Learning from Big Security

Abstract

VirusTotal contains the largest dataset for real-world malwares with labels from state-of-the-art antivirus techniques. In this paper, we study the characteristics of PE malwares on VirusTotal. We find that malwares appear in bursts and that distributions of malwares are highly skewed. We also argue that research community should pay more attention to the data on VirusTotal.

1. Introduction

As a free online service, VirusTotal [2] analyzes files submitted by real-world users to identify many different kinds of malwares such as viruses, worms, trojans, and so on. VirusTotal applies different antivirus engines to each submitted file and generates an aggregated report. All submitted files and generated reports are saved and can be accessed through VirusTotal's API. Inspired by previous work on mining software repositories [1, 3, 6, 7, 10, 11] in the software engineering and programming languages community, we believe that leveraging data on VirusTotal could also enable many "big security" applications.

The repository on VirusTotal provides a valuable resource for conducting data mining. First, there is a huge amount of data on VirusTotal. Figure 1 shows that there were more than 40 million suspicious files submitted in November 2015. This amount of data makes VirusTotal a rough estimation of malwares in the real world. Second, all data on VirusTotal are labeled by state-of-the-art antivirus techniques. VirusTotal updates each antivirus engine every five minutes. Besides noting whether a given a submitted file has been detected by an antivirus engine, VirusTotal also keeps the exact detection tag returned by each engine. There are also online active malware researchers who can comment and vote on each submitted file and thus serve as an important supplement to antivirus engines.

In industry, antivirus vendors widely use VirusTotal to identify false negatives and false positives in their products. However, they only utilize VirusTotal reports separately for each single suspicious file, failing to consider correlations among different suspicious files. In academia, researchers have begun to pay attention to mining the VirusTotal repository. For example, Graziano et al. [5] leverage submission_id information to identify malware writers who use VirusTotal as a test platform. We believe there are many more research opportunities available through mining VirusTotal.

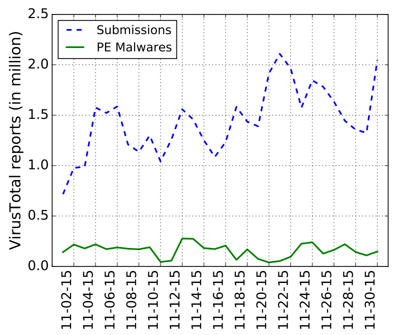

Figure 1: The number of suspicious files and the number of malwares submitted to VirusTotal in November 2015.

In this paper, we conduct an empirical study on the malware dataset on VirusTotal. An empirical study is the prerequisite to conduct data mining on VirusTotal. We first collect more than 40 million suspicious file submissions from VirusTotal, and then we focus our analysis on Windows executable malwares detected by Microsoft antivirus engine to guarantee accuracy. Our analysis includes general characteristics such as submission frequency and the generation rate of malware families, temporal characteristics, and family distribution characteristics. We hypothesize that malwares do not appear uniformly across time, rather they appear in bursts. To validate our hypothesis, we build a cache-based malware prediction technique that aims to predict malwares in which families would appear in the near future. Our malware family cache can achieve a 90% cache hit rate by only using 100 cache entries. Our technique would allow antivirus vendors to focus their efforts. We also observe that distributions of malware families are highly skewed. This observation inspires us to apply a frequent item mining algorithm to identify hot malware families. We view malware submissions as a stream based on their submission timestamps and feed this stream into space saving algorithm [9]. Our solution can precisely answer hot malware family queries in nearly-real time, by using a constant number of counters.

To sum up, we make the following contributions in this paper:

- We collect data submitted to VirusTotal in November 2015 and analyze their general characteristics. We find that most malware samples are only submitted once to

VirusTotal (Section 2) and that roughly 100-400 new malware families appear each day (Section 3).

- We hypothesize that malwares appear in bursts. Our cache-based malware prediction technique confirms our hypothesis, and it can achieve greater than 90% prediction precision (Section 3).
- We observe that family distributions of malwares are highly skewed. Leveraging this observation, we build a hot malware family mining technique that can identify hot malware families by using a constant number of counters (Section 4).
- We discuss the future research opportunities available with regard to mining data on VirusTotal (Section 5).

2. Methodology

Metadata Fields	Explanation
name	file name of the submitted sample
timestamp	timestamp for when the submission was made
source_country	the country from which the submission was made
source_id	user id of whom made the submission
tags	VirusTotal tag
link	where to download the submitted sample
size	file size
type	file type
first_seen	when the same sample was first submitted
last_seen	when the same sample was last submitted
hashes	including sha1, sha256, vhash, md5, and ssdeep
total	how many engines analyze the sample
positives	how many engines identify the sample as malicious
positives_delta	changes in **positives** fields
report	detailed detection report from each AV engine

Table 1: Metadata fields for each submission retrieved from the VirusTotal private API.

We download submission reports' metadata from the private API of VirusTotal. Table 1 shows all metadata fields and their meaning. In this paper, we only focus on Portable Executable (PE) files, and we leave the analysis of other types of malicious files for the future. We filter all records by the tag field. If the tag field contains either "peexe" or "pedll" tag, the record is considered to be a PE file. Then we rely on Microsoft antivirus engine to judge whether the PE file is a malware and if so its malware family.

It is possible that the VirusTotal private API returns redundant reports, and we use the combination of md5 and timestamp to detect and merge redundant reports.

We downloaded all reports in November 2015, collecting a total of 43 million reports of which 4.7 million are PE malwares. The number of reports and PE malwares submitted each day are shown in Figure 1.

After removing redundant submissions, we find that most malwares were submitted only once to VirusTotal. 4 million out of the total 4.7 million PE malware submissions are distinct. On average, each PE malware was submitted 1.17 times to VirusTotal. We believe that most malwares are encountered by more than one VirusTotal user. One possible reason for this observation is that VirusTotal users tend to check whether their samples have already been submitted before making their own submissions.

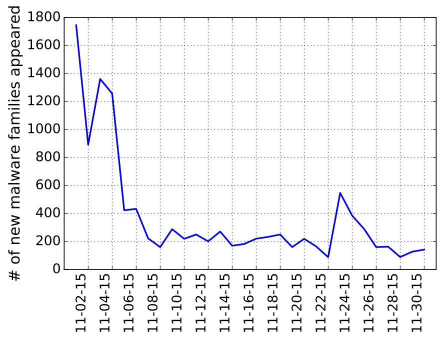

Figure 2: The number of new malware families we observed each day in November 2015.

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

The VirusTotal private API only tracks which submission reports are sent to each downloader approximately, and there is no guarantee that all submission reports on VirusTotal can be downloaded successfully. It could be possible that we missed some malwares submitted to VirusTotal. Also, we simply leverage Microsoft antivirus engine to decide whether a submission is malicious or not, and it is possible that Microsoft antivirus engine cannot make this decision precisely. However, 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 there are malwares never submitted to VirusTotal, and there are malwares submitted much later than when they appear in the real world. However, 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. Temporal Analysis

In this section, we study the temporal property of VirusTotal malwares.

We first study how many new malware families appear each day. Figure 2 shows the number of new malware families appearing each day in November 2015. We do not have any data before November 1, so there are many 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.

Observation 1: There are 100-400 new malware families appearing each day.

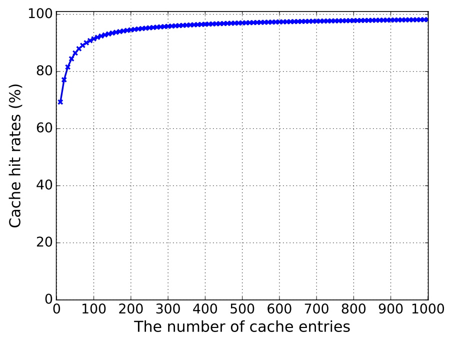

Figure 3: Relation between cache hit rate and cache size.

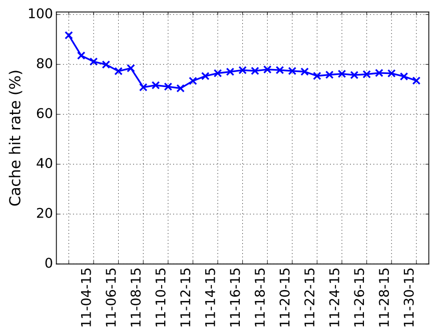

Figure 4: Cache hit rate when updating cache content once a day.

We then investigate whether there is time locality for malwares. Specifically, we hypothesize that malwares in the same family would appear in bursts. We borrow the cache mechanism from system research area to validate our hypothesis. Cache was previously used to predict bugs [8], since bugs are introduced not uniformly in time but in bursts and with a strong time locality. We view malware submission reports as a stream according to each report's submission timestamp, and feed the stream into a cache. If the cache hit rate is high, we can conclude that the appearance of malwares also has time locality.

There are several notions in cache terminology: block size is defined as how many entries would be added into the cache or evicted from the cache together. Pre-fetch means loading entries that the cache has not encountered yet. Replacement policy controls how to evict entries from cache, when the cache is full. We use the most simple cache setting in our evaluation. We fix the block size to be 1, disable pre-fetch, and use LRU (least recently used) replacement policy. Our malware family cache works as follows.

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

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

We implement the LRU cache by using python-2.7. Our experiment is conducted on an aws c4.4xlarge virtual machine, which contains 16 virtual cpus and 30G memory.

We conduct two experiments. In the first experiment, we explore 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%.

In the second experiment, we fix the cache size at 100, and 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. Figure 4 shows the cache hit rate for each day. Most cache hit rate falls in a range between 70% and 80%.

Observation 2: The appearance of malwares in each family has a strong time locality.

Discussion. Resources to combat malwares are limited. Any techniques that could allow antivirus vendors to focus their efforts would be great. Our cache-based malware prediction technique can predict which malware families will appear in the near future with high precision. We note that there are many other cache configurations, for example, those with different cache block sizes, different pre-fetch mechanisms, and different replacement policies. We leave the exploration of the effect of their combinations for the future. Currently, we use malware family as prediction granularity. In the future, we could try to predict malwares using a finer granularity. For example, ssdeep values are also provided in VirusTotal metadata, and these values can be used to cluster malwares. We could cluster malwares in each family first, and then use cluster as prediction granularity.

4. Distribution Analysis

In this section, we study how malwares are distributed in each malware family.

As shown in Figure 5, only a small number of malware families are hot. The distributions of malware families follow the well-known Pareto principle, and more than 90% of malware families take place in only 10% of malwares.

Observation 3: The distributions of malware families are highly skewed.

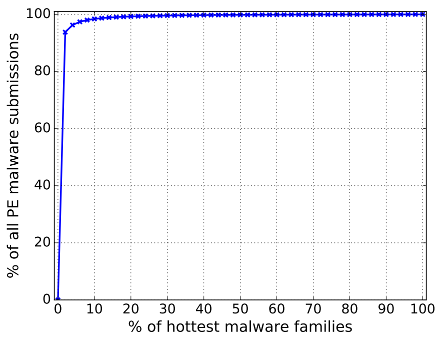

Figure 5: Skewness of malware families appearing in November 2015.

The skewness of malware families indicates we could 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 space saving algorithm [9] that was proposed for streams in Internet advertising and that has already been applied in other areas, like mining hot calling contexts in profilers [4]. 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 by using python-2.7 and conduct experiments in the same system as we did in Section 3. Following previous works on frequent item mining [4], 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 [4], 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 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.

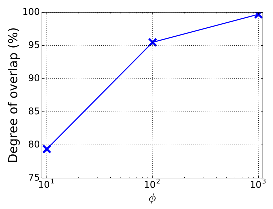

Figure 6: Relation between ϕ and degree of overlap.

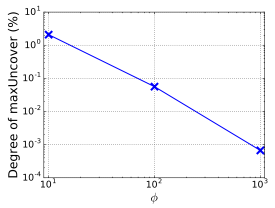

Figure 7: Relation between ϕ and degree of maxUncover.

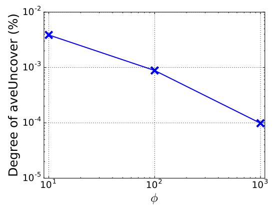

Figure 8: Relation between ϕ and degree of aveUncover.

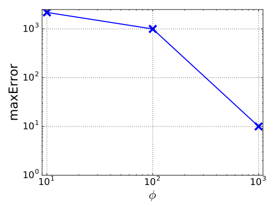

Figure 9: Relation between ϕ and maxError.

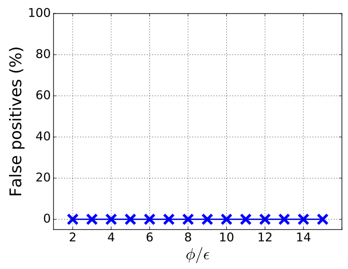

Figure 10: False positives in (ϕ, ε)-HMF as a function of ε. The value of ϕ is fixed to 10^{-2}.

As shown by Figure 10, space saving algorithm constantly reports 0 false positives in our experiments.

Discussion. The distributions of malware families are highly skewed. This is the reason why we could precisely identify hot malware families. There are in total 11311 distinct malware families in our tested data. Although this number is small, new malware families would appear every day and applying frequent item mining algorithms can identify hot malware families from possibly infinite malware families by using constant memory.

As we discussed in Section 3, malware family is a relatively coarse granularity. In the future, we could consider how to mine frequent items in the submission stream using a finer granularity.

5. Research Opportunities

Beyond the research described above, we illustrate a number of research opportunities as follows.

Utilizing other metadata fields. We currently only leverage timestamp and Microsoft detection reports. There are many other metadata fields. Conducting data mining on these fields could enable many other "big security" applications. For example, there are existing techniques that leverage submission_id information to identify malware writers [5]. For example, future research could leverage ssdeep information to cluster malwares and conduct malware prediction and hot malware mining using a finer granularity.

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 evaluat all those reports systematically and to better combine reports from different vendors for future work.

Improving antivirus products. There are also 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.

Studying other types of malicious files. Besides PE files, there are other types of malicious files on VirusTotal such as malicious apps, malicious URL, and malicious binary files on other systems. What the characteristics of these files are and whether they follow the same patterns as PE files remain open issues.

Leveraging other information on VirusTotal. Besides the static information we 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.

6. Related works

Many research efforts [1, 3, 7, 10, 11] 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 [10] can fill uncompleted programs with call innovations by using statistical models trained from extracted sequences of API calls from large code bases. JSNICE [11] 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. [7] 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 [5, 12] 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

Data on VirusTotal has largely been ignored by research community. In this paper, we conduct an empirical study on PE malwares on VirusTotal. Our study is mainly performed to understand the temporal characteristics and the family distribution characteristics of malwares. We also build two techniques, cache-based malware prediction and hot malware family mining, to validate our observations. 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] P. Bielik, V. Raychev, and M. Vechev. Programming with "Big Code": Lessons, Techniques and Applications. In *SNAPL*, 2015.

[4] 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.

[5] 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.

[6] P. J. Guo, T. Zimmermann, N. Nagappan, and B. Murphy. Characterizing and predicting which bugs get fixed: An empirical study of Microsoft Windows. In *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering - Volume 1*, ICSE '10, pages 495–504, New York, NY, USA, 2010. ACM. ISBN 978-1-60558-719-6. . URL http://doi.acm.org/10.1145/1806799.1806871.

[7] 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.

[8] 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.

[9] 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.

[10] 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.

[11] 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.

[12] 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/.