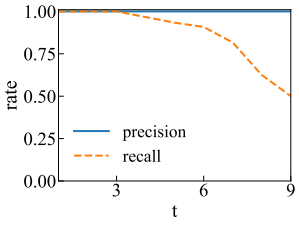


(b) Malware sets and Benign-II

Figure 14: Precision and recall for different threshold t when using all engines.



(a) Malware sets and Benign-I



(b) Malware sets and Benign-II

Figure 15: Precision and recall for different threshold t when only using reputable engines.

TP rate). Engines at the top right corner are likely to heavily rely on obfuscation as feature — they detect most of the malware and also mis-classify most obfuscated benignware. Interestingly, a number of high-reputation engines belong to this group (*e.g.*, AVG, McAfee, Microsoft, Sophos, Symantec). The most effective engines are located at the bottom right corner. They manage to maintain a relatively low FP rate, and still detect most malware (*e.g.*, Jiangmin, Zillya, Kaspersky).

Observation 9: *Obfuscation in benign files can easily cause false positives to VirusTotal engines. High-reputation engines are not necessarily better at handling obfuscations.*

6.2 Aggregating Engines' Labels

While individual engines have an uneven performance, it is possible label aggregation could help to improve the overall accuracy. Here we test two different aggregation methods. First, we use a simple threshold t to determine if a file has a malicious label. If t or more engines give a malicious label, then we mark the file as malicious. The results are shown in Figure 14. Second, we only consider the nine high-reputation engines mentioned in previous works and apply the same threshold method. The results are in Figure 15. We test two different settings: "all malware + Benign-I" and "all malware + Benign-II" to isolate the impact of benign file obfuscation.

When we consider obfuscated benign files, as shown in Figure 14(a), it is very difficult to get the precision high without significantly scarifying recall. There is only a small range of t (between t =45 and t =55) where we can detect half of the malware with a 95%+ precision. The situation is not necessarily better when we only consider high-reputation engines.

As shown in Figure 15(a), it is almost equally difficult to get an over 90% precision without scarifying 50% of recall.

When we consider non-obfuscated benign files (Figure 14(b)), it is clear that using a small threshold t (between 2–15) is a good choice. However, when we consider high-reputation engines only (Figure 15(b)), it is better to stick to an even smaller threshold (*e.g.*, $t < 3$). If we require all nine high-reputation engines to vote "malicious", then we will again lose 50% of recall.

Observation 10: *A small threshold value can balance the precision and the recall as long as the benign files are not obfuscated.*

6.3 Comparing with Desktop Engines

A subset of engines also provide the desktop version of the anti-malware engines. A prior work [57] shows VirusTotal often runs the stripped-down version of the engines, and thus is more likely to miss true malicious instances. Note that this conclusion is drawn from *URL scanning* for phishing website detection [57]. Below, we explore if the same conclusion applies to *malware scanning*.

Experiment Setup. Out of all VirusTotal vendors [3], we find 36 vendors also offer a desktop version of their product (the list of engines is shown in Table 5 in the Appendix). We install them separately on 36 Windows-10 virtual machines (version 1809). We validated that our obfuscated malicious samples still have their malicious actions in the VM environment. For the four ground-truth datasets, we scan their files four times (in four different days). For each time of the experiment, we use fresh virtual machines and install the latest versions of the desktop engines. We *disconnect* the Internet while the engines scan the files, to prevent the engines from uploading the files or reporting the results to their servers. This allows us to *isolate* the analysis engine (*e.g.*, classifier) on the desktop from the engine in the cloud (on VirusTotal) to compare them fairly. It's possible some desktop engines do not run the analysis locally but solely rely on its remote server for analysis. To this end, we run the main experiment without the Internet, and later run a validation test with the Internet.

Comparison Results (w/o Internet). All 36 engines have some inconsistency between the desktop and VirusTotal versions. For each engine and each dataset, we calculate the *inconsistency rate*, which is the number of files with different detection results (between VirusTotal and desktop scans) divided by the total number of files. We report the average inconsistency rate over different experiment dates for the engine.

All 36 engines have a non-zero inconsistency rate on malware datasets. The average inconsistency rate is 25.4% on Malware-I and 29.2% on Malware-II. Some engines have an inconsistency rate over 90% on Malware-I (98.6% on ZoneAlarm, 90.3% on Tencent and 98.9% on Qihoo-360) because their VirusTotal engines could detect most samples, but their desktop engines didn't report any of them. The inconsistency rates on the benign datasets are lower (23.4% on Benign-I and 0% on Benign-II).

To examine which version is more accurate, we compare precision, recall, and the F1-score over four datasets for each engine. F1-score is the harmonic mean of precision and recall. For precision, 26 engines (out of 36) have a higher average precision on their desktop versions than VirusTotal across datasets. 25 engines have a higher average recall on VirusTotal than their desktop versions. After computing F1-score to merge precision and recall together, we find that 24 engines (out of 36) have higher average F1-Scores on VirusTotal than their desktop versions (20.2% higher on average). Overall, the result shows the online engines at VirusTotal are more aggressive and tend to cover more malware, while desktop versions are more conservative to keep a small number of false alarms. Our result is different from that of the URL scanning reported in [57] (where vendors' URL engines at VirusTotal cover fewer phishing websites than their standalone versions).

Sanity Check (w/ Internet). We performed a sanity check by running the experiments again with VMs connecting to the Internet. We compare the results to those when VMs were disconnected to the Internet on the same day. In total, 23 out of 36 engines' results remain consistent, with or without Internet. 13 out of 36 engines have different results, indicating that the remote servers play a role in desktop engines' decision. Among the 13 engines, seven engines have a lower precision after connecting to the Internet; nine engines have a higher recall. Overall, the results of desktop engines are getting closer to those of VirusTotal engines, but the conclusion above is still valid: desktop engines are still more conservative with a higher precision and lower recall. The gap is smaller with the Internet connection.

Observation 11: *Inconsistency exists between the desktop version and the online version (at VirusTotal) for all engines. Surprisingly, for most of the vendors, their VirusTotal engines are able to detect more malware than their desktop versions.*

6.4 Comparison with the Main Dataset

As a sanity check, we have validated the key observations we had on the main dataset using the ground-truth datasets too. Due to space limit, we keep our discussions brief. First, ground-truth datasets have more hazard flips (6,708) than non-hazard flips (5,855). Second, flips also widely exist across files, dates, and engines. The majority of the flips are still highly correlated with engines' model update (73.7%). Third, engines that are highly correlated in the main dataset are still highly correlated in the ground-truth dataset. The strong influencer-influenced relationships observed in the main dataset are also observed in the ground-truth datasets

(primarily in Malware-I and Malware-II).

7 Discussion

Our measurement results have several important implications regarding the methods of using VirusTotal for file labeling.

Data Preprocessing. Our results show that hazard flips count for the majority of all the label flips and they affect label stabilization of individual engines. The good news is that hazard flips, by definition, are short-lived, and it only incurs a small cost to get rid of them. We recommend VirusTotal users to submit the same files to VirusTotal in three consecutive days to identify and remove potential hazards.

Label flips happen widely across engines, files and time. They do not necessarily disappear if researchers wait for a longer time. The benefit of querying the labels over a long period of time (*e.g.*, months) is quite limited.

Label Aggregation. We show that threshold-based aggregation is surprisingly effective in stabilizing the *aggregated labels* against the label flips of individual engines, but the threshold t needs to be set properly. For example, the aggregated labels are still easily influenced by the flips when the threshold is too small ($t = 1$) or too big ($t = 40$ or $t = 50$). If the threshold t is chosen within a reasonable range (2–39), the aggregated labels are more likely to stay stable.

A stable aggregated label does not necessarily mean the label is correct. Our ground-truth analysis shows that choosing a small threshold (*e.g.*, $t < 15$) helps strike a good balance between precision and recall for the aggregated labels. However, it becomes very difficult to find a good threshold when the benign files contain obfuscated code. Our recommendation is that researchers should not use a small threshold if their files are obfuscated (especially the potentially benign ones). A better idea could be only considering engines that perform well on obfuscated files (see Figure 13).

Engine Independence. Most existing papers treat all engines equally and do not consider possible correlations of their labeling decisions. Our experiments confirm the existence of both correlation and causality relationships between engines. In particular, we identify groups of engines whose label sequences are highly similar to each other (Section 5.1). A practical suggestion is to consider them as "redundant votes" and reduce their weights during label aggregation. We also identified several engines whose labeling exhibits causal relationships (Section 5.2). This does not necessarily mean one engine directly copies results from other engines – it is also possible these engines change labels due to the impact of third parties (blacklists), but some engines react slower than others.

High-reputation Engines. Several existing papers hand-picked high-reputation engines for label aggregation. Our analysis shows that most of these engines perform well (*e.g.*, having more stabilized labels, being an influencer instead of being influenced). However, we find one high-reputation engine (F-Secure) constantly acting as an outlier. It is easily influenced by other engines, and its label accuracy is subpar. We notice that high-reputation engines are not always more accurate. Four of them are not good at handling obfuscated benign files, producing many false positives (*e.g.*, Symantec).

Limitations & APK Experiments. As discussed in Section 3, a key limitation is that our datasets are not diverse enough (*e.g.*, the main dataset only has PE files, the ground-truth datasets are focused on ransomware). We defer in-depth analysis on other file types to future work. Here, we briefly run a quick measurement on Android APK files (another popular file type for malware) to cross-examine the results.

The methodology is similar to our main experiment on PE files. We selected 2,071 fresh APK samples (with no prior history at VirusTotal), and collected their daily labels from December 26, 2019 to February 09, 2020 (46 days). About half of the files are labeled as "benign" (1,144) by all engines on *day-1*, and the other half (927) are labeled as "malicious" by at least one engine. 59 engines have returned their labels and we collected in total 5,303,106 data points.

We find the major observations on PE files still hold for APK files, with some differences. First, there are 16,453 flips, including 9,984 hazard flips. Among the APK files that have no flip (1,264 files, 60% of all files), the vast majority of them have been labeled as benign by all engines for the entire measurement period. This is similar to what we observed on PE files. Second, the top three engines with most flips are Microsoft (41%), Fortinet (15%), and Tencent (10%). The engine ranking is different from that of PE files, possibly due to the different specialties of engines. Third, in terms of engine label correlations, we also identified tightly clustered engines for APK files. For example, GData, BitDefender, Ad-Aware, Emsisoft, and MicroWorld-eScan are still clustered together. The cluster is largely similar to that under PE files, and the only difference is ESET-NOD32 is no longer in the cluster. Finally, interestingly, engines that were highly-influenced under PE files (*e.g.*, F-Secure, Comodo, AegisLab, Arcabit) now become "influencers" under APK files. Overall, the takeaway is that engines face common problems such as label instability, and they have their own specialties for different malware types.

Malware Coverage Issues. VirusTotal is arguably the biggest public malware database that researchers can access *for free*. Even so, its malware coverage still has limitations [38, 48]. Beyond file label stability and accuracy (the focus of our paper), another challenge is to further improve the malware coverage of VirusTotal's database. In some way, VirusTotal is already trying to improve its coverage by providing free malware scanning services to the public to gather new malware samples from users, companies, and other security vendors. A recent report shows that VirusTotal receives over one million file submissions every day [65]. Future work could look into new incentive mechanisms to encourage the broader sharing of malware intelligence.

Data Sharing. To benefit future researchers and practitioners, we have released the raw data collected in this paper (timestamped file labels) and a number of ranked lists. The engines can be ranked based on different criteria, such as the number of (hazard) flips, influence scores under different influence models, and label accuracy. We have attached the ranking method and the data with each ranked list. Our released data is available at https://sfzhu93.github.io/projects/vt/index.html.

8 Conclusions

In this paper, we surveyed 115 research publications that use VirusTotal for data annotation. Then we took a data-driven approach to reason and validate their data labeling methodologies. We collected a dataset of 14,000 files' timestamped labels over a year from 65 anti-malware engines at VirusTotal. We validated the benefits of threshold-based labeling methods in tolerating temporary label flips. We also pointed out the questionable approaches such as hand-picking trusted engines, and ignoring the strong correlations among engines. Future work will focus on extending the experiments to other file types, using more diverse ground-truth datasets, and developing new label aggregation methods.

Acknowledgement

We would like to thank our shepherd Gianluca Stringhini and the anonymous reviewers for their helpful feedback; Xiao Zhang at Palo Alto Networks, Inc. for sharing malware hashes; Yiying Zhang for her valuable comments on the initial version of this paper. Limin Yang and Gang Wang were supported by NSF grants CNS-1750101 and CNS-1717028.

References

[1] Binutils - GNU Project - Free Software Foundation. https://www.gnu.org/software/binutils/.

[2] C# Websocket Implementation. https://github.com/statianzo/Fleck.

[3] Contributors - VirusTotal. https://support.virustotal.com/hc/en-us/articles/115002146809-Contributors.

[4] Coreutils - GNU core utilities. https://www.gnu.org/software/coreutils/.

[5] Cygwin. https://cygwin.com/.

[6] Mono. https://www.mono-project.com/.

[7] Notepad++ official repository. https://github.com/notepad-plus-plus/notepad-plus-plus.

[8] Oreans technology : Software security defined. https://oreans.com/codevirtualizer.php.

[9] Oreans technology : Software security defined. https://oreans.com/themida.php.

[10] Statistics - VirusTotal. https://www.virustotal.com/en/statistics/.

[11] VirusTotal. https://www.virustotal.com/.

[12] VirusTotal, Chronicle and Google Cloud. https://blog.virustotal.com/2019/06/virustotal-chronicle-and-google-cloud.html.

[13] Randy Abrams. VirusTotal Tips, Tricks and Myths. https://www.virusbulletin.com/uploads/pdf/magazine/2017/VB2017-Abrams.pdf.

[14] Kevin Allix, Quentin Jérôme, Tegawendé F. Bissyandé, Jacques Klein, Radu State, and Yves Le Traon. A forensic analysis of android malware – how is malware written and how it could be detected? In *COMPSAC*, 2014.

[15] Daniel Arp, Michael Spreitzenbarth, Malte Hubner, Hugo Gascon, Konrad Rieck, and CERT Siemens. Drebin: Effective and explainable detection of android malware in your pocket. In *NDSS*, 2014.

[16] Ulrich Bayer, Paolo Milani Comparetti, Clemens Hlauschek, Christopher Kruegel, and Engin Kirda. Scalable, behavior-based malware clustering. In *NDSS*, 2009.

[17] Stephen D. Brown and Zvonko G. Vranesic. *Fundamentals of digital logic with VHDL design*. 2009.

[18] Zhenquan Cai and Roland H.C. Yap. Inferring the detection logic and evaluating the effectiveness of android anti-virus apps. In *CODASPY*, 2016.

[19] Margaux Canet, Amrit Kumar, Cédric Lauradoux, Mary-Andréa Rakotomanga, and Reihaneh Safavi-Naini. Decompression quines and anti-viruses. In *CODASPY*, 2017.

[20] Curtis Carmony, Xunchao Hu, Heng Yin, Abhishek Vasisht Bhaskar, and Mu Zhang. Extract me if you can: Abusing pdf parsers in malware detectors. In *NDSS*, 2016.

[21] Meeyoung Cha, Hamed Haddadi, Fabricio Benevenuto, and Krishna P Gummadi. Measuring user influence in twitter: The million follower fallacy. In *AAAI*, 2010.

[22] Mahinthan Chandramohan, Hee Beng Kuan Tan, and Lwin Khin Shar. Scalable malware clustering through coarse-grained behavior modeling. In *FSE*, 2012.

[23] Kai Chen, Peng Wang, Yeonjoon Lee, XiaoFeng Wang, Nan Zhang, Heqing Huang, Wei Zou, and Peng Liu. Finding unknown malice in 10 seconds: Mass vetting for new threats at the google-play scale. In *USENIX Security*, 2015.

[24] Binlin Cheng, Jiang Ming, Jianmin Fu, Guojun Peng, Ting Chen, Xiaosong Zhang, and Jean-Yves Marion. Towards paving the way for large-scale windows malware analysis: generic binary unpacking with orders-of-magnitude performance boost. In *CCS*, 2018.

[25] Melissa Chua and Vivek Balachandran. Effectiveness of android obfuscation on evading anti-malware. In *CODASPY*, 2018.

[26] Christian Collberg, GR Myles, and Andrew Huntwork. Sandmark-a tool for software protection research. *IEEE S & P*, 2003.

[27] Fady Copty, Matan Danos, Orit Edelstein, Cindy Eisner, Dov Murik, and Benjamin Zeltser. Accurate malware detection by extreme abstraction. In *ACSAC*, 2018.

[28] Wayne W Daniel. *Applied nonparametric statistics*. PWS-Kent, 2nd edition, 1990.

[29] Yue Duan, Mu Zhang, Abhishek Vasisht Bhaskar, Heng Yin, Xiaorui Pan, Tongxin Li, Xueqiang Wang, and X Wang. Things you may not know about android (un) packers: a systematic study based on whole-system emulation. In *NDSS*, 2018.

[30] Yu Feng, Saswat Anand, Isil Dillig, and Alex Aiken. Apposcopy: Semantics-based detection of android malware through static analysis. In *FSE*, 2014.

[31] Sean Ford, Marco Cova, Christopher Kruegel, and Giovanni Vigna. Analyzing and detecting malicious flash advertisements. In *ACSAC*, 2009.

[32] Manuel Gomez-Rodriguez, David Balduzzi, and Bernhard Schölkopf. Uncovering the temporal dynamics of diffusion networks. In *ICML*, 2011.

[33] Amit Goyal, Francesco Bonchi, and Laks V.S. Lakshmanan. Learning influence probabilities in social networks. In *WSDM*, 2010.

[34] Mariano Graziano, Davide Canali, Leyla Bilge, Andrea Lanzi, and Davide Balzarotti. Needles in a haystack: Mining information from public dynamic analysis sandboxes for malware intelligence. In *USENIX Security*, 2015.

[35] Mahmoud Hammad, Joshua Garcia, and Sam Malek. A large-scale empirical study on the effects of code obfuscations on android apps and anti-malware products. In *ICSE*, 2018.

[36] Heqing Huang, Cong Zheng, Junyuan Zeng, Wu Zhou, Sencun Zhu, Peng Liu, Suresh Chari, and Ce Zhang. Android malware development on public malware scanning platforms: A large-scale data-driven study. In *BigData*, 2016.

[37] Médéric Hurier, Guillermo Suarez-Tangil, Santanu Kumar Dash, Tegawendé F Bissyandé, Yves Le Traon, Jacques Klein, and Lorenzo Cavallaro. Euphony: Harmonious unification of cacophonous anti-virus vendor labels for android malware. In *MSR*, 2017.

[38] Colin C Ife, Yun Shen, Steven J Murdoch, and Gianluca Stringhini. Waves of malice: A longitudinal measurement of the malicious file delivery ecosystem on the web. In *AsiaCCS*, 2019.

[39] Luca Invernizzi, Stanislav Miskovic, Ruben Torres, Christopher Kruegel, Sabyasachi Saha, Giovanni Vigna, Sung-Ju Lee, and Marco Mellia. Nazca: Detecting malware distribution in large-scale networks. In *NDSS*, 2014.

[40] Alex Kantchelian, Michael Carl Tschantz, Sadia Afroz, Brad Miller, Vaishaal Shankar, Rekha Bachwani, Anthony D. Joseph, and J. D. Tygar. Better malware ground truth: Techniques for weighting anti-virus vendor labels. In *AISec*, 2015.

[41] Amin Kharraz, Sajjad Arshad, Collin Mulliner, William Robertson, and Engin Kirda. Unveil: A large-scale, automated approach to detecting ransomware. In *USENIX Security*, 2016.

[42] Doowon Kim, Bum Jun Kwon, and Tudor Dumitraş. Certified malware: Measuring breaches of trust in the windows code-signing pki. In *CCS*, 2017.

[43] Doowon Kim, Bum Jun Kwon, Kristián Kozák, Christopher Gates, and Tudor Dumitraş. The broken shield: Measuring revocation effectiveness in the windows code-signing pki. In *USENIX Security*, 2018.

[44] Eugene Kolodenker, William Koch, Gianluca Stringhini, and Manuel Egele. Paybreak: defense against cryptographic ransomware. In *AsiaCCS*, 2017.

[45] Deguang Kong and Guanhua Yan. Discriminant malware distance learning on structural information for automated malware classification. In *KDD*, 2013.

[46] David Korczynski and Heng Yin. Capturing malware propagations with code injections and code-reuse attacks. In *CCS*, 2017.

[47] Platon Kotzias, Leyla Bilge, and Juan Caballero. Measuring PUP prevalence and PUP distribution through pay-per-install services. In *USENIX Security*, 2016.

[48] Bum Jun Kwon, Jayanta Mondal, Jiyong Jang, Leyla Bilge, and Tudor Dumitraş. The dropper effect: Insights into malware distribution with downloader graph analytics. In *CCS*, 2015.

[49] Stevens Le Blond, Adina Uritesc, Cédric Gilbert, Zheng Leong Chua, Prateek Saxena, and Engin Kirda. A look at targeted attacks through the lense of an ngo. In *USENIX Security*, 2014.

[50] Bo Li, Phani Vadrevu, Kyu Hyung Lee, and Roberto Perdisci. Jsgraph: Enabling reconstruction of web attacks via efficient tracking of live in-browser javascript executions. In *NDSS*, 2018.

[51] Li Li, Tegawendé F. Bissyandé, Damien Octeau, and Jacques Klein. Reflection-aware static analysis of android apps. In *ASE*, 2016.

[52] Gen Lu and Saumya Debray. Weaknesses in defenses against web-borne malware. In *DIMVA*, 2013.

[53] Long Lu, Vinod Yegneswaran, Phillip Porras, and Wenke Lee. Blade: An attack-agnostic approach for preventing drive-by malware infections. In *CCS*, 2010.

[54] Guozhu Meng, Yinxing Xue, Chandramohan Mahinthan, Annamalai Narayanan, Yang Liu, Jie Zhang, and Tieming Chen. Mystique: Evolving android malware for auditing anti-malware tools. In *AsiaCCS*, 2016.

[55] Aziz Mohaisen and Omar Alrawi. Av-meter: An evaluation of antivirus scans and labels. In *DIMVA*, 2014.

[56] Nick Nikiforakis, Steven Van Acker, Wannes Meert, Lieven Desmet, Frank Piessens, and Wouter Joosen. Bitsquatting: Exploiting bit-flips for fun, or profit? In *WWW*, 2013.

[57] Peng Peng, Limin Yang, Linhai Song, and Gang Wang. Opening the blackbox of virustotal: Analyzing online phishing scan engines. In *IMC*, 2019.

[58] Moheeb Abu Rajab, Lucas Ballard, Noe Lutz, Panayiotis Mavrommatis, and Niels Provos. Camp: Content-agnostic malware protection. In *NDSS*, 2013.

[59] Neil Rubenking. Check Point ZoneAlarm Free Antivirus+. https://www.pcmag.com/review/322439/check-point-zonealarm-free-antivirus-2017.

[60] Armin Sarabi and Mingyan Liu. Characterizing the internet host population using deep learning: A universal and lightweight numerical embedding. In *IMC*, 2018.

[61] Sebastian Schrittwieser, Stefan Katzenbeisser, Johannes Kinder, Georg Merzdovnik, and Edgar Weippl. Protecting software through obfuscation: Can it keep pace with progress in code analysis? *ACM Computing Surveys (CSUR)*, 2016.

[62] Edward J. Schwartz, Cory F. Cohen, Michael Duggan, Jeffrey Gennari, Jeffrey S. Havrilla, and Charles Hines. Using logic programming to recover c++ classes and methods from compiled executables. In *CCS*, 2018.

[63] Marcos Sebastián, Richard Rivera, Platon Kotzias, and Juan Caballero. Avclass: A tool for massive malware labeling. In *RAID*, 2016.

[64] Mahmood Sharif, Jumpei Urakawa, Nicolas Christin, Ayumu Kubota, and Akira Yamada. Predicting impending exposure to malicious content from user behavior. In *CCS*, 2018.

[65] Linhai Song, Heqing Huang, Wu Zhou, Wenfei Wu, and Yiying Zhang. Learning from big malwares. In *APSys*, 2016.

[66] Michael Spreitzenbarth, Felix Freiling, Florian Echtler, Thomas Schreck, and Johannes Hoffmann. Mobile-sandbox: Having a deeper look into android applications. In *SAC*, 2013.

[67] Nedim Srndic and Pavel Laskov. Detection of malicious pdf files based on hierarchical document structure. In *NDSS*, 2013.

[68] Vince Steckler. Avast and AVG become one. https://blog.avast.com/avast-and-avg-become-one.

[69] Gianluca Stringhini, Oliver Hohlfeld, Christopher Kruegel, and Giovanni Vigna. The harvester, the botmaster, and the spammer: On the relations between the different actors in the spam landscape. In *AsiaCCS*, 2014.

[70] Bo Sun, Akinori Fujino, and Tatsuya Mori. Poster: Toward automating the generation of malware analysis reports using the sandbox logs. In *CCS*, 2016.

[71] Kurt Thomas, Elie Bursztein, Chris Grier, Grant Ho, Nav Jagpal, Alexandros Kapravelos, Damon Mccoy, Antonio Nappa, Vern Paxson, Paul Pearce, Niels Provos, and Moheeb Abu Rajab. Ad injection at scale: Assessing deceptive advertisement modifications. In *S&P*, 2015.

[72] Kurt Thomas, Juan A Elices Crespo, Ryan Rasti, Jean-Michel Picod, Cait Phillips, Marc-André Decoste, Chris Sharp, Fabio Tirelo, Ali Tofigh, Marc-Antoine Courteau, et al. Investigating commercial pay-per-install and the distribution of unwanted software. In *USENIX Security*, 2016.

[73] Ke Tian, Steve T. K. Jan, Hang Hu, Danfeng Yao, and Gang Wang. Needle in a haystack: Tracking down elite phishing domains in the wild. In *IMC*, 2018.

[74] Zacharias Tzermias, Giorgos Sykiotakis, Michalis Polychronakis, and Evangelos P. Markatos. Combining static and dynamic analysis for the detection of malicious documents. In *EuroSec*, 2011.

[75] Haoyu Wang, Zhe Liu, Jingyue Liang, Narseo Vallina-Rodriguez, Yao Guo, Li Li, Juan Tapiador, Jingcun Cao, and Guoai Xu. Beyond google play: A large-scale comparative study of chinese android app markets. In *IMC*, 2018.

[76] Liang Wang, Antonio Nappa, Juan Caballero, Thomas Ristenpart, and Aditya Akella. Whowas: A platform for measuring web deployments on iaas clouds. In *IMC*, 2014.

[77] Pei Wang, Qinkun Bao, Li Wang, Shuai Wang, Zhaofeng Chen, Tao Wei, and Dinghao Wu. Software protection on the go: A large-scale empirical study on mobile app obfuscation. In *ICSE*, 2018.

[78] Michelle Y. Wong and David Lie. Tackling runtime-based obfuscation in android with tiro. In *USENIX Security*, 2018.

[79] Christian Wressnegger and Konrad Rieck. Looking back on three years of flash-based malware. In *EuroSec*, 2017.

[80] Mingyuan Xia, Lu Gong, Yuanhao Lyu, Zhengwei Qi, and Xue Liu. Effective real-time android application auditing. In *S&P*, 2015.

[81] Ke Xu, Yingjiu Li, Robert H. Deng, and Kai Chen. Deeprefiner: Multi-layer android malware detection system applying deep neural networks. In *EuroS&P*, 2018.

[82] Zhaoyan Xu, Antonio Nappa, Robert Baykov, Guangliang Yang, Juan Caballero, and Guofei Gu. Autoprobe: Towards automatic active malicious server probing using dynamic binary analysis. In *CCS*, 2014.

[83] Yinxing Xue, Junjie Wang, Yang Liu, Hao Xiao, Jun Sun, and Mahinthan Chandramohan. Detection and classification of malicious javascript via attack behavior modelling. In *ISSTA*, 2015.

[84] Wei Yang, Deguang Kong, Tao Xie, and Carl A. Gunter. Malware detection in adversarial settings: Exploiting feature evolutions and confusions in android apps. In *ACSAC*, 2017.

[85] Mu Zhang, Yue Duan, Heng Yin, and Zhiruo Zhao. Semantics-aware android malware classification using weighted contextual api dependency graphs. In *CCS*, 2014.

[86] Yibing Zhongyang, Zhi Xin, Bing Mao, and Li Xie. Droidalarm: An all-sided static analysis tool for android privilege-escalation malware. In *AsiaCCS*, 2013.

[87] Ziyun Zhu and Tudor Dumitras. Chainsmith: Automatically learning the semantics of malicious campaigns by mining threat intelligence reports. In *EuroS&P*, 2018.

Appendix

Table 3: 9 high-reputation engines and the papers that mentioned them. *Kaspersky and Symantec are mentioned in two papers.*

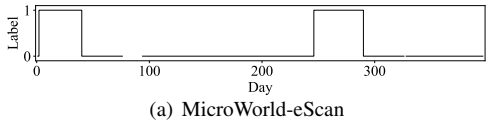
Kaspersky [22, 71], Symantec [22, 45] AVG [71],
F-Secure [22], Ikarus [22], McAfee [45],
Microsoft [45], ESET-NOD32 [45], Sophos [71],

Table 4: Seed ransomware files and number of engines detected each file on June 1, 2019 (out of 65 engines)

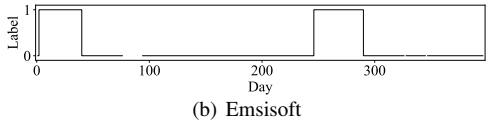
MD5	# Engines
40c5113e35dd653ca1fc1524d51da408	56
5dc58c702f21ba786e3a51eb8c37bd14	56
8b6bc16fd137c09a08b02bbe1bb7d670	52
bbd4c2d2c72648c8f871b36261be23fd	49

Table 5: Engines with both VirusTotal and desktop versions.

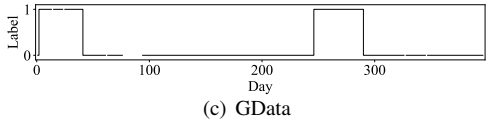
Ad-Aware, Avast, AVG, Avira, BitDefender, CAT-QuickHeal, ClamAV, CMC,Comodo, ESET-NOD32, F-Prot, F-Secure, GData, Ikarus, Jiangmin, K7AntiVirus, Kaspersky, Kingsoft, Malwarebytes, McAfee, Microsoft (Windows Defender), MicroWorld-eScan, NANO-Antivirus, Panda, Qihoo-360, Rising, SUPERAntiSpyware, Symantec (Norton), TACHYON, Tencent, TotalDefense, TrendMicro, Vipre, Webroot, Zillya, ZoneAlarm



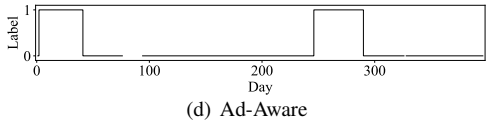
(a) MicroWorld-eScan



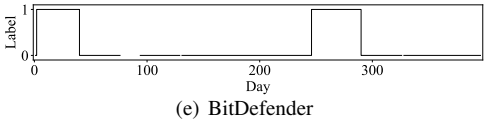
(b) Emsisoft



(c) GData



(d) Ad-Aware



(e) BitDefender

Figure 16: An example file with almost identical detection results reported by the five engines in Cluster-I (Section 5.1). *MD5:* `2efcdc93a9de94b1604f6665d2a0589a`.

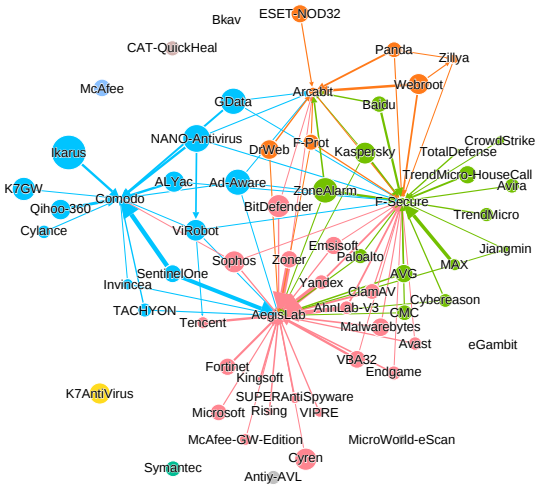


Figure 17: Active influence graph for 0→1 flips. *Edges weight<0.25 are removed. A thicker edge A to B means a larger influence from A to B. A larger node means a bigger influencer.*

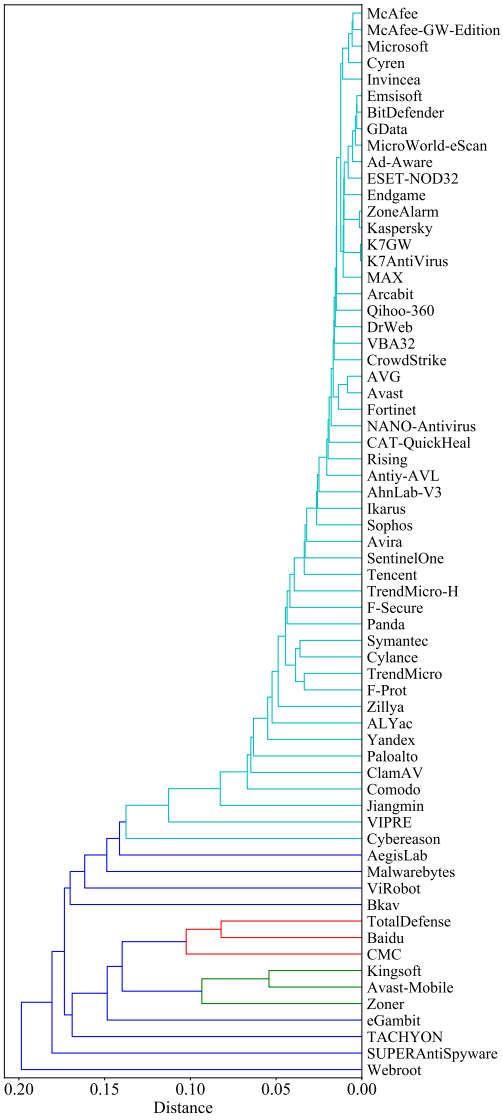
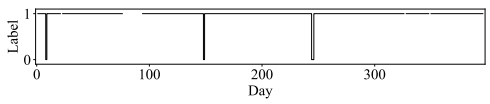
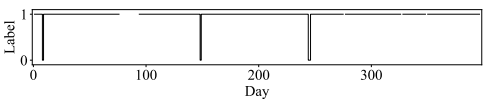


Figure 18: The dendrogram of engine clusters. *x-axis shows the distance threshold. A shorter distance indicates two engines have more similar label sequences. Clusters with distance shorter than the threshold are gradually merged together.*



(a) Microsoft



(b) McAfee

Figure 19: An example file with almost identical detection results reported by the two engines in Cluster-II (Section 5.1). *MD5:* `bffa5f6881f9abfed54037b446e34b94`.