# Mens Sana In Corpore Sano: Sound Firmware Corpora for Vulnerability Research

René Helmke*, Elmar Padilla*, and Nils Aschenbruck°
*Fraunhofer FKIE, Cyber Analysis & Defense, Germany, {firstname.lastname}@fkie.fraunhofer.de
°Osnabrück University, Distributed Systems Group, Germany, aschenbruck@uos.de

***Abstract*—Firmware corpora for vulnerability research should be *scientifically sound*. Yet, several practical challenges complicate the creation of sound corpora: Sample acquisition, e.g., is hard and one must overcome the barrier of proprietary or encrypted data. As image contents are unknown prior to analysis, it is hard to select *high-quality* samples that can satisfy scientific demands. Ideally, we help each other out by sharing data. But here, sharing is problematic due to copyright laws. Instead, papers must carefully document each step of corpus creation: If a step is unclear, replicability is jeopardized. This has cascading effects on result verifiability, representativeness, and, thus, soundness.**

**Despite all challenges, how can we maintain the soundness of firmware corpora? This paper thoroughly analyzes the problem space and investigates its impact on research: We distill practical binary analysis challenges that significantly influence corpus creation. We use these insights to derive guidelines that help researchers to nurture corpus replicability and representativeness. We apply them to 44 top tier papers and systematically analyze scientific corpus creation practices. Our comprehensive analysis confirms that there is currently no common ground in related work. It shows the added value of our guidelines, as they discover methodical issues in corpus creation and unveil miniscule step stones in documentation. These blur visions on representativeness, hinder replicability, and, thus, negatively impact the soundness of otherwise excellent work.**

**Finally, we show the feasibility of our guidelines and build a new corpus for large-scale analyses on Linux firmware: LFwC. We share rich meta data for good (and proven) replicability. We verify unpacking, deduplicate, identify contents, provide ground truth, and demonstrate LFwC's utility for research.**

## I. Introduction

Embedded systems are part of everyone's life. Their proliferation in households, industries, and critical domains makes them highly lucrative targets for cyber attacks with devastating impact, e.g., [1]–[4]. Thus, finding vulnerabilities in the firmware running on these systems is an important task.

Firmware vulnerability research becomes a matryoshka doll of nested analysis problems when no source code is available: Acquisition is hard, data may be encrypted, and architectures are manifold [5]. Heterogeneity and resource constraints defy established analysis methods for general-purpose systems [6].

Network and Distributed System Security (NDSS) Symposium 2025
24-28 February 2025, San Diego, CA, USA
ISBN 979-8-9894372-8-3
https://dx.doi.org/10.14722/ndss.2025.230669
www.ndss-symposium.org

Automated firmware vulnerability research has, thus, become a prevalent research topic [5], [6]: Common static methods are cross-platform code similarity or taint analysis [7], [8]. Dynamic approaches explore scalable emulation to create test beds for techniques like fuzzing [6].

Regardless of the method conducted, there is a need for high-quality firmware corpora for sound evaluations. This is intuitive, as related fields show that careful curation, rich meta data, and meticulous documentation foster scientific rigor, enable replicability, and emulate real conditions [9], [10].

We have conducted a literature review on firmware corpora and found little consensus on their creation: Some researchers, e.g., scrape the Internet [11], [12] while others select few images [13]. Some describe unpacking [8]; others do not [13]. Some collect product data [12]. Others do not [11]. Copyright and intellectual property laws limit sample sharing. Thus, some share no data [14], but others provide source links [11].

All of the above affects soundness. Unpacking is an example [5]: If we share too few details, replication may fail. This may push us towards small corpora, which can add bias. We may bulk collect to improve unpacking odds; but without filters, we affect representativeness: If data is riddled with, e.g., old samples, it may no longer represent today's vulnerabilities.

To sidestep these problems, we could craft synthetic tests, but they can not fully model real conditions [15]. Thus, this paper focuses on real firmware. Of course, analysis methods affect our corpora; not all work targets the same systems. But ideally, we may agree on unified and sound data requirements. This paper is a comprehensive analysis of the problem space of sound firmware corpora. We provide guidelines to improve their soundness, and contribute a new corpus that follows these guidelines. More specifically, our contributions are as follows:

- We distill common firmware analysis problems from related work [5], [6] to pinpoint corpus creation challenges: What makes the creation of scientifically sound corpora so hard? We set ground for a practical perspective on (and better understanding of) the problem space.
- With the challenges in mind, we propose a framework of data requirements to increase the soundness of firmware corpora: Three superordinate goals are nurtured by six requirements and 16 measures. It can support researchers by raising attention to the small step stones that lie in their way towards scientifically sound corpora. As a Latin

idiom states: *Mens sana in corpore sano* – These are guidelines to support *a healthy mind in a healthy body*.

- We show that there is no common ground on corpus documentation, even in otherwise excellent work: We review 44 top tier papers to collect data on our framework. We discover that missing meta data, scarce documentation, and inflated corpus sizes blur visions on representativeness and hinder replicability. This demonstrates the need for a set of best corpus practices that the community can agree on, highlights the added value of our framework, and defines the research gap that this analysis paper fills.
- We describe a new Linux Firmware Corpus (LFwC) to show our framework's feasibility. It contains 10,913 high-quality, meticulously documented, and fully unpackable images from ten manufacturers. It covers 2,365 devices and 22 device classes. We share rich meta data with the community and perform deduplication and content identification. We use an established open source tool for replicable, effective, and verified unpacking. The scripts are available at https://github.com/fkie-cad/linux-firmware-corpus. Access to the meta data can be requested on Zenodo [16].

## II. An Analysis of Corpus Creation Challenges

Establishing a practice-oriented perspective on the problem space of scientific firmware corpora allows us to propose sane data requirements and conduct fair analyses of existing work. We consult surveys by Wright et al. [6] and Qasem et al. [5], who identify binary analysis challenges from two perspectives: The former targets vulnerability research and the latter emulation. Yet, explicit questions on *which* and *how* challenges affect corpora remain unanswered.

We approach these questions by collecting and analyzing challenges from the surveys [5], [6]: We deduplicate similar items, reorganize them, and distill their impact on corpus creation by including examples from related work. We drop challenges with no clear impact. Superordinate goals lack clarity, e.g., *improving accuracy* [5]. We mark *General* and *Method-Specific* challenges that do not apply to all papers.

We use the taxonomy by Muench et al. [14] for firmware classification. They define four types on the axes of operating system (OS) abstraction and specialization: general purpose (Type-0), retrofitted general purpose (I), special purpose (II), and bare-metal (III). Respective examples are Windows, a retrofitted Linux, VxWorks, and bare-metal binaries based on Contiki-NG. Appendix A describes the types in detail.

Fig. 1 shows the distilled challenges with impact on corpus creation. The first four are general, the latter four are method-specific. The connections show all source challenges with their newly associated descriptive class.

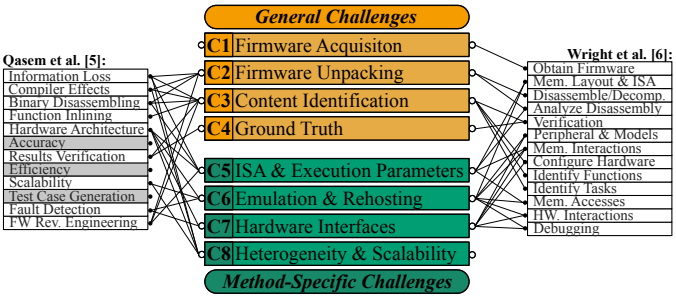


Fig. 1. Firmware analysis challenges that can have an impact on scientific firmware corpus construction. We distill common challenges from the surveys of Wright et al. [6] and Qasem et al. [5] and group them into eight descriptive classes. Items from the source surveys that show no clear impact on corpus creation are in grey: *Accuracy*, *Test Case Generation*, and *Efficiency*. We mark *General* and *Method-Specific* challenges.

**C1** *Firmware Acquisition.* Empirical data shows high Type-I accessibility [7], [8], [11], [12], [17]–[20]: Research scrapes the Internet to download samples from manufacturer or third party sites. This is non-trivial, as scrapers must navigate volatile pages and links. If access is restricted, which is common when entering industrial settings with Type-II and -III systems, one resorts to Hardware-In-the-Loop (HIL) testing or invasive extraction. The latter means over-the-air captures, reading debug ports or extracting chips. The devices a paper targets dictate feasible acquisition and corpus sizes. Yet, this is not bound to firmware types. E.g., some Type-I firmware remains unmaintained and unpublished: Scalable acquisition is infeasible. Vice versa, some manufacturers make Type-III easily accessible.

**C2** *Firmware Unpacking.* Researchers must unpack firmware before analyzing its contents. But many archive types exist and manufacturers often use proprietary formats or encryption [11]. Thus, there is significant effort to identify, reverse engineer, and decrypt samples. The community uses tools that unpack known formats, e.g., [21]–[24]. Yet, the challenge persists as formats and encryption changes.

**C3** *Content Identification.* Prior to analysis, firmware contents are often unknown. This yields two problems: First, content serves as unpacking validation. If there is no ground truth on contents, there is uncertainty in unpacking success. Second, we must ensure that contents match analysis requirements. One can not evaluate, e.g., kernel analyses on images without kernel. Information loss is a related issue, e.g., binary information is needed but stripped from the image [5].

**C4** *Ground Truth.* Firmware with ground truth is needed for result validation. Samples might have known vulnerabilities of weaknesses targeted by a method, e.g., taint-style bugs for data flow analyses. Another example would be firmware with unstripped executables to test correct function identification. This is a problem, as such data is not always available.

**C5** *ISA & Execution Parameters.* Methods that analyze or execute code must often be aware of the Instruction Set Architecture (ISA) and execution parameters like base address, entrypoint, and memory layout [5], [6]. For Type-I and -II, this is a smaller issue as systems use known formats providing these parameters, e.g., ELF. Yet, the documentation of proprietary Type-III systems is often restricted and they

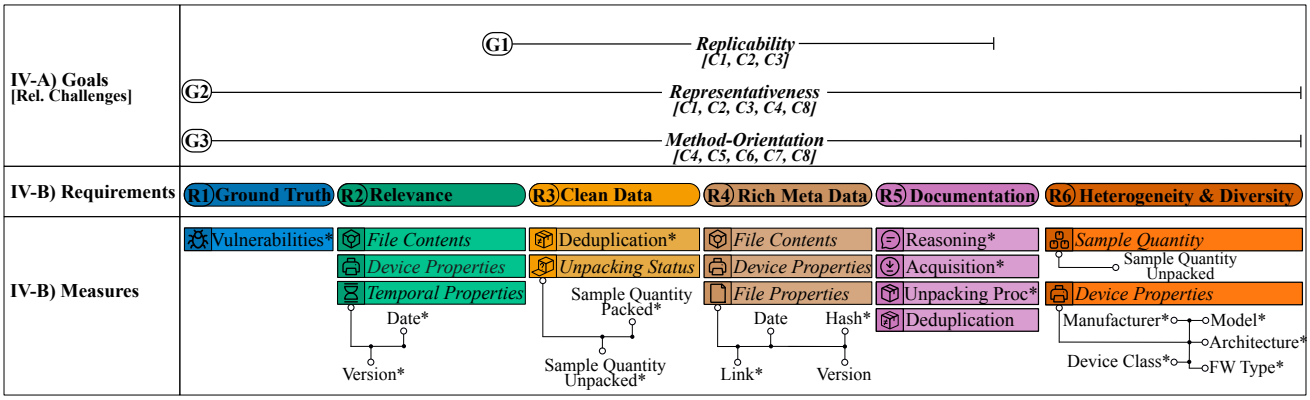


Fig. 2. A framework of guidelines for the creation of scientifically sound firmware corpora. It consists of three layers: On top are the superordinate scientific goals **Replicability**, **Representativeness**, and **Method-Orientation**. They are associated with the identified problems in corpus creation from Section II. To achieve these goals, a corpus must fulfill six requirements, which is the second layer: **Ground Truth**, **Relevance**, **Clean Data**, **Rich Meta Data**, **Documentation**, and **Heterogeneity & Diversity**. In layer three, we add a list of 16 unique and practical measures designated by an asterisk (*). They help to assess the fulfillment of the previously mentioned requirements. Note that measures can serve multiple requirements. Abstract measures are written in *cursive*. We do not claim list completeness, as varying paper scenarios and method constraints may imply additional or substitute measures.

operate on unknown execution parameters. Researchers can only use samples where this information is available.

**C6** *Emulation.* Dynamic methods like fuzzing are faced with emulation challenges [6]: Generally, one can only test devices with components supported by the emulators and detail information on the execution environment. In all other cases, one must laboriously establish compatibility, or develop strategies to automate the process [12], [25]–[27]. Example implementation tasks are ISA translation layers and peripheral behavior, which must consider timing, interrupts, or direct memory access. This is especially relevant considering the heterogeneity, specialization, and restricted documentation of Type-II to -III systems.

**C7** *Hardware Interfaces.* HIL methods can only work with sample devices where interfacing during runtime is possible. This might be an Ethernet port or debug interface such as I2C, JTAG, SPI, or UART on the PCB.

**C8** *Heterogeneity & Scalability.* HIL and its need for real device interaction show that methods are unequally scalable. Static analyses, e.g., can entirely work on firmware samples, are not constrained by execution, and, thus, scale better. This influences corpus sizes, as we can see when comparing, e.g., 5,000 FirmUp [7] samples with three Avatar [28] samples. Heterogeneity and varying method applicability are related: Cross-platform code similarity, e.g., scales better across ISAs than symbolic execution for a few ARM flavors.

## III. Requirements for Sound Firmware Corpora

Firmware corpora are the data source we use to assess our binary analysis methods. Thus, their composition and documentation play important roles in evaluating method performance; they should be *scientifically sound*: The results we derive from them should be transparent, comprehensible, and verifiable; conclusions one can draw should accurately describe or approach an objective truth and clarify limitations. With the challenges from Section II in mind, we will now propose a framework of data requirements that serve as practical guidelines for the creation of sound firmware corpora.

Fig. 2 illustrates a system with three layers: There are superordinate goals, which are nurtured by six requirements. The requirements describe properties to consider for corpus creation. Goals and requirements are abstract concepts that may be hard to measure in practice. For instance, who draws the line between old and actual samples to conclude upon relevance? Thus, we identified 16 practical *measures* that feed into the system and allow estimations and comparisons on requirement fulfillment. Intuitively, there is no claim of completeness: Scenarios may need specialized measures. Covering all possibilities is infeasible when aiming for generalizability. Thus, we include items with assumed universal applicability.

### A. Goals of Scientifically Sound Firmware Corpora

Three intuitive goals shall steer us towards soundness: Replicability, Representativeness, and Method-Orientation. First, a sound corpus provides **Replicability (G1)**, allowing for independent result verification. It enables comparability, as the corpus can serve as benchmark for multiple papers. However, roadblocks hinder accessibility for third parties: C1 shows that firmware acquisition and sharing is non-trivial due to, e.g., copyright laws. C2 and C3 show the issues of unpacking, validation, and content identification to parameterize analysis methods and verify ground truth. With C1 as primary roadblock that complicates sharing of (pre-prepared) samples, the goal of Replicability forces us to apply best effort approaches within our own possibilities: To meticulously document each aspect of the corpus from acquisition to unpacking, and propagate as much meta data as legally possible. The latter helps with content identification and corpus reconstruction.

Corpora should be **Representative (G2)**. Plohmann et al. [9], who maintain a scientific malware corpus with similar

problem space as firmware corpora, explain representativeness: It *"means that the selection of samples contained in the data set should be prevalent and suitable for the deduction of results that are of real-world relevance"* [9]. For firmware corpora, this means that they should adequately model real-world distributions of *relevant* samples with heterogeneous properties like manufacturers and models. What *relevant* means will be separately discussed in Section III-B. Ideally, we can model market distributions of device properties. But those are often unknown and C1 complicates sample acquisition, which affects C8. Another practical, but less accurate, approach is to aim for the largest possible heterogeneity to demonstrate scalability, applicability, impact, and performance of an analysis method.

Malpedia [9] shares similar problems with firmware corpus creation: Malware samples are often packed or encrypted, too. Thus, both share the challenges C2 and C3, affecting ground truth (C4). This impacts verifiable representativeness; it is hard to check the contents of a box without looking into it. Thus, we adopt the *quality over quantity* credo [9]. By *quality*, we mean that all samples can be unpacked while their contents are known, deduplicated, appropriately match the scenario, and avoid data skew. Balancing firmware sample quality and quantity is a tightrope act, which becomes apparent when looking at machine learning and its common pitfalls [29]: Here, one seeks a sweet spot between quantity for training and quality to avoid overfitting and improve model performance.

Finally, we propose the goal of **Method-Orientation (G3)**: Research questions differ, papers target firmware of different availability, and analysis scalability varies. These aspects ultimately dictate the feasibility of other goals (C5 to C8). We provide three examples: First, emulative approaches can only include firmware that is successfully re-hosted, with sufficient fidelity to derive results of real-world relevance. Second, symbolic execution might be bound to certain ISAs; there is little use to include MIPS as heterogeneity property when the engine only supports ARM. Third, as HIL analyses must interact with the physical devices, how is it practicable to consider more than a handful of samples? C4 is also related since there may be varying ground truth requirements – why should a code similarity paper limit itself to samples with known bugs when the primary evaluation would greatly benefit from function symbol ground truth instead?

All of these considerations have major impact on the goals G1 and G2, especially considering quality over quantity. Yet, it appears beneficial to try and attain all other goals for the sake of soundness within the practical limitations of each scenario.

### B. Key Corpus Requirements and Their Measures

Six requirements nurture three goals in Fig. 2. There are 16 associated measures to assess their fulfillment: *Date*, *Version*, *Packed*/*Unpacked*, *Link*, *Hash*, *Vulnerabilities*, *Deduplication*, *Reasoning*, *Acquisition*, *Unpacking Process*, *Manufacturer*, *Model*, *Architecture*, *Device Class*, and *FW Type*. They can feed into multiple requirements.

**R1** *Ground Truth.* Performance evaluations should search for new and known vulnerabilities. The first demonstrates impact and proofs that the method can, indeed, find new bugs. The latter helps to verify results and paints scenarios that show what could have been if a tool would have been available in the past. As this contributes to representativeness, one should include samples with known *Vulnerabilities*.

**R2** *Relevance.* G2 demands samples of relevance. In general, we can measure it using temporal properties: Samples have a Release *Date* and *Version*. These provide information on actuality and history. Depending on a paper's scenario, they may vary in effect and meaning for relevance: Papers that search for new bugs should consider most recent versions of devices that are still in active use. Their end-of-support date could be an indicator for actuality. If papers explore, e.g., the proliferation of known bugs, history becomes relevant; then, consider old and new versions. Methods may also target certain relevant Device Properties or File Contents (cf. R6).

**R3** *Clean Data.* For replicability and sample quality, corpora should contain *Clean Data*. Sample *Deduplication* via, e.g., file hashes, helps to clean the results from findings in multiple samples with similar contents. We give two examples that can introduce duplicates to the corpus: First, scrapers might catch images that are already duplicates on the origin server. Second, some manufacturers create base firmware images for whole product lines, which only differ in few files, e.g., drivers. Reporting the sample unpacking status also contributes to cleanliness: Researchers may provide detail information on the quantities of *Packed* and effective corpus sizes of analyzable and *Unpacked Samples*.

**R4** *Rich Meta Data.* Rich meta data helps to ensure replicability within legal possibilities. Supplemental data could be file properties like Download *Links*, *Hashes*, *Versions*, and Release *Dates*. The latter two also contribute to R2. Such information helps to find samples in the Internet when links are broken. *Hashes* are, in this regard, disputable when acquisition is (semi-)invasive: When we, e.g., dump memory during runtime, it is type-dependent if an image hash is useful. ROMs can yield reusable hashes, which could help to verify extraction success. But if data changes, the hash changes too, jeopardizing its use. Similarity hashes might help in this case. Other useful data is the File Contents, e.g., Linux kernel versions, libraries used, or ISA. Device properties like *Manufacturer* and *Model* can provide further insights on sample distributions. Also, meta data serves as proof of relevance and helps third parties to easily extract corpus subsets that nurture their own research goals.

**R5** *Documentation.* Providing as much information as possible on sample *Acquisition* and the *Unpacking Process* supports replicability. Are there, e.g., any scripts or scrapers? If so, can they be shared? If not, which steps were followed to obtain samples? Which unpackers were used and how did researchers validate their success? If unpackers were custom, is it possible to share them? It is also useful to describe *Deduplication*. Understandable *Reasoning* about sample selection is another aspect of representativeness:

Which device properties are relevant for the paper and why were the samples chosen? Example reasons may be ground truth, availability, heterogeneity, or method limitations.

**R6** *Heterogeneity & Diversity.* Device properties should be as heterogeneous and diverse as possible. Example properties are *Manufacturers*, *Models*, *Architectures*, *Firmware Types*, and *Device Classes* such as routers, switches, and network cameras [11]. This helps to draw a good picture of analysis performance across different devices, demonstrates method applicability, and reduces biases introduced by certain properties. Applicability, e.g., could be misinterpreted if a corpus only includes manufacturers that do not strip required information like build artifacts [30]. Of course, greater heterogeneity implies larger sample quantity, but quality should be preferred: There is no benefit in inflating corpus sizes with similar contents or unanalyzable samples. Analysis methods ultimately dictate quantities and permutable properties. Thus, we may approach heterogeneity with best efforts according to G3 and report *Unpacked Samples*.

## IV. Analysis: Current Corpus Creation Practices in Top Tier Research

We assess if there is common ground on corpus creation practices in research: We systematically review 44 top tier papers, collect data on our framework, and use the insights to analyze and reveal methodical shortcomings in literature.

### A. Paper Selection & Data Collection Methodology

Fig. 3 shows our paper selection process. We distill top tier papers on vulnerability research that underwent rigorous peer reviews to analyze state of the art scientific practices.

Collection started by downloading all papers from *CCS*, *NDSS*, *SP*, and *USENIX Security* ①. These are the four cybersecurity conferences with the highest rating of A* in the CORE2023. For actuality, we considered papers from 2013 to 2023. We skimmed the abstracts and removed all papers that do not focus on vulnerability research ②. The resulting set contained 263 papers. We then screened their full-text for the keyword *Firmware* and removed items without a match, as they likely do not explore firmware then ③. 65 papers remained. Assuming that high-quality research references other high-quality research, we read the related work sections for referenced work between 2013 and 2023 that focuses on *Firmware* security as well. We applied the steps ② and ③ to these references, too. Thus, we effectively dropped the A* requirement and pulled in 32 more papers from workshops and conferences like *IoT SP*, *ACSAC*, *NDSS BAR*, and *RAID* ④. We skimmed the evaluation methods of the grown set of 97 papers and discarded all papers that do not create or use a firmware corpus ⑤. The final set, listed in Table I, has 44 papers from 10 workshops and conferences. We read every paper and collected data on our requirements using the 16 measures ⑥. Appendix B provides the catalogue of criteria to assess the fulfillment of all 16 measures.

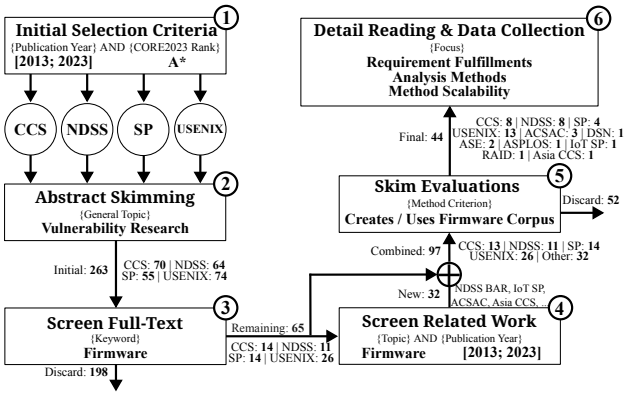


Fig. 3. Out of an initial set of 263 peer-reviewed papers from the past ten years, we distilled 44 relevant ones. For each of the remaining papers, we collected data on our 16 corpus measures, which shall support the goals of Replicability and Representativeness (cf. Section III).

TABLE I
Overview of Reviewed Research Papers

| Paper | Conference | Year | Type | Method | Scal. |
|---|---|---|---|---|---|
| Cui et al. [31] | NDSS | 2013 | S | P | ● |
| Costin et al. [11] | USENIX Security | 2014 | S | P | ● |
| Avatar [28] | NDSS | 2014 | H | SE;HIL;E | ○ |
| Pewny et al. [32] | SP | 2015 | S | CS | ◐ |
| Firmalice [33] | NDSS | 2015 | S | SE;FA | ◐ |
| PIE [34] | ACSAC | 2015 | S | FA;ML | ◐ |
| FIRMADYNE [12] | NDSS | 2016 | D | E | ● |
| discovRE [13] | NDSS | 2016 | S | CS | ● |
| Costin et al. [17] | Asia CCS | 2016 | H | P;E | ● |
| Genius [18] | CCS | 2016 | S | CS;ML | ● |
| BootStomp [35] | USENIX Security | 2017 | S | SE;FA | ◐ |
| FirmUSB [36] | CCS | 2017 | S | SE | ◐ |
| Gemini [37] | CCS | 2017 | S | CS;ML | ● |
| Muench et al. [14] | NDSS | 2018 | H | E;HIL;F | ○ |
| DTaint [38] | DSN | 2018 | S | FA | ● |
| Tian et al. [39] | USENIX Security | 2018 | S | P | ● |
| VulSeeker [40] | ASE | 2018 | S | CS;ML | ● |
| FirmUp [7] | ASPLOS | 2018 | S | CS | ● |
| IoTFuzzer [41] | NDSS | 2018 | D | HIL;F | ○ |
| FIRM-AFL [42] | USENIX Security | 2019 | D | E;F | ● |
| FirmFuzz [43] | IoT SP | 2019 | H | E;F | ● |
| SRFuzzer [44] | ACSAC | 2019 | D | HIL;F | ○ |
| Pretender [27] | RAID | 2019 | D | E;HIL | ○ |
| HALucinator [45] | USENIX Security | 2020 | H | E;F | ◐ |
| FirmScope [19] | USENIX Security | 2020 | S | FA | ● |
| PDiff [46] | CCS | 2020 | S | SA | ● |
| P$^2$IM [47] | USENIX Security | 2020 | H | E;F | ◐ |
| Karonte [8] | SP | 2020 | S | FA | ● |
| Laelaps [48] | ACSAC | 2020 | H | E;SE;F | ● |
| FirmAE [26] | ACSAC | 2020 | H | E;F | ● |
| CPscan [49] | CCS | 2021 | S | FA | ● |
| Diane [50] | SP | 2021 | H | HIL;FA;F | ○ |
| DICE [51] | SP | 2021 | D | E;F | ● |
| ECMO [52] | CCS | 2021 | H | E | ● |
| iFIZZ [53] | ASE | 2021 | H | E;HIL;F | ○ |
| Jetset [54] | USENIX Security | 2021 | H | SE;E | ◐ |
| SaTC [55] | USENIX Security | 2021 | S | FA | ● |
| Snipuzz [56] | CCS | 2021 | D | HIL;F | ○ |
| $\mu$Emu [57] | USENIX Security | 2021 | H | SE;E;F | ● |
| SymLM [58] | CCS | 2022 | S | ML | ● |
| Marcelli et al. [59] | USENIX Security | 2022 | S | ML | ● |
| Greenhouse [25] | USENIX Security | 2023 | H | E;FA;F | ● |
| FirmSolo [20] | USENIX Security | 2023 | H | E;F | ● |
| VulHawk [60] | NDSS | 2023 | S | CS;ML | ● |

**Scal**ability: ● = *Scalable*; ○ = *Not Scalable*; ◐ = *Uncertain*. **Method:** CS = *Code Similarity*; E = *Emulation*; F = *Fuzzing*; FA = *Flow Analysis*; HIL = *Hardware-In-the-Loop*; ML = *Machine Learning*; P = *Pattern*; SE = *Symbolic Execution*. **Type:** S = *Static*; D = *Dynamic*; H = *Hybrid*.

TABLE II
CORPUS CREATION PRACTICES IN TOP TIER RESEARCH FROM 2013 TO 2023: COLLECTED DATA ON THE MEASURES FOR SOUND FIRMWARE CORPORA

| | **Requirement Applies to Measure** | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Requirement** | ***Packed #*** | ***Unpacked #*** | ***Deduplication*** | ***Unpack Proc.*** | ***Reasoning*** | ***Acquisition*** | ***Vulnerabilities*** | ***Rel. Dates*** | ***Versions*** | ***Links*** | ***Hashes*** | ***Manufacturer*** | ***Models*** | ***Dev. Classes*** | ***ISAs*** | ***FW Types*** |
| **R1)** Ground Truth | – | – | – | – | – | – | ✔ | – | – | – | – | – | – | – | – | – |
| **R2)** Relevance | – | – | – | – | – | – | – | ✔ | ✔ | – | – | ✔ | ✔ | ✔ | ✔ | ✔ |
| **R3)** Clean Data | ✔ | ✔ | ✔ | – | – | – | – | – | – | – | – | – | – | – | – | – |
| **R4)** Rich Meta Data | – | – | – | – | – | – | – | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ |
| **R5)** Documentation | – | – | ✔ | ✔ | ✔ | ✔ | – | – | – | – | – | – | – | – | – | – |
| **R6)** Heterogeneity | – | ✔ | – | – | – | – | – | – | – | – | – | ✔ | ✔ | ✔ | ✔ | ✔ |
| **Paper** | **Collected Data on the Measures for Scientifically Sound Fimware Corpora** | | | | | | | | | | | | | | | | |
| Cui et al. [31] | 373 | ○ | ○ | ● | ● | ○ | ● | ● | ◐ | ○ | ○ | 1 | 63 | 1 | 2 | II |
| Costin et al. [11] | 32,356 | 26,275 | ○ | ● | ◐ | S | ◐ | ○ | ○ | ● | ● | ◐ | ◐ | ◐ | ◐ | ◐ |
| Avatar [28] | 3 | 3 | ● | ○ | ● | M | ◐ | ○ | ○ | ○ | ○ | 3 | 3 | 3 | 1 | II-III |
| Pewny et al. [32] | 6 | 6 | ● | ○ | ● | M | ● | ○ | ● | ● | ○ | 6 | 6 | 3 | 3 | 0-I |
| PIE [34] | 4 | 4 | ● | ○ | ◐ | ○ | ○ | ○ | ○ | ○ | ○ | ◐ | 4 | 4 | 1 | III |
| Firmalice [33] | 3 | 3 | ● | ○ | ● | M | ● | ○ | ○ | ○ | ○ | 3 | 3 | 3 | 2 | I |
| FIRMADYNE [12] | 23,035 | 9,486 | ● | ● | ◐ | S | ● | ● | ● | ● | ● | 42 | ◐ | ◐ | 7 | I-II |
| discovRE [13] | 3 | 3 | ● | ○ | ● | M | ● | ○ | ● | ● | ● | 3 | 3 | 3 | 4 | 0-I |
| Costin et al. [17] | 1,925 | 1,925 | ○ | ○ | ● | ○ | ◐ | ○ | ◐ | ○ | ○ | ◐ | ◐ | ◐ | 9 | I |
| Genius [18] | 33,045 | 8,126 | ○ | ○ | ○ | S;R | ◐ | ○ | ○ | ● | ○ | 26 | ◐ | ◐ | ◐ | ◐ |
| BootStomp [35] | 5 | 5 | ● | ◐ | ● | M | ● | ○ | ◐ | ○ | ○ | 4 | 4 | 1 | 1 | III |
| FirmUSB [36] | 2 | 2 | ● | ◐ | ○ | M | ○ | ○ | ○ | ○ | ○ | 2 | 2 | 1 | ◐ | III |
| Gemini [37] | 33,045 | 8,126 | ○ | ○ | ○ | R | ◐ | ○ | ○ | ● | ○ | 26 | ◐ | ◐ | ◐ | ◐ |
| Muench et al. [14] | 4 | 4 | ● | ○ | ● | M | ◐ | ○ | ○ | ○ | ○ | 4 | 4 | 4 | 1 | 0-III |
| DTaint [38] | 6 | 6 | ● | ○ | ● | ○ | ● | ○ | ● | ○ | ○ | 4 | 6 | ◐ | 2 | I |
| Tian et al. [39] | 2,018 | ◐ | ○ | ● | ● | S | ⊕ | ○ | ● | ◐ | ○ | 11 | ◐ | 1 | ⊕ | I |
| VulSeeker [40] | 4,643 | ○ | ○ | ◐ | ○ | R | ◐ | ○ | ○ | ● | ○ | ◐ | ◐ | ◐ | ◐ | ◐ |
| FirmUp [7] | ◐5,000 | ◐2,000 | ○ | ● | ○ | S | ● | ○ | ○ | ○ | ○ | ◐ | ◐ | ◐ | ◐ | ◐ |
| IoTFuzzer [41] | 17 | ⊕ | ● | ⊕ | ● | ○ | ● | ○ | ● | ○ | ○ | 12 | 17 | 10 | ◐ | ◐ |
| FIRM-AFL [42] | 11 | 11 | ● | ○ | ○ | M;R | ● | ○ | ● | ○ | ○ | 5 | 11 | 2 | ◐ | I |
| FirmFuzz [43] | 6,427 | 1,013 | ● | ○ | ● | S | ● | ○ | ○ | ○ | ○ | 3 | ◐ | 1 | 2 | I |
| SRFuzzer [44] | 10 | ⊕ | ● | ⊕ | ○ | M | ○ | ○ | ● | ○ | ○ | 5 | 10 | 1 | 2 | ◐ |
| Pretender [27] | 6 | ⊕ | ● | ⊕ | ◐ | M | ○ | ○ | ○ | ● | ● | 2 | 3 | 1 | 1 | III |
| HALucinator [45] | 16 | 16 | ● | ○ | ● | M | ◐ | ○ | ○ | ● | ○ | 3 | 4 | 1 | 1 | III |
| FirmScope [19] | 2,017 | ◐ | ○ | ● | ◐ | S | ● | ○ | ○ | ◐ | ○ | 99+ | ◐ | 1 | ⊕ | I |
| PDiff [46] | 715 | ○ | ○ | ○ | ○ | ○ | ● | ○ | ○ | ○ | ○ | 8 | ◐ | 3 | 2 | I |
| P$^2$IM [47] | 10 | 10 | ● | ◐ | ● | M | ○ | ○ | ○ | ● | ○ | 3 | 4 | 10 | 1 | II-III |
| Karonte [8] | 53;899 | ◐ | ● | ● | ● | S;R | ● | ● | ● | ● | ● | 25 | ◐ | ◐ | 3 | I-III |
| Laelaps [48] | 30 | ⊕ | ● | ◐ | ● | ○ | ○ | ○ | ○ | ○ | ○ | 2 | 4 | 24 | 1 | II-III |
| FirmAE [26] | 1,306 | 1,124 | ● | ● | ● | S | ● | ● | ○ | ● | ● | 8 | ◐ | 2 | 2 | I |
| CPscan [49] | 28 | 28 | ● | ○ | ○ | ○ | ○ | ○ | ● | ○ | ○ | 10 | 28 | ◐ | ◐ | I |
| Diane [50] | 11 | ⊕ | ● | ⊕ | ○ | ○ | ● | ○ | ● | ○ | ○ | 9 | 11 | 4 | ◐ | ◐ |
| DICE [51] | 7 | ⊕ | ● | ⊕ | ● | M | ○ | ○ | ○ | ● | ○ | 6 | 7 | 7 | 1 | II-III |
| ECMO [52] | 815 | 815 | ○ | ● | ◐ | ○ | ○ | ○ | ○ | ○ | ○ | 2 | 37 | 1 | 1 | I |
| iFIZZ [53] | 10 | 10 | ● | ● | ● | ○ | ◐ | ○ | ● | ○ | ○ | 7 | 10 | 4 | 2 | I |
| Jetset [54] | 13 | 13 | ● | ○ | ◐ | M;R | ○ | ○ | ○ | ○ | ○ | 4 | 13 | 3 | 3 | I-III |
| SaTC [55] | 39;49 | 39;49 | ● | ● | ○ | ○;R | ○ | ○ | ● | ● | ● | 6;4 | 6;◐ | 2;◐ | 2;3 | ◐ |
| Snipuzz [56] | 20 | ⊕ | ○ | ⊕ | ● | M | ○ | ○ | ● | ○ | ○ | 17 | 20 | 8 | ◐ | ◐ |
| $\mu$Emu [57] | 21 | 21 | ● | ○ | ○ | M;R | ● | ○ | ● | ● | ○ | ◐ | 21 | ◐ | 1 | II-III |
| SymLM [58] | 8 | 8 | ● | ○ | ◐ | R | ⊕ | ○ | ○ | ○ | ○ | ◐ | 8 | ◐ | 1 | II-III |
| Marcelli et al. [59] | 2 | 2 | ● | ○ | ● | M | ● | ○ | ○ | ○ | ○ | 2 | 2 | 1 | 2 | I |
| Greenhouse [25] | 7,141 | 5,690 | ● | ● | ● | S;R | ● | ○ | ○ | ○ | ○ | 9 | 1,764 | 2 | 3 | I |
| FirmSolo [20] | 8,737 | 1,470 | ● | ◐ | ● | ○;R | ● | ○ | ◐ | ◐ | ○ | ◐ | ◐ | ◐ | 2 | I |
| VulHawk [60] | 20 | 20 | ○ | ○ | ○ | ○ | ● | ○ | ○ | ○ | ○ | 3 | 20 | ◐ | ◐ | ◐ |
| LFwC (Section V) | 14,583 | 10,913 | ● | ● | ● | S | ● | ● | ● | ● | ● | 10 | 2,365 | 22 | 9 | I-II |

**Semicolon (;):** Multiple Methods/Corpora/Data Points. **Symbols:** ✔ = *Requirement Applies to Data Column Below*; ● = *Documented/Proof of Presence in Data*; ○ = *Undocumented/Proof of Absence in Data*; ◐ = *Partially Documented/Missing Data to Proof Absence or Presence*; ⊕ = *Not Applicable in Paper Scenario*. **Acquisition:** S = *Web-Scraping*; M = *Manual Collection*; R = *Samples from Related Work*. **Firmware Types:** As described in Appendix A.

### B. General Statistics & Result Overview

Table I shows the papers and gives insights on method types, i.e., static, dynamic, or hybrid. Publication year, conference, scalability, and analysis method are noted as well. Each year from 2013 to 2023 is represented, with rising quantities until 2021. Fewer papers were published in 2022 and 2023. The four most represented conferences are *USENIX Security* (13 papers), *CCS* (8), *NDSS* (8), and *SP* (4). 22 papers use static, seven dynamic, and 15 hybrid approaches. 28 methods are rated as scalable (●). FIRMADYNE [12], e.g., analyzes 9,500 samples. We rated seven papers as unscalable (○); all apply HIL. For nine papers, there was uncertainty regarding scalability (◐). HALucinator [45] and P$^2$IM [47], e.g., both use emulation but need manual modelling. Their corpora are small and do not reveal scalability. Modelling efforts can not be estimated without in-depth system experience.

Table II provides the results on all measures in our framework. The upper half maps the 16 measures to their associated requirements from Section III-B with a check mark (✔). The lower half provides the collected data for each paper. Where possible, we provide concrete values from the papers. If not, we resort to a symbolism marking complete (●), partial (◐), or missing (○) documentation. When a measure is not applicable to a specific scenario, we insert the ⊕ symbol.

The data allows us to study various practices in corpus creation. In the following, we share our observations, group the results by measure to study paper performance, and discuss selected details on common practices. Finally, we group findings by requirement and conclude upon their fulfillment.

### C. Preliminary Observations

► **A paper's scenario dictates feasible quantities.** G3 says that all numbers in Table II are relative to their paper scenario. Aspects like sample accessibility and scalability influence experiments. The data backs this claim: Table II reveals that 17 out of 44 papers use corpora between 373 and 33,000 samples. All of them use scalable methods according to Table I; the majority of them scrapes the accessible Type-I. Only one of the 17 papers includes the specialized and harder to acquire Type-III. Vice versa, 27 papers use corpora of two to 49 samples. Out of these, 70% either target Type-III or use HIL. Low quantities must not mean bad practice, as they can also reveal limits of feasibility in spite of best efforts. This does not change the fact that they introduce statistical uncertainty.

► **The measures are practicable and relevant.** We have created a framework that addresses scientifically relevant aspects of corpus creation. Thus, we proposed concrete measures to test requirement fulfillment (cf. Section III-B). We aimed to select relevant measures with universal applicability. Table II holds 704 data points across the 16 measures and 44 papers. We considered that measures may not be applicable (⊕), too. This is only true for 17 out of 704 data points: The high viability of all measures in literature let us conclude that our framework is practicable and can find broad application.

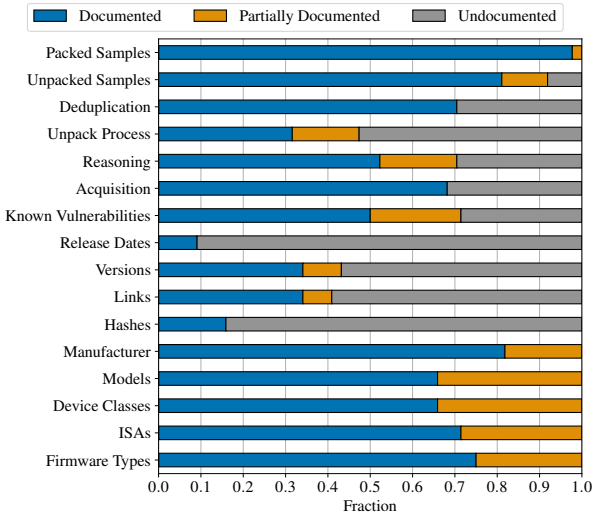


Fig. 4. Aggregated results of all collected data points for each measure in Table II. Data points that mark non-applicability of a measure are considered.

### D. Quantitative Result Analysis by Measure

We quantitatively analyze the data in Table II to identify the cumulative measure performance across all papers and discuss current practices in research. We group data by measure and convert concrete numbers to the value ●. Thus, we establish a comparison baseline using four discrete values: A paper documents the subject of a measure fully (●), partially (◐), or not at all (○). The fourth is non-applicability (⊕). We calculated the fraction of documented data points for a measure across all applicable papers. Fig. 4 shows the unweighted results.

► **[Sample Quantities] All document packed samples. Some omit unpacked quantities.** All papers precisely specify the quantity of packed samples. FirmUp [7], however, does not. The performance drops to 91% for unpacked samples: 36 papers (81%) provide precise unpacking numbers while four papers only give partial information (10%) [7], [8], [19], [39]: One gives approximations, three other include unpacking as system component but do not provide clear numbers. Three papers do not share any unpacked quantities [31], [40], [46].

► **[Deduplication] 30% of the papers do not describe sample deduplication.** As we will discuss in detail in Section IV-E, sample deduplication is important to avoid skewness in analysis results due to, e.g., duplicate findings. We note that the performance on this measure is over-evaluated in terms of documentation awareness: The 70% already includes papers that share artifacts, which helped us to determine if any deduplication took place (cf. Appendix B).

► **[Unpacking Process] 52% of the papers do not describe the unpacking process.** Sample unpacking is a significant barrier to any kind of replicability and, thus, result verification. 20 (52%) of all papers that are applicable to this measure do not document the critical unpacking process. 12 (32%) document it in detail, e.g, Greenhouse [25], Firm-

Scope [19], and Karonte [8]. Six (16%) document it partially.

► **[Reasoning] 13 papers do not justify sample selection.** It is useful for third parties to understand *why* a corpus contains certain samples, as such information gives insights on possible limitations and goals. It further helps to contextualize the work and interpret results. Possible reasons for sample selection could be, e.g., availability, required firmware properties like ISAs, or a device class of particular interest. 30% of papers do not give a reason, 18% give a reason that was not entirely comprehensible to us, and 52% justify comprehensively.

► **[Acquisition] 32% of the papers do not document acquisition.** Sharing *how* samples were acquired points independent research into the direction of corpus replication, be it through scraping or manual firmware extraction. 14 out of 44 (32%) papers do not provide any information on this matter.

► **[Known Vulnerabilities] 50% of the papers have no or incomplete documentation on the existence of known bugs in their corpora.** The existence of known vulnerabilities in corpora helps to obtain verifiable evidence showing the fruitfulness of a new analysis method. *If* there are known bugs fitting to the paper, it is a choice to use them as benchmark. 21 out of 42 papers fully document the existence of ground truth (50%). DTaint [38], e.g., rediscovers six verifiable CVEs in their corpus. Nine papers partially document this subject (21%). VulSeeker [40], e.g., searches for CVE-2015-1791, but does not explain which samples are affected in the corpus. Experiments using other CVEs are mentioned, but also not explained. 12 papers do not mention ground truth (29%).

► **[File & Temporal Properties] Release dates, versions, links, and hashes are rarely documented.** Considering temporal properties that could help to estimate relevance, only four out of 44 papers report firmware release dates (4%) and 15 (34%) report firmware versions. For the latter, there are four more papers with partial data: BootStomp [35], e.g., reports experiments on an *older* and *newer* bootloader version by Qualcomm, but does not name the identifiers. File properties beneficial to replicability are also rarely documented: 15 out of 44 papers share links for download or device acquisition. If such links become invalid, readers can fall back to file hashes to find alternative sources. Three papers provide an incomplete sample list, e.g., FirmSolo [20], who use the fully documented FIRMADYNE [12] corpus but then add 50 samples of unknown origin. Hashes are available in seven out of 44 cases.

► **[Device Properties] All papers discuss corpus composition regarding heterogeneous device properties.** In all papers, there is full or partial information on the device properties Manufacturer, Model, Device Class, ISA, and Firmware Type. This is a positive result as it shows that all papers provide insights on heterogeneity. Yet, between 25% and 34% of papers only give partial information. They can be grouped into two classes: First, there are papers that bulk scrape images but do not collect meta data, e.g., Costin et al. [11]. Second, some papers give incomplete information on these properties. FirmUp [7], e.g., lists example manufacturers, but not all of them. In both cases, some device properties remain unknown, which makes it harder to assess corpus composition.

### *E. On Quality over Quantity for Representative Results*

Our requirements are extensive and strict. As we will see in Section V, creating a corpus that caters to all included measures is difficult. It demands attention to aspects that are usually *not* core paper contributions. Furthermore, documentation occupies space that is a valuable asset in papers.

It is understandable and of little surprise that we found qualitative issues in corpus creation when we gathered the data for Table II. We found that discussions on common practices that are *not* covered by our measures provide valuable insights on the importance of the quality over quantity credo from G2:

► **Put special attention to packed and unpacked sample quantities while writing and reading.** Most papers provide all sample quantities we searched for. But we often caught ourselves revising the collected data while reading through papers, especially with large corpora. Authors should put special attention to clearly communicate corpus statistics, as imprecise wording can unintentionally skew perspectives on representativeness. Two arbitrarily chosen examples from otherwise excellent work show how easy it is to fall into this trap: Costin et al. [11] report in the abstract that they *unpacked* 32,000 images. Yet, later they describe 32,356 *processed files* and 26,275 *successfully unpacked* images; the net corpus size shrinks by 6,000. Genius [18] combines two corpora from related work [11], [12]. The abstract reports evaluations on a *data set* of 33,045 *devices*. Later statements reveal that 8,126 *firmware images were successfully unpacked* for evaluation. There is 76% loss. The former puts perspective on the importance of high-quality, analyzable samples. As for the latter, peeking into one [12] of the corpora reveals that multiple images exist for one device, showing that interchanging the terms *devices* and *firmware images* leads to ambiguity.

► **Bulk collection, combinations, and deduplication.** Data uniqueness serves sample quality, as duplicates lead to skewed analysis results. The fact that we did not find deduplication information in ten out of 17 papers with corpora of 373 or more samples is worrying, as bulk collection may catch the same sample multiple times. Thus, we can not verify that deduplication took place. The practice of combining multiple corpora from related work that scraped partially overlapping sources feeds into this issue, e.g., Feng et al. [18].

► **Contents: Open source samples inflate corpora.** OpenWrt [61] and DD-WRT [62] are two open source distributions that provide an alternative OS for consumer-grade network devices. Their container formats are well-known and can be easily unpacked. Thus, it is understandable that we observe the inclusion of such highly available and analyzable samples into corpora to evade the unpacking barrier. Yet, their disproportional inclusion inflates firmware corpora with content-based duplicates and skews heterogeneity. These projects share

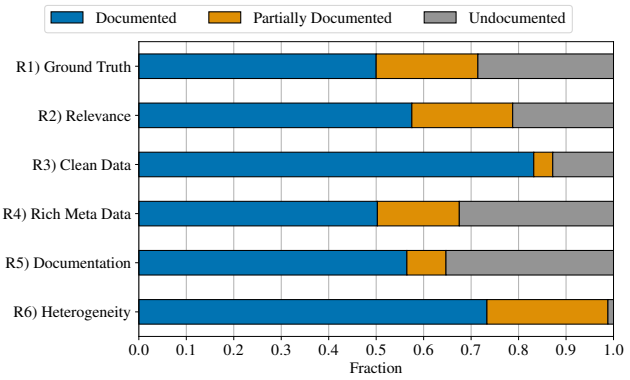


Fig. 5. Aggregates the results of all collected data points for the associated measures in Table II. The associated measures are unweighted *within* a requirement, but weighted *across* requirements, because they can contribute towards multiple goals. All 44 papers are included and data points that mark non-applicability of a measure are considered.

a common code base across devices, ISAs, and build settings. We compared the contents of two factory images of OpenWRT v23.05.0, released on 2023-10-12. We selected builds for the ASUS RT-AC87U [63] and NETGEAR R8000 [64] consumer routers and used FACT v4.2-dev [24] for unpacking. SHA256 comparisons yield that the included files overlap by 95%. Angelakopoulos et al. [20] also see this problem and point out that they removed 4,020 DD-WRT and OpenWrt samples from the 9,486 successfully unpacked samples of FIRMADYNE [12]. These samples represent 42% of the original evaluation corpus. Removing them impacts representativeness.

### F. Are Current Practices Meeting our Requirements?

Is current research meeting our requirements? Like Fig. 4, we aggregate the collected data points from Table II across all applicable papers, but group them by requirement instead. We calculate the share of full (●), partial (◐), and missing (○) documentation per requirement. Data is unweighted *within* a single requirement, but weighted *across* requirements, as some measures contribute to multiple requirements (cf. Table II).

Fig. 5 shows that researchers put effort into corpus creation. Yet, there is room for improvement: They could include more meta data for better replicability and representativeness (R4). One should provide release dates, versions, download links, and file hashes (R2, R4). Subjects covered by R5 should be thoroughly documented. Especially unpacking steps often remain unclear. Regarding our observations on the impact of the quality over quantity credo (cf. Section IV-E), we argue that there are many step stones such as missing deduplication that must be documented to draw a better picture on representativeness and provide clean data for R3. Researchers may conduct more experiments that search for known vulnerabilities in firmware (R1). Finally, it is wise to improve the precision on all aspects of R6 – through documentation or artifact sharing.

Thus, current practices in firmware vulnerability research meet our requirements only partially: None of the 44 reviewed papers documents the subject of all 16 measures. The results of this literature analysis show that there is currently no common ground on sound firmware corpus creation and documentation. Missing meta data, incomplete documentation, and inflated corpora blur visions on representativeness and replicability.

Generally, we see that otherwise excellent work may fall into the trap of the methodological and practical challenges we discussed in Section II. A brief analysis, in which we re-clustered the results from Fig. 5 by publication year, did not reveal any rising or declining trends of documentation awareness in this research branch.

## V. LFwC: A New Corpus to Demonstrate the Practicability of The Proposed Requirements

We built a proof of concept Linux Firmware Corpus (LFwC) to assess the feasibility of our requirements. It is based on data until June 2023 and consists of 10,913 deduplicated and unpacked firmware images from ten known manufacturers. It includes recent and historical firmware, covering 2,365 unique devices across 22 classes. To provide an overview of LFwC, we added corpus data points to the bottom of Table II.

We share plenty of meta data and publish all scripts and tools for replicability. We tear down LFwC's unpacking barrier with an open source process. Access to the meta data can be requested on Zenodo [16]. The tools and artifacts are available at https://github.com/fkie-cad/linux-firmware-corpus.

### A. Corpus Creation

Fig. 6 shows the corpus creation process explained below.

Ⓐ **General Reasoning.** We formulate two statements:

**A1** *Purpose & Firmware Targets.* LFwC aims to add value for vulnerability research while being as sound as possible. It shall cover multiple paper scenarios with a sizable quantity of images. We target Linux firmware, as it is prevalent in research. More precisely, we target commercial off-the-shelve (COTS) network appliances due to their availability.

**A2** *Definition of Sample Relevance & Meta Data Reasoning.* Samples with vulnerability ground truth, but also historical and actual versions, are relevant. Aside from the meta data covered by our guidelines, we aim to include insights on discovered Linux kernels and ISAs. Filtering of such information allows researchers to create sub-corpora that suit their specific scenario. OSS samples are not included.

Ⓑ **Manufacturer Selection & Device Classes.** We did not model market distributions in our corpus because there was no such information available. Instead, we have selected ten manufacturers of consumer network appliances by subjectively perceived prevalence, sample availability, and portfolio: The broader the portfolio of device classes, the better for heterogeneity. The manufacturers are: ASUS, AVM, D-Link, EDIMAX, EnGenius, Linksys, NETGEAR, TP-Link, TRENDnet,

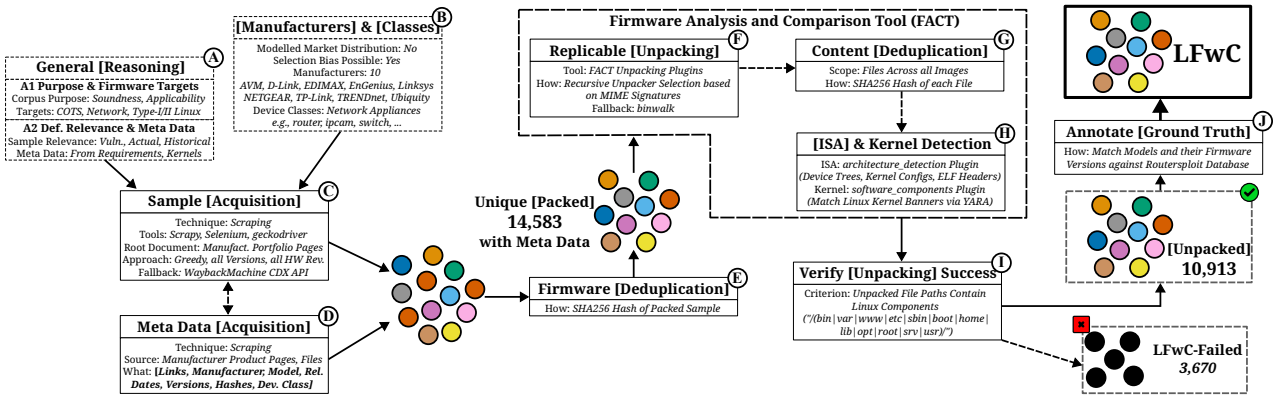


Fig. 6. We summarize and document the LFwC corpus creation process in a flow graph. There are ten processing steps, identified by the letters A-J. We provide a process revolving around the Firmware Analysis and Comparison Tool (FACT) [24] for replicable unpacking success, content deduplication, and analyses for additional meta data. Addressed measures from our Requirements Framework in Section III are marked by [Square Brackets].

and Ubiquiti. Examples for covered device classes are routers, switches, IP cameras, NAS systems, and network printers.

Ⓒ **Sample Acquisition.** For each manufacturer's page, we created scrapers with Scrapy [65] v2.9.0. We interfaced with Selenium and geckodriver v0.30.0 to render JavaScript pages. For each manufacturer, we navigated through the steps of browsing the portfolio, inspecting devices, opening the support, and downloading the firmware by hand. Along the way, we dissected page layouts to identify and implement relevant interactions for automation. We pointed the scrapers to the manufacturer portfolio overviews as root. Then, we let them traverse through the pages to collect all firmware. If there were multiple hardware or firmware versions, we downloaded images for all of them. If the vendors did not provide historical versions, we implemented fallback scrapers that use the WaybackMachine [66] to get samples from `archive.org`.

Ⓓ **Meta Data Acquisition.** The scrapers extract as much meta data as possible from the manufacturers. This includes the release date, version, manufacturer, model, and device class. As for classes, we manually assigned labels to each product line. In total, there are 22 labels: switch, router, ipcam, repeater, mesh, controller, accesspoint, powerline, modem, power_supply, wifi-usb, recorder, nas, phone, board, kvm, converter, san, printer, media, encoder, and gateway. We saved download links and calculated file hashes like SHA256, as well as the fuzzy SSDeep and TLSH.

Ⓔ **Firmware Deduplication.** We deduplicate firmware before unpacking by calculating the SHA256 sample hash. In total, we collected 14,583 unique but packed firmware files.

Ⓕ **Replicable Unpacking.** We found that the open source Firmware Analysis and Comparison Tool (FACT) [24] addresses many issues of replicable unpacking. The tool gained attention in the non-academic security community and provides plugin-based firmware processing for automated unpacking and static analyses. Plugins register themselves to process files based on their MIME type. Results are attached to each file as meta data. The contents of a firmware are both subject to analysis and further recursive unpacking, which means that the tool selects the appropriate unpacker *and protocols the unpacking results on file-basis*. The core has unpackers for over 110 file types; some of them are reverse-engineered by the community, increasing the odds of unpacking success in comparison to, e.g., the academic's default binwalk [21], which is included as fallback when other approaches fail. All collected images were uploaded to FACT v4.2-dev. We provide Vagrant [67] recipes to easily deploy FACT instances. Our shared scripts ingest the shared LFwC meta data to replicate corpus downloading and unpacking. If the included links are dead, a fallback searches on the WaybackMachine.

Ⓖ **Content Deduplication.** FACT implements hash deduplication to find duplicate files across all firmware images. Files can be correlated, similar to Costin et al. [11]. Duplicates can be pruned to clean analysis results, which solves the issue on duplicate contents in firmware corpora (cf. Section IV-E).

Ⓗ **ISAs & Kernels.** We used FACT's architecture_detection and software_components plugins. The former identifies ISAs in firmware samples. It finds device trees, Linux kernel build configurations, and ELF file headers to skim them for common architecture identifiers such as arm64 or mips32el. The latter uses YARA [68] rules to find Linux banner version strings embedded in kernel builds. This provides evidence that the firmware contains a Type-I or -II Linux system. We export the results to complement the set of meta data included in LFwC.

Ⓘ **Verify Unpacking Success.** After unpacking, we collected the files of all 14,583 firmware samples. We marked a firmware image as successfully unpacked if we were able to find common Linux path components of extracted files. Examples are */bin/*, */lib/*, and */var/*. The full list is included in Fig. 6. This approach was chosen because the absence of, e.g., a Kernel banner from step Ⓗ does not imply failures: Samples must not contain full root file systems, as it can also be an incremental device update. We verified that 10,913 samples were successfully extracted. LFwC only contains these images

TABLE III
LFwC: Corpus Statistics Overview

| Manufact. | Samples | Devices | $\frac{\text{Samples}}{\text{Device}}$ [$\mu$] | $\frac{\text{Size}}{\text{Sample}}$ [$\mu$] | $\frac{\text{Files}}{\text{Sample}}$ [$\mu$] |
|---|---|---|---|---|---|
| AVM | 797 | 201 | 3.97 | 22 MiB | 2,194 |
| TP-Link | 1,163 | 477 | 2.44 | 14 MiB | 1,446 |
| ASUS | 1,647 | 205 | 8.03 | 39 MiB | 3,780 |
| D-Link | 1,929 | 458 | 4.21 | 19 MiB | 1,234 |
| EDIMAX | 200 | 155 | 1.29 | 4 MiB | 395 |
| EnGenius | 143 | 61 | 2.34 | 9 MiB | 1,144 |
| Linksys | 308 | 166 | 1.86 | 13 MiB | 1,363 |
| NETGEAR | 2,580 | 270 | 9.56 | 24 MiB | 2,145 |
| TRENDnet | 752 | 191 | 3.94 | 9 MiB | 826 |
| Ubiquiti | 1,394 | 181 | 7.70 | 78 MiB | 11,676 |
| **Total** | 10,913 | 2,365 | 4.61 | 29 MiB | 3,219 |

and is, together with FACT, a fully unpackable corpus. The remaining 3,670 *failed* images are a separate data set we share.

Ⓙ **Vulnerability Ground Truth.** We tried to establish vulnerability ground truth. The approach is twofold: First, we map all devices in the corpus to RouterSploit [69], which consolidates exploits for network appliances. There is meta data attached to each exploit, linking to possibly affected firmware versions and security advisories. This yielded over 800 initial matches between samples and exploits. As the meta data in RouterSploit is not complete, we manually inspected the matches and compared version strings with source security advisories. We distilled 105 samples from the list with possibly applicable remote exploits, e.g., CVE-2017-5521 and CVE-2013-3093. We note that version data in security advisories is often faulty [70] and effective matching is an open research topic [71]. To find additional evidence, we manually verified the presence of CVE-2016-10177 to CVE-2016-10186 in LFwC using static analysis. This set of vulnerabilities affected versions of the D-Link DWR-932B in 2016.

### B. Brief Insights on Corpus Composition

We give brief insights on LFwC's sample composition. For more detail information, e.g., included ISAs and kernel distributions, we refer to Appendix C. Table III shows the file statistics across manufacturers. LFwC contains firmware images for 2,365 different devices and provides, on average, between four and five firmware versions per device. The mean sample size is 29 MiB and the mean number of included files is 3,219. Yet, the relatively large samples by Ubiquiti skew these numbers. With 2,580 samples (24%), NETGEAR is most represented. D-Link (18%), ASUS (15%), Ubiquiti (13%), and TP-Link (11%) follow.

Fig. 7 shows the samples per manufacturer across their year of release. LFwC has images from 2005 to June 2023, which proofs that it holds historical and recent samples. From years 2005 to 2019, we observe exponential growth in sample quantities. This makes sense considering the market growth over the past 20 years and changing patch behavior due to security awareness. Starting with the COVID-19 pandemic, there is a decrease of samples between 2020 and 2022. The data for 2023 is incomplete, as the corpus was created in June 2023. For 747 samples, we could not extract release dates.

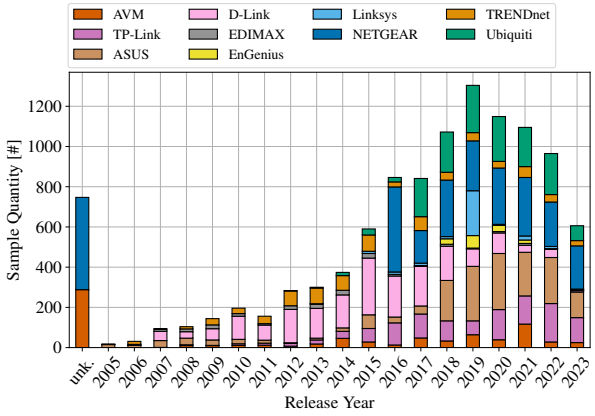


Fig. 7. LFwC firmware distribution per release date. For 747 samples, our scrapers could not extract any release date from the sources.

### C. Case Study I: Proof of Replicability

We share meta data for replication. Thus, there is an inherent risk that LFwC's included download links become inaccessible over time. The requirements in Section III propose to include file- and device-related data, e.g., product data and hashes. This information helps to acquire files from other sources.

We conducted a case study to investigate the replicability of LFwC: In June 2024, one year after creating the corpus, we simulated the scenario of an independent researcher that only uses the shared meta data and tools to reconstruct LFwC. We divided the process into four phases:

1) Pass the shared meta data to our download tool, which obtains the samples using the original links.
2) Then, use the archive.org fallback (cf. Section V-A).
3) Search firmware images on VirusTotal [72] via hashes.
4) Use the device name, file name, and version data to search for samples on forums and archives.

The first three steps were automated and the fourth was manual. We noted the phase of successful acquisition for each sample and collected the results. The total size of packed samples is 353 GiB. We used a 1 Gbps Internet connection.

Table IV shows the results per manufacturer and phase: After one year, we were able to successfully replicate 99.73% of LFwC (10,883 out of 10,913 samples) within five hours. 10,786 samples were acquired from the included direct download links, which left us with the task to recover 177 samples from alternative sources. Entering the fallbacks, we found 13 missing AVM and seven missing ASUS images on archive.org. Another 50 were found on VirusTotal [72], most of them from AVM (34) and NETGEAR (12). Then, we invested roughly 40 minutes for manual search on the Internet and gathered another 27 samples, of which we were able to verify hash identity. Most links in this category are dead because the manufacturers updated their devices and moved the older firmware packages

TABLE IV
ONE YEAR AFTER CORPUS CREATION: RESULTS FOR THE CASE STUDY ON THE INDEPENDENT REPLICABILITY OF LFWC THROUGH META DATA

| Manufact. | Samples LFwC | Samples Replicated | 1: Link Samples | 1: Link Ratio | 2: Archive Samples | 2: Archive Ratio | 3: VirusTotal Samples | 3: VirusTotal Ratio | 4: Manual Samples | 4: Manual Ratio | Missing Samples | Missing Ratio |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| AVM | 797 | 790 | **731** | **0.91** | 13 | 0.02 | 34 | 0.04 | 12 | 0.02 | 7 | 0.02 |
| TP-Link | 1,163 | 1,163 | **1,163** | **1.00** | 0 | 0.00 | 0 | 0.00 | 0 | 0.00 | 0 | 0.00 |
| ASUS | 1,647 | 1,642 | **1,633** | **0.99** | 7 | <0.01 | 2 | 0.01 | 0 | 0.00 | 5 | 0.00 |
| D-Link | 1,929 | 1,927 | **1,911** | **0.99** | 0 | 0.00 | 2 | 0.01 | 14 | 0.01 | 2 | <0.01 |
| EDIMAX | 200 | 200 | **200** | **1.00** | 0 | 0.00 | 0 | 0.00 | 0 | 0.00 | 0 | 0.00 |
| EnGenius | 143 | 143 | **143** | **1.00** | 0 | 0.00 | 0 | 0.00 | 0 | 0.00 | 0 | 0.00 |
| Linksys | 308 | 308 | **308** | **1.00** | 0 | 0.00 | 0 | 0.00 | 0 | 0.00 | 0 | 0.00 |
| NETGEAR | 2,580 | 2,564 | **2,551** | **0.98** | 0 | 0.00 | 12 | <0.01 | 1 | <0.01 | 16 | <0.01 |
| TRENDnet | 752 | 752 | **752** | **1.00** | 0 | 0.00 | 0 | 0.00 | 0 | 0.00 | 0 | 0.00 |
| Ubiquiti | 1,394 | 1,394 | **1,394** | **1.00** | 0 | 0.00 | 0 | 0.00 | 0 | 0.00 | 0 | 0.00 |
| **Total** | 10,913 | 10,883 | **10,786** | **0.99** | 20 | <0.01 | 50 | <0.01 | 27 | <0.01 | 30 | <0.01 |

TABLE V
ELF BINARIES EXTRACTED FROM LFWC

| Arch | Not Deduplicated [#k] Execs | Libs | Objs | Σ | Deduplicated [#k] Execs | Libs | Objs | Σ |
|---|---|---|---|---|---|---|---|---|
| **ARM** | 638.1 | 630.8 | 30.0 | 1,568.9 | 102.6 | 56.4 | 34.5 | 193.4 |
| **MIPS** | 416.7 | 397.5 | 90.6 | 904.8 | 87.6 | 43.3 | 26.6 | 157.5 |
| **x86** | 26.8 | 34.4 | 6.0 | 67.3 | 3.8 | 4.8 | 1.6 | 10.2 |
| **Other** | 13.2 | 6.5 | 11.1 | 30.8 | 4.3 | 1.0 | 1.7 | 7.0 |
| **Σ** | 1,094.8 | 1,069.3 | 407.8 | **2,571.8** | 198.3 | 105.5 | 64.4 | **368.2** |

**Execs:** x-pie-executable, x-executable. **Libs:** x-sharedlib, x-archive.
**Objs:** x-object.

to another URL, which was recovered. 30 samples could not be found within the 40 minutes time span of manual acquisition.

Overall, the results provide evidence that independent researchers can efficiently replicate LFwC through its meta data and automated tools; with reasonable manual effort. The fact that 99% of the samples could still be acquired through their originally scraped links, is a positive result. However, one should expect that this rate decreases over time and more samples must be collected through fallback methods.

### D. Case Study II: Proof of Scientific Corpus Utility

This brief case study demonstrates the scientific utility of LFwC: Together with FACT, we successfully perform a large-scale trend analysis using static methods.

Similar to Yu et al. [73], we raise the following question: *How did the use of binary hardening methods for ELF files change over time in the firmware corpus at hand?* We focus on five compiler-based user space methods: Canaries, Non-Executable Stacks (NX), Relocation Read-Only (RELRO), Position-Independent Code (PIC), and Fortify Source. Yu et al. [73] explain these hardening methods in detail.

With LFwC fully unpacked in FACT v4.2-dev, we query the tool's API to aggregate all included files that have common ELF MIME types. We purged all kernel objects (.ko extension) and images (FACT Linux kernel detection) from the data set.

Table V shows the data and its MIME types: We extracted 2.6m ELFs from all 10,913 images. The majority uses ARM or MIPS. FACT's deduplication helped to create a final data set of 368.2k unique ELFs across all firmware images.

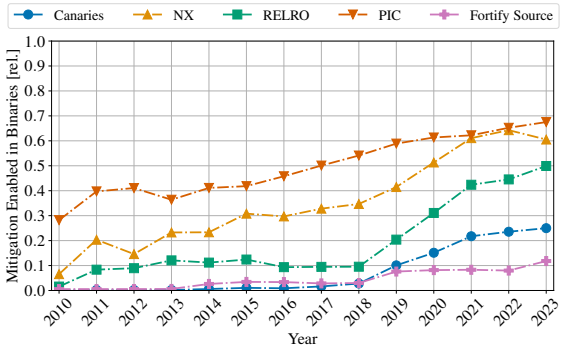


Fig. 8. ELFs with enabled hardening techniques per firmware release year.

We applied checksec v2.6.0 [74] to the 386.2k files to detect the presence of binary hardening methods in ELF headers. We traced back each ELF to its origin firmware, associate the firmware's release date, and grouped the analysis results by year. For each method, Fig. 8 plots the fraction of ELFs included in firmware from 2010 to 2023 where the method is enabled. From 2010 to 2023, usage grows steadily for all hardening methods. While PIC is most often enabled and grew almost linearly from 2010 (30%) to 2023 (67%), we observe an accelerating trend for all other methods starting with 2018. As of 2023, NX (60%) almost catched up to PIC, followed by RELRO (50%), Canaries (25%), and Fortify Source (12%). Overall, we can observe a positive trend in our data set. This is reasonable, as security awareness grew and resources needed to use these methods became more affordable.

This contradicts the results by Yu et al. [73] from 2022: As shown in Figure 4 of their paper, they observe almost 100% NX usage, while Canaries fluctuate, and the other methods have almost no change in adoption rates. The authors note that their data set does not seem balanced, as five out of 34 manufacturers provide 78% of the ELF binaries. As Synology makes up for 46% of all ELFs, changes in their build chain can significantly impact results.

Also, we can use LFwC to show that the 100% NX adoption rate from Yu et al. [73] can not be representative. NX requires CPU support, as the hardware has to ultimately protect memory regions from execution. ARM and x86 families support

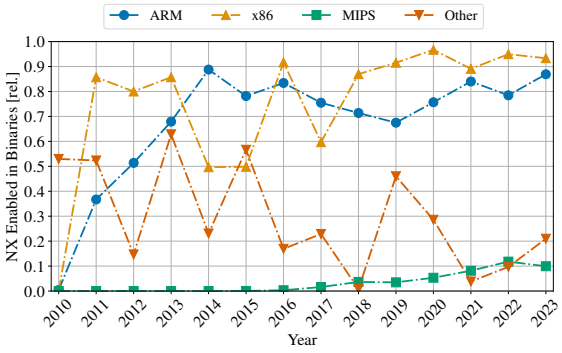


Fig. 9. ELFs with enabled NX bit per release year and ISA.

NX since decades, but MIPS specifications only introduced this feature recently as eXecute Inhibit (XI) [75]. As there is a certain time-to-market for each specification, we can assume that fewer MIPS devices support this feature than x86 and ARM devices. Fig. 9 shows the adoption rate of NX in our corpus by architecture over the past decade. As expected, the ARM and x86-based ELFs almost all adopted NX as of 2023. However, for MIPS, NX adoption only started in 2017. Six years later, the adoption rate is at 10%, which is reasonable. As we show in Table V, 42% of the ELF files in our corpus are MIPS and ISAs other than ARM and x86 are negligible. At the same time, 41% of the ELF files are MIPS in the corpus by Yu et al. [73]. Thus, both data sets should be comparable, but the results vastly deviate. We invite interested researchers to further explore LFwC and investigate the observations we made as part of this short case study, in which we showed that our corpus can be used for static analyses.

### *E. Ethical Discussion on Construction and Distribution*

For ethical firmware collection, we taught our scrapers to obey to the `robots.txt` of manufacturers. We assume that said file correctly implements any restrictions mentioned in their ToS/EULA. We also reduced infrastructure load by throttling requests and execution threads. The replication scripts we provide are equally throttled. We did not greedily mirror and process all information available on the manufacturer's pages. These pages can include data, e.g., forum posts, that might fall under special data protection laws. Thus, we designed our scrapers in a way that steers them directly towards the desired sample downloads over the product pages. These were, alongside the specifically included meta data fields, the only data we persisted. Furthermore, we argue that it is unethical to include detail FACT [24] reports in the meta data we share. These include security-relevant data, e.g., file system trees, hashes of included files, or complete lists of detected software components. This data provides a low entry barrier for malicious actors to perform large-scale presence checks of emerging zero day vulnerabilities across thousands of devices. Thus, we decided to control access to the meta data. LFwC shall help to improve, and not jeopardize, device security.

### *F. Corpus Limitations & Future Expansion*

Our creation methods imply various limitations. First, we criticize the sample distribution: As we only scrape easily accessible, network-centric samples, LFwC does not include devices where acquisition is less scalable, more complex, and invasive. Cisco IOS images, e.g., are gated by login pages. We further limited the sources to ten manufacturers. Thus, market distributions are not accurately modelled and trend analyses on samples might lack fidelity. OSS samples are also not included. Generally, LFwC is limited to Type-I and -II Linux images. Their distribution is unknown: Device-specifics like timing constraints might incentivize manufacturers to implement user space applications in kernel space, which blurs the lines between the two classes (cf. Appendix A). A detail content analysis would be required for all 10,913 samples to distinguish, which is infeasible within the scope of this paper.

Second, LFwC's compatibility is limited to static methods. We can not test the presence of execution parameters that are possibly needed without having concrete dynamic scenarios at hand. This does not mean that it is unsuitable for dynamic analyses, but that we did not verify LFwC's compatibility.

Third, we note that FACT uses static heuristics for unpacking, verification, and meta data acquisition. We also mapped Routersploit ground truth statically and manually. Thus, there is a chance of false-positive findings in the corpus meta data.

Finally, we observe that all shared meta data for replication remains volatile and the relevancy of samples fades as the corpus ages. Thus, we plan to provide at least annual updates that refresh the links and add new manufacturers or devices to the corpus. As our scrapers are open source, we invite researchers to join us in this task.

## VI. Conclusion

We brought attention to an important, but often overlooked aspect of binary vulnerability research: The creation of representative and replicable firmware corpora for sound and independently verifiable experiments. For this purpose, we pinpointed eight challenges that can significantly affect corpus creation. We used them to derive a strict framework of corpus requirements, nurtured by 16 measures. The goal was to give broadly applicable and practical guidelines that can be used to improve soundness – given current copyright laws and unpacking barriers. We revised the status quo of corpus creation practices through a systematic literature review on top tier research. We showed that our framework serves its purpose, as it helps to find and avoid many methodological pitfalls: Missing meta data, incomplete documentation, and inflated corpus sizes blur visions on representativeness and hinder replicability. Our measures show practical ways, like an extended set of meta data, that one could implement to improve corpus soundness.

The requirements led to the creation of LFwC, a new Linux firmware corpus with rich meta data, high-quality deduplicated

samples, and verified contents. It is a valuable addition to the community, as it demonstrates verified replicability, shows its scientific utility, and takes the needed space to transparently document creation and composition. Together with our requirements, it is a step towards scientifically sound corpora. We plan yearly updates to preserve relevance and replicability.

## References


[1] M. Antonakakis, T. April, M. Bailey, M. Bernhard, E. Bursztein, J. Cochran, Z. Durumeric, J. A. Halderman, L. Invernizzi, M. Kallitsis, D. Kumar, C. Lever, Z. Ma, J. Mason, D. Menscher, C. Seaman, N. Sullivan, K. Thomas, and Y. Zhou, "Understanding the Mirai Botnet," in *Proceedings of the USENIX Security Symposium (USENIX Security 17)*. Vancouver, British Columbia, Canada: USENIX Association, 2017, pp. 1093–1110. [Online]. Available: https://www.usenix.org/conference/usenixsecurity17/technical-sessions/presentation/antonakakis

[2] T. M. Chen and S. Abu-Nimeh, "Lessons from Stuxnet," *Computer*, vol. 44, no. 4, pp. 91–93, 2011. [Online]. Available: https://dx.doi.org/10.1109/MC.2011.115

[3] W. Largent. New VPNFilter Malware Targets at least 500K Networking Devices Worldwide. [Online]. Available: https://blog.talosintelligence.com/vpnfilter/

[4] DRAGOS, Inc. CHERNOVITE's PIPEDREAM Malware Targeting Industrial Control Systems (ICS). [Online]. Available: https://www.dragos.com/blog/industry-news/chernovite-pipedream-malware-targeting-industrial-control-systems/

[5] A. Qasem, P. Shirani, M. Debbabi, L. Wang, B. Lebel, and B. L. Agba, "Automatic Vulnerability Detection in Embedded Devices and Firmware: Survey and Layered Taxonomies," *ACM Computing Surveys*, vol. 54, no. 2, 2021. [Online]. Available: https://doi.org/10.1145/3432893

[6] C. Wright, W. A. Moeglein, S. Bagchi, M. Kulkarni, and A. A. Clements, "Challenges in Firmware Re-Hosting, Emulation, and Analysis," *ACM Computing Surveys*, vol. 54, no. 1, 2021. [Online]. Available: https://doi.org/10.1145/3423167

[7] Y. David, N. Partush, and E. Yahav, "FirmUp: Precise Static Detection of Common Vulnerabilities in Firmware," in *Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS'18)*. Williamsburg, VA, USA: Association for Computing Machinery, 2018, p. 392–404. [Online]. Available: https://doi.org/10.1145/3173162.3177157

[8] N. Redini, A. Machiry, R. Wang, C. Spensky, A. Continella, Y. Shoshitaishvili, C. Kruegel, and G. Vigna, "Karonte: Detecting Insecure Multi-binary Interactions in Embedded Firmware," in *Proceedings of the IEEE Symposium on Security and Privacy (SP'20)*. Virtual Conference: IEEE, 2020, pp. 1544–1561. [Online]. Available: https://dx.doi.org/10.1109/SP40000.2020.00036

[9] D. Plohmann, M. Clauß, S. Enders, and E. Padilla, "Malpedia: A Collaborative Effort to Inventorize the Malware Landscape," in *Proceedings of the Botnet & Malware Ecosystems Fighting Conference (Botconf'17)*, Montpellier, France, 2017. [Online]. Available: https://www.botconf.eu/botconf-presentation-or-article/malpedia-a-collaborative-effort-to-inventorize-the-malware-landscape/

[10] G. Klees, A. Ruef, B. Cooper, S. Wei, and M. Hicks, "Evaluating Fuzz Testing," in *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security (CCS'18)*. Toronto, Canada: Association for Computing Machinery, 2018, p. 2123–2138. [Online]. Available: https://doi.org/10.1145/3243734.3243804

[11] A. Costin, J. Zaddach, A. Francillon, and D. Balzarotti, "A Large-Scale Analysis of the Security of Embedded Firmwares," in *Proceedings of the USENIX Security Symposium (USENIX Security 14)*. San Diego, California, USA: USENIX Association, 2014, pp. 95–110. [Online]. Available: https://www.usenix.org/system/files/conference/usenixsecurity14/sec14-paper-costin.pdf

[12] D. D. Chen, M. Egele, M. Woo, and D. Brumley, "Towards Automated Dynamic Analysis for Linux-based Embedded Firmware," in *Proceedings of the Network and Distributed System Security Symposium (NDSS'16)*. San Diego, California, USA: The Internet Society, 2016. [Online]. Available: https://dx.doi.org/10.14722/ndss.2016.23415

[13] S. Eschweiler, K. Yakdan, and E. Gerhards-Padilla, "discovRE: Efficient Cross-Architecture Identification of Bugs in Binary Code," in *Proceedings of the Network and Distributed System Security Symposium (NDSS'16)*. San Diego, California, USA: The Internet Society, 2016. [Online]. Available: https://dx.doi.org/10.14722/ndss.2016.23185

[14] M. Muench, J. Stijohann, F. Kargl, A. Francillon, and D. Balzarotti, "What You Corrupt Is Not What You Crash: Challenges in Fuzzing Embedded Devices," in *Proceedings of the Network and Distributed System Security Symposium (NDSS'18)*. San Diego, California, USA: The Internet Society, 2018. [Online]. Available: https://dx.doi.org/10.14722/ndss.2018.23166

[15] S. Geng, Y. Li, Y. Du, J. Xu, Y. Liu, and B. Mao, "An Empirical Study on Benchmarks of Artificial Software Vulnerabilities," *arXiv*, 2020. [Online]. Available: http://arxiv.org/abs/2003.09561

[16] R. Helmke, "Linux Firmware Corpus - Full LFwC Meta Data for Replication," Jul. 2024. [Online]. Available: https://doi.org/10.5281/zenodo.12659436

[17] A. Costin, A. Zarras, and A. Francillon, "Automated Dynamic Firmware Analysis at Scale: A Case Study on Embedded Web Interfaces," in *Proceedings of the Asia Conference on Computer and Communications Security (Asia CCS'16)*. Xi'an, China: Association for Computing Machinery, 2016, p. 437–448. [Online]. Available: https://doi.org/10.1145/2897845.2897900

[18] Q. Feng, R. Zhou, C. Xu, Y. Cheng, B. Testa, and H. Yin, "Scalable Graph-Based Bug Search for Firmware Images," in *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security (CCS'16)*. Vienna, Austria: Association for Computing Machinery, 2016, pp. 480–491. [Online]. Available: https://dx.doi.org/10.1145/2976749.2978370

[19] M. Elsabagh, R. Johnson, A. Stavrou, C. Zuo, Q. Zhao, and Z. Lin, "FIRMSCOPE: Automatic Uncovering of Privilege-Escalation Vulnerabilities in Pre-Installed Apps in Android Firmware," in *Proceedings of the USENIX Security Symposium (USENIX Security 20)*. Virtual Conference: USENIX Association, 2020, pp. 2379–2396. [Online]. Available: https://www.usenix.org/system/files/sec20-elsabagh.pdf

[20] I. Angelakopoulos, G. Stringhini, and M. Egele, "FirmSolo: Enabling Dynamic Analysis of Binary Linux-based IoT Kernel Modules," in *Proceedings of the USENIX Security Symposium (USENIX Security 23)*. Anaheim, California, USA: USENIX Association, 2023, pp. 5021–5038. [Online]. Available: https://www.usenix.org/system/files/usenixsecurity23-angelakopoulos.pdf

[21] ReFirmLabs. Binwalk. [Online]. Available: https://github.com/ReFirmLabs/binwalk

[22] A. Hemel. The Binary Analysis Tool (BAT). [Online]. Available: https://github.com/armijnhemel/binaryanalysis

[23] ONEKEY. Unblob. [Online]. Available: https://unblob.org/

[24] Fraunhofer FKIE. FACT - Firmware Analysis and Comparison Tool. [Online]. Available: https://github.com/fkie-cad/FACT_core

[25] H. J. Tay, K. Zeng, J. M. Vadayath, A. S. Raj, A. Dutcher, T. Reddy, W. Gibbs, Z. L. Basque, F. Dong, Z. Smith, A. Doupé, T. Bao, Y. Shoshitaishvili, and R. Wang, "Greenhouse: Single-Service Rehosting of Linux-Based Firmware Binaries in User-Space Emulation," in *Proceedings of the USENIX Security Symposium (USENIX Security 23)*. Anaheim, California, USA: USENIX Association, 2023, pp. 5791–5808. [Online]. Available: https://www.usenix.org/system/files/usenixsecurity23-tay.pdf

[26] M. Kim, D. Kim, E. Kim, S. Kim, Y. Jang, and Y. Kim, "FirmAE: Towards Large-Scale Emulation of IoT Firmware for Dynamic Analysis," in *Proceedings of the Annual Computer Security Applications Conference (ACSAC'20)*. Austin, Texas, USA: Association for Computing Machinery, 2020, pp. 733–745. [Online]. Available: https://doi.org/10.1145/3427228.3427294

[27] E. Gustafson, M. Muench, C. Spensky, N. Redini, A. Machiry, Y. Fratantonio, D. Balzarotti, A. Francillon, Y. R. Choe, C. Kruegel, and G. Vigna, "Toward the Analysis of Embedded Firmware through Automated Re-hosting," in *Proceedings of the International Symposium on Research in Attacks, Intrusions and Defenses (RAID'19)*. Chaoyang District, Beijing, China: USENIX Association, 2019, pp. 135–150. [Online]. Available: https://www.usenix.org/system/files/raid2019-gustafson.pdf

[28] J. Zaddach, L. Bruno, A. Francillon, and D. Balzarotti, "Avatar: A Framework to Support Dynamic Security Analysis of Embedded Systems' Firmwares," in *Proceedings of the Network and Distributed*

*System Security Symposium (NDSS'14)*. San Diego, California, USA: The Internet Society, 2014. [Online]. Available: https://dx.doi.org/10.14722/ndss.2014.23229

[29] D. Arp, E. Quiring, F. Pendlebury, A. Warnecke, F. Pierazzi, C. Wressnegger, L. Cavallaro, and K. Rieck, "Dos and Don'ts of Machine Learning in Computer Security," in *Proceedings of the USENIX Security Symposium (USENIX Security 22)*. Boston, Maryland, USA: USENIX Association, 2022, pp. 3971–3988. [Online]. Available: https://www.usenix.org/system/files/sec22-arp.pdf

[30] R. Helmke and J. vom Dorp, "Extended abstract: Towards reliable and scalable linux kernel cve attribution in automated static firmware analyses," in *Proceedings of the International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment (DIMVA'23)*. Hamburg, Germany: Springer-Verlag, 2023, pp. 201–210. [Online]. Available: https://doi.org/10.1007/978-3-031-35504-2_10

[31] M. C. Ang Cui and S. J. Stolfo, "When Firmware Modifications Attack: A Case Study of Embedded Exploitation," in *Proceedings of the Network and Distributed System Security Symposium (NDSS'13)*. San Diego, California, USA: The Internet Society, 2013. [Online]. Available: https://www.ndss-symposium.org/wp-content/uploads/2017/09/03_4_0.pdf

[32] J. Pewny, B. Garmany, R. Gawlik, C. Rossow, and T. Holz, "Cross-Architecture Bug Search in Binary Executables," in *Proceedings of the IEEE Symposium on Security and Privacy (SP'15)*. San Jose, California, USA: IEEE, 2015, pp. 709–724. [Online]. Available: https://dx.doi.org/10.1109/SP.2015.49

[33] Y. Shoshitaishvili, R. Wang, C. Hauser, C. Kruegel, and G. Vigna, "Firmalice - Automatic Detection of Authentication Bypass Vulnerabilities in Binary Firmware," in *Proceedings of the Network and Distributed System Security Symposium (NDSS'15)*. San Diego, California, USA: The Internet Society, 2015. [Online]. Available: https://dx.doi.org/10.14722/ndss.2015.23294

[34] L. Cojocar, J. Zaddach, R. Verdult, H. Bos, A. Francillon, and D. Balzarotti, "PIE: Parser Identification in Embedded Systems," in *Proceedings of the Annual Computer Security Applications Conference (ACSAC'15)*. Los Angeles, California, USA: Association for Computing Machinery, 2015, pp. 251–260. [Online]. Available: https://doi.org/10.1145/2818000.2818035

[35] N. Redini, A. Machiry, D. Das, Y. Fratantonio, A. Bianchi, E. Gustafson, Y. Shoshitaishvili, C. Kruegel, and G. Vigna, "BootStomp: On the Security of Bootloaders in Mobile Devices," in *Proceedings of the USENIX Security Symposium (USENIX Security 17)*. Vancouver, BC, Canada: USENIX Association, 2017, pp. 781–798. [Online]. Available: https://www.usenix.org/system/files/conference/usenixsecurity17/sec17-redini.pdf

[36] G. Hernandez, F. Fowze, D. J. Tian, T. Yavuz, and K. R. Butler, "FirmUSB: Vetting USB Device Firmware Using Domain Informed Symbolic Execution," in *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security (CCS'17)*. Dallas, Texas, USA: Association for Computing Machinery, 2017, pp. 2245–2262. [Online]. Available: https://doi.org/10.1145/3133956.3134050

[37] X. Xu, C. Liu, Q. Feng, H. Yin, L. Song, and D. Song, "Neural Network-Based Graph Embedding for Cross-Platform Binary Code Similarity Detection," in *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security (CCS'17)*. Association for Computing Machinery, 2017, pp. 363–376. [Online]. Available: https://dx.doi.org/10.1145/3133956.3134018

[38] K. Cheng, Q. Li, L. Wang, Q. Chen, Y. Zheng, L. Sun, and Z. Liang, "DTaint: Detecting the Taint-Style Vulnerability in Embedded Device Firmware," in *Proceedings of the IEEE/IFIP International Conference on Dependable Systems and Networks (DSN'18)*. Luxembourg: IEEE, 2018, pp. 430–441. [Online]. Available: https://dx.doi.org/10.1109/DSN.2018.00052

[39] D. J. Tian, G. Hernandez, J. I. Choi, V. Frost, C. Raules, P. Traynor, H. Vijayakumar, L. Harrison, A. Rahmati, M. Grace, and K. R. B. Butler, "ATtention Spanned: Comprehensive Vulnerability Analysis of AT Commands Within the Android Ecosystem," in *Proceedings of the USENIX Security Symposium (USENIX Security 18)*. Baltimore, Maryland, USA: USENIX Association, 2018, pp. 273–290. [Online]. Available: https://www.usenix.org/system/files/conference/usenixsecurity18/sec18-tian.pdf

[40] J. Gao, X. Yang, Y. Fu, Y. Jiang, and J. Sun, "VulSeeker: A Semantic Learning Based Vulnerability Seeker for Cross-Platform Binary," in *Proceedings of the ACM/IEEE International Conference on Automated Software Engineering (ASE'18)*. Montpellier, France: Association for Computing Machinery, 2018. [Online]. Available: https://dx.doi.org/10.1145/3238147.3240480

[41] J. Chen, W. Diao, Q. Zhao, C. Zuo, Z. Lin, X. Wang, W. C. Lau, M. Sun, R. Yang, and K. Zhang, "IoTFuzzer: Discovering Memory Corruptions in IoT Through App-based Fuzzing," in *Proceedings of the Network and Distributed System Security Symposium*. San Diego, California, USA: The Internet Society, 2018. [Online]. Available: https://dx.doi.org/10.14722/ndss.2018.23159

[42] Y. Zheng, A. Davanian, H. Yin, C. Song, H. Zhu, and L. Sun, "FIRM-AFL: High-Throughput Greybox Fuzzing of IoT Firmware via Augmented Process Emulation," in *Proceedings of the USENIX Security Symposium (USENIX Security 19)*. Santa Clara, California, USA: USENIX Association, 2019, pp. 1099–1114. [Online]. Available: https://www.usenix.org/system/files/sec19-zheng_0.pdf

[43] P. Srivastava, H. Peng, J. Li, H. Okhravi, H. Shrobe, and M. Payer, "FirmFuzz: Automated IoT Firmware Introspection and Analysis," in *Proceedings of the International ACM Workshop on Security and Privacy for the Internet-of-Things (IoT SP'19)*. London, United Kingdom: Association for Computing Machinery, 2019, pp. 15–21. [Online]. Available: https://dx.doi.org/10.1145/3338507.3358616

[44] Y. Zhang, W. Huo, K. Jian, J. Shi, H. Lu, L. Liu, C. Wang, D. Sun, C. Zhang, and B. Liu, "SRFuzzer: An Automatic Fuzzing Framework for Physical SOHO Router Devices to Discover Multi-Type Vulnerabilities," in *Proceedings of the Annual Computer Security Applications Conference (ACSAC'19)*. San Juan, Puerto Rico, USA: Association for Computing Machinery, 2019, pp. 544–556. [Online]. Available: https://dx.doi.org/10.1145/3359789.3359826

[45] A. A. Clements, E. Gustafson, T. Scharnowski, P. Grosen, D. Fritz, C. Kruegel, G. Vigna, S. Bagchi, and M. Payer, "HALucinator: Firmware Re-hosting Through Abstraction Layer Emulation," in *Proceedings of the USENIX Security Symposium (USENIX Security 20)*. Virtual Conference: USENIX Association, 2020, pp. 1201–1218. [Online]. Available: https://www.usenix.org/system/files/sec20-clements.pdf

[46] Z. Jiang, Y. Zhang, J. Xu, Q. Wen, Z. Wang, X. Zhang, X. Xing, M. Yang, and Z. Yang, "PDiff: Semantic-based Patch Presence Testing for Downstream Kernels," in *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security (CCS'20)*. Virtual Conference: Association for Computing Machinery, 2020, pp. 1149–1163. [Online]. Available: https://dx.doi.org/10.1145/3372297.3417240

[47] B. Feng, A. Mera, and L. Lu, "P$^2$IM: Scalable and hardware-independent firmware testing via automatic peripheral interface modeling," in *Proceedings of the USENIX Security Symposium (USENIX Security 20)*. Virtual Conference: USENIX Association, 2020. [Online]. Available: https://www.usenix.org/system/files/sec20-feng.pdf

[48] C. Cao, L. Guan, J. Ming, and P. Liu, "Device-Agnostic Firmware Execution is Possible: A Concolic Execution Approach for Peripheral Emulation," in *Proceedings of the Annual Computer Security Applications Conference (ACSAC'20)*. Austin, Texas, USA: Association for Computing Machinery, 2020, pp. 746–759. [Online]. Available: https://doi.org/10.1145/3427228.3427280

[49] L. Fu, S. Ji, K. Lu, P. Liu, X. Zhang, Y. Duan, Z. Zhang, W. Chen, and Y. Wu, "CPscan: Detecting Bugs Caused by Code Pruning in IoT Kernels," in *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security (CCS'21)*. Virtual Conference: Association for Computing Machinery, 2021, pp. 794–810. [Online]. Available: https://dx.doi.org/10.1145/3460120.3484738

[50] N. Redini, A. Continella, D. Das, G. De Pasquale, N. Spahn, A. Machiry, A. Bianchi, C. Kruegel, and G. Vigna, "Diane: Identifying Fuzzing Triggers in Apps to Generate Under-constrained Inputs for IoT Devices," in *Proceedings of the IEEE Symposium on Security and Privacy (SP'21)*. Virtual Conference: IEEE, 2021, pp. 484–500. [Online]. Available: https://dx.doi.org/10.1109/SP40001.2021.00066

[51] A. Mera, B. Feng, L. Lu, and E. Kirda, "DICE: Automatic Emulation of DMA Input Channels for Dynamic Firmware Analysis," in *Proceedings of the IEEE Symposium on Security and Privacy (SP'21)*. Virtual Conference: IEEE, 2021, pp. 1938–1954. [Online]. Available: https://dx.doi.org/10.1109/SP40001.2021.00018

[52] M. Jiang, L. Ma, Y. Zhou, Q. Liu, C. Zhang, Z. Wang, X. Luo, L. Wu, and K. Ren, "ECMO: Peripheral Transplantation to Rehost Embedded Linux Kernels," in *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security (CCS'21)*. Virtual Conference: Association for Computing Machinery, 2021, pp. 734–748. [Online]. Available: https://doi.org/10.1145/3460120.3484753

[53] P. Liu, S. Ji, X. Zhang, Q. Dai, K. Lu, L. Fu, W. Chen, P. Cheng, W. Wang, and R. Beyah, "IFIZZ: Deep-State and Efficient Fault-Scenario Generation to Test IoT Firmware," in *Proceedings of the International Conference on Automated Software Engineering (ASE'21)*. Virtual Conference: Association for Computing Machinery, 2021, pp. 805–816. [Online]. Available: https://dx.doi.org/10.1109/ASE51524.2021.9678785

[54] E. Johnson, M. Bland, Y. Zhu, J. Mason, S. Checkoway, S. Savage, and K. Levchenko, "Jetset: Targeted Firmware Rehosting for Embedded Systems," in *Proceedings of the USENIX Security Symposium (USENIX Security 21)*, Virtual Conference, 2021, pp. 321–338. [Online]. Available: https://www.usenix.org/system/files/sec21-johnson.pdf

[55] L. Chen, Y. Wang, Q. Cai, Y. Zhan, H. Hu, J. Linghu, Q. Hou, C. Zhang, H. Duan, and Z. Xue, "Sharing More and Checking Less: Leveraging Common Input Keywords to Detect Bugs in Embedded Systems," in *Proceedings of the USENIX Security Symposium (USENIX Security 21)*. Virtual Conference: USENIX Association, 2021, pp. 303–319. [Online]. Available: https://www.usenix.org/system/files/sec21-chen-libo.pdf

[56] X. Feng, R. Sun, X. Zhu, M. Xue, S. Wen, D. Liu, S. Nepal, and Y. Xiang, "Snipuzz: Black-Box Fuzzing of IoT Firmware via Message Snippet Inference," in *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security (CCS'21)*. Virtual Conference: Association for Computing Machinery, 2021, pp. 337–350. [Online]. Available: https://doi.org/10.1145/3460120.3484543

[57] W. Zhou, L. Guan, P. Liu, and Y. Zhang, "Automatic Firmware Emulation through Invalidity-guided Knowledge Inference," in *Proceedings of the USENIX Security Symposium (USENIX Security 21)*. Virtual Conference: USENIX Association, 2021, pp. 2007–2024. [Online]. Available: https://www.usenix.org/system/files/sec21-zhou.pdf

[58] X. Jin, K. Pei, J. Y. Won, and Z. Lin, "SymLM: Predicting Function Names in Stripped Binaries via Context-Sensitive Execution-Aware Code Embeddings," in *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security (CCS'22)*. Los Angeles, California, USA: Association for Computing Machinery, 2022, pp. 1631–1645. [Online]. Available: https://dx.doi.org/10.1145/3548606.3560612

[59] A. Marcelli, M. Graziano, X. Ugarte-Pedrero, Y. Fratantonio, M. Mansouri, and D. Balzarotti, "How Machine Learning is Solving the Binary Function Similarity Problem," in *Proceedings of the USENIX Security Symposium (USENIX Security 22)*. Boston, Massachusetts, USA: USENIX Association, 2022, pp. 2099–2116. [Online]. Available: https://www.usenix.org/system/files/sec22-marcelli.pdf

[60] Z. Luo, P. Wang, B. Wang, Y. Tang, W. Xie, X. Zhou, D. Liu, and K. Lu, "VulHawk: Cross-architecture Vulnerability Detection with Entropy-based Binary Code Search," in *Proceedings of the Network and Distributed System Security Symposium (NDSS'23)*. San Diego, California, USA: The Internet Society, 2023. [Online]. Available: https://dx.doi.org/10.14722/ndss.2023.24415

[61] The OpenWrt Project. OpenWrt. [Online]. Available: https://openwrt.org/

[62] embeDD GmbH. The DD-WRT Project. [Online]. Available: https://dd-wrt.com/

[63] The OpenWrt Project. Factory Image v23.05.0 for ASUS RT-AC87U. Release: 2023-10-12, MD5: fd3580876e26e34a870f9c6e935cc296. [Online]. Available: https://downloads.openwrt.org/releases/23.05.0/targets/bcm53xx/generic/openwrt-23.05.0-bcm53xx-generic-asus_rt-ac87u-squashfs.trx

[64] ——. Factory Image v23.05.0 for NETGEAR R8000. Release: 2023-10-12, MD5: 130b8dce90fa899be3ccb6f1d2c61aaf. [Online]. Available: https://downloads.openwrt.org/releases/23.05.0/targets/bcm53xx/generic/openwrt-23.05.0-bcm53xx-generic-netgear_r8000-squashfs.chk

[65] Zyte. Scrapy - An open source and collaborative framework for extracting the data you need from websites. [Online]. Available: https://scrapy.org/

[66] The Internet Archive. Wayback CDX Server API. [Online]. Available: https://archive.org/developers/wayback-cdx-server.html

[67] HashiCorp. Vagrant - Development Environments Simplified. [Online]. Available: https://www.vagrantup.com/

[68] VirusTotal. YARA - The Pattern Matching swiss Knife for Malware Researchers. [Online]. Available: https://virustotal.github.io/yara/

[69] threat9. RouterSploit - Exploitation Framework for Embedded Devices. [Online]. Available: https://github.com/threat9/routersploit

[70] L. A. Benthin Sanguino and R. Uetz, "Software Vulnerability Analysis Using CPE and CVE," *ArXiv*, 2017. [Online]. Available: https://doi.org/10.48550/arXiv.1705.05347

[71] Y. Dong, W. Guo, Y. Chen, X. Xing, Y. Zhang, and G. Wang, "Towards the Detection of Inconsistencies in Public Security Vulnerability Reports," in *Proceedings of the USENIX Security Symposium (USENIX Security 19)*. Santa Clara, California, USA: USENIX Association, 2019, pp. 869–885. [Online]. Available: https://www.usenix.org/system/files/sec19-dong.pdf

[72] Google Inc. VirusTotal. [Online]. Available: https://www.virustotal.com/

[73] R. Yu, F. Nin, Y. Zhang, S. Huang, P. Kaliyar, S. Zakto, M. Conti, G. Portokalidis, and J. Xu, "Building Embedded Systems Like It's 1996," in *Proceedings of the Network and Distributed System Security Symposium (NDSS'24)*. San Diego, California, USA: The Internet Society, 2022. [Online]. Available: https://www.ndss-symposium.org/wp-content/uploads/2022-31-paper.pdf

[74] B. Davis. checksec.sh. [Online]. Available: https://github.com/slimm609/checksec.sh

[75] J. vom Dorp and R. Helmke, "Home Router Security Report 2022," Fraunhofer FKIE, Tech. Rep., 2022.

[76] R. Helmke, "Linux Firmware Corpus - Repository Snapshot," Jul. 2024. [Online]. Available: https://doi.org/10.5281/zenodo.13627731

## APPENDIX A
## FIRMWARE TAXONOMY

Embedded devices are diverse: They operate in many environments, are highly specialized, and are bound to application-specific constraints or cost functions [6]. This leads to many combinations of hardware layouts, ISAs, software stacks, and peripherals. Thus, heterogeneity shapes the firmware corpora and analysis methods in research.

To better describe firmware and its heterogeneous property space in this paper, we adopt the taxonomy by Muench et al. [14]. They define four types on the axes of OS abstraction and specialization. This appendix describes them in detail:

**Type-0** *Conventional Desktop and Server Systems.* This type describes common desktop or server OSs, e.g., Windows, Linux, and Mac OS. They ship rich user environments and general purpose kernels that add many hardware abstractions and features, like scheduling or file systems. Type-0 has many functions and high modularity, but requires certain components and more resources than other types. It is not strictly firmware [14], but influences the other types. Dominant ISAs are x86 and aspiring ARM platforms.

**Type-I** *General Purpose Embedded Systems*. Type-I retrofits Type-0 for embedded use: Because manufacturers know device purpose and hardware, they can strip unused components and add proprietary code. This allows less powerful devices to host Type-0 applications. Routers, e.g., may use networking from Type-0 OSs, but do not need media codecs. Linux is a common OS in Type-I systems [11], [12]. Here, manufacturers couple stripped kernels with slim user space environments like BusyBox [6]. Other popular Type-I devices use Windows IoT, Android, or iOS. MIPS and ARM are common ISAs [11].

**Type-II** *Special Purpose Embedded Systems*. Type-II runs on single purpose systems common in, e.g., industrial, automotive, or healthcare settings. Respective devices are robotic arms, wallboxes, or ECGs. Type-II handles requirements where Type-I fails, e.g.: There is no memory management

unit or there is limited processing power. Hardware layouts or special components may be unsupported by Type-I. There could be no file system, or tasks may be too time-critical for general purpose scheduling. Kernel and user space still exist, but lines become blurry [14]. MIPS and ARM are present, but there are plenty alternatives. Example Type-II OSs are VxWorks, Zephyr OS, $\mu$Clinux, and FreeRTOS.

**Type-III** *Bare-metal Embedded Systems*. Type-III, or bare-metal, systems have few to no OS abstraction: The firmware is a monolithic executable that shows no separation between kernel and user space. Applications execute in a single loop and have direct access to hardware [14]. Being minimalistic and low level, Type-III shows highest flexibility and least requirements. Thus, it runs on controllers or other integrated circuits. It is also used in highly constrained Internet-of-Things (IoT) devices, sensors, and actuators. The code can be custom or proprietary, but may use OS-like libraries for protocol stacks, interrupt handlers, and memory management. Example libraries are Contiki-NG and Mbed OS.

## APPENDIX B
## CATALOG OF CRITERIA FOR THE LITERATURE REVIEW

This appendix provides a catalog of criteria that we applied to each paper to derive the literature review results in Fig. 5.

### A. General Information

- If papers reused existing corpora, we considered all information on corpus creation practices from the sources and their artifacts as well.
- We only considered data sets with real-world samples.
- When papers used multiple real-world firmware corpora, we considered all of them.
- If a measure did not fit a paper's research question, we explicitly marked the data point as not applicable (⊕). Example: An HIL-based method like IoTFuzzer [41] does not need sample unpacking. Thus, neither unpacked sample quantities, nor unpacking procedures can be documented.

### B. Fulfillment Criteria for all 16 Measures

► **Packed & Unpacked Samples, Manufacturers, Models, Device Classes, and ISAs.** Let ***M*** be one of the measures above. ***M*** is ...

**([0-9]+ / ●) fulfilled**, if the paper states concrete quantities of ***M*** for *all* samples ...
1) ... in any kind of text (main, appendices, footnotes)
2) ... [OR] in figures or tables,
3) ... [OR] in shared meta data/linked repositories.
4) ... [OR] through complete sample lists in 1-3 that can be used to independently calculate the quantities.

**(◐) partially fulfilled**, if the paper desribes ***M*** ...
1) ... *only for a subset* of samples as stated in ●,
2) ... [OR] *imprecisely*, e.g., some round values [7].

**(◯) unfulfilled** in all other cases.

► **Deduplication, Unpacking, and Acquisition.** Let ***M*** be one of the measures above. ***M*** is ...

**(●) fulfilled**, if the paper explicitly documents ***M*** ...
1) ... in any kind of text (main, appendices, footnotes)
2) ... [OR] in figures or tables,
3) ... [OR] in shared meta data/linked repositories.
4) ... [OR] through a complete list of sample data or scripts to independently verify or replicate ***M***.

**(◐) partially fulfilled**, if the paper documents ***M*** ...
1) ... *only for a subset* of samples as stated in ●,
2) ... [OR] *incomplete*, such that independent research must assume substeps for verification or replication.

**(◯) unfulfilled** in all other cases.

► **Rel. Dates, Versions, Links, Hashes, and FW Types.** Let ***M*** be one of the file properties above. ***M*** is ...

**(●) fulfilled**, if the paper provides ***M*** for *all* samples ...
1) ... in any kind of text (main, appendices, footnotes)
2) ... [OR] in references,
3) ... [OR] in figures or tables,
4) ... [OR] in shared meta data/linked repositories.
5) ... [OR] (Links and Hashes) through shared samples.

**(◐) partially fulfilled**, if the paper fulfills ***M*** *only for a subset* of samples as stated in ●.

**(◯) unfulfilled** in all other cases.

► **Sample Selection Reasoning and Ground Truth**

Let ***M*** be one of the above. ***M*** is ...

**(●) fulfilled**, if researchers provide reasons for sample selection or consciously include bug ground truth before evaluation. Reasons could be arbitrary and must not seem reasonable to everyone. However, they are explicitly discussed and, thus, documented and disputable. Examples: The method to evaluate only supports certain ISAs, manufacturers, or OSs.

**(◐) partially fulfilled**, if there is no explicit selection reasoning, but possible selection motives are derivable from the paper. E.g., a paper rediscovers already known bugs in samples. A lack of documentation leaves the reader guessing if the researchers deliberately included samples with ground truth or not.

**(◯) unfulfilled** in all other cases.

## APPENDIX C
## LFWC: INSIGHTS ON CORPUS META DATA

Fig. 10 shows device classes per manufacturer. With 5,328 (49%) samples, routers are most prevalent in LFwC, followed by switches (1,525/14%) and access points (1,345/12%). The remaining classes have a prevalence between 162 (powerline) and 851 (repeater). For illustration, we bundled the other 14 classes, with samples from 3 to 116, into *misc* (569, or 5%).

Fig. 11 shows the detected Linux kernel banners, grouped by major and minor version. We removed manufacturer colors

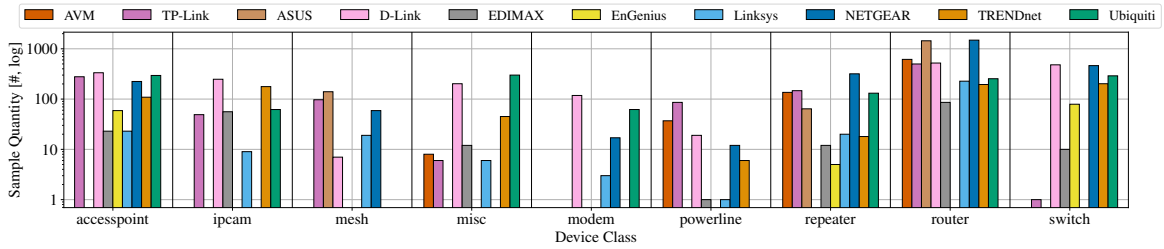


Fig. 10. Distribution of device classes in LFwC. The three most prevalent classes are routers (49%), switches (14%), and access points (12%). We bundled device classes with less than 150 samples into the meta class *misc*. It contains: controller, board, converter, encoder, gateway, kvm, media, nas, phone, power_supply, printer, recorder, san, and wifi-usb.

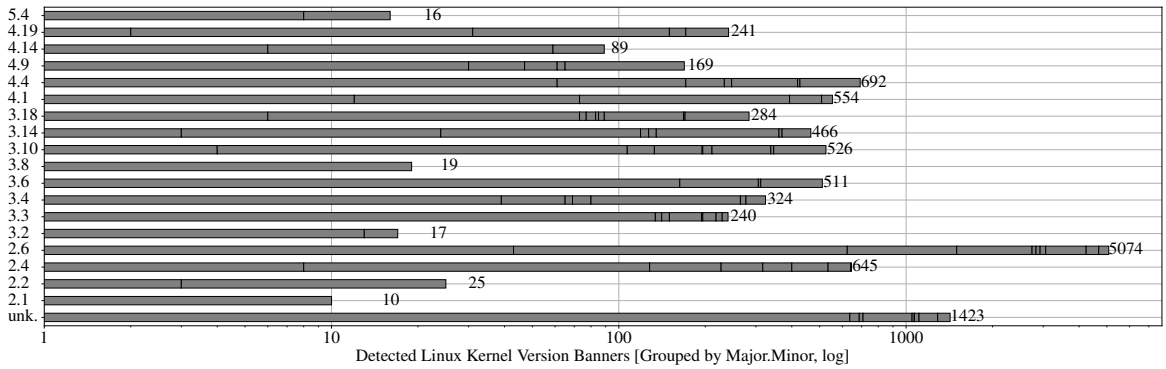


Fig. 11. Detected Linux kernel banners in LFwC samples. Versions range from 2.1 to 5.4, with Linux kernel 2.6 remaining the most prevalent version in our data set (51%). In total, we found 9,901 kernel banners across all included images. We could not find a banner in 1,423 samples. Note that a sample can ship with multiple kernels hosted by subsystems of independent device components.

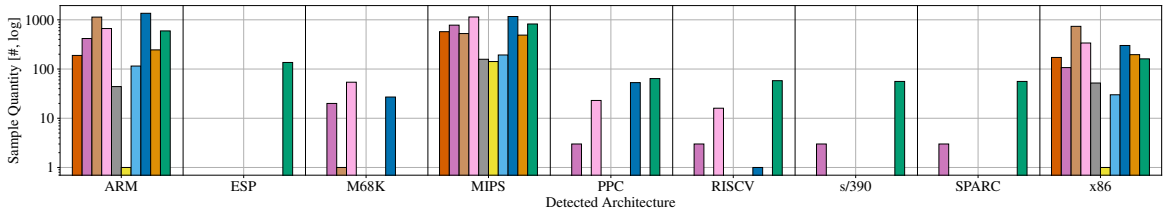


Fig. 12. Distribution of the nine detected ISAs in LFwC across all vendors on a logarithmic scale. The three most prevalent ISA families are MIPS (5,993 samples), ARM (4,764), and x86 (2,095). There are 13,429 unique findings on ISAs across all samples, because included subsystems must not run the same ISA as the main system.

as the scale introduced false visual cues of over- and under-represented shares. Versions range from 2.1 to 5.4, with 2.6 being most prevalent (5,074, or 51%). With 692 findings (7%), version 4.4 is the second most prevalent, followed by 2.4 with 645 findings. With 16 matches for version 5.4 and 10 matches for version 2.1, both extremes are rare in LFwC. In total, we found 9,901 banners in LFwC. The above numbers are already sanitized, as FACT found roughly 400 false positives originating from a pptp-linux client binary with similar banner format. Also we found multiple banners in some samples. This is because images can contain multiple subsystems and, thus, kernels. Cable modems embedded in routers, e.g., can run independent kernels interfacing with the main system. This data is present because the updates for sub components are managed by the main system. In general, we found that the existence of these subsystems in firmware images is often overlooked in related work on large scale analyses.

Finally, Fig. 12 shows the nine detected ISAs. There are 13,429 unique ISA findings hinting at the subsystems embedded into devices. In 5,993 images, we found MIPS, making it the most common ISA in LFwC. ARM was found in 4,764 images. The third group is x86 with findings in 2,095 images – most of them in powerful devices like managed switches and enterprise routers. With findings in 59 to 143 samples, the remaining ISAs, usually found in micro-controller sub-components, are underrepresented.

TABLE VI
CONFIG. 1: ORIGINAL SERVER SPECIFICATIONS (FULL CORPUS)

| Component | Specifications |
|---|---|
| CPU | 2x Intel Xeon E5-2650 v3@ 2.30 GHz, NUMA |
| RAM | 157 GiB DDR4 @ 2133MHz |
| Board | Dell 0HFG24, LGA 2011 (PowerEdge R430) |
| SSD (OS) | 512 GiB |
| HDD (Data) | 4 TiB |
| Internet Speed | 1 Gbps |
| OS | Ubuntu 22.04.4 LTS (Bare Metal) |
| Python | 3.10.12 |
| FACT | v4.2-dev (commit 0984d0ca) |

TABLE VII
CONFIG. 2: DOWN-SCALED SPECIFICATIONS (CORPUS SUBSET)

| Component | Specifications |
|---|---|
| CPU | VT-x/AMD-V Capable, 4 Cores for VM |
| RAM | 16 GiB for VM |
| Storage | 100 GiB |
| Internet Speed | High-Speed, Unmetered |
| OS | Arbitrary Modern Linux |
| VirtualBox | $\geq$7.0 |
| Vagrant | ~2.4 |

# APPENDIX D
# ARTIFACT APPENDIX

## A. Description & Requirements

**[Access]** The artifacts are available on GitHub[1]. We persist a git tag (`ndss-25-ae`) that marks the version we submitted for artifact evaluation. A ZIP-archive is on Zenodo [76]. As we stated in our ethical discussion on corpus distribution, the detail meta data required for LFwC replication is gated (cf. Section V-E). Researchers can request access to the corpus [16].

**[Hardware Requirements]** LFwC requires 354 GiB for samples, and 2.5 TiB for unpacking and content analysis. Unpacking and analysis take several months on server-grade hardware. Table VI shows our server specifications (Config. 1). As the other artifacts run on modern x86-64 desktop machines, we decided to add a second, down-scaled version of LFwC with fewer samples. Our artifacts include a virtual machine for quick deployment. As shown in Table VII, it requires 4 cores, 16 GiB RAM, and 100 GiB of storage (Config. 2).

**[Software Requirements]** For Config. 1, an Ubuntu 22.04 LTS system, with at least Python 3.10, is required. Also, users must install FACT commit `0984d0ca`. For a step-by-step guide, we refer to the official pages[2]. Config. 2, as shown in Table VII, is flexible and can be quickly deployed: Users take a recent Linux version, and then install VirtualBox[3] $\geq$7.0 and Vagrant[4] 2.4. The latter will be used to auto-deploy the virtual machine. We provide installers for all remaining dependencies.

## B. Setup Guide

We assume that the previous requirements are already fulfilled. The user starts with a shell – either on the server (Config. 1), or local (Config. 2):

First, users should obtain a copy of the repository via git:

```
$ git clone \
    https://github.com/fkie-cad/linux-firmware-corpus.git \
    ~/linux-firmware-corpus
$ cd ~/linux-firmware-corpus
$ # optional: change to ndss-25-ae branch
$ git checkout ndss-25-ae
```

**Config. 1** users install via `prepare` script and are finished:

```
$ cd ~/linux-firmware-corpus && ./prepare
```

**Config. 2** users skip installer invokation. The following command downloads a FACT image, deploys it in VirtualBox, mounts the repository, and invokes `prepare`:

```
$ cd ~/linux-firmware-corpus && vagrant up
```

Deployment takes 10 minutes, depending on download speed. Once finished, users can browse to http://localhost:5000 and are greeted by FACT. The machine is controlled as follows:

```
$ vagrant up # start the machine
$ vagrant halt # stop the machine
$ vagrant destroy # remove the machine + included data
```

The setup is now finished for **Config. 2**. Users can enter the virtual machine via the following command:

```
$ vagrant ssh # establish ssh connection with the machine
```

To change virtual machine resources, edit the `Vagrantfile`.

All steps described in the following subsections must be executed on the virtual machine, or dedicated server. However, the browser can be opened locally for Config. 2, as ports are forwarded to the host by VirtualBox.

## C. Artifact Claims

1) **Literature Analysis:** The literature analysis in Section IV is explorable. All statements from Section IV-B, IV-C, and IV-D are verifiable.
2) **Corpus Replication:** LFwC is replicable (cf. Section V). The included meta data and tools let researchers acquire 99% of all samples (as of June 2024, cf. Section V-C).
3) **Corpus Analysis:** All figures from Section V and Appendix C can be recreated. Corpus meta data is explorable and researchers can create sub-corpora.

## D. Literature Analysis Data Set

**[Preparation]** Go to `notebooks`. Run `./jupyter`. It asks to open a URL with format http://localhost:8888/lab?token=*. Open it in a browser to enter Jupyter Lab[5].

[1] https://github.com/fkie-cad/linux-firmware-corpus
[2] https://github.com/fkie-cad/FACT_core/blob/master/INSTALL.md
[3] https://www.virtualbox.org/
[4] https://www.vagrantup.com/
[5] https://jupyter.org/

**[Execution][1 human-minute + 1 compute-minute]** In Jupyter Lab, navigate to the `notebooks` sub-folder and open `IV_literature_review.ipynb`. Run all tiles by clicking on the ►► button below the opened notebook tab.

**[Expected Results][15 human-minutes]** The code loads the paper analysis data. Scrolling down, all tables and figures from Section IV appear. Each statement from Section IV-B, IV-C, and IV-D is quoted. There is code that generates evidence for statement verification.

**[Interactive]** Researchers are invited to interactively explore the data using Jupyter Lab and pandas[6] queries. To (re-)run a specific cell, it must be selected. Then, click ► in the top bar below the tab view.

### *E. LFwC: Corpus Replication & Unpacking (Config. 1)*

**[Preparation]** Get LFwC [16]. Navigate to `replication` and place `lfwc-full.csv` inside the folder.

**[Execution][15 human-minutes + 4 compute-months]**

1) Download 354 GiB of LFwC samples into `corpus`:

```
$ ./replicate-lfwc --corpus-csv lfwc-full.csv \
      download --corpus-dir corpus
```

2) Verify sample downloads:

```
$ ./replicate-lfwc --corpus-csv lfwc-full.csv verify \
      --corpus-dir corpus --json >> verify.json
```

3) Inspect `verify.json` for missing/corrupt samples.
4) If samples are missing, use archive.org:

```
$ ./replicate-lfwc --corpus-csv lfwc-full.csv \
      download --corpus-dir corpus \
      --continue --use-wayback-machine
```

5) If samples are missing, use product and file data to manually search for sources on the Internet (cf. Section V-C).
6) Upload the samples to FACT:

```
$ ./replicate-lfwc --corpus-csv lfwc-full.csv \
      upload-to-fact --corpus-dir corpus
```

**[Expected Results]** After the download step, the `corpus` folder should contain the samples. After (and during) upload, the firmware images should appear in FACT, running on http://localhost:5000. Each sample is now unpacked and analyzed.

**[Interactive][30 human-minutes]** LFwC can be interactively explored, either via FACT's web interface or its REST API. We invite researchers to watch a talk[7] on its capabilities, which teaches tool-specific knowledge.

### *F. LFwC: Corpus Replication & Unpacking (Config. 2)*

For Config. 2, researchers can follow all steps from Appendix D-E. To reduce compute times and hardware requirements, we provide a tool to create sampled sub-corpora of LFwC. The following steps describe how to use the script. After corpus creation, please follow Appendix D-E and substitute `lfwc-full.csv` with the newly created mini corpus.

**[Preparation]** Get LFwC [16]. Go to `downscaling`. Put `lfwc-full.csv` inside the folder.

**[Execution][1 human-minute + 1 compute-minute]** Create a random corpus of 50 samples, with 5 samples per manufacturer and a maximum sample size of 30 MiB:

```
./build_corpus --full_corpus lfwc-full.csv \
               --output lfwc-mini.csv
```

If desired, invoke with `--help` to change creation parameters.

**[Expected Results]** A mini corpus in `lfwc-mini.csv`. This can be used with the instructions in Appendix D-E.

### *G. LFwC: Corpus Meta Data Analysis*

**[Preparation]** Get LFwC [16]. Navigate to `notebooks`. Place `lfwc-full.csv` in the `public_data` subfolder. Run `./jupyter`. It asks to open a URL with format http://localhost:8888/lab?token=*. Open it in a browser to enter Jupyter Lab.

**[Execution][1 human-minute + 3 compute-minutes]** In Jupyter Lab, navigate to the `notebooks` sub-folder and open `V_lfwc.ipynb`. Run all tiles by clicking on the ►► button below the opened notebook tab.

**[Expected Results]** The code loads LFwC's meta data file. Scrolling down, all tables and figures from Section V and Appendix C are created.

**[Interactive]** Researchers are invited to interactively explore LFwC's meta data using Jupyter Lab and pandas queries. Using this function, users can independently query the data and create subsets. Depending on their own research questions, they could, e.g., derive corpora that only include firmware with specific device types, Linux kernel versions, or architectures. At the bottom of the notebook, there is an example to show how to query the data. To (re-)run a specific cell, it must be selected. Then, click ► in the top bar below the tab view.

### *H. (Bonus) LFwC: Historical Scrapers*

We share our scrapers, but most of them likely fail, as Internet resources disappear and web layouts change over time.

**[Preparation]** Navigate to `scrapers`.

**[Execution][1 human-minute + var. download-hours]** Execute: `./scrape <manufacturer>`. Valid values are: archive_avm, archive_linksys, asus, avm, dlink, edimax, engenius, linksys, netgear, tplink, trendnet, ubiquiti. On 2024-06-30, we could verify that the D-Link scraper is still functional.

**[Expected Results]** Scraped samples should appear in `firmware_files`. The collected meta data is written into a JSON file that has the same name as the scraped manufacturer.

[6] https://pandas.pydata.org/

[7] https://passthesalt.ubicast.tv/videos/improving-your-firmware-security-analysis-process-with-fact/