Detecting emerging malware in the cloud before VirusTotal can see it

Thanh Nguyen[1], Gan Feng[1], Andreas Pfadler[2], Anastasia Poliakova[1], Ali Fakeri-Tabrizi[1], Hongliang Liu[1], and Yuriy Yuzifovich[1]

[1] *Security Innovation Labs - Alibaba Cloud,* [2] *Alibaba DAMO Academy*

This paper was presented at Botconf 2022, Nantes, 26-29 April 2022, www.botconf.eu
It is published in the Journal on Cybercrime & Digital Investigations by CECyF, https://journal.cecyf.fr/ojs

Abstract

In this paper, we present a new methodology to discover emerging malware where new malware candidates are continuously discovered by our general anomaly detection, and the graph learning system predicts the behavior and the threat family using fuzzy similarity to support further analysis by the security researchers, or for the automatic enforcement and remediation. This methodology can be applied at large scale to detect and analyze emerging malware while providing rich contextual information.

Keywords: fuzzy hash, ssdeep, similarity graph, algorithms.

1 Introduction

Fuzzy hashing is widely used by malware analysts to analyze emerging malware [1, 2, 3, 4, 5, 6, 7, 8]. Can machine intelligence perform this analysis automatically and at scale?

VirusTotal and other threat intelligence providers give excellent visibility into evolving threats over the years, which we all appreciate. However, one concern remains: what if there is no "patient 0" who can notify VirusTotal of a new threat? How many attacks would we miss as a result? Alibaba Cloud has a dominant position in the Asia Pacific market for the cloud services, and we detect and block a large number of regional cyberattacks and discover new malware samples on a daily basis. Many of these attacks and samples are either not reported at all or are reported several days late on VirusTotal. Meanwhile, advanced malware techniques like polymorphic can bypass the cryptographic hashing signature-based threat intelligence lookups. Thus, the market has asked us a question: can we detect new malware before VirusTotal can see them?

In this paper, we present our approach to automatically detect and predict newly emerging malware within the cloud provider infrastructure using fuzzy hashing, a graph database, and graph algorithms, before VirusTotal or any other 3rd party detection engine can report it to the community.

Fuzzy hashing, combined with the similarity function, enables the quantification of the similarity between two malware samples. For example, ssdeep algorithm, the de facto standard among context-triggered piece-wise hashing algorithms (CTPH), uses a modified Levenshtein edit distance to describe the malware sample content similarity. Using ssdeep and the edit distance, we constructed a large-scale malware similarity graph and applied sub-graph pattern discovery and clustering graph algorithms. Connecting a pair of graph nodes corresponding to malware samples with a high edit distance similarity, we automated the emerging malware discovery. Not only we built a static graph off our collection of malware samples and our ssdeep repository, we also constructed a pipeline that is continuously updating our similarity graph with newly observed samples to detect emerging malware and predict their family and behavior.

We contribute by introducing a general graph-based framework of the malware binary similarity analysis, as well as graph algorithms-based automated discovery. This framework proved to be useful for modeling malware sample similarity in graphs

without being dependent on any specific fuzzy hashing algorithm. We performed engineering optimizations of the pairwise ssdeep computation to dramatically reduce the cloud computing costs which effectively enabled us to accomplish our goal to have a production-grade pipeline. These optimizations relied on the sparsity of the graph resulted from our focus only on medium to high similarity scoring.

The paper is organized as follows. First, we review the fuzzy hashing algorithms with an emphasis on ssdeep. Then we construct the malware similarity graph and apply graph algorithms for emerging sample discovery. We also elaborate on the engineering optimization to ensure that our large-scale graph and its regular updates with new samples are implemented cost-effectively. After that, we provide a few case studies that include polymorphic malware, ransomware, cryptocurrency miners and other malware families. Finally, we conclude the paper with ideas and research directions to extend our work in the future.

2 Fuzzy Hashing and ssdeep

2.1 Introduction to Fuzzy Hashing

Ten years ago, traditional static analysis using cryptographic MD5 and SHA256 hashes was “the most commonly used technique in malware research". However, as more malware writers have used polymorphism and reused the source code, this traditional approach has reached its limits. Hence, there has been a recent trend towards fuzzy hashing, or more specifically the adoption of Similarity Preserving Hashes (SPD) as in e.g. [1, 2, 3, 4, 5, 6, 7, 8]. Although these techniques differ in their effectiveness and computational costs, they share two key steps for SPD: feature selection and digest generation (See Fig.1). Feature selection extracts features such as a bag of n-grams or a rolling hash over sliding windows, while digest generation creates a more compact presentation for the selected features [6].

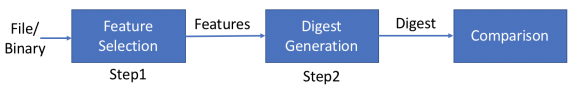

Figure 1: Fuzzy hash calculation

Popular Fuzzy Hashes. Among current fuzzy hashing methods, ssdeep [9] has gained enormous popularity in malware detection and analysis, especially since there is an open source implementation available. ssdeep excels at detecting minor changes, but may be vulnerable when content is swapped or fragmented. Sdhash [10] uses fixed length blocks (64-byte) with high entropy, which are then converted to a Bloom filter bitmap. Sdhash is good at handling containment and cross-sharing but suffers from high computational complexity. TLSH [11] uses a bag of triplets mapped into a 32-byte container and is more robust to random changes and adversarial manipulations than ssdeep or sdhash. However, TLSH focuses on the fragmentation and resemblance detection, and does not perform well for the containment detection and is computation-heavy. MvHash [12] calculates a majority vote for each input byte within a corresponding neighborhood, then transforms the byte sequences using run length encoding (RLE) before finally computing a rolling hash over a 7-byte sliding window, where the final digest is a sequence of Bloom filters. Although this method is generally faster than sdhash, it is possible for an attacker to manipulate it to cause the similarity score to approach zero even when the objects are very similar[13]. LZJD [14] is more accurate, yet expensive with respect to the storage space, requiring roughly 50 times more space when compared to ssdeep.

2.2 ssdeep Digest Generation

ssdeep uses context triggered piecewise hashing (CTPH) and a rolling hash algorithm to determine when blocks start and stop [9]. The figure below illustrates how to generate an ssdeep hash: first, the binary is split into the sequence of chunks based on the trigger points (instead of fixed-sized blocks), then each chunk is hashed into smaller number of bits and placed sequentially into small containers. ssdeep can match inputs that have homologies, where strings have sequences of identical bytes in the same order, although bytes in between these sequences may be different in both the content and length.

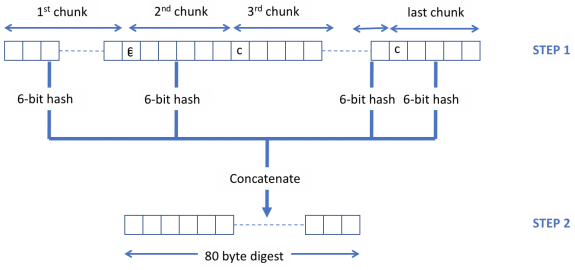

Figure 2: ssdeep calculation

A literal ssdeep hash consists of a block size and two sets of base-64 hashes separated by a colon, as shown in Fig.3 where we present examples of ssdeep hashes for Phoenix miner variants.

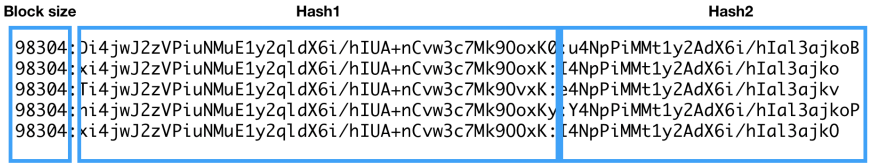

Figure 3: Example of Phoenix miner's[15] ssdeep fuzzy hash [16]

3 The Similarity Graph

3.1 Graph Construction

At Alibaba Cloud, we collect MD5, SHA and ssdeep hashes of anomalous binary files. Over time, we have built a library of more than one hundred million ssdeep hashes collected from cloud instances that had security agent functionality enabled. To build the similarity graph, we used Levenshtein edit distance [17] as the similarity metric for pairs of ssdeep hashes within our hash library.

Mathematically, the Levenshtein distance $lev(a,b)$ between two strings a and b (of length $|a|$ and $|b|$ respectively) is defined in Fig.4, where the tail of a string x is a string of all but the first character of x, and $x[n]$ is the nth character of the string x.

$$lev(a,b) = \min \begin{cases} |a| & if\ |b| = 0 \\ |b| & if\ |a| = 0 \\ lev(tail(a), tail(b)) & if\ a[0] = b[0] \\ 1 + min \begin{cases} lev(tail(a), b) \\ lev(a, tail(b)) \\ lev(tail(a), tail(b)) \end{cases} & otherwise \end{cases}$$

Figure 4: Levenshtein edit distance.

In Fig.5 and Fig.6 we present examples of ssdeep hashes and their similarity scores for several variants of an actual crypto-miner and XOR.DDoS trojan malware. It is important to note that we have optimized the edit distance calculation and modified Wu's algorithm to speed up the entire distance calculation process (See more details in Section 4.3).

In the next step, the fuzzy hash similarity graph is constructed from all pairwise similarities using the Levenshtein edit distance. A graph G is typically defined as a tuple $G = (V, E)$ where V is a set of vertices (also called nodes or points) and $E = \{\{x, y\} | x, y \subset V, x \neq y\ and\ lev(x, y) > \epsilon\}$ is a set of edges (also called links or lines), which are unordered pairs of vertices (undirected graph). We add an edge between two vertices (ssdeep hashes) if their pairwise ssdeep similarity is higher than a threshold ϵ. We then enrich this graph nodes with attributes such as MD5, SHA1/SHA256 and labels with tags from 3rd party vendors such as VirusTotal and Avira.

It is worth noting that in this similarity graph framework ssdeep algorithm can be replaced by imphash [18], Lempel-Ziv Jaccard Distance (LZJD) [14] or any other fuzzy or forensic similarity hash, and most algorithms in our framework will remain operational.

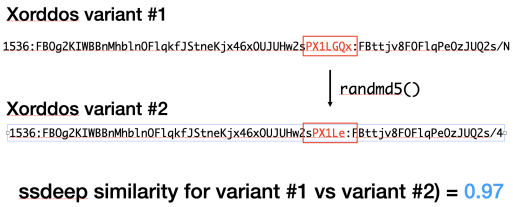

Figure 5: ssdeep and edit distance for XOR.DDoS.[16]

Figure 6: ssdeep and edit distance for miners.[16]

3.2 Graph as a Tool

With rich contextual information at the node and edge levels, graphs are an excellent tool to model and understand relational data. Moreover, there exists now a wide range of well understood algorithms, which help in uncovering previously unknown insights from data which has been modeled in a Graph.

Graphs support multiple traversal operations, in particular second and high order for graph traversal, and allow for modeling of information propagation. Furthermore there exists pattern-based matching algorithms which also allow for convenient and efficient querying of large graphs. As a central tool of discrete mathematics and machine learning, graphs thus offer a flexible and general basis for both classical, as well as modern machine learning based approaches, which allow one to learn implicit structure from the underlying data. For interesting and worthwhile "success stories" one may for instance consider PageRank[19], Graph Neural Networks (as for instance used in [20]), label propagation [21] or graph embedding based approaches such as e.g. node2vec [22].

With the advent of huge volumes of graph data collected e.g. in social networks or security systems such as ours, graph databases [23] and graph computing platform [24] have been developed to allow for storing large scale multi-facet security information, efficient querying, as well as the production de-

ployment of graph based machine learning algorithms [25, 26, 27, 22].

A key advantage of graph-based modeling is that individual data points (vertices) may naturally be considered together with their context defined via their k-hop neighborhoods (that is all neighbors within k steps along an edge). In Fig.7 we illustrate how a proper graph model may provide such context: An Anomaly Detection system reports the binary "X" (blue nodes) as a suspicious, and a graph-based learning system suggests that this binary has a high similarity with "Y" (red nodes). If we know somehow (through e.g. external confirmation by another system) that "Y" is malicious, then automatically the risk of maliciousness of "X" will increase. This situation could then be the starting point for applying a machine learning algorithm on this graph, such as e.g. label propagation [21], in order to obtain predicted maliciousness labels for all previously unlabelled nodes.

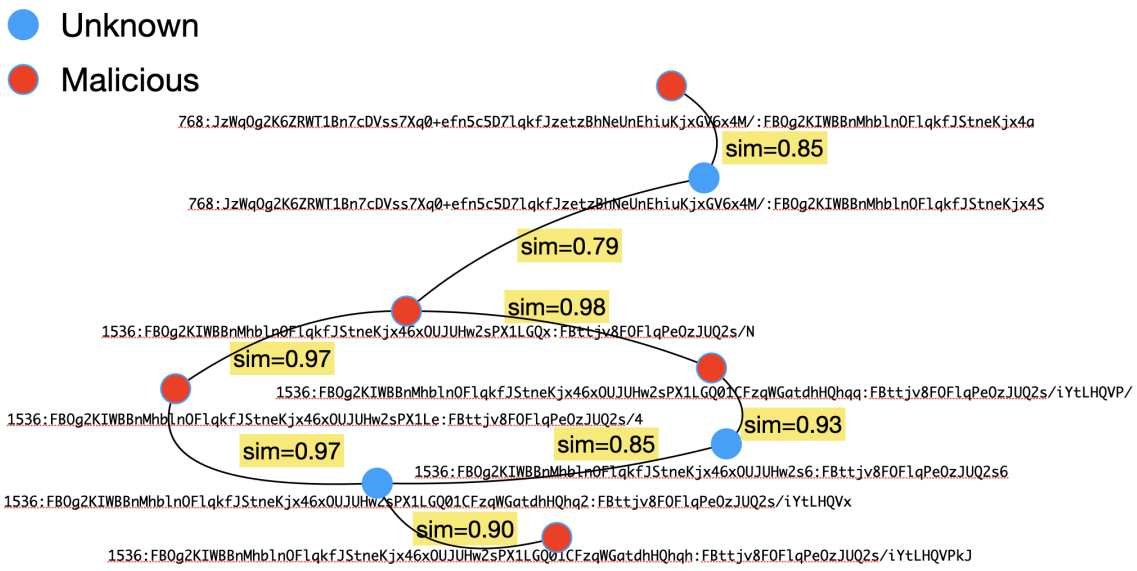

Figure 7: Example of a ssdeep similarity sub-graph [16]

Instead of focusing only on traversing along edges in a graph, one may also focus on graph patterns, that is sub-graphs of certain types. Different sub-graph patterns and topologies can be learned and extracted from e.g. our similarity graph, where we present some examples in Fig.8. In our experience, graph patterns have been a powerful tool, helping us to map new findings to one or a group of already known activities.

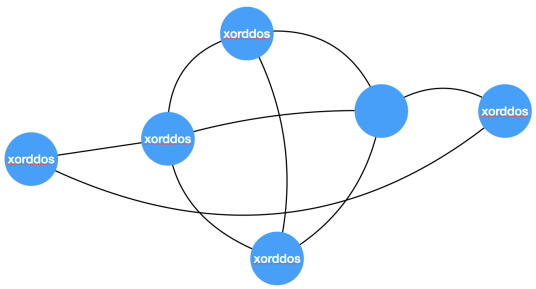

Figure 8: Example of sub-graph patterns

Graph Clustering. Based on the similarity graph, we can apply algorithms such as graph pattern finding, graph-based clustering and community detection to automatically learn and extract patterns in an unsupervised way [28, 29, 30, 31]. Label Propagation Algorithm [21] can be used to predict labels for clusters or patterns. We will present an example for this particular application in the next section.

4 Production Deployment and Optimization

We will now describe how our similarity graph-based detection system for emerging malware has been deployed to production and provide more details on the key challenges we overcame while scaling up our pipeline.

In our system we have implemented the following steps:

1. Collection of ssdeep hashes: For any newly observed candidate binary flagged by our existing anomaly detection system, we compute its ssdeep hash and add it to our existing library of ssdeep hashes maintained in our internal data warehouse system.
2. Computation of pairwise similarities between ssdeep hashes and the construction of the ssdeep

similarity graph G.

3. Cluster the graph G and apply the Label Propagation Algorithm in order to predict labels for as of yet unlabeled

In the remainder of this subsection we will share more engineering details and briefly describe the performance optimizations we accomplished.

4.1 Choice of ssdeep

Research results indicate that LZJD is a more effective fuzzy hash algorithm than ssdeep, with a better ability to match any related file fragments, while ssdeep is generally considered more practical, having been applied previously in a number of threat intelligence systems [9, 32].

One LZJD hash contains 1024 4-bytes integer minimal hashes, which would amount to a storage cost of 4096 bytes per hash. Meanwhile, the average length of hundreds of millions ssdeep hashes is only 75.5 bytes. Hence, when compared to ssdeep, LZJD would require more than 50 times the storage space. At our scale of hundreds of millions of binaries in the library, with hundreds of thousands of new binaries added every day, ssdeep was the best choice. This decision significantly reduced the amount of CPU and storage resources needed to run our pipeline in production.

4.2 Optimizing ssdeep

There have been many efforts to optimize ssdeep for large scale deployments [32]. Given our scalability requirements, optimal implementation of the ssdeep hash computation has been one of the critical priorities of our project. First, we opted to use the C-language implementation from libfuzzy instead of a previously used Java-based implementation. This has enabled us to reduce the overall runtime for the entire ssdeep computation process by about 83% and enabled us to use 91% fewer computing resources in our internal big data system.

We run our ssdeep computation in a map-reduce framework on top of our internal data warehouse. To fully utilize parallelism we split the ssdeep computation across the map and reduce operations. This also allowed us to efficiently implement a pre- and post-filtering of irrelevant hashes in the map and reduce stages, as well as clean up redundant characters.

Further optimizations include cleaning up redundant characters, pre-filtering and early stopping with a block size ratio filter and a 7-gram block heap. Given the fact that we are mostly interested in finding high similarity score only (to increase the sparsity of the graph), early stopping helps with reducing the number of pairs for which an ssdeep similarity is to be computed.

Beside leveraging pre-filtered block size range and 7-gram block heap, we also optimized the edit distance algorithm to reduce the running time from $O(N^2)$ (naive approach) to *linear* time using bit-wise Manber and Wu algorithm [1], which has been used in Unix *diff*. We went further by mapping from the original edit distance to a bit-string Longest Common Sub-sequence problem for additional speed up and made significant improvement of the running time again [33].

In our benchmark, these optimizations had a cumulative improvement of being 5.5x faster and 10x less computationally intensive, when compared with our initial naive implementation.

4.3 Reducing the Search Space

With about 100 million ssdeep hashes in our sample database, there are 10 quadrillion pairs to be compared, which is practically impossible to calculate at a reasonable cost, or regularly. Luckily, with our in-house General purposed Anomaly Detection (GAD) engine designed to capture any type of anomalous activities to discover potential IoCs (indicators of compromise), we were able to successfully reduce the number of pairwise comparisons to a manageable level.

4.4 Fast Clustering

Due to the pre-filtering and early stopping of ssdeep calculation, some pairwise similarities may not necessarily exist. For example, $sim(A, B)$ and $sim(A, C)$ may exist, while $sim(B, C)$ does not exist or its similarity is below the threshold. We therefore applied the parallel single linkage clustering algorithm with fast-ssdeep-clustering [34], which can deal with missing links and greatly reduces the overall computation time for the clustering step. We also leveraged our distributed cloud computing platform to perform a map reduce-based label propagation algorithm (LPA) [21] to find communities in a graph and cluster similar samples via multiple iterations. The source binary with minimum distance is maintained via multiple iterations and is used for cluster identification.

4.5 Additional Features and Cluster Validation

Observed clusters tend to be noisy. We use several additional features to validate and identify cohesive clusters that are close to identified malicious samples. The features include the average edit distance similarity and block size variance, filename and process path similarity, counts of hidden files or hidden paths, [1] "deleted" labels [2], the validation ratio based on MD5, etc. "Good" clusters should have high average similarity score, small block size variance, with many hidden paths or files, relate to binaries found in the

[1] Files and folders can be hidden in Linux for many reasons. For example, users don't accidentally modify the contents of these files, hidden for privacy issues, etc. Hidden file or folder starts with a. dot(.), and are often applied for configuration files and logs.

[2] meaning a file was automatically deleted by local antivirus engines from the cloud instances

same folder, same filename or filename with similar patterns[3]. Last but not least, the MD5 validation ratio from the 3rd party vendors must be greater than 0.5.

5 Result validation

5.1 Dataset

We created a ssdeep similarity graph dataset for self-evolving trojans and a few other malware families such as Mirai, Agent Tesla, Heodo, etc. The graph has more than two hundred thousands nodes and twenty millions of edges. The data is publicly available at the Github repository in the Appendix.

5.2 Comparing results with 3rd party security vendors

We analyze the detection result with many 3rd party detection engines such as Avira and VirusTotal, the online search engine. From 211,071 unique binaries detected for self-evolving trojans, Avira can detect 153,925 binaries (72.9%). It is interesting that VirusTotal detects only **1.2%** of these binaries. Although VirusTotal combines results from different vendors, the shared data were potentially small snapshots and subsets.

To ensure the quality of our work, we validate the finding samples on the instances that receive general security alerts from Alibaba cloud security center such as weak password, cracking, intrusion, vulnerability, etc. Indeed, we see a good overlap (75%) on the instances that receive other alerts coming from the cloud security center.

5.3 A Real Case Study

XOR.DDoS is one of the most widespread Linux malware families. Discovered in 2014, this malware has changed many hats so far. The original purpose of XOR.DDoS was to build a bot-based infrastructure and to launch DDoS attacks. As time passed, XOR.DDoS writers expanded its capabilities and started to target Windows machines as well as Linux. Not long ago this malware has adopted Docker service exploitation and added detection evasion mechanism. We found the variants of XOR.DDoS and calculated their ssdeep similarities as shown below.

The original XOR.DDoS file is an ELF executable file of the size of 247KB, MD5 `d6a6dee6afa6879b729a0af3cde7ff33`, SHA1 `47ed693d195558507e4258527f7d4d4968d34f38`, SHA256 `dba757c20fbc1d81566ef2877a9bfca9b3ddb84b9f04c0ca5ae668b7f40ea8c3`, ssdeep `6144:3SDFOrnwRgUbMisI6sdkH+M6hWOcy5KOZW7U6NCEqPdf/mqYG:2ZRgUY/fsJcO1KOiXfqPdeG`, and notably identified as malicious by VirusTotal as shown in Fig.9.

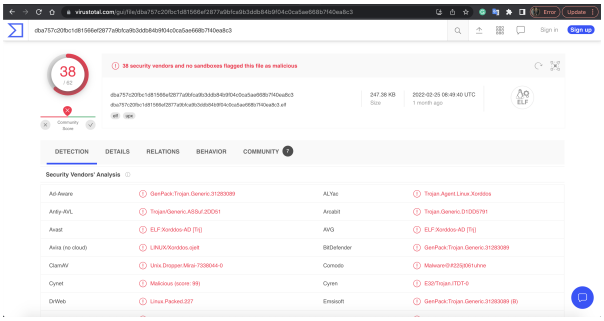

Figure 9: XOR.DDoS with VirusTotal detection [35]

We then found a new suspicious binary with the following MD5 by our anomaly detection model: `6a749f7b071e713affdcd759bc90707e`. The similarity score that can range from 0 (no common code) to 100 (binaries are identical), was 97 when compared with the known XOR.DDoS sample, which indicates a high similarity. While VirusTotal had no data for this particular MD5 at the time of our detection and classification, as shown in Fig.10, our algorithm proceeded with labeling this new binary as XOR.DDoS.

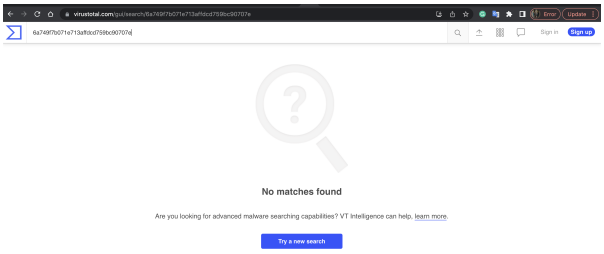

Figure 10: XOR.DDoS with no VirusTotal detection [36]

Because XOR.DDoS has been around for a while, so the pattern of its behaviour on the infected system is well known. One of the most important patterns is the algorithm XOR.DDoS uses to create its folder. It is typically located in /usr/bin/ and use ten random letters for the name of its directory.

```
/usr/bin/gsqykkwuag
/usr/bin/tldpjssjet
/usr/bin/nrhfapuwjp
/usr/bin/kgpeplprzq
/usr/bin/uhflpmyerd
```

According to the logs from the machine, a new binary was discovered and we can see that the new binary follows the same pattern of creating a new directory. Thus, we can say that the newly detected binary is XOR.DDoS variant and was discovered earlier than 3rd parties, including VirusTotal.

[3]We check if all filenames in the same cluster are either very similar or very dissimilar (high entropy) due to the distribution of random characters in the filenames of similar patterns.

5.4 Findings

To demonstrate the effectiveness of our approach, we are sharing several clusters of self-evolving trojans, miners on the cloud, ransomware, and others. ssdeep and associated graph-based clustering works well here due to the polymorphic nature of XOR DDoS and due to the strong code reuse practice by malware in general.

5.4.1 Self-evolving Trojan

XOR.DDoS is a self-evolving trojan. This malware family can update the binary content to result in a new SHA hash for each sample, as well as a new file name every time, to make it challenging for threat intelligence feeds and the intelligence community as a whole to detect and report all individual variants. Very few of them are identified by VirusTotal, while the *majority* of them are not. In fact, the commercial engines provide a variety of different malware names, such as `Trojan.Linux.GenericA.18093`, `Linux.Agent.253320`, `Linux.Packed.227`, `Linux.Agent.9!c`, etc., which cause further confusion. The dense clusters in Fig.11 shows the strong connections of many self-evolving trojans through ssdeep similarity.

Figure 11: The cluster of self-evolving trojans.

5.4.2 Miners on the Cloud

Bitcoin mining is the process of creating new bitcoins by solving computationally hard math problems that verify transactions and requires a significant amount of computing power. Despite the fact that mining itself is legal, it requires too much computing power to be efficient for cloud customers. Thus, appearance of any mining activity on cloud machine strongly indicates malware infection or insider abuse. Many cloud companies have warned that "malicious actors were observed performing cryptocurrency mining within compromised cloud instances". Recently, Google reported that 80% of the recent successful attacks on the customer infrastructure in their cloud computing service were used to perform cryptocurrency mining [37].

At Alibaba Cloud, we have observed and detected many cloud instances related to coin mining by leveraging our file hash similarity graph. Fig.13 shows the number of cloud instances and new binaries that are related to coin mining in early November.

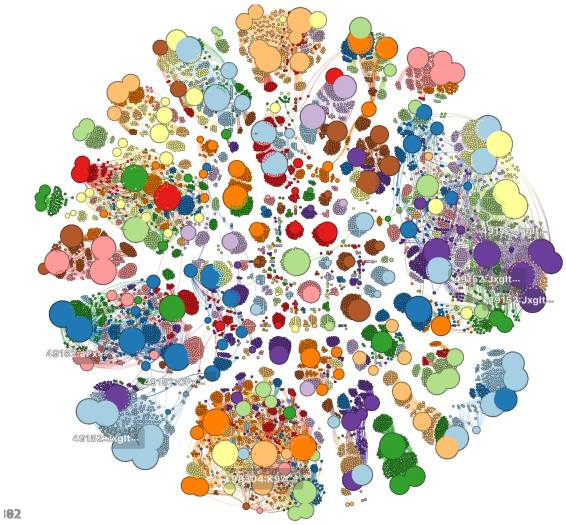

Figure 12: The miner cluster.

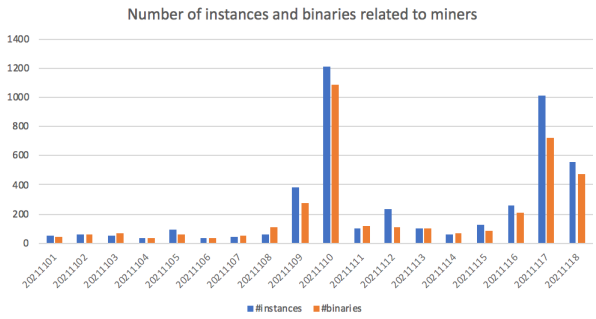

Figure 13: The miner on the cloud.

5.4.3 Ransomware

Ransomware is a malware designed to deny a user or organization access to files on their computers by encrypting these files and demanding a payment for the decryption key. Lately this malware family became one of the most popular malware. The success of this malware type encouraged malicious actors to adopt common practices of the malware protection and detection evasion, such as code modification. However, our method allowed to detect new ransomware samples by tracking the reused code. We see clusters where the ransomware shares common libraries such as

```
/usr/lib/libk5crypto.so.3.1
/usr/local/bin/regtest
```

```
{local env}/libaes.so
```

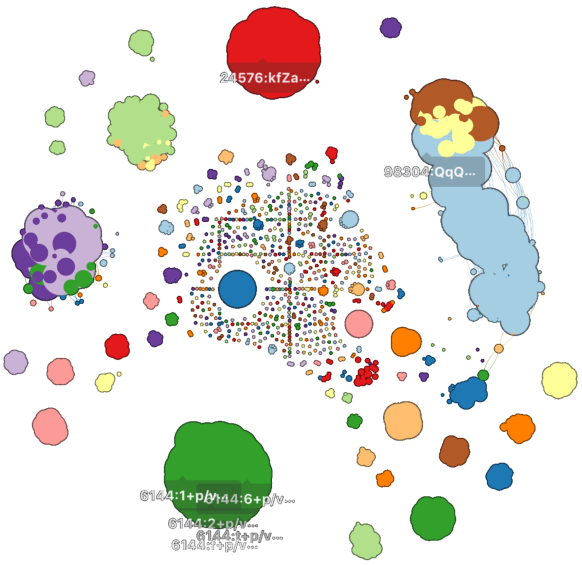

Figure 14: The ransomware cluster.

5.4.4 Mirai

While Mirai malware become one of the first "open source" botnets as its code was published on GitHub in 2016, there is no end-of-life planned any time soon. Simple yet efficient malware design provides malicious actors with great opportunities to modify the exploits it uses to get to the target every time a new remote code execution or a new code injection vulnerability is found. Even now we frequently observe this family in the wild.

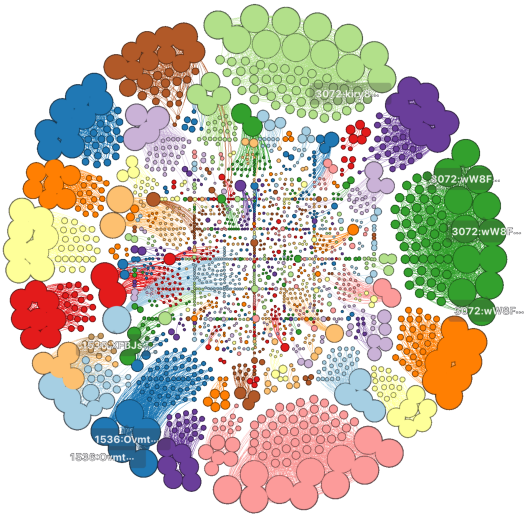

Figure 15: Mirai cluster.

5.4.5 Agent Tesla

Agent Tesla is an advanced RAT which functions as a keylogger and a password stealer. We observed that Agent Tesla spyware has lower ssdeep similarity, for which we can potentially apply different fuzzy hashing functions such as LZJD for improving the binary pairwise similarity. Nevertheless, overall pipeline and the graph framework still remains highly effective to detect new AgentTesla variants.

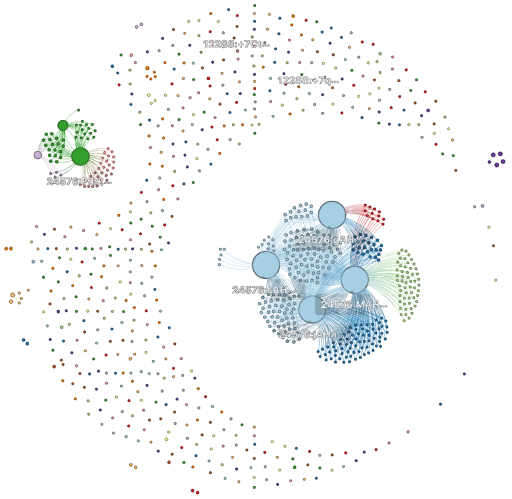

Figure 16: Agent Tesla cluster.

6 Conclusion

At Security Innovation Labs of Alibaba Cloud, ssdeep similarity graph is an important component in our multi-data plane approach connecting binary content similarity to other models using such data sources as DNS, HTTP and the honeypot data [38].

In this paper, we present the method, the components, and the way to orchestrate and optimize them to discover new malware in the cloud automatically. We believe that the machinery that we have built opens up the possibility for *larger scale* and *faster* emerging threat detection and analysis, and could improve the cybersecurity collaboration between the East and the West.

7 Acknowledgement

The authors would like to thank Georgy Okrokvertskhov for the support from his engineering team, and Amir Asiaee for his useful comments and his help evaluating the data. This research was supported by Security Innovation Lab and DAMO Academy of Alibaba Cloud.

Author details

Thanh Nguyen

Security Innovation Labs
Alibaba Cloud
t.nguyen@alibaba-inc.com

Gan Feng

Security Innovation Labs
Alibaba Cloud
fenggan.fg@alibaba-inc.com

Andreas Pfadler

DAMO Academy
Alibaba Group
andreaswernerrober@alibaba-inc.com

Anastasia Poliakova

Security Innovation Labs
Alibaba Cloud
poliakova.anastasi@alibaba-inc.com

Ali Fakeri

ali.fakeri@gmail.com

Hongliang Liu

Security Innovation Labs
Alibaba Cloud
hongliang.liu@alibaba-inc.com

Yuriy Yuzifovich

Security Innovation Labs
Alibaba Cloud
yuriy.yuzifovich@alibaba-inc.com

References

[1] V. Hugo G Moia and M. A. Amaral Henriques, "Similarity digest search: A survey and comparative analysis of strategies to perform known file filtering using approximate matching," *Security and Communication Networks*, vol. 2017, pp. 1–17, 09 2017.

[2] A. P. Namanya, I. Awan, J. P. Disso, and M. Younas, "Similarity hash based scoring of portable executable files for efficient malware detection in iot," *Future Gener. Comput. Syst.*, vol. 110, pp. 824–832, 2020.

[3] N. Sarantinos, C. Benzaid, O. Arabiat, and A. Al-Nemrat, "Forensic malware analysis: The value of fuzzy hashing algorithms in identifying similarities," pp. 1782–1787, 08 2016.

[4] B. Rahbarinia, M. Balduzzi, and R. Perdisci, "Exploring the long tail of (malicious) software downloads," *2017 47th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pp. 391–402, 2017.

[5] M. Botacin, V. Hugo G Moia, F. Ceschin, M. Henriques, and A. Grégio, "Understanding uses and misuses of similarity hashing functions for malware detection and family clustering in actual scenarios," *Forensic Science International: Digital Investigation*, vol. 38, p. 301220, 09 2021.

[6] L. Liebler and H. Baier, "Towards exact and inexact approximate matching of executable binaries," *Digital Investigation*, vol. 28, pp. S12–S21, 2019.

[7] S. Peiser, L. Friborg, and R. Scandariato, *JavaScript Malware Detection Using Locality Sensitive Hashing*, pp. 143–154. 09 2020.

[8] N. Naik, P. Jenkins, N. Savage, L. Yang, K. Naik, J. Song, T. Boongoen, and N. Iam-On, "Fuzzy hashing aided enhanced yara rules for malware triaging," pp. 1138–1145, 12 2020.

[9] V. Roussev, "An evaluation of forensic similarity hashes," *Digital Investigation*, vol. 8, 08 2011.

[10] F. Breitinger, H. Baier, and J. Beckingham, "Security and implementation analysis of the similarity digest sdhash," 08 2012.

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

[12] F. Breitinger, K. P. Astebøl, H. Baier, and C. Busch, "mvhash-b - a new approach for similarity preserving hashing," in *2013 Seventh International Conference on IT Security Incident Management and IT Forensics*, pp. 33–44, 2013.

[13] D. Chang, S. Sanadhya, and M. Singh, "Security analysis of mvhash-b similarity hashing," *Journal of Digital Forensics, Security and Law*, 01 2016.

[14] E. Raff and C. Nicholas, "Lempel-ziv jaccard distance, an effective alternative to ssdeep and sdhash," *Digital Investigation*, 08 2017.

[15] virustotal.com, "https://www.virustotal.com/gui/file/599393e258d8ba7b8f8633e20c651868258827d3a43a4d0712125bc487eabf92."

[16] A. Pfadler, A. Poliakova, G. Feng, T. Nguyen, A. Fakeri-Tabrizi, H. Liu, and Y. Yuzifovich, "Detect emerging malware on cloud before virustotal can see it," *Botconf*, 2021.

[17] V. I. Levenshtein, "Binary codes capable of correcting deletions, insertions, and reversals," *Soviet Physics Doklady*, 02 1966.

[18] N. Naik, P. Jenkins, N. Savage, L. Yang, T. Boongoen, and N. Iam-On, "Fuzzy-import hashing: A malware analysis approach," in *2020 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE)*, pp. 1–8, 2020.

[19] L. Page, S. Brin, R. Motwani, and T. Winograd, "The pagerank citation ranking: Bringing order to the web.," Technical Report 1999-66, Stanford InfoLab, November 1999. Previous number = SIDL-WP-1999-0120.

[20] W. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs," *Advances in neural information processing systems*, vol. 30, 2017.

[21] K. Berahmand, S. Haghani, M. Rostami, and Y. Li, "A new attributed graph clustering by using label propagation in complex networks," *Journal of King Saud University - Computer and Information Sciences*, 2020.

[22] A. Grover and J. Leskovec, "node2vec: Scalable feature learning for networks," *CoRR*, vol. abs/1607.00653, 2016.

[23] A. Bhattacharyya and D. Chakravarty, "Graph database: A survey," in *2020 International Conference on Computer, Electrical Communication Engineering (ICCECE)*, pp. 1–8, 2020.

[24] W. Fan, T. He, L. Lai, X. Li, Y. Li, Z. Li, Z. Qian, C. Tian, L. Wang, J. Xu, Y. Yao, Q. Yin, W. Yu, K. Zeng, K. Zhao, J. Zhou, D. Zhu, and R. Zhu, "Graphscope: A unified engine for big graph processing," *Proc. VLDB Endow.*, vol. 14, pp. 2879–2892, 2021.

[25] J. Wing, "Scenario graphs applied to security," 01 2005.

[26] C. Phillips, "A graph-based system for network-vulnerability analysis," in *in Proceedings of the 1998 workshop on New security paradigms*, pp. 71–79, ACM Press, 1998.

[27] X. Tao, Y. Liu, F. Zhao, C. Yang, and Y. Wang, "Graph database-based network security situation awareness data storage method," *EURASIP Journal on Wireless Communications and Networking*, 2018.

[28] B. Abu Jamous, R. Fa, and A. Nandi, *Graph Clustering*, pp. 227–246. 04 2015.

[29] J. Creusefond, "A comparison of graph clustering algorithms," 06 2015.

[30] Y. Peng, X. Zhu, F. Nie, W. Kong, and Y. Ge, "Fuzzy graph clustering," *Information Sciences*, vol. 571, 04 2021.

[31] B. Auffarth, "Spectral graph clustering," 01 2007.

[32] B. Wallace, "Optimizing ssdeep for use at scale.," technical report, Cylance, USA, November 2015.

[33] L. Allison and T. I. Dix, "A bit-string longest-common-subsequence algorithm," *Information Processing Letters*, vol. 23, no. 5, pp. 305–310, 1986.

[34] T. OI, "https://github.com/a4lg/ffuzzypp."

[35] virustotal.com, "https://www.virustotal.com/gui/file/dba757c20fbc1d81566ef2877a9bfca9b3ddb84b9f04c0ca5ae668b7f40ea8c3."

[36] virustotal.com, "https://www.virustotal.com/gui/search/6a749f7b071e713affdcd759bc90707e."

[37] "Cryptocurrency miners using hacked cloud accounts, google warns," *The Guardian*, 2021.

[38] A. Fakeri-Tabrizi, H. Liu, A. Polyakova, and Y. Einav, "Honeypot + graph learning + reasoning = scale up your emerging threat analysis," *Botconf*, 2020.

Appendix 1

Sample dataset can be found at `https://github.com/phunterlau/ssdeep-graph-dataset`