Acquisitional Rule-based Engine for Discovering Internet-of-Thing Devices

Xuan Feng, *Beijing Key Laboratory of IOT Information Security Technology, IIE, CAS, China, and School of Cyber Security, University of Chinese Academy of Sciences, China;* Qiang Li, *School of Computer and Information Technology, Beijing Jiaotong University, China;* Haining Wang, *Department of Electrical and Computer Engineering, University of Delaware, USA;* Limin Sun, *Beijing Key Laboratory of IOT Information Security Technology, IIE, CAS, China, and School of Cyber Security, University of Chinese Academy of Sciences, China*

https://www.usenix.org/conference/usenixsecurity18/presentation/feng

This paper is included in the Proceedings of the
27th USENIX Security Symposium.

August 15–17, 2018 • Baltimore, MD, USA

ISBN 978-1-939133-04-5

Open access to the Proceedings of the
27th USENIX Security Symposium
is sponsored by USENIX.

Acquisitional Rule-based Engine for Discovering Internet-of-Thing Devices

Xuan Feng[12], Qiang Li[3*], Haining Wang[4], Limin Sun[12]

[1] Beijing Key Laboratory of IOT Information Security Technology, IIE, CAS, China
[2] School of Cyber Security, University of Chinese Academy of Sciences, China
[3] School of Computer and Information Technology, Beijing Jiaotong University, China
[4] Department of Electrical and Computer Engineering, University of Delaware, USA

Abstract

The rapidly increasing landscape of Internet-of-Thing (IoT) devices has introduced significant technical challenges for their management and security, as these IoT devices in the wild are from different device types, vendors, and product models. The discovery of IoT devices is the pre-requisite to characterize, monitor, and protect these devices. However, manual device annotation impedes a large-scale discovery, and the device classification based on machine learning requires large training data with labels. Therefore, automatic device discovery and annotation in large-scale remains an open problem in IoT. In this paper, we propose an Acquisitional Rule-based Engine (ARE), which can automatically generate rules for discovering and annotating IoT devices without any training data. ARE builds device rules by leveraging application-layer response data from IoT devices and product descriptions in relevant websites for device annotations. We define a transaction as a mapping between a unique response to a product description. To collect the transaction set, ARE extracts relevant terms in the response data as the search queries for crawling websites. ARE uses the association algorithm to generate rules of IoT device annotations in the form of (type, vendor, and product). We conduct experiments and three applications to validate the effectiveness of ARE.

1 Introduction

Nowadays most of the industries have owned and run different Internet-of-Thing (IoT) devices, including, but not limited to, cameras, routers, printers, TV set-top boxes, as well as industrial control systems and medical equipment. Many of these devices with communication capabilities have been connected to the Internet for improving their efficiency. Undeniably, the development and adoption of online IoT devices will promote economic growth and improvement of the quality of life. Gartner reports [1] that nearly 5.5 million new IoT devices were getting connected every day in 2016, and are moving toward more than 20 billion by 2020.

Meanwhile, these IoT devices also yield substantial security challenges, such as device vulnerabilities, mismanagement, and misconfiguration. Although an increasingly wide variety of IoT devices are connected to residential networks, most users lack security concerns and necessary skills to protect their devices, e.g., default credentials and unnecessary exposure. It is difficult for end users to identify and troubleshoot the mismanagement and misconfiguration of IoT devices. Even if an IoT device has a serious security vulnerability, users have no capability of updating patches in a timely manner due to their limited knowledge.

In general, there are two basic approaches to addressing security threats: reactive defense and proactive prevention. The reactive defense usually requires downloading firmware images of devices for offline analysis, leading to a significant time latency between vulnerability exploit and detection [38]. By contrast, a proactive security mechanism is to prevent potential damages by predicting malicious sources, which is more efficient than the reactive defense against large-scale security incidents (e.g., Mirai Botnet [21]). In order to protect IoT devices in a proactive manner, discovering, cataloging, and annotating IoT devices becomes a prerequisite step.

The device annotation contains the type, vendor, and product name. For instance, an IoT device has a type (e.g., routers or camera), comes from a vendor (e.g., Sony, CISCO, or Schneider), with a product model (e.g., TV-IP302P or ISR4451-X/K9). The number of device annotations is enormous, and we cannot enumerate them by human efforts. In prior works [21, 25, 28, 35–37, 40], fingerprinting and banner grabbing are the two conventional methods for discovering and annotating devices. However, the fingerprinting approach [35, 36, 40] cannot be applied to the IoT device discovery and annota-

[*] Qiang Li is the corresponding author.

tion because of the high demand for training data and a large number of device models. The banner grabbing approach [21,25,28,37] usually generates device annotations in a manual fashion, which is impossible for large-scale annotations, particularly given the increasing number of device types. In this paper, we aim to automatically discover and annotate IoT devices in the cyberspace while mitigating the cost in terms of manual efforts and the training data.

The key observation we exploit is that the response data from those IoT devices in application layer protocols usually contain the highly correlated content of their manufacturers. A variety of websites on the Internet are used to describe the device products since their initial sale, such as description webpages of the products, product reviews websites, and Wikipedia. Our work is rule-based, and the automatic rule generation is mainly based on the relationship between the application data in IoT devices and the corresponding description websites. Although the basic idea is intuitive, there are two major challenges in practice, blocking the automation process of building rules for IoT devices. First, the application data is hardcoded by its manufacturer. Second, there are massive device annotations in the market. Notably, manufacturers would release new products and abandon outdated products, due to the business policy. Manually enumerating every description webpage is impossible.

To address these technical challenges, we propose an Acquisitional Rule-based Engine (ARE) that can automatically generate rules for discovering IoT devices in the cyberspace. Specifically, ARE utilizes the transaction dataset to mine rules. We define a transaction as a mapping between a unique response from an IoT device to its product description. ARE collects the transaction dataset as follows: (1) ARE receives the application-layer response data from online IoT devices; (2) ARE uses relevant terms in the response data as the search queries; and (3) ARE crawls the websites from the list of the searching result. For those relevant webpages, ARE uses named-entity recognition (NER) to extract device annotation, including device type, vendor, and product. ARE learns rules from the transaction dataset through the apriori algorithm. Furthermore, ARE provides RESTful APIs to applications for retrieving the rules for discovering and annotating IoT devices in the cyberspace.

We implement a prototype of ARE as a self-contained piece of software based on open source libraries. We manually collect two datasets as the ground truth to evaluate the performance of ARE rules. ARE is able to generate much more rules than the latest version of Nmap in a much shorter time. Our results show that the ARE rules can achieve a precision of 96%. Given the same number of application packets, ARE can find more IoT devices than Nmap tool. Note that ARE generates rules without the human efforts or the training data, and it can dynamically learn new rules when vendors distribute new products online.

To demonstrate the effectiveness of ARE, we perform three applications based on IoT device rules. (1) The Internet-wide Device Measurement (IDM) application discovers, infers and characterizes IoT devices in the entire IPv4 address space (close to 4 billion addresses). The number of IoT devices exposed is large (6.9 million), and the distribution follows long-tail. (2) The Compromised Device Detection (CDD) application deploys 7 honeypots to capture malicious behaviors across one month. CDD uses ARE rules to determine whether the host is an IoT device. We observe that thousands of IoT devices manifest malicious behaviors, implying that those devices are compromised. (3) The Vulnerable Device Analysis (VDA) application analyzes the vulnerability entries with device models. We observe that hundreds of thousands of IoT devices are still vulnerable to malicious attacks.

Furthermore, ARE enables the security professionals to collect the device information by leveraging those rules in a large-scale measurement study or security incident. To facilitate this, 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.

In summary, we make the following contributions.

- We propose the framework of ARE to automatically generate rules for IoT device recognition without human effort and training data.
- We implement a prototype of ARE and evaluate its effectiveness. Our evaluation shows that ARE generates a much larger number of rules within one week and achieves much more fine-grained IoT device discovery than existing tools.
- We apply ARE for three different IoT device discovery scenarios. Our main findings include (1) a large number of IoT devices are accessible on the Internet, (2) thousands of overlooked IoT devices are compromised, and (3) hundreds of thousands of IoT devices have underlying security vulnerabilities and are exposed to the public.

The remainder of this paper is organized as follows. Section 2 provides the background of device discovery, as well as our motivation. Section 3 describes how the core of ARE, i.e., the rule miner, derives rules of IoT devices. Section 4 details the design and implementation of ARE. Section 5 presents the experimental evaluation of ARE. Section 6 illustrates the three ARE-based applications. Section 7 surveys the related work, and finally, Section 8 concludes.

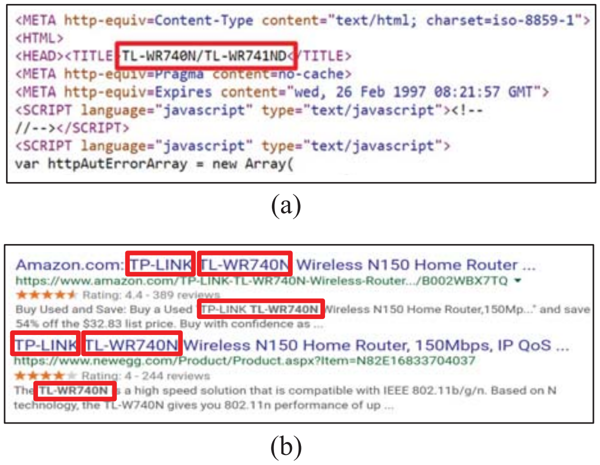
2 Background and Motivation

In this section, we first present the background of IoT device discovery and annotation. Then, we describe the motivation for automatic rule generation.

2.1 IoT Device Discovery

Fingerprinting-based Discovery. In network security, fingerprinting has been used for more than two decades, which requires a set of input data and a classification function. The focus of the prior research [40] [36] [35] is on the fingerprints of operating systems (OS) rather IoT devices. To fingerprint an IoT device, the input data includes a pair of queries and responses from IoT devices, and the class label (known as category or target) is what the IoT device belongs to. The learning algorithms infer a classification model for mapping the input data to the class labels based on the training data. When the number of class labels is large, the learning algorithms require a large training data to achieve high precision and coverage. However, currently there is no training data for IoT devices. In contrast to the limited number of OS classes, the number of device models is vast, and it is infeasible to collect the training data manually. A device class includes device type, vendor, and product model. To bootstrap our research, we have scraped some websites collecting about 1,000 IoT device manufacturers, and every vendor has hundreds of products. Also, it is noteworthy that the number of products we have collected is substantial, but it only constitutes a small portion of IoT devices as the number of IoT devices continues growing at even a faster pace. Therefore, it is very challenging to collect a significant amount of the training data that is sufficient for IoT device fingerprinting.

Banner-grabbing Discovery. In practice, researchers use banner grabbing [21, 25, 28, 37], instead of fingerprinting, to discover IoT devices, due to a large number of IoT devices and the lack of training data. Banner-grabbing is to extract textual information in the application layer data for labeling an IoT device. Antonakakis *et al.* [21] applied the Nmap [8] banner rules to analyze online devices from CENSYS and Honeypot. Fachkha *et al.* [28] wrote rules through manual efforts to identify industrial control system in the cyberspace. Shodan [37] and Censys [25] are two popular search engines for discovering online devices. They both execute Internet-wide scans with different protocols (e.g., HTTP, SSH, FTP, and Telnet). Shodan also utilizes the set of rules combined with the Nmap tool and manual collection. Censys utilizes Ztag [16] to identify online devices, which requires annotations for new types of devices. However, the rule generation in banner grabbing is a manual process. The technical knowledge is needed to write a rule for banner grabbing. This manual process is often arduous and incomplete, making it difficult to keep up-to-date with the increasing number of device models. So far, Nmap has several thousand rules for device discovery (over multi-year development). Moreover, the banner information itself is always incomplete, only containing a part of device annotation.



```
<META http-equiv=Content-Type content="text/html; charset=iso-8859-1">
<HTML>
<HEAD><TITLE>TL-WR740N/TL-WR741ND</TITLE>
<META http-equiv=Pragma content=no-cache>
<META http-equiv=Expires content="wed, 26 Feb 1997 08:21:57 GMT">
<SCRIPT language="javascript" type="text/javascript"><!--
//--></SCRIPT>
<SCRIPT language="javascript" type="text/javascript">
var httpAutErrorArray = new Array(
```

(a)

Amazon.com: TP-LINK TL-WR740N Wireless N150 Home Router ...
https://www.amazon.com/TP-LINK-TL-WR740N-Wireless-Router.../B002WBX7TQ ▾
★★★★★ Rating: 4.4 - 389 reviews
Buy Used and Save: Buy a Used "TP-LINK TL-WR740N Wireless N150 Home Router,150Mp..." and save 54% off the $32.83 list price. Buy with confidence as ...

TP-LINK TL-WR740N Wireless N150 Home Router, 150Mbps, IP QoS ...
https://www.newegg.com/Product/Product.aspx?Item=N82E16833704037
★★★★☆ Rating: 4 - 244 reviews
The TL-WR740N is a high speed solution that is compatible with IEEE 802.11b/g/n. Based on N technology, the TL-W740N gives you 802.11n performance of up ...

(b)

Figure 1: The application layer data (HTML) appears in the online embedded devices. (b) There are several relevant websites about this device in the search engine.

2.2 Automatic Learning Rules

Our Motivation. As we mentioned before, manufacturers usually hardcode the correlated information into IoT devices to distinguish their brands. After the initial sale of products, there are many websites describing device products such as product reviews. As an example, Figure 1(a) shows the response packet of an online IP-camera having the term “TL-WR740/TL-WR741ND” in the HMTL file. If we use “TL-WR740/TL-WR741ND” as the search query in the Google search engine, we will obtain a URL list including the description documents. Figure 1(b) shows that Amazon and NEWEGG websites provide the annotation description for this device. In the development of ARE, we leverage a set of existing tools (web crawler, NLP, and association algorithms) to address several practical problems in the process of automatic rule generation. These techniques are briefly introduced below.

Web Crawler. ARE needs to find the description webpages for IoT devices. It is a challenging task to crawl every webpage, especially given that we cannot catalog every IoT device. Fortunately, today’s search engines have crawled the Web and found documents to add to their searchable indexes. The search engines also keep the history snippets even if a product is out-of-date without correlated webpages. We propose to use a search query to narrow down the scale of web crawling. ARE selects the terms from the response data and encapsulates them into a query. For instance, a search query (Figure 1) is formatted as “search engine/search?hl=en&q=%22TL+WR740N+WR741ND+&btnG=Search”, where the mark (?) indicates the end of the URL and (&) separates arguments, q is the start of the query, the plus mark $(+)$ represents a space, and $btnG = Search$ denotes that the search button is pressed on the web interface. The web crawler obtains the description webpages from the search result list.

Natural Language Processing. To present IoT device annotation, ARE needs to extract the relevant terms from a related description website. Name Entity Recognition (NER) is used to determine the words into predefined classes, such as organization names and location names. NER is a typical technique for processing natural language. The problem is that NER cannot directly identify device annotations from the description websites. The reason is that the standard NER is highly domain-specific, not designed for extracting device annotations and cannot achieve high precision. In this paper, ARE uses a rule-based NER and local dependency to identify device entities.

Data Mining. ARE needs to discover and infer the relationships from the transaction set. Specifically, the association algorithms (as a set of data mining) can identify the relationships between items, and then derive the rules. ARE utilizes the association algorithms to generate the IoT device rules. There are two parameters affecting the association algorithms, support and confidence. ARE will choose an item whose value is larger than the minimum support and generate rules whose values are larger than the minimum confidence.

3 Rule Miner

Prior work [21, 25, 28, 37] used the banner grabbing to discover and annotate devices. Developers manually write those rules. Over its 20-year development, Nmap has encouraged developers to write rules to expand its library. In this paper, we propose a rule miner for automating the rule generation process without any human efforts or training data. It can derive additional rules that are missed by developers. Moreover, the rule miner learns new rules dynamically over time.

3.1 Transaction

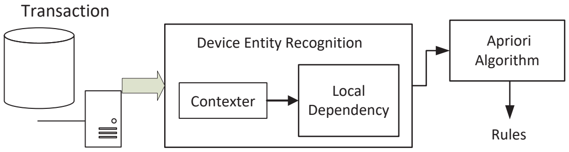


Figure 2: Rule miner for automatic rule generation.

A manufacturer usually plants its information into the product’s application layer data. Also, there are many websites including product information, such as product reviews and official documents. Such product information plays a vital role in the rule miner. We define the concept of “transaction” to associate the application-layer data from an IoT device with the corresponding description of an IoT device in a webpage, and our rule generation is based on the transaction set.

Definition 1 *Transaction: a transaction is a pair of textual units, consisting of the application-layer data of an IoT device and the corresponding description of an IoT device from a webpage.*

Based on the definition 1, the transaction set can be formatted as $T = \{t_1,\ t_2,\ ...,\ t_m\ \}$, where m is the number of transactions. Each transaction can be formulated as $t_i = \{\ p_i,\ w_j\}$, where t_i contains two parts: (1) p_i is the application-layer data of the device i and (2) w_j is the description webpage j. We use the response data to approximately represent p_i from the ith device. For the jth webpage, multimedia information (e.g., advertisements, audio, and videos) should be removed and the textual information is used to approximately represents w_j.

For application-layer data p_i, we convert the response data into the sequence of search queries $\{q_i^1, q_i^2, ..., q_i^k\}$, where k is the number of the query sequence (detailed in Section 4.2). We use the search query to crawl webpages, and the search engine would return a search list $\{w_1, w_2, ..., w_l\}$. For the webpage w_j, we extract the device annotation from its textual content (detailed in Section 3.3). Note that compared with fingerprinting and banner grabbing techniques, our transaction collection is an automated process without human effort.

3.2 Overview of Rule Miner

Based on the extracted features (i.e., search queries and device annotations) from the transaction set, which characterize the association between a response data and a webpage, we define a rule in its general format as $\{l_1^i, l_2^i, ..., l_n^i\} \Rightarrow \{t^j, v^j, p^j\}$. The value i denotes an IoT device i, , and l_1^i to l_n^i is the keywords extracted from the application layer data. The tuple (t^j, v^j, p^j) extracted from the webpage j indicates the device type, device vendor, and device product, respectively.

Table 1: Context textual terms.

Entity	Context terms
Device Type	camera, ipcam, netcam, cam, dvr, router nvr, nvs, video server, video encoder, video recorder diskstation, rackstation, printer, copier, scanner switches, modem, switch, gateway, access point
Vendor	1,552 vendor names
Product	[A-Za-z]+[-]?[A-Za-z!]*[0-9]+[-]?[-]?[A-Za-z0-9] *∧[0-9]2,4[A-Z]+

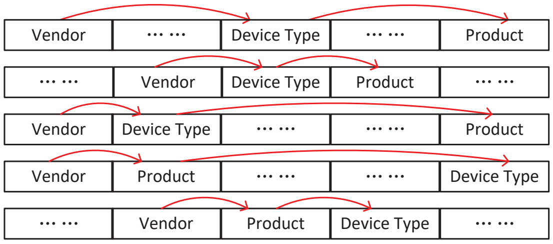


Figure 3: The local dependency of the device entity.

As defined above, a rule is an association between a few features extracted from the application-layer data and the device annotation extracted from relevant webpages. Here we use A to denote the features extracted from the application-layer data in IoT devices, and use B to denote the device annotation extracted the description webpages. A rule can be described as the format $\{A \Rightarrow B\}$. The goal of the rule miner is to learn the rules of IoT devices in an automatic manner.

Figure 2 presents the overview of the rule miner, illustrating how it learns the rules of IoT devices. In the transaction set, every transaction contains the application-layer data and the relevant webpages. To easily represent the annotation, the rule miner applies the unified form *(device type, vendor, product)* for describing IoT devices. We propose device entity recognition (DER) to extract this information from webpages. DER is derived from the NER technique in the NLP tools, and uses the contexter and local dependency among words to identify the device information. The rule miner uses the apriori algorithm to learn the relationship between A and B. Although the apriori algorithm is straightforward, it is able to generate rules satisfying the inference process for discovering and annotating IoT devices.

3.3 Device Entity Recognition

As aforementioned, a standard NER is not designed for extracting IoT device information. If we directly apply NER to the description webpage, the precision is poor due to the fact that NER is highly domain-specific. We propose the device entity recognition (DER), derived from NER. DER defines three classes (type, vendor, product) to represent the device annotation, including device types, vendors, and product names, respectively. Relevant words in a webpage would be classified as one label among three predefined classes.

DER is a combination of the corpus-based NER and rule-based NER. In the corpus-based NER, we are interested in device types and vendor names. Table 1 presents 21 words for IoT device types and 1,552 different terms for vendor names. For a device type, we enumerate common device types, including router, camera, TV set, modem, and printer. They are typical consumer-oriented IoT devices, which are connected to the Internet. For a device vendor, we enumerate vendors from Wikipedia and manually collect from official vendor websites. We only need one hour to collect device types and vendors, which is a very reasonable manual effort for the DER module. If a new device type and vendor is added, we will update the corpus list for DER. Note that the device type and vendor can be easily expanded.

In the rule-based DER, we use regular expressions to extract the product name entity. The challenge here is that the number of product names is too large, and it is impossible to enumerate all their naming patterns in practice. We use the observation that in general a device product name is the combination of letters and numbers (perhaps containing “-_”). Hence, we use the regex to cover the candidate product model entities. The 3rd row in Table 1 shows the regex of product names. If a word satisfies the regex, DER classifies it into a product label.

In this way, DER can heuristically identify all possible entities in webpages. However, this heuristic method has poor performance on device annotation extraction, due to high false positives especially in terms of device type and product name. This is because an irrelevant webpage may include at least one keyword of device type such as “switch” or a phrase that meets the requirement of regex for a product name. To address this problem, DER leverages the local dependency between entities for accurate device entity recognition.

Our observation is that true IoT entities always have strong dependence upon one another. Figure 3 presents the order of true IoT entities appearing in a webpage. Two kinds of local dependency usually occur: (1) the vendor entity first appears, followed by the device-type entity, and finally the product entity; (2) the vendor entity first appears, and the product entity appears second without any other object between the vendor entity, and the device-type entity follows. If the relationship is established and matches those two dependency rules, DER will select the tuple (device type, vendor, product) as the

Table 2: A few example rules learned for IoT devices.

Illustrating Rules
{ “Panasonic”, “KX-HGW500-1.51” } ⇒ {IPCam, Panasonic,KX-HGW500}
{ “TL-WR1043ND”, “Wireless”,“Gigabit”, “Ŏ0a9”,“Webserver” } ⇒ { Router, TP-Link, WR1043N }
{ “Welcome”,“ZyXEL”, “P-660HN-51”,“micro_httpd” } ⇒ { Router, Zyxel, P-600HN }
{ “Juniper”,“Web”, “Device”,“Manager”, “SRX210HE”,“Ŏ0a9” } ⇒ { Gateway, Juniper, SRX210 }
{ “Brother”,“HL-3170CDW”, “seriesHL-3170CDW”, “seriesPlease”,“debut/1.20” } ⇒ { Printer, Brother, HL-3170 }

device annotation. Otherwise, we exclude the webpage from the transaction set.

For every transaction, a device annotation can be classified into the following two categories:

- The tuple (device type, vendor, product) is complete. In this case, we use two entity appearing sequence orders to eliminate the multiple duplicate labels.
- The product entity cannot be recognized in the format (device type, vendor, null). Among multiple duplicate labels, DER selects the device annotations in the following order: the vendor entity first appears, and then the device-type entity follows.

3.4 Rule Generation

The rule miner uses the apriori algorithm to derive the relationship between search queries extracted from the response data $(q_i^1, q_i^2, ..., q_i^k)$ and device annotation extracted from a webpage (t_j, v_j, p_j) in the transaction set. The general form of the rule is: $\{q_i^1, q_i^2, ..., q_i^k\} \Rightarrow \{t_j, v_j, p_j\}$. When the response data holds the value q, we infer $\{t, v, p\}$ as its device annotation. ARE is able to discover an IoT device by simply and efficiently matching its response data with the rules in the library.

Parameters. There are two parameters for the apriori algorithm: support and confidence. The argument support is used to indicate the frequency of the variable appearing, and the argument confidence is the frequency of the rules under the condition in which the rule appears. In the transaction set $T = \{t_1, t_2, ..., t_n\}$, we can calculate those two parameters of the rule $A \Rightarrow B$ as follows:

$$sup(A) = |\sum_i^n A \in t_i| / |T|$$

$$conf(A \Rightarrow B) = sup(A \bigcup B) / sup(A)$$

The apriori algorithm first selects the frequent tuples in the dataset and discards the item whose support value is smaller than the support threshold. Then, the algorithm derives the rules whose confidence values are larger than the confidence threshold. The algorithm can generate all rules with support $\geq sup(A)$ and confidence $\geq conf(A \Rightarrow B)$. Note that the use of the parameter $sup(A)$ slightly differs from the one in the conventional apriori algorithm. In the transaction set, we use search query to eliminate the irrelevant items for the rule $A \Rightarrow B$. Thus, the transaction set includes the underlying mapping between part A and part B.

We conduct the experiment to validate the threshold of the apriori algorithm. We randomly choose an IP address chunk to generate the data set, which contains 2,499 transactions across 250 application response packets, across 5 device types (printer, access point, router, modem, and camera), 48 vendors and 341 products. To avoid the bias, we remove the tuples if they only appear one time in our data set. We observe that the settings of $sup(A) = 0.1\%$ and $conf(A \Rightarrow B) = 50\%$ work well in practice.

For data mining, the parameter selection of the apriori algorithm depends on the data set. When the device annotation becomes larger and more diverse, there are more infrequent rules in the transaction set. The parameter $sup(A)$ should further decrease to identify those infrequent pairs (A, B), which may be not-so-obvious. For the confidence of a rule $conf(A \Rightarrow B)$, it is desirable that rules always hold with few false positives. When the confidence increases, we can achieve high precision but missing some rules. The threshold of the parameter $conf(A \Rightarrow B)$ should further decrease if applications would like to collect more device annotations.

Conflict Rules. When multiple rules have the same tuple $\{q_1, q_2, ..., q_i\}$ but different device annotations $\{t, v, p\}$, they conflict with one another. When two different vendors have similar descriptions for their products, rules would have conflicts with each other. In this case, manual observation can distinguish those conflict rules for the application response packets. Similar to the Nmap tool, ARE does not remove those conflict rules. When confidences of the rules are approximately close to one another, we output each device annotation with a confidence. For instance, given the rules, $A \Rightarrow B$ and $A \Rightarrow C$, when the application matches the condition A, the output is 50% of the annotation B or C. Otherwise, we use the majority voting to output the highest confidence of the rules.

Example Rules. Table 2 shows a few example rules automatically learned by the rule miner based on the transaction set. The left part is the sequence of words extracted from the response data, acting as the search query. The right part is the device information, including device type, vendor, and product. Some rules seem apparent and are easily found, such as the first rule. Some rules are not so obvious, such as the fourth and fifth rules. Nmap developers usually provide users with those hardcoded and apparent rules in the service library. By contrast, our rule miner would generate rules without human effort. When we add new instances into the transaction set, the rule miner could automatically learn new rules over time.

3.5 Discussion

The rule miner leverages NLP techniques and association algorithms to learn rules, which can help applications to discover and annotate IoT devices in the cyberspace. Here we discuss ARE's limitations, including fake response data, the middle equipment, original equipment manufacturer (OEM), private binary protocols, and the extensibility.

Fake Responses. A transaction is the association between the response data from IoT devices and relevant webpages from the search engine. If the response data is faked (e.g., a honeypot can simulate IoT devices), the transaction set may contain erroneous information, leading to inaccurate rules. Furthermore, attackers may change the application data when they compromise a device. In those two cases, the transaction set for learning device rules could be corrupted. Fortunately, the amount of fake response data is small in comparison with the large number of regular IoT devices. Attackers may also have to cancel their malicious activities and do not change the application data, because such intrusive behaviors can be easily detected by administrators.

Middleboxes. Many IoT devices are behind the middleboxes such as firewalls/NAT in residential/enterprise/local networks and may not be accessible to the outside world. For instance, universal plug and play (UPnP) may attach multiple devices to a computer for connecting to a network. In such cases, rules cannot help to find those IoT devices behind middleboxes. However, if applications have the permission to search the local networks, the transactions can be re-collected inside the local networks and the rule miner can learn new rules. Our prototype system can be seamlessly deployed in large residential/enterprise/local networks that manage a fleet of IoT devices within their networks to collect transactions (see Section 4). That is, ARE could be also used for internal scans.

OEM. OEM is that one manufacturer produces parts of a device for another manufacturer, leading to the mixture of parts from the original and other vendors. Some manufacturers may resell subsystems to assemble devices for different manufacturers, which causes ambiguity. In this case, neither fingerprinting nor banner grabbing techniques can resolve the OEM problem. ARE offers a best-effort service to generate rules of IoT devices.

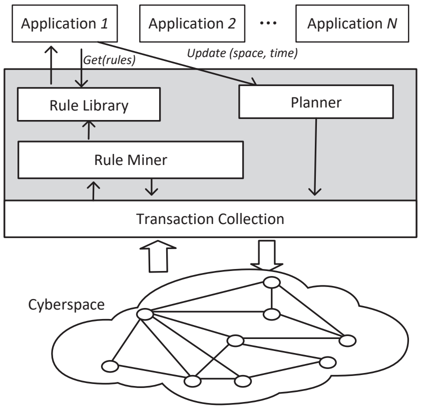


Figure 4: ARE architecture for learning device rules.

Private Binary Protocol. ARE leverages the fact that many application protocols include device information. If application protocols are private and binary, their packets cannot be tokenized into the text for generating query search keywords. However, some vendors use proprietary binary protocols for business considerations. Nowadays, there is no tool able to analyze proprietary protocols for IoT devices. ARE cannot provide rules for those IoT devices either.

Extensibility. ARE is used to generate rules for the application response packets, not limited to IoT devices. For instance, online services may provide the application responses for their requests. If the response packets include the information of services, ARE can generate rules for those services.

4 ARE: Design and Implementation

In this section, we present the design and implementation of ARE for automatically discovering IoT devices. ARE consists of four components: transaction collection, rule miner, rule library, and planner. The transaction collection module has the capability of gathering transactions in a variety of networks. The rule miner module is the core of ARE for learning IoT device rules. The rule library and planner modules provide interactive interfaces to applications for discovering and annotating IoT devices in the cyberspace. Below, we first illustrate how ARE works and then detail the system design and implementation of ARE.

4.1 ARE Architecture

Figure 4 shows a high-level architecture of ARE. It works as the middleware, and the function of each component is briefly described as follows. **(1) Transaction Collection.** According to the transaction definition 1, the collection module gathers data in a network for the rule miner. This module works in two steps. The first step is to collect response data in the network and filter out the response data from non-IoT devices. The second step uses the web crawler to obtain the description webpages of IoT devices, and then removes redundant content from the webpages. **(2) Rule Miner.** ARE leverages the rule miner to automate the rule generation process from the transaction set without human effort. Furthermore, this module can dynamically learn rules, e.g., when manufacturers release new IoT device products. **(3) Rule Library.** The rule library is a standard file, which stores each rule in the format $\{A \Rightarrow B\}$ with a timestamp. A denotes keywords in the response data, and B is the device annotation (t, v, p). Applications interact with ARE through the API *Get(rules)*, and the rule library returns the latest rules to users. **(4) Planner.** The planner module updates the rule library in ARE for applications. The API *Update(network, time)* notifies the planner module to generate new rules in the current network and gather data from this space, and the outdated rules would be removed.

4.2 Transaction Collection

We present the overview of the transaction collection in Figure 5. The response data collection (RDC) is used to gather the application-layer data in a network and then filter out the response data from non-IoT devices. The web crawler extracts the search queries from the response data and inputs them to the search engine. The search engine returns the result lists of webpages, and the web crawler crawls the HTML files in these webpages.

Response Data Collection. We can directly use public data sets about application service responses (such as HTTP, FTP, TELNET, and RTSP) from Censys [25]. After getting the raw response data, we should remove some erroneous responses. For HTTP response data, we remove some error responses in terms of IETF status codes, such as responses with statute codes (5XX) and redirection codes (3XX). For FTP response data, we remove some response packets that include some keywords like ("filezilla, serve-u"), because they are common software running on a computer. For Telnet response data, we would remove a character sequence with the particular code (IAC 0xFF), which is used for negotiating the communication between different operating systems.

After the pre-screening above, the response data containing short and simple packets (such as TELNET, FTP and RTSP response data) has been completely cleaned up. However, the HTTP response data may still contain many non-IoT packets. For example, the packets from some commercial websites selling camera devices include device-relevant textual content. So, we need to further filter out the HTTP response data from non-IoT devices. We observe that consumer-oriented IoT devices have limited computing and memory capacities, usually deploying at homes, offices, facilities and elsewhere. Thus, we find that IoT devices have the following features in their HTTP response data, which can be leveraged for effective IoT device identification.

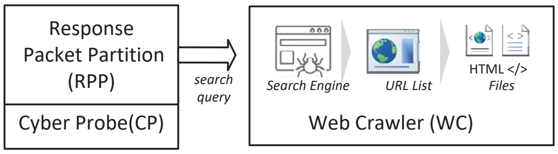


Figure 5: The overview of the transaction collection.

- Generally, IoT devices use a lightweight web server (e.g., boa and lighthttp) rather than a heavyweight web server (e.g., Apache, IIS, and Ngnix).

- The webpage of IoT devices is simple, such as a login or configuration page. Compared with a regular webpage, the number of terms (scripts, words, pictures) in the webpage of IoT devices is very small.

- The webpage of IoT devices usually does not have the external links to other websites, and if it does, the number of links is also small.

Using these observations, we can filter the non-IoT devices and the rest of the response data is added into the candidate IoT devices.

Web Crawler. The web crawler first extracts the sequence of search queries from the response data. There is much redundant textual information unrelated to manufacturers. We first remove hyperlinks, field names, time, script block, and symbols (such as $< p >$ and $< \backslash p >$). Then, we remove dictionary words in the response data. The reason is that the names of vendors and product models are usually non-dictionary words. Note that if the dictionary word is also in our brand and device type list, we will keep it. Dictionary words have little relation to device manufacturers. After that, we use the term frequency-inverse document frequency (TF-IDF) to measure the concentration degree of a word in the response data. If the TF-IDF score is higher, we think the term is more relevant to the description webpage.

A practical problem here is the restrictions on the amount of API accessing in today's search engines. For instance, Google provides 100 queries per day for free users and has a limitation of 10,000 queries per day. To address this issue, the web crawler simulates the browser behavior and sends individual browser instances to the search engine. Every time it is accessed, the web crawler module uses a different user-agents and sleeps for a random time after multiple requests. If one access instance fails, we will perform the retransmission operation at the end of the search query queue. The search engine will return a URL list for every search query. Based on these lists, we can reduce the scale of web crawling. Each item in the URL list returned by browser instances is a complete HTML page. There is much redundant content in these webpages, such as advertisements, pictures, audios, videos, and dynamical scripts. For each webpage, the web crawler removes the irrelevant information and only keeps the textual content, including title, URL, and snippet. Fortunately, the indexing algorithms in today's search engines have already found the most relevant websites for the search query. In our experiment, the top 10 webpages work well in practice for locating relevant information on IoT devices.

In the implementation, we write a custom Python script to pipeline from the response data into webpage crawling. The web crawler uses the enchant library [17] to remove dictionary words and the NLP toolkit [7] to calculate the TF-IDF values. The web crawler uses the python urllib2 library to simulate and automatically visit the search engines. The Beautiful Soup [4] library is used to extract the content from the webpage.

4.3 Implementation of Rule Miner

The rule miner automatically learns rules of IoT devices from the transaction set. We use Python scripts to implement DER, which is the core of rule miner. The NLP toolkit [7] is used to process the text content, including word splitting, stemming and removing stop words. We also use apriori algorithm [3] in Python Package to generate rules for IoT devices.

In practice, the rule miner has to handle the scenarios where the response data does not include sufficient device information to initiate the subsequent web crawling process for rule generation. For example, from the FTP response packet "220 Printer FTP 4.8.7 ready at Jan 19 19:38:22," we can only extract one useful keyword "Printer" as a search query. With only one search query being extracted, no local dependency can be exploited to achieve accurate and fine-grained device annotation. Thus, there is no need to initiate the web crawling process and no rule is created. However, we can still use the DER module to extract one label in the response data, achieving a coarse-grained device annotation. There are two categories for such one-entity annotations, including *(device type, null, null)* and *(null, vendor, null)*. Note that none of the existing tools (Nmap and Ztag) can address this problem caused by the lack of information in the response data.

4.4 Applications on ARE

We explicate how applications work with ARE. As shown in Figure 4, an application interacts with ARE by calling APIs (*Get()* and *Update()*). If the rule library meets its requirements, the application directly uses rules for discovering IoT devices. Otherwise, the RDC module would gather the application layer data in the network based on the parameters of *Update()*. The rule miner module would generate rules according to the recently collected data. In the implementation of the rule library and planner, ARE provides the REST APIs to applications, including GET and POST operations. RESTful GET is used to retrieve the representation of rules from ARE, and POST is used to update the rule library. The rule library stores rules in the text files.

In the design of ARE, we aim to provide rules for applications for discovering IoT devices while minimizing the requirements of manual effort and training data. To demonstrate the effectiveness of ARE, we develop three ARE-based applications.

Internet-wide Measurement for IoT Devices. Like prior Internet-wide measurements [21,26,31,33,35], we build the measurement application using the rules from ARE to collect, analyze, and characterize the deployment of these IoT devices across the real world.

Detecting Compromised IoT Devices. Like [21,29], we build several honeypots to capture malicious behaviors in the cyberspace. After capturing their malicious traffic, we track their IP addresses and use the ARE rules to identify whether it is an IoT device. If so, we extract its device type, vendor, and product information, and then we analyze its malicious behaviors.

Detecting Vulnerable IoT Devices. Like [23, 24], we build a vulnerability analysis application through the dataset from the National Vulnerability Database [12]. If a CVE item occurs in IoT devices, we extract the rules of those devices from ARE and use the rules to discover vulnerable online devices with a high probability.

5 Evaluation

In this section, we first elaborate on the system setting for ARE experiments. Then, we show the experimental results for ARE evaluation, which include that (1) the number of rules generated by ARE is nearly 20 times larger than those of the existing tools, (2) our rules can achieve very high precision and coverage, and (3) the time cost introduced by ARE is low.

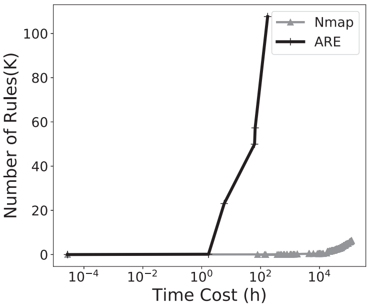


Fig. 6: Time cost comparison for generating the rules.

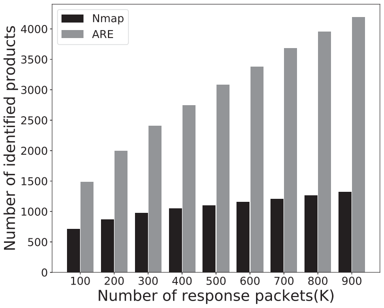


Fig. 7: Comparison with Nmap.

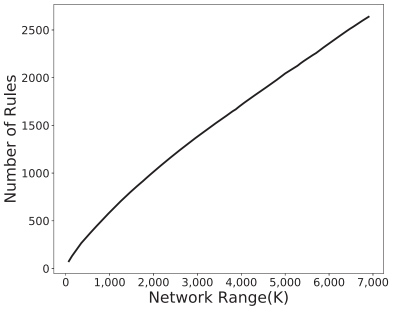


Fig. 8: Dynamic rule learning for ARE.

5.1 Setting

In the transaction collection, the RDC module only searches public IPv4 addresses for collecting response data of four application protocols (HTTP, FTP, RTSP, and TELNET). Most IoT devices usually have a built-in Web, FTP, Real-time Media, or TELNET user interfaces. ARE can be expanded supporting more application protocols without much modification. So far, ARE cannot learn device rules if a device only appears behind the home/private networks. However, ARE can be deployed into local networks behind a firewall for internal IoT device discovery without any modification.

We use two datasets for evaluating ARE performance. In the first dataset, we randomly choose 350 IoT devices from the Internet. The selection process uses the Mersenne Twister algorithm in Python's random package. We manually label those IoT devices, and the ground truth labels include 4 different device types (NVR, NVS, router, and ipcamera) 64 different vendors, and 314 different products. The labeling process is done by analyzing their application layer responses, searching some keywords through the search engine, and finally receiving the labels. Note that this process requires rich experience on IoT recognition. The second dataset consists of 6.9 million IoT devices that our application collects on the Internet. Because the number of devices is vast, we apply the same random algorithm to sample 50 IoT devices iteratively for 20 times. In total, the second dataset contains 1,000 devices across 10 device types and 77 vendors.

5.2 Performance

Number of Rules. We first compare the labeling performance between ARE and Nmap. Nmap [8] is an open-source tool for network discovery and security scanning. The number of rules in the Nmap library has been increasing for two decades, from the initial version V3.40 to the latest version V7.60. The latest version of Nmap has 6,504 rules [9] for four application protocols (HTTP, FTP, RTSP, and TELNET). Figure 6 compares the time cost in rule generation between ARE and Nmap, where the Y-axis is the number of rules and X-axis is the time cost in the logarithmic scale ($\log_{10}$). ARE is able to generate 115,979 rules in one week. While the number of rules generated by ARE is almost 20 times larger than that of Nmap, the ARE's time cost is negligible compared to Nmap's, The reason is that the rule generation of Nmap requires the professional background/experience to write a rule manually, which is a long-term process. By contrast, ARE automates the rule generation process.

Table 3: Precision and coverage of rules on the dataset.

	Precision	Coverage
The first dataset	95.7%	94.9%
The second dataset	97.5%	—

Table 4: Rules generated by ARE.

Category	Num	Percentage %
(device type, vendor, product)	107,627	92.8
(device type, vendor, null)	8,352	7.2

Precision of Rules. We further evaluate the performance of ARE rules by using precision. The precision is equal to $|TP|/|FP+TP|$, where TP is the number of true positives and FP is the number of false positives. Table 3 lists the precision of ARE rules. In the first dataset, the precision of rules is 95.7%. In the second dataset, the ARE rules can achieve a 97.5% precision.

Coverage of Rules. Table 3 also lists the coverage of ARE rules. The coverage is $|TP|/|FP+FN|$, where FN is the number of false negatives. For the first dataset, the

Table 5: Average time cost of one ARE rule generation.

Stage	Latency (second)
Application layer data	0.5022
Response packet partition	0.0017
Web crawler	0.4236
Apriori algorithm	0.1166

coverage of rules is 94.9%. For the second dataset, the coverage is unknown, because we cannot determine the number of false negatives in device annotation. Further, Table 4 lists the detailed results of the rules generated by ARE. There are 92.8% of rules that can completely label IoT devices in the form of *(device type, vendor, product)*. Only 7.2% of rules just label device type and vendor. As a comparison, Nmap only has about 30% of rules with a fine-grained annotation.

We use the hash algorithm to calculate MD5 checksums of the application-layer packets from Censys [25], and then remove the duplicated packets. Based on these response packets, we use both ARE and Nmap rules for device identification. Figure 7 shows the performance of device identification along with the number of the application-layer packets. Given the same number of response packets, ARE achieves a larger coverage than Nmap. When the number of application-layer packets increases, ARE can find even more devices than Nmap. Note that the distribution of IoT devices on the Internet is a typical long tail rather than uniform distribution on the Internet. This implies that some rules can find much more devices than other rules. For popular IoT products, ARE rules can classify them with robust labels. For little-known IoT products, ARE rules can still classify them because we generate rules based on the embedded information.

Dynamic Rule Learning. We also conduct experiments to evaluate the learning capacity of ARE. Figure 8 shows that the number of rules is increasing as ARE learns with the increase of network space. The rule miner can learn new rules when ARE is deployed into different networks (e.g., residential/enterprise networks). Thus, ARE has the capability for dynamic rule learning.

Overhead of ARE. Finally, we conduct experiments to measure the time cost of ARE. Our ARE prototype is running on a commercial desktop computer (Windows 10, 4vCPU, 16GB of memory, 64-bit OS), indicating that CPU and memory costs of ARE can be easily met. The ARE process is running in a single thread. Table 5 lists the average time cost of individual components of ARE for one rule generation. The acquisition of application-layer data takes 0.5022 seconds, and the web crawling takes 0.4236 seconds. Those components require the message transmissions, and the time cost is dependent upon the network conditions. As comparison, the packet partition and the apriori algorithm induce little time cost. Overall, the time cost of ARE for automatic rule generation is low in practice, and we could further reduce the time cost by running ARE in multiple threads.

Table 6: Automatic Internet-wide identification.

Device Type	Number (%)	Vendor	Number (%)
Router	1,249,765 (18.3)	Mikrotik	641,982 (9.3)
NVR	785,810 (11.3)	Zte	352,498 (5.1)
DVR	644,813 (9.3)	Tp-link	325,751 (4.7)
Modem	466,286 (6.7)	Sonicwall	279,146 (4.0)
Camera	379,755 (5.5)	D-link	215,122 (3.1)
Switch	180,121 (2.6)	Dahua	153,627 (2.2)
Gateway	127,532 (1.8)	Hp	106,327 (1.5)
Diskstation	35,976 (0.5)	Asus	101,061 (1.5)

6 ARE-based Applications

In this section, we present the experimental results obtained from three ARE-based applications, which further demonstrate the effectiveness of ARE.

6.1 Internet-wide Device Measurement

IoT devices are usually deployed across many different places, such as homes, infrastructure facilities, and transportation systems. Traditionally IoT devices are behind a broadband router with NAT/PAT/Firewall, but many of them are now directly exposed on the Internet. Thus, it is necessary to conduct an Internet-wide measurement of IoT devices to have a deep understanding of their deployment and usage on the Internet. Previous Internet-wide measurements have focused on network topology [22], websites [27], and end hosts [31, 33]), but few has been done on IoT devices. ARE greatly facilitates such an Internet-wide measurement to infer, characterize, and analyze online IoT devices.

In the IDM application, we use three application-layer datasets from Censys [25], including HTTP, FTP, and Telnet. Additionally, we deploy the collection module on the Amazon EC2 [20] with 2 vCPU, 8GB of memory, and 450Mbps of bandwidth, which collects the RTSP application-layer response data. Overall, we found 6.9 million IoT devices, including 3.9 million from HTTP, 1.5 million from FTP, 1 million from Telnet, and 0.5 million from RTSP. Using ARE rules, the IDM application can give an annotation to every IoT device. Furthermore, we use MaxMind's GEOIP [34] database to find the location of an IoT device, which has a relationship between IP address and the city-level location label.

Table 7: Geographic distribution.

District	Number	Percentage (%)
United States	1,403,786	20.26
China	466,007	6.73
Brazil	442,781	6.39
India	297,446	4.29
Mexico	289,976	4.18
Taiwan	273,024	3.94
Republic of Korea	255,924	3.69
Russia	239,236	3.45
Egypt	204,237	2.95
Vietnam	199,415	2.88

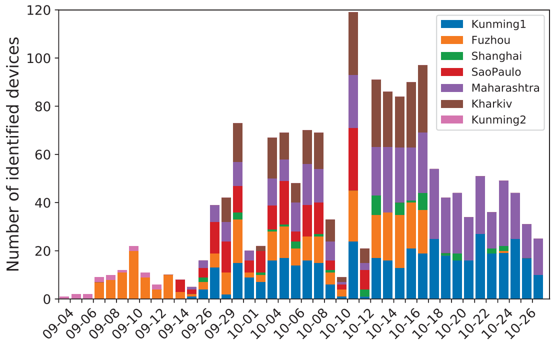


Figure 9: Compromised IoT device distribution.

Discovery. Based on the analysis of millions of IoT devices, we have three discoveries. (1) Although a large portion of IoT devices may be behind firewalls in home/enterprise networks, the number of visible and reachable IoT devices on the Internet is still very large. Even if only 0.01% of IoT devices are accessible to the external networks, considering the sheer size of active IoT devices (billions), the absolute number of exposed devices will reach the level of millions. (2) The long-tail distribution is common for IoT devices, including device types, vendors, and locations. Table 6 lists the distribution of the top 10 device types and vendors. We observe that nearly 31% of IoT devices are from the top 10 device vendors. The location distribution of IoT devices is a typical long-tail, as shown in Table 7. The top 10 countries (127 countries in total) occupy nearly half of the IoT devices. (3) Many devices should not be visible or reachable from the external networks. It is common for routers, gateways, switches, and modems to be visible and reachable on the Internet. However, the monitoring devices, such as camera and DVR, should not be directly exposed to the external networks. Unfortunately, there are more than two million of those types of IoT devices accessible on the Internet, as shown in Table 6.

6.2 Compromised Device Detection

Our detection of compromised IoT devices is based on the capture of malicious IoT traffic behaviors. A recent work [21] leverages honeypot traffic to detect the Miria botnet infections based on unique packet content signatures. After the collection of suspicious IPs, the Nmap identification rules [9] are used to obtain the device type. Similarly, we develop the CDD application to discover compromised devices.

In particular, we deploy seven honeypots as vantage points for monitoring traffic on the Internet, across four countries (Brazil, China, India, and Ukraine) and six cities, including Fuzhou, Kharkiv, Kunming (2 honeypots), Maharashtra, Sao Paulo, and Shanghai. The monitoring duration is nearly two months. We use the open-source Cowrie SSH/Telnet Honeypot [6] in the CDD. Every honeypot is configured with weak SSH/Telnet credentials and instructed to forward traffic functions to the CDD application. If a honeypot captures one IP address that attempts to connect to our honeypot with SSH or Telnet, we will leave this IP into the Kafka queue [2]. The CDD runs on Amazon EC2, and sends a request to each IP address in the Kafka queue for receiving a response data. Then ARE rules are used to identify IoT devices from the response data. The rationale behind such a design lies in the fact that a normal IoT device should never access honeypots. If an IoT device accesses our honeypot, there are only two reasons: it is misconfigured or compromised.

Discovery. Figure 9 shows the number of compromised devices captured by the CDD application. We can capture about 50 different compromised IoT devices every day. In total, we detect nearly 2,000 compromised IoT devices among 12,928 IP addresses attempting to connect to our honeypots. Many compromised IoT devices attempt to brute force the SSH/TELNET credentials of our honeypots. After mounting a successful brute-force attack, the devices will execute some commands on one of our honeypots, indicating that these IoT devices are compromised and they try to compromise more devices. Table 8 lists the distribution of the top 5 device types and vendors for compromised devices. We can see that among different device types, DVR has by far the largest number of compromised devices, followed by network attached storage device (NAS) and router. In addition, we also observe that a few smart TV boxes are compromised and exhibit malicious behaviors.

6.3 Vulnerable Device Analysis

The disclosure of underlying vulnerable devices is also valuable to the security community. From the defensive

Table 8: Device type and vendor for compromised devices.

Device Type	Num	(%)	Vendor	Num	(%)
DVR	1168	67.7	Hikvision	231	13.4
NAS	189	10.9	Dahua	216	12.5
Router	173	10.0	Qnap	189	10.9
Webcam	92	5.3	Mikrotik	81	4.7
Media device	83	4.8	TVT	79	4.5

Table 9: Top 10 CWE by the number of CVEs.

CWE ID	Weakness Summary	Number of IoT devices
200	Information Disclosure	573,656
22	Path Traversal	363,894
352	CSRF	348,031
264	Permission, Privileges, Access Control	345,175
255	Credentials Management	342,215
79	Cross-site Scripting	331,649
119	Buffer Overflow	149,984
399	Resource Management Errors	93,292
284	Improper Access Control	69,229
77	Command Injection	64727

perspective, it can help us find out which online devices are still vulnerable and perform security patches for critical infrastructure immediately. Normally one vulnerability of IoT devices is associated with a particular model of IoT devices. For instance, a buffer overflow vulnerability CVE-2015-4409 has occurred in the Hikvision DS-76xxNI-E1/2 series and Hikvision DS-77xxxNI-E4 series devices.

We develop the VDA application to reveal underlying vulnerable devices. VDA first crawls the vulnerability information from the NVD website [12] [5]. For every vulnerability item, VDA obtains their vendor names and product names. Then VDA uses the regex to match rules with the vulnerability information. We extract the category information of vulnerabilities and group similar weakness descriptions. One vulnerability usually occurs on multiple platforms and device models. Table 9 lists the Common Weakness Enumeration (CWE) of online IoT devices, in which the left column is the CWE ID, the middle column is the weakness description, and the right column is the number of IoT devices with this type of vulnerability. The VDA application aims to reveal underlying vulnerable devices accessible on the Internet.

Discovery. From Table 9, we can see that there is still a large number of underlying vulnerable devices in the cyberspace. The majority of the top 10 vulnerabilities in the CWE list are related to improper implementation (Path Traversal, Credentials Management, and Improper Access Control), which could be easily avoided if a developer pays more attention to security. On the CVE website, the security patches have been distributed for those IoT devices. However, updating security patches of IoT devices is a non-trivial task for many users. They must download the firmware from the official support website or via administrative tools, and then install the firmware into the ROM to reprogram integrated chip circuits of the devices.

7 Related Work

IoT device recognition has gained much interest recently, mostly due to the increasing number of IoT devices that are connected to the Internet. The research community has also proposed many recognition techniques, particularly in two methodologies: fingerprinting and banner-grabbing.

Fingerprinting. We have witnessed a 20-year development for fingerprinting technologies, which map the input to a narrower output for object identification [8, 10, 11, 15, 18, 19, 32, 35, 36, 39]. Dependent upon the method of data collection, fingerprinting can be divided into active and passive. Active fingerprinting is to send probing packets to remote hosts for extracting features and inferring the classification model. One classic usage is OS fingerprinting, which identifies the OS of a remote host based on the different implementations of a TCP/IP network stack. Nmap [8] is the most popular tool for OS fingerprinting, which sends 16 crafted packets for extracting features. Xprobe [15] uses ICMP packets to extract OS features. The retransmission time between vantage points and hosts can be exploited as another feature for OS fingerprinting. Snacktime [11], Hershel [36], and Faulds [35] use this feature to fingerprint OSes on the large scale. Passive fingerprinting is to collect the traffic/behavior of an object without sending probing packets. P0f [10] is the passive fingerprinting tool that extracts ongoing TCP packets to infer different OS versions. Kohno et al. [32] proposed monitoring TCP traffic for calculating the clock skews as features.

In general, a fingerprinting tool consists of three major components: feature selection, training data collection, and learning algorithms. Prior works are focused on how to select distinctive features for fingerprinting OS versions. However, due to the lack of training data, we cannot apply fingerprinting techniques for identifying IoT devices. Furthermore, the number of different IoT device models is vast, and it is impossible to manually collect the training data. Thus, we propose ARE that is able to learn the rules for automatic IoT device identification without any training data or human effort.

Banner-grabbing. The banner-grabbing technique is to profile the text information of applications and software services. Nowadays various tools have been used to gather web applications for administrative and security auditing purposes. WhatWeb [14] is a website auditing tool that uses 1,000 plugins (similar to regex) to recognize the platform version of a website. Wapplyzer [13] is an open-source tool for identifying web applications, which extracts response headers of websites and uses regex patterns for matching. Nmap [8] also provides a service library to identify application and web services for end users. For annotating IoT devices, people currently tend to use banner-grabbing in practice. In the analysis of the Mirai botnet [21], the regex in banner-grabbing is used to annotate the device type, vendors, and products. Xuan et.al [30] proposed to utilized the banner of industrial control protocols to find a critical infrastructure equipment. Shodan [37] and Censys [25] use a set of rules in the banner-grabbing technique to identify online devices.

To use those banner-grabbing tools, developers usually need the necessary background knowledge to write the regex/extensions for grabbing application information. This has to be done in a manual fashion, which incurs high time cost, impeding a large-scale annotation. By contrast, ARE overcomes these obstacles by automatically generating rules.

8 Conclusions

As the increasing number of IoT devices are connected to the Internet, discovering and annotating those devices is essential for administrative and security purposes. In this paper, we propose an Acquisitional Rule-based Engine (ARE) for discovering and annotating IoT devices. ARE automates the rule generation process without human effort or training data. We implement a prototype of ARE and conduct experiments to evaluate its performance. Our results show that ARE can achieve a precision of 97%. Furthermore, we apply ARE to three application cases: (1) inferring and characterizing millions of IoT devices in the whole IPv4 space, (2) discovering thousands of compromised IoT devices with malicious behaviors, and (3) revealing hundreds of thousands of IoT devices that are still vulnerable to malicious attacks.

Acknowledgments

We are grateful to our shepherd Gang Wang and anonymous reviewers for their insightful feedback. This work was supported in part by the National Key R&D Program of China (Grant No. 2016YFB0801303-1), Key Program of National Natural Science Foundation of China (Grant No. U1766215) and National Natural Science Foundation of China (Grant No. 61602029).

References

[1] 20.8 billion IoT devices by 2020. `https://www.gartner.com/newsroom/id/3598917`.

[2] Apache Kafka. `https://kafka.apache.org`.

[3] Apyori, a simple implementation of Apriori algorithm with Python. `https://pypi.python.org/pypi/apyori/1.1.1`.

[4] Beautiful Soup, A Python library designed for quick turnaround projects. `https://www.crummy.com/software/BeautifulSoup/`.

[5] Common Vulnerabilities and Exposures. `http://cve.mitre.org/`.

[6] Cowrie SSH/Telnet Honeypot. `https://github.com/micheloosterhof/cowrie`.

[7] Natural language toolkit. `http://www.nltk.org/`.

[8] Nmap, network security scanner tool. `https://nmap.org/`.

[9] Nmap service detection probe list. `https://github.com/nmap/nmap/blob/master/nmap-service-probes`.

[10] P0f: The passive OS and application tool for penetration testing, routine network monitoring, and forensics, 2004. `http://freshmeat.net/projects/p0f/`.

[11] Snacktime: A perl solution for remote os fingerprinting.

[12] U.s. national institute of standards and technology. national vulnerability database. `https://nvd.nist.gov/home.cfm`.

[13] Wappalyzer identify technology on websites.

[14] Whatweb identifies websites. `https://github.com/urbanadventurer/whatweb/wiki`.

[15] Xprobe2 - a remote active operating system fingerprinting tool. `https://linux.die.net/man/1/xprobe2`.

[16] ZTag, an utility for annotating raw scan data with additional metadata. `http://github.com/zmap/ztag`.

[17] Abiword. Enchant. `http://www.abisource.com/projects/enchant/`, 2010.

[18] Acar, G., Eubank, C., Englehardt, S., Juárez, M., Narayanan, A., and Díaz, C. The web never forgets: Persistent tracking mechanisms in the wild. In Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security, Scottsdale, AZ, USA, November 3-7, 2014, pp. 674–689.

[19] Acar, G., Juárez, M., Nikiforakis, N., Díaz, C., Gürses, S. F., Piessens, F., and Preneel, B. Fpdetective: dusting the web for fingerprinters. In 2013 ACM SIGSAC Conference on Computer and Communications Security, CCS'13, Berlin, Germany, November 4-8, 2013, pp. 1129–1140.

[20] Amazon. Amazon elastic compute cloud (amazon ec2). `https://aws.amazon.com/ec2/`, 2013.

[21] Antonakakis, M., April, T., Bailey, M., Bernhard, M., Bursztein, E., Cochran, J., Durumeric, Z., Halderman, J. A., Invernizzi, L., Kallitsis, M., Kumar, D., Lever, C., Ma, Z., Mason, J., Menscher, D., Seaman, C., Sullivan, N., Thomas, K., and Zhou, Y. Understanding the mirai botnet. In 26th USENIX Security Symposium, USENIX Security 2017, Vancouver, BC, Canada, August 16-18, 2017., pp. 1093–1110.

[22] BEVERLY, R. Yarrp'ing the internet: Randomized high-speed active topology discovery. In Proceedings of the 2016 ACM on Internet Measurement Conference, IMC 2016, Santa Monica, CA, USA, November 14-16, 2016, pp. 413–420.

[23] CUI, A., COSTELLO, M., AND STOLFO, S. J. When firmware modifications attack: A case study of embedded exploitation. In 20th Annual Network and Distributed System Security Symposium, NDSS 2013, San Diego, California, USA, February 24-27, 2013.

[24] CUI, A., AND STOLFO, S. J. A quantitative analysis of the insecurity of embedded network devices: results of a wide-area scan. In Twenty-Sixth Annual Computer Security Applications Conference, ACSAC 2010, Austin, Texas, USA, 6-10 December 2010, pp. 97–106.

[25] DURUMERIC, Z., ADRIAN, D., MIRIAN, A., BAILEY, M., AND HALDERMAN, J. A. A search engine backed by internet-wide scanning. In Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, Denver, CO, USA, October 12-6, 2015, pp. 542–553.

[26] DURUMERIC, Z., KASTEN, J., ADRIAN, D., HALDERMAN, J. A., BAILEY, M., LI, F., WEAVER, N., AMANN, J., BEEKMAN, J., PAYER, M., AND PAXSON, V. The matter of heartbleed. In Proceedings of the 2014 Internet Measurement Conference, IMC 2014, Vancouver, BC, Canada, November 5-7, 2014, pp. 475–488.

[27] ENGLEHARDT, S., AND NARAYANAN, A. Online tracking: A 1-million-site measurement and analysis. In Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, October 24-28, 2016, pp. 1388–1401.

[28] FACHKHA, C., BOU-HARB, E., KELIRIS, A., MEMON, N., AND AHAMAD, M. Internet-scale probing of cps: Inference, characterization and orchestration analysis. In Proceedings of Network and Distributed System Security Symposium (2017), vol. 17.

[29] FARINHOLT, B., REZAEIRAD, M., PEARCE, P., DHARMDASANI, H., YIN, H., BLOND, S. L., MCCOY, D., AND LEVCHENKO, K. To catch a ratter: Monitoring the behavior of amateur darkcomet RAT operators in the wild. In 2017 IEEE Symposium on Security and Privacy, SP 2017, San Jose, CA, USA, May 22-26, 2017, pp. 770–787.

[30] FENG, X., LI, Q., WANG, H., AND SUN, L. Characterizing industrial control system devices on the internet. In 24th IEEE International Conference on Network Protocols, ICNP 2016, Singapore, November 8-11, 2016.

[31] HEIDEMANN, J. S., PRYADKIN, Y., GOVINDAN, R., PAPADOPOULOS, C., BARTLETT, G., AND BANNISTER, J. A. Census and survey of the visible internet. In Proceedings of the 8th ACM SIGCOMM Internet Measurement Conference, IMC 2008, Vouliagmeni, Greece, October 20-22, 2008, pp. 169–182.

[32] KOHNO, T., BROIDO, A., AND CLAFFY, K. C. Remote physical device fingerprinting. IEEE Transactions on Dependable and Secure Computing 2, 2 (April 2005), 93–108.

[33] LEONARD, D., AND LOGUINOV, D. Demystifying service discovery: implementing an internet-wide scanner. In Proceedings of the 10th ACM SIGCOMM Internet Measurement Conference, IMC 2010, Melbourne, Australia - November 1-3, 2010, pp. 109–122.

[34] MAXMIND. Maxmind geoip2. `https://www.maxmind.com/en/geoip2-services-and-databases`, 2016.

[35] SHAMSI, Z., CLINE, D. B. H., AND LOGUINOV, D. Faulds: A non-parametric iterative classifier for internet-wide OS fingerprinting. In Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS 2017, Dallas, TX, USA, October 30 - November 03, 2017, pp. 971–982.

[36] SHAMSI, Z., NANDWANI, A., LEONARD, D., AND LOGUINOV, D. Hershel: single-packet os fingerprinting. In ACM SIGMETRICS / International Conference on Measurement and Modeling of Computer Systems, SIGMETRICS '14, Austin, TX, USA - June 16 - 20, 2014, pp. 195–206.

[37] SHODAN. The search engine for Internet-connected devices. `https://www.shodan.io/`.

[38] SHOSHITAISHVILI, Y., WANG, R., HAUSER, C., KRUEGEL, C., AND VIGNA, G. Firmalice - automatic detection of authentication bypass vulnerabilities in binary firmware. In 22nd Annual Network and Distributed System Security Symposium, NDSS 2015, San Diego, California, USA, February 8-11, 2015.

[39] STAROV, O., AND NIKIFORAKIS, N. XHOUND: quantifying the fingerprintability of browser extensions. In 2017 IEEE Symposium on Security and Privacy, SP 2017, San Jose, CA, USA, May 22-26, 2017, pp. 941–956.

[40] VENKATARAMAN, S., CABALLERO, J., POOSANKAM, P., KANG, M. G., AND SONG, D. X. Fig: Automatic fingerprint generation. In Proceedings of the Network and Distributed System Security Symposium, NDSS 2007, San Diego, California, USA, 28th February - 2nd March 2007.