Fraunhofer FKIE

Home Router Security Report 2022

Fraunhofer
FKIE

FRAUNHOFER-INSTITUT FÜR KOMMUNIKATION, INFORMATIONSVERARBEITUNG UND ERGONOMIE FKIE

Home Router Security Report 2022

Johannes vom Dorp
René Helmke

November 2022

Executive Summary

In this report, 122 currently available third-party home routers are statically analyzed based on their firmware image to identify trends in adoption of security-related software features. Similar to our previous report from 2020 [1], we chose seven vendors selling their devices on the European retail market.

For all selected vendors, we assume relevance based on perceived public brand recognition and home router portfolio breadth. All of them offer a variety of budget, mid-tier, and high-end home router devices – targeting a variety of different retail market customer groups. Although we do not claim our corpus to be representative for any kind of market distribution, we still believe our findings to give an important insight on router security.

Analysis data was consolidated on 2022-03-31 by downloading publicly accessible home router firmware images from all vendor homepages, excluding any end-of-life products.

To gain insight into adoption of security best practices we analyzed the firmware regarding four security related questions:

- When did the device receive its last firmware update?
- What operating system versions are used and how many known vulnerabilities can we heuristically attribute to these binary findings?
- Are the included software components compiled with binary hardening techniques?
- Does the firmware use hard-coded login credentials - and if so, are they easy to guess?

Our fully automated analysis method was able to correctly unpack and analyze 109 of 122 firmware images. All 109 devices contained an embedded Linux operating system, which is the base of two of our features. The remaining 13 devices are unpacked incorrectly and thus omitted from further analysis.

In comparison to the 2020 analysis, we observe a positive trend regarding the time since last firmware update – which can be an indicator of improving device hygiene. However, as we report varying Linux kernel usage patterns, we show that discovered operating system versions vary greatly between vendors. Despite the observation that excessively old kernels are less frequently used, we still encounter a significant amount of version 2.6 Linux kernels and report that more than 70 % of all discovered kernels reached End-of-Life (EOL) status. Furthermore, publicly documented kernel vulnerabilities were assigned using a newly developed heuristic that reduces false positives [2]. This leads to a drastic decrease in reported vulnerability candidates, though most devices are still assigned at least one potential vulnerability marked as critical and all at least one marked as high. These candidates should be further examined by vendors and independent security experts. Due to the changed heuristic, the results for this metric are not comparable to the 2020 report.

We observe that the adoption of binary hardening techniques is about equal to 2020. However, a positive trend is that far fewer hard-coded credentials could be found in current firmware images. Although the total number of such hard-coded credentials is smaller, the number of weak passwords remains comparable to 2020. About 10 % of devices contain a weak password

we were able to crack.

The mixed results underline once more that security features are adopted differently in different devices and across vendors. Although some positive trends can be seen compared to 2020, there is still room for improvements. Especially Linux versions that do not receive security support anymore should be replaced. Additionally, modern embedded architectures have support for binary hardening techniques, which can be included in build processes. Finally, all hard-coded credentials should be checked on necessity and on their value. At least passwords that are set to *admin, password or root* and other guessable values should be replaced by stronger passwords if it is necessary to store them.

The FACT[1] software used for this analysis is open source and can be used to independently reproduce the results in this report. All vendors under analysis were provided with the results and report before release and were able to assess them. The results of ensuing vendor discussions are given in a separate appendix along with our views on the vendor positions. In this document, all vendors were pseudonymised and their order scrambled in result presentations.

[1] `https://fkie-cad.github.io/FACT_core/`

Contents

1 Introduction

Still a minor topic in IT security when compared to malware and vulnerabilities on general purpose systems and services, vulnerabilities in networked embedded systems such as COTS network hardware are increasingly gaining public notice. Both criminal activities such as Mirai-based IoT botnets and state-sponsored attacks on critical infrastructure expose the large attack surface brought by these simple network participants. In 2020, we conducted a first survey on the security level on home routers as a widespread and security-critical embedded system, finding a high number of issues with commonly distributed hardware of that device class [1].

The motivation for the report remains: Home routers are both accessible directly through the internet and serve as a gatekeeper between devices on the local network — some of which might have differing levels of trustworthiness. The ongoing COVID-19 pandemic and its projected long-term impact on work-from-home practices additionally stress the information security responsibilities of these devices.

As a fresh look on the state of security in third-party home routers we present a study similar to [1], with the latest[1] firmware for 122 routers currently sold by a set of seven vendors that appear on European markets. Our updated and improved methodology still builds upon FACT[2], which is able to extract the components from the firmware images, identify the included software and run a number of analyses on each individual component of the firmware. Based on the analysis we infer four security-related aspects:

- When were the devices updated last time?
- What operating system versions are used and how many known vulnerabilities can we heuristically attribute to these binary findings?
- Which exploit mitigation techniques do the vendors use? How often do they activate these techniques?
- Are there any hard-coded login credentials? Can these be guessed easily?

Similar to the findings of 2020, the results, as shown in the following sections, indicate areas of improvement for many devices in the data set that could be extracted correctly. While the employed static methods do not serve definitive proof of exploitable vulnerabilities, our findings are to be understood as indicators of potential security issues. Guessable hard-coded credentials and heuristically attributed vulnerabilities in operating system kernels are candidates that require further careful, technical, and individual verification through vendors or third-party security analysts.

The number of hard-coded login credentials is significantly lower than in the 2020 report. Looking specifically at the number of guessable passwords, the number is comparable though, showing no improvement in this most critical aspect. Meanwhile, release cycles have shortened, especially in regards to the maximum age of the latest update. In 2020, the oldest firmware was more than five years old and the average age was more than one year. For the current analysis, half of the devices received an update within the last eight months and the oldest firmware is

[1] Firmware samples were collected on 2022-03-31.

[2] `https://fkie-cad.github.io/FACT_core/`

slightly younger than three years. Similarly, the age of the Linux versions in use was improved. While nearly a fourth of all identified Linux versions was still in the very outdated 2.6 stream, a general shift towards versions of the 4.x streams is observable. Still, more than 70 % of kernels reached EOL status. With the attribution of known vulnerabilities based on the commonly used CVE database, our heuristic methodology was largely updated and improved to allow more reliable results with lower false-positive rates, as we document and demonstrate in [2]. While this yields a much lower number of attributed critical vulnerabilities per device, the remaining candidates are, in comparison to our 2020 attribution methodology, of higher significance for subsequent manual verification. No device to which our new method is applicable was free of potentially applicable CVEs with a *high* graded vulnerability score. The binary hardening, based on exploit mitigation techniques, shows no improvements compared to 2020 as well. Only NX and PIE are employed by more than 50 % of the observed executables and there is only one vendor that applies Canaries on more than 5 % of its executables.

In the following, we give a detailed description of the data set and the individual analyses. We compare our results to the 2020 study where possible, as some methods have changed. The seven included vendors were randomly pseudonymised using the letters ABCDEFG in our discussions and result presentation. However, they remain consistent between results.

2 Evaluation Corpus

On the 31st of March 2022 we collected the latest firmware releases from seven vendors based on the firmware that was available from their support pages and the devices that were currently promoted on the vendors websites. All chosen vendors run well-recognized retail market brands and offer a variety of budget, mid-tier, and high-end home routers – targeting various different customer groups. Similar to 2020 we provide information on the set of analyzed firmware on GitHub[1].

For vendor selection, we created a list of prevalent vendors and distributors that sell, rent, or lend their devices not necessarily exclusively, but also to European end users. Here, we attribute relevance by perceived public brand recognition and home router portfolio. Furthermore, choices are influenced by sample accessibility and technical limitations: We require access to firmware images in order to analyze them and our tools must be able to unpack them, which is not possible when they are encrypted and we do not know the secret key.

Aside of well-known retail market brands, we identify Internet Service Provider (ISP) devices as an important sample group, as they are an intuitive choice for many end users. While many vendors publicly serve firmware through their support pages, this is not the case for various ISP devices. Unfortunately, many vendors that supply OEM routers to ISPs, but also sell their systems independently, e.g., Huawei and Lancom, either encrypt their devices or do not make firmware downloads available. To address this problem, we had various discussions with large ISPs to achieve non-disclosure agreements so we can include missing samples and gather decryption keys. Unfortunately, in all cases we either could not obtain an NDA or subsequently did not receive firmware samples in time for this report. To avoid selectively picking analyzable samples that are only available for a limited subset of ISPs and their corresponding devices, we decided to exclude them from the data set to finish this report in a timely feasible manner. Furthermore, we had to exclude large vendors like Huawei and Lancom due to missing or encrypted samples. Consequently, the corpus does not claim to offer an accurate and representative market view of German, European or international router sales or application. Nevertheless, we continue our efforts with ISPs and vendors to add more firmware to future analysis corpora.

In total we analyzed 122 firmware samples. The analysis is set up based on FACT, a firmware analysis tool that provides full firmware analysis automation from component extraction to attribution of potential vulnerabilities. With FACT we were able to correctly extract 109 of the 122 firmware images for further analysis. For the CVE attribution an additional post-processing, which is published in [2], and an alternative data source are applied. These are shortly described in the related analysis section.

A short observation regarding operating systems and CPU architectures found in the corpus is given in the following sections.

2.1 Operating Systems

Current home routers offer a complex set of features that includes internet access, wireless networking, routing and configuration. Common additional functionality includes network storage, DECT-device and home automation device connection. In effect, vendors commonly develop their firmware based upon software development kits (SDKs) provided by the chip vendors to

[1] `https://github.com/fkie-cad/embedded-evaluation-corpus/blob/master/2022/FKIE-HRS-2022.md`

reduce redundant, hardware dependent work. The most common operating system (OS) provided through these SDKs still is embedded Linux, which we found on 109 of 122 devices at least once. Some subcomponents like the internet access modem on the devices are complex enough that they use an additional Linux OS. Other operating systems that we found were omitted from further analysis as our methods do not apply to them.

2.2 CPU Architectures

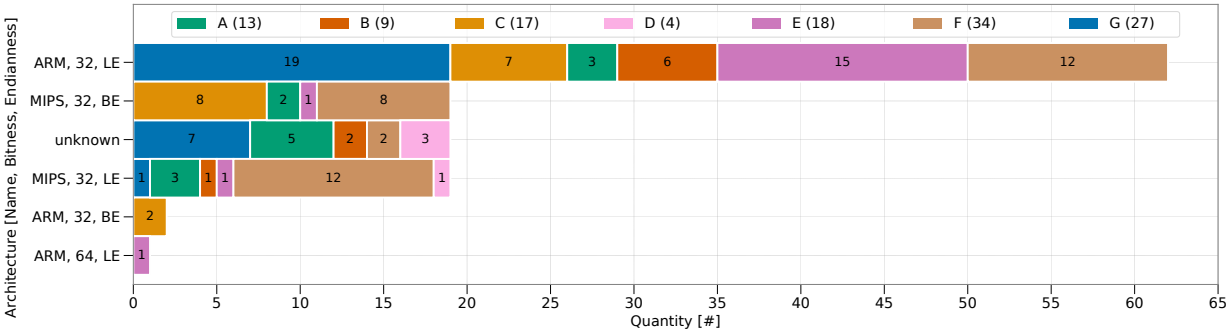

Figure 2.1: Distribution of CPU Architectures. Letters A-G represent pseudonymised vendors.

The CPU architecture does not have a direct impact on security, aside from possible limitations in regards to binary hardening techniques. Our analysis shows that all devices where we could identify a distinct architecture use either an ARM or a MIPS CPU. ARM and MIPS are the most common CPU ISAs in complex embedded devices and offer a full set of features necessary for security purposes. Among these, little-endian ARM was the most commonly observed architecture. Interestingly, one device already uses a 64-bit CPU, which is not yet commonly adopted in embedded devices.

In comparison to our 2020 data set, we observe a slight shift towards ARM architectures (from 45% in 2020 to 53% in 2022). Furthermore, there is a decline of MIPS usage (from 47% in 2020 to 31% in 2022). However, we also report a growing amount of ISAs that remain unknown due to unpacking and analysis errors, which increased from 8.7% to 15.6%. A reason is obfuscation in firmware images, e.g., encryption or newly introduced container formats that our tools can not handle yet. Thus, it is possible that sample sizes for both ARM and MIPS clusters might vary in reality.

3 Evaluation

We investigate four security relevant features of the firmware images in our corpus. The analysis is done in FACT[1] with post processing to remove false analysis artifacts. Post processing steps are unique to each feature and are described in the subsection of each feature. Based on the resulting data we generate statistics and charts to visualize the state of each feature for all devices and each vendor independently. The four security relevant features are:

1. **Age of latest release.** Does the vendor maintain all of their products regularly? In other words, how often do they fix issues?
2. **Operating System.** How old are the OS versions powering the devices? How many critical vulnerabilities are known for these versions?
3. **Binary hardening.** Do the vendors activate exploit mitigation techniques?
4. **Login Credentials.** Are there any hard-coded credentials that might allow unintended access to the device?

The following subsections give detailed information for each feature. The subsections are structured as follows:

- A **description** of the feature, the underlying analysis and its impact on security,
- our **findings** in regards to the feature, both in summary for all devices and for each vendor separately,
- a statement on reliability of our results including **limitations** and verification strategies and
- a **comparison** to the findings in the 2020 report.

For some analyses, an additional subsection includes information on feedback the vendors provided.

3.1 Age of Latest Release

Description

Frequent firmware releases provide proof that a vendor maintains their devices in terms of usability, features, stability and security. Most router firmware is built heavily upon open source components for the operating system, cryptographic libraries and network services. Thus, independent of the internally developed firmware components, the collection of components face a steady supply of updates and patches. Especially for stability and security, this implies that practically every firmware can receive frequent updates to make use of the newest version of its components. By extension, this also implies that outdated firmware necessarily misses updates to these software components, many of which are critical to security. As analysis of software versions shows that components are often not up to date at the time of a firmware release anyway, conversely, frequent firmware updates do not imply improved security. But it does provide the opportunity to close the most critical security issues in a timely manner and it provides the

[1] `https://github.com/fkie-cad/FACT_core`

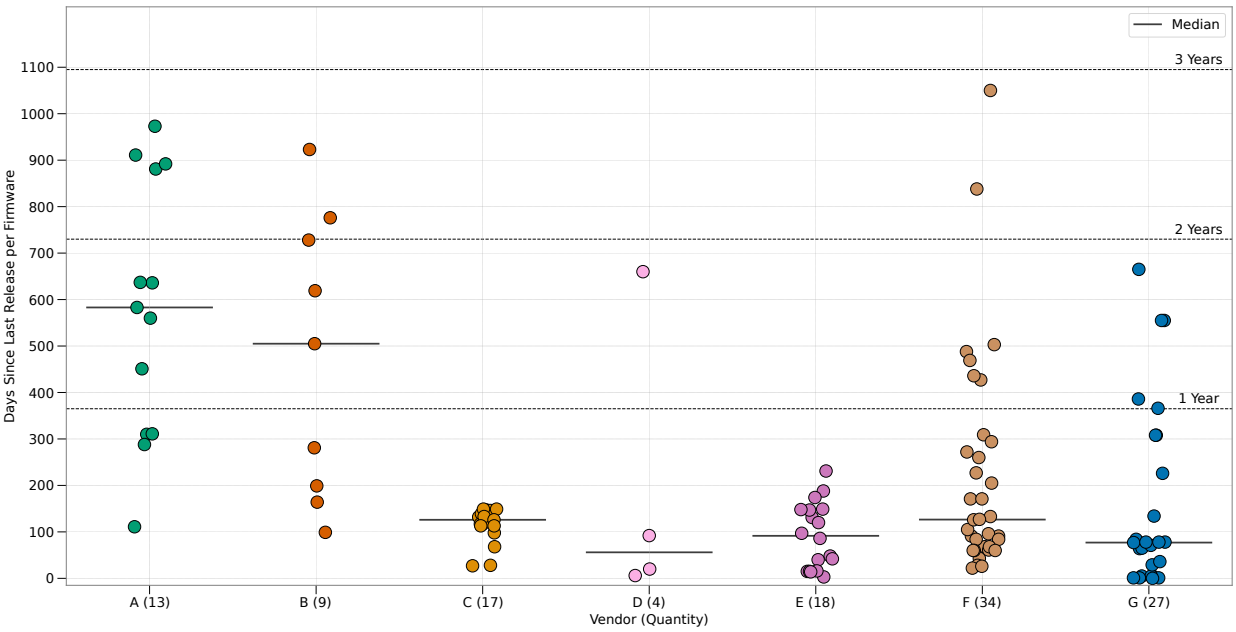

Figure 3.1: Days since last release before 31st March 2022.

aforementioned proof of active maintenance by the vendor.

The analysis section provides a chart of the age of the most recent firmware updates in days since the data collection deadline on the 31st of March (see Chapter 2) for each device. No post processing was needed for this analysis. The **Limitations** section goes into some detail regarding the reliability of inferences that can be taken from this statistic.

Findings

Figure 3.1 shows the results of our age analysis. Of the 122 routers in our data set, 95 received a firmware update in the last 365 days, leaving 27 devices with a firmware older than one year. Eight of the 27 updates are older than two years, with none being older than three. The maximum age is 1050 days (about two years and 10.5 months). The mean age is 230 days, meaning that the average device received an update in the last eight months. In that time frame (13th August 2021 to 31st March 2022) 114 CVEs have been attributed to the Linux kernel, eight to OpenSSL and dnsmasq and three to wpa_supplicant and hostapd, all software components that are used by most Linux-based router firmware.

Five of the seven vendors show an arguably reasonable update cycle with at least half their devices receiving an update within 127 days. The other two vendors have a median age of more than 500 days for their devices, showing a distinct split in the two vendor groups. Reviewing these numbers suggests that frequent updates are possible even for lower price products. One device is listed at multiple points of sale for below 30 € and had received an update six days before the data collection. Thus even though release cycles have shortened in recent years an even higher frequency, e.g. aided by state of the art continuous deployment pipelines, should be regarded as a necessary step to ensure the security of devices.

Limitations

There are two primary limitations to the significance of the results for this feature: First, the age is a snapshot that does not tell about the time between the current release and the previous and upcoming releases. One edge case to represent this limitation is a router that received an update on the 31st of March specifically. The age of zero days is not representative of the length of the firmware update cycle for the device and the mean of 157 days for all devices of the same vendor is probably a better guess for when to expect the next update. While for some vendors a history of previous releases is public, making it possible to calculate the frequency between versions, this is not possible for multiple vendors in our data set that only publish the most recent firmware. Thus, this statistic cannot be applied broadly. Second, vendors do not update all software components in every firmware release. While it can be assumed that highly public vulnerabilities such as log4j will be covered in a security update after such an event, each update might only partially address the sum of newly discovered bugs and vulnerabilities.

An additional limitation is the possibility that vendors do not supply all firmware updates through their websites. It could be the case that minor updates are only published *over-the-air* directly to the router or it could be that a vendor stops publicly offering the firmware altogether leaving only the last public version online. We did not observe these two cases for the vendors in scope.

Comparison

As briefly noted on the **Findings** paragraph the release cycles of router firmware seem to have shortened since the last data set was generated in 2020. The most notable difference is the lower maximum ages observed in this report. In 2022 no firmware older than three years was found. In 2020 there were 10 devices where that was the case, one having not received an update for more than five years. One reason these outliers might have vanished is that device vendors could have pruned end-of-life devices from their websites more diligently. That said, the median, a number less affected by a small number of outliers, is also much lower this time, reinforcing the observation.

The three vendors that were positively noted in 2020 show good results in this report, again. Additionally, this year two other vendors join the group of these three vendors in showing reasonable update latencies for most devices.

3.2 Operating System

Description

The operating system is an essential part of router firmware. It provides key elements like resource management, networking, protocols, drivers, and hardware abstractions to the vendor applications that ultimately implement the functionality customers get in touch with. Linux[2] is open source, well maintained, customizable, and supports many hardware components, making it a popular choice among vendors. Development costs are reduced with large communities working on both the operating system and its software ecosystem.

As for most important software projects, Linux frequently receives reports of new software vulnerabilities. A recent example is Dirty Pipe[3], which can be exploited to gain full system compromise due to the kernel having unrestricted privileges as the trust anchor of security policy enforcement. Thus, keeping kernels up-to-date is pivotal to decrease risks of exploitation.

[2] `https://kernel.org`

[3] `https://lolcads.github.io/posts/2022/06/dirty_pipe_cve_2022_0847/`

Our analysis of the operating systems targets two aspects: First, we identify the operating system version of the Linux kernel to infer age and support status of the operating system. Second, we attribute which vulnerabilities of the Linux kernel might apply to analyzed firmware. The analysis makes use of the software detection functionality of FACT to find out which part of the firmware contains the Linux kernel and which version of the kernel is used. Which kernel streams still receive support and which reached End-Of-Life (EOL) can be inferred from the Linux Kernel Mailing List[4].

The attribution of CVEs is a more complicated task and received some research efforts since the release of the last report in 2020. In discussions upon the release of the last report, it was asserted that the method of simply attributing CVEs based on kernel version alone was not sufficient to paint a realistic picture of the actual attack surface of the devices under analysis. Mainly, two limitations were identified: Linux kernels are extremely heterogeneous as they are highly configurable so that vulnerabilities, especially those found in drivers, might only apply to a small subset of kernels. In addition, vendors can cherry-pick security patches from later releases of the used kernel stream or even more recent kernel streams to close vulnerabilities that should affect their kernel based on version.

The latter limitation is hard to bridge with static analysis, though a more thorough review of the topic is given in the **Limitations** paragraph below. The former limitation was addressed through an improved attribution method presented in detail in [2]. The improved method identifies the components that are present in the kernel to select only the vulnerabilities that affect components, which are used. As the research paper shows, most vulnerability descriptions for Linux kernel CVEs in official CVE databases include which source code file is affected. This file reference can then be used to identify the affected component. The application of this new method in combination with kernel-to-CVE assignment of the National Vulnerability Database (NVD) that was also used in 2020 is marked as *File-based Matching* in the **Findings**. As an alternative for kernel-to-CVE assignment, a second data source was introduced compared to the last report. On *linuxkernelcves.com*[5] (LKC), an automated method to identify affected kernel versions based on patch commits is hosted. We include this data source in two additional statistics. First, simple matching of kernel version to vulnerabilities with the LKC assignment is marked as *Commit-based Matching*. Second, combining the LKC assignment with the new component based matching method is marked as *Combined*. Due to the limitations of the above mentioned heuristics, which are inherent to large-scale static analysis approaches, our CVE findings are to be understood as candidates that require further careful, technical, and individual verification through vendors and third-party security analysts.

Note that both CVE attribution methods are not applicable to all kernels. To identify the components included in a Linux kernel for *File-based Matching* we rely on optional metadata that can be removed from the kernel during compilation and thus is not always contained in the firmware [2]. On the other hand, LKC provides the necessary data for *Commit-based Matching* also only for some but not all Linux kernel streams. Consequently, we restrict our analysis of kernel-to-CVE assignment to those kernels for which both methods were applicable.

Finally, in this report we are only interested in CVEs that pose a certain degree of real-world risk, expressed through the Common Vulnerability Scoring System (CVSS)[6] v3.x. As we found that a considerable amount of Linux CVEs below a scoring of 7.0 are of rather theoretical nature, we chose to only inspect CVEs exceeding said threshold. Thus we report findings of the severity classes *High* ($\geq$ 7.0) and *Critical* ($\geq$ 9.0). Some old CVEs do not have a CVSS v3.x score. In these cases, we fall back to CVSS v2.0.

[4] `https://lkml.org/`
[5] `https://www.linuxkernelcves.com/`
[6] `https://nvd.nist.gov/vuln-metrics/cvss`

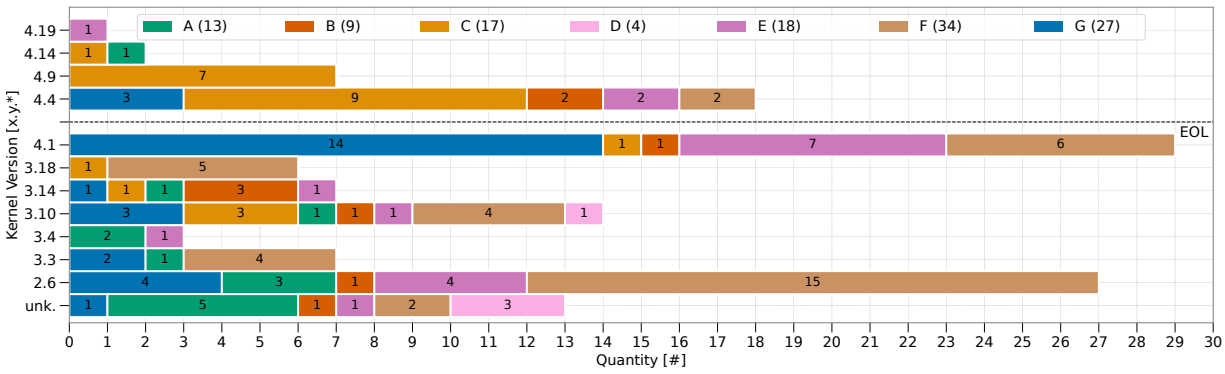

Figure 3.2: Linux Versions. Letters A-G represent pseudonymised vendors.

Findings

Figure 3.2 shows the Linux kernel versions our methods identify across all router images. Patch levels are clustered into *<Major>.<Minor>* for illustrative purposes only. In total we found 121 Linux kernels in 109 images. For the remaining 13 firmware images we identify no kernel version (denoted as *unk.*). As described in 2.1, complex systems can include multiple operating systems, explaining the firmware images with multiple Linux kernels.

The version range is from 2.6 to a single occurrence of 4.19. The horizontal dashed line in Figure 3.2 separates EOL kernels (below line) from versions that are still actively maintained (above line). Out of the 121 identified kernels, 93 do not receive any official updates anymore – roughly 77%. For reference, since the last 2.6 stream reached EOL, 1200[7] new Linux kernel CVEs were assigned. Even when removing CVEs that do not affect a given kernel based on configuration, the amount of effort necessary for cherry-picking patches for the remaining vulnerabilities has to be assumed to be very high.

With 27 instances the long obsolete 2.6 stream still makes up about a fifth of the found kernels. The oldest kernel in our data set is a single instance of 2.6.31 (Release: 2009[8]). The remaining findings have version 2.6.36 (Release: 2010[9]). Another large cluster in the set of EOL Linux versions is the Long Term Support stream 4.1 (deprecated 2018[10], 29 kernels).

As for actively maintained Linux kernels, note that the 4.4 stream, which was officially deprecated in Feb. 2022[11], is still included. This is because the Civil Infrastructure Platform continues development for 10 to 20 more years[12], as it is the first Super Long Term Support version. We argue that this is a reasonable and resilient choice, which five of the seven different vendors seem to adopt in different paces.

Figure 3.3 shows separate bar plots for each vendor to break down and compare kernel usage distributions. All vendors except for one primarily build their firmware on top of EOL Linux versions. However, there are small exceptions in our findings (1-3 kernels are still maintained).

Vendors that mark two ends of the usage spectrum are C and F. For vendor C, 17 out of 23 kernels are still in active development, with the majority of kernel versions being in Super Long Term Support. Vendor F is on the other end of the spectrum: With 15 out of 36 findings, F

[7] https://www.cvedetails.com/product/47/Linux-Linux-Kernel.html?vendor_id=33
[8] https://lkml.org/lkml/2009/9/9/357
[9] https://lkml.org/lkml/2010/10/20/409
[10] https://www.spinics.net/lists/announce-kernel/msg02259.html
[11] https://lore.kernel.org/lkml/16438771372402490kroah.com/
[12] https://wiki.linuxfoundation.org/civilinfrastructureplatform/cipkernelmaintenance

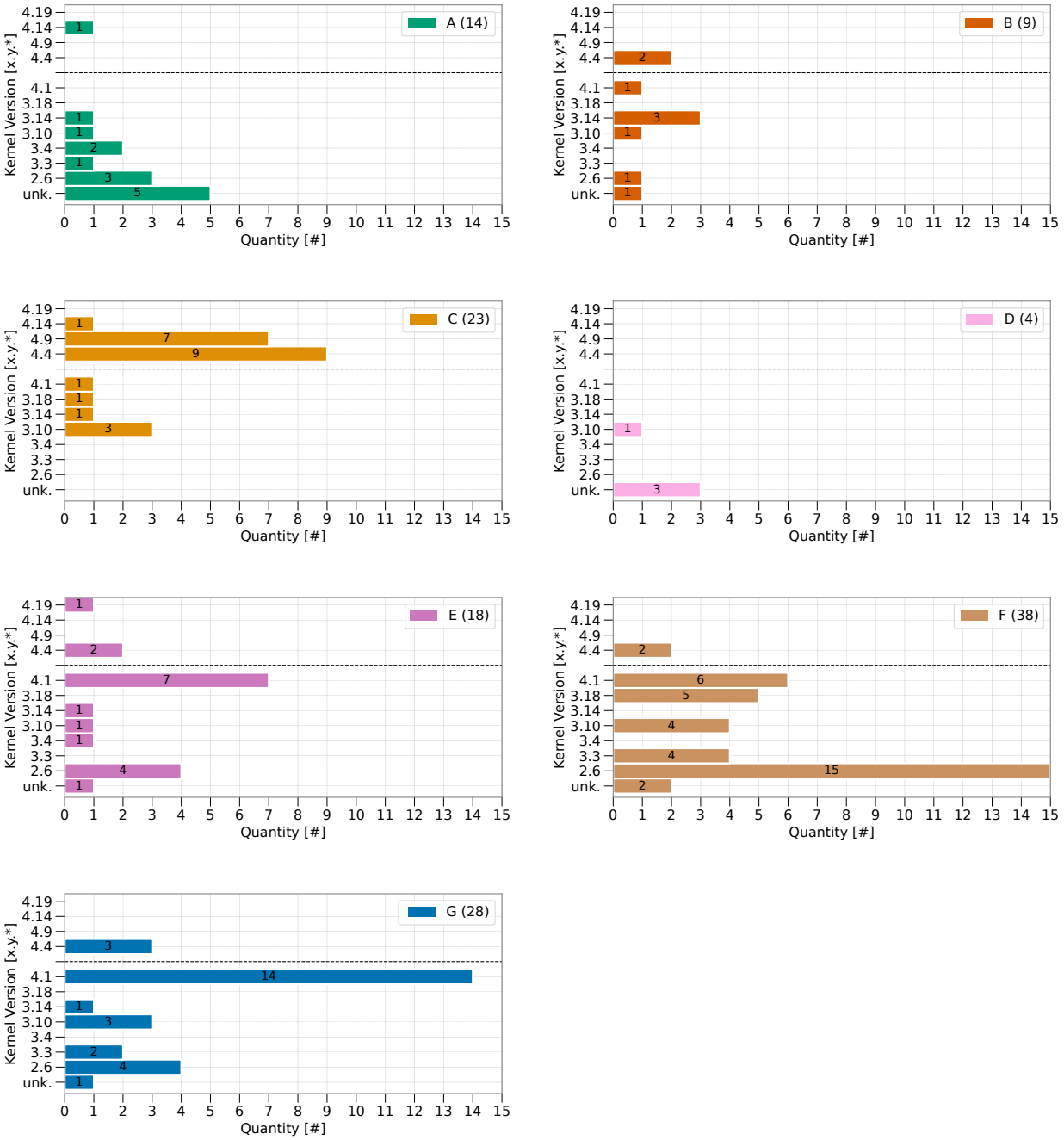

Figure 3.3: Linux Version Usage per Vendor. Letters A-G represent pseudonymised vendors.

represents the majority (56 %) of version 2.6 matches. Furthermore, there are 4 findings of version 3.3 – which is no Long Term Support release that got discontinued the same year it was introduced[13].

Next, we present our results on heuristic and static Linux kernel CVE matching as described in [2]. Methods applicability is limited by two factors: First, the Commit-based Matching data set does not support all version streams present in our firmware corpus. Second, our File-based Matching can only succeed when we find a kernel build configuration file. Thus, we limit our observations to the intersection of kernel findings analyzable by both methods.

[13] `https://lkml.iu.edu/hypermail/linux/kernel/1206.0/01162.html`

Out of the 121 Linux kernel samples we identified, our methods are applicable to 61, distributed across all considered vendors. Analyzed sample versions range from 3.10.104 to 4.19.151.

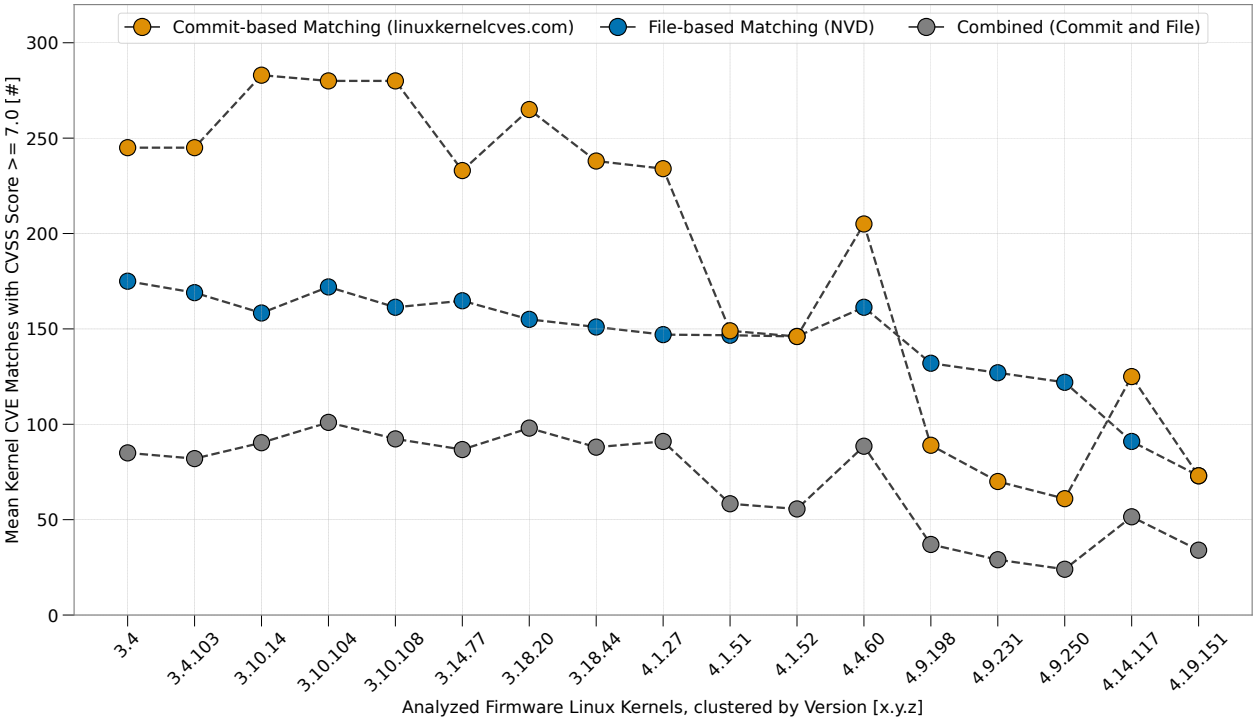

Figure 3.4: Number of heuristically attributed High Severity CVEs in Linux Kernel per Firmware Image

In a first step, we cluster all kernels across vendors by Linux version, and then calculate the mean count of positive CVE matches with a CVSS score $\geq$ 7.0. Figure 3.4 shows the results for Commit-based Matching, File-based Matching, and the combination of both. Note that we consider official patch levels and order kernels by version (left to right, ascending).

Both methods, including their combination, show a measurable and significant decline in CVE matches when newer kernel versions are used. E.g., the Commit-based Matching yields means of around 250 CVEs with CVSS $\geq$ 7.0 for kernel version 3.4.0, while for kernel version 4.19.151 the results report only 75. This is easily explainable: The older a kernel version, the more time there is to find vulnerabilities. However, the implications of this seemingly trivial observation are of high significance for router security: As of 2022, all analyzed versions in our data set that are left of the 4.4.60 marker on the X-axis in Figure 3.4 are EOL. Consequently, they do not receive official updates anymore.

In the next step, we cluster CVE findings by vendor and attribution method. Figure 3.5 shows the results for all heuristically attributed CVEs with a CVSS $\geq$ 7.0. Each point represents a kernel image, with one point each for the three matching methods. The colors correspond to the methods. Horizontal lines mark the medians per vendor and method.

We find evidence of possible applicability for CVEs with at least High severity in each kernel we analyzed, regardless of attribution method and vendor. The medians across all same-method findings are 149 (Commit-based), 147 (File-based), and 58 (Combined). Differences between vendors are observable in all matching methods. Most vendors have a high variance in the Commit-based method, but show more narrow distributions for the other two methods. Looking specifically at the combined method one vendor has a median of 37 High severity CVEs attributed to its kernels, while another has a median 89 High severity CVEs. This difference is more than 100 %.

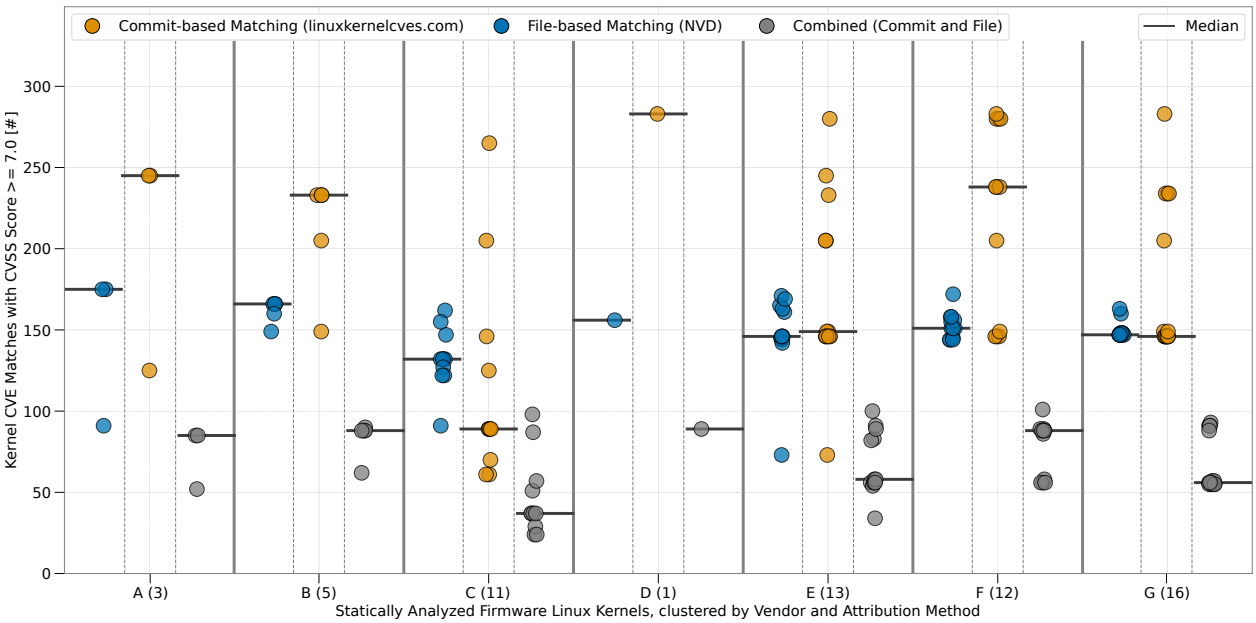

Figure 3.5: Number of heuristically attributed High and Critical Severity CVEs per Linux kernel, grouped in triplets of applied method per vendor. Letters A-G represent pseudonymised vendors

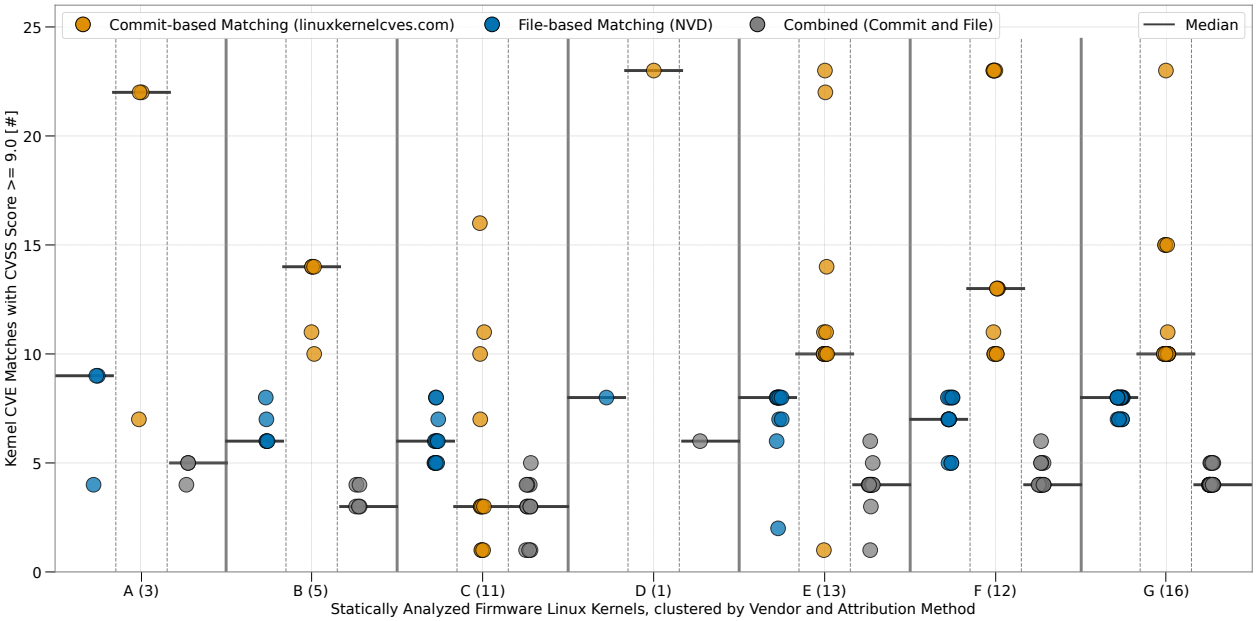

Figure 3.6: Number of heuristically attributed Critical Severity CVEs per Linux kernel, grouped in triplets of applied method per vendor. Letters A-G represent pseudonymised vendors

In figure 3.6, we remove all High severity CVEs and show that for each analyzed kernel and vendor, there are CVE findings of Critical severity (CVSS $\geq$ 9.0): For all vendors the count of possibly applicable CVEs is within the range of 1 and 10 measured by File-based and Combined techniques. The Commit-based Matching reports quantities between 11 and 23 for four of the vendors.

As the low CVE numbers of vendor C directly correspond to C using more recent kernel streams than most other vendors as shown in Figure 3.3, and Figure 3.4 corroborates the observation that newer kernel streams contain less known vulnerabilities, more vendors should try to migrate their development towards more recent kernel streams.

Limitations

Similar to the other presented analyses, the presented operating system detection and CVE attribution are limited by the common concerns of static analysis. The analysis is limited to the firmware image and does not take the runtime environment and hardware into account.

For Linux version detection there is no guarantee that our search patterns catch all versions correctly, which can lead to the possibility of mismatches. Furthermore, in firmware images where multiple Linux kernels are found we deliberately chose to make no distinction between main kernels and the ones running on subcomponents. Identifying the main kernel can be done by manual analysis in some cases while in other cases it is not feasible in scale. We derived selection methods across all 109 devices, i.e., by path or detected kernel version (prefer highest/lowest). These delivered inconsistent and ambiguous results, which is why we discarded them[14].

In terms of CVE attribution, we already discussed in our previous report [1] that there is a high probability of false-positive matches as CVE databases might miss information or provide incorrect data on vulnerable versions [2, 3]. To mitigate this limitation we diversified our analysis compared to 2020 to include both *Commit-* and *File-based Matching* techniques. Both techniques are static approaches that can gather evidence for CVE applicability, but do not pose ultimate proof – this is only possible through non-accessible vendor information or bug exploitation during device runtime. The drawbacks of File-based Matching are documented in our paper [2], and the drawbacks of Commit-based Matching can be found in the project's Readme file[15]. Thus, there is still a chance of false-positives and false-negatives, which is why we chose to not only report combined results. Also, take note that our heuristic attribution methods only consider 61 out of 121 kernels due to technical limitations. Here, method applicability selects analyzed kernels.

Finally, we have no insights on how and when vendors decide to cherry-pick or develop custom patches. Thus, false-positives are probable, as vendors are known to apply such patches for critical vulnerabilities.

Comparison

Compared to the 2020 results, vendors are still late in terms of meeting EOL deadlines and we still find a significant amount of 2.6.x kernels in our data set. The newer and still actively maintained kernel streams 4.9, 4.14, and 4.19 entered the stage in low quantities and the subset of version 4.1 findings grew significantly. We observe that the usage of the Super Long Term Support stream 4.4 grew from 7.5% to 16.5%. However, it is too early to say whether the 4.4 stream will be as popular among vendors in the future as the 2.6 stream was in the past.

As for known Linux kernel vulnerabilities, our methods changed drastically. Thus, there is no real comparability to the findings we reported in 2020.

[14] Highest and lowest choices miss semantics on accurately guessing the actual main kernel, and unpacking artifacts complicate selection where unpackers failed to correctly restore paths from binary images. For example, we found two kernels in a device where no meaningful path information could be extracted:
2.6.36 (File /261c593231aad31462fe8d11ca9a8a0fd506eee4e1416dd68548e5ec50ef5e45_77320105),
3.18.44 (File /0f954fd3e5cb23c21f6175fc23eb2402ed9620fd0a5c1505e6ccb90802e9b9af_5621713)

[15] https://github.com/nluedtke/linux_kernel_cves/blob/master/README.md

3.3 Binary Hardening

Description

The idea of binary hardening techniques, also commonly referred to as exploit mitigations, is simple: Insert additional checks into the program execution to prevent programming errors turning into security vulnerabilities. In theory, these techniques can even prevent exploitation of unknown vulnerabilities, although they cannot prevent every exploitation attempt. However, even in cases where binary hardening fails to prevent an attack, it often complicates exploit development, which in turn might give users a little more time for updating the firmware of vulnerable devices. There are many binary hardening techniques[16] suitable to protect embedded devices. Especially Linux-based devices can be protected easily, because the necessary tools are free and tested in practice for years.

We analyze the usage of four different binary hardening techniques that all can be enabled or disabled on a file-by-file basis. Our data shows the percentage of binary files per firmware sample that were compiled with the corresponding binary hardening technique enabled.

The analyzed binary hardening techniques are:

- **Non-Executable Bit (NX)** marks regions of the memory as non-executable. If an attacker wants to execute his own code, he has to store it somewhere in the victim's memory. The idea of NX is to mark those regions of memory as non-executable that should not provide executable code, while marking regions with executable code as read-only to prevent manipulation through an attacker. Depending on the processor architecture, this bit might also be called *XI* or *XN*.

- **Position-Independent Executable (PIE)** allows a program to be loaded at a random location in memory. If mitigation techniques like NX are enabled, an attacker needs to reuse legitimate program or library code already loaded into memory. With exploitation techniques like return-oriented programming (ROP)[17] an attacker might still achieve arbitrary code execution. However, to use ROP the attacker needs to know where to find the code in the victims memory. Since the operating system can randomize the code locations of position-independent executables, exploit techniques like ROP are much harder to implement. We consider dynamic shared objects (DSO) and relocatable files (REL), too. They are, by definition, position-independent.

- **RELocation Read-Only (RELRO)** protects the Global Offset Table (GOT)[18] against manipulation during program runtime. The GOT maps memory locations of functions from shared libraries or global variables, so that the program can find them. If an attacker can manipulate this mapping then he can redirect a legitimate function call to another function, which in turn simplifies implementation of exploitation techniques like ROP. RELRO marks the GOT as read-only after loading the program into memory to prevent manipulation by an attacker during runtime. There is a partial RELRO[19] mode protecting global variables only and a full RELRO mode protecting the whole GOT. In our analysis, we count partial mode and full mode as enabled.

- **Stack Canaries** mitigate buffer overflow attacks. In C, you have to reserve space in memory to store incoming data. If the program does not check if the incoming data is larger than the reserved space (called buffer), the incoming data may overwrite other data in

[16] `https://7h3ram.blogspot.com/2012/07/exploit-mitigation-techniques-on-linux.html`
[17] `https://en.wikipedia.org/wiki/Return-oriented_programming`
[18] `https://en.wikipedia.org/wiki/Global_Offset_Table`
[19] `https://ctf101.org/binary-exploitation/relocation-read-only/`

the memory. This can involve data affecting the program execution enabling an attacker to execute malicious code. The basic idea of canaries is to store a special byte sequence called canary on specific positions in memory. These sequences are checked for changes during runtime of the program, which can detect buffer overflow attacks for which the corresponding vulnerability does not provide the attacker with fine-grained control over the amount of overwritten memory. Note that this mitigation can still only detect some but not all buffer overflow attacks.

Findings

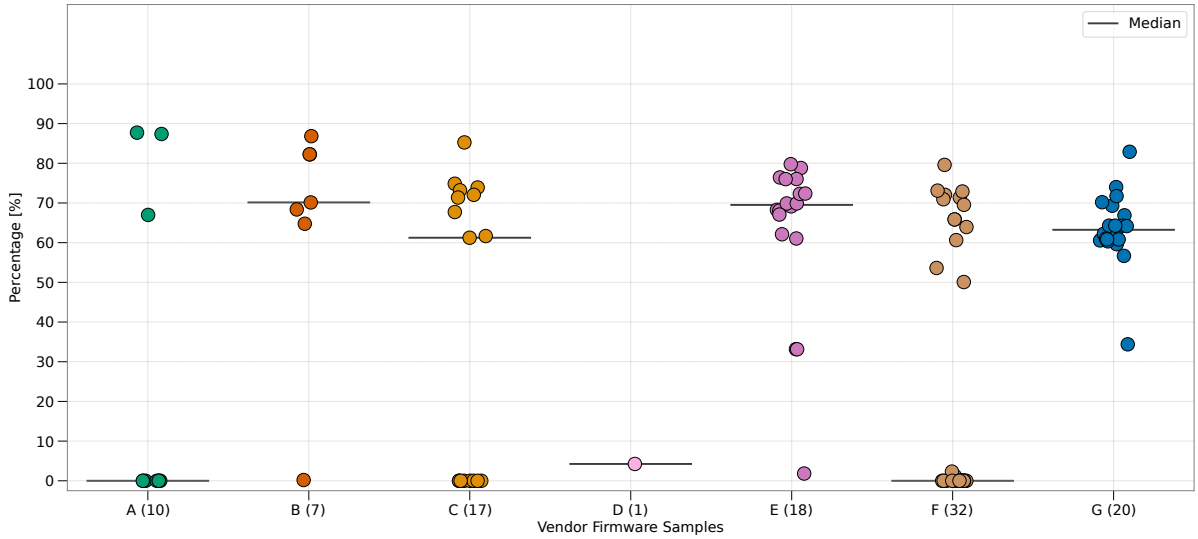

Figure 3.7: Percentage of Executables with NX Enabled per Firmware Image

The usage of the NX technique is quite common as can be seen in figure 3.7. When it is used, vendors enable it on the majority of binaries, although some exceptions persist.

A rather unexpected result is that in 37 firmware samples we found no or almost no usage of NX at all. Further investigation showed a strong correlation between no usage of NX and usage of the MIPS CPU architecture. One possible explanation could be that these devices are based on MIPS processors without support for this binary hardening technique. Since we did not analyze the actual processor types built into the analyzed devices, we could not check this theory. In first place, we saw few reasons why vendors would either turn off NX support during compilation or deliberately stick to processors without support for such a well-known hardening technique. However, in discussions with vendors, it was communicated that the choice of a processor architecture is a rather complex task, as criteria like technical interfaces, processor speeds, chip availability, and power consumption have to be considered.

Figure 3.8 shows that the usage of position-independent executables can be found in almost all devices, but to a varying degree. For example, while the majority of firmware samples from one vendor employ PIE on over 95% of executables, other vendors have PIE enabled for between 20% and 80% of all binaries for most devices, with the exact number varying wildly from firmware to firmware. We investigated the two cases of firmware with a reported PIE usage of 0% and identified these as cases of incorrectly unpacked firmware. Thus, the reported numbers for these two firmware images are incorrect.

In Figure 3.9 one can see a division between firmware where an effort was made to use RELRO, i.e. using RELRO on at least 50% of all binaries, and firmware with usage rates of less than 10%

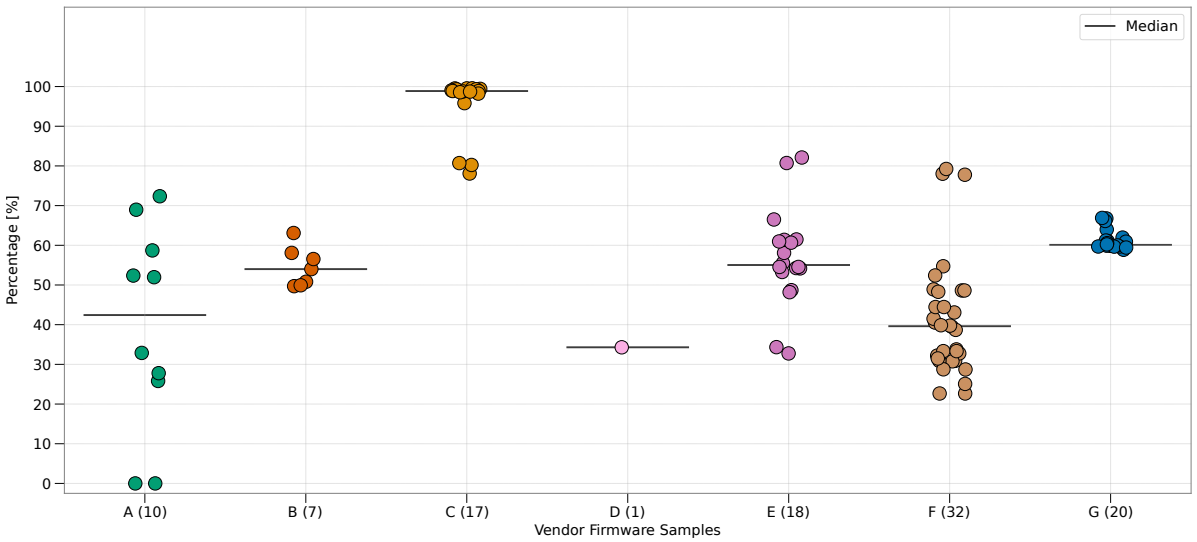

Figure 3.8: Percentage of Executables with PIE Enabled per Firmware Image

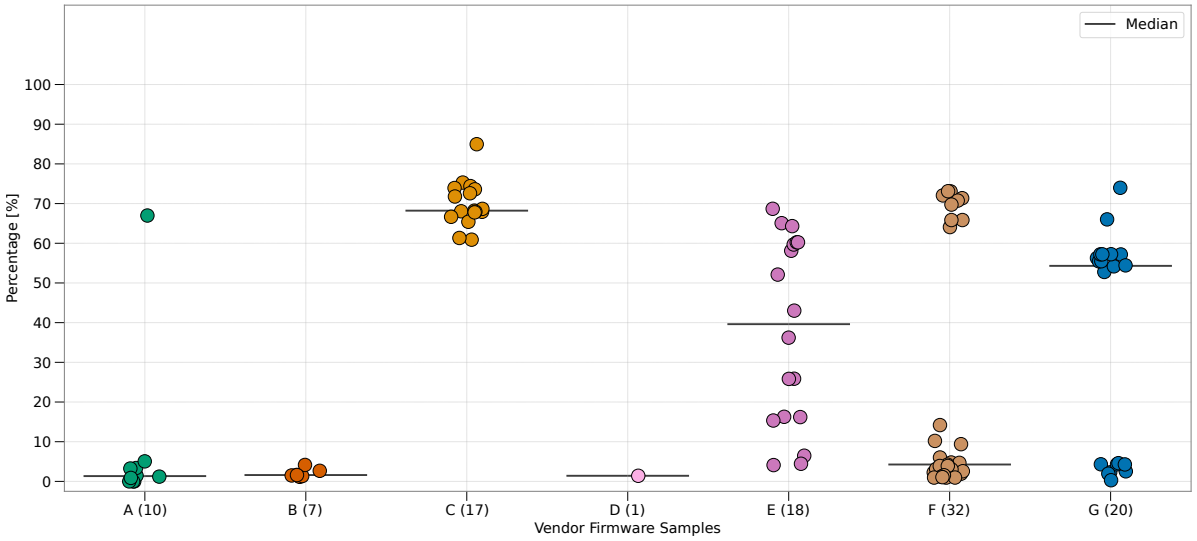

Figure 3.9: Percentage of Executables with RELRO Enabled per Firmware Image

for RELRO. One vendor apparently tries to enable RELRO on all their devices. Three other vendors are negative outliers, with only one firmware sample combined between them containing a significant number of RELRO-enabled binaries. Only one vendor has a significant number of devices with usage ratios for RELRO between 10% and 50%.

The usage of stack canaries is still uncommon, as can be seen in figure 3.10. Again, there is one positive outlier among the vendors with a visible effort for enabling stack canaries for the majority of binaries on all devices. Other vendors only have a small number of devices where significant amounts of binaries contain stack canaries. This shows that the vendors at least experiment with consequent usage of stack canaries. However, the majority of firmware samples of these vendors still contain less than 20% of binaries with stack canaries enabled.

Figure 3.11 summarizes the mean usage statistics of all binary hardening techniques for each vendor as radar charts. Only one vendor can be seen to consistently enable NX, PIE, RELRO and stack canaries on almost all devices. On the other hand, for three vendors the average device still contains almost no binary hardening at all.

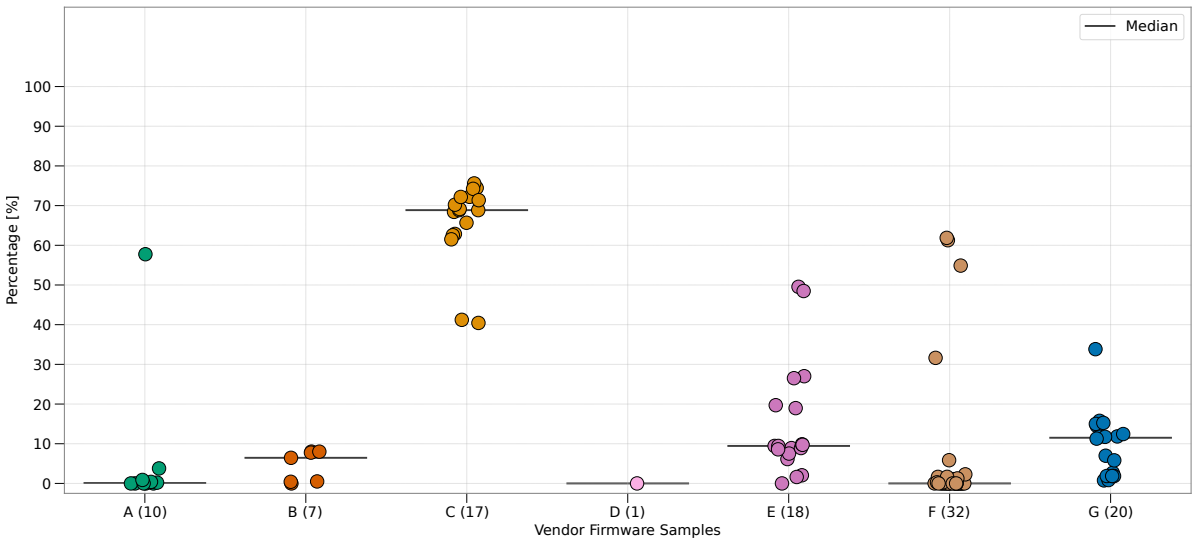

Figure 3.10: Percentage of Executables with Canary Enabled per Firmware Image

Limitations

The used detection method is based on checksec[20], which tries to find indicators for the usage of specific mitigation techniques in binaries. However, as these are just indicators, there is a risk of both false positive and false negative results.

Some sources of binaries with no enabled binary hardening even in firmware images that otherwise used such techniques were quite frequently identified as analysis errors. These files usually were artifacts of the unpacking process, which were either unpacked incorrectly or not part of the Linux file system for the device.

Another common source for binaries without detected binary hardening were Linux kernel modules. This is due to limitations of checksec, which currently cannot detect whether mitigation techniques similar to NX and RELRO are used for kernel modules.

Based on the last two reasons, our analysis underestimates the usage ratios for binary hardening compared to reality. However, based on our observations, a relative comparison of firmware images should still be valid.

Some binary hardening techniques need additional requirements on the execution environment to work correctly. For example, position-independent executables need ASLR to be activated in the Linux kernel (or KASLR for kernel modules). In addition, NX can only be enforced if the processor of the device supports it, which is not always the case. Our analysis does not include checks for those requirements, i.e. we did not check whether the processor types built into actual devices do support NX or not.

Comparison

Since we changed the script used for detection of binary hardening after the last report in 2020 a quantitative comparison of concrete percentage numbers is not possible between both reports. Instead, the following comparison focuses on qualitative differences in the results since 2020.

[20] `https://github.com/slimm609/checksec.sh`

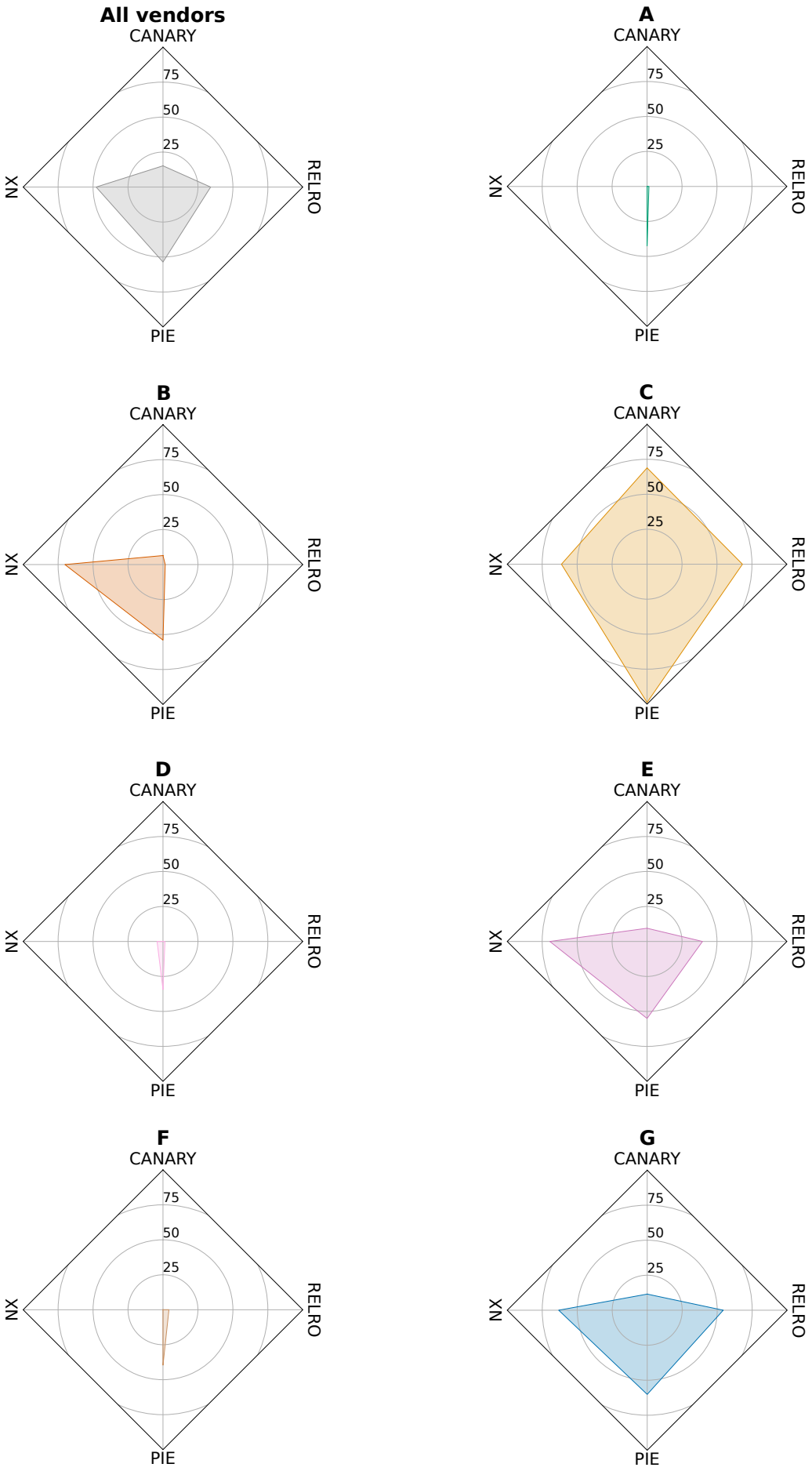

Figure 3.11: Mean of enabled binary hardening. Letters A-G represent pseudonymised vendors

Compared to the firmware samples collected for the HRSR 2020 [1] not much has changed in the usage statistics for the different binary hardening techniques.

Most notably, we did not encounter as many firmware samples with disabled NX in 2020. As already noted above, one explanation could be that this is just a reflection of vendors using MIPS-based processors without support for NX in their devices. Since the last report also included many MIPS-based devices, this would imply that in 2020 binaries for such devices were still compiled with NX enabled regardless of whether the actual processor supported it or not. However, we could not verify this theory without information about the concrete processor types built into the devices.

Apart from that, two vendors started to enable RELRO for a larger number of their firmware images since 2020, while still having many devices where it is apparently disabled by default. One can also see more vendors experimenting with the usage of stack canaries, but only on a small number of devices.

In 2020, the report also included usage numbers for the FORTIFY_SOURCE technique. As we could not verify the reliability of the used detection method for FORTIFY_SOURCE, we decided to not include numbers for that binary hardening technique in this report.

3.4 Hard-coded Login Credentials

Description

Common security best practices include strong and unique password choices. In general, this prevents attacks like password brute forcing and credential stuffing. For embedded devices Mirai has successfully shown that widely deployed default credentials can be utilized for large-scale attacks. This is addressed in the *Embedded Top 10 Best Practices*[21] of OWASP where »[do] not hard-code secrets such as passwords, usernames, tokens, private keys or similar variants« is part of the 4th best practice. Now login credentials to the device are not the same as credentials used for authenticating against a web-based device configuration. Web-based device configuration is a commonly used feature where it can be assumed that a user (a) sets a safe password themselves or (b) consciously decided to keep the default for usability reasons. Only the most technically inclined users that know their way around a command line probably use login credentials. Thus, it is unlikely that these credentials are changed. The impact of broken login credentials is further discussed in the limitations section below.

The findings presented in the following are gathered by analyzing the Linux credential files found in the firmware images and collecting all points where a password is set. As passwords are always stored in some kind of hash, a second step tries to crack the password hash. The cracking is attempted by applying two lists of common passwords with the tool *John the Ripper*[22]. The first list contains 10,000 general-purpose passwords[23], the second list contains passwords specifically used in routers[24].

[21] https://owasp.org/www-project-embedded-application-security/#div-project
[22] https://www.openwall.com/john/
[23] https://github.com/danielmiessler/SecLists/blob/master/Passwords/Common-Credentials/10k-most-common.txt
[24] https://routerpasswords.com

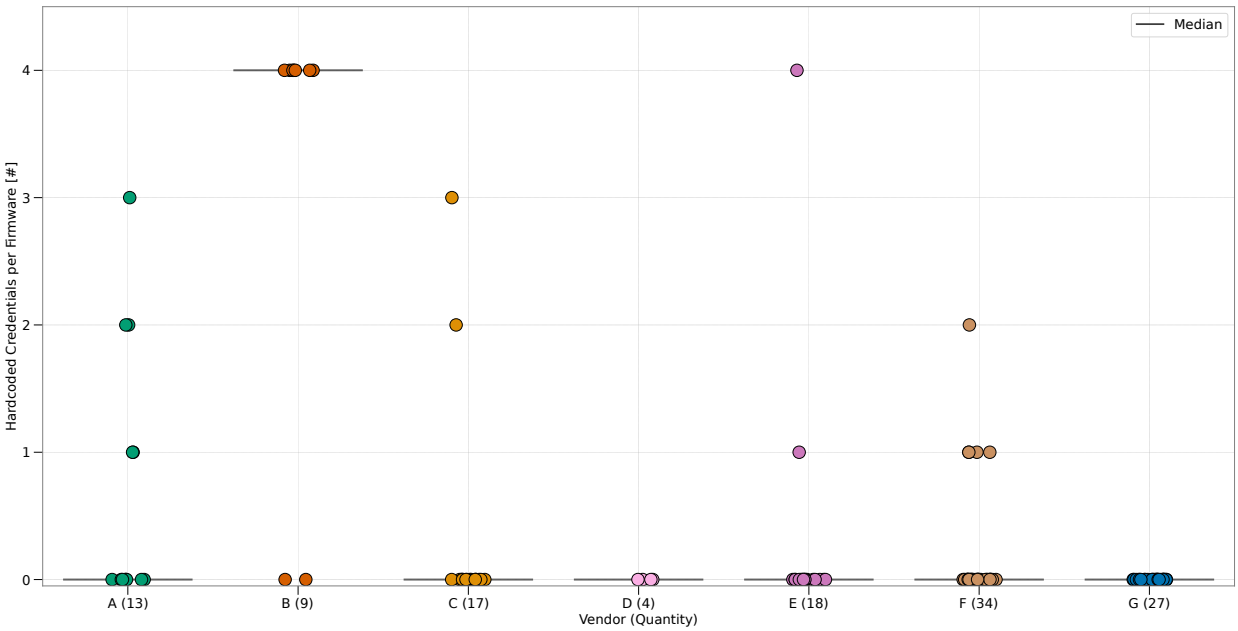

Figure 3.12: Number of Hard-coded Credentials per Firmware Image

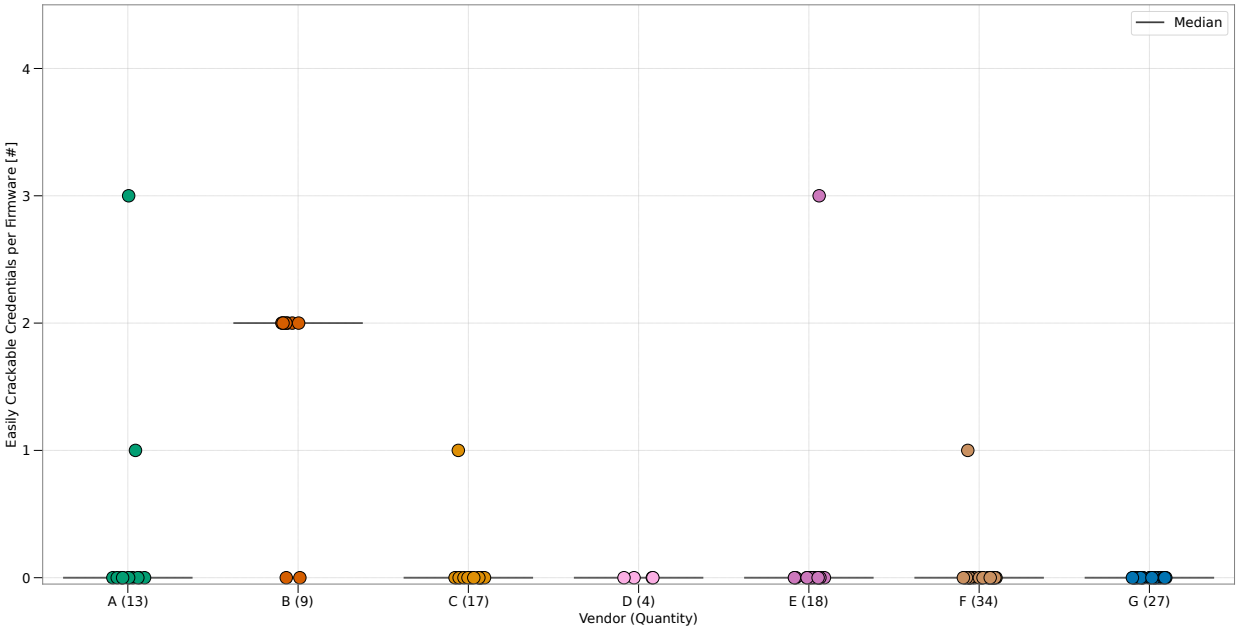

Figure 3.13: Number of Well Known Hard-coded Passwords per Firmware Image

Findings

We divide our analysis into hard-coded credentials in general and such credentials that are easy to crack with proper tooling. Figure 3.12 shows the number of all hard-coded credentials found per firmware image. The median already suggests that for all but one vendor at least half of the firmware images do not contain any hard-coded credentials. The maximum number of credentials comes out at four, which is found in eight images. 13 more firmware images contain one, two or three sets of credentials, giving us 21 devices in whose firmware we identified hard-coded credentials. While this number is encouraging, the second part of the analysis, shown in Figure 3.13, is more concerning. Of the 21 devices with hard-coded credentials, 12 devices, so more than half, contain passwords that are easy to crack. Meanwhile, the complexity of passwords shows to be quite trivial as all cracked passwords are one of four options: `admin`, `root`, `password` and one that is `amazon`.

Limitations

Assessing the impact of a credential finding is based on two additional observations: Is the password overwritten during the firmware update process? Is there a running network service that uses the credential? Additionally, if a password is not overwritten and a service actually uses the password, an important distinction is if the service can be reached from external or internal networks. In theory, these observations can be done in static analysis, but device initialization is often too complex to gather all relevant information from it. Thus, even for the guessed passwords in this analysis the two (plus one) observations have to be done to assess if the password represents a security problem or if it is merely a bad practice.

Comparison

An interesting development since the last report is that the total number of hard-coded credentials and the number of devices with at least one hard-coded credential has drastically decreased. This might be a reaction to the EU specification for security in consumer IoT [25] which lists »No universal default passwords« as first baseline requirement. At the same time, the number of easily guessed passwords has increased. This trend is concerning, as a hard-coded password in itself is not an issue as long as it is not known or cracked. Vendors should therefore prioritize removing the trivial passwords in favor of removing hard-coded credentials — though removing all is still the best option.

[25] `https://www.etsi.org/deliver/etsi_ts/103600_103699/103645/02.01.02_60/ts_103645v020102p.pdf`

List of Abbreviations

CPU	Central Processing Unit
CPE	Common Platform Enumeration
CVE	Common Vulnerabilities and Exposures
CVSS	Common Vulnerability Scoring System
DSO	Dynamic Shared Object
EOL	End-Of-Life
FACT	Firmware Analysis and Comparison Tool
GOT	Global Offset Table
IoT	Internet of Things
ISA	Instruction Set Architecture
LTS	Long Term Support
NVD	National Vulnerability Database
NX	Non-Executable Bit
OEM	Original Equipment Manufacturer
OS	Operating System
OWASP	Open Web Application Security Project
PIE	Position-Independent Executable
REL	RELocatable File
RELRO	RELocation Read-Only
ROP	return-oriented programming
XI	eXecute Inhibit
XN	eXecute Never

Bibliography

[1] P. Weidenbach, J. vom Dorp, Home Router Security Report 2020, Fraunhofer Institute for Communication, Information Processing and Ergonomics (FKIE), Technical Report, 2020

[2] R. Helmke, J. vom Dorp, Towards Reliable and Scalable Linux Kernel CVE Attribution in Automated Static Firmware Analyses, 2022

[3] L. A. Benthin Sanguino, R. Uetz, Software Vulnerability Analysis Using CPE and CVE, ArXiv, 2017

A Appendix: Vendor Discussions

For transparency reasons, this appendix documents our extensive discussions with vendors, their opposing views, but also their contributions to this report.

For each feature result we discussed in Section 3 and the corpus description in Section 2, there is a subsection containing the corresponding discussions below. We report communication with each vendor separately. Our statements are printed in cursive, vendor statements are in regular font.

A.1 Evaluation Corpus

Vendor C

Vendor C states that, as a comparative test for the German market, the corresponding analysis corpus would need to include samples for ISP devices in order to achieve market representativeness. With respect to market share, the third-party vendors included in our corpus could not be described as large, but rather as marginal or minor. In particular, vendor C exemplifies that we would not include any routers from Deutsche Glasfaser or Huawei, Speedport devices from Deutsche Telekom, O2's Homebox, or Easybox and Station models from Vodafone. Thus, vendor C claims that we would not consider over 50 % of the German market, which would significantly skew the overall picture of router security in Germany.

Our observation that ISP firmware samples are not widely accessible through public means was communicated to Vendor C. In return, Vendor C has sent us exemplary links to publicly accessible ISP firmware samples, e.g., for the Digitalisierungsbox by Deutsche Telekom and some Vodafone Easybox devices. They point out that these examples would contradict our observation in general. The mentioned data would also be analyzable by our FACT tool.

The vendor asks for justification on the omission of the previously sent samples (and ISPs in general), as well as a a more objective and plausible justification for corpus construction and included vendor relevance. Vendor C asks us to make sure that the reader understands that our results would not be representative.

The vendor does not agree with the currently implemented accommodations to further clarify non-representativeness in this report.

We explained that the Home Router Security Report is not a comparative test for the german market, and we do not claim market representativeness. Regarding the ISP firmware samples, we refer to our arguments in Section 2 and would like to emphasize our efforts to close NDAs with ISPs. We still believe that our data provides valuable insights a) on a device level, and b) in relative comparison among included vendors.

Other Vendors

Vendors A, B, D, E, F, and G did not comment on the evaluation corpus.

A.2 Age of Latest Release

Vendors A, B, C, D, E, F, and G did not comment on the findings from this metric.

A.3 Operating Systems

Vendor C

CVE Attribution Methods Accuracy, Manual Verification, and alternative ground truth.

Vendor C states that our automated static CVE attribution heuristics would generate false-positive rates of at least 84 % for their devices. Reasons for such high false-positive rates would include vendors cherry-picking official patches, incorrect kernel version ground truth by the NVD, and actually present vulnerable code that would, however, be irrelevant for the use case. As for incorrect ground truth, they proposed that we should use *linuxkernelcves.com* instead of the NVD after sighting our provided result set. Said source would be more suitable for Linux CVEs, as the project tracks CVE fix references in code commit messages. Furthermore, vendor C asks the FKIE for manual verification of at least sampled data from the result CVE set.

Due to missing information and no suitable environments for qualitative dynamic analyses on this subject, we can not verify vendor C's claim of at least 84 % false-positive rates with reasonable efforts.

We communicated to the vendor that we could not entirely build our analysis on top of the proposed linuxkernel-cves.com data set. It is, like our own approach [2], a static analysis heuristic that a) introduces another category of false-positives and false-negatives (automation might misidentify commits), and b) does not support all (EOL) kernel streams we encounter in the data set. Furthermore, in contrary to [2], the actual code has not yet been published and, thus, can not be reviewed.

We agree with vendor C that Linux kernel CVE attribution is especially hard due to unsound ground truth. Thus, we developed the previously mentioned attribution method, and decided to report linuxkernelcves.com results alongside the NVD results, but not exclusively due to the considerations stated above.

Identification of security-relevant Linux kernels.

Vendor C states that in two analyzed firmware samples, our CVE attribution would not consider the correct main system kernel, which would be of version 4.4.60 for both. They state that these analyzed kernels would be associated to modem subsystems. Vendor C asks us to re-do our analysis and manually alter Figures 3.4, 3.5, and 3.6 in such a way that the correct kernels would be considered for the vendor. Alternatively, they ask us to publicly justify why we did not alter the data regardless of their provided information on main- and subsystems for their devices in particular.

We found three versions (3.14.26, 4.4.60, 4.14.117) in one firmware, and two versions (3.18.20, 4.4.60) in the other. Given the limitations, our methods could neither consider 4.4.60, nor 3.14.26 because the required meta data was not found. We did not distinguish between Linux kernels of the main system and subcomponents, and did not filter the kernel subset for CVE attribution by other means than technical limitations. This includes that method requirements must be fulfilled as well: If there is a kernel missing in the CVE results, the kernel configuration was not detected and we can not apply attribution methods. Furthermore, we do not compare devices, but observe different kernel usage between vendors and report potentially applicable, publicly known software flaws.

We did not alter the result set or their presentation for any vendor. This is because we had to weight result accuracy against reproducibility and comprehensibility. We opted to prioritize the latter ones and document result inaccuracies in the method limitations and vendor discussions.

Vendor E

Vendor E mentions that they identify room for improvement in terms of using newer kernel versions and report not further specified quantities of false-positives yielded by our CVE attribution heuristics. While they would put in efforts to track and fix applicable CVEs in-house, kernel usage would be ultimately bound to the chip vendor SDKs in use for router development.

We acknowledge the existence of false-positives (but also missing false-negatives) in our result set, as they are inherent to static analysis methods. Yet, usage of deprecated kernels in chip vendor SDKs only moves the observed issue away from router vendors to their suppliers and does not resolve the issue of decentralized, high effort, and potentially error-prone kernel security management per vendor.

Other Vendors

Vendors A, B, D, F, and G did not comment on the findings from this metric.

A.4 Binary Hardening

Vendor C

NX-Bit and MIPS processors.

Vendor C would like to clarify that a vendor's decision for or against usage of processors with non-executable memory features would not be arbitrary and would not be a function over chip pricing. Aside of security features, the processor's technical capabilities would just be as important in order to meet actual application requirements. Exemplary SoC criteria would be available interfaces, processor speeds, power usage, and chip availability. Especially the last would have gained significant importance due to recent chip shortages. Consequently, the semiconductor market would only offer limited choices for vendors. Furthermore, vendor C states that developing products on a single architecture would be preferable, but not practicable. Thus, they would rely on a variety of architectures, as they would identify benefits and disadvantages for each.

As for MIPS devices in particular, vendor C observes that the XI-bit would just have been introduced in the recent generation of MIPS processor cores, e.g., MIPS32 1074 Kc/f. However, there would not yet be any SoC available that could be suitable for router usage. Together with our provided data, they conclude that the 8 out of 17 data points with 0 % NX usage correspond to the MIPS devices in their portfolio.

We report that none except for one of the MIPS-based firmware samples in our corpus shows binary findings with XI enabled – which might be an indicator of general absence of such functionality. However, our firmware corpus is not representative and we do not have expert knowledge on the semiconductor market, which is why we can not objectively verify the statements of vendor C.

Although we understand the vendor's comments on platform choice, we still regard it as desirable to choose SoCs with binary hardening features, as some have proven to satisfy consumer router application requirements as well. E.g., the alternative of ARM-based processors is something that vendors also have been using in router devices for years and our observations from 2020 and this report's iteration back this view (see Figures 2.1 and 3.7). On the other hand, we follow up on vendor C's stated interface-requirements and also consider that a possible factor affecting vendor choices might be that components for novel communication techniques could generally reach certain platforms faster than others or not at all.

Finally, we observe that the XI feature is part of the MIPS32 architecture specification since 2015[1]*. This leads to the question of why demand and supply have not yet affected availability of XI-enabled MIPS SoCs suitable for router*

[1] `https://s3-eu-west-1.amazonaws.com/downloads-mips/documents/MD00090-2B-MIPS32PRA-AFP-06.02.pdf`

appliances after more than seven years.

Used checksec script not suitable for kernel modules.

PIE – Vendor C contributes that the PIE binary hardening technique would not be applicable to kernel modules and shared objects because their code's position-independence would be designated by the terms DSO and REL.

As the used checksec tool detects and consolidates these features in the PIE check[2], we updated the technique's description in Section 3.3 to avoid further misunderstandings.

NX/XI – Vendor C contributes that non-executable stacks would not be designated as NX, but as XI in MIPS specifications.

We updated the NX feature description to improve clarity on this point, as our methods cover this case: Checksec uses readelf[3], which in turn is able to detect the MIPS XI and the ARM XN bits. They are reported under the umbrella term 'NX'.

RELRO and NX in kernel space – Vendor C identifies an error in our methods, as we did not prune executable kernel space objects from our RELRO and NX result set. Both NX and RELRO are user space concepts that are not applicable to the kernel space. Thus, checksec would always report for the corresponding objects that the RELRO and NX mitigations are disabled. The vendor does not agree that we respond by documenting the methodical error in the corresponding limitations of Section 3.3. Instead, C asks us to revise our analysis and alter the Figures 3.7, 3.9, and 3.11 in such a way that Linux kernel modules are excluded from the set of analyzed binaries in the RELRO and NX dimensions.

We verify both our raw data and the checksec tool and can confirm the observation of vendor C. Yet, we do not follow our decision to prioritize reproducibility and comprehensibility before result accuracy. Also, our metrics do not categorically introduce any bias that would favor certain vendors over others. We believe it is more transparent to report all tooling results as they are and that the error is sufficiently documented in the methods limitations.

Other Vendors

Vendors A, B, D, E, F, and G did not comment on the findings from this metric.

A.5 Hard-coded Credentials

Vendor C

Vendor C states that it would make a significant difference whether our findings would reside in the main system or a Linux subsystem of, e.g., an integrated WiFi or cellular component. The vendor criticizes that our heuristics need to be able to differentiate between Linux systems that would be of relevance for security.

We refer to the discussed limitations in Section 3.4 and argue that some subsystems may, too, expose functionality that is affected by some hard-coded credentials. Furthermore, the existence of such credentials, even if not directly usable without prior compromise, might be beneficial in scenarios to locally elevate privileges in post-exploitation. We would not categorically dismiss all subsystem findings due to the vendor's statement above and rather report all results as they are.

[2] `https://github.com/slimm609/checksec.sh/blob/932b9922d0f1d8bb48cf13453fd5d60157461ae0/src/functions/filecheck.sh#L37-L49`

[3] `https://man7.org/linux/man-pages/man1/readelf.1.html`

Vendor E

Vendor E confirmed that our findings correctly identified one device's firmware that contains security-relevant hard-coded credentials. The affected version was updated between the data collection and analysis stages of this report.

The vendor shared the update with us and we were able to verify the fix through static comparison with the older version.

Other Vendors

Vendors A, B, D, F, and G did not comment on the findings from this metric.