A pipeline for processing large datasets of potentially malicious binaries with rate-limited access to a cloud-based malware analysis platform

Dávid Maliga
CrySyS Lab
Budapest University of Technology and Economics
Budapest, Hungary
dmaliga@crysys.hu

Roland Nagy
CrySyS Lab
Budapest University of Technology and Economics
Budapest, Hungary
rnagy@crysys.hu

Levente Buttyán
CrySyS Lab
Budapest University of Technology and Economics
Budapest, Hungary
buttyan@crysys.hu

***Abstract*—In this paper, we present a pipeline that we designed for cleaning and processing large datasets of potentially malicious binaries using access to a rate-limited cloud-based malware analysis platform. Our goal is to efficiently filter out and discard benign files, to extract metadata from the remaining, likely-to-be-malware samples, and to create graph-based databases containing only metadata of verified malware. The main issue that we have to solve is the limited quota for accessing online malware analysis platforms that can be used for deciding about the maliciousness of a binary and obtaining metadata from static and dynamic analysis of samples. Our pipeline solves the problem by reaching a state where every sample in the database is either confirmed malware (based on its VirusTotal report) or similar to a confirmed malware with a minimal amount of requests made to the online platform. A database in such a state is already usable in practice, while confirming the malicious nature of and extracting metadata for all the samples in it can be continued in the background.**

***Index Terms*—malware datasets, metadata extraction, graph database, binary similarity**

I. Introduction

Malware (i.e., malicious software) is a long-standing problem in computer security. Indeed, malicious actors have created malware for many types of computers (including personal computers, servers in the cloud, smart phones, and, recently, embedded Internet of things (IoT) devices) for fun or for profit, causing substantial inconveniences and damage of different kinds for owners and operators of IT systems. In addition, there is an arms race between attackers and security professionals, in which attackers constantly create new pieces of malware that try to evade the malware detection techniques developed by security professionals. As a consequence, thousands of new malware samples appear every day, including brand new species and just variants of known families. To keep up with the pace of creating new malware, security professionals must invent novel, automated methods for malware detection. A popular recent approach is to use machine learning (ML) for this purpose.

ML-based malware detection requires large datasets of malicious and benign programs for training and testing ML models. For this reason, security professionals try to capture potentially malicious programs in large quantities with honeypots, and share the captured samples with others in the security community in the form of malware feeds or, sometimes, publish datasets containing captured samples for research purposes.

However, the quality of such feeds and datasets is often unclear; for instance, they may also contain benign files captured erroneously by the honeypots. Therefore, feeds and datasets of potentially malicious programs shared within the professional community or published for research purposes need to be cleaned, and preferably processed further, to be useful. Cleaning essentially means filtering out benign files and keeping only likely-to-be malware samples, whereas further processing tasks may include extracting metadata from the samples that can serve as features for ML-based techniques.

Both cleaning malware feeds and datasets and extraction of metadata from a large amount of samples can be supported by cloud-based malware analysis platforms such as VirusTotal[1], Any.Run[2], and FileScan.IO[3]. These online services allow for submitting suspicious files to them, which they analyze in different ways, and they respond with an analysis report from which one can figure out if the submitted sample was indeed a malware and extract various metadata of the sample. For instance, a VirusTotal (VT) analysis report contains the votes of multiple antivirus (AV) engines regarding the maliciousness of the submitted sample and it also contains information about the dynamic behavior of the sample when executed in a sandbox. Hence, a VT report can be used conveniently to decide if the sample was indeed a malware and to extract metadata that would otherwise be difficult to obtain just from the sample itself (e.g., by static analysis).

The problem that we address in this paper is that cloud-based malware analysis platforms typically limit the rate at which one can submit samples and request their analysis. The

[1]https://www.virustotal.com/, Last accessed: 2024.07.31
[2]https://any.run/, Last accessed: 2024.07.31
[3]https://filescan.io/, Last accessed: 2024.07.31

reasons for such rate limitations are that analyzing samples is demanding in terms of computational resources and platform operators want to preserve the availability of their services. So, when using cloud-based malware analysis platforms for cleaning feeds and large datasets of potentially malicious files and for extracting metadata of the samples in them, one faces the challenge that the process will last for an extended period of time, while, at the same time, it would often be useful to use the feed or the dataset as soon as possible. Imagine, for instance, an antivirus vendor that receives a malware feed from a honeypot operator: the antivirus vendor would obviously like to use the samples in the feed to retrain their ML-based malware detection model and distribute the retrained model to their clients as soon as possible. Perhaps, processing delay is not such a pressing issue for malware datasets published for research purposes; however, the issue here is that such datasets can be huge (much larger than daily feeds as they contain samples captured over a longer period of time), containing potentially hundreds of thousands of samples, and processing them may require months. So even in this case, one would like to reduce the time needed for being able to begin using the dataset.

In this paper, we propose a pipeline for cleaning and processing large datasets of potentially malicious binaries using rate-limited access to a cloud-based malware analysis platform. In the implementation of our pipeline, we used VirusTotal as the online analysis platform, but the design of our pipeline is general, not limited to VirusTotal, and it can be easily adapted to other online platforms too. In order to make a large dataset of potentially malicious samples usable as soon as possible, our pipeline tries to quickly reach a state in which every sample in the dataset being cleaned is either proven to be malware according to its VT report or similar to a sample which is proven to be malware. So, while in this state, we have no proven information for the latter kind of samples, they are likely to be malware too due to their similarity to samples known to be malware. We believe that a dataset in such a state is already usable. Moreover, reaching such a state requires only a limited amount of requests submitted to the online analysis platform (see the next paragraph for understanding why). Once this state is reached, we continue requesting analysis reports for all as-yet-unproven samples, while maintaining the property that every sample in the dataset being cleaned is either proven to be malware or similar to a sample which is proven to be malware. Hence, the dataset remains to be usable throughout the rest of the potentially long cleaning process.

The main idea of our design is first to construct a graph representing the dataset where the nodes correspond to the samples and two nodes are connected by an edge if the corresponding samples are similar; then to compute a dominating set[4] of the graph; and finally to request, from the online platform, the analysis reports of the samples corresponding to the dominating set. Thanks to the typically clustered nature of the graphs representing malware datasets, the dominating set is much smaller than the entire graph, resulting in a relatively small number of queries to the online platform with respect to the size of the entire dataset. Moreover, if the analysis reports for the samples corresponding to the dominating set indicate that they are indeed malware, then, by definition of the dominating set, we reached the state where every sample in the dataset is either proven to be malware or similar to a sample proven to be malware.

The rest of our paper is organized as follows: Section II introduces the tools and services we used to implement our pipeline. Section III gives an overview on how the pipeline works, while Section IV discusses the details of the algorithms we developed to clean and to process malware feeds or datasets. Section V describes how we validated our solution by using it to clean and to extract metadata from a large IoT malware dataset. Finally, Section VI concludes our paper.

II. Background

In this section, we briefly introduce the tools and services we use as building blocks of our pipeline and describe the kind of metadata we extract and store for each malware sample.

A. VirusTotal

Our pipeline uses VT for deciding if a given sample in a feed or dataset is malware. VT is a cloud-based malware analysis platform, where users can search for malware samples by hash value or upload suspicious files for deciding whether they are malware or not. When a sample is submitted for analysis, it is evaluated by multiple AV products and analyzed by additional static and dynamic analysis methods. The platform presents to the user the results of the analyses and how the different AV products categorized the sample. While VT provides this functionality for free, it is mainly intended to be used by human users. More intense interactions with VT using scripts is possible, but rate-limited: the public API allows only 500 requests per day and only 4 requests per minute. VT also has a subscription-based private API, which allows for more requests, but even with that, VT quotas cannot be considered limitless. So downloading VT reports for submitted samples is considered to be an expensive operation, and it is a bottleneck in every malware feed or dataset processing pipeline, including ours.

B. AVClass

AVClass[5] is a python script that can label large amount of malware samples, using, for example, VT reports. It is capable of aggregating the labels given by the different AV products.

[4]A dominating set D of a graph G is a subset of G's vertices such that each vertex of G is either in D or a neighbor of a vertex in D. Note that computing the minimum dominating set is an NP-hard problem, but there exist efficient greedy algorithms that compute a good approximation of the minimum dominating set (i.e., a sufficiently small dominating set).

[5]https://github.com/malicialab/avclass, Last accessed: 2024.07.31

By default, the output of AVClass will most likely contain malware family names, but it is capable of aggregating other tags assigned by the AV engines, which capture information about the nature of the malware (e.g., ransomware, dropper, trojan), behavioral information (e.g., spam, ddos), or properties of the files themselves (e.g., if they are packed). We use AVClass for extracting some metadata of the samples from their VT reports.

C. TLSH

TLSH stands for Trendmicro Locality Sensitive Hash, and it is a similarity digest scheme [1]. Similarly to cryptographic hash functions, it maps an input of arbitrary size to a fixed length output, but unlike cryptographic hash functions, it lacks the avalanche effect property. As a result, for similar inputs, TLSH gives similar hash values. Beside the hash computation algorithm, a difference score computation algorithm was published as well, which can be used to quantify the (dis)similarity of two TLSH hash values, and thus their corresponding inputs. A difference score of 0 means that the two inputs are identical or nearly identical, while the higher the score is, the less similar the inputs are.

In our pipeline, we use TLSH to determine similarity between malware samples in order to build a similarity graph[6]. In this graph, every node represents a malware sample, and two nodes are connected if and only if they are considered similar (i.e., the TLSH difference of their TLSH hash values is below a certain threshold). This similarity relationship is useful in the different stages of the processing, when not all information is available about every sample yet. In particular, we can infer certain properties of a sample (e.g., which malware family it may belong to) that has no available VT report yet by looking at the already available VT reports of similar samples.

D. Neo4j

We use Neo4j, a graph-based database management system, for storing the metadata extracted from the samples of a feed or dataset and the similarity relationships between the samples. Beside storing the structure of a graph (in our case, the similarity graph of the samples), Neo4j can also store metadata as attributes of nodes (representing samples) and edges (representing similarity relationships). Moreover, it can store large amounts of data in an ACID-compliant, transactional manner, while it is capable of interacting with libraries developed for many commonly used programming languages. Despite not being open-source software, Neo4j has a so called community edition, available under GPL license. Some features, like high availability are not supported by this edition, but these limitations do not hinder its use in our pipeline.

E. Metadata

For each malware sample that our pipeline processes, we extract metadata from the file itself and from its corresponding VT report.

[6]We note that our approach is not limited to using TLSH, but it can also work with other similarity hash functions such as Ssdeep [2] or Sdhash [3].

For identification purposes, we compute several hashes from each file, like MD5, SHA-1, SHA-256. We also compute the file's TLSH hash for being able to relate it to similar files. We extract information about the architecture which the executable file was compiled for, including processor type and bitness. We also store if it is an executable file or a shared library, if it is statically or dynamically linked, and its file size. We compute the entropy values of the sample using a tool called `bintropy` [4]; these values can be used to determine if the sample is packed or not.

From the downloaded VT reports, we extract when the sample was first submitted to VT, how many AV products found it to be malicious, and the labels given by these AV tools, aggregated using AVClass. A timestamp is stored as well to indicate when were the labels aggregated.

III. Design choices

Our objective is to clean and process feeds and datasets of potential malware binaries using a cloud-based malware analysis platform with rate-limited access. Fig. 1 illustrates a high-level overview of our approach. We designed a pipeline that can receive malware feeds and datasets from a variety of sources, including honeypot farms, commercial malware feeds and public malware repositories or datasets. Our pipeline has multiple internal processes (e.g., filtering, metadata extractor, maintenance) that help deciding if a sample is indeed malware and, if so, extract or retrieve some of its metadata, or discard the sample and log its SHA-256 value. The metadata of the samples and the similarity relationships between the samples are then stored in a graph-based database. Our pipeline uses VirusTotal as an external component providing rate-limited malware analysis services online.

We store all the metadata in Neo4j, a graph-based database. In this database, each sample is represented by a node with the sample's metadata as properties of the node, and similar samples are connected by relationships. We chose Neo4j because it enables us to organize and store the similarity relationships between samples conveniently.

Another design choice was to create separate metadata databases for different CPU architectures. The main reason for this was to reduce database complexity and to allow for efficient querying and data management. For example, when training a malware detection model for ARM devices, it makes sense to train only on ARM samples and query only the ARM database for features for the model.

As mentioned in Section I, we want to ensure that every node in the graph is either confirmed to be malware according to its VT report or similar to a sample that is proven to be malware. To achieve this, we use labelling in Neo4j. The properties of a node in the database represent the metadata of a sample, with some metadata extracted from the sample itself and some from its VT report. If the properties of a node only contain the metadata extracted from the sample itself, the node is labeled `Incomplete`. If it also contains the metadata extracted from the VT report, the node is labeled `Complete`. Based on this, the main requirement can be stated as follows:

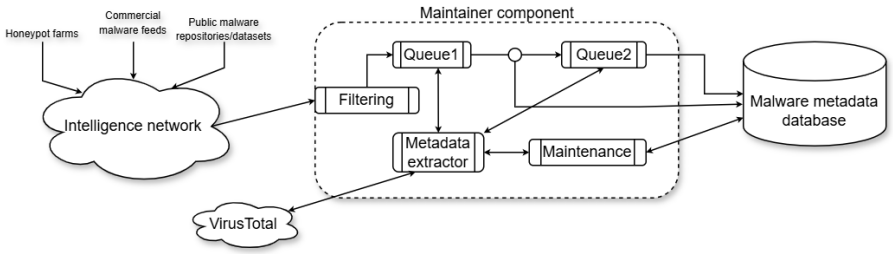

Fig. 1: High-level overview of our proposed pipeline

at least one relationship from each node of the graph labeled as `Incomplete` must connect to a node labeled as `Complete`.

An important point is that our pipeline uses the VT report of a sample to determine whether it is malware or not. More specifically, if more than 5 AV engines used by VT flag a sample as malware, then it is considered as malware. We chose this threshold because AVClass could not return family labels for samples with 5 positive votes or less. Since we cannot be certain if those samples are malware, we prefer to discard them to maintain the quality of the databases produced by our pipeline.

As mentioned earlier, two nodes are in a relationship if the corresponding samples are similar to each other, and to measure the (dis)similarity of samples, we use the TLSH similarity digest scheme. More specifically, we consider two samples similar, if their TLSH difference is smaller than a threshold. Determining the appropriate threshold value was an important part of our design. For this purpose, we constructed similarity graphs for 2000 randomly chosen malware samples using different threshold values; calculated the average clustering coefficients of these graphs; and chose the threshold that resulted in the largest clustering coefficient. We performed the above steps, and hence, obtained a threshold value for each CPU architecture. In other words, the appropriate threshold may be different for different architectures. For more details on our threshold selection methodology, the reader is referred to our prior work [5].

IV. Processing queues

In this section, we discuss how the pipeline processes new samples, extracts their metadata, and inserts them into the output databases. Two processing queues play an important role in the processing of a new dataset and its integration into the database. The first processing queue (Queue 1) minimizes the number of requests made to VirusTotal by integrating samples into the database based on their similarity to already confirmed malware in the database. The second processing queue (Queue 2) is used to process the samples that could not be added to the database by the first queue. The operation of the second queue is also designed to minimize the number of requests made to VirusTotal. After processing by Queues 1 and 2, the database typically still contains nodes labeled as `Incomplete`. This means that these nodes are not yet confirmed to be malware, although they are similar to already confirmed malware samples. The role of the Maintenance process is to process these remaining `Incomplete` samples in the background, and turn them `Complete`, which includes downloading their VT reports; requesting a re-analysis if needed; deciding if they are indeed malware; and filling in all their missing metadata.

A. Queue 1

For the following discussion, we assume that the output database already contains metadata and similarity relationships of a set of samples; an empty database is just a special case when that set of samples is empty. Every new sample that we want to add to the database is processed by Queue 1, which performs the following operations: (i) Check if the SHA-256 value of the given sample already exists in the database; if so, then the sample is already in the database and no further steps are needed. (ii) Check if the given sample is similar to a sample in the database that is labeled `Complete`. If this is not the case, the sample is passed for further processing to Queue 2. Otherwise, (iii) extract metadata from the sample and add a node with the extracted metadata to the database labeled as `Incomplete`. Finally, (iv) add also all the relationships of the new sample with similar samples in the database.

Note that Queue 1 does not use at all the cloud-based malware analysis platform; it adds new samples to the database based on their similarity to already confirmed malware (nodes labeled `Complete`) in the database. In addition, Queue 1 adds new samples to the database in such a way that it preserves the property that every node in the database is either confirmed malware or similar to a confirmed malware.

B. Queue 2

Queue 2 processes a set of new samples that were not found to be similar to any confirmed malware samples in the database by Queue 1. The processing steps are the following: (i) Represent the set of new samples to be added to the database as a graph where the nodes represent the samples and two nodes are connected by an edge if the corresponding samples are similar (according to the similarity threshold defined earlier). (ii) Calculate a dominating set of the graph representing the new samples. (iii) Download the VT reports of the samples in the dominating set. (iv) Extract metadata from all new samples and metadata from the VT reports of the samples in the dominating set. (v) Add the new samples

with their metadata to the database labeling the samples in the dominating set `Complete` and the rest of the new samples (having no VT reports yet) `Incomplete`. Finally, (vi) add also all the similarity relationships among the new samples and between the new samples and the existing samples in the database.

Certain specific cases may require special treatment. One such case is when a new sample in the dominating set is not yet known to VT or its VT report indicates that it is a benign sample. In this situation, we discard the sample and repeat the above steps (such that the set of new samples no longer contains the discarded one). Note that we save VT reports locally continuously, so when repeating the above steps, we do not actually download VT reports for any sample multiple times, in order to minimize quota usage.

C. Maintenance

After the samples have passed through the processing queues, the output database contains nodes labeled as `Complete` or `Incomplete`. A node labeled `Incomplete` always has a similarity relationship with at least one node labeled `Complete`. For this reason, the database is considered usable at this stage. Once the database reaches this point, we start requesting VT reports for samples labeled `Incomplete` in the background and turn them into `Complete` nodes. During this completion process, we ensure the preservation of the property that each sample in the database is either proven to be malware or similar to a sample that is proven to be malware. This keeps the database usable during the potentially long maintenance process.

The maintenance process is potentially long, because it must download VT reports for the `Incomplete` nodes in the database while respecting the quotas. When a VT report is downloaded, metadata can be extracted from it, and hence, the `Incomplete` node can be turned `Complete` by adding the extracted metadata to the database. However, there are two special cases to consider: (i) no VT report is available or it may indicate that the sample is benign and (ii) the VT report is not trustworthy[7]. In case (i), we simply delete the sample (including its metadata and relationships) from the database. Note that, as this sample was an `Incomplete` node, deleting it does not invalidate the property that every sample in the database is either confirmed malware or similar to a confirmed malware. In case (ii), we request a re-analysis of the sample by VT and obtain a fresh VT report. If the fresh report still indicates that the sample is malware, we insert it in the database in the way described above.

V. Evaluation

A. Dataset

Our approach is evaluated using datasets (A, B, C, D-1 and D-2) [6], [7] collected by Yokohama National University (YNU) by capturing potentially malicious files with two honeypots [7]–[9]. The datasets contain $234,145$ samples, out of which samples in three datasets (A, B, C) have additional VT reports attached. These samples helped us to initialize the database with `Complete` nodes and we evaluated the processes that were explained in Section IV using samples from the remaining two datasets (D-1, D-2). Note that these honeypots are specifically designed to simulate vulnerable IoT devices, so the datasets contain malware targeting such devices. We further narrowed the scope to ELF binaries developed for the ARM and MIPS architectures. After filtering out the ARM and the MIPS samples, and checking their already available VT reports, our initial ARM database contained $22,351$ nodes and $12,338,886$ edges, while the MIPS database contained $27,745$ nodes and $26,263,112$ edges (both constructed with TLSH difference threshold 50).

B. Measurement of processing queues

We measure the time to add metadata of a new sample to the database. This includes measuring the average, minimum and maximum elapsed time to compare the sample with all nodes in the database, extract metadata from the sample, and then insert the sample and its relationships into the database. For the results see Table I.

TABLE I: Processing times of new samples expressed in seconds (s)

Architecture	Avg. elaps.	Min. elaps.	Max. elaps.
ARM	73.523	1.087	791.1
MIPS	39.498	1.029	442

We also measured the number of samples processed by the processing queues. Regarding dataset D-1:

- $27,913$ ARM and $14,181$ MIPS samples were added to the database by Queue 1;
- $20,180$ ARM and $10,136$ MIPS samples were passed to Queue 2; and
- 194 ARM and 151 MIPS samples were already in the database (duplicates from datasets A, B, or D).

Regarding dataset D-2:

- $1,958$ ARM and $1,481$ MIPS samples were added to the database by Queue 1;
- $1,370$ ARM and 848 MIPS samples were passed to Queue 2; and
- $1,745$ ARM and $1,178$ MIPS samples were already in the database.

C. Quota usage

After some pre-filtering, dataset D-1 contained $48,287$ ARM samples and $24,768$ MIPS samples to be added to the database, and in dataset D-2, we had $5,073$ ARM samples and $3,507$ MIPS samples to be processed with our pipeline. Tables II and III show the amount of VT quota usage by Queue 2 and the maintenance processes (note that Queue 1 does not use any VT quota).

[7]We consider a VT report untrustworthy when it was done too close (within 1 week) to the time the sample was first submitted to VirusTotal, and only a few (at most 10) AV engines judged it to be malicious at that time.

TABLE II: VirusTotal quota used to fully process samples in dataset D-1

Purpose of quota usage	ARM	MIPS	Total
Queue 2	14,754	7,828	22,582
Complete `Incomplete` nodes	33,339	16,789	50,128
Check untrustworthy nodes	1,802	1,946	3,748
Total	49,895	26,563	76,485

TABLE III: VirusTotal quota used to fully process samples in dataset D-2

Purpose of quota usage	ARM	MIPS	Total
Queue 2	1,295	817	2,112
Complete `Incomplete` nodes	2,031	1,512	3,543
Check untrustworthy nodes	12	10	22
Total	3,338	2,339	5,677

D. Final result

Initially, there were a total of 234,315 binaries in the five datasets. After the full evaluation, 101,631 samples remained in the databases; 101,620 samples were discarded after filtering for ARM and MIPS binaries, and an additional 31,064 samples were discarded due to duplicates and because they were found benign or unknown by VT. After all the processing, the metadata database for the ARM samples consists of the metadata of 55,497 samples that were found malicious based on their VT report and there are 25,119,599 similarity relationships between these samples. The MIPS metadata database consists of metadata of 46,134 malware samples and there are 23,069,282 similarity relationships among them. The databases that we created are available on GitHub[8] together with the source code of our pipeleine components[9].

VI. Conclusion

In this paper, we proposed a pipeline for cleaning and processing large datasets of potentially malicious binaries using rate-limited access to a cloud-based malware analysis platform. Our primary design objective was to minimize the time needed to reach a state in which the dataset being cleaned and processed is usable in practice. We achieved this by building a metadata database from the samples in the dataset and ensuring that we quickly bring this database into a state where every sample included in the database is either confirmed malware or similar to a confirmed malware. So, while in this state, we have no proven information for the latter kind of samples, they are likely to be malware too due to their similarity to samples known to be malware. Once this state is reached, our pipeline continues requesting analysis reports for all as-yet-unproven samples from the online malware analysis platform, and by doing so, converts the incomplete database to a complete one in the background, while it remains usable throughout the entire process.

We evaluated our pipeline by cleaning and processing five IoT malware datasets (A, B, C, D-1 and D-2) made publicly available by the Yokohama National University. Our pipeline processed 101,631 samples and created two metadata databases, one for samples developed for the ARM architecture and another for MIPS binaries, containing metadata for 55,497 and 46,134 samples, respectively. We measured the performance of our pipeline in terms of processing time and VT quota usage. We reached the state where the metadata databases were already usable with only 22,582 and 2,112 VT requests compared to the total numbers of 76,485 and 5,677 VT requests needed for completing all records, for the ARM and MIPS samples, respectively. This means that the usable state of the databases was reached with around 30-40% of the total effort, which we consider a significant result.

[8]https://github.com/CrySyS/CrySyS-IoT-MMDB-2024

[9]https://github.com/CrySyS/CrySyS-IoT-MMDB-Maintainer

VII. Acknowledgements

The research presented in this paper was supported by the European Union project RRF-2.3.1-21-2022-00004 within the framework of the Artificial Intelligence National Laboratory and by the European Union's Horizon Europe Research and Innovation Program through the DOSS Project (Grant Number 101120270). The presented work also builds on results of the SETIT Project (2018-1.2.1-NKP-2018-00004), which was implemented with the support provided from the National Research, Development and Innovation Fund of Hungary, financed under the 2018-1.2.1-NKP funding scheme. The authors are also thankful to VirusTotal for the academic license provided for research purposes.

References

[1] J. Oliver, C. Cheng, and Y. Chen, "TLSH – A Locality Sensitive Hash," in *2013 Fourth Cybercrime and Trustworthy Computing Workshop*, 2013, pp. 7–13.

[2] J. Kornblum, "Identifying almost identical files using context triggered piecewise hashing," *Digital Investigation*, vol. 3, pp. 91–97, 2006.

[3] V. Roussev, "Data Fingerprinting with Similarity Digests," in *Advances in Digital Forensics VI*, K.-P. Chow and S. Shenoi, Eds., Berlin, Heidelberg, 2010, pp. 207–226.

[4] R. Lyda and J. Hamrock, "Using Entropy Analysis to Find Encrypted and Packed Malware," *IEEE Security & Privacy*, vol. 5, pp. 40–45, 2007.

[5] L. Buttyán, R. Nagy, and D. Papp, "SIMBIoTA++: Improved similarity-based IoT malware detection," in *Proceedings of the IEEE Conference on Information Technology and Data Science (CITDS)*, Debrecen, Hungary, May 2022.

[6] Y. M. Pa Pa, S. Suzuki, K. Yoshioka, T. Matsumoto, T. Kasama, and C. Rossow, "IoTPOT: A novel honeypot for revealing current IoT threats," *Journal of Information Processing*, vol. 24, pp. 522–533, 2016.

[7] Y. M. P. Pa, S. Suzuki, K. Yoshioka, T. Matsumoto, T. Kasama, and C. Rossow, "IoTPOT: Analysing the rise of IoT compromises," in *9th USENIX Workshop on Offensive Technologies (WOOT 15)*, Washington, D.C., aug 2015. [Online]. Available: https://www.usenix.org/conference/woot15/workshop-program/presentation/pa

[8] S. Kato, R. Tanabe, K. Yoshioka, and T. Matsumoto, "Adaptive Observation of Emerging Cyber Attacks targeting Various IoT Devices," in *2021 IFIP/IEEE International Symposium on Integrated Network Management (IM)*, 2021, pp. 143–151.

[9] R. Tanabe, T. Tamai, A. Fujita, R. Isawa, K. Yoshioka, T. Matsumoto, C. Gañán, and M. van Eeten, "Disposable botnets: examining the anatomy of IoT botnet infrastructure," in *Proceedings of the 15th International Conference on Availability, Reliability and Security*, New York, NY, USA, 2020.