Understanding Security Risks of Embedded Devices Through Fine-Grained Firmware Fingerprinting

Qiang Li, Dawei Tan, Xin Ge, Haining Wang, *Fellow, IEEE*, Zhi Li, and Jiqiang Liu, *Senior Member, IEEE*

Abstract—An increasing number of embedded devices are connecting to the Internet, ranging from cameras, routers to printers, while an adversary can exploit security flaws already known to compromise those devices. Security patches are usually associated with the device firmware, which relies on the device vendors and products. Due to compatibility and release-time issues, many embedded devices are still using outdated firmware with known vulnerabilities or flaws. In this article, we conduct a systematic study on device vulnerabilities by leveraging firmware fingerprints. Specifically, we use a web crawler to gather 9,716 firmware images from official websites of device vendors, and 347,685 security reports scattered across data archives, blogs, and forums. We propose to generate fine-grained fingerprints based on the subtle differences between the filesystems of various firmware images. Furthermore, machine learning algorithms and regex are used to identify device vulnerabilities and corresponding device firmware fingerprints. We perform real-world experiments to validate the performance of the firmware fingerprint, which yields high accuracy of 91% precision and 90% recall. We reveal that 6,898 reports have the firmware and related vulnerability information, and there are more than 10% of firmware vulnerabilities without any patches or solutions for mitigating underlying security risks.

Index Terms—Firmware, fingerprinting, embedded device, vulnerability

1 Introduction

Nowadays, 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 usually accessible and visible through IP addresses on the Internet, which, however, brings in a new security concern that adversaries could compromise them by exploiting the already known security vulnerabilities. For instance, Mirai exploited hundreds of thousands of webcams and DVR devices to launch a Distributed Denial-of-Service (DDoS) attack, overwhelming Krebs on Security, OVH, and Dyn services [1]. More importantly, one pervasive but easily overlooked fact is that those embedded devices are reported to be hacked all through known vulnerabilities.

- *Qiang Li, Dawei Tan, Xin Ge, and Jiqiang Liu are with the School of Computer and Information Technology, Beijing Jiaotong University, Beijing 100044, China. E-mail: qiangcas@gmail.com, {18140036, 18120470, jqliu}@bjtu.edu.cn.*
- *Haining Wang is with the Department of Electrical and Computer Engineering, Virginia Tech, Blacksburg, VA 24061 USA. E-mail: hnw@udel.edu.*
- *Zhi Li is with the Institute of Information Engineering, Chinese Academy of Sciences (CAS), Beijing 100093, China, and also with the School of Cyber Security, University of Chinese Academy of Sciences, Beijing 100049, China. E-mail: lizhi@iie.ac.cn.*

Manuscript received 7 Apr. 2020; revised 9 Feb. 2021; accepted 3 Oct. 2021. Date of publication 14 Oct. 2021; date of current version 11 Nov. 2022.
The work was supported in part by the National Key R&D Program of China under Grant 2018YFB0803402, in part by the National Natural Science Foundation of Chinaunder under Grant 61972024, and in part by the National Key R&D Program of China under Grant 2020YFB2103802.
(Corresponding authors: Zhi Li and Qiang Li.)
Digital Object Identifier no. 10.1109/TDSC.2021.3119970

Firmware is a type of software semi-permanently installed in the hardware of embedded devices, providing necessary control, monitoring, and data manipulation, referred to as "software for hardware". For embedded devices, the device firmware is the common platform of the vulnerability, called official Common Platform Enumeration (CPE) [2]. The same vulnerability is not triggered when running on different firmware. 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 [3]. However, finding and understanding vulnerable embedded devices in the wild are by no means trivial due to the challenges in two aspects: (1) there lacks firmware knowledge for embedded devices and (2) known flaws and relevant patches are scattering on the Internet.

In this paper, we present a systematic study on vulnerabilities of embedded devices through their firmware fingerprints. To obtain firmware information of online embedded devices, we propose a novel approach for generating the fine-grained firmware fingerprints. Fine-grained fingerprints indicate more detailed information about a device's firmware, including vendor and product version. Based on the firmware information, we further extract their vulnerabilities/flaws and corresponding patches from different third-party sources, e.g., security forums and databases. In other words, the firmware information is very useful to trace out vulnerable embedded devices available on the Internet. Thus, our study can quantify the security risks of device vulnerabilities on the Internet.

The fine-grained fingerprint generation is to utilize an intuitive insight 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. Primarily, 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 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.

For device vulnerabilities/flaws, we use the web crawler to collect the relevant information from various online security sources and filter out irrelevant content on webpages. Online sources include security forums (seclist.org/fulldisclosure), mailing lists (seclist.org), technical blogs (blog.osvdb.org), and data archive websites (nvd.nist.gov). More specifically, we utilize machine learning algorithms to determine whether an online document is related to security flaws/vulnerabilities of embedded devices. The regex is used to extract the information about device firmware. Furthermore, we manually analyze the patches of those vulnerabilities to determine whether they are fixed for embedded devices.

To validate firmware fingerprints, we implement a prototype system and conduct real-world experiments. Our results show that our firmware fingerprints can achieve a 91% precision and a 90% recall. To understand device vulnerability, we download 9,716 firmware images from seven manufacturers' websites and 444,923 security reports scattered across bug-reporting blogs, forums, mailing lists, and National Vulnerability Database (NVD). We find that many firmware updates are associated with the security patches and over 10% of the firmware flaws have no patches available when they are made public, some of which are not released by their manufacturers even today.

Overall, our major contributions [1] are summarized as follows:

- We propose a new approach to generating firmware fingerprints at the fine-grained level based on the filesystems' differences, which can remotely detect closely related firmware versions.
- We analyze 9,716 firmware images from vendors' official websites and 444,923 security reports. We quantitatively assess device vulnerabilities and their security risks based on firmware fingerprints.

1. The preliminary version of this article [4] was published in the Proceedings of INFOCOM 2018.

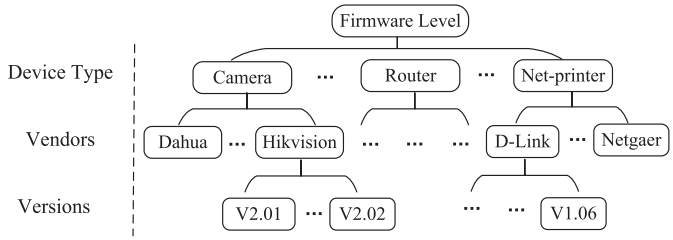


Fig. 1. The fine-grained levels of firmware.

- We implement a prototype and conduct real-world experiments for validating its effectiveness. The results show that our system can accurately recognize the firmware of online embedded devices and automatically analyze firmware flaws through security reports.

The remainder of the paper is structured as follows. Section 2 describes the background of firmware recognition and vulnerability discovery. Section 3 presents the automatic fingerprint generation of online embedded devices. Section 4 presents the firmware-vulnerability extraction from online security reports. Section 5 describes the implementation and real-world experiments. Section 6 details the qualification of vulnerable devices across the Internet. Section 7 surveys related work, and finally, Section 8 concludes.

2 Background

In this section, we first describe the background of the firmware of embedded devices on the Internet. Then, we present relevant device vulnerabilities associated with firmware.

2.1 Firmware

Firmware is the native software embedded in the hardware of devices, which is retained in non-volatile memory, such as ROM or EPROM. If owners of embedded devices need to fix bugs, add features, or improve performance, vendor developers must update the corresponding source code of firmware. After that, the updated source code needs to be re-compiled into a binary image, called firmware. Vendors distribute the firmware with a different version in their technical support websites, and device owners will use them to update their devices. Thus, the change between mixed-version firmware images is determined by subtle differences in source code.

Firmware is vendor- and chipset-specific, indicating that devices from different vendors and product versions have different firmware. The firmware category has three levels, which are shown in Fig. 1 with respect to the device type, manufacturers, and product versions. First, the firmware runs at different device types, like webcams or routers. The device type is the most coarse-grained level for the firmware category and the easiest case to recognize on the Internet. 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 device vendors would release firmware images to install on their products and systems. The challenge of identifying this category is that there are many vendors/manufacturers, and it is hard to keep the fingerprinting updated with the addition of numerous new devices. Third,

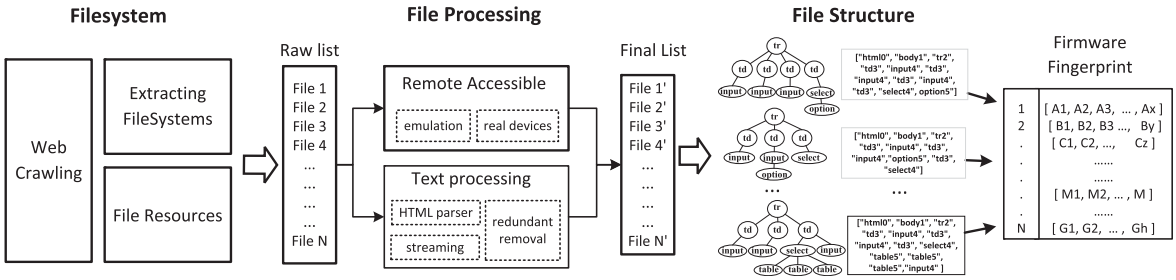


Fig. 2. The overview of system design for generating fine-grained fingerprints of firmware.

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. As far as we know, there is no previous work that can recognize the firmware at such a fine-grained level.

Recognizing 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 vendor. 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 without compromising them. 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 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.

2.2 Device Vulnerability

Typically, a device vulnerability is discovered by professionals (i.e., a vendor, hacker, or a researcher). Prior works [5], [6], [7] detect vulnerabilities in embedded devices by using the static analysis and the dynamic analysis. Once discovered, the vulnerability information may be disclosed on the Internet through different sources such as security forums and official websites. The examples of public sources include SecLists, OSVDB blog, Coresecurity forum, and NVD. Those online sources have gathered a collection of different types of security reports updated by individual users or vendors.

For embedded device vulnerabilities, vendors should distribute the security patches to users for updating the device firmware. The embedded device needs to update its firmware to the latest version, which is more secure than the outdated version. From the defense perspective, updating to the latest version is the preferred solution, specifically for fixing a security flaw in the outdated version. However, many embedded devices are equipped with the outdated/unsecured firmware in the wild. 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. Third, updating the 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 then they install the firmware into the ROM to reprogram integrated chip circuits of embedded devices. Therefore, many embedded devices are still using the outdated version of firmware, even if the update has been distributed.

In this work, we use firmware fingerprints to discover and identify embedded device tags, including the device type, vendor name, and product version. We further utilize online sources to understand the security risks of vulnerabilities associated with the firmware of embedded devices in the wild.

3 Fingerprinting Firmware

In this section, we present the firmware fingerprint generation approach. 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 to discover online embedded devices on the Internet. Other information (e.g., the hash, relevant metadata, versions) can only be obtained locally, not remotely. Thus, we cannot use such information to detect the firmware version for embedded devices in a remote manner.

Architecture. Fig. 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 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. Below, we present the details of the system design for fine-grained firmware fingerprints.

3.1 Filesystem

Collecting the firmware filesystem is the first and largely independent component for generating fingerprints.

Firmware Images. 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. Because each website has different HTML templates, we manually design download scripts to obtain firmware images from 7 official websites of manufacturers through either the website or FTP server. For every firmware image, 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. We use BINWALK [8] to unpack binary files of firmware images. BINWALK is a third-party tool to analyze and reverse-engineer binary files, which extract file systems through matching file patterns of the firmware. Once matched, Binwalk would unpack the filesystem through invoking the extraction tools [9], [10], [11]. When a filesystem is extracted, we would use file resources to generate firmware fingerprints. Note that 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. In general, most firmware filesystems have more than 1,000 files. Then, we store all files for each firmware image.

File List. 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 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. We use the location, filename, and extension to extract files from this directory. The location is the relative path similar to the directory for files, which is the parameter URL in the HTTP method. We use the backslash to concatenate the directory name with the sub-directory name. For each item in the raw list, we use the "location/filename" to represent the file. Furthermore, 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 for generating the firmware fingerprint.

3.2 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 Accessibility. 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. We use the GET method and set URL as a concatenation of the directory and filename. Firmware images are running in either the emulation environment or real devices.

We propose using the system emulator QEMU [12] 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, 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. 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.

3.3 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. Fig. 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

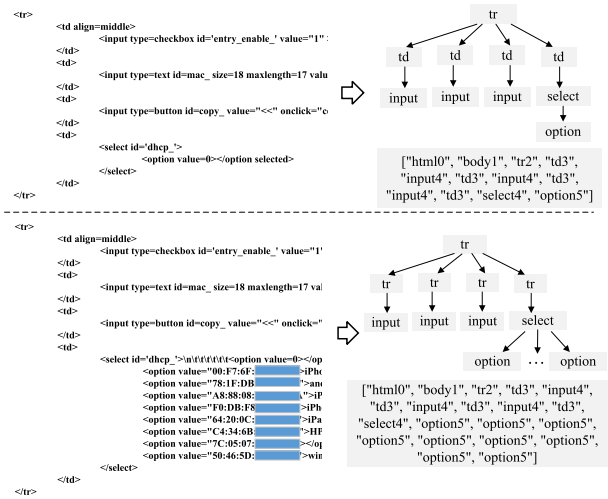


Fig. 3. The same file of the same firmware in the two different environments.

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 (Fig. 3, upper). When the real device is running the firmware, the field "option" shows MAC addresses and names of eight attached devices (Fig. 3, lower). We can't directly use the response data to represent the local file because of the 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. Fig. 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. Fig. 3 shows that two documents are transferred to the vector. After removing end-node "option5", these two files are the same.

3.4 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. Fig. 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 $< request, response >$

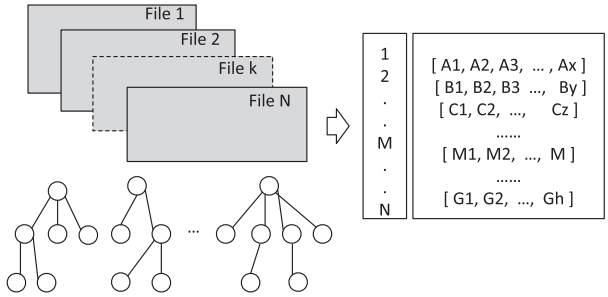


Fig. 4. Firmware fingerprints based on DOM trees of local files.

. 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 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 5), we identify the information of the firmware running on the device, such as device type, vendor, and version.

4 Device Vulnerability Detection

In this section, we present the methodologies designed to (1) collect vulnerability documents from online sources and (2) identify relevant firmware patches from those reports.

Fig. 5 presents the overview of firmware-vulnerability information extraction, which consists of two main components: security report collection (SRC) and firmware-patch identification (FPI). We leverage different online resources, including technique blogs, security forums, mail-lists, and data achieves, for report collection. The SRC uses the web crawler to collect documents from those third-party sources on the Internet. Specifically, the SRC utilizes HTML templates to filter out irrelevant content and remove duplicated webpages if they provide similar content. The FPI uses machine learning algorithms to determine whether a webpage contains vulnerability information. The FPI utilizes the regex to extract firmware information from those security reports, including device types, vendors, versions, and vulnerabilities. We manually analyze the patch status of every firmware vulnerability in those security reports. Below we elaborate on the details.

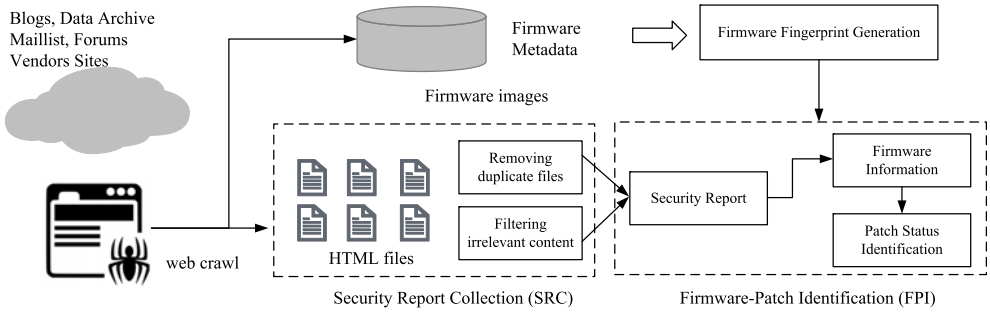


Fig. 5. The overview of firmware vulnerability discovery from various data sources.

4.1 Security Report Collection

Nowadays, there are many technical articles about device vulnerabilities from various online sources. We leverage 22 popular online sources to obtain technical articles, as listed in Table 1, which are security forums (seclist.org/fulldisclosure), mailing lists (seclist.org), technical blogs (blog.osvdb.org), and data archive websites (packetstromsecurity.com). Because each website has different HTML templates and configurations, our web crawler has a corresponding scraping script to scrape webpages. Specifically, we perform breadth-first searching on websites to scrape webpages until no new links can be found. For every webpage, we use the URL as its identifier and store its HTML document. Note that we also download the firmware metadata on the official vendor websites and manually extract the relevant information, including device type, vendor name, and product version.

Data Preprocessing. There are irrelevant webpages in the dataset we collected. Many irrelevant terms also appear in those webpages, such as advertisements, pictures, dynamical scripts, menu bar, and navigation bar. To automatically remove the irrelevant content in a report, we utilize a unique observation: irrelevant content uses particular HTML tags for differing textual content, such as $<\text{img}>$, $<\text{input}>$, $<\text{script}>$, and $<\text{iframe}>$. Specifically, we keep the main content associated with HTML tags $<\text{div id="body"}>$ and remove the irrelevant content of each webpage. The SRC generates a report to represent its corresponding webpage.

Removing Duplicated Reports. Online data sources may contain similar content with one another due to cross-reference and reproduction. For example, the CVE webage [2] and the CISCO webpage [3] refer to the same vulnerability content, but their URL identifiers are different. To automatically remove duplicate content, we leverage the following heuristic rule: if the similarity degree of two documents is large enough, the duplicate one will be removed. We use the cosine distance to calculate the similarity degree.

$$cos(A, B) = \frac{A * B}{|A| * |B|} = \frac{\Sigma_{i=1}^{n} A_i * B_i}{\sqrt{\Sigma_{i=1}^{n}(A_i^2)} * \sqrt{\Sigma_{i=1}^{n}(B_i^2)}},$$

where A and B represent two documents from our dataset. Instead of directly using a word in the document, we use "word embeddings" to represent every word in the document. Word2vec [13], [14] uses two-layer neural networks to learn the linguistic context of words in the large corpus. Every word is assigned to a corresponding vector in the vector space, and similar words share common context with a closer distance. In the word vector, the window size refers to the distance between words, similar to the semantic context. For instance, a window size of 3 represents 3 words behind and 3 words ahead (6 total). In practice, when the similarity value is larger than 0.97, there is a high probability that two documents have the same content.

4.2 Firmware-Patch Identification

Aforementioned, not all webpages are related to the vulnerabilities of firmware. We first use the machine learning algorithm to determine whether a report has vulnerability information. Then, we use the regex to extract the relevant device information, including device type, vendor name, and product version.

Vulnerability Classification. Some webpages might describe security/attacks and malicious activity analysis, rather than vulnerability information. The FPI module can derive the classification model about those webpages. The textual document is the input of the classification model, and the output is the bi-classifier of a report. Note that we also use the word2vec [13], [14] to represent those textual reports. Specifically, we use the Skip-gram model to generate word vectors for representing a textual document. The softmax function is used to calculate the probability of the classification result. Our model is a binary classifier ($y = 1$, $y = 0$), the softmax function normalizes the class tags into a probability distribution. We use the largest probability of $y_{1,0}$ to represent the classification result. Given a report, the classifier can decide

TABLE 1
The Overview of the Websites

Category	Website
Blogs	attrition.org, darkode.me, packetstormsecurity.com cgisecurity.com, coresecurity.com, darkreading.com helpnetsecurity.com, secureworks.com, grc.com blog.vulners.com, blog.osvdb.org, securiteam.com
Forums	blackhatworld.com, isc.sans.edu nulled.to, secuniaresearch.flexerasoftware.com
Maillist	seclists.org, securityfocus.com
Data Archive	ics-cert.us-cert.gov, exploit-db.com, nvd.nist.gov

2. The webpage about CVE-2019-1663 in mitre.org: https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2019-1663.

3. The webpage about CVE-2019-1663 in vulners.com: https://vulners.com/cisco/CISCO-SA-20190227-RMI-CMD-EX.

TABLE 2
The Device-Related Terms

Category	Terms
Device Type	Router, Switch, Gateway, Camera, Scanner, Modem IPcam, Printer, NVS, Copier, NVR, Video Server Netcam, Firewall, Video Recorder, DVR, Rackstation Diskstation, Video Encoder, Access Point
Device Vendor	Cisco, D-Link, TP-Link, Moxa, Foscam, Lexmark Linksys, Tenda, Buffalo, Belkin, Dahua, Netgear Huawei, WatchGuard, SonicWall, QNAP, Hikvision Honeywell, TRENDnet, Polycom, EDIMAX Juniper, AVTECH, GeoVision, AirLink101, Verizon Axis, Advantech, Siemens, Zyxel, ZTE, ASUS
Product	[A-Za-z0-9_.-]*

whether it is vulnerability-related or not based on its textual content. Further, we extract vulnerability information from those security reports. We use three categories to extract the relevant vulnerability terms, including vulnerability types from the Common Weakness Enumeration, corresponding common acronyms, i.e., CRLF, and buffer overflow.

Extracting Relevant Firmware Information. The relevant firmware information mainly consists of device type, vendor name, and product version. For instance, a relevant term is *"Cisco RV130W Router"*, where Cisco is the vendor name, and RV130W is the product version. We utilize regular expression to identify those relevant entities. Table 2 presents contextual textual terms about embedded device information. For device types, we use 20 keyword terms to identify those entities in the report, including printer, router, and camera. Typically, those types belong to consumer-oriented devices in the market. For device vendors, we choose 32 device vendors to represent mainstream manufacturers. First, those vendors, as mentioned before, usually provide a technical support page on their official websites for downloading firmware and relevant detailed descriptions. Our fingerprint technique generates those firmware fingerprints for embedded devices. Second, those vendors are well-known manufacturers with a large market share.

For product names, there are numerous product names about embedded devices. We use the combination of letters and numbers (sometimes containing "-") to represent product names. The regular expression is used to extract those entities, as shown in Table 2. However, the regex would introduce a large number of false positives. For instance, "RV130W" is a product name, and "R1512" is a version name. To reduce the number of false positives, we use the search engines to check whether an entity belongs to product names. We combine the vendor name and product name as a search query and search it in commercial search engines (e.g., Google and Bing). For instance, *(router, Cisco, RV130W)* is formatted as "https://www.google.com/search?q=router+Cisco+RV130W". If the product name is extracted, we will provide a term tuple *(device type, vendor, product name)*.

Identifying Patch Status. Those security reports often describe the patch status about embedded devices. Some reports might claim that there is no patch for the firmware; other reports might claim that vulnerability does have a patch. We provide an analysis of the patch status of firmware vulnerabilities from online sources. Also, we gather publicly disclosed vulnerabilities from the National Vulnerability Database (NVD) to identify their patches. We manually divide those patches of embedded devices into three categories, which are described as follows:

1) *Patch*. If the device vendors or online sources (listed in Table 1) claim a patch for the vulnerability, we believe that the firmware vulnerability is patched. We further manually inspect whether the reference URL provides the technical support for the firmware.
2) *No-Patch & Solution.*Although there exists no patch for the firmware vulnerability, the device vendors or online sources provide the solution to migrate the impact of the vulnerability. CVE-2016-8726 can be mitigated, in terms of the risk of exploitation, by disabling the web application for the Access Point device with the firmware 1.1 version.
3) *No-Patch & No-Solution.*We find that there is neither patch nor solution for this type of firmware vulnerabilities.

5 Fingerprinting Evaluation

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.

5.1 Implementation

We implement a prototype of firmware fingerprint generation as a self-contained piece of software based on open source libraries. For seven official websites of manufacturers, we write crawling scripts based on Scrapy [15] to download firmware images. Every binary image is pipelined to our Python script that is written based on BINWALK [8] and QEMU [12] for extracting the filesystem and simulating the firmware, respectively. We use the Natural Language Toolkit [16] to process the file content and Beautiful Soup [17] to extract DOM trees for the files. We transfer the files into vectors according to the pre-order traversal and organize vectors together as the matrix for firmware fingerprinting.

The BINWALK and QEMU are necessary tools for generating firmware fingerprints of embedded devices. It is possible that we cannot obtain firmware fingerprints due to the failure of BINWALK or QEMU. We manually analyze the architecture and OS information of 9,716 firmware versions for embedded devices. Their distributions are shown in Figs. 6 and 7. The MIPS and ARM are the two most popular architectures in firmware images, with 72.54% and 18.99%, respectively. For OS, we can see that Linux has 46.43%, VxWorks has 1.88%, WinCE has 1.34%, and eCos has 0.37%. Our work focuses on Linux-based and MIPS/ARM for analyzing firmware through BINWALK and QEMU. If the QEMU fails to emulate firmware, we purchase corresponding products on the market and use them to generate firmware fingerprints. The reason is that the products of embedded devices include the firmware version information. Thus, we can still obtain the firmware fingerprints of embedded devices with the cost of finance.

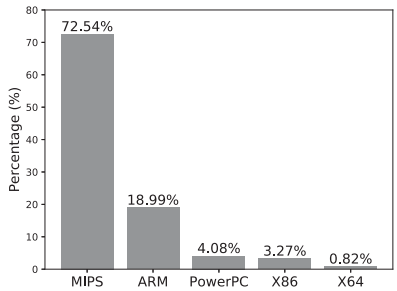


Fig. 6. The architecture distribution of the firmware.

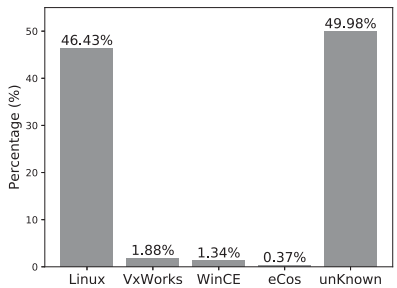


Fig. 7. The OS distribution of the firmware.

5.2 The Similarity of Firmware Filesystems

First, we downloaded 9,716 firmware from seven manufacturers' websites. The device type is either the gateway or 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 3 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 vendor Belkin has the largest file number at nearly 3,000, and the vendor Netgear has the smallest file number at only 2. We use 5,296 filesystems (successfully unpacked) from Table 3. The file content is extracted, and we use a hash algorithm to calculate its MD5 checksums. Fig. 8 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;

TABLE 3
Summary of Firmware Images and Filesystems

Device type	Vendor	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

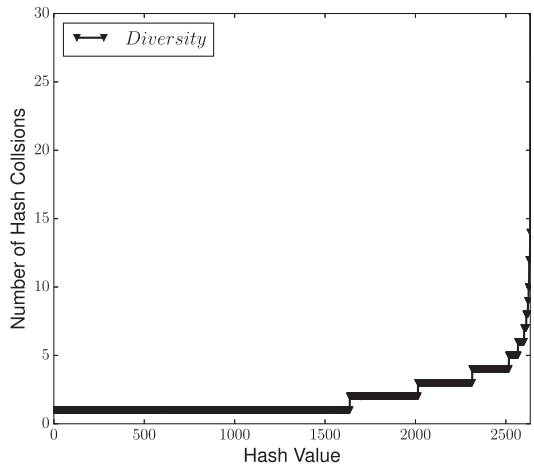


Fig. 8. The distribution of the directory "WWW" of firmware filesystems extracted form Table 3.

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 values of filesystems from two firmware images are the same, we cannot distinguish them. The only solution is to use the binary code similarity match detection, such as [18], [19]. Note that the code similarity match detection is used for offline detection, not online detection.

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

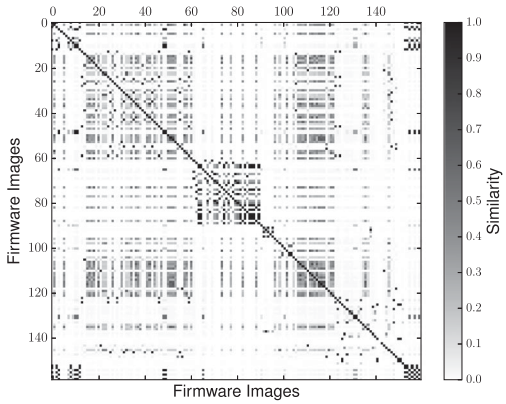


Fig. 9. The similarity matrix of different filesystems of firmware images.

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. Fig. 9 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.

5.3 Performance

There are two kinds of approaches to recognize embedded devices on the Internet: rule-based and model-based. The rule-based approach is to use the keywords of the banner of remote hosts in application-layer protocols. For instance, Nmap [20], Ztag [21], and ARE [22] belong to this approach for identifying embedded devices. However, many embedded devices do not feed back the banner with the firmware version information. Without keywords of the banner, those approaches cannot recognize the firmware versions of embedded devices. The model-based approach is to use machine learning to build the classification model of embedded devices. A practical challenge is the lack of the training datasets for the firmware versions of embedded devices. Another technical challenge is that the closely related firmware versions of embedded devices share the same web services or telnet services, e.g., Netgear V1.0.0.34 and V1.0.0.33. Therefore, model-based approaches have a poor performance for recognizing firmware versions of embedded devices.

First, we conducted experiments to recognize manufacturers of the firmware without the version information. Existing device search engines (Shodan [23]) 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.

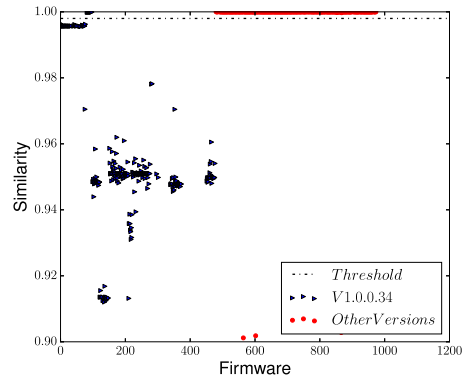


Fig. 10. The Similarity Threshold Value of the Firmware Netgear V1.0.0.34

Moreover, we have conducted the experiments to validate the performance of firmware fingerprints at fine-grained levels. Fig. 10 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. Fig. 11 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. Fig. 12 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%.

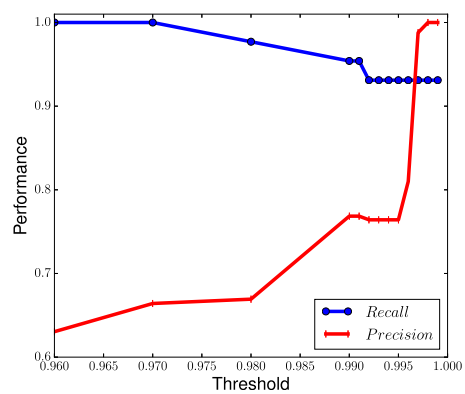


Fig. 11. The precision and recall along with the threshold of the firmware.

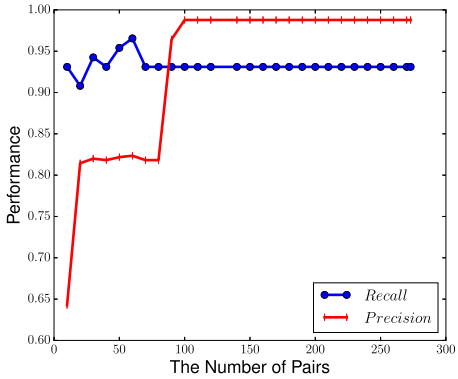


Fig. 12. The precision and recall along with the number of pairs $(request, response)$.

TABLE 4
Real Devices With Mixed-Version Firmware on the Internet

	Version	Number		Version	Number
D-Link	V1.01	25	Netgear	V1.0.0.34	3,138
	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	3,906

5.4 Measurement in the Wild

Based on the fingerprints, we attempt to uncover which online embedded devices are still using outdated-version and vulnerable firmware. Our measurement is focused on three popular vendors, D-Link, Beklin, and Netgear, as their firmware also has security vulnerabilities. We generate firmware fingerprints of the three vendors. D-Link firmware versions range from V1.01 to V4.14, Netgear versions range from V1.0.0.34 to V1.0.0.44, and Beklin versions range from V1.00.06 to V1.00.30. To discover the mixed-version firmware on the Internet, we deploy the prototype system on Amazon EC2 running with 450Mbps of bandwidth.

We only use ports 80 and 8080, because the directory "WWW" usually provides the response data on those two ports. We employ the scanner tool ZMap [24] to send a TCP-SYN packet to every IP address. If the host gives a response, we add it to the live list; otherwise, we drop it out. After the horizontal scanning, our live list includes millions of live IP addresses. As aforementioned, we can identify which vendor the firmware is from by using one pair of $< request, response >$. Then, we send a request to every host on this list and use the pair of $< request, response >$ for identification. If the similarity score is larger than the threshold, we identify that a particular version of firmware is running on this IP address. Table 4 lists online embedded devices running with the different versions of the firmware that belong to D-Link, Beklin, and Netgear. We can see that there is no uniform distribution for firmware versions. The latest version only occupies a small portion of devices compared with other types of firmware. It conforms to our previous observation that many embedded devices are still using outdated-version firmware even though their manufacturers have distributed updated patches.

Finally, we use proof-of-concept exploits to validate whether those online embedded devices in Table 4 are vulnerable. For ethical considerations, we do not run the exploitation scripts for validation in practice. Instead, we crawl the vulnerability information from the CVE website [25]. For each item in the vulnerability list, we obtain their common weakness enumeration specification (CWE) from the National Vulnerability Database [2]. The CWE provides information about software security vulnerabilities according to the system architectures. We compare the version information in Table 4 with the CWE item to determine whether an online embedded device is vulnerable. If it does match, the online embedded device running with the firmware is vulnerable. We observe that many embedded devices are vulnerable on the Internet. For instance, Netgear firmware V1.0.0.34, which is vulnerable to Cross-Site Scripting (CVE-2013-3069) and SMB Symlink Traversa (CVE-2013-3073), is used in 3,138 embedded devices. Our findings show that embedded devices are still insecure, because of using outdated versions of the firmware.

6 Device Vulnerability Quantification

In this section, we first validate the efficacy of firmware-vulnerability extraction from security reports. Then we present the major findings of device vulnerabilities and patch status.

6.1 Implementation

We implement a prototype system of device vulnerability extraction for the firmware based on open source libraries. For crawling websites, we use wget and the scrapy crawling framework [15] to implement web crawler scripts. We use the Beautiful Soup library [17] and the Natural Language Toolkit [16] to implement the data preprocessing module. Specifically, BeautifulSoup generates the HTML tag of every element on a webpage while keeping the main content and external links. NLTK calculates the similarity value between any two documents and filters out duplicated ones. We use fastText [26] to learn the classification model for identifying whether a webpage includes the vulnerability information. Every report is converted into an embedding word vector by fastText [26]. Features include the top 500,000 most frequent words in the document, which are mapped to 200 size word vector in a hash bucket. We set the learning rate as 0.1 to run a grid search for learning the classification.

6.2 Effectiveness

We conduct experiments to validate the effectiveness of our system, including data preprocessing, classification, and information extraction. We collect a labeled dataset for the evaluation, which contains 4,000 webpages. We manually provide the ground truth labels for these webpages.

Data Preprocessing. As we mentioned before, many online sources might involve the same content because of the cross-reference and reproduction. We use the three metrics to characterize the effectiveness of our system, including recall, performance, and accuracy. When the similarity threshold is 0.97, the precision is 0.97 and the accuracy is

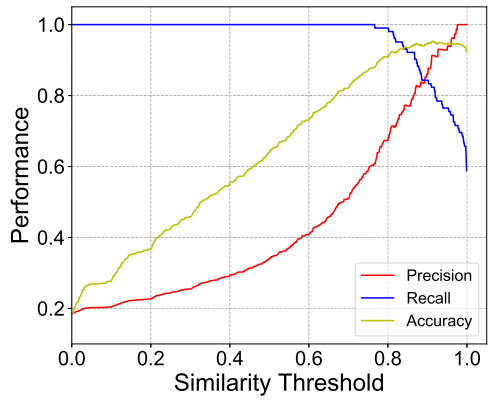


Fig. 13. The performance of the similarity threshold to detect the duplicated webpages.

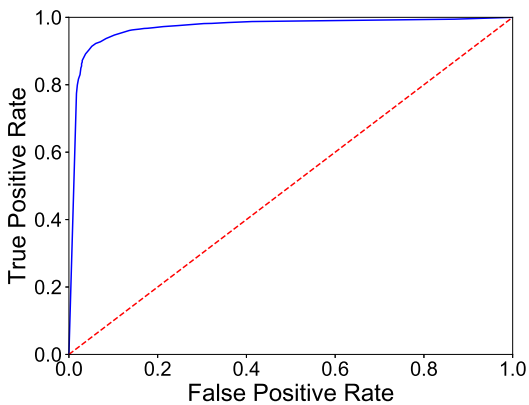


Fig. 14. The performance of the classifier.

TABLE 5
The Overview of the Security Reports

	Total Num	Vulnerability-related Num	Firmware-related Num
Blogs	57,830	47,855	1,390
Forums	138,104	2,198	368
Mail list	107,902	45,098	4,218
Data Archive	43,849	28,688	922

0.94. Fig. 13 shows the performance of our data preprocessing along with the similarity threshold. In practice, our system can detect a similar webpage with high accuracy.

Classification. The classification model determines whether a document contains vulnerability information. We use 10-fold cross-validation on the dataset to derive parameters of the classification model. Fig. 14 plots the performance of the classifier. The result shows that our classification model achieves 94% precision, 92% recall, and 93% accuracy on average. The classification model has only a small fraction of false positives and negatives, which have a limited impact on the overall document classification.

Information Extraction. Aforementioned, we use the regex to extract device-relevant terms from the security reports. Over the labeled dataset, the precision of our firmware extraction is 100%. Note that we do not have the number of false negatives because the regex matching cannot find the missed terms. The information extraction is modeled as a Named Entity Recognition (NER) problem in NLP. However, the NER is highly domain-specific, and those designed for one domain hardly work well on the other domains.

TABLE 6
The Distribution of Device Types and Vendors Over Firmware Vulnerabilities

Device Type	Number	Vendor Name	Number
Firewall	3,247	Cisco	3,954
Router	3,238	Linksys	632
Switch	2,207	Dlink	587
Gateway	917	Netgear	382
Scanner	647	Juniper	374
Modem	512	Verizon	306
Camera	464	TP-link	255
Access Point	460	Sonic	208
Printer	274	ASUS	169
DVR	110	AXIS	140

TABLE 7
The Distribution of Vulnerability Categories

Category	Number
Denial of Service	799
Code Execution	412
Cross Site Scripting	408
Buffer Overflow	390
Cross Site Request Forgery	264
Command Injection	243
Authentication bypass	220
SQL Injection	204
Privilege Escalation	181
Information Disclosure	142

6.3 Findings

We extract firmware-related vulnerability information from online resources and present the major findings below. After the data preprocessing, we collect 347,685 webpages from online resources (Table 1), including technical blogs, security forums, mail lists, and data archive. Table 5 shows an overview of the number of vulnerability-related documents and firmware-related reports. Our classification detects nearly 121,641 vulnerability-related reports, and there are 6,898 reports containing the firmware information.

These reports disclose vulnerabilities including relevant device types, vendors, and products. Table 6 lists the top 10 device types and vendor names across firmware vulnerabilities. It is obvious that the distribution of firmware vulnerabilities follows a long-tail for embedded devices. For device vendors, Cisco has the most security flaws, with its largest market shares. Cisco, Linksys, and D-Link are the top 3 device vendors with the firmware vulnerabilities of 3,954, 632, and 587, respectively. We further investigate the vulnerability types for the firmware of embedded devices. Table 7 lists the top 10 vulnerability types in the format of CWE. Specifically, denial of service, code execution, and cross-site scripting are the top 3 vulnerabilities/flaws on embedded devices. We observe that many vulnerabilities could be used as exploit scripts to compromise embedded devices.

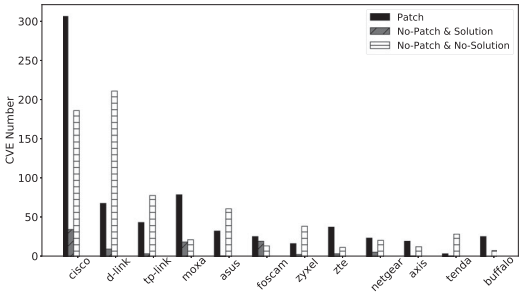


Fig. 15. The distribution of patch status of firmware vulnerability.

As we mentioned before, we use publicly disclosed vulnerabilities from NVD to identify their patch statuses. Fig. 15 shows the distribution of patch statuses of firmware vulnerabilities across device vendors, including vendor patch, no-patch&solution, and no-patch&no-solution. We observe that only 685 vulnerabilities have the patches to update the firmware, available at the official vendor websites and third-party websites. For device vendors, Cisco has the largest number of patches for its products. The vendors with the large market are all reputable for providing vendor patches for fixing vulnerabilities. However, we find that 694 vulnerabilities have neither any patch nor any solution for the firmware.

7 Related Work

Firmware. The firmware is the software platform of embedded devices, where device vulnerabilities are related to the firmware. Prior works detected firmware vulnerabilities through static analysis and dynamic analysis. Static analysis is performed over the source code and detects underlying vulnerabilities without executing the firmware. Costin *et al.* [27] collected 32 thousand firmware images and unpacked them to run simple static analysis tasks over those files within the filesystem. Yan *et al.* [28] proposed Firmalice that detected the backdoor vulnerabilities of the firmware through symbolic execution and enumerating security policies. Chen *et al.* [5] proposed Dtaint that detected the Taint-style vulnerability in the firmware through constructing interprocedural data flows.

Dynamic analysis is performed over the firmware on a real device or virtual host. Zaddach *et al.* [7] presented AVATAR to simulate peripherals of embedded devices for the firmware images and performed dynamic analysis on the virtual hosts. Chen *et al.* [29] presented FIRMADYNE to reduce costs and time for simulating firmware images over the hardware of devices for a large-scale dynamic analysis. Cui *et al.* [30] focused on a particular embedded device (HP LaserJet printer) for exploiting a vulnerability caused by the firmware update.

However, firmware vulnerability detection (both static analysis and dynamic analysis) relies on professional background and significant manual cost. In contrast, our work is about generating firmware fingerprints and using them to understand device vulnerabilities caused by the firmware.

Fingerprinting. Fingerprinting is a technique for identifying the operating system (OS), applications, or network services. State-of-the-art tools (e.g., Nmap [20]) usually utilize the differences between TCP/IP implementations to identify OS versions. They also use service banners in the application-level protocols to find network services and devices. Shodan [23] and Censys [21] 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. Feng *et al.* [22] proposed the ARE tool which utilized natural language processing to generate banner rules to discover devices on the Internet automatically. Yang *et al.*[31] leveraged the neural network to learn the classification model of embedded devices as devices fingerprints. However, the granularity of this information is too coarse to identify firmware information and corresponding device vulnerabilities. Much of the firmware of embedded devices use Linux systems and supports more network protocols, making those existing tools ineffective for fingerprinting the firmware.

The clock skew [32], [33] was proposed for fingerprinting devices by exploiting the differences in time synchronization based on network time protocol. Yang *et al.* [34] proposed the iFinger which utilizes register states to generate fingerprints of industrial control systems. 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 a firmware rather than individual devices. Our work utilizes filesystem differences among firmware images to generate firmware fingerprints for embedded devices.

Vulnerability Information. Online resources consist of a large amount of vulnerability information, including flaw descriptions and corresponding software platforms. Prior works propose to extract vulnerability information from online sources. Li *et al.* [35] conducted a large-scale empirical study of security patches centering on the National Vulnerability Database (NVD). They leveraged the web crawler to scrape the relevant information from the URL links of vulnerability-related webpages. Song *et al.* [36] leveraged the online sources (aggregation websites) to understand the characteristics of webcams' distribution and their privacy/security implications. Sabottke *et al.* [37] extracted the vulnerability-related information on the Twitter for predicting the influence over the real world. Liao *et al.* [38] utilized natural language processing to extract the indicators of compromise information from the textual descriptions of websites, such as articles, posts, and white papers. By contrast, we focus on extracting the firmware-related vulnerabilities from different online sources. We gather nearly 347,685 documents and find 12,321 firmware-related vulnerabilities for embedded devices.

8 Conclusion

Nowadays, embedded devices play a crucial role in our daily lives, connecting to the Internet, including routers, IP-cameras, and printer. The vulnerability is 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 information. Furthermore, we gather 9,716 firmware

images from official websites of device vendors and 347,685 security reports from online resources. 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. We found that 6,898 reports have the firmware and corresponding vulnerability information, and more than 10% firmware vulnerabilities have neither any patches nor any solutions for mitigating underlying risks.

References

[1] M. Antonakakis *et al.*, "Understanding the mirai botnet," in *Proc. USENIX Secur. Symp.*, 2017, pp. 1092–1110.

[2] NVD-CPE, "Computer security resouce center for official common platform enumeration dictionary," Accessed: 2017. [Online]. Available: https://nvd.nist.gov/products/cpe

[3] Security notification - HTTP buffer overflow vulnerability in Hikvision NVRs devices. Accessed: 2015. [Online]. Available: http://www.hikvision.com/En/Press-Release-details_435_i1023.html

[4] Q. Li, X. Feng, R. Wang, Z. Li, and L. Sun, "Towards fine-grained fingerprinting of firmware in online embedded devices," in *Proc. IEEE INFOCOM -IEEE Conf. Comput. Commun.*, 2018, pp. 2537–2545.

[5] C. Kai, Q. Li, L. Wang, Q. Chen, Y. Zheng, L. Sun, and Z. Liang, "DTaint: Detecting the taint-style vulnerability in embedded device firmware," in *Proc. 48th Annu. IEEE/IFIP Int. Conf. Dependable Syst. Netw.*, 2018, pp. 430–441.

[6] F. Yamaguchi, N. Golde, D. Arp, and K. Rieck, "Modeling and discovering vulnerabilities with code property graphs," in *Proc. IEEE Symp. Secur. Privacy*, 2014, pp. 590–604.

[7] J. Zaddach, L. Bruno, A. Francillon, and D. Balzarotti, "AVATAR: A framework to support dynamic security analysis of embedded systems' firmwares," in *Proc. Symp. Netw. Distrib. Syst. Secur.*, vol. 23, 2014, pp. 1–16.

[8] Binwalk, the tool for analyzing, reverse engineering, and extracting firmware images. Accessed: 2020. [Online]. Available: http://binwalk.org/

[9] Firmware modification kit, the tool for extracting the firmware into its component parts, and the file system image. Accessed: 2011. [Online]. Available: https://bitsum.com/firmware_mod_kit.htm

[10] Tools for extracting, modding and re-packaging firmwares of DJI multirotor drones. Accessed: 2018. [Online]. Available: https://github.com/o-gs/dji-firmware-tools

[11] J. Zaddach and A. Costin, "Embedded devices security and firmware reverse engineering," *Black-Hat USA*, 2013.

[12] F. Bellard, "QEMU, a fast and portable dynamic translator," in *Proc. Annu. Conf. USENIX Annu. Tech. Conf.*, 2005, pp. 41–46.

[13] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," 2013, *arXiv:1301.3781*.

[14] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," in *Proc. Neural Inf. Process. Syst.*, 2013, pp. 3111–3119.

[15] Scrapy, A fast and powerful scraping and web crawling framework. Accessed: 2020. [Online]. Available: https://scrapy.org

[16] Natural language toolkit. Accessed: 2021. [Online]. Available: http://www.nltk.org/

[17] Beautiful Soup, A Python library designed for quick turnaround projects. Accessed: 2004. [Online]. Available: https://www.crummy.com/software/BeautifulSoup/

[18] X. Xu, C. Liu, Q. Feng, H. Yin, L. Song, and D. Song, "Neural network-based graph embedding for cross-platform binary code similarity detection," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2017, pp. 363–376.

[19] Z. Yu, R. Cao, Q. Tang, S. Nie, J. Huang, and S. Wu, "Order matters: Semantic-aware neural networks for binary code similarity detection," in *Proc. AAAI Conf. Artif. Intell.*, vol. 34, no. 01, 2020, pp. 1145–1152.

[20] Nmap," 1997. [Online]. Available: https://nmap.org/

[21] Z. Durumeric, D. Adrian, A. Mirian, M. Bailey, and J. A. Halderman, "A search engine backed by internet-wide scanning," in *Proc. 22nd ACM SIGSAC Conf. Comput. Commun. Secur.*, 2015, pp. 542–553.

[22] X. Feng, Q. Li, H. Wang, and L. Sun, "Acquisitional rule-based engine for discovering internet-of-things devices," in *Proc. 27th USENIX Secur. Symp.*, 2018, pp. 327–341.

[23] Shodan, "The search engine for Internet-connected devices," Accessed: 2012. [Online]. Available: https://www.shodan.io/

[24] Z. Durumeric, E. Wustrow, and J. A. Halderman, "ZMap: Fast Internet-wide scanning and its security applications," in *Proc. 22th USENIX Secur. Symp.*, 2013, pp. 605–620.

[25] CVE, Common Vulnerabilities and Exposures. Accessed: 2021. [Online]. Available: http://cve.mitre.org/

[26] F. A. R. Lab, "A library for learning of word embeddings and text classification," 2016. [Online]. Available: https://fasttext.cc/

[27] A. Costin, J. Zaddach, A. Francillon, D. Balzarotti, and S. Antipolis, "A large-scale analysis of the security of embedded firmwares," in *Proc. 23th USENIX Secur. Symp.*, 2014, pp. 95–110.

[28] Y. Shoshitaishvili, R. Wang, C. Hauser, C. Kruegel, and G. Vigna, "Firmalice-automatic detection of authentication bypass vulnerabilities in binary firmware," in *Proc. Symp. Netw. Distrib. Syst. Secur.*, 2015, pp. 1–15.

[29] D. D. Chen, M. Egele, M. Woo, and D. Brumley, "Towards automated dynamic analysis for linux-based embedded firmware," in *Proc. Symp. Netw. Distrib. Syst. Secur.*, 2016, pp 1–15.

[30] A. Cui, M. Costello, and S. J. Stolfo, "When firmware modifications attack: A case study of embedded exploitation," in *Proc. Symp. Netw. Distrib. Syst. Secur.*, vol. 1, 2013, pp. 1–15.

[31] K. Yang, Q. Li, and L. Sun, "Towards automatic fingerprinting of IoT devices in the cyberspace," *Comput. Netw.*, vol. 148, pp. 318–327, 2019.

[32] T. Kohno, A. Broido, and K. C. Claffy, "Remote physical device fingerprinting," *IEEE Trans. Dependable Secure Comput.*, vol. 2, no. 2, pp. 93–108, Apr.–Jun. 2005.

[33] S. Zander and S. J. Murdoch, "An improved clock-skew measurement technique for revealing hidden services," in *Proc. 17th Conf. Secur. Symp.*, 2008, pp. 211–225.

[34] K. Yang, Q. Li, X. Lin, X. Chen, and L. Sun, "iFinger: Intrusion detection in industrial control systems via register-based fingerprinting," *IEEE J. Sel. Areas Commun.*, vol. 38, no. 5, pp. 955–967, May 2020.

[35] F. Li and V. Paxson, "A large-scale empirical study of security patches," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2017, pp. 2201–2215.

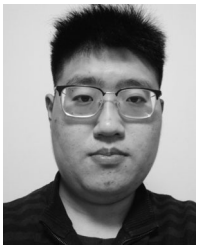
[36] J. Song, Q. Li, H. Wang, and L. Sun, "Under the concealing surface: Detecting and understanding live webcams in the wild," *Proc. ACM Meas. Anal. Comput. Syst.*, vol. 4, no. 1, pp. 1–25, 2020.

[37] C. Sabottke, O. Suciu, and T. Dumitraş, "Vulnerability disclosure in the age of social media: Exploiting twitter for predicting real-world exploits," in *Proc. 24th USENIX Secur. Symp.*, 2015, pp. 1041–1056.

[38] X. Liao, K. Yuan, X. Wang, Z. Li, L. Xing, and R. Beyah, "Acing the IOC game: Toward automatic discovery and analysis of open-source cyber threat intelligence," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur., CCS*, 2016, pp. 755–766.

Qiang Li received the PhD degree in computer science from the University of Chinese Academy of Sciences in 2015. He is currently an associate professor with the School of Computer and Information Technology, Beijing Jiaotong University, China. His research interests include Internet of Things, networking systems, network measurement, artificial intelligence and machine learning for cybersecurity, and mobile computing.

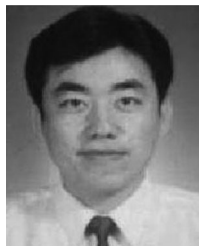
Dawei Tan recieved the BE degree from Beijing Jiaotong University, China, in 2018. He is currently working toward the MS degree with Beijing Jiaotong University, China. His research interests include the cyber security and Internet measurement.



Xin Ge recieved the BE degree from Ludong University, China, in 2017. He is currently working toward the MS degree with Beijing Jiaotong University, China. His research interests include the cyber security and Internet measurement.

Haining Wang (Fellow, IEEE) received the PhD degree in computer science and engineering from the University of Michigan, Ann Arbor, MI, USA, in 2003. He is currently a professor with the Department of Electrical and Computer Engineering, Virginia Tech, USA. His research interests include security, networking systems, cloud computing, and cyber-physical systems.

Zhi Li received the BS degree from Xidian University, China, in 2008 and the PhD degree from the Graduate University of Chinese Academy of Sciences, Beijing, China, in 2013. He is currently an associate researcher with the Beijing Key Laboratory of IoT Information Security Technology, Institute of Information Engineering, China Academy of Sciences, Beijing. His research interests include Internet of Things (IoT) security, network surveying and mapping.



Jiqiang Liu (Senior Member,IEEE) received the BS and PhD degrees from Beijing Normal University, in 1994, and 1999, respectively. He is currently a professor with the School of Computer and Information Technology, Beijing Jiaotong University. His research interests include cryptographic protocols, privacy preserving, and network security.