Creating EMBeD, the Embedded Malware Benchmark Dataset

Dávid Maliga
CrySyS Lab
Budapest University of Technology and Economics
david.maliga@crysys.hu
ORCID: 0009-0005-0106-8538

Dorottya Papp
CrySyS Lab
Budapest University of Technology and Economics
dpapp@crysys.hu
ORCID: 0000-0002-9976-614X

Levente Buttyán
CrySyS Lab
Budapest University of Technology and Economics
buttyan@crysys.hu
ORCID: 0000-0003-4233-2559

***Abstract*—IoT devices, such as wireless routers, IP cameras, and smart home gadgets, are widely adopted in everyday life. Unfortunately, it is both technically feasible and economically viable for attackers to infect them with malware. In response to this threat, the research community has made significant efforts to develop various malware detection methods for IoT devices. However, the published solutions are rarely accompanied by the dataset of malware samples on which they were evaluated by their authors. As a matter of fact, *currently there does not exist any publicly available benchmark dataset of IoT malware binaries that would make independent validation and comparison of IoT malware detection methods possible in a meaningful way*. To address this issue, in this paper, we make the first steps towards an Embedded Malware Benchmark Dataset (EMBeD) by proposing a methodology to create such a dataset. We are currently using this methodology to build EMBeD, which we intend to make publicly available for the benefit of the research community.**

***Index Terms*—IoT, malware, benchmark dataset**

I. Introduction

The Internet-of-Things (IoT) penetrated in many areas of life, including smart homes and smart buildings, industrial facilities, modern vehicles and the transport infrastructure. From a technological perspective, IoT devices are embedded computers that often run a known (usually Linux-based) operating system, their wired or wireless communication is based on the TCP/IP protocol stack, and they can host a mix of open-source and proprietary software. Unlike traditional computers, however, they generally have far fewer resources: they may run on battery, or have less memory and processing power to perform various tasks.

As embedded IoT devices resemble the architecture of computers in traditional IT systems, it is technically feasible for attackers to target these devices by exploiting open ports, weak authentication, and implementation flaws. These attacks are not only technically feasible but can also yield economic benefits for attackers, who can sell stolen data or demand ransom using next-generation ransomware for critical infrastructure. The embedded IoT device itself is valuable, as it possesses computing power that an attacker can exploit for their own purposes. Moreover, attacks can be automated and IoT devices can be compromised at large scale by infecting them with malware (malicious software). It is therefore not surprising that for nearly a decade, we have been aware of malware specifically targeting embedded IoT devices [1], such as Mirai [2] or Aisuru[1].

In response to the emerging threats, the research community has made – and continues to make – significant efforts to develop various defense techniques, and a significant portion of those are specifically concerned with malware detection [3], [4]. Malware detection approaches can be classified into two main classes: network-based detection and endpoint-based detection. While the former is more prevalent in industry solutions, the latter is gaining momentum via operating systems such as Fortinet's FortiOS[2] and Kaspersky's KasperskyOS[3]. The advantage of endpoint-based detection methods is that they overcome a major limitation of network-based detection approaches, namely, that they can operate in the presence of secure, encrypted network connections as well.

However, validating malware detection techniques requires datasets, and the current trend in research is for individual research groups to compile their own datasets. Unfortunately, some of these datasets are not publicly available [1], [5], while others, though publicly available, contain only derived data, e.g., the Bot-IoT dataset [6] and UNSW Attack Data [7]. The problem with using the former datasets is that the research results based on them are not reproducible. The problem with the second type of datasets is that they can only be used with certain types of defense techniques (e.g., network-based detection). There are only a few datasets that provide malware to the research community in the form of binary files, thereby facilitating the validation of both endpoint-based and network-based detection research. Examples of such datasets include CUBE-MALIOT-2021[4] and the datasets from the Yokohama National University (YNU)[5]. Unfortunately, these datasets are not fit out of the box to be used for comparing malware detection approaches. CUBE-MALIOT-2021 has not been updated for 5 years, during which the threat landscape

[1]https://krebsonsecurity.com/2025/10/aisuru-botnet-shifts-from-ddos-to-residential-proxies/ (May 6, 2026)
[2]https://www.fortinet.com/products/fortigate/fortios (Apr 14, 2026)
[3]https://os.kaspersky.com/ (Apr 14, 2026)
[4]https://github.com/CrySyS/cube-maliot-2021 (Apr 14, 2026)
[5]https://sec.ynu.codes/iot/available_datasets (Apr 14, 2026)

has undergone changes, and the datasets from YNU contain duplicates, resulting in an unclean dataset. In summary, *there is no publicly available dataset of IoT malware binaries that could act as a benchmark for the research community, i.e., making independent validation and comparison of IoT malware detection methods possible in a meaningful way*.

In this paper, we make the first steps towards creating a benchmark dataset of IoT malware binaries for the research community by presenting a methodology to build such a dataset. Specifically, we outline our workflow for creating a *benchmark* dataset, called Embedded Malware Benchmark Dataset, or EMBeD for short, by combining (1) sampling pre-existing malware datasets and (2) hunting for samples in existing malware repositories, with the specific goals of achieving a balance in terms of the represented malware families, avoiding under-represented features, and having a volume of samples sufficiently large for machine-learning-based malware detection approaches. Our proposed workflow is an automated workflow that can be run yearly to ensure that EMBeD remains up-to-date even in the face of changing threats.

What we report is work-in-progress in the sense that EMBeD is not yet fully built, but we are currently using our proposed methodology to build it. However, we feel that the design of our methodology is a significant milestone, which is worth to be shared with the research community, along with some initial results that our workflow has produced so far. Once EMBeD is completed, we will make it publicly available for the benefit of the research community.

The paper is structured as follows: Section II provides an overview of the requirements posed against datasets for them to be considered benchmarks. Section III reviews existing malware datasets in the IoT domain and highlights their shortcomings with respect to the requirements for a benchmark. Section IV details our workflow and first results for creating EMBeD that can later be used as a benchmark. Finally, Section V concludes the paper.

II. What makes a good benchmark dataset?

Benchmark datasets are critical to research. They are used to evaluate and compare the performance of algorithms or models, and as such, they are carefully curated to represent specific tasks. Benchmark datasets are used as public standards for model comparison, playing a critical role in evaluating framework performance, strengths and weaknesses.

A dataset must meet several requirements to be considered a benchmark dataset [8]–[11]:

- *Availability and accessibility*: The dataset should be open (with explicit license), discoverable and accessible, hosted at a reliable site for long-term use.
- *Ease of use*: The dataset should be well documented, have an accompanying demonstration to showcase how to process the data. The dataset should not be too big to prevent challenges in processing it.
- *Compliance with machine learning requirements*: The dataset should be balanced and should address at least one clear machine learning task. It should have predefined training and testing splits to ensure consistent evaluation across experiments, as well as positive and negative cases.
- *Cleaned and comprehensible*: The data set should provide clean data points that cover a broad spectrum of scenarios and edge cases. It should also be non-redundant: overlapping cases within the dataset should be excluded. The dataset should also address the concept drift problem, namely that the distribution of data included in the dataset may change over time.
- *Requirements for data points*: The data points in the dataset should be representative, i.e., they should closely reflect the types of data present in real-world applications. They should be labeled and should have enough number of independent features to be interesting, but with only the relevant characteristics retained.

Applying these requirements for malware benchmark datasets reveals two major challenges. The requirement for adding negative examples would have the benchmark include pieces of benign software as well, which is typically licensed in some way. As a result, such inclusions must respect the license terms, potentially limiting the pool of benign software that can be considered for inclusion.

The second challenge is how to handle the concept drift problem. Specifically in the malware detection domain, the challenge arises because the threat landscape is constantly changing [12], [13]. New malware families and variants are discovered, and new packing and obfuscation methods are employed by attackers to evade detection. The introduction of new technologies in IT systems also steers malware authors towards new targets. And these are just a few examples to showcase the dynamic nature of this field.

III. Related Work

The existing literature on IoT malware detection includes a wide range of datasets. Many of these are proprietary [1], [14], [15], and therefore failing the first requirement of availability and accessibility. Open datasets concerning IoT malware can be split between those useful for evaluating network-based detection methods and those for endpoint detection methods. The former category usually consists of network traces in for form of PCAP files. Examples include the BoT-IoT dataset [6], the IoT-23 dataset[6], the IoT network intrusion dataset [16], the MedBIoT dataset [17], and the TON-IoT dataset [18], [19] (which also includes behavioral features from devices, such as processor, memory, and process activities). However, because our focus in on malware detection at endpoints (IoT devices), they are out of scope for our work and we do not dissect their features in depth.

Datasets useful for developing malware detection methods for IoT endpoints have a wide range of dataset size but they don't necessarily contain malware binaries. For example, the IoT Malware Dataset [20] contains only the opcodes of binaries, not the binaries themselves. Similarly, the IOT_Malware[7]

[6]https://www.stratosphereips.org/datasets-iot23 (Apr 15, 2026)

[7]https://www.kaggle.com/datasets/anaselmasry/iot-malware (Apr 15, 2026)

dataset contains image representations of executables, without any description on how to interpret the images or how the dataset was compiled. Such datasets severely limit the detection approaches that the dataset can be used to evaluate.

There exists two larger datasets of IoT malware binaries: the CUBE-MALIOT-2021 dataset and the datasets provided by the Yokohama National University (YNU). The former is a collection of IoT malware samples consisting of 29,209 ELF binaries compiled for the ARM platform and 18,715 ELF binaries compiled for the MIPS platform, collected from industry malware feeds. However, the dataset is 5 years old at the time of writing and has not been updated since. As a result, it's relevance to today's threat landscape is questionable. The YNU dataset is a collection of smaller datasets of IoT malware captured from two honeypots, IoTPOT [21] and its improved version, X-POT, which is updated yearly. However, the description of the dataset does not mention any preprocessing steps being applied to the binaries, such as cleaning, removal of duplicates, or filtering. The YNU dataset is enhanced by the CrySyS-IoT-MMDB-2024[8] dataset that publishes metadata for the malware binaries compiled for ARM and MIPS via a locally deployed Neo4j database. Unfortunately though, the limitations of the YNU dataset still apply.

We take note the existence of the EMBER2024 dataset [22], the latest update to the EMBER benchmark widely used as a benchmark for Windows malware. However, it only contains 250 ELF binaries and not all of them are relevant for the IoT threat landscape. Dambra *et al.* [23] also collected and curated a dataset of PE malware executables. Their work details in depth the collection and data preparation steps they took to curate their dataset for their study. The resulting dataset fulfills much of the requirements to make it a benchmark dataset: it is balanced, cleaned, and comprehensible, as well as publicly available on GitHub. However, because it is concerned with Windows malware, it is out of scope for our work.

IV. Our workflow for compiling a benchmark dataset of IoT malware

In this section, we provide an overview of the workflow that we designed and we are currently using to create EMBeD, the Embedded Malware Benchmark Dataset. Our workflow, shown in Figure 1, consists of several steps. We first process publicly available malware datasets using the pipeline described in [24] to create a malware metadata database, which supports later steps in our workflow. At this stage, we also filter out samples with underrepresented features from the input datasets. Next, we enhance malware family labels using metadata and sample similarity. Finally, we enrich underrepresented families by hunting for more samples from those families in existing malware repositories and sample overrepresented ones in order to arrive to an approx. equal and sufficiently large number of samples per family.

[8]https://github.com/CrySyS/CrySyS-IoT-MMDB-2024 (Apr 16, 2026)

A. Data preparation

Maliga et al. [24] presented a pipeline for processing publicly available malware datasets and extracting metadata of the samples they contain. The metadata extraction includes static features of the samples, such as their hashes, architecture information (type and bitness), and file attributes (e.g., linking and entropy). The pipeline also obtains metadata from VirusTotal (VT) reports, including the number of antivirus (AV) engines that flagged the sample as malicious and the family labels assigned by these engines. The main goal of the pipeline is to filter out benign samples and samples that cannot be convincingly judged as malware from the input datasets, and create a metadata database of the remaining, likely-to-be malware samples.

We leverage this pipeline and pre-process with it the input datasets that we use to construct our benchmark dataset from. In addition, already at this stage, we use the extracted metadata to identify samples with underrepresented features and exclude them from further processing by our workflow.

B. Enhancing family labeling

For a benchmark dataset of malware samples, it is essential that the samples are labeled correctly with malware family names. The VT report of a sample typically contains multiple, and sometimes too generic, family names. In order to assign a single family name to each sample with high confidence, we follow two approaches: (i) we evaluate the accuracy of the AV engines that appear in VT reports, and (ii) we analyze the similarity between labeled samples.

As we described above, the VT report of a sample contains family labels given by different AV engines to the sample, and we need to aggregate these labels into a single family label. For this, we could use simple majority voting: for each sample, we could count the votes of AV engines for each family label, and the label with the highest number of votes could become the family label for the sample. However, this approach can be inaccurate, as AV engines may not reliably assign family labels to samples, potentially skewing the results, particularly when a single vote decides the outcome. Hence, it is also important to consider how accurately AV engines assign labels to samples in general, and to give higher weights to the votes of more accurate AV engines.

For quantifying the accuracy of an AV engine e in family labeling, we use the following formula:

$$accuracy_e = \frac{agree_e}{agree_e + disagree_e} \tag{1}$$

where $agree_e$ is the number of samples in the database where the family label given by e matches the label given by a majority of the AV engines and $disagree_e$ is the number of samples where the family label given by e does not match the majority label or e did not give any family label at all or did not detect the sample as malicious. The accuracy score of AV engine e is then used to weight the vote of e when determining the final family label for every sample.

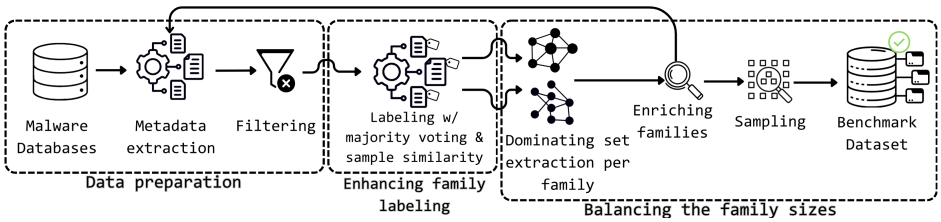

Fig. 1: The workflow for creating a benchmark dataset.

Next, we use the similarity relationship between samples to verify the family labels assigned by the weighted voting mechanism described above. For measuring the (dis)similarity between samples, we use a fuzzy hashing scheme called Trendmicro Locality Sensitive Hash (TLSH) [25]. TLSH takes an input of any size and produces a fixed-length hash value, similar to cryptographic hash functions. However, unlike cryptographic hash functions, similar inputs result in similar hash values. Along with the hash function, a difference-scoring algorithm is also given that quantifies the difference between two TLSH hash values (i.e., how dissimilar they are). The difference score is an integer, where 0 indicates that the two inputs are (almost) identical, whereas a higher score indicates that the inputs are less similar.

By computing the difference scores for all pairs of the remaining samples in the database, we create a similarity graph where each node represents a sample and there is an edge between two nodes if their difference score is below a given threshold. We use this similarity graph to verify the assigned family label of every sample. Here, our intuition is that if the family label f of a sample s does not match that of its close neighbors in the graph (i.e., very similar samples), while the majority of these neighbors agree on the same label $f' \neq f$, then the label f of s is inconsistent. As we do not have automated methods to determine the true family of a sample, we prefer excluding samples with such inconsistent labels from the benchmark.

The way we determine if the family label of a sample s is consistent or not with those of its neighbors in the similarity graph is the following: For each neighbor i of s, let $\bar{C}_i$ be the vector of weighted and normalized votes given to each malware family appearing in the input database by the AV engines when analyzing sample i. So the j-th element of $\bar{C}_i$ is the weighted and normalized vote on sample i belonging to malware family j. We aggregate these vectors for all neighbors of s as follows:

$$F_s = \frac{\sum_i \left(1 - \frac{1}{1+e^{-\lambda(d_i-25)}}\right) \bar{C}_i}{\sum_i \left(1 - \frac{1}{1+e^{-\lambda(d_i-25)}}\right)} \quad (2)$$

where the sum is taken over all neighbors i of sample s in the similarity graph, d_i is the difference score between sample s and its neighbor i, and λ controls the steepness and 25 is the inflection point of the sigmoid function that weights each neighbor's contribution to the aggregated sum. So vector F_s contains the weighted average of the votes on each malware family taken over all the neighbors of s. We then choose the family with the highest weighted average vote in F_s and check if that matches the family label currently assigned to sample s. If not, then we consider the family label of s inconsistent (with that of its neighbors), and we discard s from the similarity graph and from further analysis. Such deletion of a sample may influence the labeling consistency of its neighbors in the similarity graph, therefore we iteratively recompute and check the consistency of the family labeling, and delete inconsistently labeled samples, until all remaining samples have consistent family labels.

C. Balancing the family sizes

After enhancing the family labeling, the graph includes samples with more accurate family labels. In the next step, we create similarity subgraphs for each family, meaning that each subgraph contains only the vertices that correspond to samples from a specific family. Then, we calculate a dominating set for each subgraph. A dominating set is a subset of nodes in a graph such that every node in the graph is either in the dominating set or is adjacent to a node in the dominating set. To calculate a dominating set for each subgraph, we use a well-known greedy algorithm[9].

Our intuition is that the samples of a dominating set form a representative subset of samples of the given malware family, as every other family member is similar to at least one dominating sample. Depending on the size of the resulting dominating set, there are different cases that can occur for each family:

- if the size of the dominating set is close to the desired size N, then we can directly use the samples in the dominating set to include them in our benchmark dataset;
- if the size of the dominating set is much smaller than N, then we should enrich the given family with additional samples; and
- if the size of the dominating set is much larger than N, then we should sample the dominating set and keep only approx. N samples for the benchmark.

One approach to enriching underrepresented families is to use the VT Livehunt and Retrohunt features. Livehunt allows us to perform real-time hunting. To do this, we need to define Yara[10] rules. YARA is a tool for identifying malware by defining rules that match patterns such as byte sequences and strings. By defining Yara rules for a family, we can use

[9]We use algorithm 7 from https://www.cse.msu.edu/~cse835/Papers/Graph_connectivity_revised.pdf with the addition that we always select vertex $w \in (V - D \cup N(D))$ with the largest degree.

[10]https://yara.readthedocs.io/en/latest/ (Apr 30, 2026)

Livehunt to find new samples that match those rules as they are submitted to VT or requested for reanalysis in real time. On the other hand, Retrohunt allows us to search for samples in the malware VT database. Similarly, Retrohunt evaluates the Yara rules against the samples in the VT database, returning a list of matches. To generate Yara rules for a family, we can use the samples in the dominating set. We analyze these samples with a tool called yarGen[11], which generates Yara rules based on the common patterns of the input samples. By using the generated Yara rules in Livehunt and Retrohunt, we can find additional samples belonging to the same family, thereby enriching underrepresented families with new samples.

Another approach is to do a similarity search in a malware repository. For this, we can use the TLSH hashes of the samples in the dominating set to search for similar samples in a malware repository, such as VT or Kaibou[12]. By finding samples similar to those in the dominating set of a malware family, we can identify additional samples that likely belong to the same family, thereby enriching underrepresented families.

To ensure that the newly added samples are indeed relevant, we need to run another iteration of the entire workflow. If needed, we repeat these steps until we have the required number N of samples in each family.

If the size of the dominating set for a family exceeds N, we can slightly increase the (dis)similarity threshold we use for constructing the similarity graph, which results in more edges between the nodes, and thus, a smaller dominating set. If needed, we can repeat this step until we get around N samples in the dominating set of the given malware family.

D. Preliminary results

Here, we present the preliminary results of applying our workflow to create a benchmark dataset of IoT malware. Using the pipeline in [24], we created a malware metadata database, processing several publicly available datasets, including the YNU datasets, CUBE-MALIOT-2021, and samples from a proprietary feed received from an industrial partner.

Our goal was to create a proof-of-concpet benchmark targeting approx. 100 samples per family. For this proof-of-concept specifically, we kept only unpacked and unencrypted samples compiled for the MIPS architecture. Then, we examined the metadata of the remaining samples and found that the features of 64-bit bitness and dynamic linking are underrepresented significantly, so we filtered out samples with such features. Following all these filtering steps, $46,279$ samples remained for further processing.

Next, we applied the family labeling enhancement methods. First, we calculated the accuracy score for each AV engine and determined the initial family labels of the samples based on the accuracy-weighted votes of the AV engines. Then, we used the proposed method for checking the consistency of the family label of every sample with the family labels of its neighbors in the similarity graph. We performed multiple iterations of excluding samples with inconsistent family labels and re-computing and re-checking family label consistency across neighbors to reach a steady state where all family labels were consistent.

Then, we created a similarity subgraph for each remaining family and computed its dominating set. The results of this step are shown in Table I. As one can see, the mirai and gafgyt families were overrepresented, while all other families had significantly fewer samples than the target (100 samples per family) in their dominating sets. So we needed to enrich these families with additional samples.

	Original		After hunting		
Family label	$\|V\|$	$\|D\|$	$\|V\|$	$\|D\|$	Diff.
mirai	37951	849	38085	888	+134
gafgyt	6818	196	6836	198	+18
hajime	93	2	736	10	+643
dofloo	71	2	128	3	+57
mrblack	59	1	75	1	+16
tsunami	40	2	65	8	+25
kaiji	19	7	504	30	+485
ddostf	14	1	36	1	+22
bricker	11	2	12	2	+1

TABLE I: The results of the workflow. $|V|$ is the number of samples, and $|D|$ is the size of the dominating set. The green cells indicate the families that can be included in the benchmark dataset.

To enrich the underrepresented families, we generated Yara rules for each family and used them in Livehunt and Retrohunt on VirusTotal, and performed similarity searches in Kaibou. After enrichment, we ran another iteration of the workflow and repeated all the previous steps. The results of this iteration are also shown in Table I.

In case of hajime, dofloo and kaiji there were fewer than 100 samples in the dominating set, but still enough to lower the (dis)similarity threshold, thereby increasing the size of the dominating set, such that finally we were able to select the target number of samples into the benchmark. For the mirai and gafgyt families, where the size of the dominating set was larger than 100, we increased the (dis)similarity threshold of the similarity graph in order to reduce the size of the dominating set to approx. 100. Unfortunately, for the other families that appeared in the input datasets, we were not able to hunt a sufficient number of new samples, so those families remained underrepresented, and hence, we had to exclude them from the benchmark, which finally contained samples from 5 malware families (mirai, gafgyt, hajime, kaiji, dofloo), with approx. 100 samples per family. However, continuing enrichment may eventually provide more samples from those families too. We also note that the number of samples in the 4 largest families is sufficient to create a benchmark covering only these families but containing 500 samples per family.

V. Summary

In this paper, we have presented a methodology for creating a benchmark dataset of IoT malware binaries. We have

[11]https://github.com/Neo23x0/yarGen (Apr 26, 2026)
[12]https://ukatemi.com/products/kaibou/ (Apr 27, 2026)

outlined the requirements for a good benchmark and reviewed existing datasets in the literature against these requirements. We then detailed our workflow for creating EMBeD, an Embedded Malware Benchmark Dataset. We illustrated how the workflow works by creating a proof-of-concept benchmark, targeting 100 samples per malware family. We were able to reach this target only for 5 families, despite some enrichment of the initial datasets with hunting on VirusTotal and similarity searches on Kaibou. However, we note that VirusTotal limited the number of matches for a given Yara ruleset and rate-limited our access to the platform. We are confident that even better results could have been achieved without such limitations.

VI. Acknowledgment

We are thankful to Aanjhan Ranganathan and his team for their help in searching for existing IoT malware datasets. We are thankful to VirusTotal for providing us with an academic license. We are also thankful to Ukatemi Technologies for providing us access to Kaibou. This work was supported by the National Research, Development and Innovation Fund of Hungary (NKKP-ADVANCED_25-153325).

References

[1] E. Cozzi, P.-A. Vervier, M. Dell'Amico, Y. Shen, L. Bilge, and D. Balzarotti, "The tangled genealogy of IoT malware," in *Proceedings of the 36th Annual Computer Security Applications Conference*, ser. ACSAC '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 1–16. [Online]. Available: https://doi.org/10.1145/3427228.3427256

[2] M. Antonakakis, T. April, M. Bailey, M. Bernhard, E. Bursztein, J. Cochran, Z. Durumeric, J. A. Halderman, L. Invernizzi, M. Kallitsis, D. Kumar, C. Lever, Z. Ma, J. Mason, D. Menscher, C. Seaman, N. Sullivan, K. Thomas, and Y. Zhou, "Understanding the Mirai botnet," in *26th USENIX Security Symposium (USENIX Security 17)*. Vancouver, BC: USENIX Association, Aug. 2017, pp. 1093–1110. [Online]. Available: https://www.usenix.org/conference/usenixsecurity17/technical-sessions/presentation/antonakakis

[3] P. Victor, A. H. Lashkari, R. Lu, T. Sasi, P. Xiong, and S. Iqbal, "Iot malware: An attribute-based taxonomy, detection mechanisms and challenges," *Peer-to-Peer Networking and Applications*, vol. 16, no. 3, pp. 1380–1431, May 2023. [Online]. Available: https://doi.org/10.1007/s12083-023-01478-w

[4] S. Kumar, P. Ahlawat, and J. Sahni, "IoT malware detection using static and dynamic analysis techniques: A systematic literature review," *SECURITY AND PRIVACY*, vol. 7, no. 6, p. e444, 2024. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/spy2.444

[5] E. Cozzi, M. Graziano, Y. Fratantonio, and D. Balzarotti, "Understanding Linux malware," in *2018 IEEE Symposium on Security and Privacy (SP)*, 2018, pp. 161–175.

[6] N. Koroniotis, N. Moustafa, E. Sitnikova, and B. Turnbull, "Towards the development of realistic botnet dataset in the Internet of Things for network forensic analytics: Bot-IoT dataset," *Future Generation Computer Systems*, vol. 100, pp. 779–796, 2019. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167739X18327687

[7] A. Hamza, H. H. Gharakheili, T. A. Benson, and V. Sivaraman, "Detecting volumetric attacks on IoT devices via SDN-based monitoring of MUD activity," in *Proceedings of the 2019 ACM Symposium on SDN Research*, ser. SOSR '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 36–48. [Online]. Available: https://doi.org/10.1145/3314148.3314352

[8] M. Hall, "What makes a good benchmark dataset?" https://agilescientific.com/blog/2019/4/3/what-makes-a-good-benchmark-dataset, 2019.

[9] "What makes a good benchmark or truth dataset?" https://www.magnalabs.co/resources/what-makes-a-good-benchmark-or-truth-dataset, Magna Labs, 2024.

[10] A. Sarkar, Y. Yang, and M. Vihinen, "Variation benchmark datasets: update, criteria, quality and applications," *Database*, vol. 2020, pp. 1–16, 2020.

[11] N. Sourlos, R. Vliegenthart, J. Santinha, M. E. Klontzas, R. Cuocolo, M. Huisman, and P. van Ooijen, "Recommendations for the creation of benchmark datasets for reproducible artificial intelligence in radiology," *Insights into Imaging*, vol. 15, no. 1, p. 248, Oct 2024. [Online]. Available: https://doi.org/10.1186/s13244-024-01833-2

[12] R. Jordaney, K. Sharad, S. K. Dash, Z. Wang, D. Papini, I. Nouretdinov, and L. Cavallaro, "Transcend: Detecting concept drift in malware classification models," in *Proceedings of the 26th USENIX Conference on Security Symposium*, ser. SEC'17. USA: USENIX Association, 2017, p. 625–642.

[13] F. Barbero, F. Pendlebury, F. Pierazzi, and L. Cavallaro, "Transcending TRANSCEND: Revisiting malware classification in the presence of concept drift," in *2022 IEEE Symposium on Security and Privacy (SP)*, 2022, pp. 805–823.

[14] M. Dib, S. Torabi, E. Bou-Harb, and C. Assi, "A multi-dimensional deep learning framework for IoT malware classification and family attribution," *IEEE Transactions on Network and Service Management*, vol. 18, no. 2, pp. 1165–1177, 2021.

[15] T.-L. Wan, T. Ban, S.-M. Cheng, Y.-T. Lee, B. Sun, R. Isawa, T. Takahashi, and D. Inoue, "Efficient detection and classification of Internet-of-Things malware based on byte sequences from executable files," *IEEE Open Journal of the Computer Society*, vol. 1, pp. 262–275, 2020.

[16] H. Kang, D. H. Ahn, G. M. Lee, J. D. Yoo, K. H. Park, and H. K. Kim, "IoT network intrusion dataset." IEEE Dataport, 2019. [Online]. Available: https://dx.doi.org/10.21227/q70p-q449

[17] A. Guerra-Manzanares, J. Medina-Galindo, H. Bahsi, and S. Nõmm, "MedBIoT: Generation of an IoT botnet dataset in a medium-sized IoT network," in *6th International Conference on Information Systems Security and Privacy*, vol. 1, 2020.

[18] T. M. Booij, I. Chiscop, E. Meeuwissen, N. Moustafa, and F. T. H. d. Hartog, "ToN_IoT: The role of heterogeneity and the need for standardization of features and attack types in IoT network intrusion data sets," *IEEE Internet of Things Journal*, vol. 9, no. 1, pp. 485–496, 2022.

[19] A. Alsaedi, N. Moustafa, Z. Tari, A. Mahmood, and A. Anwar, "TON_IoT telemetry dataset: A new generation dataset of IoT and IIoT for data-driven intrusion detection systems," *IEEE Access*, vol. 8, pp. 165 130–165 150, 2020.

[20] H. HaddadPajouh, A. Dehghantanha, R. Khayami, and K.-K. R. Choo, "A deep recurrent neural network based approach for Internet of Things malware threat hunting," *Future Gener. Comput. Syst.*, vol. 85, no. C, p. 88–96, Aug. 2018. [Online]. Available: https://doi.org/10.1016/j.future.2018.03.007

[21] 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.: USENIX Association, Aug. 2015. [Online]. Available: https://www.usenix.org/conference/woot15/workshop-program/presentation/pa

[22] R. J. Joyce, G. Miller, P. Roth, R. Zak, E. Zaresky-Williams, H. Anderson, E. Raff, and J. Holt, "EMBER2024 - A benchmark dataset for holistic evaluation of malware classifiers," in *Proceedings of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining V.2*, ser. KDD '25. New York, NY, USA: Association for Computing Machinery, 2025, p. 5516–5526. [Online]. Available: https://doi.org/10.1145/3711896.3737431

[23] S. Dambra, Y. Han, S. Aonzo, P. Kotzias, A. Vitale, J. Caballero, D. Balzarotti, and L. Bilge, "Decoding the secrets of machine learning in malware classification: A deep dive into datasets, feature extraction, and model performance," in *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '23. New York, NY, USA: Association for Computing Machinery, 2023, p. 60–74. [Online]. Available: https://doi.org/10.1145/3576915.3616589

[24] D. Maliga, R. Nagy, and L. Buttyán, "A pipeline for processing large datasets of potentially malicious binaries with rate-limited access to a cloud-based malware analysis platform," in *2024 32nd International Conference on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS)*, 2024, pp. 1–6.

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