Public Review for

Using Application Layer Banner Data to Automatically Identify IoT Devices

Talha Javed, Muhammad Haseeb, Muhammad Abdullah, Mobin Javed

This paper is of the ”repeatable technical papers” type, which are technical contributions that provide their artefacts, e.g., software, datasets. This paper attempts to replicate a Usenix Security 2018 paper. It describes the efforts of the authors at re-implementing the solution described in the Usenix Security paper. Unfortunately, the authors of the Usenix Security paper did not reply to requests for artefacts, making the effort all the more worthwhile and challenging for the authors of this paper.

The re-implementation in this paper is evaluated based on ground truth data collected by the authors. Consistently with the type of paper, the authors release all software and datasets, to help the reproducibility their results. The authors discuss in detail various aspects that are design choices that were not originally described in the Usenix Security paper and are likely to have an impact on the results. Besides the details of the re-implementation, the datasets considered in this paper differ from the ones used in the Usenix Security paper, with a more diverse set of IoT devices, therefore asking for further studies to delineate the respective influence of the technique and the data. The results reveal significant differences in the performance of the technique of the original paper, with a way lower performance of the technique than reported in the original paper. Such a difference in the performance of the evaluated technique asks for further work to understand the respective impacts of the datasets used and of the implementation details.

This paper is a great example for our community. We hope it will encourage additional reproducibility studies, and maybe even encourage the authors of the original paper to respond to requests for artefacts, thereby helping the community to understand the considered problem better.

Public review written by
Steve Uhlig
Queen Mary University of London, UK

Using Application Layer Banner Data to Automatically Identify IoT Devices

Talha Javed
NUST Pakistan
tjaved.bscs15seecs@seecs.edu.pk

Muhammad Haseeb
LUMS Pakistan
20100192@lums.edu.pk

Muhammad Abdullah
LUMS Pakistan
19030003@lums.edu.pk

Mobin Javed
LUMS Pakistan
mobin.javed@lums.edu.pk

ABSTRACT

In this paper, we re-implement a recent work published in Usenix Security 2018: “Acquistional Rule Based Engine for Discovering Internet-of-Things Devices”. The paper introduced an NLP-based engine for automatically identifying the type, vendor, and product of IoT devices given banner data as input. We report on our efforts to reproduce the original implementation of the engine, documenting ambiguities around implementation and evaluation details that we encountered, as well as how we addressed them in our work. We evaluate our implementation on two ground truth datasets, finding that it fails to achieve the accuracy reported by the original authors. Our findings highlight the importance of recent community efforts towards a culture of reproducibility by presenting an example of how ambiguities in a research paper combined with lack of access to the original datasets can significantly affect a faithful re-implementation and evaluation.

CCS CONCEPTS

• **General and reference → Evaluation**; • **Networks → Network management**;

KEYWORDS

IoT, Device Fingerprinting, Reproducibility

1 INTRODUCTION

The recent years have seen an enormous growth in the diversity of Internet-connected devices. In addition to the traditional networking and computing devices, IoT devices such as smart fridges, smart TVs, and smart home assistants are part of many networks today.

Reliable fingerprinting of Internet-connected devices, i.e., identifying the device type, vendor, and product model is valuable for various network management and security applications, as well as broadly for various Internet measurement studies (for example, the authors in [1] use device fingerprinting to understand what devices are targeted by the Mirai botnet). Towards this end, two notable tools have gained popular use in the community: (i) First is Nmap, the open source network mapper, which identifies the device type and vendor through a combination of remote OS fingerprinting and identifying what services are running on the device [7]; (ii) Second is Ztag (part of the open source ZMap project) [8], which takes banners of services running on the device as input and labels them using a predefined rule database. A major limitation of both fingerprinting tools is that they rely on manually crafted rules. Given the growing number of vendors manufacturing IoT devices, identification of devices via hand-crafted rules is no longer a scalable solution.

A recent paper, “Acquistional Rule Based Engine for Discovering Internet-of-Things Devices” (here on referred to as ARE) addresses the “manually crafted” rules problem, and proposes a solution that given banner data, can automatically learn new rules to identify previously unseen devices [4]. The key idea behind this paper is to search the service banner data on the Internet, and use Natural Language Processing (NLP) techniques to mine the returned webpages for the likely device type, vendor, and product corresponding to the given banner.

Per the paper, the ARE authors intended to make the project open-source, as well as to provide an API based on the engine: *“... we release ARE as an open source project for the community. ARE is available to public at http://are1.tech/, providing public the APIs on the tuple (type, vendor, product) and the annotated data set”.* However, the above mentioned link is no longer operational at the time of this writing — likely due to the project not being maintained any longer. Further, we were unsuccessful in our email communication attempts to obtain the datasets and source code used by the authors.

In this work, we independently attempt to re-implement ARE and reproduce the results of the ARE paper. We report on a number of ambiguities (some crucial ones) regarding implementation details, and how we address them. Due to the lack of availability of ground truth datasets used in the original paper, we compile two alternate ground truth datasets for evaluating our implementation. The first dataset contains 4,919 unique banners collected from publicly accessible Internet-connected devices, labeled using Ztag. The second dataset contains 129 unique banners collected from devices in home networks, labeled using a combination of OUI (Organizationally Unique Identifier) database lookups and manual labeling by the device owners. We created the first dataset in a manner similar to the original paper, i.e, collecting a random sample of banners from public Internet-connected devices. We added the second dataset to complement the first by providing a sample of banners from IoT devices found in home networks today. These two datasets although different than the original paper, allow us to study the promise of ARE in automatically learning rules for IoT devices — its main advantage over prior fingerprinting approaches that require manual effort.

We find that our ARE implementation performs quite poorly when compared to the results reported in the original paper. The authors report precision and recall numbers close to 95–97%, but we observe a precision of 4–63% and a recall of 1–21% on our datasets. We outline the ambiguities in the paper that may have led us to making design choices different than the original work, consequently affecting a faithful re-implementation. We publicly release our datasets and source code for the community [6].

The rest of the paper is organized as follows: we present background on three device fingerprinting tools (Nmap, Ztag, and ARE) in Section 2. Section 3 discusses our re-implementation of ARE. Section 4 describes the datasets we use in this paper. Section 5 discusses the performance of ARE. In Section 6, we discuss how to reproduce the results in this paper. We conclude in Section 7 by providing a few pointers on best practices and workflows that can help improve the reproducbility of a work.

2 BACKGROUND

In this section, we first present the reader with a brief background on two popular device fingerprinting tools, Nmap and Ztag, sketching how they work and their detection power in terms of the number of device types, vendors, and products their databases are able to label. We follow this with a detailed description of how the ARE engine works.

2.1 Nmap

Nmap detects the vendor (and product labels in some instances) of the devices via two approaches: (i) remote OS fingerprinting, and (ii) service detection. The former involves sending a series of probe packets to the device, and looking for patterns, for example in ISNs. The latter involves two methods: first is application exclusivity, i.e., if an application is known to run only on a given OS for e.g., Windows, Nmap infers the OS as such, and the vendor as Microsoft; second is extracting OS information from service banners. However, the rules based on the above methods are largely manually crafted, and are capable of detecting 27 device types and 804 vendors,[1] with the device types dominated by traditional networking and computing devices.

2.2 Ztag

Ztag relies on an expert rule database for labeling devices, with very little public information available on how these rules are constructed. In order to determine Ztag's labeling capability, we processed one snapshot of Censys data (dated 7/31/19) that has Ztag labels for a full IPv4 Internet scan, and found that the labels contain 37 unique device types, 185 unique vendors, and 1,895 unique product labels. When compared to Nmap, Ztag is capable of labeling Scada devices (such as power monitor, scada controller, and water flow controller) in addition to many devices supported by Nmap.

2.3 ARE: Acquistional Rule-Based Engine

Acquisitional Rule-based Engine is a framework for automatically learning rules that map an IoT device's application layer data (banner) to its <device type, vendor, product name> tuple. It requires no manual input or labeling. The underlying assumption behind the engine is that banners from IoT devices usually contain keywords, which when searched on Google, lead to webpages that contain the respective device's information. ARE learns rules by processing the banner data in the following fashion:

(1) Preprocessing: Remove irrelevant text (dictionary words/ stop words/ tags, etc) from the banner. The remaining words are grouped to get queries to be searched on Google.

(2) Named Entity Recognition: For each query from (1), the top ten webpages returned by Google Search are processed to extract device types (by matching against a predefined list), vendors (by matching against a predefined list), and product names (extracted using a regular expression).

(3) Local Dependency: The extracted list of device types, vendors, and product names is then examined for local dependency, i.e, whether they appear together in the text in a certain order.[2] The entities that appear together in the text are grouped into a device annotation tuple, of the form ⟨device type, vendor, product⟩ or ⟨device type, vendor⟩ if the product is missing.

(4) Apriori Algorithm: Each device annotation tuple gives a new transaction of the form: banner => ⟨device type, vendor, product⟩ (or banner => ⟨device type, vendor⟩). ARE then leverages the Apriori algorithm to infer rules from the transactions. The Apriori algorithm is a data mining algorithm used for finding what items frequently appear together in an item-set dataset. An item-set dataset contains entries called "transactions" comprising of multiple items (the length of the transaction can vary). The algorithm learns rules of the format: *if x then y*, indicating if x appears in the database, then usually y appears with it. We explain this further by using the following example transactions in the format {banner, device type, vendor, product(?)}:

```
{mikrotik router v500.1,router,mikrotik,ipad 2},
{mikrotik router v500.1,router,tenda},
{mikrotik router v500.1,alarm system, mikrotik},
{mikrotik router v500.1,router,mikrotik,500.1},
{SSH 2.0 dropbear_2019,security camera,tenda,p9.0},
{SSH 2.0 dropbear_2019,smart coffee machine,smartBrew},
{SSH 2.0 dropbear_2019,nas, microsoft,i9}
```

The Apriori algorithm first finds frequent item sets in a bottom up manner, by defining a *support* threshold parameter, and only retaining the itemsets that pass the threshold at each level. For a support threshold of 2, the frequent item-sets for the above transactions are as follows:

Level 1: Individual items

```
mikrotik router v500.1    4
router                    3
mikrotik                  3
SSH 2.0 dropbear_2019     3
```

Level 2: Two-item combinations

```
mikrotik router v500.1, router    3
mikrotik router v500.1, mikrotik  2
```

Level 3: Three-item combinations

```
mikrotik router v500.1, router, mikrotik  2
```

Next, the algorithm extracts rules of the form *if x then y* from the frequent item sets. Here, the algorithm uses a second parameter *confidence*, defined as:

[1] We obtained these numbers by parsing the fingerprint database available at: https://svn.nmap.org/nmap/nmap-service-probes.

[2] We refer the reader to Figure 3 in the original ARE paper for details.

$$\text{Confidence}(X \rightarrow Y) = \frac{\text{Support count}(X \cup Y)}{\text{Support count}(X)}$$

A rule X → Y having confidence of 70% means that at least 70% of the times X appears together with Y. The rules that satisfy a confidence threshold of 70% in this example are as follows:

From Level 2:

```
mikrotik router v500.1 => router     3/4 (75%)
router => mikrotik router v500.1     3/3 (100%)
mikrotik router v500.1 => mikrotik   3/4 (75%)
mikrotik  => mikrotik router v500.1  3/3 (100%)
```

From Level 3:

```
[router, mikrotik]  => mikrotik router v500.1 2/2 (100%)
```

Based on the above rules, the banner “mikrotik router v500.1”, will be labeled as ⟨device_type: router, vendor: mikrotik⟩.

3 RE-IMPLEMENTING ARE

In this section, we list the ambiguities we found while attempting to re-implement ARE, and how we handled them. Such ambiguities often arise because research papers in general strive to convey the high-level ideas and findings of the approach as clearly as possible, while keeping the implementation details to a minimum.[3] However, as we highlight below with the help of examples from the ARE paper, these omissions can hinder a faithful re-implementation:

(1) After removing the tags, stop words, and dictionary words from the banner data, the next step is to generate a *sequence* of queries from the remaining keywords, but the paper is unclear on how exactly to do this. The paper also mentions using Term Frequency-Inverse Document Frequency (TF-IDF) values to find *important* keywords, implying these are more deserving to be present in the queries, but is ambiguous on how the TF-IDF values are eventually used. In our implementation, we use TF-IDF to pick the *five* most important words from the banner.[4] We then form a search string from these words, by joining them with the space separator.

(2) The ARE engine then extracts device types, vendors, and product names from the crawled webpages by using dictionary lists and regexes. The paper is unclear on how to get the list of 1522 vendors (in Table 1 of the paper) the authors used for this extraction. We used a list of 3,194 unique vendors obtained from www.iotone.com/suppliers (3,008) and the Ztag database (186).

(3) Related to (2), the regex for extracting product names given in Table 1 of the paper seems to be broken. It does not, for example, detect the following product names in Table 2 of the paper: SRX210HE, P-660HN-51, HL-3170CDW. We initially used this regex in our implementation, but upon observing that it does not extract any product name from webpages, we modified the regex. We leveraged the same idea as described in Section 3.3 of the original paper, i.e., product names typically contain a combination of letters and numbers, perhaps containing the symbol “-”. We picked 20 random vendors from our vendor list, and searched the Internet for their products, compiling a list of product names. In addition, we added to this list the product names listed in Table 2 of the original paper that should have been matched by the original regex. The modified regex we use is: [A-Za-z]+[-]?[A-Za-z]*[0-9]+[-]?[-]?[A-Za-z0-9].*?[0-9a-zA-Z]* and it catches all the 25 product names in our list.

Since we based the regex modifications on a seed list of 25 product names, we then determined how well it performs when run on webpages. We tested the regex on a random sample of 10 webpages, returned by our query searches against our ground truth Censys dataset. The old regex does not match any products (zero TPs and zero FPs), whereas our modified regex resulted in 108 TPs and 504 FPs. Since the product name format matches many other non product name strings, false positives are an inherent limitation of the regex approach to extract product names. However, we note that these false positives get filtered further down the pipeline because most of these strings do not have a local dependency with the vendor and device type in the text.

(4) The paper also mentions that ARE should filter out banners from non-IoT devices. However, further details on how to detect whether a banner is from an IoT device seem to be incomplete. The authors only provide filtering rules for HTTP banners, omitting the other banner types. Further, for HTTP, the only unambiguous filter mentioned is that banners containing heavyweight server names like Apache, IIS, and Nginx should be dropped. The paper lacks details on the thresholds for the other two HTTP filtering heuristics, i.e., (i) the number of scripts, words, pictures in the webpage of IoT devices is small, and (ii) the number of external links in webpages of IoT devices is small. Given the lack of thresholds, our implementation only performs filtering on the three server names mentioned in the paper.

(5) Finally, the paper is unclear on whether the local dependency finding (between device type, vendor and product) is done per sentence or per page. We do it per sentence since it seems to be the more logical choice from a natural language point of view.

4 DATASETS

The original ARE paper uses two datasets for evaluation. The first contains 350 randomly chosen IoT devices from the Internet. This dataset contains 4 different device types (NVR, NVS, router, and ipcamera), 64 vendors, and 314 products. The second contains 1,000 devices across 10 device types and 77 vendors. (The paper does not mention the list of device types and number of products in the second dataset). The authors obtained ground truth labels for the first dataset by searching the application layer responses on the Internet, and manually labeling them. It is unclear how the authors obtained ground truth labels for the second dataset. We refer to the first dataset as Internet-350 and the second as Internet-1000 in this paper.

Since these datasets are not publicly available, we collect two different ground truth datasets for evaluation of ARE. The two datasets are complementary; the first contains banners from publicly accessible devices (replicating the strategy in the ARE paper), and the second contains banners from devices in home networks behind NATs. The first dataset contains similar banner types as in the original paper, whereas the second dataset contains additional banner types. This helps us study the power of the ARE approach

[3] As these can be distracting for the majority of the readers not interested in reproducing the work.

[4] We chose five based on the observation that a higher number of keywords leads to very specific queries for which Google does not return any search results. Using a lower number of keywords results in missing some important keywords. We did not however carry out rigorous experimentation to determine the optimal value.

Table 1: Evaluation datasets used in this work.

	Censys	Ferret
Unique banners	4,919	129
Unique device types	28	9
Unique vendors	76	22
Unique products	564	63

Table 2: Performance comparison of our implementation with the original implementation in the ARE paper.

Dataset	Implementation	Precision	Recall
Internet-350	ARE paper	95.7%	94.9%
Internet-1000	ARE paper	97.5%	(Not reported)
Censys-4919	This paper	62.5%	1.0%
Ferret	This paper	4.0%	20.8%

on two complementary datasets. Table 1 summarizes our datasets and we describe them below:

Censys dataset: We collected this dataset as follows: we first queried Censys and obtained the list of all the IP addresses where the corresponding Ztag labels and at least one of FTP, TELNET, and HTTP banners was available. We obtained this dataset on Feb 20, 2020. From this list, we picked 10K IP addresses randomly[5], and obtained the corresponding banners from Censys. The dataset contains 4,919 unique banners [3,783 HTTP, 957 FTP, and 179 TELNET], corresponding to 28 device types, 76 vendors, and 564 products. We use the Ztag labels from Censys as ground truth.

Home network (Ferret) dataset: Our second dataset was collected using an Android application, Ferret. This application scans the given network for connected devices, and probes them to collect HTTP, SSH, and UPnP banners. The application automatically identifies the vendor by looking up the MAC address of the device in the OUI (Organizationally Unique Identifier) database [5]. The application then also prompts the user to identify the devices by presenting them an interface with a list of IP addresses, along with instructions on how to identify the device type, vendor (if not already identified by OUI lookup), and product corresponding to a given IP address. This gives us the ground truth labels, which we then manually checked, discarding any banners with bogus labels. We asked volunteers (students in an undergraduate Computer Security class) to run the application in their home networks, and obtained banner data from 69 networks. This dataset contains 129 unique banners [4 HTTP, 34 SSH, and 91 UPNP], corresponding to 09 unique device types, 22 vendors, and 63 products.

We make both the datasets available to the community. For the Ferret dataset, we anonymized any unique identifiers in the dataset (for example, the "uuid" field in UPNP banners).

5 EVALUATION

For evaluation, we use the same support and confidence threshold values, as in the paper (0.1% support and 50% confidence). We use the top *five* keywords as search queries identified by TF-IDF. We mention the values of all the configurable parameters in the `config.py` file provided in the source code (these include parameters we used for each stage of the pipeline, such as web page download timeout and maximum file size for the web crawler stage).

Table 3: ARE pipeline numbers on the Censys and Ferret datasets.

	Censys	Ferret
Unique banners	4,919	129
Valid IoT banners	4,025	125
Query Generation		
At least one query word	4,025	125
Web Crawler		
At least one search result	2,710	73
Named Entity Recognition		
At least one device type	2,672	71
At least one vendor	2,690	73
At least one product label	2,673	73
Local Dependency Finder		
At least one <device, vendor>	2,189	26
At least one <device, vendor, product>	1,569	13
Apriori Algorithm		
At least one rule	40	26

Table 4: Performance of ARE pipeline with varying number of banners in the Censys dataset.

Banners	IoT Banners	Precision	Recall
500	402	38.23%	26.37%
1K	810	56.92%	11.98%
2K	1,627	79.62%	13.95%
3K	2,458	85.71%	9.56%
4K	3,284	79.38%	3.90%
4.9K	4,025	62.5%	1.0%

In addition to the design decisions mentioned in Section 3, we add the following capabilities to our implementation in order to handle our datasets: (i) we use a wider device-type list than the original ARE implementation; the original paper uses a list of 21 device types — we use a list comprising all the device types that Nmap, Ztag, and the original ARE implementation can label (103 in total: 21 ARE and 82 Nmap and Ztag). This is required because our ground truth labels (obtained from Ztag and manual labeling) contain more device types than the original ARE paper.[6] Further, using a wider list ensures that the engine is not dataset specific, rather it is able to handle IoT device types from any dataset, and (ii) we add pre-processing support for UPnP banners to process our second dataset, as the original paper does not handle them. We observed that the UPnP banners contain timestamps and unique

[5] using the Mersenne Twister algorithm in Python's random package, as in the original paper.

[6] For example, our dataset includes scada controller, environment monitor, and power monitor device types, not present in the ARE device types list.

identifiers that when present in the search queries do not result in any search results. We add pre-processing to remove them.[7]

Metrics: The ARE paper uses *coverage: TP/(FP+FN)* and *precision: TP/(FP+TP)* to evaluate the quality of the learnt rules. We find that the paper is unclear on how the authors define false positives and false negatives in this context (is it at the granularity of rules or banners?). Further, given that the engine can learn multiple rules against a banner, the authors do not clarify whether they pick the highest confidence rule against each banner while computing the metrics or keep all the rules for the given banner. We keep all the learnt rules, since a given banner can indeed correspond to multiple device types, vendors, and products. We define a *rule* as *false positive* if it labels the banner incorrectly, and *true positive* if it labels the banner correctly. A *false negative* "rule" is not meaningful in this context. We use the same definition of precision as in the paper i.e, TP/ (FP+TP), but compute coverage as the number of labeled banners out of the total valid IoT banners in the dataset. (Given the above discussion, we believe that the definition of coverage in the original paper i.e, TP/(FP + FN) is incorrect).

We now discuss ARE's performance on our datasets.

Coverage: ARE learnt rules for 40 out of the 4,919 banners in the Censys data, and 26 out of the 125 banners in the Ferret data, resulting in very poor coverage. In contrast, the ARE paper claims a coverage of 94.9% on the Internet-350 ground truth dataset. The authors do not report coverage on the Internet-1000 dataset.

Table 3 sketches how many banners get dropped at each stage of the ARE pipeline. After filtering the error and non-IoT banners, 82% Censys and 97% Ferret banners remain. We compute the percentage reduction for the rest of the stages with respect to the banners retained in this stage (i.e., 4,025 banners in case of Censys and 125 in case of Ferret). The next stage of searching queries corresponding to these banners on Google results in a significant drop of banners; only 67% and 58% of the valid IoT banners from the previous stage result in at least one webpage. We do not see a considerable drop in the next stage, i.e., named entity recognition, indicating that the pages are high quality, containing at least one device type, vendor, or product. Local dependency detection further reduces the banners to 54.4% in case of Censys and 20.8% in the case of Ferret. Finally, after the Apriori algorithm stage only 1% of Censys banners and 20.8% of Ferret banners remain.

We ran the ARE engine on different subsets of the Censys dataset to study how the precision and recall vary with the number of banners in the dataset. Table 4 shows decreasing coverage with increasing number of banners. This could be due to two reasons, (i) the search queries formed are dependent on what other banners are present in the dataset (due to TF-IDF weighting), and (ii) the Apriori algorithm results are affected by what other banners are present in the dataset. Increasing the number of banners results in an increase in the number of transactions, which may result in the correct transactions not finding enough *support* to form a rule. The ARE paper does not comment on the decreasing coverage with the increasing number of banners.

Precision: ARE's precision is also fairly low compared to the 95-97% precision claimed by the original paper. 35 out of the 56 rules ARE learnt on the Censys dataset are correct, and 11 out of the 276 rules on the Ferret dataset are correct. The device types in the correct rules are { camera, nas, printer, router } and the vendors are { axis, hp, synology, tp-link }. The identified vendor in the correct rules is always present in the banner itself (with one exception, i.e., tp-link). All the correct rules are of the form: <device only>, <vendor only>, or <device, vendor>. None of the correct rules contain products. These results indicate that most of the banners that ARE labels correctly could also have been easily labeled via a much simpler approach, i.e., directly extracting device types and vendors from the banners.

6 REPRODUCING RESULTS OF THIS WORK

We provide the data and source code used in this work at [6]. The README file explains how to set up and run the code. We caution the reader that the pipeline involves fetching data from the Internet and because the returned search results may vary by time and location of querying, it may not be possible to obtain the exact precision and recall numbers we report (but the reproduced numbers should be close). Further, reproducing results on the full Censys-4919 dataset requires access to a paid Google search JSON API key.[8] However, reproducing results on the Ferret data and a subset of Censys data should be possible with a free version of the API.

7 CONCLUSION

We re-implemented a recently proposed framework for automatically learning rules to label IoT devices. While pursuing this effort, we encountered several ambiguities regarding implementation and evaluation details, which we documented in this work. Due to the lack of availability of groundtruth datasets used in the original paper, we evaluated our implementation on datasets different than the original paper, finding that we fail to achieve the accuracy reported by the ARE authors. Our lack of success in reproducing the results of the original paper could be due to differences (including potential errors) in our implementation, differences in evaluation metrics, or due to the different evaluation datasets we used.

Our reproducibility effort highlights that seemingly unimportant details often omitted in a research paper can lead to an increased cost in terms of effort for follow-on work that attempts to build upon the original work. While it may not be possible to carefully document every detail and control knob setting in the main body of the paper, adopting practices to share datasets and source-code (including a config file detailing the settings used), or a technical report detailing the experimental conditions under which the authors achieved the results is necessary for the community to build on the original work. For best practices and workflows to help improve reproducibility of a research work, we refer the interested readers to *The Dagstuhl Beginners Guide to Reproducibility for Experimental Networking Research* [3]. Another good resource on the topic is the more extensive Dagstuhl Seminar 18412 report: *Encouraging Reproducibility in Scientific Research of the Internet* [2], which documents broader community discussions on moving towards a culture of reproducible research.

[7] We refer the reader to our source code for pre-processing details.

[8] Custom Search JSON API provides 100 search queries per day for free. Additional requests cost $5 per 1000 queries, up to 10k queries per day.

REFERENCES

[1] Manos Antonakakis, Tim April, Michael Bailey, Matthew Bernhard, Elie Bursztein, Jaime Cochran, Zakir Durumeric, J. Alex Halderman, Luca Invernizzi, Michalis Kallitsis, Deepak Kumar, Chaz Lever, Zane Ma, Joshua Mason, Damian Menscher, Chad Seaman, Nick Sullivan, Kurt Thomas, and Yi Zhou. 2017. Understanding the Mirai Botnet. In *Proceedings of the 26th USENIX Conference on Security Symposium (SEC'17)*. USENIX Association, Berkeley, CA, USA.

[2] Vaibhav Bajpai, Olivier Bonaventure, kc claffy, and Daniel Karrenberg. 2019. Encouraging Reproducibility in Scientific Research of the Internet. *Dagstuhl Reports* 8, 10 (Jan 2019), 41–62.

[3] Vaibhav Bajpai, Anna Brunstrom, Anja Feldmann, Wolfgang Kellerer, Aiko Pras, Henning Schulzrinne, Georgios Smaragdakis, Matthias Wählisch, and Klaus Wehrle. 2019. The Dagstuhl Beginners Guide to Reproducibility for Experimental Networking Research. *SIGCOMM Comput. Commun. Rev.* 49, 1 (2019), 24–30.

[4] Xuan Feng, Qiang Li Li, Haining Wang, and Limin Sun. 2018. Acquisitional rule-based engine for discovering internet-of-things devices. In *Proceedings of the 27th USENIX Conference on Security Symposium (SEC'18)*. USENIX Association.

[5] IEEE. 2019. IEEE Registration Authority: Assignments. https://regauth.standards.ieee.org/standards-ra-web/pub/view.html#registries.

[6] Talha Javed. 2020. Accompanying Source Code and Datasets for this Work. https://github.com/talhajavedmukhtar/ARE-rt.

[7] Nmap. 2019. Nmap: the Network Mapper - Free Security Scanner. https://nmap.org.

[8] ZTag. 2019. Zmap/Ztag: Tagging and Annotation Framework for Scan Data. https://github.com/zmap/ztag.