ARGUS: Assessing Unpatched Vulnerable Devices on the Internet via Efficient Firmware Recognition

Wei Xie
National University of Defense Technology
Changsha, China
xiewei@nudt.edu.cn

Chao Zhang
Tsinghua University
Beijing, China
chaoz@tsinghua.edu.cn

Penfei Wang
National University of Defense Technology
Changsha, China
pfwang@nudt.edu.cn

Zhenhua Wang
National University of Defense Technology
Changsha, China
wzh15@nudt.edu.cn

Qiang Yang
National University of Defense Technology
Changsha, China
q.yang@nudt.edu.cn

ABSTRACT

Assessing unpatched devices affected by a specified vulnerability is a vital but unsolved issue. Using a proof-of-concept tool on the Internet is illegal, while identifying vulnerable device models and firmware versions via fingerprints is a safer method. However, device search engines such as Shodan do not claim to accurately identify device models or versions, and existing works on firmware online recognition neglect the efficiency challenge of scanning redundant fingerprints. Consequently, this fingerprint-checking method has few real-world verifications on the Internet.

We propose ARGUS, a simple but practical framework to identify device models and firmware versions. At its core is a heuristic fingerprint crush saga (FCS) scheme inspired by the phone game "Candy Crush Saga". It can improve efficiency by an average of 156 times compared to scanning fingerprints of all web files by default. This efficiency improvement enables us to widely assess the proportion of unpatched devices affected by 176 CVE vulnerabilities, which is 64.3% on average on the Internet. This result quantitatively proves that the majority of users do not periodically update device firmware.

CCS CONCEPTS

• **Security and privacy → Vulnerability management**; **Embedded systems security**.

KEYWORDS

Vulnerability, Assessment, Device, Web, Fingerprint

ASIA CCS '21, June 7–11, 2021, Hong Kong, China

ACM ISBN 978-1-4503-8287-8/21/06…$15.00
https://doi.org/10.1145/3433210.3453685

ACM Reference Format:
Wei Xie, Chao Zhang, Penfei Wang, Zhenhua Wang, and Qiang Yang. 2021. ARGUS: Assessing Unpatched Vulnerable Devices on the Internet via Efficient Firmware Recognition. In *Proceedings of the 2021 ACM Asia Conference on Computer and Communications Security (ASIA CCS '21), June 7–11, 2021, Hong Kong, China.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3433210.3453685

1 INTRODUCTION

Unpatched devices affected by disclosed vulnerabilities have caused serious (distributed denial of service (DDoS) attacks, such as attacks using Internet of Things (IoT) botnets such as Mirai [1] and VPNFILTER [17]. These security events suggest that the harmful effects of known vulnerabilities can be longlasting, especially in IoT scenarios. On the one hand, exploiting known vulnerabilities is usually more straightforward than exploiting zero-day vulnerabilities due to more public technical details. On the other hand, devices such as web cameras and home routers are often unattended and lack automatic update approaches. This fact can keep a disclosed vulnerability exploitable for years after its disclosure.

Unfortunately, it is difficult to legally and efficiently measure the proportion of unpatched vulnerable devices on the Internet. Therefore, few comprehensive and quantitative experiments have proven the above viewpoint (i.e., that many devices on the Internet remain unpatched and vulnerable for years).

The most direct way to check the existence of a known vulnerability in an online device is using a proof of concept (PoC) tool [16][21]. However, this PoC-checking method faces legitimacy concerns and technical challenges. On the one hand, it is time-consuming to develop corresponding PoC tools for various vulnerabilities affecting various device models and firmware versions. On the other hand, triggering a real-world device's vulnerability without the device owner's authorization is illegal in most countries, even if the PoC's payload is harmless.

The fingerprint-scanning method is a more legitimate means to assess the proportion of unpatched vulnerable devices on the Internet. (e.g. Fig. 1). The public report of an IoT vulnerability usually contains affected device models and firmware versions[8]. It is possible to compute the proportion of unpatched vulnerable devices on the Internet by scanning fingerprints of models and

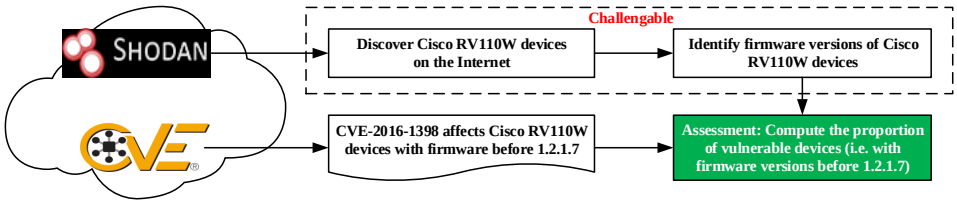

Figure 1: An instance of the fingerprint-scanning method: measuring unpatched Cisco RV110W devices vulnerable to CVE-2016-1398.

versions. This fingerprint-scanning method is more legitimate than the PoC-checking method because it triggers no vulnerability.

However, the fingerprint-scanning method also faces challenges. On the one hand, current device search engines (DSEs), such as Shodan [19] and Censys [3] [10], have been widely applied to discover devices on the Internet. However, they have never claimed to identify models or versions of devices accurately. On the other hand, although two previous works [6][14] have suggested distinguishing the firmware of online devices by fingerprinting embedded web applications, they neglected to address efficiency. Usually, hundreds of HTTP requests are required to identify a single target. Computing the proportion of unpatched devices with one vulnerability requires identifying many targets online. Consequently, the fingerprint-scanning method has been applied to only two CVE vulnerabilities to date, which is inadequate for a general assessment of unpatched vulnerable devices on the Internet.

The main contributions of this paper include the following:

- We propose ARGUS, which is a simple but practical framework that can identify device models and firmware versions on the Internet.
- We propose an FCS scheme that addresses the neglected efficiency issue of scanning redundant and useless fingerprints, improving efficiency by an average of 156 times compared to scanning all web file fingerprints by default.
- We determine that the average proportion of unpatched devices affected by 176 CVE vulnerabilities is 64.3% on the Internet. This is the first quantitative study proving that many users do not periodically update device firmware.

The rest of the paper is organized as follows. Section 2 analyzes the challenges of the fingerprint-scanning method. Section 3 details ARGUS and the FCS scheme. Section 4 evaluates our work by testing real-world devices and vulnerabilities. Section 5 discusses the limitations of this paper. Section 6 reviews the related literature. Section 7 offers a brief conclusion.

2 CHALLENGES

As shown in Fig. 1, applying the fingerprint-scanning method on the Internet to measure the proportion of unpatched devices affected by a specific vulnerability faces two challenges.

Challenge-A: Accurately Identifying Device Models on the Internet. The first step of the fingerprint-scanning method is to build a list of devices belonging to a targeted model. However, no DSE claims to identify any device models accurately. First, in terms of false negatives, some specific device models are unidentifiable to DSEs. DSEs collect only the banner messages of devices, which do not necessarily contain the specified device model's name. For instance, searching for "RV110W" as a keyword in any DSE cannot find real RV110W devices. The string "RV110W" does not exist in any banner messages from an RV110W device. Second, there are always false-positive records in DSEs for two reasons. (1) A website will also be returned as a matched record if its homepage includes the specified device model's name. (2) A DSE needs a period to crawl all over the Internet, and a recorded device may have changed its IP address after reboots. In section 4, we provide quantitative evaluations of these situations.

Challenge-B: Efficiently Recognizing Firmware Versions on the Internet. Two previous works [6] [14] have proven that fingerprinting embedded web applications is feasible to identify firmware versions. However, they did not address efficiency. Typically, the online identification of the firmware version of one device has to scan hundreds of web files by default. This is regardless of whether these files are distinguishable among versions and whether they are accessible without authentication. For instance, each firmware version of the Cisco RV110W device model has approximately 541 web files. This means that the online identification of one RV110W device version requires sending and receiving approximately 541 HTTP packets by default, which significantly restrains the efficiency of application at a large scale. A subset of web files is possibly sufficient as a fingerprint to uniquely identify a firmware version. However, there is currently no research to confirm this fingerprint from the vast number of candidate combinations of web files among different versions. Moreover, since some web files are inaccessible without authentication, they cannot be used as fingerprints even if they are unique. This means that this challenge is not a pure math problem that can be solved by a classic min-set algorithm.

3 ARGUS

In this paper, we propose ARGUS as a practical application of the fingerprint-scanning method. Its architecture (shown in Fig. 2) mainly consists of five modules. The collector, preprocessor, and extractor are responsible for automated initialization tasks, such as downloading, decompressing, annotating firmware images, locating web directories, and extracting information from web files. The

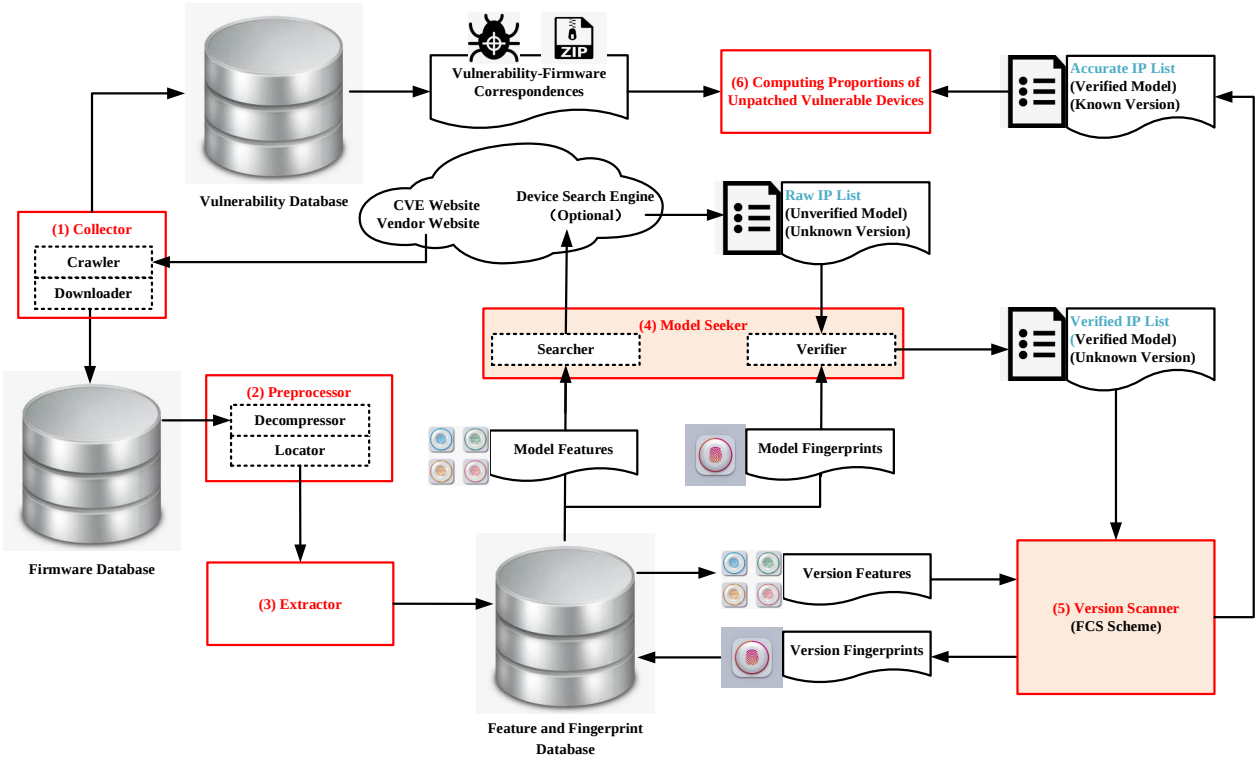

Figure 2: The Architecture of ARGUS

model seeker and the version scanner are responsible for device model identification and firmware version recognition, respectively, corresponding to the above two challenges.

This paper does not detail the above initialization tasks, which have been widely researched in previous works [2] [22] [4] [5] [6] [14]. Although there are still some difficulties in terms of initialization, e.g., how to decompress encrypted firmware or how to annotate unlabeled firmware, these problems are not the focus of this paper and can be bypassed by selecting only valid firmware images as our dataset. For instance, by only testing the firmware images unencrypted and well-labeled by vendors, e.g., WNR2000v5-V1.0.0.70.zip, this paper does not have to address the decompresses and annotation difficulties.

3.1 The Model Seeker

ARGUS uses a three-step approach to discover devices of a specified model on the Internet. (1) The extractor automatically analyzes firmware images, extracting the specified device model's features and fingerprints. (2) The searcher in the model seeker searches according to the features to discover candidate devices on the Internet that should belong to the specified model. (3) The verifier in the model seeker checks if a fingerprint exists in a candidate device to confirm whether the device truly belongs to the specified model.

It is worth noting that to build a raw IP list of candidate devices, using a DSE is convenient but not essential for ARGUS. A self-programmed script based on Zmap [11] can also feasibly build the raw list. In addition, we do not have to find every device on the Internet because this paper measures the proportion rather than the total number of unpatched vulnerable devices.

3.1.1 Discovering Candidate Devices. According to Challenge-A, using a specific device model as a keyword searched in a DSE can sometimes find no real-world devices belonging to the specific model. This situation occurs when no banner message contains the device's model name.

Assisted by firmware analysis, ARGUS can discover any models' online devices by searching for other features of the model in addition to its name. We define a model feature as a particular string in a banner message (e.g., a link in a device's login page). This approach is insufficient for uniquely identifying the specified model but sufficient for excluding many unmatched models. ARGUS can build a smaller raw IP list by searching for model features, only including relatively accurate candidates to be further verified.

For instance, the logo "/image/cisco_logo_about.png" on the page "login.asp" exists in all firmware versions of Cisco RV110W devices and some similar models. The login page has been crawled and stored by Censys as a banner message. Thus, there will be many records when searching the keyword "/image/cisco_logo_about.png". Some of the records are authentic Cisco RV110W devices; others

belong to similar models. This method is better than using RV110W as a keyword, from which we cannot find any authentic record.

3.1.2 Removing Mismatched Records. According to Challenge-A, regardless of what keywords are used, the raw results of DSEs always contain false-positive records such as mismatched models, websites, and outdated IP addresses.

ARGUS can remove all kinds of incorrect candidate records by verifying model fingerprints. We define a model fingerprint as a web file that contains a string of the model name, which is accessible via a specific URL. A false-positive record such as a website has little possibility of providing the same specific URL whose response contains the specified model's name.

For instance, the extractor automatically finds that, in all firmware versions of Cisco RV110W devices, there is a file "/lang_pack/EN.js" containing the string "RV110W". The verifier can check it as a model fingerprint to verify each candidate in the raw IP list. It removes all kinds of invalid records and builds a verified IP list that is fed to the version scanner.

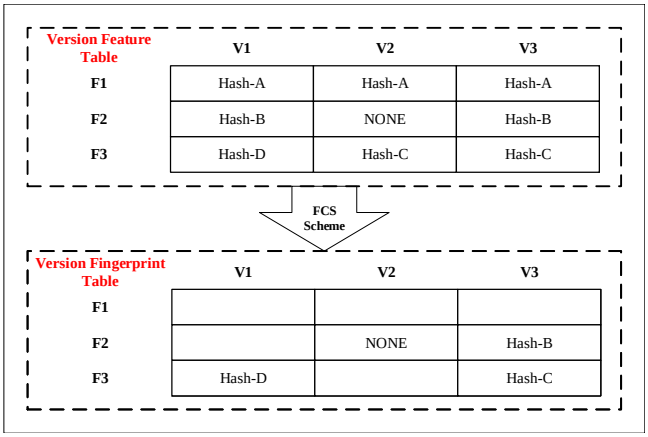

Figure 3: The structure of a feature table and its relationship with a fingerprint table

3.2 The Version Scanner

We can implement the version scanner by modifying existing open-source tools such as Wapplyzer, BlindElephant, plecost, w3af, whatweb, wpscan, and joomscan. These tools are designed for extracting fingerprints of websites. However, we would rather self-develop the version scanner and the model seeker using Python, as there are many technical differences between traditional websites and embedded web applications in IoT devices.

3.2.1 Feature Table and Fingerprint Table. First, we define MD5 hash digests of web files in a firmware image as the firmware version's features. Research [6] has proven that hash digestion is more dependable than the other three features of web files. However, it is noteworthy that the proposed FCS scheme is general and portable to previous related works regardless of a concrete definition of a version feature.

Then, we define a firmware version's fingerprint as one feature or a small set of features sufficient to distinguish the firmware version from other versions. Previous related works do not distinguish the two terms and test all version features as version fingerprints. This paper's main contribution is proposing a general scheme that can efficiently confirm version fingerprints from version features. It can accelerate online scans by recursively avoiding testing identical web files.

For each device model, the extractor automatically generates a feature table. The version scanner finally simplifies the feature table into a fingerprint table. The feature table columns represent firmware versions, while rows of the feature table denote embedded web files. Each cell in the feature table is the MD5 hash digest of the corresponding file in the corresponding version. In cases where a particular web file only exists in specific versions, the cells of the other versions that do not have the file are set as "NONE" by the extractor.

Fig. 3 shows a simple example to illustrate the mechanism of confirming fingerprints from features. Suppose a device model only has three firmware versions `V1`, `V2`, `V3` and three web files `F1`, `F2`, `F3`. Cell (2, 2) is marked as `NONE`, denoting that `F2` does not exist in `V2`. This is obviously a unique value of `F2` and therefore can be used as a fingerprint of `V2`. Similarly, `Hash-D` in cell (3, 1) can be used as a fingerprint of `V1` since it is a unique value in the third row. The fingerprint of `V3` is slightly more complex, as the third column does not contain one single cell that can distinguish `V3` from `V1` and `V2`. However, the combination of `Hash-B` in cell (2, 3) and `Hash-C` in cell (3, 3) can identify `V3` together, indicating that two HTTP packets are required to recognize a device running firmware `V3`. In addition, the hash digests of `F1` are identical to `Hash-A` in all versions, meaning that `F1` is useless for distinguishing versions. Therefore, the proposed scheme will directly delete the first row without sending any requests.

3.2.2 The Proposed FCS Scheme. Since a real-world case is much more complicated than the above simple example, making a feature table into a fingerprint table requires a general scheme. Generally, one device model has tens of firmware versions and hundreds of web files, creating a feature table with thousands of cells. Thus, confirming a uniquely identifiable combination of cells from millions or even billions of possibilities is a difficult task. Moreover, this issue is more than a theoretical math problem that can be solved by a min-set algorithm. For instance, some web files are inaccessible without user authentication. There seems to be no general method to know which files are publicly accessible unless all versions' files are scanned online. Therefore, many cells may be useless in practice, even if they can form a unique combination to identify a particular firmware version.

Inspired by the famous phone game "Candy Crush Saga", we propose a heuristic "fingerprint crush saga" (FCS) scheme. It can rapidly confirm a few fingerprints from a large number of features by iteratively deleting identical rows and unmatched columns of the feature table and by gaining knowledge from successful identifications.

The Beginning. The FCS scheme obtains an IP address from the verified IP list, in which each record has been proven to belong to the specified device model. The scheme then creates a temp table for the tested IP by copying the feature table of the device model.

Algorithm 1: The FCS Scheme

```
for each IP in the verified IP list do
    The Beginning:;
    T ← the feature table of the device model;
    // Rows of T represent embedded web files;
    // Columns of T represent firmware versions;
    // Cells of T represent corresponding hashes;
    N ← 1;
    //N represents the current row to check;
    The 1st Step:;
    if all cells in row N are identical then
        Remove row N from T;
        N ← N + 1;
        goto The 1st Step;
    The 2nd Step:;
    F ← the web file represented by row N;
    Send an HTTP request to the IP to access F;
    R ← the hash digest of the respond's body;
    if R matches no cells in row N then
        Remove row N from T;
        N ← N + 1;
        goto The 1st Step;
    The 3rd Step:;
    C ← columns having row N unmatch with R;
    Remove columns C from T;
    if row N is not the last row in T then
        N ← N + 1;
        goto The 1st Step;
    The End:;
    //The IP runs firmware of the rest of version(s);
    //T is the fingerprint of the rest of version(s);
    Move T to the top of the feature table to accelerate
      subsequent scans;
```

It initializes the row index representing the currently scanned file line as `N ← 1`.

The 1st Step. A conditional statement is judged to ascertain whether the cells of the current row have differences in some columns. If the cells are all the same, the current file is identical in all versions. Then, the row is removed from the temp table because the file is useless for recognizing versions. After that, the row index will increase by one step as `N ← N + 1` to repeat the same procedure until finding a row (file) having differences in some columns (versions).

The 2nd Step. The FCS scheme sends an HTTP request to the tested IP to access the web file represented by the current row `N`. Suppose the hash digest of the response body does not match any cells in all columns of the row `N` [1]. In this case, it is usually because the file requires authentication for accessing or contains dynamic scripts such as PHP [2]. Regardless of the reason for the mismatching, the row will be removed from the temp table, as the file is practically useless for firmware recognition. The FCS scheme then returns to the first step to check the next row and reruns the second step. It loops until finding a row in which some cells are distinguishable among columns (versions) and matched with the hash digest of the corresponding response body.

The 3rd Step. The FCS scheme removes those columns (versions) whose cells in the current row (file) do not match the hash digest of the response body (the response has proven that the tested device does not belong to those versions represented by the unmatched columns). After removing the columns, some rows that previously contain different cells in some columns only have identical cells in the remaining columns presently. This means that these rows will be removed later in the first step without sending requests in the second step. In other words, removing columns from the temp table can accelerate the speed of testing subsequent rows. Finally, if the current row is the last one, the FCS scheme goes to the end step; otherwise, it returns to the first step until completing tests for all rows in the temp table.

The End. After testing all rows, the temp table becomes a part of the fingerprint table. The remaining file(s) can together identify the remaining version(s). That is, if a device's responses match all the file(s) represented by the remaining row(s), it belongs to the version(s) represented by the column(s). Moreover, the FCS scheme can learn from each newly discovered fingerprint by moving the matched rows to the top of the feature table to accelerate the subsequent IP scans. It means immediately identifying devices of the same version without relocating its fingerprint from features [3]. Finally, the FCS scheme continues to test the next IP in the verified IP list until completing scans of all IP addresses.

An Instance. For clarity, an instance is shown in Fig. 4. It is not easy to tell the fingerprint of the firmware version `V5` by an initial glance. Five HTTP packets accessing the web files from `F1` to `F5` need to be sent by default. They verify all five features to identify an online device of `V5`. Using the FCS scheme, we can confirm that two features are enough to construct a fingerprint of `V5` by sending only three HTTP packets. Then, in subsequent scans after fingerprint confirmation, we can identify a device of `V5` by sending only two HTTP packets to verify the fingerprint, increasing the efficiency by 2.5 times. Moreover, the larger the feature table is, the more HTTP packets are prevented from being sent.

Fault-Tolerance. The above FCS scheme is robust to transient failures in three aspects. (1) It treats an HTTP failure with status code 404 (file not found) or 401 (unauthorized) as a normal situation in the 2nd step. (2) It resends failed HTTP requests and waits longer to deal with temporary network failures. (3) One device's failed

[1] Suppose the response indicates that the requested file does not exist in the tested device (i.e., the corresponding cell in the feature table is marked as `NONE`). In this case, it is also a matched situation.

[2] Although testing most dynamic scripts results in failures, some constant "dynamic" files can pass the 2nd step and finally be confirmed as a part of a fingerprint. Having a dynamic extension such as PHP does not always mean that the file's content contains dynamic codes. For instance, if renaming an HTML file as a PHP file without changing its content, the renamed PHP file will work the same as the original HTML file and have the same hash digest. Thus, there is no difference when handling the constant "dynamic" file compared to handling the HTML file.

[3] Static analysis lacks a general method to know whether a particular web file is inaccessible without authentication or whether a dynamic script truly has dynamic codes executed at the backend. However, we can avoid repeatedly scanning these useless files by confirming the fingerprints of firmware versions once for all.

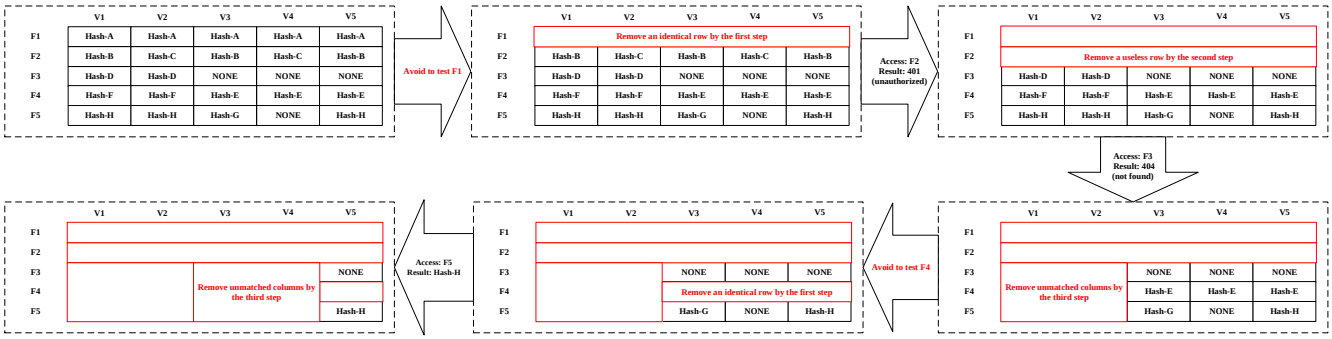

Figure 4: An instance of the FCS scheme: to identify a device running the V5 version, all five files need to be checked by default, but the FCS scheme confirms that checking F3 and F5 is sufficient.

test (e.g., because of moving offline while being tested) would not disturb the processes and results of other devices' tests.

FCS Scheme vs. Math Algorithms. We have mentioned in section 2 that a pure math algorithm cannot solve Challenge-B, whatever operations the math algorithm performs (e.g., matrix operations, set operations, group operations). A math algorithm may find the smallest combination of elements (files) uniquely identifying a set (firmware version). However, it cannot tell whether an element (file) in the combination can be verified online. In practice, a file in a device web interface may be inaccessible without authentication or show different content when statistical analysis and dynamic testing are performed. Further complicating matters, in some firmware versions, a specific file may not require authentication, while in other firmware versions, the same file may require authentication. In other words, Challenge-B is not a pure math problem. Its solution requires both offline reasoning and online testing simultaneously. The FCS scheme tests whether a candidate fingerprint is verifiable online while determining which files could be the candidate fingerprint. Essentially, FCS is not a math algorithm but an instance of a pruning algorithm applied to online fingerprint identification.

4 EVALUATION

To show the popularity of embedded web interfaces, we conduct a simple survey on devices opening web interfaces on the Internet, which is shown in Table 1. First, we search in Censys with the keyword "device" and obtain 13,520,347 records in which 13,352,131 records are tagged as "HTTP" and/or "HTTPS". The proportion is about 98.8%. Then, we set the keywords as "router", "camera", and "printer". The record numbers are 2,927,770, 373,738, and 356,453, while the proportion tagged as "HTTP" and/or "HTTPS" is 86.7%, 99.3%, and 80.2%. These results indicate the following: (1) most devices on the Internet have web interfaces; (2) the number of routers publicly accessible is approximately 7.8 times that of cameras or prints [4].

[4]Many IoT devices use Linux-based embedded systems, which are treated the same by ARGUS.

Table 1: Results of different keywords searched in Censys.

Searched Keyword	Device	Router	Camera	Printer
Total of Found	13,520,347	2,927,770	373,738	356,453
Tagged as HTTP(S)	13,352,131	2,538,126	371,037	286,022
Proportions	98.8%	86.7%	99.3%	80.2%

4.1 Dataset Selection

To evaluate ARGUS, we test 138 firmware images (versions) belonging to 10 device models from 5 vendors, as shown in Table 2. They are affected by 176 CVE vulnerabilities from 2014 to 2020. In terms of real-world experiments on the Internet, our dataset's vulnerabilities and firmware versions far exceed those of related works. Although thousands of firmware images are collected by ARGUS, similar to previous works, the dataset suitable for the real-world test has to meet three criteria. (1) All firmware images of a selected device model do not have downloading, decompressing, and annotating issues. (2) The selected model has at least hundreds of web files distinguished among different versions. (3) The selected model is affected by vulnerabilities reported in recent years, and the vulnerable versions are known.

Table 2: Selected device models for the real-world test

Vendor	Model	Firmware Images	Web Files	Vulnerabilities
Asus	RT-AC66U	27	641	12
	DSL-N12E	14	538	3
Cisco	RV110W	9	541	25
	RV320	17	2577	23
D-link	DIR-815	27	537	9
	DCS-933L	10	176	3
Netgear	WNR2000v5	13	491	90
	DGN2200	8	343	4
Zyxel	NBG-418N	5	215	3
	NBG-419N	8	471	4
Average		13.8	653	17.6

Table 3: Verifying device models of records from Shodan and Censys.

Vendor	Model	Shodan				Censys			
		Tested	Timeout	Mismatched	Matched	Tested	Timeout	Mismatched	Matched
Asus	RT-AC66U	956	332(34.7%)	0	624(65.3%)	1111	583(52.5%)	9(0.8%)	519(46.7%)
	DSL-N12E	29	19(65.5%)	0	10(34.5%)	22	10(45.5%)	0	12(54.5%)
Cisco	RV110W	0	-	-	-	0	-	-	-
	RV320	0	-	-	-	1078	802(74.4%)	24(2.2%)	252(23.4%)
D-link	DIR-815	837	502(60.0%)	15(1.8%)	320(38.2%)	1242	907(73.0%)	12(1.0%)	323(26.0%)
	DCS-933L	1158	687(59.3%)	5(0.4%)	466(40.2%)	1005	587(58.4%)	14(1.4%)	404(40.2%)
Netgear	WNR2000v5	4780	1439(30.1%)	6(0.1%)	3335(69.8%)	1003	386(38.5%)	0	617(61.5%)
	DGN2200	727	629(86.5%)	2(0.3%)	96(13.2%)	198	117(59.1%)	2(1.0%)	79(39.9%)
Zyxel	NBG-418N	719	250(34.8%)	7(1.0%)	462(64.3%)	532	138(25.9%)	1(0.2%)	393(73.9%)
	NBG-419N	0	-	-	-	133	83(62.4%)	6(4.6%)	44(33.1%)
Total		9206	3858	35	5313	6324	3613	68	2643
Average			53.0%	0.5%	46.5%		54.4%	1.2%	44.4%

4.2 Identifying Device Models

ARGUS provides an approach to verify raw DSE results. DSEs never claim they can accurately identify any device models. However, the most popular way to discover a specified model's online devices is to search for the model's name as a keyword in a DSE. We set the selected models' names as keywords (e.g., RT-AC66U) searched in Shodan and Censys. Then, we test the records tagged as HTTP/HTTPS services. We test all Shodan records and freely available Censys records using ARGUS to verify device models. The results in Table 3 show that this "name-search" method based on Shodan and Censys has average accuracies of 46.5% and 44.4%, respectively. It is worth noting that the timeout results are not necessarily stale DSE records. There is also a possibility that a firewall between a tested device and our lab blocks communications. Nevertheless, our experiment still shows that ARGUS can help verify device models and find accurate DSE records.

ARGUS also provides an approach to find "unidentifiable" devices whose model cannot be directly identified by any DSE. For instance, using the keyword "RV110W" either in Shodan or Censys cannot find any Cisco RV110W devices. Nevertheless, ARGUS can successfully find them via a feature and a fingerprint automatically extracted from firmware. Specifically, ARGUS first searches for devices whose login page contains a linked picture "/image/cisco_logo_about.png", finding 20,126 records. Then, ARGUS verifies them by requesting the URL of "https://IP/lang_pack/EN.js" to see if the string "RV110W" exists in the response, confirming 7% in total.

4.3 Recognizing Firmware Versions

In this subsection, we apply the FCS scheme to rapidly identify firmware versions of real-world devices. We first detail how an acceleration occurs by taking the D-link DCS-933L camera as an example. Then, we give the overall efficiency evaluation by testing the device models in our dataset. Specifically, we divide the procedure for online recognition of various firmware versions of one device model into two phases: fingerprint confirmation and version identification.

Accelerating Fingerprint Confirmation. With the FCS scheme's help, ARGUS can significantly increase the speed of confirming a new firmware version's fingerprint. For instance, the D-link DCS-933L camera has ten firmware versions, each of which has 176 web files. When ARGUS tries to identify the firmware version of the first target in the verified IP list, it recognizes no fingerprints of any versions. By default, sending 176 HTTP packets scanning all the web files (including those identical among versions and those inaccessible without authentication) is required to identify the target's version. According to the FCS scheme, however, ARGUS only sends 36 HTTP packets in total, confirming three matched hash digests of web files as the fingerprint of the target's version (i.e., 1.01b7). Specifically, as shown in Fig 5, the acceleration of the fingerprint confirmation is achieved in the following steps:

Packet A The 10th HTTP packet confirms that the 12th file has a match, 6 versions are ruled out, 164 files and 4 versions remain.

Packet B The 21st HTTP packet confirms that the 36th file has a match, 2 versions are ruled out, 140 files and 2 versions remain.

Packet C The 36th HTTP packet confirms that the 123rd file has a match, only 1 version remains, avoiding testing the last 53 files.

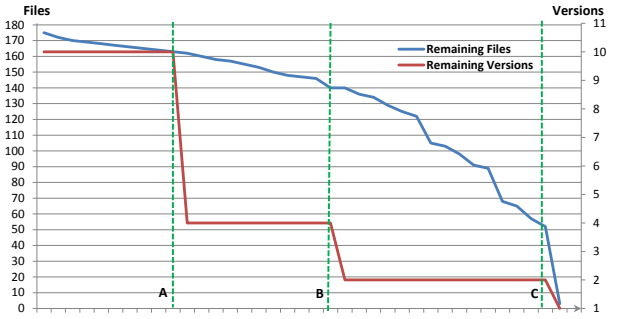

Figure 5: The process of identifying the first DCS-933L device's version and confirming its fingerprint quickly

Finally, the fingerprint in Table 4 indicates that in subsequent scans, only 3 HTTP packets are sufficient to identify a target that runs the same firmware version (i.e., 1.01b7).

Table 4: The confirmed fingerprint of the 1.01b7 version of the D-link DCS-933L device model.

Packet	Web URL	Hash Digest
A	http://targt/crossdomain.xml	fe390351906266330dd1c6809b5ed47c
B	http://targt/api/sounddb.jar	28bdcc0c3d0c30d6f307c96d0badd621
C	http://targt/api/h264dec.ver	41cf2677cc4ec9356dad8e76dfb87448

Accelerating Version Identification. The confirmation of the first target's fingerprint can accelerate subsequent scans on other targets by prioritizing the confirmed fingerprint checking. This is due to two facts. On the one hand, checking the same fingerprint can immediately identify devices running the same firmware version (i.e., 1.01b7). On the other hand, the three fingerprint files of 1.01b7 have different hash digests from the other versions; therefore, checking these three files in priority probably confirms new fingerprints of other versions at once. Fig. 6 takes D-link DCS-933L as an example. The average of HTTP packets required to identify a target declines with identifying more targets. Identifying a new target's version without known fingerprints may make the line move up slightly and temporarily. After confirming all versions' fingerprints, the line gradually becomes horizontal at number 5. This means that 5 HTTP packets are sufficient to identify the firmware version of a D-link DCS-933L device on average.

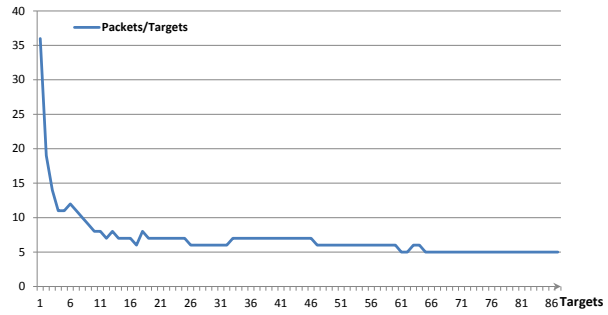

Figure 6: The average number of HTTP packets sent for identifying versions of subsequent DCS-933L devices

More generally, we define P_{FCS} as representing the average of HTTP packets sent according to the FCS scheme to identify a target, i.e.,

$$P_{FCS} = \frac{HTTP\ packets\ sent\ according\ to\ the\ FCS\ scheme}{Targets\ identified} \quad (1)$$

and

$$P_{DEFAULT} = The\ number\ of\ web\ files\ in\ a\ device \quad (2)$$

which represents the number of HTTP packets sent by default to scan features of all web files (including those identical among versions and those inaccessible without authentication). According to Table 5, on average, $P_{DEFAULT}$ is 156.3 times more than P_{FCS}, which means that the FCS scheme can significantly improve the efficiency of recognizing the firmware version of an online device.

Table 5: Efficiency improvement by the FCS scheme

Vendor	Model	$P_{DEFAULT}$	P_{FCS}	$P_{DEFAULT}/P_{FCS}$
Asus	RT-AC66U	641	5	128.2
	DSL-N12E	538	2	269.0
Cisco	RV110W	541	3	180.3
	RV320	2577	11	234.3
D-link	DIR-815	537	5	107.4
	DCS-933L	176	5	35.2
Netgear	WNR2000v5	491	4	122.8
	DGN2200	343	3	114.3
Zyxel	NBG-418N	215	1	215.0
	NBG-419N	471	3	157.0
Average		653	4.2	156.3

Theoretically, the FCS scheme is portable to other related works that neglect the efficiency challenge of scanning redundant fingerprints. However, we cannot quantitatively evaluate this point due to the difficulties in replicating previous works. Moreover, it makes little sense to compare ARGUS's absolute efficiency with related works with different hardware/network environments, different definitions of features/fingerprints, and different tested datasets. Nevertheless, no matter how related works define a target feature/fingerprint (e.g., hash digests, file structures), they always need to scan redundant features/fingerprints online. The FCS scheme can always accelerate the scanning process. Acceleration is vital in practice because different versions of the same model always have many identical web files. The importance can be proven by the above efficiency evaluation, which compares the FCS scheme with the default approach of scanning all web files.

4.4 Assessing Unpatched Devices

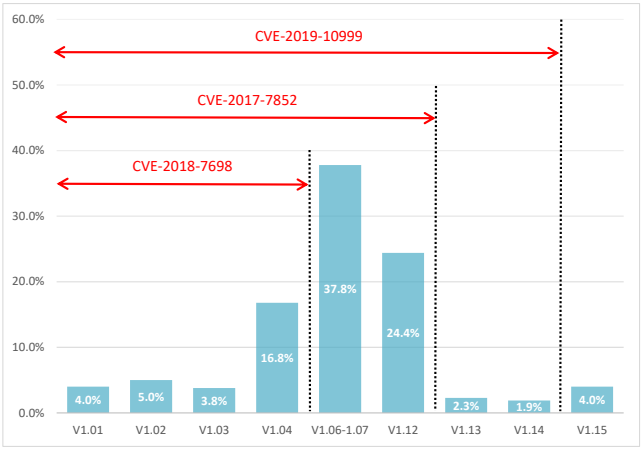

Figure 7: Computing the proportions of unpatched DCS-933L devices vulnerable to the 3 CVEs by accumulating the percentages of affected firmware versions.

Since the CVE website usually publishes affected versions of a vulnerability, accumulating the percentages of devices with the

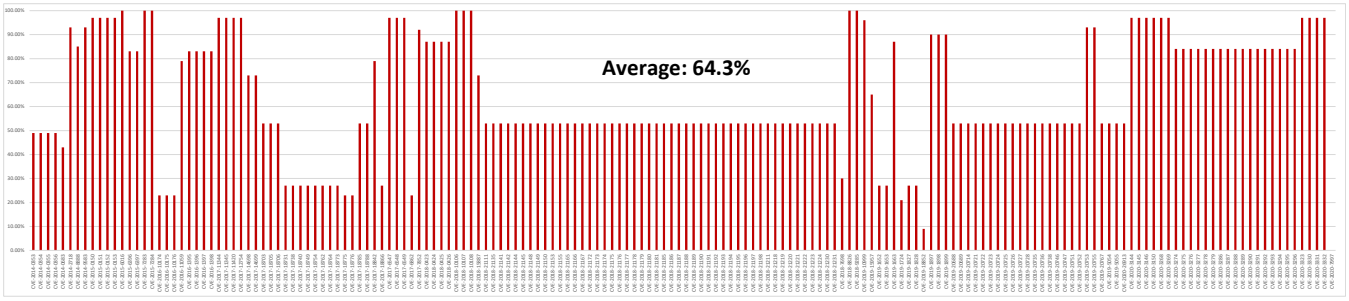

Figure 8: Proportions of unpatched devices vulnerable to the 176 CVEs.

affected versions can measure the proportion of unpatched devices still exploitable to the vulnerability. For instance, Fig. 7 shows the proportion of unpatched vulnerable DCS-933L cameras on the Internet. It is worth noting two points that do not considerably influence the result as follows. (1) The v1.06 and v1.07 versions cannot be distinguished, as they have only identical web files. However, this at least indicates that no vulnerability of the web interface exists between the two versions. (2) The vendor's official website does not provide firmware images of intermediate versions, which means that these versions are unlikely to be prevalent on the Internet.

More generally, we assess unpatched devices still vulnerable to the 176 CVEs in our dataset. The result in Fig. 8 proves two viewpoints. (1) The lower-bound proportion of unpatched devices on the Internet is 64.3% in our dataset by average, quantitatively proving that the threats of known device vulnerabilities are serious. (2) The proportions of unpatched vulnerable devices do not have apparent relationships with the vulnerability publication time. This result indicates that most users do not periodically update device firmware.

As shown in Table 6, this paper is more comprehensive than previous works related to assessing vulnerable devices. (1) Most previous works titled "assessing vulnerable IoT devices" actually focus on weak passwords rather than vulnerabilities (e.g., [7][18]). (2) Most previous assessments related to vulnerabilities are based on the PoC-checking method and face legitimacy concerns (e.g., [16][21]). (3) Although the work [6] proposed the fingerprint-scanning method, it did not assess any vulnerabilities or devices on the Internet. (4) The most recent work [14] based on the fingerprint-scanning method is evaluated on the Internet but only towards 17 firmware versions and 2 CVE vulnerabilities, which is a much smaller sample than in this paper.

Table 6: Comparisons of related works for vulnerable device assessments.

Related Works	Object	Method	Environment	Versions	CVE
Patton et al.[18]	Password	-	Internet	-	-
Cui et al.[7]	Password	-	Internet	-	-
Markowsky et al.[16]	Vulnerability	PoC-checking	Internet	-	0
Williams et al.[21]	Vulnerability	PoC-checking	Internet	-	0
Costin et al.[6]	Vulnerability	fingerprint-scanning	Laboratory	31	0
Li et al.[14]	Vulnerability	fingerprint-scanning	Internet	17	2
ARGUS	Vulnerability	fingerprint-scanning	Internet	138	176

5 LIMITATIONS

Non-optimal Efficiency. Although the proposed FCS scheme is more efficient than the default approach of scanning all web files, its efficiency is not the highest theoretically. After testing the web files of known fingerprints, testing the remaining web files is random if there is no match (i.e., needing to confirm a new version's fingerprint). It is possible to further increase efficiency by introducing a decision tree, which will be the subject of future work. In addition, we use the number of sent HTTP packets (i.e., the number of HTTP requests) to measure efficiency. This is only an approximate method. Although HTTP requests using the GET method are approximately the same size, the HTTP response size can vary dramatically.

Scalability Restriction. Some common traits restrict the feasibility of ARGUS. First, research based on firmware analysis requires the successful downloading and decompression of firmware images. Unfortunately, some vendors do not provide all firmware images for downloads, and decompressing an encrypted firmware image can be tricky. Second, the research based on fingerprints of embedded web applications cannot distinguish firmware images with identical web files. It is also inapplicable to ultralightweight devices such as wristbands, which rarely support embedded web applications. In addition, some users may modify their devices for anonymization. ARGUS can identify the correct models and versions of customized devices only when the fingerprints of the devices are not modified. However, some professional users may patch device vulnerabilities by themselves rather than using official updates. In this case, our assessments would still view the versions of the self-patched devices as vulnerable.

Accuracy Discussion. Although ARGUS can improve the accuracy of a DSE, we cannot evaluate ARGUS's accuracy on the Internet. On the one hand, we cannot verify whether a device on the Internet actually belongs to the identified model and version without the device's ownership. On the other hand, we believe the number of honeypots should be much lower than the number of real devices on the Internet, but this still affects accuracy slightly.

Legitimacy Debate. We cannot claim that ARGUS is universally lawful according to different laws all over the world. The fingerprint-scanning method is safer than the PoC-checking method because it does not trigger vulnerabilities. However, scanning devices on the Internet without ownership or authorization may also raise legitimacy debates in some countries. We can only claim that ARGUS

is as legitimate as existing DSEs such as Shodan or Censys that periodically crawl all over the Internet without any authorization.

6 RELATED WORKS

6.1 Online Device Discovery

There have been several famous DSEs, none of which have claimed to discover devices of specified models accurately. Shodan [19] is the first search engine that can find devices connected to the Internet. It can find a variety of IoT devices according to service banners, such as metadata about embedded servers, service options, and welcome messages. However, some Shodan records might be outdated because scanning the whole Internet takes a period of time. Censys [3] [10] is another powerful search engine that is efficient in updating maps of online devices. It uses Zmap [11], which is a modular, open-source network scanner specifically designed to perform Internet-wide scans. Zmap enables Censys to survey the entire IPv4 address space in 45 minutes, even on a single machine. Nevertheless, neither Shodan nor Censys provides a service to identify devices accurately. They both depend on banner information that does not always contain identifiers of device models and/or firmware versions.

Xuan et al. [12] proposed an acquisitional rule-based engine (ARE), which can automatically generate rules for discovering and annotating IoT devices without any training data. The ARE can build device rules by leveraging application-layer response data from IoT devices and product descriptions in relevant websites for device annotations, thus defining a transaction as a mapping between a unique response to product description. The authors also proposed an association algorithm to generate rules of IoT device annotations in the form of type, vendor, and model. Experiments and three applications have validated the effectiveness of the ARE.

Wang et al. [20] noted that the ARE as well as other DSEs cannot identify devices without vendor or product (i.e., model) keywords in the response data (i.e., service banners). They proposed IoTTracker, which can identify these “stubborn" devices by leveraging the high similarity of response data between IoT devices of the same vendor or product. IoTTracker can identify 40.76% more devices than the ARE according to their experiments. However, IoTTracker can only identify the vendor and the series rather than the device’s exact model. For instance, a Cisco RV110W device can be identified as a “Cisco RV" series device, while the model number “110W” remains unidentified, which may be, e.g., RV130W, RV320, or RV325.

In general, current research can efficiently discover devices on the Internet but is less able to identify an IoT device’s exact model.

6.2 Firmware Fingerprint Identification

Some works have focused on device fingerprint identification, but they target hardware rather than firmware. Kohno et al. [13] introduced the notion of remote fingerprinting a specific physical device as opposed to an operating system or class of devices. The authors accomplished their goal by exploiting small clock skews and microscopic deviations in the device hardware. Desmond et al. [9] proposed a fingerprinting technique that differentiates between unique devices over a wireless local area network (WLAN) through a timing analysis of 802.11 probe request frames. Li et al. [15] proposed a novel PrinTracker, which can trace a physical object to its source 3D printer based on its fingerprint. However, there is still a lack of a general method to fingerprint diverse firmware versions of various IoT devices.

Costin et al. [6] first proposed a general approach based on fingerprints of embedded web applications to identify vendors, models, and versions of devices. The authors proved that the cryptographic hash digests of the HTML content in embedded web files are the most stable and accurate features for firmware fingerprinting. However, the authors did not test their work on the Internet, so the efficiency of identifying firmware versions of real-world devices cannot be judged.

Li et al. [14] also proposed identifying firmware versions of IoT devices based on fingerprints of embedded web files. They leveraged the natural language processing technique to process the content and the document object model of embedded web files to generate firmware fingerprints. However, they also did not distinguish fingerprints from features. As a result of checking many redundant fingerprints, according to their experiments, 75 HTTP packets must be sent on average to identify the firmware version of a single device. We believe this is less efficient in large-scale online applications.

In brief, fingerprinting embedded web files has been proven to be a feasible approach to identify firmware versions of IoT devices. However, a widely neglected issue is scanning redundant fingerprints, preventing this approach from achieving large-scale verifications on the Internet.

6.3 Vulnerable Device Assessment

Most previous works on vulnerable device assessment actually address default or weak passwords instead of firmware vulnerabilities. For instance, Cui et al. [7] presented a quantitative lower bound on the number of “vulnerable" embedded devices globally. They identified over 540,000 publicly accessible embedded devices configured with factory default root passwords. Similarly, Patton et al. [18] investigated “vulnerable" devices on the Internet by using Shodan but only focused on devices having default passwords rather than firmware vulnerabilities.

Although some previous works assessed devices with vulnerabilities, most of them leveraged the PoC-checking method, creating legitimacy concerns because of triggering real-world vulnerabilities. For instance, Markowsky et al. [16] demonstrated three types of scans for vulnerable IoT devices on the Internet, including using Masscan to find devices vulnerable to Heartbleed and using Nmap and PFT to find and connect to vulnerable networked printers. Similarly, Williams et al. [21] performed a large-scale vulnerability assessment of IoT devices on the Internet. They integrated Nessus to determine whether known vulnerabilities exist in a vast number of IoT devices. In addition to legitimacy concerns, these works were also limited because not all vulnerabilities for all models and versions have corresponding PoC tools.

To the best of our knowledge, only two previous works mentioned the feasibility of assessing unpatched vulnerable devices by identifying firmware versions [6][14]. However, only two CVE vulnerabilities have been assessed by using this fingerprint-scanning method so far.

7 CONCLUSION

This paper proposes ARGUS, a simple but practical framework to assess unpatched vulnerable devices on the Internet. It can identify device modes and firmware versions via fingerprints of embedded web files. The core component of ARGUS is a heuristic FCS scheme. It addresses the widely neglected efficiency challenge of online scanning redundant fingerprints. On average, it can improve efficiency by 156 times compared to the default scanning of all web files. This efficiency improvement enables us to quantitatively measure the severe effects of known IoT vulnerabilities for the first time. For example, the average proportion of unpatched devices affected by 176 CVE vulnerabilities is 64.3% on the Internet. The result also proves that most users do not periodically update the firmware of their devices. Our future work includes the following: (1) introducing a decision tree to further increase efficiency; (2) improving accuracy by combining more fingerprints from various protocols and applications (e.g., SSL fingerprints, FTP fingerprints, OS fingerprints).

ACKNOWLEDGMENT

We would like to thank the anonymous reviewers for their valuable comments and helpful suggestions.

REFERENCES

[1] Manos Antonakakis, Tim April, Michael Bailey, Matt Bernhard, Elie Bursztein, Jaime Cochran, Zakir Durumeric, J Alex Halderman, Luca Invernizzi, Michalis Kallitsis, et al. 2017. Understanding the mirai botnet. In *26th USENIX Security Symposium (USENIX Security2017)*. 1093–1110.

[2] Binwalk. [n.d.]. A tool for analyzing, reverse engineering, and extracting firmware images. [Online]. Avaliable: https://github.com/ReFirmLabs/binwalk.

[3] Censys. [n.d.]. A search engine based on Internet-wide scanning for the devices and networks. [Online]. Avaliable: https://censys.io/.

[4] Daming D. Chen, Manuel Egele, Maverick Woo, and David Brumley. 2016. Towards Automated Dynamic Analysis for Linux-based Embedded Firmware. In *Network and Distributed System Security Symposium*.

[5] Andrei Costin, Zaddach Jonas, Francillon Aurelien, and Balzarotti Davide. 2014. A Large-Scale Analysis of the Security of Embedded Firmwares. In *USENIX Security Symposium*. 95–110.

[6] Andrei Costin, Apostolis Zarras, and Aurelien Francillon. 2017. Towards Automated Classification of Firmware Images and Identification of Embedded Devices. In *IFIP International Conference on ICT Systems Security and Privacy Protection*. 233–247.

[7] Ang Cui and Salvatore J. Stolfo. 2010. A quantitative analysis of the insecurity of embedded network devices:results of a wide-area scan. In *Twenty-Sixth Computer Security Applications Conference, ACSAC 2010, Austin, Texas, Usa, 6-10 December*. 97–106.

[8] CVE. [n.d.]. Common Vulnerabilities and Exposures. [Online]. Avaliable: http://cve.mitre.org.

[9] Loh Chin Choong Desmond, Cho Chia Yuan, Chung Pheng Tan, and Ri Seng Lee. 2008. Identifying unique devices through wireless fingerprinting. In *ACM Conference on Wireless Network Security, WISEC 2008, Alexandria, Va, Usa, March 31 - April*. 46–55.

[10] Zakir Durumeric, David Adrian, Ariana Mirian, Michael Bailey, and J Alex Halderman. 2015. A search engine backed by Internet-wide scanning. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*. ACM, 542–553.

[11] Zakir Durumeric, Eric Wustrow, and J. Alex Halderman. 2013. ZMap: fast internet-wide scanning and its security applications. In *Usenix Conference on Security*. 605–620.

[12] Xuan Feng, Qiang Li, Haining Wang, and Limin Sun. 2018. Acquisitional Rule-based Engine for Discovering Internet-of-Things Devices. In *27th USENIX Security Symposium (USENIX Security 18)*. USENIX Association, Baltimore, MD, 327–341. https://www.usenix.org/conference/usenixsecurity18/presentation/feng

[13] Tadayoshi Kohno, Andre Broido, and K. C Claffy. 2005. Remote Physical Device Fingerprinting. *IEEE Transactions on Dependable and Secure Computing* 2, 2 (2005), 93–108.

[14] Qiang Li, Xuan Feng, Raining Wang, Zhi Li, and Limin Sun. 2018. Towards Fine-grained Fingerprinting of Firmware in Online Embedded Devices. In *IEEE INFOCOM 2018 - IEEE Conference on Computer Communications*.

[15] Zhengxiong Li, Aditya Singh Rathore, Chen Song, Sheng Wei, Yanzhi Wang, and Wenyao Xu. 2018. PrinTracker: Fingerprinting 3D Printers using Commodity Scanners. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 1306–1323.

[16] Linda Markowsky and George Markowsky. 2015. Scanning for vulnerable devices in the Internet of Things. In *IEEE International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications*. 463–467.

[17] Talos of Cisco. 2018. New VPNFilter malware targets at least 500K networking devices worldwide. [Online]. Avaliable: https://blogs.cisco.com/security/talos/vpnfilter.

[18] Mark Patton, Eric Gross, Ryan Chinn, Samantha Forbis, Leon Walker, and Hsinchun Chen. 2014. Uninvited Connections: A Study of Vulnerable Devices on the Internet of Things (IoT). In *Intelligence and Security Informatics Conference*. 232–235.

[19] Shodan. [n.d.]. A search engine for Internet-connected devices. [Online]. Avaliable: https://www.shodan.io/.

[20] Xu Wang, Yucheng Wang, Xuan Feng, Hongsong Zhu, Limin Sun, and Yuchi Zou. 2019. IoTTracker: An Enhanced Engine for Discovering Internet-of-Thing Devices. In *2019 IEEE 20th International Symposium on" A World of Wireless, Mobile and Multimedia Networks"(WoWMoM)*. IEEE, 1–9.

[21] Ryan Williams, Emma Mcmahon, Sagar Samtani, Mark Patton, and Hsinchun Chen. 2017. Identifying vulnerabilities of consumer Internet of Things (IoT) devices: A scalable approach. In *IEEE International Conference on Intelligence and Security Informatics*. 179–181.

[22] Weidong Zhang, Hong Li, Hui Wen, Hongsong Zhu, and Limin Sun. 2018. A graph neural network based efficient firmware information extraction method for IoT devices. In *2018 IEEE 37th International Performance Computing and Communications Conference (IPCCC)*. IEEE, 1–8.