Measurement Study Towards a Unified Firmware Updating Scheme for Legacy IoT Devices

Bing-Kai Hong*, Jr-Wei Huang*, Tao Ban†, Ryoichi Isawa†, Shin-Ming Cheng*, Daisuke Inoue†, Koji Nakao†
*National Taiwan University of Science and Technology, Taipei, Taiwan
{M10315020, B10415030, smcheng}@mail.ntust.edu.tw
†National Institute of Information and Communications Technology, Tokyo, Japan
{bantao, isawa, dai, ko-nakao}@nict.go.jp

***Abstract*—This paper provides a measurement study on the IoT firmware. Based on a thorough review of the state of art of IoT firmware emulation and vulnerability scan tools and techniques, we propose a unified framework that could monitor the security status of users' IoT devices and keep the device firmware up to date to prevent malware infection. Towards this goal, we conduct a measurement study on an IoT firmware image set obtained from three major vendors. The result of the measurement study indicates that the market is in a pressing need of such a universal framework for improving the security situation of a mass of vulnerable IoT devices.**

***Index Terms*—IoT firmware update, virtualization, vulnerability scan**

I. Introduction

A. Introduction

The past decade has witnessed the fast adventure of the ecosystem of discrete computing devices know as Internet of Things (IoT). By drastically changing the way we generate and make use of data, IOT had brought forth new and transformative opportunities for e-business and industry alike and led to significant improvements in efficiency, productivity, profitability, decision-making and effectiveness [1].

As the proliferation of IoT continues—it is estimated to reach 50 billion connected devices by 2020, and the number will be ten times more by 2030 [2]—cyberthreats towards it are also on the rise: According to a survey by Kaspersky, in the first half of 2018, over 120,000 new malware targeting IoT are discovered, which is three times the number of 2017 and ten times of that of 2016 [3]. Moreover, the threat level for IoT malware also rises rapidly due to the emergence of IoT malware that targets critical Internet infrastructure and the wide application of vulnerable IoT devices to high-risk environments such as health and safety industries. Unfortunately, very few countries have regulations for IoT devices. Moreover, it is not always a primary concern for device vendors in a rush to sell the equipment: Making devices more secure can add time and cost to product development [4]. As summarize in a Symantec report [5], IoT security have to treat with the following challenges new from past practices with conventional computer systems.

- Extraordinary large number of endpoints to monitor;
- Lack of identity or permission management;
- Ample amount of inherently unprotected devices (e.g., RFID tags and sensors can be read and hacked easily);
- All devices have IP addresses which means they can be discovered and hacked;
- Low-energy devices do not have power or storage to negotiate complicated handshake protocols or store encryption keys; and
- Ease-of-use is preferred over security when designing IoT compatible devices.

As with computers, IoT devices commonly require security patches and updates, hereafter referred as firmware updating, to protect against known vulnerabilities and attacks from outsiders. Other incentives of firmware update include bug fixing, configurations upgrading, and services implantation. Despite the importance of firmware updating, it is unlikely that regular firmware updating can be included as a part of the contract when a user acquires an IoT device. The situation is even worse due to the lack of a unified, user-friendly interface to update multiple devices. As the consequence of lack of regular firmware updates, a large swarm of outdated legacy IoT devices are command & controlled by the IoT botnets such as *Mirai* and *Hajime* to launch attacks towards critical Internet infrastructure [6].

There is a pressing need to design and implement an effective and efficient paradigm for firmware updating. Also, there have been some attempts aiming at this goal [7]. An ideal solution to firmware update could be a fully automated process in an authenticated channel where device vendors can hot-fix the security vulnerabilities of the devices deployed at the users' environment. However, there seems to be a long way to have this solution implemented with all connected IoT devices as long as the legacy devices considered.

In this paper, we propose a unified firmware update scheme that keeps the user regularly informed about the security situation of his/her IoT devices and provide operational support when applicable. Implemented as a system that integrates functionalities including security vulnerability database and a portal of firmware configuration, it tracks the vulnerability information of the registered devices and presents it to the user in an easy to understand way. Based on the information, the user can obtain the most recent security status information about the devices (e.g., the release of new firmware patches

978-1-7281-2556-5/19/$31.00 ©2019 IEEE
DOI 10.1109/AsiaJCIS.2019.00010

and known vulnerabilities) and choose the timing to update the firmware.

The feature of the new scheme is highlighted as follows.

- First, the scheme can cover a wide range of IoT devices and provide an appropriate level of services (e.g., information such as security status and operation such as firmware update) pertinent to the user's request and the device's functional capability.
- Second, by integrating well-known vulnerability databases and state-of-the-art emulation and vulnerability-scan tools, it aims at a coverage of well studied and new vulnerabilities as complete as possible.
- Finally, it supports a unified, user-friendly interface (e.g., a mobile application) so that security status and vulnerability information of registered devices can be conveyed smoothly to the user for decision making.

This paper is structured as follows: Section II surveys related work. Section III describes our system framework in detail. Section IV provides the results of a measurement study of current IoT market. Section V gives some discussions on the limitation of the work. Section VI concludes the paper.

II. Related Work

In this section, we survey some related work that serve as implementation components of the proposed scheme: security vulnerability databases which can serve as trustable information sources of vulnerability information of the devices, emulation tools to build hosting environment for the firmware and simulate the function of the firmware for deep inspection, and firmware vulnerability scanning tools that could automatically generate vulnerability report for a firmware that is new to the system.

A. Vulnerability Databases

NVD (National Vulnerability Database) is the U.S. government repository of standards based vulnerability management data [8], which was launched by NIST (National Institute of Standards and Technology) in 2005. It includes a software-vulnerability database synchronized with the CVE (Common Vulnerabilities and Exposures) list [9], and any analysts in the world can submit an entry of a newly found vulnerability to the CVE list. When any updates to CVE appear, the vulnerability database in NVD is also soon updated by NVD analysts.

The vulnerability database in NVD contains some entries of, for example, an attack vector (e.g., *Network* and *Local*), a base score of severity (e.g., *5.3 MEDIUM* and *7.1 HIGH*), and a description corresponding to a vulnerability of products. For this database, a search engine named *Search Common Platform Enumerations (CPE)* is provided via NVD's website[1], and, through CPE, the vulnerability check of an IoT device can be done by inputting a product version (e.g., *dap-2360* and *p-660hw_d3*) to be checked. If a product has any registered vulnerabilities, the search result will list the entries corresponding to those vulnerabilities.

Similar to NVD, our system possesses a vulnerability database, so our system can give a good collaboration by (manually) feeding the vulnerabilities found by our system as entries to CVE; this contributes to keeping the freshness of CVE and the vulnerability database in NVD.

B. Firmware emulation

Firmware emulation enables analysts to run firmware images without actually having any IoT devices, and they can conduct vulnerability checks against those running firmware images. This is very reasonable and efficient since the analysts are not required to purchase the devices. In this section, we introduce three ways to emulate firmware images.

The first is a way of combining Linux commands as follows. After obtaining a Linux-embedded firmware image typically from a vendor's website, an analyst extracts root-system files from the image such as `/bin/busybox`, `/dev/console`, and `/sbin/httpd` with an extractor. For example, `extract-ng.sh`[2], a bash script for extraction, automatically extracts such root-system files by executing some Linux commands including *binwalk*[3] and *dd*. The analyst then executes extracted programs to fuzz. Examples of those programs include httpd, a daemon of web server, and it will be emulated with QEMU as "`chroot . ./qemu-mips-static sbin/httpd`", where `chroot` temporally changes the system root directory to the current directory with the argument "." and `qemu-mips-static` is an emulator to run a MIPS-version of `httpd` in `sbin` (as "*/sbin/httpd*") on a Linux running on another CPU architecture like x86_64.

The second way is to use Avatar [10], a framework that adopts a hybrid approach combining Linux-embedded firmware emulation with physical IoT devices to be analyzed. Usually, the firmware exchanges some messages with hardware components (e.g., Ethernet and field bus) embedded in the physical IoT device, and the firmware could stop running if the emulated hardware does not respond to a message from the firmware. To treat with this, when Avatar receives a message from the firmware, it forwards the message to the physical device. Avatar also forwards a response sent from the physical device to the firmware. At this time, Avatar can fuzz the firmware with fuzzing functionality implemented in the firmware emulation of Avatar. This is a significant advantage of the hybrid approach, which means implementing the fuzzing in the emulation part is much easier than in the physical IoT device.

The last is to use Firmadyne, an automated system that performs emulation and dynamic analysis of Linux-based embedded firmware without any physical IoT devices [11]. It includes (software) components for firmware emulation like a user-space NVRAM (nonvolatile random-access memory)

[1]https://nvd.nist.gov/products/cpe/search

[2]https://github.com/lattera/dd-wrt

[3]https://tools.kali.org/forensics/binwalk

library and modified kernels for MIPS and ARM, respectively. When emulating firmware, Firmadyne first extracts root system files from the firmware image by using `binwalk`, and then it executes those extracted programs with a modified kernel on QEMU. It also possesses a script that can check the emulated firmware for 60 known vulnerabilities using exploits from Metasploit [12] and 14 previously-unknown vulnerabilities discovered by the developers of Firmadyne.

For the measurement study in this paper, we adopt and modify Firmadyne because it unifies Linux commands, including `binwalk` and it also does not require any physical devices. The modification enables the checking for more vulnerabilities with Routersploit [13] and other tools.

C. IoT firmware vulnerability scan

Firmware vulnerability scan methods can be divided into static-analysis-based and dynamic-analysis-based.

1) Static-analysis-based approaches: Firmware vulnerability scan methods are usually categorized into static analysis and dynamic analysis.

2) Static-analysis-based approaches: Static analysis involves no dynamic execution of the firmware under test and can detect possible defects in an early stage, before running the firmware in an emulation environment which is usually complicated. Examples of this kind of methods include Wang et al.'s method [14] and Firmalice [15]. Wang et al.'s method constructs a model based on known vulnerable functions as training data, which is used to check if firmware involves vulnerable functions. Firmalice focuses on a privileged operation in the firmware like `system(input("command:"))` executed in control statements (e.g., `if-else` and `switch`), and it shows a control flow graph that contains a privileged program point with the input required to trigger the bypass (i.e. a control flow to a backdoor).

3) Dynamic-analysis-based approaches: Dynamic analysis executes firmware in a testing environment and directly trace its execution while checking for vulnerabilities. There are three methods specialized for web services running following this idea: Arachni [16], Zed Attack Proxy (ZAP) [17], and w3af [18]. These methods send a particular payload to a web service for triggering service errors such as buffer overflow, (remote) code execution, and command injection. IoTFuzzer [19] and Firmadyne [11] provide functions to trace services other than web services. Targeting IoT devices that are controlled by mobile applications (e.g., IP cameras, printers, and smart plugs), IoTFuzzer extracts commands from mobile applications, which are used to communicate between mobile applications and IoT devices. Based on these commands, IoTFuzzer checks IoT devices with the fuzzed payload for discovering memory-corruption vulnerabilities. As is briefly introduced in section II.B, Firmadyne [11] can also check IoT devices for vulnerabilities with firmware emulation.

In this paper, we adopt Firmadyne from the above systems to check a wide variety of IoT devices. Our system treats with IoT devices that are not controlled by mobile applications. Also, our system aims to check as many services as possible in addition to web services.

III. System Framework

The IoT consists of all kinds of devices from legacy devices lack of regular firmware updates to new IoT devices with a secure firmware update architecture implemented following the most recent guidelines such as those described in [7]. In Fig. 1, we illustrate a firmware update scheme that tries to manage the automation of firmware updating to a range of IoT devices as wide as possible.

A. Components in the Scheme

- User: The user is the authenticated operator of a fleet of IoT devices. In the scheme, the user is required to interact with other entities. He/she can provide and receive the necessary information to and from other entities in the environment, and then make decisions about system operations such as firmware updating. Meantime, the scheme is designed to provide pertinent information to the user to enable smart decision making as relief from the operational burden. In any case, firmware updating will only be performed with the user's consent.
- Device: A device refers to the entire IoT product connected to the Internet to implement a designated function, regardless of its implementation detail. In general, we assume the user has at least network access to collect related information from the device and to issue the command to perform firmware updating.
- Status Tracker: The status tracker offers device management function to monitor the firmware update process. A status a tracker may, for example, want to know the current state of the firmware update cycle of the device. As operation automation could be implemented based on appropriate status information of the IoT device, the status tracker plays an essential part in the proposed scheme. Ideally, an intelligent status tracker provides device management function to monitor the firmware update process, and send such information to the agent to enable more services.
- Agent: The agent, which interacts with most of the entities, is the core component in the scheme. It is an interpreter of publicly available vulnerability database system that provides the vulnerability information of registered IoT devices to the user. When a previously unknown IoT device is registered, the agent can also initiate a vulnerability scanner to obtain the vulnerability information.
- Vulnerability database: The vulnerability database serves as the information source of device vulnerabilities. Search results from public vulnerability database such as NVD will be reinterpreted by the agent to provide device-specific vulnerability to the user.
- Vulnerability scanner: The vulnerability scanner is composed of a crawler, an emulator, and a vulnerability scanner. The crawler proactively searches out for the

newest firmware images from IoT device vendor repositories. The emulator provides the running environment of the firmware. The vulnerability scan tools generate vulnerability report for the firmware.

B. Interactions of the component

The proposed scheme follows the general guideline below. The user provides information about the devices; the system will continuously monitor the latest update of the firmware and provide information to the user if any vulnerability of the registered devices is detected. Then it will try to acquire the user's consent so that firmware updating could be enabled efficiently. In the following, we define the interactions between the entities defined in the previous section. Base on the functional capability of the device, five levels of automation are defined.

- Level 1, Offline devices: the user could only get the model number and firmware number of the device by some documents like a manual. The user could even have not physical access to the device, but they need to be kept informed about the security status of the device. This category of devices include devices with missing manuals, hijacked devices, and devices of interest for information gathering. For these devices, the agent can provide information such as the security status of the devices. No operational functions will be provided.
- Level 2, Online: the device provides some means (e.g., API) for retrieving the device information through a network connection. These devices include legacy IoT devices accessible from a private network or the Internet. For these devices, similar to the previous level, the agent can provide information such as the security status of the devices. No operational functions will be provided.
- Level 3, Remote operational: the device provides some means (e.g., command line tools) to perform firmware update from a network connection. For these network accessible and configurable IoT devices, in addition to security information provision, the agent can also redirect the user to the network-based configuration tool to enable firmware updating.
- Level 4, Advanced remote controllable: the device is implemented with a unified API that supports network-based access and control. For these devices, the agent can provide a unified authentication and firmware updating mechanism. Firmware updating will be enabled with the user's consent.
- Level 5, Advanced remote controllable: the device is implemented with an advanced firmware updating infrastructure that could support full automation, e.g., with the architecture in [7] implemented. For these devices, the agent can provide a unified authentication mechanism. It will call the firmware updating routine after user's consent is approved.

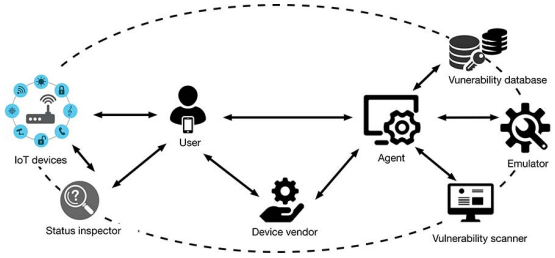


Fig. 1. The overview of our system framework.

C. Implementation

In this section, we present the implementation detail of the proposed scheme.

1) Implementation of web crawler: Our scheme is designed to detect the new firmware as soon as its release. As the firmware image download method provided by the vendors are different from each other. Some vendors prefer providing download links with FTP services, while others prefer providing product list only so that we have to access the product web page and find the download link. We create a web crawler to treat with different ways of firmware image distribution from vendors.

2) Implementation of database: We store detail firmware data (such as vendor, device type, model number, CPU architecture, firmware version number) and analysis results (such as vulnerability results provided by scanning tools and information retrieved from the vulnerability databases) to the database, so other system components can easily access the data.

3) Implementation of vulnerability scan: Our vulnerability analysis makes use of three open source tools. For vulnerabilities related to IoT devices, web service is important because most IoT devices let the user use the website to do the configuration. We make use of two web scanners. The first is Arachni, whose function is powerful and can detect many kinds of vulnerabilities such as OWASP top 10, side channel attacks. The second is w3af, which is popular among web-service vulnerability scanning tools. Finally, we choose Routersploit [13] to detect vulnerabilities on other services like FTP, telnet, and ssh, etc. By using these tools, we can detect most of the vulnerabilities on the web service and other popular services hosted by the IoT devices.

IV. Measurement Study

This section presents the result of a measurement study on firmware updating. We try to obtain an overall picture of the security situation of the IoT firmware repositories on the market.

A. Statistics on vendor, device type, and CPU architecture.

We use a crawler to download the firmware images from three major IoT device vendors' (hereafter referred to as vendor A, B and C) firmware repositories. For the measurement study, we download a total number of 15,839 firmware image

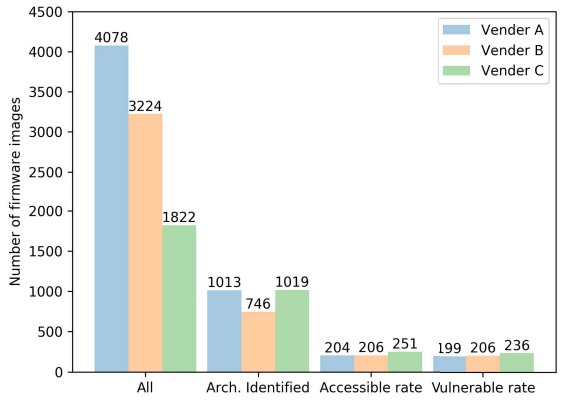


Fig. 2. Access success rate of firmware images.

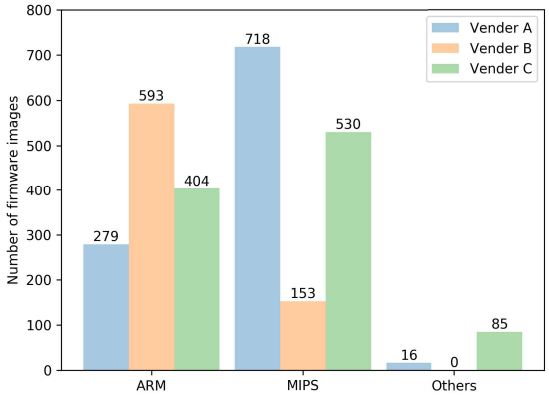


Fig. 3. Distribution of firmware images on different CPU architectures.

files. After using the hash functions to check the uniqueness of the images files, we got 10,986 unique image files. Next, we try to identify the firmware architectures of the firmware using *binwalk*, and use emulation tool to boot up the firmware. As a result, we find that 7.5% of all firmware images are recognized as ARM-base firmware and boot up successfully, and 16.3% of them are recognized as MIPS-base firmware and boot up successfully.

There have been many errors encountered during the process when a simulation tool emulates a firmware. A short list of the system errors includes boot up errors, IP address getting errors, network errors, service error, and so forth. We use the *ping* command, to send ICMP requests to the firmware images that have successfully booted up and find that 204 of vendor A's firmware, 206 of vendor B's firmware and 251 of vendor C's firmware are accessed successfully. The success rate of booting up is summarized in Fig. 2.

We divided the firmware images according to the implementation CPU architectures to show each vendor's product distribution. As can be seen from the result in Fig. 3, for all the three vendors, ARM and MIPS enjoy much higher popularity than other CPU architectures. Based on the model name of the firmware images, our crawler also can identify the IoT device types. As shown in Fig. 4, the firmware images can be divided into three categories. For the first router category, only vendor A and vendor C have devices released. The second and third categories are access points and switches, respectively. For vendor A, it has almost uniform distribution among the three categories. However, for the other vendors, they have a specific preference on the type of device they produce.

B. Results on emulation success rate and open service port.

Fig. 5 illustrate our port scan result. After using *ping* command to check the firmware's networking setting, we have 667 firmware images which can response the ICMP requests. Next, we use NMAP to scan the firmware and show the top 10 open ports. The result indicates that 50.4% of IoT devices

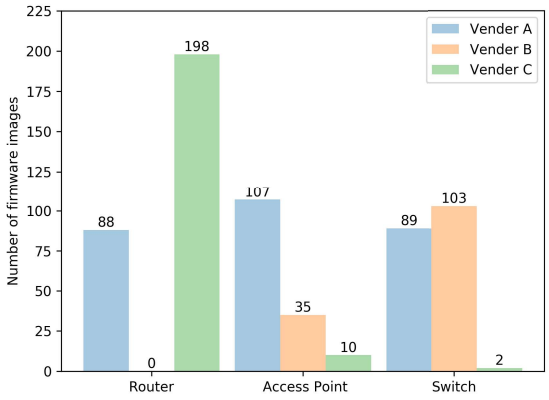


Fig. 4. Type of IoT devices.

support web service hosted at ports 80 and 443. However, only 21.6% of IoT devices which use HTTPS on port 443 are considered secure. More access point devices tend to open port 443 than router devices. Port 53 is DNS service, which is supported by 30.7% of devices. As we can see router devices takes a large proportion of port 53. DNS service could often be a hacker's target. Ports 22 and 23 are for remote access service, ssh and telnet, respectively. These two services are supported by 46.3% of devices which often be attacked by some IoT malware using a brute-force attack; ssh service is considered safer than telnet service because telnet service does not use any security mechanism, such as authentication and data encryption. In the figure, we can see only a few access point devices have supported the ssh remote access service.

C. Case study: vulnerability tracking during version up

Next, we study about the relationship between firmware versions and vulnerability numbers. We pick up four IoT devices to trace their versions and number of vulnerabilities.

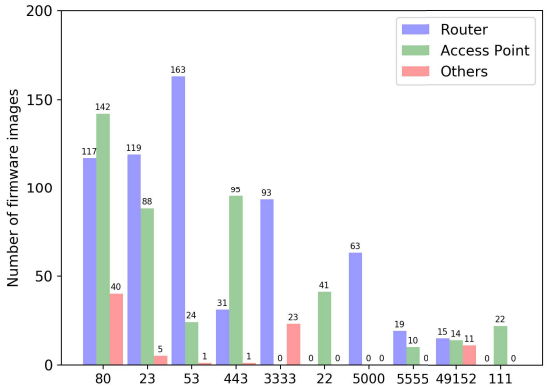


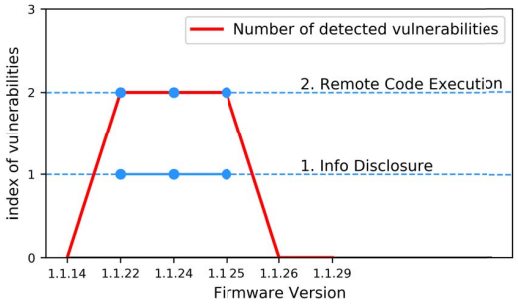
Fig. 5. Port scan result.

The first IoT device is in Fig. 6(a). As we can see, two vulnerabilities appeared from version 1.1.0.14 of firmware and were repaired on version 1.1.0.26. After that, we find no more vulnerabilities. The second device is on fig. 6(b). In the beginning, the vulnerability was discovered at version 1.00, and at version 1.08, the vulnerability still existed. Besides, version 1.08 also introduced the second vulnerability the latest version 1.09 repaired firmware vulnerabilities of the device. The third IoT device is on fig. 6(c). This device exposed two vulnerabilities at version 2.10. Until now, the latest firmware still has the same vulnerabilities. For the result of the 4th device in fig. 6(d), the authorization-bypass vulnerability existed for a long time until version 2.00. Also, we can see the cross-site-request-forgery vulnerability, was unstable between the versions. Maybe, this is because the vendors did not do some vulnerability analysis before pulling a new firmware version. The four figures show how essential firmware update is: each firmware version may bring forth difference vulnerabilities. User must make sure the router's version is the latest to ensure protection. However, sometimes the latest firmware still has un-fixed vulnerabilities. So our framework plays an essential role in this situation. We detect user's firmware version to keep the firmware always updated to the latest version and detect the firmware vulnerability to make the latest firmware more secure.

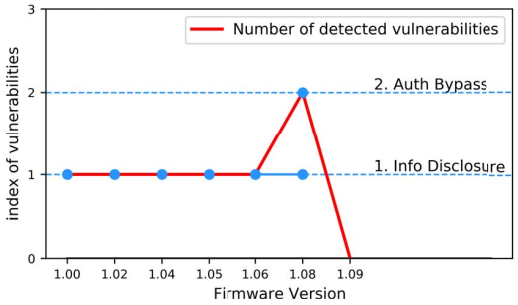
D. Limitations

At the moment, due to numerous technical difficulties, we cannot fully emulate all the firmware image files. Some of the reasons are listed below.

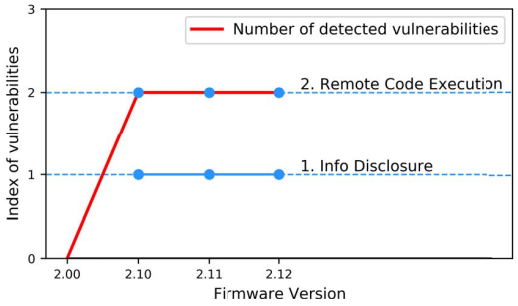
Due to the firmware image set we have at hand, the measurement study is performed mainly on the 32-bit CPU architectures, analysis for other 64-bit CPU architectures such as MIPS64, ARM64, is left as future work. We still lack some efficient means to emulate the software and hardware environments for CPU architectures such as ARM. The success rate on other CPU architectures also will be improved using



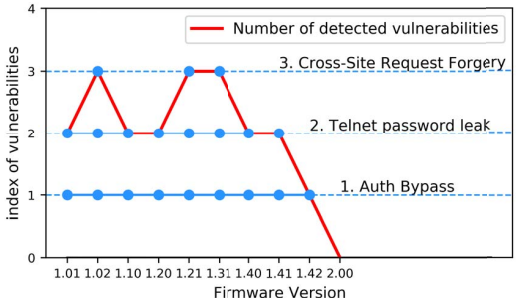
(a) first IoT device



(b) second IoT device



(c) third IoT device



(d) fourth IoT device

Fig. 6. Vulnerability improvement of the IoT devices.

an increased variation of hosting environments. The proposed framework automates the flow of simulation and analysis, an automated scheme to identify and record the reason for the failure of the emulation will be pursued in future work.

V. Conclusion and Future work

In this paper, we proposed a unified scheme which could help with the management of IoT devices at user environments. Based on the operational functionality of the IoT devices at hand, we can provide a different level of security information and operation support for improving the security status of the IoT devices. We expect the proposed scheme can help to improve the security situation of IoT devices, especially for the legacy devices that do not have some advanced firmware updating architecture implemented.

A. Future work

Due to the firmware image set we have at hand, the measurement study is performed mainly on the 32-bit CPU architectures, analysis for other 64-bit CPU architectures such as MIPS64, ARM64, is left as future work. We still lack some efficient means to emulate the software and hardware environments for CPU architectures such as ARM. The success rate on other CPU architectures also will be improved using an increased variation of hosting environments. The proposed framework automates the flow of simulation and analysis, an automated scheme to identify and record the reason for the failure of the emulation will be pursued in future work.

In the current framework, the user has to input the firmware version to the system. However, manual operation may lead to unpredictable errors during this process. An automated mechanism to retrieve the devices information and firmware version information will significantly improve the user experience. Depend on the functionality the IoT devices support, automation of this process will be pursued in two directions. First, the login pages of an IoT device often contains identifying information of the device; information retrieval based on text processing of the login page could be a feasible approach to the acquisition of device information. Second, some vendors provide the APIs which can interact with the IoT device; we can make use such APIs to obtain the device information.

References

[1] P. Fraga-Lamas, T. M. Fernández-Caramés, M. Suárez-Albela, L. Castedo, and M. González-López, "A review on internet of things for defense and public safety," in *Sensors*, 2016.

[2] A. D. Joseph Bradley, Christopher Reberger and V. Gupta, "Internet of everything: A $4.6 trillion public-sector opportunity," 2013, https://www.cisco.com/c/dam/en$_us/about/business - insights/docs/ioe - public - sector - vas - whit - paper.pdf[Online; lastaccesseddate : 11 - April - 2019].

[3] V. K. Mikhail Kuzin, Yaroslav Shmelev, "New trends in the world of iot threats," September 2018, https://securelist.com/new-trends-in-the-world-of-iot-threats/87991/ [Online; posted 18-September-2018].

[4] R. Metz, "Mit technology review," August 2013, https://www.technologyreview.com/profile/rachel-metz/ [Online; last accessed date: 11-April-2019].

[5] "Internet security threat report," April 2016, https://www.symantec.com/content/dam/symantec/docs/reports/istr-21-2016-en.pdf [Online; last accessed date: 11-April-2019].

[6] S. Ozawa, T. Ban, N. Hashimoto, J. Nakazato, and J. Shimamura, "A study of iot malware activities using association rule learning for darknet sensor data," *International Journal of Information Security*, 2019, to appear.

[7] B. Moran, M. Meriac, H. Tschofenig, and D. Brown, "A firmware update architecture for internet of things devices," April 2019, https://tools.ietf.org/pdf/draft-ietf-suit-architecture-05.pdf.

[8] NIST (National Institute of Standards and Technology), "NVD (National Vulnerability Database)," https://nvd.nist.gov.

[9] The MITRE Corporation, "CVE (Common Vulnerabilities and Exposures)," https://cve.mitre.org/index.html.

[10] M. Muench, D. Nisi, A. Francillon, and D. Balzarotti, "Avatar 2: A multi-target orchestration platform," in *Proc. Workshop Binary Anal. Res.(Colocated NDSS Symp.)*, vol. 18, 2018, pp. 1–11.

[11] D. D. Chen, M. Woo, D. Brumley, and M. Egele, "Towards automated dynamic analysis for linux-based embedded firmware." in *NDSS*, 2016, pp. 1–16.

[12] Rapid7, "metasploit," https://www.metasploit.com.

[13] threat9, "Exploitation Framework for Embedded Devices," https://github.com/threat9/routersploit.

[14] Y. Wang, J. Shen, J. Lin, and R. Lou, "Staged method of code similarity analysis for firmware vulnerability detection," *IEEE Access*, vol. 7, pp. 14 171–14 185, 2019.

[15] Y. Shoshitaishvili, R. Wang, C. Hauser, C. Kruegel, and G. Vigna, "Firmalice-automatic detection of authentication bypass vulnerabilities in binary firmware." in *NDSS*, 2015.

[16] T. Laskos, "Web application security scanner framework," https://www.arachni-scanner.com/.

[17] S. Bennetts, "Owasp zed attack proxy," *AppSec USA*, 2013.

[18] "w3af - Open Source Web Application Security Scanner," http://w3af.org/.

[19] J. Chen, W. Diao, Q. Zhao, C. Zuo, Z. Lin, X. Wang, W. C. Lau, M. Sun, R. Yang, and K. Zhang, "Iotfuzzer: Discovering memory corruptions in iot through app-based fuzzing," *Proc. 2018 NDSS, San Diego, CA*, 2018.