Towards Fine-grained Fingerprinting of Firmware in Online Embedded Devices

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

***Abstract*—An increasing number of embedded devices are connecting to the Internet at a surprising rate. Those devices usually run firmware and are exposed to the public by device search engines. Firmware in embedded devices comes from different manufacturers and product versions. More importantly, many embedded devices are still using outdated versions of firmware due to compatibility and release-time issues, raising serious security concerns. In this paper, we propose generating fine-grained fingerprints based on the subtle differences between the filesystems of various firmware images. We leverage the natural language processing technique to process the file content and the document object model to obtain the firmware fingerprint. To validate the fingerprints, we have crawled 9,716 firmware images from official websites of device vendors and conducted real-world experiments for performance evaluation. The results show that the recall and precision of the firmware fingerprints exceed 90%. Furthermore, we have deployed the prototype system on Amazon EC2 and collected firmware in online embedded devices across the IPv4 space. Our findings indicate that thousands of devices are still using vulnerable firmware on the Internet.**

I. Introduction

Numerous embedded devices connected to the Internet, such as residential gateways/routers, IP-cameras, and net-printers, play a significant role in our daily lives. Those devices are assigned with IP addresses and accessible on the Internet. The firmware is a type of software semi-permanently placed in the hardware of embedded devices, providing basic control, monitoring, and data manipulation. In general, the firmware information (especially vendor names and product versions) is associated with security vulnerabilities in embedded devices. For instance, the buffer overflow vulnerability (CVE-2015-4409) exists on Hikvision NVR DS-76xxNI-E1/2 and DS-77xxxNI-E4 devices with a firmware version prior to V3.4.0 [1]. Obtaining the details of firmware in embedded devices is a pre-requisite to exploit or secure those devices from offensive or defensive perspectives.

One pervasive but easily overlooked fact is that many embedded devices are using mixed-version firmware on the Internet. First, some outdated devices are not compatible with the latest version of the firmware. Second, manufacturers and vendors often release updates to the firmware after distributing their products. When bugs and vulnerabilities of the firmware are revealed, manufacturers and vendors are willing to publish updates to the firmware. Firmware updates always occur later than a products release. Third, updating firmware is a non-trivial process for many users. They have to download firmware images through the official support website or via the administrative tools and install the firmware into the ROM to reprogram integrated chip circuits of embedded devices. Therefore, many embedded devices are still using the older version firmware even when the update has been distributed.

Current device search engines (e.g., Shodan [2] and Censys [3]) collect most devices on the Internet and expose the data to the public. Any user can query results through keywords and obtain the data of online embedded devices. For instance, if we use the keyword "camera", Shodan would return 287,551 items about camera devices with information such as IP addresses, locations, and packet payloads. However, the granularity of this information is too coarse to help us manage and protect those devices. The exposure of embedded devices to the public indicates that firmware is also exposed in the cyberspace. Fine-grained information of firmware can inform us which online embedded device is still vulnerable.

In this paper, we propose generating the fine-grained fingerprints of the firmware. Fine-grained fingerprints indicate which device, manufacturer, and product version the firmware origins from. Based on the firmware fingerprints, we can find the devices using firmware with a particular vulnerability on the website of Common Vulnerabilities and Exposures (CVE) [4]. An intuitive insight behind our work is that different firmware images are distinct from one another due to their filesystems. Today's embedded devices usually utilize Linux-based filesystems, where tens of thousands of files reside. The filesystem can act as the signature for recognizing firmware at the fine-grained level. Especially, we leverage the directory "WWW" in the filesystem as its files can be obtained online.

We send requests to online devices and receive their response data. The greatest challenge is the inconsistency between the response data and local files of the firmware. Some files are resources (i.e., "*.jpg," "*.icon," and "*.ccs"); some files are forbidden to access to login; and some files are dynamic and changing according to different environments, like JavaScript. To eliminate this inconsistency, we propose using the natural language processing technology and

†Zhi Li is the corresponding author.

document object model (DOM) to present the file. We filter out inaccessible data and irrelevant content. A DOM tree is used to generate the matrix of the firmware fingerprint. For each item in the matrix, we use the HTTP GET method to acquire <request, response> and recognize firmware at the fine-grained level remotely. Note that our fingerprints can recognize firmware according to the accessible files in the directory "WWW" without requiring a password or login permission.

To validate firmware fingerprints, we have implemented a prototype system and conducted real-world experiments. We have crawled 9,716 firmware images across seven manufacturers' websites and thousands of product versions. Our results show that firmware fingerprints can achieve a 91% precision and a 90% recall. Furthermore, we have deployed the prototype system at a cloud server inside Amazon EC2 [5] to collect firmware on the Internet-wide scale. We combine the vulnerability in CVE websites [4] and the firmware versions for quantifying the real-world effect of the firmware to the online embedded devices. Our findings indicate that thousands of online embedded devices are still using vulnerable firmware.

Overall, our contributions are summarized as follows:

- We have proposed automatically generating firmware fingerprints at the fine-grained level based on the differences in the filesystems. It is the first work to recognize firmware on the Internet.
- We have implemented a prototype system and conducted real-world experiments for verifying the effectiveness of our fingerprinting approach.
- We have used the fingerprints to discover embedded devices still running outdated versions and vulnerable firmware on the Internet.

The remainder of the paper is structured as follows: Section II describes the motivation of firmware fingerprinting. Section III presents the automatic fingerprint generation of online embedded devices. Section IV details the implementation and real-world experiments. Section V surveys related work, and finally, Section VI concludes.

II. Motivation

In this section, we present the motivation of firmware fingerprinting. Firmware is the native software embedded in the hardware of devices, simply known as "software for hardware".

The firmware category has three levels, which are shown in Figure 1 with respect to the device type, manufacturers, and product versions. First, the firmware runs at different device types, like IP-cameras or webcams. The device type is the most coarse-grained level for firmware category and the easiest case to recognize on the Internet. Censys [3] and Shodan [2] use keywords to identify the device type. Second, the firmware comes from different manufacturers or vendors, like Netgear, Hikvision, and D-Link. This level indicates the organizations that distribute device products to the market. The manufacturers would release firmware images to install on their products and systems. The challenge of identifying this

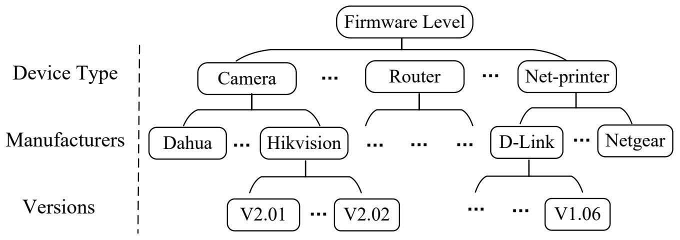


Fig. 1. The fine-grained levels of firmware.

category is that there are many manufacturers, and it is hard to keep the fingerprinting updated with the addition of numerous new devices. Third, the firmware has a different version while coming from the same device type and manufacturer. This is the hardest case to identify on the Internet. For instance, D-Link V4.12 and V4.13 firmware images are installed on routers. We observe that their filesystems are almost the same. As far as we know, there is no previous work that can recognize the firmware at such a fine-grained level.

Identifying firmware at the fine-grained levels can bring several benefits. First, the vulnerability is directly associated with the firmware version rather than the device type and manufacturer. The trigger condition of the vulnerability consists of the platform. The same vulnerability will not happen when it runs on different platforms. For embedded devices, the platform is the firmware version. From the defensive perspective, discovering the firmware can help us find out which online devices are still vulnerable. Second, we can use firmware fingerprints to find devices on the Internet and quantify the real-world impact regarding those mixed-version firmware devices (instead of offline analysis). Security is not only about the technology, but also about how to manage devices. Administrators can know the number of outdated versions of firmware being used within their enterprise network and notify device owners to update the firmware once manufacturers distribute the updates. Third, it helps manufacturers be aware of the influence of the updated version after the release date (i.e., how many users actively update their firmware over time).

We propose generating fine-grained fingerprints of firmware based on the following observation. When manufacturers fix bugs, add features, or improve performance, developers must change documents in the filesystem for updating the firmware. This modification on the filesystem can induce the subtle differences between mixed-version firmware images. We utilize the filesystem to act as the firmware signature and discover those embedded devices that still use outdated firmware versions on the Internet. Here, we only focus on the Linux-based filesystem, which is the most commonly seen in today's embedded devices, including residential routers, camera, and printers.

III. Fingerprinting Firmware

In this section, we first introduce the overview of the fingerprint generation approach. Then, we present the details of the system design for fine-grained firmware fingerprints.

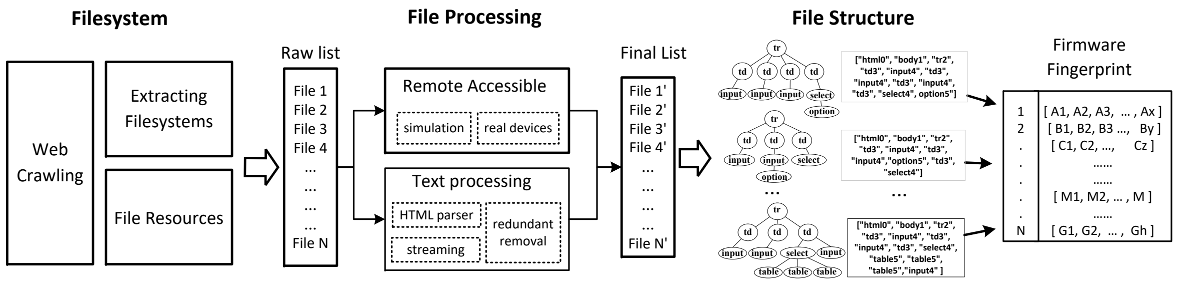


Fig. 2. The overview of system design for generating fine-grained fingerprints of firmware.

A. Overview

Architecture. Figure 2 shows the system overview for generating firmware fingerprints, including the filesystem, file processing, and file structure modules. In the filesystem module, we use web crawling scripts to download firmware images from official websites of manufacturers. The filesystem is extracted from the firmware image, and we obtain its files in terms of filename, location, and extension. We obtain the raw list from the filesystem, where each item is in the format "location/filename" associated with a file. In the file-processing module, we determine whether local files are consistent with their respective response data. For remote accessibility, we use the system emulator to simulate the firmware in the virtual environment and obtain accessible files. For the file's content, we use the natural language processing (NLP) to parse the file, remove redundant content, and stream. After the text processing, we obtain the final list with each item in the format "location/filename" associated with a string. In the file structure module, we transfer every file in the final list to a DOM tree and a vector. We put all vectors together to generate the firmware matrix. The matrix is the firmware fingerprint.

Example. To explain how our approach works, consider this example. We download the D-Link DIR-300 Wireless Router Firmware with version 1.06 from its support page on the official website and interpret the binary image to extract the filesystem. For this image, a raw list contains 389 different kinds of file types (e.g., PHP, HTML, XML) in the directory "WWW". Then we simulate the firmware image in the virtual hosts and acquire its online accessible files. There are 203 files in the final list from which we receive a positive reply. For every file, we remove redundant content and use a DOM tree to transfer it to a structured vector. The fingerprint of the firmware D-Link V1.06 is the matrix including 203 vectors. For each vector, we generate the request based on its directory and filename. We receive the response data from online devices. We then identify this firmware through calculating the similarity between responses and vectors in the matrix.

B. Filesystem

Collecting the firmware filesystem is the first and largely independent component for generating fingerprints. We crawl firmware images, extract the filesystems from their binary images and generate a raw file list based on the filename, location, and extension.

Web Crawler. Most device manufacturers provide a support page on their official websites where we can obtain the device firmware and detailed descriptions, such as the vendor, product name, release date, version number, and change logs. Our web scraper is essentially a crawler designed to parse the URL and download firmware images at manufacturers' websites. Because each website has different configuration templates, the web crawler has a corresponding scraping script for searching firmware images. Particularly, we perform breadth-first searching on websites to explore the URL of the firmware, until no new links can be found. We use keywords (e.g., "download" and "firmware") to discover these downloaded URLs. After downloading these binary images, we use the metadata on the official websites to label the image with information, such as device type, vendor name, and product version.

Filesystem Extraction. All of the firmware images we downloaded are binary files. BINWALK [6] is a third-party tool to analyze and reverse-engineer binary files. We use BINWALK to unpack binary files of firmware images. In general, most firmware filesystems have more than 1,000 files. Then, we store all files for every firmware image.

Not all files are used to recognize the firmware on the Internet. Many directories (e.g., "bin", "etc" and "lib") in the filesystem cannot be remotely accessed. Among these folders, the directory "WWW" is used to provide web services for users. Device manufacturers and vendors usually develop the web interface as administrative tools for updates and management. One advantage is that we can obtain the content of the files in the directory "WWW" by sending HTTP methods. For instance, if an embedded device with IP "10.20.30.6" owns the file "help.html" in the location "WWW/XX, we can directly send the HTTP GET request "10.20.30.6/XX/help.html" to access this file. Therefore, we use the files in this directory to generate the fingerprints of the firmware.

Furthermore, we use the location, filename, and extension to extract files from the directory "WWW". The location is the relative path similar to the directory for files, which is the parameter URL in the HTTP method. In each iterative search directory, we use the backslash to concatenate the directory name with the sub-directory name. We extract their filenames and add them to the raw list. For each item in the raw list, we use the "location/filename" to represent the file. There are various file formats in the filesystem. We only keep files with text contents. If the file consists of non-textual information (e.g., images, audio, and videos), we just drop it out. We use the file extension to filter out unqualified files, such as "*.png". We store the remaining files in the raw list.

C. File Processing

The raw list cannot directly act as the signature of the firmware because there are several inconsistent places between the local files and the response data from online devices. Some files are forbidden to access, and some are dynamic. If we use these files to generate firmware fingerprints, we would obtain ineffective response data, leading to performance degradation. We eliminate the different places between local files and the response data through a heuristic rule:

- If the local file is different from the remote response data, we delete the item; otherwise, we keep it.

We first find out whether the file can be remotely accessible through simulating the firmware. Then, we use the natural language processing technology to pull data from files, redirect, and remove redundancies.

Remote Accessiblity. We do not assume that we have the password and login permissions of embedded devices. We propose to send HTTP requests to determine whether files in the raw list are accessible. Typically, web services can give the response data with encapsulated headers, including Method, URL, Parameters (GET and POST), Cookies, and User-Agent [7]. We use the GET method and set URL as a concatenation of the directory and filename. Firmware images are running in either the simulation environment or real devices.

We propose using the system emulator QEMU [8] to run the bootstrap of kernels of the Linux-based filesystem. If we know the architecture (i.e., ARM or MIPS) and endianness (i.e., big-endian or little-endian format) of the firmware image, the QEMU emulator can simulate the images in the virtual environment. Once the firmware image is emulated, the virtual host would be allocated a network interface and an individual IP address. We use the ping request to check whether the virtual host's network connectivity is available. We do not focus on other hardware-specific peripherals of embedded devices because the firmware fingerprints only need the files in the directory "WWW". However, some manufacturers do not provide information about the architecture and endianness of the firmware images for business reasons. The state-of-the-art emulator tool cannot simulate this kind of firmware image. In this case, we suggest using real devices to run the firmware images.

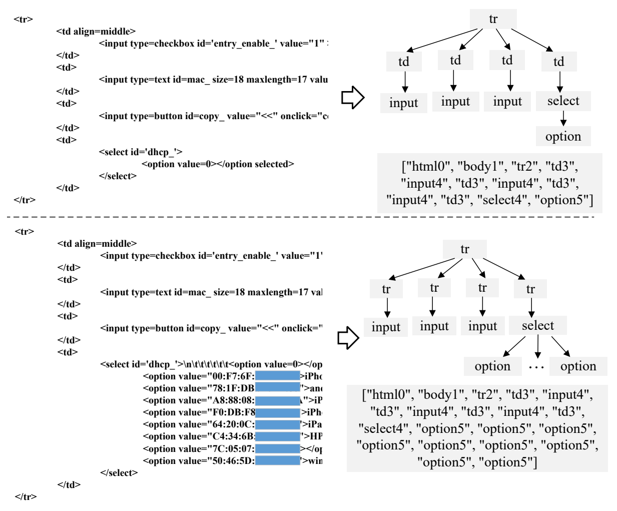
Text Processing. Once the firmware is running, we use the HTTP GET method and the URL parameter is "IP/location/filename". According to HTTP status codes [7], if the virtual host gives the response, we store the response data in the candidate list. If the virtual host does not give the response data, we would remove it from the raw list. Sometimes, the emulator cannot simulate firmware correctly because of the customization configuration of the firmware. Even with the status code is 200 OK, we still obtain the wrong response data, such as a NULL payload or an error message for the same HTTP request. In this case, we directly filter out those files. The remaining local files are stored in the raw list, and their response data is stored in the candidate list.

The redirect can bring about the inconsistency between local files and response data. Developers would move the pages to a new location for load balancing or simple randomization. For instance, firmware D-Link DIR-300 version 1.06 has a JavaScript file that redirects the request with the URL "/index.html" to the URL "bsc_internet.php". The response data would be different from local files in the filesystem. We use the regular expression to extract the new location and filename in the redirected places of files. We replace the item in the raw list with the "new location/new filename". Here, we focus on two typical files for redirections: HTML redirects and JavaScript redirects.

Content redundancy is another factor influencing the response data. We use the natural language processing technology to handle the content of files. There are usually conjunctive words, delimiter separated words, and letter-case separated words. For instance, words are divided by the symbol '/' in the hyperlinks. Regular expressions are used to split them into individual words. We use stemming to transfer words to their original or root forms (e.g., "services" is replaced by the word "service"). The stemming process reduces the amount of textual information. After stemming, we remove numbers, punctuations, and stopwords. Stopwords are some of the most common words, such as "the" and "is." There are many tags surrounded by angle brackets for the files in the raw list. These tags appear in pairs like $< p >$ and $< /p >$. The first one is the start indicator, and the second is the end tag with a forward slash. We keep these tags to extract the file structure to present the firmware in the next section (§III-D). After file processing, we obtain the final list of the firmware: each item has the directory path, filename, and the content.

D. File Structure

Dynamic files would cause inconsistent places between the response data and local files in the firmware filesystem, like JavaScript files. When the same firmware is running in different environments, dynamic files will change the file content from the original, such as a set of parameters. Figure 3 shows the same file "__adv_mac_filter" of the same firmware (D-Link DIR-300 Wireless Router, Firmware Version V1.06) in different environments. The upper one is running in the QEMU emulator, and the lower one is running on the real device. The variable "option" in the file will be dynamically changed according to the particular environment. When we use the emulator to run the firmware, the field "option" is blank (Figure 3, upper). When the real device is running the firmware, the field "option" shows MAC addresses and names of eight attached devices (Figure 3, lower). We can't directly use the response data to represent the local file because of the



```
<tr>
        <td align=middle>
                <input type=checkbox id='entry_enable_' value="1" :
        </td>
        <td>
                <input type=text id=mac_ size=18 maxlength=17 valu
        </td>
        <td>
                <input type=button id=copy_ value="<<" onclick="c
        </td>
        <td>
                <select id='dhcp_'>
                        <option value=0></option selected>
                </select>
        </td>
</tr>
```

["html0", "body1", "tr2", "td3", "input4", "td3", "input4", "td3", "input4", "td3", "select4", "option5"]

```
<tr>
        <td align=middle>
                <input type=checkbox id='entry_enable_' value="1'
        </td>
        <td>
                <input type=text id=mac_ size=18 maxlength=17 val
        </td>
        <td>
                <input type=button id=copy_ value="<<" onclick="
        </td>
        <td>
                <select id='dhcp_'>\n\t\t\t\t\t\t<option value=0></op
                        <option value="00:F7:6F:[redacted]>iPho
                        <option value="78:1F:DB[redacted]'>and
                        <option value="A8:88:08:[redacted]">iP
                        <option value="F0:DB:F8[redacted]>iPh
                        <option value="64:20:0C:[redacted]'>iPa
                        <option value="C4:34:6B:[redacted]'>HP
                        <option value="7C:05:07:[redacted]></op
                        <option value="50:46:5D:[redacted]'>win
                </select>
        </td>
</tr>
```

["html0", "body1", "tr2", "td3", "input4", "td3", "input4", "td3", "input4", "td3", "select4", "option5", "option5", "option5", "option5", "option5", "option5", "option5", "option5", "option5"]

Fig. 3. The same file of the same firmware in the two different environments.

dynamic files. Here, we use the file structure to represent the file to eliminate the inconsistent places caused by the dynamic files.

The DOM tree can be used to extract the structure of the file. Dynamic languages (i.e., JavaScript) often use the DOM tree to describe the file organization and structure. Many webpages use DOM as an interface to describe the combination of style, scripts, and HTML tags for animated documents. The DOM tree provides a representation of the document as a structured tree, where nodes are tags, objects, or scripts. Figure 3 shows two DOM trees of the file in different environments. We observe that two structures are the same if we do not count the deepest node in the DOM tree. The file structure remains stable even though the contents have changed.

We extract elements from local files and generate their DOM trees. For every DOM tree, we use the pre-order traversal to generate its vector. The pre-order traversal can be reversible, so we can recover the tree from the vector. Every element of the vector is the set of all nodes of the DOM tree, starting from the root. Removing the end-node in the DOM tree is equal to eliminating the end content of the vector. Figure 3 shows that two documents are transferred to the vector. After removing end-node "option5", these two files are the same.

E. Firmware Fingerprints

Every item in the final list can be transferred to a vector. We put these vectors together to form a matrix for presenting the fingerprint of the firmware. Figure 4 shows how the local files are transferred to DOM trees and vectors for generating the matrix. One firmware image has one matrix as its fingerprint. Every vector in the matrix has its pair $< request, response >$. If we would like to obtain a local file, we generate the request based on its vector in the matrix. We send it to online embedded devices and receive the response data. We compare the response data and the vector to identify whether the firmware is running on the device. We calculate the similarity

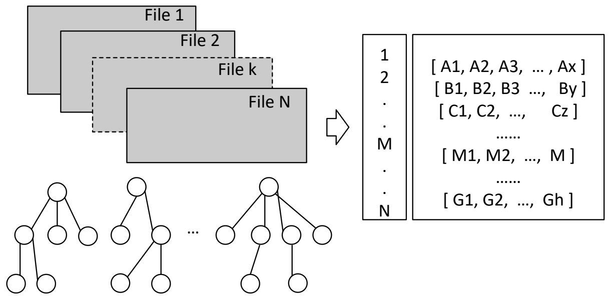


Fig. 4. Firmware fingerprints based on DOM trees of local files.

between the response data and the vector in the matrix based on following rules:

- If the length of the item is equal to the depth of the DOM tree, we remove it because of the inference of dynamic changing. We then transfer the vector to the string to compare it with the response data.
- We use the longest matching state subsequence (LMSS) to present the similarity between the string from the vector and the response data. LMSS is the maximum subsequence common in the two string sequences.
- We use the LMSS average to present the similarity value of the device. For instance, we send 50 request packets to remote devices and receive their responses. We calculate all LMSS values between every response and vectors, and the average is the similarity value of the device.

If the similarity value is larger than the threshold (see Section IV), we identify the information of the firmware running on the device, such as device type, manufacturer, and version.

IV. Real-world Experiments

In this section, we first present the implementation of the prototype system and conduct the real-world experiments to validate the effectiveness of our proposed firmware fingerprinting approach. Then, we use firmware fingerprints to discover embedded devices that are still using outdated-version and vulnerable firmware on the Internet.

A. Implementation

We have implemented a prototype system of firmware fingerprint generation as a self-contained piece of software based on open source libraries. For seven official websites of manufacturers, we have written crawling scripts to download firmware images based on Scrapy [9]. Every binary image would be pipelined to our Python script based on BINWALK [6] for extracting the filesystem and QEMU [8] for simulating the firmware. We use the Natural Language Toolkit [10] to process the file content and Beautiful Soup [11] to extract DOM trees for the files. We transfer the files into vectors according to the pre-order traversal and put vectors together as the matrix for firmware fingerprints.

B. The Similarity of Firmware Filesystems

First, we downloaded 9,716 firmware from seven manufacturers' websites. The device type is either the gateway or

TABLE I
SUMMARY OF FIRMWARE IMAGES AND FILESYSTEMS

Device type	Manufacturer	Firmware versions	Filesystems
Gateways or Routers	Belkin	120	57
	D-Link	2,110	597
	Linksys	78	53
	Netgear	2,063	870
	Zyxel	868	115
	Tomato	3,283	3,281
	TP-Link	1,194	323

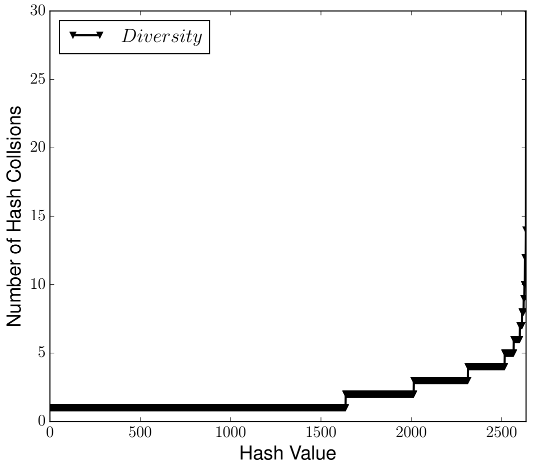


Fig. 5. The distribution of the directory "WWW" of firmware filesystems extracted form Table I.

the router, and they are the mainstream manufacturers in the market, including hundreds of device products with hundreds of versions of firmware. The distribution of firmware images across products and versions is not uniform. For instance, D-Link re-released two products with different hardware but with the same version of firmware. Here, we focus on the firmware version rather than the products because current vulnerabilities occur in the software (firmware) rather in the hardware.

Then, we extract filesystems from the downloaded firmware images. Table I shows the number of filesystems from downloaded firmware images. We are able to extract 5,296 filesystems from the firmware images. Note that only 54.5% of the firmware images can be successfully unpacked into filesystems. It is the reason that incomplete encryption filesystem or unrecognized firmware would lead to the failure of unpacking binary files. For example, TP-Link usually distributes multiple partial images for their firmware, preventing us from obtaining their filesystems. We can generate firmware fingerprints only if their filesystems are capable of being unpacked successfully.

Furthermore, we conducted experiments to validate whether the subtle differences of the filesystem still exist in the directory "WWW". In this directory, the average number of files is 123. The manufacturer Belkin has the largest file number at nearly 3,000, and the vendor Netgear has the smallest file

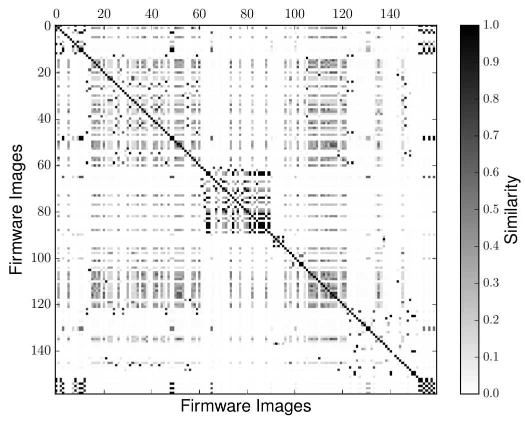


Fig. 6. The similarity matrix of different filesystems of firmware images.

number at only 2. We use 5,296 filesystems (successfully unpacked) from Table I. The file content is extracted, and we use a hash algorithm to calculate its MD5 checksums. Figure 5 shows the distribution of the directory "WWW" of firmware filesystems across different manufacturers and product versions. There are 2,635 unique hash values among those firmware images. Of those, 1,636 firmware images have distinct hash values; 377 firmware images incur one hash collision; and 300 images appear to have two hash collisions. We further observe these firmware images with the hash collision. If the firmware comes from different manufacturers or device types, there are not any collisions at all. The hash collision only happens when the firmware versions are very similar. For instance, four firmware images (D-Link TM-G5240 4.01.B02, 4.0.0B28, 4.01.B01, and 4.00B29) share the same hash value. We find that images with the hash collision are often updated by manufacturers to fix the same function, such as bugs, vulnerabilities, or performance issues. From the defensive perspective, the same vulnerability may occur across these firmware versions with the hash collision. We consider them as one type of firmware image. It is important to note that the hash-value matching is the most stringent standard for firmware classification. If the hash value of filesystems from two firmware image is the same, we cannot distinguish them.

C. Performance

We explore the degree of difference between various firmware images. We calculate the similarity of their firmware fingerprints with the equation:

$$sim(M_i, M_j) = \frac{|M_i \bigcap M_j|}{|M_i \bigcup M_j|},$$

where M is the matrix for presenting the firmware fingerprint; i and j are the firmware images i and j. If the vector in the matrix M_i is equal to the vector in the matrix M_j, we add 1; otherwise, we add zero. $|M_i \bigcap M_j|$ is the sum of equal vectors of the two matrices i and j, and the $|M_i \bigcup M_j|$ is the sum of vectors of two matrices. The similarity degree reaches its highest value at sim score 1 and its worst score at 0. Figure 6

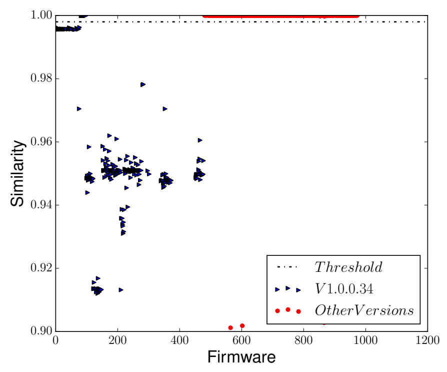


Fig. 7. The similarity threshold value of the firmware Netgear V1.0.0.34.

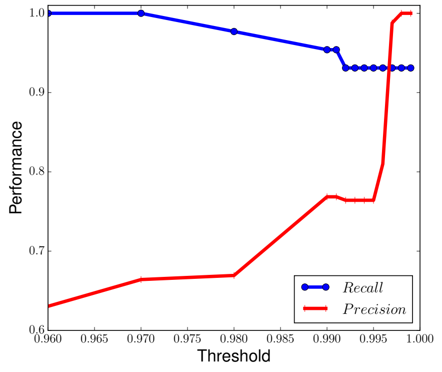


Fig. 8. The precision and recall along with the threshold of the firmware.

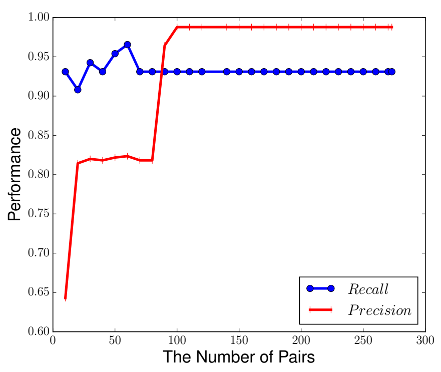


Fig. 9. The precision and recall along with the number of pairs $(request, response)$.

shows the similarity matrix of 160 firmware fingerprints. The deeper the color, the higher the similarity. When the firmware comes from different manufacturers, the similarity is nearly zero, as shown in the color white. When the version numbers are close, the similarity is high in the dark color. We have observed that there are significant differences when the firmware comes from different manufacturers and only slight differences when the images have different versions but the same manufacturer. The honeypot can imitate the firmware on the Internet by producing fake services or webpages on the Internet. Our approach can easily distinguish them because the similarity between the honeypot and truth firmware is lower than the similarity of firmware from different manufacturers.

First, we conducted experiments to recognize manufacturers of the firmware without the version information. Existing device search engines (Shodan [2]) use keywords pickup by manual efforts to identify where the firmware comes from. In our firmware fingerprints, we only randomly select one pair $< request, response >$ from the matrix. The result shows that we can achieve 100% precision and 91.4% recall.

Moreover, we have conducted the experiments to validate the performance of firmware fingerprints at fine-grained levels. Figure 7 shows the distribution of the similarity among different firmware. The red points are the firmware Netgear V1.0.0.34, and the blue points are others. The black dotted line is the separate line between the firmware Netgear V1.0.0.34 and others. We use precision and recall to present the performance.

$$Precision = \frac{TP}{TP + FP}, \qquad Recall = \frac{TP}{TP + FN}$$

where TP is the true positive number, FP is the false positive number, and FN is the false negative number. Figure 8 shows the precision and recall of the firmware along with different threshold values. The X-axis indicates the values of the threshold, and the Y-axis indicates the recognition performance of the firmware. The blue curve is the recall, and the red curve is the precision. We observe that when our threshold is high, the recall decreases, and the number of missing identifications becomes larger. We suggest determining the threshold value according to the requirements of the firmware recognition. Figure 9 shows the precision and recall of the firmware along with the number of pairs $< request, response >$. The X-axis indicates the number of pairs, and the Y-axis indicates the recognition performance of the firmware. We can see that the precision becomes larger when we increase the number of pairs; otherwise, the recall decreases. When the number of pairs $< request, response >$ is close to 75, the recall and precision both arrive at 90%.

D. Firmware Measurement

We would like to use the fingerprints to find out which online embedded devices are still using outdated-version and vulnerable firmware. We use the firmware fingerprints to recognize which host is using the firmware. Fingerprints come from three vendors: D-Link, Beklin and Netgear. D-Link firmware versions are across V1.01 to V4.14, Netgear versions contain V1.0.0.34 to V1.0.0.44, and Beklin versions contain V1.00.06 to V1.00.30. To discover the mixed-version firmware on the Internet, we have deployed the prototype system running on Amazon EC2 with 450Mbps of bandwidth. It is Ubuntu 14.04.2 LTS (GNU/Linux 3.13.0-48-generic x86 64) with two vCPU, 8GB of memory and 450Mbps of bandwidth.

The IPv4 space has nearly 4 billion addresses. For this reason, we cannot directly use dozens of pairs $< request, response >$ to access every IP address. First, we use visible and publicly available IP addresses as the measurement space rather than the ones located behind firewalls or network address translation (NAT). According to the allocated IP space by IANA [12], we choose 3.7 billion addresses for firmware measurement. We only focus on ports such as 80 and 8080, because the directory "WWW" usually provides the response data on those two ports. We reuse the scanner tool ZMap [13] to send a TCP-SYN packet to every IP address. If the host gives a response, we add it to the alive list; otherwise, we drop it out.

After the horizontal scanning, the alive list still has millions of alive IP addresses. As aforementioned, we can identify

TABLE II
REAL DEVICES WITH MIXED-VERSION FIRMWARE ON THE INTERNET

	Version	Number		Version	Number
D-Link	V1.01	25	Netgear	V1.0.0.34	3138
	V 1.06	97		V1.1.1.58	98
	V1.10	29		V1.0.0.44	119
	V1.20	1		V1.0.0.40	19
	V1.21	251			
	V1.40	6	Beklin	V1.00.06	51
	V3.01	2		V1.00.30	9
	V3.11	1		V1.00.08	28
	V3.13	1			
	V4.14	31		Total	3906

which vendor the firmware comes from by using one pair $< request, response >$. Second, we randomly select one packet from the matrix of firmware fingerprints to identify its manufacturer. We obtain a smaller list (nearly hundred thousand) where each host uses either D-Link, Beklin or Netgear firmware. Then, we use the remaining pair $< request, response >$ and send the request to every host on this list. If the similarity is larger than the threshold, we identify that a particular version of firmware is running over this IP address. Table II shows online embedded devices running those different versions of the firmware in the cyberspace. We can see that there is no uniform distribution for firmware versions. The number of the latest version only occupies a small percentage compared with other firmware. It conforms to our previous understanding that many embedded devices are still using outdated-version firmware even though their manufacturers have distributed update patches.

Finally, we use proof-of-concept exploits to validate whether those online embedded devices are vulnerable in Table II. The fundamental question of "which devices are still vulnerable" dictates that we should identify the online devices by firmware version. For ethic considerations, we do not use the exploitation scripts to validate whether the devices are vulnerable. We have crawled the vulnerabilities from the CVE website [4]. For each item in vulnerability list, we obtain their common weakness enumeration specification (CWE) from the National Vulnerability Database [14]. The CWE provides information about software security vulnerabilities according to the system architecture. We compare the version information in Table II with the CWE item to determine whether the online embedded device is vulnerable. If it does match, the online embedded device running the firmware is vulnerable. We found that many embedded devices are vulnerable on the Internet. Netgear firmware V1.0.0.34 has been used in 3138 embedded devices, which has Cross-Site Scripting (CVE-2013-3069) and SMB Symlink Traversa (CVE-2013-3073). Our findings show that embedded devices are still vulnerable because of outdated versions of the firmware.

V. RELATED WORK

Due to the growing popularity of exposing embedded devices on the Internet, several related works have analyzed firmware, fingerprinting technology, and online embedded devices.

Firmware. There are two conventional methods for analyzing firmware images, including static analysis and dynamic analysis. Static analysis is performed over the source code and detects underlying vulnerabilities without executing the firmware. Costin et al. [15] collected 32 thousand firmware images and unpacked them to run simple static analysis tasks over those files within the filesystem. The limitation is that the firmware in the runtime environment is different from the documents in the filesystems. Dynamic analysis is performed over the firmware on a real device or virtual host. Zaddach et al. [16] presented AVATAR to simulate peripherals of embedded devices for the firmware images and performed dynamic analysis on the virtual hosts. Chen et al. [17] presented FIRMADYNE to reduce costs and time for simulating firmware images over the hardware of devices for a large-scale dynamic analysis. In contrast, our work utilizes the static analysis to obtain filesystems from firmware images and dynamic analysis to eliminate the inconsistencies between firmware fingerprints and local files in the filesystems. Cui et al. [18] focused on a particular embedded device (HP LaserJet printer) for exploiting a vulnerability caused by the firmware update. Our work is about generating firmware fingerprints and using them to quantify the real-world impacts of security issues caused by mixed versions.

Fingerprint technology. Fingerprinting is a technique for identifying the operating system (OS), applications, or network services. State-of-the-art tools (e.g., Nmap [19]) usually utilize the differences between TCP/IP implementations to identify OS versions. They also use service banners in the application-level protocols to find applications and network services. Feng et.al [20] proposed to utilized the banner of industrial control protocols to discover cyber-physical system devices on the Internet. Li et.al [21] extracted the graphic user interface (GUI) from the HTTP protocol banners and used the GUI as the unique fingerprint of surveillance devices. However, these fingerprinting techniques are unable to recognize the firmware of embedded devices in a fine-grained manner on the Internet. Much of the firmware of embedded devices uses Linux systems and supports more network protocols, making those existing tools ineffective for fingerprinting the firmware. The clock skew [22], [23] was proposed for fingerprinting devices by exploiting the differences in time synchronization based on network time protocol [24]. Many embedded devices run the same version of firmware, and the distribution of firmware images across products is not uniform. By contrast, the vulnerability is associated with the version of firmware rather than individual devices.

Embedded devices. With the increasing prevalence of the Internet of Things [25], [26], embedded devices are connecting to the Internet. Cui et al. [27] spent four months finding

540 thousand online devices without access credentials based on scanner tools. Shodan [2] and Censys [3], [28] are able to discover devices and web services on the Internet and present the graphical user interface to the public. They provide a global view of online devices for researchers to identify distributed systems and potential security issues. ShoVAT [29] uses Shodan to assess the vulnerability of embedded devices. However, attackers and malicious users can also use the easy-to-use tools to find online devices before attacking. Even if these device search engines forbid this illegal use, attackers can still find online embedded devices through ZMap [13] and Masscan [30], which can scan the whole IPv4 space in a short period of time. The large-scale measurement of embedded devices has avoided the direct analysis of firmware images. The trigger condition of the vulnerability is the version of firmware images in the platform. Discovering firmware with vulnerabilities is a pre-requisite to secure those online devices.

VI. Conclusion

Online embedded devices play a crucial role in our daily lives, but device search engines and network scanning tools can expose them to the public. These embedded devices are using mixed versions of firmware on the Internet. Many vulnerabilities are triggered in a particular firmware version of embedded devices, raising serious security concerns. In this paper, we proposed a novel approach for accurately generating firmware fingerprints at the fine-grained level. The core of our approach is to use the filesystem of firmware as the signature to recognize the firmware versions. We used the natural language processing technology and document object model to obtain the firmware fingerprint. Based on firmware fingerprints, we utilized a heuristic rule to compare the similarity between the matrix and HTTP pairs to identify the firmware version. We implemented a prototype of our approach and evaluated its effectiveness through real-world experiments. The results show that our automatically generated fingerprints can achieve 90% recall and 91% precision. Furthermore, we used firmware fingerprints to quantify the real-world impacts of mixed versions of firmware on the Internet.

VII. Acknowledgments

This work was supported in part by the National Key Research and Development Program (Grant 2017YFB0802804), the National Natural Science Foundation of China (Grant 61602029, 61702504), National Defense Basic Scientific Research (Grant JCKY2016602B001), and the Fundamental Theory and Cutting Edge Technology Research Program of Institute of Information Engineering, CAS and SKLOIS (No. Y7Z0361104), as well as the U.S. ONR grant N00014-17-1-2485.

References

[1] Security Notification - HTTP buffer overflow vulnerability in Hikvision NVRs devices. [Online]. Available: http://www.hikvision.com/En/Press-Release-details_435_i1023.html

[2] Shodan. The search engine for Internet-connected devices. [Online]. Available: https://www.shodan.io/

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

[4] CVE, Common Vulnerabilities and Exposures. [Online]. Available: http://cve.mitre.org/

[5] Amazon. Amazon Elastic Compute Cloud (EC2). [Online]. Available: https://aws.amazon.com/ec2/

[6] Binwalk, the tool for analyzing, reverse engineering, and extracting firmware images. [Online]. Available: http://binwalk.org/

[7] RFC 2616, Hypertext Transfer Protocol HTTP/1.1 . [Online]. Available: http://www.rfc-base.org/rfc-2616.html

[8] F. Bellard, "QEMU, a fast and portable dynamic translator." in *Proceedings of the Annual Conference on USENIX Annual Technical Conference*, ser. ATEC '05, 2005, pp. 41–46.

[9] Scrapy, A fast and powerful scraping and web crawling framework. [Online]. Available: https://scrapy.org

[10] Natural language toolkit. [Online]. Available: http://www.nltk.org/

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

[12] IANA. Internet Assigned Numbers Authority (IANA). [Online]. Available: http://www.iana.org/

[13] Z. Durumeric, E. Wustrow, and J. A. Halderman, "ZMap: Fast Internet-wide Scanning and Its Security Applications." in *22th USENIX Security Symposium*, 2013, pp. 605–620.

[14] NVD-CPE, Computer Security Resouce Center for Official Common Platform Enumeration Dictionary. [Online]. Available: https://nvd.nist.gov/products/cpe

[15] A. Costin, J. Zaddach, A. Francillon, D. Balzarotti, and S. Antipolis, "A Large-Scale Analysis of the Security of Embedded Firmwares." in *23th USENIX Security Symposium*, 2014, pp. 95–110.

[16] J. Zaddach, L. Bruno, A. Francillon, and D. Balzarotti, "AVATAR: A Framework to Support Dynamic Security Analysis of Embedded Systems' Firmwares." in *Symposium on Network and Distributed System Security (NDSS)*, 2014.

[17] D. D. Chen, M. Egele, M. Woo, and D. Brumley, "Towards Automated Dynamic Analysis for Linux-based Embedded Firmware," in *Symposium on Network and Distributed System Security (NDSS)*, 2016.

[18] A. Cui, M. Costello, and S. J. Stolfo, "When Firmware Modifications Attack: A Case Study of Embedded Exploitation." in *Symposium on Network and Distributed System Security (NDSS)*, 2013.

[19] Nmap, network security scanner tool. [Online]. Available: https://nmap.org/

[20] X. Feng, Q. Li, H. Wang, and L. Sun, "Characterizing industrial control system devices on the internet," in *IEEE 24th International Conference on Network Protocols (ICNP)*, 2016.

[21] Q. Li, X. Feng, H. Wang, and L. Sun, "Automatically Discovering Surveillance Devices in the Cyberspace," in *the 8th ACM on Multimedia Systems Conference (MMSys)*, 2017.

[22] T. Kohno, A. Broido, and K. C. Claffy, "Remote Physical Device Fingerprinting," *IEEE Transactions on Dependable and Secure Computing*, vol. 2, no. 2, pp. 93–108, April 2005.

[23] S. Zander and S. J. Murdoch, "An Improved Clock-skew Measurement Technique for Revealing Hidden Services," in *Proceedings of the 17th USENIX Security Symposium*, 2008.

[24] NTP: The Network Time Protocol Protocol for synchronize the clocks of computers over a network. [Online]. Available: http://www.ntp.org/

[25] E. Lee, "Cyber Physical Systems: Design challenges," in *the 11th IEEE International Symposium on Object Oriented Real-Time Distributed Computing (ISORC)*, 2008.

[26] L. Atzori, A. Iera, and G. Morabito, "The Internet of Things: A survey," *Computer networks*, vol. 54, no. 15, pp. 2787–2805, 2010.

[27] A. Cui and S. J. Stolfo, "A Quantitative Analysis of the Insecurity of Embedded Network Devices: Results of a Wide-area Scan," in *Proceedings of the 26th Annual Computer Security Applications Conference (ACSAC)*, 2010.

[28] Z. Durumeric, D. Adrian, A. Mirian, M. Bailey, and J. A. Halderman, "A Search Engine Backed by Internet-Wide Scanning," in *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 2015.

[29] B. Genge and C. Enăchescu, "ShoVAT: Shodan-based Vulnerability Assessment Tool for Internet-facing Services," *Security and Communication Networks*, 2015.

[30] Masscan, Network Scanner tool for scanning Internet port. [Online]. Available: https://github.com/robertdavidgraham/masscan