Learning from Big Security

Abstract

TODO

1. Introduction

Computer malwares are always big threats to individual and organizational users. For example, a worn or virus can cause system crash leading to data loss; a trojan or a malicious URL can steal user's sensitive information leading to economic loss. In 20xx, the loss cause by xxx is about xxxx. Effort to combating malwares have never stopped, and industrial companies and research communities have made various achievements. **??**

Malwares are still increasing exponentially now, but our effort does scale with the rapidly worsening situation. For example, the volume of mobile malwares tripled in 2015, while the revenue spent on malware research increased only by about 10% each year. To focus the effort in anti-malware research and engineering, systematic measurement studies are necessary to guide practices. For example, if a anti-malware company can know the malwares that will be prevailing in the current or next time period, they would be able to focus on these malwares. However, there are few of such studies to reveal malware characteristics and the characterization methodology now.

In this paper, we make an empirical study of a malware dataset from VirusTotal. VisualTotal is a free online service using antivirus engines from security companies to identify submitted suspicious malwares. We first made use of VirusTotal public API to collect a malware dataset in November, 2015. Our dataset collection focuses on Windows executables and libraries and their analysis result by Microsoft antivirus engine to guarantee accuracy.

We perform an analysis on the characteristics of the VirusTotal dataset. In addition to general characteristics such as malware family (i.e. category, defined in Section xx) distribution and the generation rate of new malware families, we specifically look into the temporal characteristics of the malware dataset, and find that there is obvious time locality. That is, current malware families are more likely to appear in the next time period.

We then made a study of hot malware family mining algorithms. We compare the performance of frequent item mining algorithm with exhaustive counting algorithm. Our results show that, we acceptable penalty the frequent item mining algorithm gives satisfactory predictive result in terms of memory space.

<<Wenfei: edit until here>>

As a free online service, VirusTotal [1] analyzes files submitted by real-world users to identify many different kinds of malwares, like viruses, worms, trojans, and so on. VirusTotal applies different antivirus engines to each submitted file and generate an aggregated reports. All submitted files and generated reports are saved and can be accessed through public API.

The repository on VirusTotal provides a good source to conduct data mining. Firstly, there are huge amount of data on VirusTotal. Figure 1 shows that there were more than 40 million suspicious files submitted last November. This amount of data makes VirusTotal a rough estimation of malwares in the real world. Secondly, all data on VirusTotal are labeled by state-of-the-art antivirus techniques. VirusTotal updates each antivirus engine every 5 minutes. Besides whether a given a submitted file is detected by an antivirus engine, VirusTotal also keeps exact detection tag returned by each engine. There are also online active malware researchers, who can comment and vote each submitted file and serve as an important supplement of antivirus engines. We believe mining data on VirusTotal could enable many "Big Security" applications.

In industry, antivirus vendors widely use VirusTotal to identify false negatives and false positives of their products. They only utilize VirusTotal reports separately for each single suspicious file, but fail to leverage the whole repository. In academia, researchers began to pay attention to correlations among different VirusTotal reports. For example, **[ToDo: discuss Heqing's work]** We believe there are much more research opportunities through mining VirusTotal.

In this paper, we view data on VirusTotal as a stream, based on each files submission time, and design two stream mining applications: *hot malware family mining* and *malware prediction*. There are possibly infinite malware families. Hot malware family mining can precisely identify malware families, which occupy more than a given percentage of total malwares, by using a constant number of counters. Malwares does not appear uniformly across different malware families or across time, and they appear in bursts. We built a cache-based algorithm to predict malwares in which families would appear in the near future.

In summary, we made the following contributions in this paper:

- We collect data submitted to VirusTotal last November, and briefly analyze these data to understand the characteristics of VirusTotal repository (Section 2).

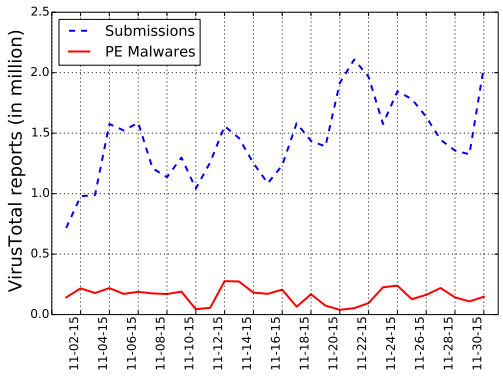

Figure 1: The number of files submitted to VirusTotal last November.

- We build two stream mining applications, one could identify hot malware family in a constant number of counters (Section 3), and the other could predict malwares in the near future (Section 4). Experimental results show that we can cover all hot malware family with a very few false positives, and we can predict future malwares with a high precision.
- We discuss the future research opportunities through mining data on VirusTotal (Section 5), and demonstrate the feasibility by using our experience of building stream mining applications.

2. Methodology

We study all PE malwares submitted in November of 2015. In this section, we will firstly discuss how we collect data, and then we will present the general characteristics we observe.

2.1 Data collection

Metadata Fields	Explanation
name	file name of the submitted sample
timestamp	timestamp when the submission is conducted
source_country	the country where the submission is conducted
source_id	user id who conducts 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 about **positives** fields
report	detailed detection report from each AV engine

Table 1: Metadata fields of each submission got from VirusTotal private API.

We download submission reports' metadata through private API of VirusTotal. Table 1 shows all metadata fields. For one report, if its tag field contains either "peexe" or "pedll", we consider the report is about a PE file. It is possible that VirusTotal private API returns redundant reports, and we use the combination of md5 and timestamp to detect and merge redundant reports. We only rely on Microsoft antivirus engine to judge whether a submission is malicious or not, and which malware family the submitted malware belongs to. In total, we collect 43308091 reports and 4732502 PE malwares submitted in November of 2015. The numbers of reports and malwares submitted each day are shown in Figure 1.

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

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 miss some malwares submitted to VirusTotal. We simply leverage Microsoft antivirus engine to decide whether one 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 scope of this paper. Although there are huge mount of malwares on VirusTotal, we do believe that 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 malwares in our study provide a representative malware sample of the real world.

2.2 General characteristics

We observe three characteristics after analyzing data we collect:

Observation 1: most malwares are submitted only once to VirusTotal. We have collected 4732502 PE malware submissions, and there are in total 4038647 distinct PE malwares. On average, each PE malware is submitted 1.17 times to VirusTotal. We believe that most malwares are encountered by more than one VirusTotal user. We think this observation is likely to be caused by the fact that VirusTotal users tend to check whether their samples have already been submitted, before conducting their submissions, and the number of submissions is not a good indicator for malware popularity.

Observation 2: 100 - 400 new malware families would appear each day. Figure 2 shows new malwares appeared in November of 2015. We do not have any data before Nov. 1st, so there are much more new malware families in the first few days. After that, the number of new malware families is roughly from 100 to 400. In total, there are 11311 malware families.

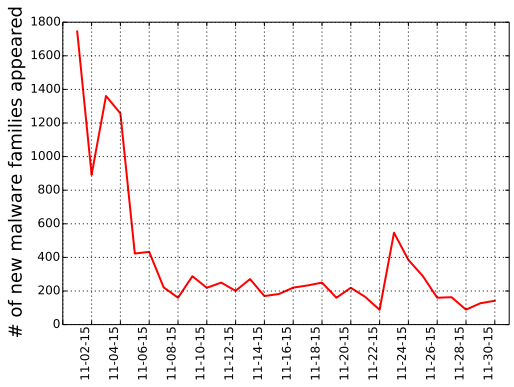

Figure 2: The number of new malware families we observed each day in November of 2015.

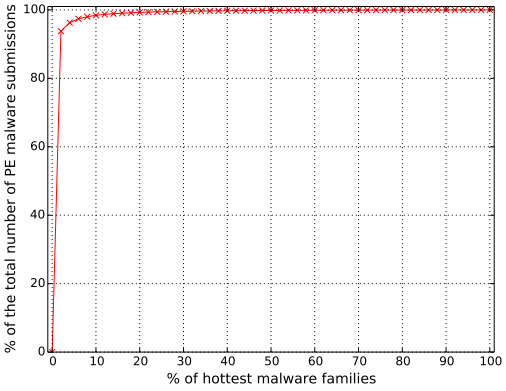

Figure 3: Skewness of malware families appearing in November of 2015.

Observation 3: The distribution of malware families are highly skewed. As shown in Figure 3, only small number of malware families are hot. The distribution of malware families follows the well-known Pareto principle, and more than 90% malware families take place in only 10% malwares. The skewness of malware family distribution suggests that we could conduct effective hot malware family mining.

3. Hot Malware Family Mining

In this section, we view data on VirusTotal as a stream, according to each submission's timestamp, and apply a frequent item mining algorithm to identify hot malware family.

3.1 Overview

Frequent item mining algorithms take two configuration parameters, ϕ and ε, where $\phi > \varepsilon$. The goal of frequent item mining algorithms are to provide 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 are all items which appear more than $\lfloor \phi N \rfloor$ times, and no items which appear less than $\lfloor \varepsilon N \rfloor$ times.

The frequent item mining algorithm we use is space saving algorithm [3], which was proposed for streams in Internet advertising, and has already been applied in other areas, like mining hot calling contexts in profilers [2]. Space saving algorithm tracks $M = 1/\varepsilon$ pairs of (f, c). f is short for malware family, and c is short for counter. Pair content 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 comes, if the malware family is already under monitoring, the related counter will be increased by 1. And if the malware family is not monitored, we will replace the malware family of the pair with 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.

3.2 Evaluation

We implement the space saving algorithm by using python-2.7. We process the downloaded VirusTotal submission data into a stream, based on each submission's timestamp. Our experiment is conducted on an aws c4.4xlarge virtual machine, which contains 16 virtual cpus and 30G memory.

Following previous works in frequent item mining [2], we measure the following metrics by using 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 and is used to measure largest frequency of malware families not covered in (ϕ, ε)-*HMF*. It is defined as follows:

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

3. *aveUncover* is short for average frequency of uncovered malware families and it is defined similar to *maxUncover*.

4. *False positives* are defined as malware families returned during 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.

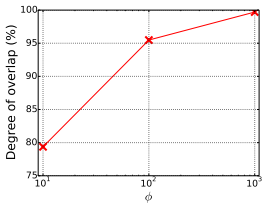

Figure 4: Relation between ϕ and degree of overlap.

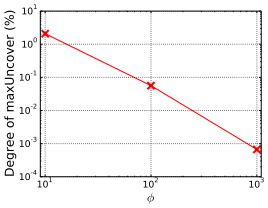

Figure 5: Relation between ϕ and degree of maxUncover.

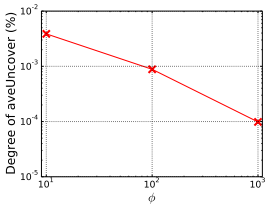

Figure 6: Relation between ϕ and degree of aveUncover.

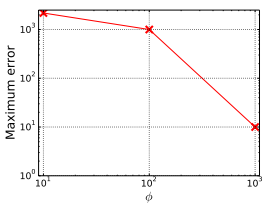

Figure 7: Relation between ϕ and maximum error.

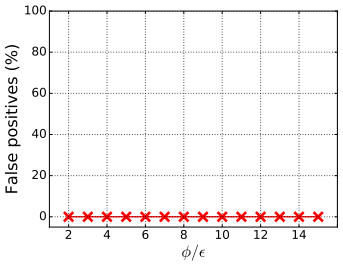

Figure 8: False positives in (ϕ,ε)-*HMF* as a function of ε. The value of ϕ is fixed to 10^{-2}.

5. *maxError* is used to measure relative error of counter values, compared with 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 of applying space saving algorithm, we set $\varepsilon = \phi/5$ as default, unless we explicitly state.

We first evaluate how overlap would change after we change ϕ. Overlap is used to describe how many malwares are monitored in (ϕ,ε)-*HMF*, the larger the better. As shown by Figure 4, after we change ϕ from 10 to 100, overlap will increase from 79.93% to 95.48%, and overlap will further increase 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 they are the lower the better. As shown by Figure 5 and Figure 6, both maxUncover and aveUncover will decrease by an order, as we increase ϕ value by an order.

Figure 7 shows how maxError change, after we change ϕ. maxError describes how precise counters in (ϕ,ε)-*HMF* are, and it is the lower the better. maxError would drop from 2178 to 999, after we change the value of ϕ from 10 to 1000, and maxError value would become to 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 the frequency of a new malware family is one more than the least counter value of all monitored malware family, when it conducts eviction.

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 8, space saving algorithm will constantly report 0 false positives, during our experiments.

3.3 Discussion

As discussed in Section 2, the distribution of malware families is 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 our solution can identify hot malware families from possibly infinite malware families by only using constant memory.

Malware family is a relatively coarse granularity. In the future, we could consider how to conduct stream mining in a finer granularity. Almost all malware samples are only submitted once to VirusTotal (Section 2), so mining hot submitted malware samples would not work. Ssdeep value for each submitted malware sample can also be queried from VirusTotal, and the edit distance between two ssdeep values can be used to measure similarity between the two malware samples. How to leverage ssdeep values to cluster malware samples, and conduct stream mining based on cluster information remains an open issue.

4. Malware Prediction

4.1 Overview

Resources to combat malwares are limited. Any techniques that could allow antivirus vendors to focus their effort would be great. In this section, we build a solution, which can predict malwares belonging to which families would appear in the near future. The basic intuition behind our technique is that malwares do not appear uniformly across different

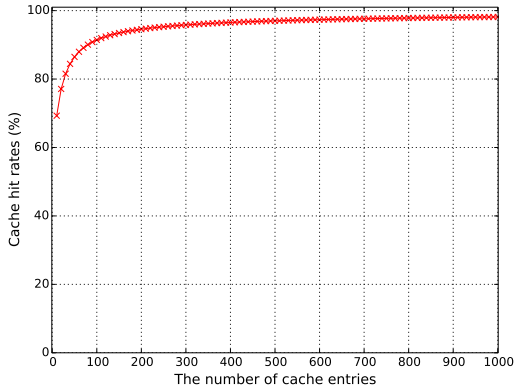

Figure 9: Relation between cache hit rate and cache size.

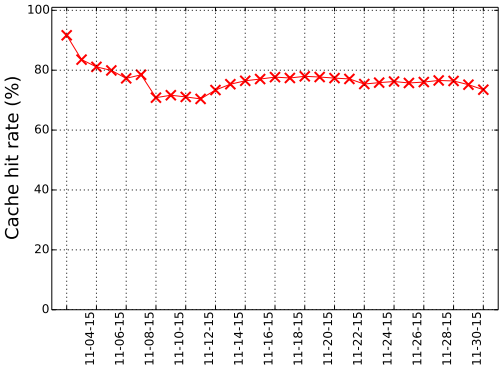

Figure 10: Cache hit rate during per-day update.

family and time, but they appear in bursts. We borrow cache mechanism from system areas to conduct our prediction.

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

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

4.2 Evaluation

We implement our cache by using python-2.7, and conduct the experiment on the same machine as in Section 3.2. We measure hit rate to evaluate our solution, and hit rate is calculated as:

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

We conduct two experiments. In the first experiment, we explore how cache hit rate changes with the number of cache entries. We change cache size from 10 to 1000. As shown in Figure 9, the rate rate will grow from 69.29% to 98.14%. By using more than 80 cache entries, the cache hit rate will be above 90%, and by using more than 230 cache entries, the cache hit rate will be above 95%.

In the second experiment, we fix the cache size to 100, and we use the cache content got from previous day to calculate cache hit rate, and only update cache content at the end of each day. Figure 10 show the cache hit rate each day. Most cache hit rate falls between 70% and 80%.

4.3 Discussion

There are many other cache block size, pre-fetch mechanism, and replacement policy proposed in previous works. We leave explore the effect of their combinations in the future. As we discussed in Section 3.3, we could also explore how to change the prediction granularity from malware family to ssdeep-based clusters.

5. Research Opportunities

1. How to utilize other fields of metadata.
2. How to improve the tag.
3. What are the characteristics of other type of malware.
4. Improve the current security solution.

6. Related works

Learning from Big Code. 1. Mining data from virustotal is a special case of mining from software repository. a. Data is not the same from traditional software repository mining, binary code vs source code

2. Many security problems could also be identified through mining from software repository.

Previous work on leveraging data on virus total.

Previous streaming mining work

7. Conclusion

References

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

[2] 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`.

[3] 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.