Extended Abstract: Towards Reliable and Scalable Linux Kernel CVE Attribution in Automated Static Firmware Analyses

R. Helmke(✉) and J. vom Dorp

Fraunhofer FKIE, Zanderstraße 5, 53177 Bonn, Germany
{rene.helmke,johannes.vom.dorp}@fkie.fraunhofer.de

Abstract. In vulnerability assessments, software component-based CVE attribution is a common method to identify possibly vulnerable systems at scale. However, such version-centric approaches yield high false-positive rates for binary distributed Linux kernels in firmware images. Not filtering included vulnerable components is a reason for unreliable matching, as heterogeneous hardware properties, modularity, and numerous development streams result in a plethora of vendor-customized builds. To make a step towards increased result reliability while retaining scalability of the analysis method, we enrich version-based CVE matching with kernel-specific build data from binary images using automated static firmware analysis. In a case study with 127 router firmware images, we show that in comparison to naive version matching, our approach identifies 68% of all version CVE matches as false-positives and reliably removes them from the result set. For 12% of all matches it provides additional evidence of issue applicability.

1 Introduction

Safety, security, and privacy threats arise alongside embedded system markets. Growing device numbers inflate attack surfaces, raising impact and scope of newly found software vulnerabilities in domains pivotal to society [8]. Thus, it is important to maintain the software security of these systems.

Embedded devices commonly make use of Embedded Linux[1] as host operating system for their firmware. Using open source components instead of developing custom solutions generally provides a solid security foundation; though the Linux kernel specifically has been attributed over 2,900 Common Vulnerabilities and Exposures (CVE)s as of 2022. Attesting the security of a Linux-based firmware thus includes checking which CVEs concern the specific kernel in-use.

While reproducible exploitation of a CVE would be optimal, various challenges [10,13] exist that make a comprehensive reproduction on a device unobtainable. First, not all CVEs have a known POC exploit to test against. Second,

[1] https://www.embedded.com/wp-content/uploads/2019/11/EETimes_Embedded_2019_Emb\discretionary-edded_Markets_Study.pdf.

D. Gruss et al. (Eds.): DIMVA 2023, LNCS 13959, pp. 201–210, 2023.
https://doi.org/10.1007/978-3-031-35504-2_10

exploitation requires either a running device or an emulated firmware. The former is not generally attainable for large-scale analysis. The latter is hindered my various challenges specific to firmware emulation [6,13].

Static analysis serves heuristics to find *imperfect* proof for CVE attribution. Yet, many approaches do not scale well as they require considerable manual work and deep knowledge of each CVE [10]. Parts are automatable but needed data may be unavailable or incorrect in repositories [1,11]. Also, automation becomes increasingly challenging considering proprietary formats, obfuscation, compiler optimizations, and symbol stripping [3,10].

In lack of better methods, firmware analysis tools [4,9] and large-scale studies [2,12,14] commonly attribute vulnerabilities by matching versions against CVE databases. One such study [12] used version matching on the Linux kernel as part of an empirical study on home router security. Due to custom build configurations, implying that each kernel includes only a subset of all possible vulnerabilities, this method is exceedingly unreliable. To improve the reliability of version-based Linux CVE attribution in large-scale scenarios, we enrich such naive matching with kernel-specific configuration data, collected through automated static firmware analysis. Hereby, we reduce the set of false-positive matches requiring further manual verification. In the following, we provide:

1. A description of our methodology for Linux CVE attribution, based solely on binary kernel representations.
2. A case study in which we compare our approach with naive version-based CVE matching using the 2020 Home Router Security Report [12] dataset.
3. An open source proof of concept implementation of the methodology[2].

2 Background & Related Work

Automated vulnerability detection is approached using various methods such as code similarity and patch analysis [5], fuzzing [7], and various emulation-based methods [13]. Large-scale detection of known vulnerabilities requires sound ground truth. Thus, we focus our discussion on sound vulnerability information and previous research on discovering known vulnerabilities on binary code.

Sound Data as Foundation for known Vulnerability Detection. Correct and detailed information on known vulnerabilities [10] is essential for effective automated detection methods. The community-driven CVE catalog[3] offers a de facto standard for vulnerability identification but comes with limitations due to errors in Common Platform Enumeration (CPE) assignments [1], missing or hard to obtain data [3] and inconsistent references to patches [11]. Additionally, for CVEs affecting closed source projects, issuers will not share technical details on fixes in public. In this work, we leverage upon the observation that the summary of most Linux kernel CVEs includes a file reference to mark which kernel part is affected.

[2] https://github.com/fkie-cad/cve-attribution-s2.
[3] https://www.cve.org/.

In consequence, most research comes with small custom datasets of selected CVEs that their proposed techniques can ingest for evaluation [10]. The necessary investigation, data aggregation, and technical bug knowledge, limits the applicability of previously -scale scenarios.

Static Vulnerability Detection in Large-Scale Firmware Analyses. In 2014, Costin et al. [2] executed a quantitative study on embedded device security by analyzing 32.000 firmware images. They attributed CVEs based on software version number and then reported unsolved challenges in result verification, as not only CVE data is incomplete, but vendors may also custom-patch files.

Cross-architecture code similarity methods (e.g. FirmUp [3]) have drastically improved and may be used as imprecise measure for verification in this case. However, acquiring and processing patches for thousands of CVEs to bootstrap code similarity methods deems infeasible based on the imprecise CVE repositories.

Zhao et al. [14] develop FirmSec, a large-scale static analysis pipeline for IoT devices. The approach extracts syntactical and control-flow graph features and, thus, provide an alternative for signature-based version detection. However, the applicability issue introduced by vendor-specific build configurations, as in the Linux kernel, is not considered.

The authors of [12] assess and compare the state of firmware security of 127 home routers in similar aspects as [2], using the automated Firmware Analysis and Comparison Tool (FACT) [4]. Identified Linux kernel versions are matched against the National Vulnerability Database (NVD)[4] to calculate how many critical CVEs affect the kernel of each firmware. In this study, the stated issues with CVE database information and kernel modularity lead to high false-positive rates, incurring high manual verification efforts.

There is few work that specifically studies high false-positive CVE attribution rates caused by the Linux kernel's modularity. With version-based CVE attribution being a common method, we identify false-positive reduction in static kernel CVE attribution as a research gap.

3 Methodology

This section describes our proposed methodology to enrich the version-based Linux kernel CVE attribution process with build-specific annotations. We show an automated static analysis pipeline that finds and extracts kernel configurations, dry builds the found kernel version, and filters CVEs based on affected version and build log-included files.

Figure 1 provides an overview of our methodology. We establish a two-stage process: In the first and left-hand stage, we unpack, analyze, and annotate each file of an ingress firmware image. Gathered information includes Linux kernel version, Instruction Set Architecture (ISA), and kernel build configuration. In the second and right-hand stage, we leverage upon said data to perform the actual CVE attribution and filtering step. Yellow boxes in Fig. 1 mark components this paper contributes.

[4] https://nvd.nist.gov/.

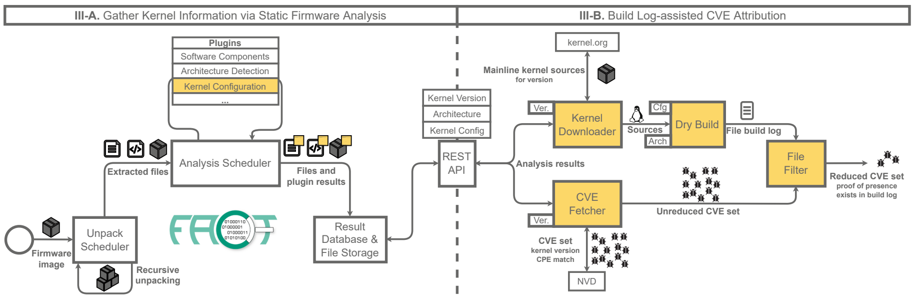

Fig. 1. Two-staged static analysis pipeline to (I) gather kernel information and (II) attribute kernel CVEs accurately.

In the two following Subsects. 3.1 and 3.2, we provide detailed technical insights on each stage and step.

3.1 Gather Kernel Information via Static Firmware Analysis

For stage one, we apply and enhance the open source firmware analysis tool FACT [4]. FACT provides automated firmware analysis capabilities including recursive extraction, kernel version, and ISA detection. In the following, we describe all steps that are of importance for the proposed attribution methodology.

Starting with an arbitrary Linux firmware image, we first use FACT internals to recursively extract all components necessary for analysis, including the kernel. Next, we identify the ISA and kernel version. The **Analysis Scheduler** achieves this by running a selected set of analyses on each extracted object. The software version detection uses YARA rules and the ISA detection leverages ELF header information, detected kernel configurations, and device trees.

We contribute the **Kernel Configuration** plugin, which detects and extracts Linux kernel build configurations in firmware images. It is pivotal to the succeeding dry build pipeline step, as it determines components included in kernel builds. In firmware, kernel configurations may be present as plain text or in binary form. Detection of plain text configurations is straight forward due to the distinctive key-value structure and well-known directive keywords. These can be used for pattern matching. If the `CONFIG_IKCONFIG` directive is enabled (Y) during build, the kernel configuration gets embedded into the binary kernel image. This embedding might be an inline string or a binary compressed representation using common algorithms like LZMA or DEFLATE. If it is set to `M`, the configuration is outsourced to a kernel module. Thus, if the file is either a kernel image or module, our plugin searches for an embedded magic word that precedes the kernel configuration data. The plugin tests for all variations and extracts, and if necessary, decompresses the configuration.

3.2 Build Log-Assisted CVE Attribution

The build log-assisted CVE attribution is the second stage of our proposed analysis pipeline in Fig. 1. Here, we first use FACT's **REST API** to consolidate the kernel version, kernel build configuration, and detected target ISA.

Then, our contributed **Kernel Downloader** fetches mainline version sources from `kernel.org`. We emphasize that our assumption of unaltered mainline kernels in firmware images is likely false because vendors may custom-patch their kernels. However, we observe that modified kernel code is not accessible in scale, regardless of the Linux kernel's GNU General Public License (GPL) that dictates vendors to publish modified open source code. For example, some vendors complicate distribution by implementing manual request procedures for each device, firmware, and version[5].

Dry Build is the next step in Fig. 1. We set the target ISA and install the extracted kernel build configuration in the downloaded kernel source project. Then, we execute a compilation dry run, which does not compile the kernel but prints each compilation recipe instead. This approach has the advantages of low computational overhead and no requirement for a cross-compilation toolchain. With this step, we gather a list of source files from the build log, which our pipeline *witnesses* to be included in the kernel build.

The **CVE Fetcher** executes simultaneously. We query the NVD dataset for all Linux kernel CVEs and filter out all records that do not refer CPEs stating the extracted Linux kernel version to be vulnerable. The result of this stage is identical to naive version-based attribution.

The **File Filter** step combines the outputs of CVE Fetcher and Dry Build: Based on the observation that Linux kernel CVEs summaries usually state the affected source files, we improve on the version-based attribution by removing every CVE that does not reference an affected file we witnessed in the build log.

4 Case Study

We perform a case study to evaluate the reliability of our enriched version-based Linux kernel CVE attribution in large-scale static analyses. For this purpose, we let our pipeline analyze the Home Router Security Report 2020 [12] corpus, which vendors reported to yield high false-positive rates using version-based CVE matching. We raise two research questions:

R1 Our methodology requires access to specific information in firmware samples and CVE repositories. How many samples and CVEs fulfill these modalities? *How applicable is our approach in a real-world scenario?*

R2 With version-based CVE matching as baseline, *what impact has the methodology on result reliability?*

In the following subsections, we first provide detailed information on our experiment and used dataset (4.1). Then, we present the results and analyze them within the context of both stated research questions (4.2 and 4.3, respectively).

4.1 Experiment & Firmware Corpus

Experiment Execution. We deploy our analysis pipeline on a x86_64 desktop system, running Ubuntu 20.04.4 LTS. FACT v4.0 (commit `38df4883`) is used

[5] https://www.zyxel.com/form/gpl_oss_software_notice.shtml.

in the first pipeline stage to detect CPU architecture, identify the kernel, and extract the kernel configuration. The second pipeline stage executes on the same machine based on a snapshot of the NVD – taken on 2022–08-30. The snapshot has records for 2,910 Linux kernel CVEs attributable through CPE. For each component in our system, we collect details on ingress and egress data, including plugin results, version-based CVE matches, and filtering decisions.

Firmware Corpus. The analyzed Home Router Security Report [12] corpus is publicly documented[6], and consists of firmware from 127 recent home routers. Devices of seven vendors are included: ASUS, AVM, D-Link, Linksys, Netgear, TP-Link, and Zyxel. Samples were scraped on 2020–03-27.

Across all 127 samples, 121 binary distributed Linux kernels from v2.4.20 to v4.4.60 are included. The most common major version is 3.x with 49 kernels, while 44 kernels have version 2.6. 11 firmware images are not analyzable due to failed operating system detection or unpacking errors. Note that firmware can contain multiple kernels, e.g., embedded devices may consist of subcomponents running their own systems.

All identified CPU architecture belong to the MIPS and ARM ISAs, with a majority having a word length of 32-bit. The ISA is unknown for 24 samples. Further corpus insights can be found in [12].

While we acknowledge the missing size and device class diversity of the dataset compared to studies like [2,6], we argue that the dataset is of sufficient size to demonstrate applicability, as it covers Linux kernels from three major releases, widely spread ISAs, and devices of multiple vendors. We also choose it to investigate the reliability of matches reported in [12] using naive version-based Linux kernel CVE attribution.

4.2 R1 Analysis – Applicability in Real-World Scenarios

We identify two methodological requirements that must be fulfilled for each firmware and Linux CVE for our approach to succeed:

S1 FACT firmware extraction and all plugin analyses of stage one must succeed to consolidate the kernel version, ISA, and kernel configuration.
S2 CVE descriptions must reference affected files to filter vulnerable components not included in the kernel build.

Using the firmware corpus, we evaluate egress and ingress data of each step in the proposed pipeline with regards to these requirements. Table 1 shows the results. Highlighted rows designate effective requirement fulfillment rates over all analyzed firmware images and Linux kernel CVEs.

For requirement S1, FACT successfully extracted 116 out of 127 firmware images. It then identified both kernel binary and kernel version for all 116 extracted images. The ISA was successfully identified in 103 images. However, our Kernel Configuration plugin finds build information in only 44 samples. Thus 34.64% of all analyzed firmware samples fulfill requirement S1. This rate is explainable

[6] https://github.com/fkie-cad/embedded-evaluation-corpus/blob/master/2020/FKIE-HRS-2020.md.

Table 1. Method Applicability Analysis for the Firmware Corpus

S1 Requirement (FACT Analysis Success)		
	FW Matches [#]	Fulfilled $\left[\frac{\text{FW Matches}}{\text{FWs Total}}\right]$
Extraction	116	0.9133
Kernel Version	116	0.9133
Architecture Detection	103	0.8110
Kernel Configuration	44	0.3464
S2 Requirement (File Reference in Linux Kernel CVE)		
	CVE Matches	Fulfilled $\left[\frac{\text{CVE Matches}}{\text{CVEs Total}}\right]$
Full Path Reference	1743	0.5990
File Only Reference	129	0.0443
No Reference	1038	0.3567

considering that a) `IKCONFIG` must be explicitly enabled to embed kernel configurations into binary representations and b) it is common practice for vendors to strip unnecessary information for memory saving and obfuscation purposes.

For requirement S2 (affected files must be referenced in Linux kernel CVE descriptions), data analysis over all Linux CVEs inside the NVD yields three different categories: Files are either referenced as **Full Path** relative to the kernel's source tree, or the reference is **File Only** (location in the source tree is unknown), or **No Reference** exists at all. Table 1 distributes all 2,910 Linux kernel CVEs across these classes, showing that the proposed approach is applicable to 1,872 (64.33%) kernel CVEs. For CVEs with no included file reference, the approach falls back to version-based CVE matching and, thus, cannot add value to result reliability.

4.3 R2 Analysis – Impact on CVE Attribution Result Reliability

We approach research question R2 by analyzing the attribution results of all 44 firmware images our methodology is applicable to (cf., Sect. 4.2). Subject samples

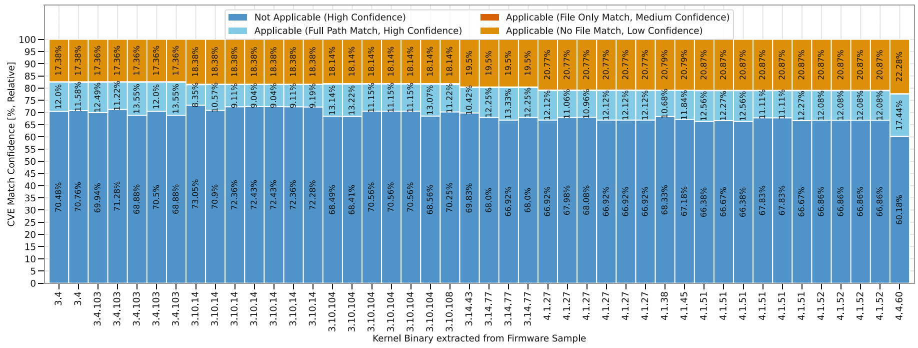

Fig. 2. Filter verdict distribution of our pipeline relative to the baseline CVE attribution results for each of the 44 analyzed kernels.

include kernels ranging from v3.4.0 to v4.4.60. At the time of this evaluation, none of these are still actively maintained by the mainline kernel team.

The baseline method attributes a median of 1,196 CVEs per firmware image, which is roughly 40% of *all* Linux kernel CVEs present in the NVD. A possible explanation lies within unsound and/or unmaintained CVE records in the NVD [1,11]. Based on the results we present in the following paragraphs, there is reason to assume that the baseline yields exceedingly high false-positive rates.

Version-based CVE attribution is an intermediate result of our methodology (cf., Sect. 3). To estimate the impact our pipeline has on result reliability, we consolidate all decisions of the build log-assisted filtering to classify them into four categories of verdict confidence:

- **Applicable (High)** – CVE references affected files and full file path is witnessed in build log.
- **Not Applicable (High)** – CVE references affected files but none of them is present in the build log.
- **Applicable (Medium)** – CVE references affected files but does not state full file paths. A file was matched and seen in the build log, but ambiguity exists due to duplicate names in the source tree.
- **Applicable (Low)** – No file references, we cannot decide on applicability and fall back to version-based matching.

The idea is to map persuasiveness of additional evidence the pipeline gathers within a trial: File matches are witnesses for CVE applicability, but not every match is equally credible.

Figure 2 shows the filter verdict distribution of our pipeline relative to the baseline CVE attribution results for each analyzed kernel. Versions are ordered from oldest (left) to newest (right). Note that a single kernel was found in each one of the 44 analyzed samples. Thus, each entry on the horizontal axis represents a unique firmware. All distribution values are medians across all samples.

The proposed Linux kernel CVE attribution methodology made a medium to high confidence decision for 80.6% of all version-based matches. The portion of high confidence applicable CVE matches is 12.04%. Relative path matches yielding medium confidence applicability are negligible with 0.19%. As indicated by the bottom bars belonging to the class of Not Applicable (High), our pipeline attributes 68.37% of all version-based CVE matches as *false-positives* and filters them out of the result set. Out of the median 1,196 matches per firmware, we reduce the set of potentially applicable CVEs to roughly 378. Thus, we significantly reduce the result set of potentially applicable CVEs requiring manual verification by analysts and vendors. The portion of low confidence applicability due to missing file references is 19.4%. Unfortunately, our methodology does not generate added value for this subset.

5 Limitations

We identify methodological shortcomings in three dimensions: applicability, sound ground truth, and functionality.

In terms of applicability, our Linux kernel CVE attribution pipeline is bound to FACT's static analysis success. If the kernel version, ISA, and build configuration remain unknown, our method cannot identify possibly included components for reliable CVE filtering. Yet, the case study in Sect. 4 shows that there is still a considerable amount of firmware fulfilling all requirements.

As for sound ground truth, reliable and true-positive CVE attribution is limited by the quality of its underlying dataset. Unsound Linux kernel CVE records that reference unaffected versions or source files can introduce false matches in our proposed method. Our assumption of vendors using mainline kernels is another limiting factor that affects reliability, but a methodical necessity due to missing insider information. Vendors may cherry-pick patches or introduce custom fixes, which are not detectable by our approach. While some of the modifications might be obtainable through GPL portals, we identify the issue of scalable accessibility. Another limitation comes from the file-based filtering. Kernel builds can in- or exclude only parts of a file based on configuration options. This can lead to exclusion of vulnerable code, while the *affected* file still appears in the build log.

Regarding functional limits, we stress the inherent limitations of static analysis. It may use heuristics to find indicators of bug presence but can hardly serve definitive proof – which usually requires triggering the bug during runtime.

Finally, the conducted case study is limited in its validity, as the used corpus lacks device class heterogeneity.

6 Conclusion and Future Work

In this paper, we present a method to improve result reliability of version-based CVE matching for the special case of binary Linux kernels in large-scale static firmware analyses. This is achieved by enriching naive version-based CVE matching with a static attribution pipeline that detects kernel configurations and ISAs in firmware images. We reconstruct the kernel build process and infer included source files. Combined with kernel CVE information, where affected files are explicitly stated, this can be used to remove most false-positive CVE annotations.

The case study shows that, with the limitations discussed in Sect. 5 in mind, our method is scalable and moderately applicable: For 34.64% of firmware images, the technical requirements are fulfilled and about 65% of all Linux kernel CVEs reference affected files in their description. Compared to naive version-based matching, the method generates a high-confidence filter verdict for 80.6% of all attributed CVEs. Specifically, 68.37% of attributed CVEs are discarded as false-positives. We contributed stage one of our pipeline to the publicly available FACT [4] and published the scripts for stage two on GitHub (https://github.com/fkie-cad/cve-attribution-s2).

In future work, we want to address the reliance on inline kernel configuration that leads to the moderate applicability by researching alternative options to infer configuration from binary kernels. Also, the fine-granular commit-based version tracking as offered by `linuxkernelcves.com` is a promising alternative data source for initial version-based attribution. Furthermore, partial file compilation and custom backports should be addressed. Finally, we plan a large-scale evaluation addressing missing device class heterogeneity, like [2,6].

References

1. Benthin Sanguino, L.A., Uetz, R.: Software Vulnerability Analysis Using CPE and CVE. ArXiv (2017). https://doi.org/10.48550/arXiv.1705.05347
2. Costin, A., Zaddach, J., Francillon, A., Balzarotti, D.: A Large-Scale Analysis of the Security of Embedded Firmwares. In: 23rd USENIX Conference on Security Symposium (SEC '14). USENIX Association, San Diego, USA (2014)
3. David, Y., Partush, N., Yahav, E.: FirmUp: precise Static Detection of Common Vulnerabilities in Firmware. In: 23rd International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '18). ACM, Williamsburg, USA (2018)
4. Fraunhofer FKIE: FACT - Firmware Analysis and Comparison Tool. https://github.com/fkie-cad/FACT_core
5. Haq, I.U., Caballero, J.: A Survey of Binary Code Similarity. ACM Comput. Surv. **54**(3), 01–38 (2021)
6. Kim, M., Kim, D., Kim, E., Kim, S., Jang, Y., Kim, Y.: FirmAE: towards Large-Scale Emulation of IoT Firmware for Dynamic Analysis. In: 2020 Annual Computer Security Applications Conference (ACSAC '20). ACM, Austin, USA (2020)
7. Manès, V.J., et al.: The Art, Science, and Engineering of Fuzzing: a Survey. IEEE Trans. Softw. Eng. 47(11), 2312–2331 (2021)
8. Neshenko, N., Bou-Harb, E., Crichigno, J., Kaddoum, G., Ghani, N.: Demystifying IoT Security: an exhaustive survey on iot vulnerabilities and a first empirical look on internet-scale IoT exploitations. IEEE Commun. Surv. Tutorials **21**(3) (2019)
9. ONEKEY GmbH: ONEKEY Automated Firmware Analysis Platform. Accessed 5 Sep 2022. https://onekey.com/
10. Qasem, A., Shirani, P., Debbabi, M., Wang, L., Lebel, B., Agba, B.L.: Automatic Vulnerability Detection in Embedded Devices and Firmware: survey and layered taxonomies. ACM Comput. Surv. **54**(2), 1–42 (2021)
11. Tan, X., et al.: Locating the Security Patches for Disclosed OSS Vulnerabilities with Vulnerability-Commit Correlation Ranking. In: 2021 ACM SIGSAC Conference on Computer and Communications Security (CCS '21). ACM, Virtual, Republic of Korea (2021)
12. Weidenbach, P., Vom Dorp, J.: Home Router Security Report 2020. Fraunhofer Institute for Communication, Information Processing and Ergonomics (FKIE), Tech. Rep. (2020). https://www.fkie.fraunhofer.de/en/press-releases/Home-Router.html
13. Wright, C., Moeglein, W.A., Bagchi, S., Kulkarni, M., Clements, A.A.: Challenges in Firmware Re-Hosting, Emulation, and Analysis. ACM Comput. Surv. **54**(1), 1–36 (2021)
14. Zhao, B., et al.: A Large-Scale Empirical Analysis of the Vulnerabilities Introduced by Third-Party Components in IoT Firmware. In: 31st ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA '22). ACM, South Korea (2022)