Learning from Big Malwares

Abstract

This paper calls for the attention to investigate real-world malwares in large scales by examining the largest real malware repository, VirusTotal. As a first step, we analyzed two fundamental characteristics of Windows executable malwares from VirusTotal. We designed offline and online tools for this analysis. Our results show that malwares appear in bursts and that distributions of malwares are highly skewed.

1. Introduction

Malwares grow exponentially [3]. AVTest [3] reports that more than 140 million new malwares appeared in 2015. Such malwares are posing increasing threats to the human society every day. For example, there were almost 2 million attempts to steal money from online bank accounts with malwares that exploit vulnerabilities in the Adobe Flash player [9].

Researchers and practitioners continue to build security tools to defend against new malwares. To assist the design of these tools, it is essential to understand malwares in the real world.

Previous works on analyzing the behaviors and evolutions of malwares [7, 16] have provided insights into how malwares circumvent the detection from existing antivirus techniques and how malware writers create new malwares. However, these works only studied a limited amount of malwares that are targeted for certain types of security threats or antivirus engines.

Studying malwares in a large scale and with high diversity, what we call *big malwares*, can expose new insights beyond these isolated studies.

VirusTotal [2] is a popular online service that real-world users use to analyze suspicious files and URLs. It applies more than 50 antivirus engines to each submitted file to detect various kinds of malwares including viruses, worms, and trojans. It then generates a summary report that includes the detection results of all these engines. VirusTotal saves and provides an open access to all user-submitted files and generated reports.

The VirusTotal repository provides a valuable resource to gain insights into the behavior of malwares. First, it contains a huge amount of real-world files. For example, there were more than 40 million suspicious files submitted in November 2015 (Figure 1). Files from VirusTotal were submitted by real-world users from all over the world since 2004 and involve various security threats. This amount of diverse data makes VirusTotal a good representative of malwares in the real world.

Second, VirusTotal applies a host of state-of-the-art antivirus engines to all submitted files, VirusTotal captures how these engines evolve over time.

Third, VirusTotal provides rich metadata. Besides reporting whether or not a submitted file has malware, VirusTotal also captures the exact type of malware, which antivirus engines detected the malware, when and by whom the file is submitted, and other useful metadata. There are also active malware researchers and engineers who comment and vote on each submitted file, providing valuable human inputs.

The VirusTotal repository exposes many new research opportunities. For example, studying malwares over a long time period and across all countries in the world can provides a high-level insight into how malwares evolve over time and over geo-locations. Studying how antivirus engines change over time together with how malwares change can reveal the effectiveness of responses to new security threats. Finally, it is interesting to investigate if we can build online malware prediction tools by applying machine learning techniques on the VirusTotal data. All these insights could in turn assist future antivirus researchers and engineers to better design malware defense mechanisms.

Unfortunately, there has been little work in looking at this valuable repository. In industry, many antivirus vendors use VirusTotal to identify false negatives and false positives in their products. However, they only use VirusTotal to examine suspicious files separately, and do not consider correlations among different suspicious files. In academia, only until recent did researchers begin to pay attention to mining the VirusTotal repository. Graziano *et al.* [6] leveraged VirusTotal user ID information to identify malware writers who use VirusTotal as a test platform.

We propose to investigate in the VirusTotal repository along two directions: offline study of its rich data and metadata, and online analysis and prediction of malwares.

As a first step, we collected one month of VirusTotal repository data with more than 40 million suspicious files and conducted an early-stage empirical study on them.

We centered our analysis around the concept of *malware families*. A malware family is a set of malwares that have common behaviors. We focus our initial attention on all Windows executable (PE) malwares detected by Microsoft antivirus engines, because our previous experience shows that Microsoft engines can generate precise results on PE files.

We focus our analysis on two dimensions: temporal properties of malwares and malware family distribution. In each dimension, we investigate both offline and online analysis mechanisms.

We began our temporal analysis with general malware characteristics including the submission frequencies and generation rates of malware families. We then study the burstiness of malwares—how closely malwares belonging

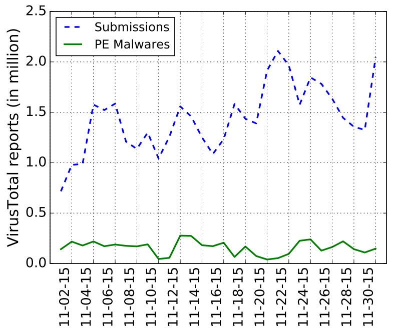

Figure 1: The number of files and malwares. ***The number of suspicious files and the number of malwares submitted to VirusTotal in November 2015.***

to the same family occur in time. To understand malware burstiness, we designed a new cache-based mechanism that can support both offline and online analysis. Moreover, our LRU cache can predict malware families occurrences in the near future, allowing antivirus vendors to better focus their efforts.

We focus our malware family distribution study on detecting skewness of malware families and identifying hot families—families that have many malwares. We observe that the distribution of malware families is highly skewed. To assist online analysis of malware family distribution, we apply a frequent item mining algorithm [11] to identify hot malware family. This solution can answer historical hot malware families in nearly-real time using constant memory.

In summary, this paper makes the following contributions:

- We are the first to collect mass data from the VirusTotal repository for large-scale malware study. We collected and preprocessed files submitted to VirusTotal in November 2015 (Section 2).
- We studied the burstiness of malwares using a new cache-based malware prediction tool. This tool achieves greater than 90% prediction precision (Section 3).
- We observe that family distributions of malwares are highly skewed, *i.e.*, some family has a huge amount of malwares while others only include a few malwares. Leveraging this observation, we built a hot malware family mining tool that identifies hot malware families (Section 4).
- We identify several key research opportunities on top of the VirusTotal repository (Section 5).

2. Data Collection

This section discusses how we collected and preprocessed data from VirusTotal. We will present the analysis of these data in the next two sections.

We downloaded the metadata and malware reports of all submitted files in November 2015 using the APIs VirusTotal provides, resulting in a total of 43 million reports. Table 1 shows the metadata fields and their meaning.

Metadata Field	Explanation
name	submitted file name
timestamp	timestamp when the submission was made
source_country	the country where the submission was made
source_id	user ID that made the submission
size	file size
type	file type
tags	labels with more specific information for each **type**
first_seen	when the same type of file was first submitted
last_seen	when the same type of file was last submitted
hashes	hash value of the submitted file
total	number of engines that analyzed the file
positives	number of engines that flagged the file as malicious
positives_delta	changes in **positives** across different engine scans
report	detailed detection report from each AV engine

Table 1: VirusTotal Metadata. ***Fields for each submission retrieved from the VirusTotal private API and their relation explanation. The same file can be submitted multiple times by different users.***

In this paper, we only focus on *Portable Executable (PE)* files and leave the analysis of other types of malicious files for future work. We filter all downloaded metadata by the tag field. If the tag field contains either "peexe" or "pedll" tags, we consider the record as a PE file. We then use Microsoft antivirus engines to judge whether a PE file is a malware. Within the 43 million reports, 4.7 million are PE malwares. The number of reports and PE malwares submitted each day are shown in Figure 1.

The Microsoft antivirus engine classifies malwares into different names [12]. We utilize this naming mechanism to decide what family a malware belong to. Specifically, if two malwares have the same name under Microsoft engine, we consider them to be from the same family.

One caveat of VirusTotal is that it is possible that the VirusTotal APIs return redundant reports for the same submitted file. We use the combination of md5 hash value and timestamp to detect and remove redundant reports.

After removing redundant reports, we find that most malwares were submitted only once to VirusTotal. Out of the total 4.7 million PE malware submissions, 4 million are distinct. On average, each PE malware was submitted 1.17 times to VirusTotal. This observation is in contradictory to the common belief that most malwares are encountered by more than one user. We suspect that the reason behind this low degree of repeated submissions by different users is that VirusTotal users tend to check whether their suspicious files have already been submitted and avoid submitting redundant files.

Similar to all previous empirical studies, all our findings, experimental results, and conclusions need to be considered with our methodology in mind.

The VirusTotal APIs only track which submission reports are sent to each downloader approximately, and there is no guarantee that all submission reports on VirusTotal can be downloaded successfully. Thus, it is possible that we missed

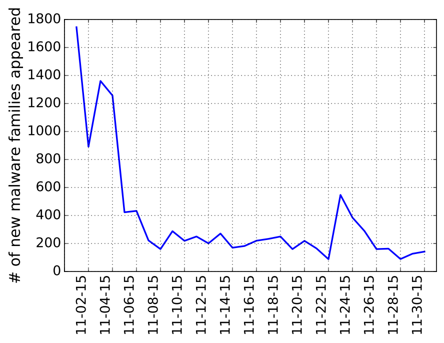

Figure 2: New malware families on VirusTotal in November 2015. ***The number of new malware families we observed every day in November 2015.***

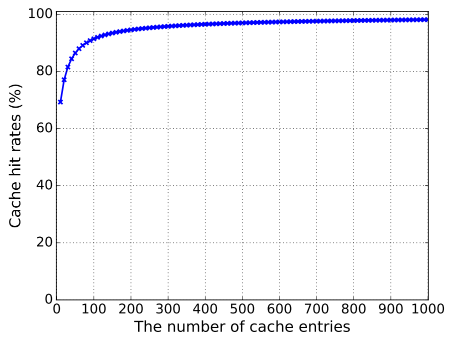

Figure 3: Relation between cache hit rate and cache size. ***Cache hit rate under different values of cache size from 10 to 1000.***

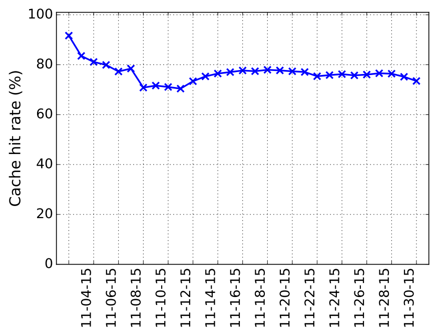

Figure 4: Cache hit rate in November 2015. ***Cache hit rate every day in November 2015 if we only update cache content at the end of every day.***

some malwares submitted to VirusTotal. Also, we only leverage Microsoft antivirus engines to decide whether or not a submission is malicious, and it is possible that Microsoft antivirus engines cannot make this decision precisely. How to get a precise label for a PE file is out of the scope of this paper. Although there is a huge amount of malwares on VirusTotal, we believe that there are malwares never submitted to VirusTotal, and there are malwares submitted much later than when they appear in the real world. Since there are no conceivable ways to study these malwares, we believe that the malwares in our study provide a representative malware sample of the real world.

3. Malware Temporal Properties

This section presents our study of the temporal properties of VirusTotal malwares and answers two fundamental questions: how many malware families appear everyday and whether or not malwares occur in bursts. To answer the second question, we design a new caching mechanism that can be used for both offline and online malware predictions. This cache-based malware prediction technique can predict which malware families will appear in the near future with high precision.

We first study how many new malware families appear everyday. Figure 2 shows the number of new malware families appearing on each day in November 2015. Since we do not include data before November 1, there are more new malware families in the first few days. After that, the number of new malware families becomes stable, falling into a range between 100 and 400. In total, there are 11311 malware families in this time period.

Observation 1: *100-400 new malware families appear each day.*

Next, we investigate whether malwares behave temporal locality. Temporal locality is an important metric that can guide the prediction of near-future malwares.

Specifically, we analyze how bursty malwares in the same family appear. Inspired by previous usage of cache mechanisms in predicting bugs [10], we design a new cache mechanism to study the burstiness of malwares.

This caching mechanism can be used to analyze historical malware reports offline as well as to analyze online malwares and predict near-future malware occurrences.

We view malware submission reports as a stream of inputs and feed the stream into a cache. Each input (*i.e.*, each report) is associated with an address (the family it belongs to) and a time (the submission time of the file). Thus, a cache hit means that the new report belongs to a family that is currently in the cache, *i.e.*, the new input has the same address as an entry in the cache. Therefore, a high cache hit rate implies that the occurrence of malwares exhibit temporal locality.

As with general cache mechanisms such as CPU caches and file system buffer caches, there are several parameters to tune in a cache. The cache block size, or cache line size, controls the granularity of cache management, *i.e.*, how many entries are inserted into and evicted from cache together. Cache prefetching loads spatially close entries into caches in advance. Cache replacement policy controls what entries to evict when cache is full. We use a simple cache setting in our evaluation. We fix the cache block size to be one, no prefetching, and use the LRU (least recently used) replacement policy. We leave the exploration of the effect of other cache configurations for future work.

Our malware family cache works as follows. We start with an empty cache. For a new file submission, if the malware family is already in the cache, we move the hit cache entry to the front of our cache entry list. If the new malware family is not in the cache, we create a new cache entry and add it into the front of the cache entry list. If the cache is full, we evict the entry at the end of the list. The cache hit rate is calculated as follows:

$$\text{hit rate} = \frac{\text{\# of hits}}{\text{\# of hits + \# of misses}}$$

We implemented this malware family cache using Python and conducted experiments on an AWS c4.4xlarge virtual machine, which contains 16 virtual CPUs and 30 GB memory.

We conducted two experiments. The first experiment explores how the cache hit rate changes with the number of cache entries. We change the cache size from 10 to 1000. As shown in Figure 3, the hit rate grows from 69.29% to 98.14%. When using more than 80 cache entries, which is less than 1% of the total number of malware families, the cache hit rate rises above 90%, and when using more than 230 cache entries, which is less than 3% of the total number of malware families, the cache hit rate rises above 95%. The high cache hit rate confirms that malwares occur in bursts.

Observation 2: *The occurrence of malwares in each family has strong temporal locality.*

To support online malware occurrence prediction, it is essential to lower the performance overhead of running our cache mechanism. To this end, we lower the cache content update frequency from once per malware report to once per day. That is, we keep cache content unchanged to count cache hits and cache misses each day and update the cache content at the end of each day. In this second experiment, we fix the cache size to 100. Figure 4 shows the cache hit rate for each day. Most cache hit rate falls in a range between 70% and 80%, showing that even when lowering performance overhead, our cache mechanism still achieves an good estimation of malware occurrences.

4. Malware Family Distribution

This section presents our study on how malwares distribute in malware families.

We first count how many malwares fall into different malware families. As shown in Figure 5, only a small number of malware families are hot, *i.e.*, containing a large amount of malwares. The distributions of malware families follow the well-known Pareto principle: more than 90% of malware families take place in only 10% of malwares. The distributions of malware families are highly skewed, making it easy to precisely identify hot malware families.

Observation 3: *Distributions of malware families are highly skewed.*

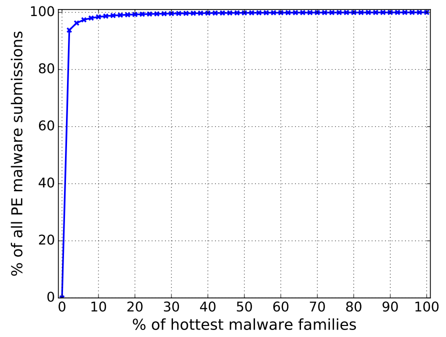

Figure 5: Skewness of malware families appearing in November 2015. ***Accumulation distribution of malwares in each malware family in November 2015.***

While the simple bin-counting mechanism works well on our one-month testing data, a total of 11311 distinct malware families, applying the same mechanism over a large dataset can cause significant memory space and performance overhead.

To reduce these overheads and better support online analysis of huge datasets, we need to seek alternative mechanisms. The skewness of malware families indicates we can apply a frequent item mining algorithm to identify hot malware families.

Frequent item mining algorithms take two configuration parameters, ϕ and ε, where $\phi > \varepsilon$. The goal of frequent item mining algorithms is to provide a nearly-real time analysis on massive data streams by using constant memory. Assuming the length of the input stream is N, the output of frequent item mining algorithms includes all items that appear more than $\lfloor \phi N \rfloor$ times and not include any item that appears less than $\lfloor \varepsilon N \rfloor$ times.

The frequent item mining algorithm we use is a space-efficient algorithm [11] proposed for streams in Internet advertising and that has already been applied in other areas, like mining hot calling contexts in profilers [5]. Space saving algorithm tracks $M = 1/\varepsilon$ pairs of (f, c). f is short for malware family, and c is short for counter. The content of these pairs represents (ϕ, ε)-*HMF* (Hot Malware Family). The M pairs are initialized with the first M encountered malware families and their frequency. When a new malware submission arrives, if the malware family is already being monitored, the related counter will be increased by 1. And if the malware family is not being monitored, we will replace the malware family of the pair with the lowest counter value with the incoming malware family and increase its counter value by 1. When querying HMF, all malware families whose counter values are larger than $\lfloor \phi N \rfloor$ will be returned.

We implement the space saving algorithm using Python and conduct experiments in the same system as we did in Section 3. Following previous works on frequent item min-

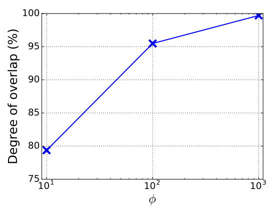

Figure 6: Relation between ϕ and degree of overlap. ***How the degree of overlap changes with the change of ϕ from 10^1 to 10^3.***

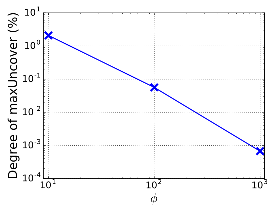

Figure 7: Relation between ϕ and degree of maxUncover. ***How the degree of maxUncover changes with the change of ϕ from 10^1 to 10^3.***

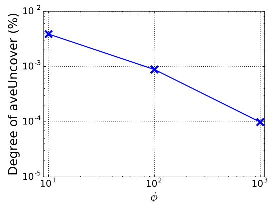

Figure 8: Relation between ϕ and degree of aveUncover. ***How the degree of aveUncover changes with the change of ϕ from 10^1 to 10^3.***

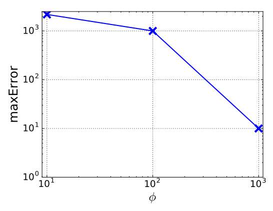

Figure 9: Relation between ϕ and maxError. ***How maxError changes with the change of ϕ from 10^1 to 10^3.***

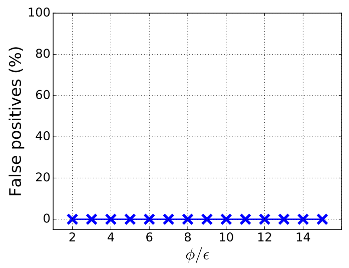

Figure 10: False positives. ***False positives in (ϕ,ε)-HMF as a function of ε. The value of ϕ is fixed to 10^{-2}. The value of ϕ/ε changes from 2 to 15.***

ing [5], we measure the following metrics by using the malware submission data we collect:

1. Degree of *overlap* is used to measure the percentage of malwares covered in (ϕ,ε)-HMF, and it is defined as follows:

$$overlap((\phi,\varepsilon)\text{-}HMF) = \frac{1}{N} \sum_{f \in (\phi,\varepsilon)\text{-}HMF} w(f)$$

   where $w(f)$ represents the real frequency of malware family f.

2. *MaxUncover* is short for maximum frequency of uncovered malware families. It is defined as follows:

$$maxUncover((\phi,\varepsilon)\text{-}HMF) = \max_{f \notin (\phi,\varepsilon)\text{-}HMF} w(f)/H(f)$$

   where $H(f)$ is the maximum frequency of all malware families.

3. *AveUncover* is short for average frequency of uncovered malware families and it is defined in a manner similar to *maxUncover*.

4. *False positives* are defined as malware families returned when querying HMF but whose real frequencies are less than $\lfloor \phi N \rfloor$. Space saving algorithm is designed to guarantee that there will be no false negatives.

5. *MaxError* is used to measure the relative error of counter values compared to their real frequencies. It is defined as follows:

$$maxError((\phi,\varepsilon)\text{-}HMF) = \max_{f \in (\phi,\varepsilon)\text{-}HMF} \frac{|c(f) - w(f)|}{w(f)}$$

There are two configuration parameters in space saving algorithm: ϕ and ε, and the number of monitored (f,c) pairs is directly controlled by ε. Following previous experience in applying space saving algorithm [5], we set $\varepsilon = \phi/5$ as the default, unless we explicitly state otherwise.

We first evaluate how the degree of *overlap* would change with the change of ϕ. The degree of *overlap* is used to describe how many malwares are monitored in (ϕ,ε)-HMF, and the larger it is, the better. As shown by Figure 6, after we change ϕ from 10 to 100, the degree of *overlap* increases from 79.93% to 95.48%. The degree of *overlap* further increases to 99.70% after we change ϕ to 1000.

We then study how maxUncover and aveUncover would change with the change of ϕ. Both maxUncover and aveUncover are used to describe malwares not monitored in (ϕ,ε)-HMF, and the lower they are, the better. As shown by Figure 7 and Figure 8, both maxUncover and aveUncover decrease by an order as we increase the ϕ value by an order.

Figure 9 shows how *maxError* changes after we change ϕ. *maxError* describes how precise the counters in (ϕ,ε)-HMF are, and the lower it is, the better. *maxError* drops from 2178 to 999 after we change the value of ϕ from 10 to 100. *maxError* value becomes 10 after we change the value ϕ to 1000. The large *maxError* value is due to the fact that space saving algorithm will conservatively assume that the frequency of a new malware family is one larger than the smallest counter value of all monitored malware families.

In the last experiment, we fix ϕ to 10^{-2} and change ϕ/ε from 2 to 15 to evaluate how false positives would change. As shown by Figure 10, space saving algorithm constantly reports 0 false positives in our experiments.

5. Research Opportunities

Although our initial investigation of VirusTotal is successful, there are a many more research opportunities beyond our initial study.

Studying correlations among metadata fields. We currently only leverage timestamp and Microsoft detection reports. There are many other metadata fields. Conducting data mining on these fields, especially their correlations, can enable many other "big malwares" applications. For example, future research can explore correlations between the features of malwares and their detection rates to understand what features are ignored by existing antivirus engines.

Utilizing other antivirus vendors' reports. We currently only leverage reports from Microsoft, but these reports may not always be accurate. There are more than 40 antivirus vendors' reports provided on VirusTotal. Some vendors reports may be better than others or better than others under some special conditions. We leave efforts evaluate all those reports systematically and to better combine reports from different vendors for future work.

Studying other types of malicious files. Besides PE files, there are other types of malicious files on VirusTotal such as malicious apps, URL, and binary files on non-Windows OSes. How these malicious file behave remains an open question.

Using other analysis granularity. Currently, we use malware family as prediction granularity. One can predict malwares using finer granularities. For example, *ssdeep* values are also provided in VirusTotal metadata, and these values can be used to cluster malwares. One promising approach is to cluster malwares in each family first, and then use cluster as prediction granularity.

Leveraging other information on VirusTotal. Besides the static information discussed in Section 2, VirusTotal also hosts some behavior data for each malware sample. Mining these data can help us understand which behaviors are more prominent in malwares, and which vulnerabilities are more likely to be used by malwares, both of which can be used as indicator to detect new maliciousness.

Training machine learning models by using VirusTotal data VirusTotal provides a huge set of labeled malwares. It is interesting to consider leveraging VirusTotal data to train a machine learning model, and applying the trained model to conduct malware detection and classification. However, we need to figure out a feature set before training the model. Which information provided by VirusTotal can be included into the feature set remains an open question. If we need features beyond information provided by VirusTotal, does the feature extraction scales with the size of VirusTotal data also remains an open issue. Neural network takes binary as inputs, and can extract features automatically. However, neural network requires all binary inputs with the same size. It is easier to resize images. If we want to apply neural network to malwares, how could we resize malware?

Improving antivirus products. Finally, there are opportunities to improve existing antivirus products. For example, many endpoint antivirus softwares, like ClamAV, are built based on a database of malwares' signatures. When these antivirus softwares scan suspicious files, all signatures in the database will be checked. It is possible to explore how to automatically extract malware signatures by leveraging VirusTotal data, instead of extracting them manually. And if we can precisely predict which malware will appear in the near future, we could reduce the size of signatures sent to clients' side and also reduce time to check the signature database.

6. Related works

Many research efforts [1, 4, 8, 13, 14] have been made to explore how to leverage "big code" repositories, such as GitHub, BitBucket, and CodePlex, and these works inspire us to explore how to leverage data on VirusTotal. SLANG [13] can fill uncompleted programs with call innovations by using statistical models trained from extracted sequences of API calls from large code bases. JSNICE [14] can predict identifier types and obfuscated identifier names for Javascript programs. JSNICE translates programs into dependence graphs and learns a CRF model by using a large training set. All predictions are made by optimizing a score function based on the learned CRF model. Karaivanov et al. [8] apply phrase-based statistical translation approaches to translate C# programs to Java. To sum up, all these techniques are built based on source code repositories, and their goals are to improve the development stage. However, VirusTotal is a repository containing binary malwares and the goal for conducting data mining on VirusTotal data is to improve antivirus techniques.

There are existing works [6, 15] regarding the conducting of data mining on VirusTotal data to identify malware development cases, where malware writers use VirusTotal as a testing platform and try to develop malwares that cannot be detected by antivirus engines. These techniques utilize submission_id information, which is different from the information we use. We believe other information on VirusTotal could also be leveraged in the future.

7. Conclusion

VirusTotal provides a fruitful opportunity to understand real-world malwares in a large scale. Unfortunately, it has been largely overlooked by the research community. In this paper, we conduct an empirical study on PE malwares on VirusTotal and analyze their temporal and family distribution characteristics. We expect our work to deepen our understanding of and bring more attention to the data on VirusTotal.

References

[1] learnbigcode. URL: http://learnbigcode.github.io/.

[2] VirusTotal. URL: https://www.virustotal.com/.

[3] AV-TEST. Malware Statistics. URL: https://www.av-test.org/en/statistics/malware/.

[4] P. Bielik, V. Raychev, and M. Vechev. Programming with "Big Code": Lessons, Techniques and Applications. In *SNAPL*, 2015.

[5] D. C. D'Elia, C. Demetrescu, and I. Finocchi. Mining hot calling contexts in small space. In *Proceedings of the 32Nd ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI '11, pages 516–527, New York, NY, USA, 2011. ACM. ISBN 978-1-4503-0663-8. . URL `http://doi.acm.org/10.1145/1993498.1993559`.

[6] M. Graziano, D. Canali, L. Bilge, A. Lanzi, and D. Balzarotti. Needles in a haystack: Mining information from public dynamic analysis sandboxes for malware intelligence. In *Proceedings of the 24th USENIX Conference on Security Symposium*, SEC'15, pages 1057–1072, Berkeley, CA, USA, 2015. USENIX Association. ISBN 978-1-931971-232. URL `http://dl.acm.org/citation.cfm?id=2831143.2831210`.

[7] A. Gupta, P. Kuppili, A. Akella, and P. Barford. An empirical study of malware evolution. In *Proceedings of the First International Conference on COMmunication Systems And NETworks*, COMSNETS'09, pages 356–365, Piscataway, NJ, USA, 2009. IEEE Press. ISBN 978-1-4244-2912-7. URL `http://dl.acm.org/citation.cfm?id=1702135.1702182`.

[8] S. Karaivanov, V. Raychev, and M. Vechev. Phrase-based statistical translation of programming languages. In *Proceedings of the 2014 ACM International Symposium on New Ideas, New Paradigms, and Reflections on Programming & Software*, Onward! 2014, pages 173–184, New York, NY, USA, 2014. ACM. ISBN 978-1-4503-3210-1. . URL `http://doi.acm.org/10.1145/2661136.2661148`.

[9] Kaspersky. Kaspersky Security Bulletin 2015 . URL: https://securelist.com/analysis/kaspersky-security-bulletin/73038/kaspersky-security-bulletin-2015-overall-statistics-for-2015/.

[10] S. Kim, T. Zimmermann, E. J. Whitehead Jr., and A. Zeller. Predicting faults from cached history. In *Proceedings of the 29th International Conference on Software Engineering*, ICSE '07, pages 489–498, Washington, DC, USA, 2007. IEEE Computer Society. ISBN 0-7695-2828-7. . URL `http://dx.doi.org/10.1109/ICSE.2007.66`.

[11] A. Metwally, D. Agrawal, and A. E. Abbadi. An integrated efficient solution for computing frequent and top-k elements in data streams. *ACM Trans. Database Syst.*, 31(3):1095–1133, Sept. 2006. ISSN 0362-5915. . URL `http://doi.acm.org/10.1145/1166074.1166084`.

[12] Microsoft. Naming malware. URL: https://www.microsoft.com/security/portal/mmpc/shared/malwarenaming.aspx.

[13] V. Raychev, M. Vechev, and E. Yahav. Code completion with statistical language models. In *Proceedings of the 35th ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI '14, pages 419–428, New York, NY, USA, 2014. ACM. ISBN 978-1-4503-2784-8. . URL `http://doi.acm.org/10.1145/2594291.2594321`.

[14] V. Raychev, M. Vechev, and A. Krause. Predicting program properties from "big code". In *Proceedings of the 42Nd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '15, pages 111–124, New York, NY, USA, 2015. ACM. ISBN 978-1-4503-3300-9. . URL `http://doi.acm.org/10.1145/2676726.2677009`.

[15] K. ZETTER. A Google Site Meant to Protect You Is Helping Hackers Attack You. URL: https://www.wired.com/2014/09/how-hackers-use-virustotal/.

[16] Y. Zhou and X. Jiang. Dissecting android malware: Characterization and evolution. In *Proceedings of the 2012 IEEE Symposium on Security and Privacy*, SP '12, pages 95–109, Washington, DC, USA, 2012. IEEE Computer Society. ISBN 978-0-7695-4681-0. . URL `http://dx.doi.org/10.1109/SP.2012.16`.