Software-based Analysis of the Security by Design in Embedded Devices

Pierre-Marie Junges, Jérome François, Olivier Festor
University of Lorraine, Inria, LORIA, France
Email: firstname.lastname@loria.fr

***Abstract*—The growth of embedded devices like IoT or networking devices makes them major targets for attackers in the Internet. They are known to face security issues because of their bad design and/or configuration. In this paper, we propose a systematic method to evaluate the security of an embedded device. It relies on a firmware analysis to extract relevant information about its software composition. Based on our large IoT database, our work aims at providing a global and long-term (10 years) analysis of the security by design of firmwares and of the awareness and versatility of vendors in regards to security issues.**

I. Introduction

Nowadays, embedded devices are omnipresent in personal homes, public spaces and in companies. They are composed of many different kinds of Commercial-Off-The-Shelf (COTS) devices including networking devices like routers or access points and, recently, Internet of Things (IoT) devices that contribute to the large growth of deployed embedded devices.

The IoT eco-system is a tremendous playground for attackers because many of them are exposed remotely without appropriate protection [1]. As IoT devices (smart home devices, cameras, sensors...) need connectivity, networking equipments like routers, gateways or access points are generally considered as part of their eco-system and also considered in research studies about security of IoT and all of them are so grouped under the term of Embedded Devices [2], [3]. Indeed, networking devices may also be impacted by the same type of attacks as shown in the case of the Mirai botnet [4].

The work presented in this paper aims at assessing the security of embedded devices including both IoT and networking devices. Several research studies evaluated the security of embedded devices. On one hand, massive scans over Internet combined with fingerprinting techniques is helpful to reveal a snapshot of the global level of exposure of specific alive devices [5], [6]. On the other hand, analyzing the firmwares of a large set of devices gives insight about the level of security by design.

We propose an hybrid approach based on static analysis to gather software components of firmwares and partial execution/emulation of selected binaries. The objective is twofold: evaluating the potential exposure of devices by analyzing server programs packaged within a firmware and assessing the device's security level through a mapping of vulnerabilities associated with these programs. Unlike other works aiming at scanning vulnerabilities in binaries [7]–[9] or source code [10], our approach solely considers software names, versions and functionalities.

Our contributions are threefold:

- an analysis framework to collect firmware images from vendors' websites, unpack and analyze them is designed and implemented,
- metrics to estimate the potential exposure to security risks of embedded devices are defined and,
- the technique is applied to 6935 devices released between 2009 and 2019. Our objective is to discuss the evolution of firmware design over 10 years rather than giving an instantaneous snapshot.

The rest of the paper is structured as follows. Section II introduces the related work. In Section III our security assessment technique is introduced. In Section IV, we describe our analysis framework. The results of our analysis are detailed in Section V followed by concluding remarks in Section VI.

II. Related Work

Firmware analysis has been largely used to discover security flaws and is based on two main techniques: static and dynamic analysis. In [11], the authors presented a dynamic analysis framework acting like a software proxy between an emulator, performig the instructions, and a real physical device executing the related I/O operations. Firmadyne [2] uses the QEMU [12] emulator and searches for known vulnerabilities.

An hybrid firmware analysis framework was proposed in [7], [10] that features port scanning, a credential analysis [10] and a vulnerability assessment of the emulated web pages and the web service source code [7], [10].

In [9], the authors present a technique to discover hard-coded authentication bypass backdoors. The authors of [3] extract security-related information (*e.g.,* RSA private keys, passwords) and discovered unknown vulnerabilities in 693 firmware images out of the 32,000 initially downloaded. A deep-learning based framework is described in [13] and compared embedded functions in binary code to a learning set of known vulnerable methods.

Unfortunately dynamic analysis does not scale [3] because of the difficulty to correctly emulate the hardware resources. Our approach leverages an hybrid analysis, both static and dynamic. Because it is exclusively based on software meta information (*e.g.,* name, version), the dynamic analysis is very lightweight as we do not need to emulate the full functionalities of a program. Our goal is not to detect new security flaws in the source or binary code but to analyze the software composition. We thus rely on CVEs (Common Vulnerabilities

and Exposures) similarly to [8]. In [8], the authors leveraged binary and ASCII signatures to match versions of zlib and OpenSSL libraries in printer firmwares. Our work is not device-specific and focuses on server programs.

In [14], Internet-wide scans were performed to extract RSA and DSA keys from TLS handshakes and the private keys were reversed if an insecure pseudo random number generator was used. In [15], [16], authors discovered that half a million devices had their factory default credentials settings active. Other researchers leverage fingerprinting techniques to detect the presence of exposed industrial systems [6] or IoT [5].

Those techniques allow to assess the global exposure of deployed devices in Internet and are complementary to a firmware analysis. Our work aims at evaluating the security of embedded devices but on a different perspective, by evaluating the potential security risks induced by the firmware design.

III. Security assessment method

A. Problem definition

Finding evidence of security threats in the original designed firmware is our main objective. In [1], the authors provide a comprehensive list of security threats related to IoT and among them, weak programming practices (firmware released with known vulnerabilities) and unnecessary open ports. Similarly, OWASP published the top 10 IoT 2018[1] which also highlights vulnerabilities and risks of IoT, in particular *top 2: insecure Network Services* (unneeded services that can be exploited by attackers) and *top 5: use of insecure or outdated components*.

Our approach aims at assessing an individual firmware in regards to these two issues. The objectives are to:

1) infer the level of potential exposure to attacks by examining server programs in the firmware which are known to be exploited by attackers (*e.g.,* SSH, HTTP servers);
2) assess the level of security risk by design by extracting software components of the firmware and mapping them to known vulnerabilities.

B. Hybrid firmware analysis

Properly running a firmware image is challenging because it often needs access to hardware resources (*e.g.,* /dev/nvram, /dev/gpio) that cannot be properly emulated. However, as explained in III-A, we are interested in information regarding the exposure of the device linked to the design of its firmware program. To reach our objectives, we rely on information related to software present in the latter.

Indeed, because a device may run any software available in its firmware, information such as the software versions can be mapped to known vulnerabilities using existing databases. In this paper, we considered the CVEs provided by the MITRE.

For instance, CVE 2016-7406 published in 2017 is about *dropbear* SSH server with a version up to *2016.73* hence, firmwares released after 2017 that include a *dropbear* binary version up to *2016.73* are by design vulnerable and we can conclude that such products were released by the vendors with a known vulnerability.

Hence, vulnerable COTS devices can be detected by comparing their release and CVEs publication dates.

As illustrated in the previous example, required information about software is limited to software names and versions. In addition, it might be required to check functionalities of the programs since some vulnerabilities are specifically related to a different alternative of the same software version depending on optional functionalities which have been packaged in the binary. To gather all the required information, two complementary methods are used:

- Static analysis: the firmware is unpacked and the list of programs present in the file system are retrieved. We can infer the names of the programs with their version numbers that sometimes appear in their names also.
- Partial execution: software is executed to only get the *help* message, which contains information, such as the version or the functionalities, and is generally accessible (*i.e.,* even if the program cannot work properly) without full firmware emulation.

In the remainder of this paper, we considered a software as an executable running a service.

C. Assessment metrics

For each device, having the firmware allows to extract the software information explained before, which is leveraged to infer the device exposure to security risks.

Assuming a set of devices D and set of programs P considered as sensitive, $\forall d \in D$ we denote d_sw the set of programs (retrieved by the static analysis) of the device d. The potential level of exposure, $exp(d)$, of a device $d \in D$ is the number of sensitive programs it contains:

$$exp(d) = |d\_sw \cap P| \tag{1}$$

The more services a device deploys, the more vulnerable the device might be. Thus, by combining the type of devices and services found, an overview of the level of exposition for each type can be deduced.

The vulnerability security assessment of a device d is calculated for a given software, $s \in d\_sw$ (using static analysis and partial execution). In the following equations, ver_s^d and ver_s^{last} are the version of s in d and the latest version of s when d was released respectively. $date(v)$ is the release date of version v.

The first metric to evaluate the security of a device is its freshness regarding the software version it uses. The use of former version of programs might be considered as an indicator for security issues. For example, authors in [17] revealed the use of ten years old Linux kernel versions in embedded devices.

In our case, the freshness is formally defined as follows, *i.e.* the number of versions that have been released since the version included in the firmware:

$$\delta^d_{\#version}(d, s) = ver_s^{last} - ver_s^d \tag{2}$$

[1]https://owasp.org/www-pdf-archive/OWASP-IoT-Top-10-2018-final.pdf

TABLE I: Services investigated

services	software
DNS	bind8, bind9, dnsmasq, tinydns, knotd, nsd, maradns, deadwood, pdnsd, posadis, pdns, unbound, yadifad
FTP	ftpd, vsftpd, uftpd, uftp, bftpd, proftpd
HTTP	httpd, nginx, boa, lighttpd, hiawatha, mongoose, thttpd, jjhttpd, minit_httpd, uhttpd, cherokee, jetty, monkey, hfs, navi, shttpd
NTP	ntpd, openntpd, sntp, sntpd, ntimed, ntpsec, chronyd
RPC	rpc.statd, rpcd
SNMP	snmpd, net-snmp, nagios, prometheus, zabbix, icinga, cacti
SSDP/UPnP	ssdp, upnpd, miniupnpd, minissdpd, minidlna, ushare, mediatomb, gmediaserver, gerbera
SSH	dropbear, openssh, apache mina, copssh, teleport, wolfssh
TELNET	telnetd, utelnetd

Also, this metric is quantified as a number of days:

$$\delta^d_{dver}(d,s) = date(ver^{last}_s) - date(ver^d_s) \quad (3)$$

For the sake of clarity, the versions are considered as numbers in the equations. In practice, they are generally in different formats (*e.g.* 1.42b), so the function to compute $\delta^d_{\#version}(d,s)$ and $\delta^d_{dver}(d,s)$ are adapted accordingly.

However, these indicators are approximate as an updated version may not fix security issues and a former version can still be secure (without vulnerabilities) during many years. Therefore, relying on vulnerabilities that are known to be effective is more fair. The following metric, called *vulnerability delay*, $\delta^d_{dvuln}(d,s)$, represents the number of days between the release date of a device d and the first vulnerability publicly disclosed related to the particular software s:

$$\delta^d_{dvuln}(d,s) = date(ver^d_s) - date(cve_{ver^d_s}) \quad (4)$$

where $date(cve_{ver^d_s})$ is the publication date of the first CVE related to version ver^d_s of the investigated program. It is worth to mention that this metric can only be computed for devices with vulnerable software. It must then always be evaluated in regards to the total number of vulnerable devices.

Thus, only high and critical CVEs are considered. CVE criticality is evaluated by a standardized score called Common Vulnerability Scoring System [18] (CVSS), and, because our interest is focused on devices connected to the Internet, we only consider CVEs with a network Attack Vector (AV).

For all the metrics aforementioned, the lower their value, the better. Freshness metrics are always positive. A device is considered secure by design when its vulnerability delay is negative meaning that the first vulnerability was published after its initial release.

D. Services investigated

Our analysis focuses on the services and associated software listed in Tab. I. We investigated SSH, TELNET and HTTP because they are often used as entry points by attackers [19]. According to security reports [20], [21] reflection and/or amplification attacks are often performed using the following protocols, DNS, NTP, RPC, SNMP, SSDP/UPnP. Finally, because some embedded devices may store private data (*e.g.,* camera footage), we also considered the FTP protocol.

IV. Framework design

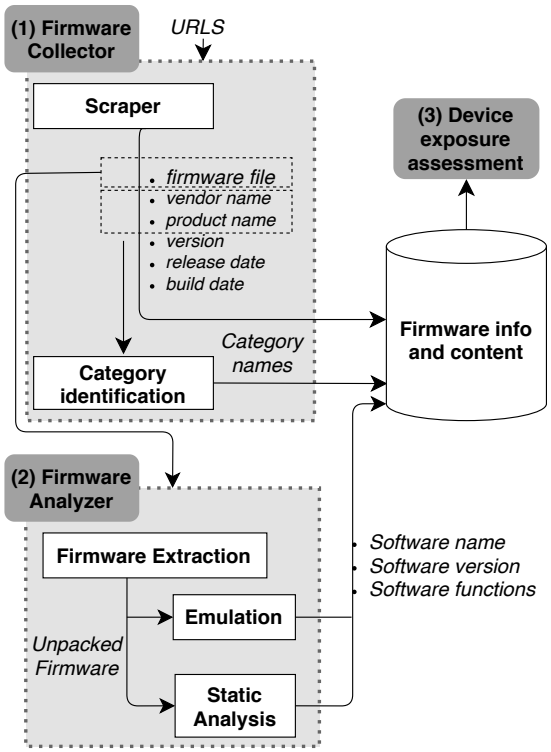


Fig. 1: Firmware analysis framework

In this section, we describe the operations performed by our firmware analysis framework summarized in Fig. 1. The starting point is a list of URLs provided by the user corresponding to the initial webpages from where the firmware collector (1) crawls websites to find firmware files. This initial stage done with the scraper allows to extract different information about a firmware such as the product name. Also, each firmware file is profiled by the type(s) or categorie(s) of device it is related to (see section IV-A1). Relying on the downloaded firmware file, the second component, the firmware analyzer (2) detailed in section IV-B, is in charge of unpacking it and extracting software information by executing the different binaries with emulation or, if not possible, by static analysis. The result is a database containing meta-information about the firmwares and the programs they include. This knowledge is leveraged to assess the level of exposure to security risks in the last phase (3) based on metrics defined in section III-C.

A. Firmware collector

1) Scraper and device categorization : The first step of the firmware collector downloads firmware images and extracts useful metadata (also from the webpage where the link to the firmware is provided). Tools have been already proposed to collect firmware images from the Internet [2], [3] but were often not maintained over time due to website and URL address changes. Because our work focuses on firmware images from well known IoT manufacturers like D-Link, TP-Link or NETGEAR, we created a scrapper based on *Scrapy* [22] to search and download content from vendors' websites.

Our analysis focuses on small and embedded devices, including IoT and networking ones. For fine-grained analysis, distinguishing the different types of device is helpful to observe if differences exist in the by design level of exposure to the attacks. In order to define device categories, we rely on Google searches based on the product name (firmware metadata) and vendor name (provided by the user with the URL as inputs of the scraper) and on finding relevant keywords in the top 5 returned pages. These keywords have been manually defined as device categories (router, switch, cpl, vpn, firewall, camera, sensor, network video recorder...).

Then, from the 2529 distinct product names contained in our database described in Section V-A, at least one category has been assigned to 2505 product names (99.05%).

B. Firmware analyzer

The role of the firmware analyzer is to extract information about software included in the firmware. Our objective is to find software related to the services we selected in Section III-D and to extract the names and versions.

To unpack firmware files, *binwalk* [23] is leveraged. The content of each unpacked firmware is analyzed to find software binaries in UNIX-like directories (*e.g.,* /bin, /sbin, etc.).

Given the list of extracted binaries, our framework searches for the ones having a name related to a program defined in Tab. I. However, those files could be empty or even not executable. Hence, we focused our analysis exclusively on executable files because we can extract 1) relevant information such as the executable version or its functionalities from their executions, 2) any string present in the binary code (*e.g.,* hardcoded credentials, version) using text string searching.

Because our system runs on a x86_64 Ubuntu system, an executable e compiled for another CPU architecture such as MIPS, ARM or PPC could not be executed. Hence, we relied on the QEMU user emulator [12] to emulate the correct CPU architecture, infered from the ELF (Executable and Linkable Format) header of e, and perform the execution.

Naively executing a dynamic executable using the QEMU emulator fails because the dynamic linker attempts to load shared libraries from the host lib directories such as /lib or /usr/lib. However, because the file system of the unpacked firmware is stored, we can set the *-L* option of the QEMU emulator to the expected root environment. To identify software information, a set of commands and scripts to analyze the output are defined manually beforehand. It generally consists in running the executable with the help or version option (*e.g., -h*, *-v*) and searching for particular patterns in its output.

However, some executions may fail because 1) the CPU architecture requested may not be supported by QEMU (*e.g.,* ubicom) or 2) accesses to inexistant hardware components such as */dev/nvram* or */dev/gpio* were requested. Hence, our framework can detect such failures and will automatically switch to the text string searching alternative (using the *strings* GNU tool).

TABLE II: Most represented device types

rank	**all products**	**investigated products**
1	router (network, 29.84%)	router (network, 35.43%)
2	switch (network, 18.66%)	access point (network, 15.14%)
3	access point (network, 11.98%)	bridge (network, 13.22%)
4	camera (IoT, 8.77%)	switch (network, 9.56%)
5	bridge (network, 8.13%)	camera (IoT, 6.46%)

V. Evaluation

A. Dataset

Our scraper downloaded 9106 unique firmwares (152 GB) associated to 12,047 products (*i.e.,* devices) from ten manufacturers: Asus, Belkin, D-Link, Edimax, Linksys, NETGEAR, Reolink, Tp-Link, Trendnet and Ubiquiti. In the rest of this paper, we defined a device as a couple <product name, product version>. Thus, the one-to-many relationship between a firmware and devices is due to multiple devices sharing the same firmware.

Then, we successfully unpacked 5896 (64.75%) firmware images associated to 8301 (68.90%) devices. We explained the unpacking failures by the presence of encrypted firmware images or infinite decompression loops. Thus, because the firmwares images were downloaded in May 2020, we focused our analysis over the decade between 2009 and 2019. Hence, 6935 (57.5%) devices remained for analysis among which 86.08% are networking devices, 12.62% IoT devices and 1.30% undefined (*i.e.,* others) devices.

Additionally, Tab. II highlights the five most widespread device types present in the entire dataset and in the set of the investigated devices.

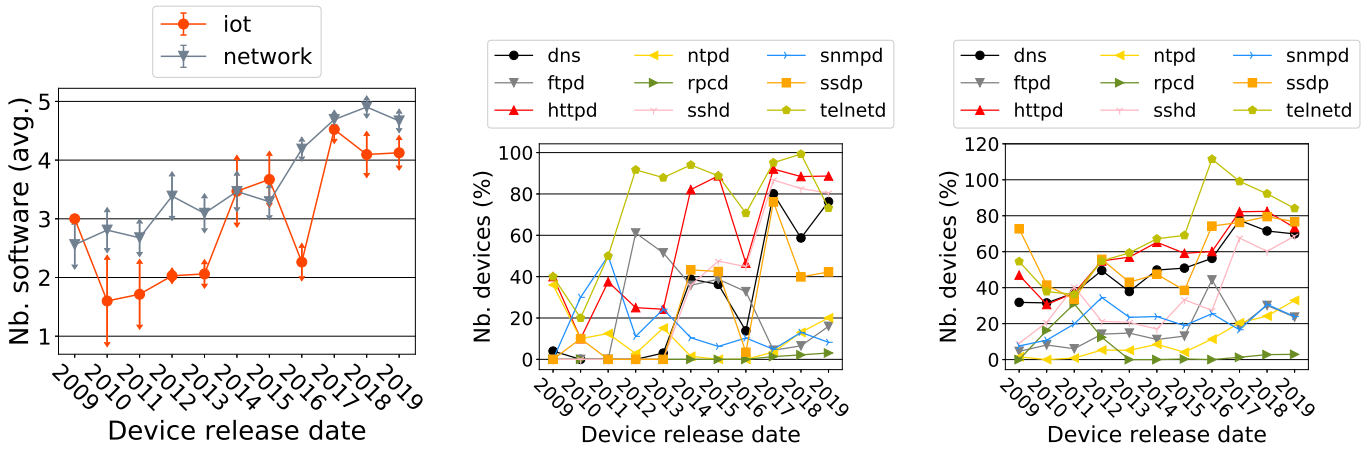
B. Service exposure overview

Our first objective is to assess the level of exposure of the devices which is evaluated by counting the numbers of executables listed in Tab. I included in a firmware.

Fig. 2a) shows the average number of software found in IoT and networking devices with a 95% confidence interval. For IoT devices, the number of available services raised from 3 in 2009 to about 4 in 2017. However, we observed a decline in 2016 to 2.26 services. In 2016, camera devices are over-represented, they account for 87.93% of the IoT devices in our database instead of 51.20% in average over the global period. Therefore, cameras does not follow the general trend of hosting more services. For networking devices, the services available nearly doubled reaching 4.66 in 2019.

In general, more services have been deployed within embedded devices over the years. The devices continued to be more exposed and so can be potential entry points for attackers.

Identifying what are those services is helpful to have a finer view on the exposure level. Between 2012 and 2019, the four most represented executables are related to TELNET, HTTP, DNS and SSDP/UPnP. In 2017, 1816 devices embed a runnable TELNET server whereas TELNET was already known to be exploited by attackers. In Fig. 2b) and Fig. 2c), telnet-related software remains vastly present in both IoT devices and networking devices even if their presence slowly declines, in 2016, for networking devices and, in 2018, for



a) Average number of software per device b) Service distribution in IoT devices c) Service distribution in networking devices

Fig. 2: Overview of the level of device exposure

IoT. As highlighted in Fig. 2c), in 2016, 110% of the 835 network-related devices have TELNET server software. While this looks erroneous, it is due to devices embedding multiple, most of the time two, programs acting as TELNET servers. In the vast majority, this is observed for routers having both *utelnetd* and *telnetd*. We also observed this phenomenon with *ssdpd* and *upnpd* in routers and access points. From a security perspective, keeping a minimal set of libraries and software is a good practise to avoid increasing the potential attack surface, which is antagonist to what we observed.

In contrast, in both Fig. 2b) and Fig. 2c), we observed a large increase of Secure Shell (SSH). The adoption of this secure protocol for remote connections highlights a better consideration of security by device vendors, that is moderated by the general use of TELNET in parallel as discussed before.

Also, because remote shell connections enabled by TELNET and SSH have been used to create large-scale botnets (Psyb0t [24] in 2009, LightAidra/Aidra [25] and Carna [26] in 2012 or Mirai [4] in 2016), vendors could have restricted their uses. Unfortunately, they are still often present.

Moreover, in Section V-D, we showed that SSH software are not exempt from security issues.

C. HTTP servers analysis

In complement to the remote connections provided by SSH or TELNET, it is often possible to interact with a device through its web interface. As shown in Fig. V-B, HTTP is one of the most deployed services with 4644 (66.96%) products able to run an HTTP server.

Fig. 3 shows that vendor-specific httpd-software (*proprietary*) were present in 36.52% until 2015 followed by a notable decrease to 16.17%. We assume this is due to the growth of the IoT market and in general home devices, that include also networking devices. To keep the pace with the market demand and reduce the time-to-market and the cost in parallel, reusing well-maintained third-party software and libraries was preferred. This also avoids proprietary software to be vulnerable to specific attacks (*e.g.,* CVE-2018-16119,

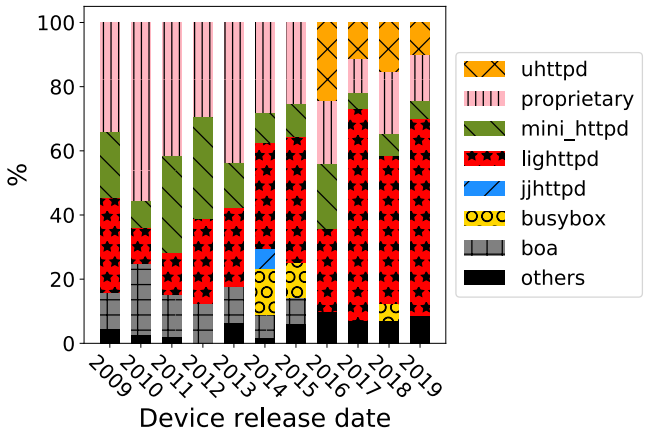
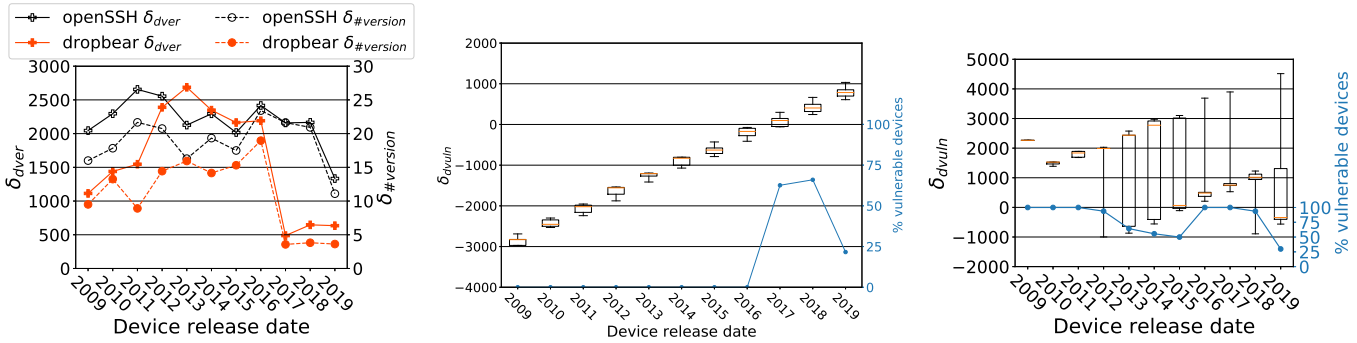


Fig. 3: HTTP server programs distribution (*others* groups all programs with less than 5%)

CVE-2017-14250 or CVE-2016-1555 etc.), with the overhead cost of patching this induces. Thus, proprietary software is slightly replaced by open-source software and so automatically benefit from security fixes provided by the community.

Increasing from 29.55% (13 devices) in 2009 to 65.93% (1018 devices) in 2017, *lighttpd* was found in 55.1% of the devices in total making it the most represented HTTP server.

Considering *lighttpd*, CVEs with a high criticality are mostly related to mods installed alongside *lighttpd*. However, we found that 391 (15.37%) devices, released after 2013, are vulnerable to the CVE-2013-4559 targeting versions before 1.4.33 (published on 27/09/2013) independently of the installed mods. This vulnerability allows privilege escalation. On and after 2018, 1002 (39.39%) devices, released on and after 2018, have a version between 1.4.36 (26/05/2016) and 1.4.49 (11/03/2018) and are thus impacted by 2 CVEs: (1) CVE-2019-11072 allows remote attackers to cause denial of service but is disputed (no agreement on the vulnerability) whereas the other, (2) CVE-2018-19052 is a path traversal



a) Freshness of software versions b) *dropbear* vulnerability delay c) *OpenSSH* vulnerability delay

Fig. 4: Security assessment of exposed SSH software

flaw related to a particular mod_alias (impact limited to the software with the mod installed). Therefore, the large use of *lighttpd* in embedded devices make them less prone to major vulnerabilities over the years because related vulnerabilities are either more mod-specific or not fully confirmed.

D. Secure Shell (SSH) servers analysis

As described in V-B, the proportion of Secure Shell (SSH) increased over the years. Globally, 3720 (53.64%) devices can run a SSH server. Based on our hybrid analysis (Section III-B), 3647 (98.04%) were identified: *dropbear* (3286: 90.10%) and *OpenSSH* (363: 9.95%). So, a very few software are used for SSH and two devices include both *dropbear* and *OpenSSH*.

Freshness provided by equations (2) and (3) is depicted in Fig. 4a). There is a clear drop after 2016, especially for *dropbear*. Hence, the firmwares released after these dates tend to be more up-to-date but there is still a gap of 3.66 versions (equivalent to around 600 days) and 17.88 versions (equivalent to around 1500 days) with the latest version for *dropbear* and *OpenSSH* respectively. It is known that botnets [4], [26], [27] were relatively active in 2016 and results in a bad reputation for embedded devices which may explain this change.

Considering *dropbear* (Fig. 4b)), no devices were released with a known vulnerable version until 2016. δ_{dvuln} is negative during this period because of a vulnerability published in 2017 (CVE-2016-7406). A second vulnerability was published the same year (CVE-2017-9078). As a result, around 65% of the devices released in 2017-2018 with *dropbear* were vulnerable but, in 2019, the percentage of vulnerable devices decreased to 21.68% (181 devices). We concluded that vendors rapidly updated *dropbear* software in their firmwares according to the CVEs published. It is worth to mention that δ_{dvuln} still increases linearly in 2017-2019. Thus, the remaining vulnerable devices are still impacted by the first vulnerability.

For *OpenSSH* (Fig. 4c)), the observations are quite different. In the first years (until 2019), almost all devices come with a vulnerable version even if there was an attempt to have more secure versions as shown by a decrease of δ_{dvuln} in 2010. The percentage of vulnerable devices highlights two waves of major updates in the device firmwares. The first one, between 2012 and 2015, results in almost 50% of device released with a secure version (OpenSSH_5.5p1 or OpenSSH_6.0p1). This explains also the large diversity in δ_{dvuln}. Unfortunately, CVE-2015-5600 targeting *OpenSSH* versions up to 6.9 was published in 2015 and vendors were slow to upgrade their software. The second wave, starting in 2018, does not seem to be global due to a high diversity in δ_{dvuln} but shows that some devices are really up to-date while others still exhibit old vulnerabilities. Indeed, in 2019, 70.13% of the devices included a secure version (OpenSSH_7.4p1 or OpenSSH_7.5p1).

In a nutshell, most vendors do not actually update their *OpenSSH* versions according to the security threats published in CVE whereas those relying on *dropbear* are more reactive to include secure versions.

VI. Conclusion

In this paper, we performed a security assessment on 6935 embedded devices released between 2009 and 2019. Our assessment aimed at identifying the level of security by design of embedded devices but, most of all, at outlining the major trends over the last ten years regarding vendors' awareness and versatility towards security issues. While we noticed the adoption of a secure protocol, SSH, for remote connections, TELNET surprisingly remains and is probably used as a backup solution. Because open-source software can benefit from a large community support and so rapid security fixes, vendors rely on them more frequently. However, they prefer to use the same version for many years which unfortunately leads to have large delay between the release of a patched version and its inclusion in firmware.

Many changes in firmware design (*i.e.,* newer software version, type of software deployed) have been observed between 2015 and 2017. It is concurrent to major botnets, for example IoT_reaper/IoTroop[2] targeting IoT devices. Hence, those changes seem to be driven by active threats rather than by proactive design decisions.

In future work, we plan to perform an Internet-wide scan of embedded devices to weight our assessment in regards to the distribution of deployed devices.

[2]https://research.checkpoint.com/2017/{iotroop-botnet-full-investigation/, new-iot-botnet-storm-coming/}, last accessed 13/10/2020

REFERENCES

[1] N. Neshenko, E. Bou-Harb, J. Crichigno, G. Kaddoum, and N. Ghani, "Demystifying iot security: An exhaustive survey on iot vulnerabilities and a first empirical look on internet-scale iot exploitations," *IEEE Communications Surveys Tutorials*, vol. 21, no. 3, pp. 2702–2733, 2019.

[2] D. Chen, M. Egele, M. Woo, and D. Brumley, "Towards automated dynamic analysis for linux-based embedded firmware," 01 2016.

[3] A. Costin, J. Zaddach, A. Francillon, and D. Balzarotti, "A large-scale analysis of the security of embedded firmwares," in *23rd {USENIX} Security Symposium ({USENIX} Security 14)*, 2014, pp. 95–110.

[4] M. Antonakakis, T. April, M. Bailey, M. Bernhard, E. Bursztein, J. Cochran, Z. Durumeric, J. A. Halderman, L. Invernizzi, M. Kallitsis, D. Kumar, C. Lever, Z. Ma, J. Mason, D. Menscher, C. Seaman, N. Sullivan, K. Thomas, and Y. Zhou, "Understanding the mirai botnet," in *26th USENIX Security Symposium*, 2017.

[5] "Towards automatic fingerprinting of iot devices in the cyberspace," *Computer Networks*, vol. 148, pp. 318 – 327, 2019.

[6] J. François, A. Lahmadi, V. Giannini, D. Cupif, F. Beck, and B. Wallrich, "Optimizing Internet Scanning for Assessing Industrial Systems Exposure," in *7th International Workshop on TRaffic Analysis and Characterization*, Paphos, Cyprus, 2016. [Online]. Available: https://hal.inria.fr/hal-01371674

[7] A. Costin, A. Zarras, and A. Francillon, "Automated dynamic firmware analysis at scale: a case study on embedded web interfaces," in *Proceedings of the 11th ACM on Asia Conference on Computer and Communications Security*, 2016, pp. 437–448.

[8] A. Cui, M. Costello, and S. J. Stolfo, "When firmware modifications attack: A case study of embedded exploitation," in *Network and Distributed System Security Symposium, NDSS*, 2013.

[9] Y. Shoshitaishvili, R. Wang, C. Hauser, C. Kruegel, and G. Vigna, "Firmalice - automatic detection of authentication bypass vulnerabilities in binary firmware," in *Network and Distributed System Security Symposium (NDSS)*, 2015.

[10] V. Visoottiviseth, P. Jutadhammakorn, N. Pongchanchai, and P. Kosolyudhthasarn, "Firmaster: Analysis tool for home router firmware," in *2018 15th International Joint Conference on Computer Science and Software Engineering (JCSSE)*, 2018, pp. 1–6.

[11] J. Zaddach, L. Bruno, A. Francillon, and D. Balzarotti, "Avatar: A framework to support dynamic security analysis of embedded systems' firmwares," in *NDSS*, 2014.

[12] F. Bellard, "Qemu, a fast and portable dynamic translator," in *USENIX Annual Technical Conference, FREENIX Track*, 2005.

[13] P. Sun, L. Garcia, G. Salles-Loustau, and S. Zonouz, "Hybrid firmware analysis for known mobile and iot security vulnerabilities," in *2020 50th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, 2020, pp. 373–384.

[14] N. Heninger, Z. Durumeric, E. Wustrow, and J. Halderman, "Mining your ps and qs: detection of widespread weak keys in network devices," 08 2012, pp. 35–35.

[15] A. Cui and S. J. Stolfo, "A quantitative analysis of the insecurity of embedded network devices: Results of a wide-area scan," in *Proceedings of the 26th Annual Computer Security Applications Conference*, ser. ACSAC 10. New York, NY, USA: Association for Computing Machinery, 2010, p. 97106. [Online]. Available: https://doi.org/10.1145/1920261.1920276

[16] A. Cui, Y. Song, P. Prabhu, and S. Stolfo, "Brave new world: Pervasive insecurity of embedded network devices," 09 2009, pp. 378–380.

[17] P. Weidenbach and J. vom Dorp, "Home router security report 2020," Fraunhofer Institute for Communication, Information Processing and Ergonomics, Tech. Rep., 2020.

[18] "Nvd - vulnerability metrics," available at https://nvd.nist.gov/vuln-metrics/cvss.

[19] C. Kolias, G. Kambourakis, A. Stavrou, and J. Voas, "DDoS in the iot: Mirai and other botnets," vol. 50, pp. 80–84, 01 2017.

[20] "Alert (ta14-017a) - udp-based amplification attacks," US Department of Homeland Security CISA, Tech. Rep., 12 2019, available at https://us-cert.cisa.gov/ncas/alerts/TA14-017A.

[21] "Cyber threat intelligence report," available at https://www.netscout.com/threatreport.

[22] "A fast and powerful scraping and web crawling framework," available at https://scrapy.org/.

[23] "Binwalk the #1 open source firmware extraction tool," available at https://www.refirmlabs.com/binwalk/.

[24] "Psyb0t," available at https://en.wikipedia.org/wiki/Psyb0t.

[25] "Lightaidra, open source code," available at https://github.com/eurialo/lightaidra.

[26] T. Krenc, O. Hohlfeld, and A. Feldmann, "An internet census taken by an illegal botnet: A qualitative assessment of published measurements," *SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 3, p. 103111, Jul. 2014. [Online]. Available: https://doi.org/10.1145/2656877.2656893

[27] E. Bertino and N. Islam, "Botnets and internet of things security," *Computer*, vol. 50, no. 2, pp. 76–79, 2017.