Published: 18 November 2025

IOT 2025: The 15th International Conference on the Internet of Things
November 18 - 21, 2025
Vienna, Austria

RESEARCH-ARTICLE

UniBOM – A Unified SBOM Analysis and Visualisation Tool for IoT Systems and Beyond

VADIM SAFRONOV, University of Oxford, Oxford, Oxfordshire, U.K.

IONUT BOSTAN

NICHOLAS ALLOTT

ANDREW MARTIN, University of Oxford, Oxford, Oxfordshire, U.K.

IOT '25: Proceedings of the 15th International Conference on the Internet of Things (November 2025)
https://doi.org/10.1145/3770501.3770512
ISBN: 9798400715952

UniBOM – A Unified SBOM Analysis and Visualisation Tool for IoT Systems and Beyond

Vadim Safronov*
University of Oxford
Oxford, United Kingdom
vadim.safronov@cs.ox.ac.uk

Ionut Bostan*
NquiringMinds
Southampton, United Kingdom
ionut@nquiringminds.com

Nicholas Allott
NquiringMinds
Southampton, United Kingdom
nick@nquiringminds.com

Andrew Martin
University of Oxford
Oxford, United Kingdom
andrew.martin@cs.ox.ac.uk

Abstract

Modern networked systems rely on complex software stacks, which often conceal vulnerabilities arising from intricate interdependencies. A Software Bill of Materials (SBOM) is effective for identifying dependencies and mitigating security risks. However, existing SBOM solutions lack precision, particularly in binary analysis and non-package-managed languages like C/C++.

This paper introduces UniBOM, an advanced tool for SBOM generation, analysis, and visualisation, designed to enhance the security accountability of networked systems. UniBOM integrates binary, filesystem, and source code analysis, enabling fine-grained vulnerability detection and risk management. Key features include historical CPE tracking, AI-based vulnerability classification by severity and memory safety, and support for non-package-managed C/C++ dependencies.

UniBOM's effectiveness is demonstrated through a comparative vulnerability analysis of 258 wireless router firmware binaries and the source code of four popular IoT operating systems, highlighting its superior detection capabilities compared to other widely used SBOM generation and analysis tools. Packaged for open-source distribution, UniBOM offers an end-to-end unified analysis and visualisation solution, advancing SBOM-driven security management for dependable networked systems and broader software.

CCS Concepts

• Security and privacy → Systems security; **• Software and its engineering → Software notations and tools**; **• Networks → Wireless access points, base stations and infrastructure**.

Keywords

Security, SBOM, Accountability, Internet of Things, Software tools, Firmware analysis

ACM Reference Format:
Vadim Safronov, Ionut Bostan, Nicholas Allott, and Andrew Martin. 2025. UniBOM – A Unified SBOM Analysis and Visualisation Tool for IoT Systems and Beyond. In *The 15th International Conference on the Internet of Things (IOT 2025), November 18–21, 2025, Vienna, Austria.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3770501.3770512

1 Introduction

As IoT technologies become increasingly pervasive across diverse safety-critical applications — ranging from smart cities to industrial IoT and healthcare — dependable networked systems supporting such infrastructures must navigate increasingly complex software stacks with numerous interdependent components. Ensuring the dependability and security of these systems is a critical challenge, particularly given the distributed nature of IoT applications and the diversity of technological standards they encompass. The high heterogeneity of IoT devices — spanning variations in types, protocols, manufacturers, update cycles, and software versions — significantly complicates the task of guaranteeing consistent, high-level protection for IoT applications operating across such diverse environments. These complexities often expose hidden vulnerabilities within interconnected software layers, compromising system dependability, reliability, and security.

Memory-related vulnerabilities in IoT systems account for over 70% of known security threats [37], a figure consistent with reports from CISA [11] and leading technology organisations [19, 31, 35]. When exploited on IoT safety-critical ecosystems, a single compromised router can become the entry point for attacking IoT devices, which subsequently compromises entire networks, escalating to widespread cyber threats. Notable incidents highlight the urgency of this issue: the Dyn DNS attack [21] and Mirai botnets [6], where a multitude of IoT devices were used to launch a massive Distributed Denial of Service (DDoS) attacks, the SolarWinds breach [27], and Log4j incidents [40] all exemplify the wide systemic consequences of vulnerabilities (however they arise).

Employing Software Bill of Materials (SBOM) advances our ability to develop dependable IoT ecosystems which are both secure and resilient by design. SBOM provides a standardised framework for enumerating software dependencies, enabling tracking of potential vulnerabilities within the analysed software and its subcomponents. Where SBOM concepts are supported by practical tools [32, 33], existing and novel threats can, in principle, be the subject of rapid

---

*Equal contribution.

IOT 2025, Vienna, Austria

ACM ISBN 979-8-4007-1595-2/25/11
https://doi.org/10.1145/3770501.3770512

response. For instance, deploying a robust SBOM-based tool capable of rapidly mapping software dependencies within safety-critical IoT systems affected by a new unknown threat and enabling agile and comprehensive recovery can protect these systems and their interconnected neighbours from the propagation and exploitation of that threat on a global scale.

Despite the availability of various SBOM generation tools [2, 5, 23, 29], existing solutions provide limited support for the comprehensive analysis of dependable IoT networked systems where a significant portion of software is written in C/C++, operating without standardised dependency management tools such as Conan [1], or relying on frequent binary updates. These limitations arise from the reliance of most SBOM tools on metadata-based approaches, which lack the capability to extract and analyse binary images and C/C++ source code [49]. While individual tools dedicated to either binary analysis or source code dependency analysis for C/C++ are available [26, 41], there is no unified end-to-end solution that currently integrates these approaches for generating a fine-grained SBOM for dependable networked systems. Additionally, existing tools often lack advanced visualisation and classification capabilities, such as historical threat analysis, prioritisation of severe vulnerabilities, and comprehensive threat categorisation.

To address these limitations and provide a robust solution for researchers and industry professionals managing the security of modern IoT systems and dependable systems more broadly, we propose UniBOM — an open source tool for SBOM generation, analysis, and visualisation[1][2]. UniBOM enhances visibility and accountability in dependable system software security by offering the following key functionalities:

- **SBOM generation for binary images:** Supports firmware binary analysis using the built-in Binwalk utility [26] to extract firmware filesystems, followed by SBOM generation with Syft [5] and CVE/CWE analysis with Grype [4].
- **Tracking non-package-managed C/C++ dependencies:** Facilitates fine-grained SBOM generation by supporting non-conan C/C++ source code analysis, leveraging integration with the CCScanner tool [41].
- **AI-based vulnerability classification:** Addressing the prevalence of memory-safety threats in memory-unsafe languages like C/C++, the tool categorises identified vulnerabilities into memory-related categories and CVSS severity levels, prioritising high-risk vulnerabilities for more fine-grained and effective threat management.
- **Web GUI for visualising advanced security analyses:** A web application that enables users to visualise SBOM analysis results generated by UniBOM or other SBOM tools. It highlights and categorises vulnerabilities based on severity and relevance to memory safety, offering interactive dashboards for vulnerability distribution and historical trend analysis of software components and the emergence of vulnerabilities over time.

This paper demonstrates the primary functionalities of UniBOM and is organised as follows: Section 2 provides a concise overview of SBOM concepts. Section 3 discusses related work on SBOM generation tools and methodologies, providing context for UniBOM's contributions. Section 4 details the tool's design and highlights its main functionalities. Section 5 showcases UniBOM's capabilities in SBOM generation, emphasising improved vulnerability detection through more precise SBOM creation. This is demonstrated through a case study involving 258 wireless gateway firmware binaries and source code analysis of four widely used IoT operating systems. Section 6 introduces the web application, which presents UniBOM analysis results via interactive dashboards, trend analysis, and risk visualisation, categorising vulnerabilities by severity and memory-safety relevance for comprehensive assessment. Section 7 concludes the paper, summarising key insights and future directions.

2 Software Bill of Materials

The Software Bill of Materials (SBOM), inspired by manufacturing industry practices [47][24], was formalised in 2018 by the National Telecommunications and Information Administration (NTIA) to enhance software security practices, and has since been refined and expanded [7]. An SBOM provides a detailed inventory of the components comprising a software application, specifying their origins, dependencies, and associated references to known or potential vulnerabilities.

Key foundational elements within the SBOM ecosystem include the Common Platform Enumeration (CPE), Common Vulnerabilities and Exposures (CVE), and Common Weakness Enumeration (CWE). Figure 1 illustrates the connections between these components, facilitating the identification, classification, and mitigation of security flaws and vulnerabilities. An SBOM, essentially a list of software components (CPEs), ensures that all elements are identified and traced to their origins.

CPE is a standardised naming schema maintained by the U.S. National Institute of Standards and Technology (NIST) [9], designed to identify software versions and packages. The current version, CPE 2.3, assigns structured names to software products, capturing attributes such as part, vendor, product, and version[30]. For example, the GNOME GLib library version 2.66.4 is encoded as `cpe:2.3:a:gnome:glib:2.66.4:::::::*`, where "cpe:2.3" indicates the CPE version (2.3), "a" specifies the application component, "gnome" identifies the vendor, "glib" is the product, and "2.66.4" denotes the version.

CVE provides a unified system to identify, assess, and catalogue vulnerabilities. Hosted on the National Vulnerability Database (NVD) [9], it assigns unique IDs to facilitate vulnerability tracking, discussion, and resolution [10]. For instance, CVE-2021-42376, a NULL Dereference vulnerability, relates to the product identified as `cpe:2.3:a:busybox:busybox:1.33.2`. A CVE typically categorises security vulnerabilities in software through key components. The CVE ID ("CVE-YYYY-NNNNN") includes "YYYY," representing the year of discovery, and "NNNNN," a unique identifier for the vulnerability. The description provides details about the vulnerability and its potential impact on affected systems. Weaknesses identify the type of security issue (CWE). The base score quantifies the severity of the vulnerability, while the base severity provides a qualitative rating as "Low," "Medium," "High," or "Critical".

---

[1] UniBOM CLI Tool: https://github.com/nqminds/UniBOM
[2] UniBOM Web GUI: https://github.com/nqminds/UniBOM-Viewer

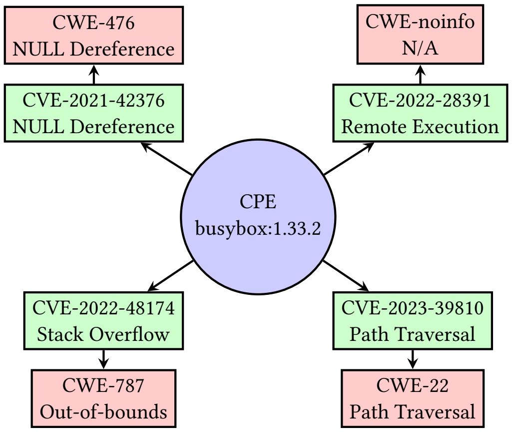


Figure 1: Radial representation of CPE, CVE, and CWE relationships for Busybox 1.33.2.

CWE is a comprehensive glossary of common security weaknesses in software and hardware, with updates made several times a year [8]. Each CWE may encompass multiple CVE instances, as it categorises a specific type of weakness that can lead to various vulnerabilities. For example, CWE-787, Out-of-Bounds Write, has been the root cause of numerous vulnerabilities, such as the stack overflow vulnerability in BusyBox version 1.33.2 (CVE-2021-42373) and a memory corruption issue in OpenSSL (CVE-2022-0778).

Overall, CPEs identify software elements across systems, CVEs link these elements to known vulnerabilities, and CWEs provide insight into potential weak points within internal components.

3 Related Work

SBOM generation and analysis methods can be broadly categorised into binary analysis, metadata-based analysis, and source code inspection.

Binary-focused SBOM generation tools [13, 14, 20, 26] perform software composition analysis on compiled binaries, leveraging string literals, embedded metadata, and language-specific features to identify software components and their dependencies.

Metadata-based SBOM generation tools [2, 5, 12, 29] analyse software package metadata and dependency information. These tools support diverse ecosystems by parsing metadata from container images, build files, and package managers to map dependencies effectively.

Source code analysis tools [22, 42, 45] generate SBOMs by examining code repositories to identify components, dependencies, and their interrelationships. Compared to binary or metadata-focused approaches, they can provide more detailed SBOM insights by uncovering hidden interdependencies, including those potentially associated with present CVEs.

UniBOM distinguishes itself by integrating all three methods: binary extraction via Binwalk [26], filesystem and metadata analysis through Syft [5], and source code analysis using CCScanner [42]. Additionally, UniBOM's web-based GUI visualises SBOM analysis, highlighting vulnerabilities by severity and memory safety relevance, with interactive dashboards for distribution and historical trends of components and vulnerabilities.

While a few visualisation tools for SBOM analysis are available [15, 43], their analytical capabilities are limited in scope and do not include features such as historical vulnerability analysis or vulnerability categorisation based on memory-safety relevance, which UniBOM aims to provide as part of a more comprehensive approach.

4 UniBOM Architecture

UniBOM is a Node.js-based command-line interface (CLI) tool designed for comprehensive SBOM generation and vulnerability analysis across a wide range of input types, including Docker images, source code, root filesystems, C/C++ codebases, and firmware binaries. Its architecture integrates various tools for filesystem extraction, code and filesystem scanning, SBOM generation and analysis, as well as advanced CVE analysis and categorisation, including Binwalk [26], Syft [5], Grype [4], and CCScanner [42], all packaged in Docker containers for simplified installation and usability. By combining these capabilities into an end-to-end architectural pipeline (Figure 2), UniBOM offers a professional-grade solution for SBOM generation and vulnerability analysis, supporting both research and industrial security applications and effectively addressing complex and diverse input types with precision. The UniBOM pipeline operates through the following steps:

- **Step 0: Input Format Identification.** The pipeline begins by recognising the input format, which can range from container images to binary firmware or source code.
- **Step 1: Tool Selection and SBOM generation.** Based on the input format and desired SBOM granularity, UniBOM selects the appropriate processing tool for SBOM generation, leveraging Syft for general-purpose SBOM and CCScanner for C/C++ codebases, addressing limitations in recognising third-party dependencies without a package manager.
- **Step 2: CVE/CWE Extraction.** For each dependency in the generated SBOM, UniBOM performs vulnerability detection by mapping identified components and dependencies to the CPE v2.3 format and, through requesting vulnerability databases, extracting of associated CVEs and their general CWEs, for overall report and further advanced security analysis.
- **Step 3: Advanced Security Analysis.** Beyond SBOM generation and CVE/CWE detection, UniBOM provides advanced capabilities, including:
  - Historical tracing of vulnerabilities for identified CPEs.
  - Memory safety assessments for detected CVEs and CWEs, offering deeper insights into potential security impacts.
  - Cross-version SBOM comparisons to track changes and identify evolving risks.

4.1 Usage Overview

4.1.1 Binary Extraction. The CLI tool incorporates Binwalk, a specialised utility for reverse engineering and extracting data from firmware binary images.

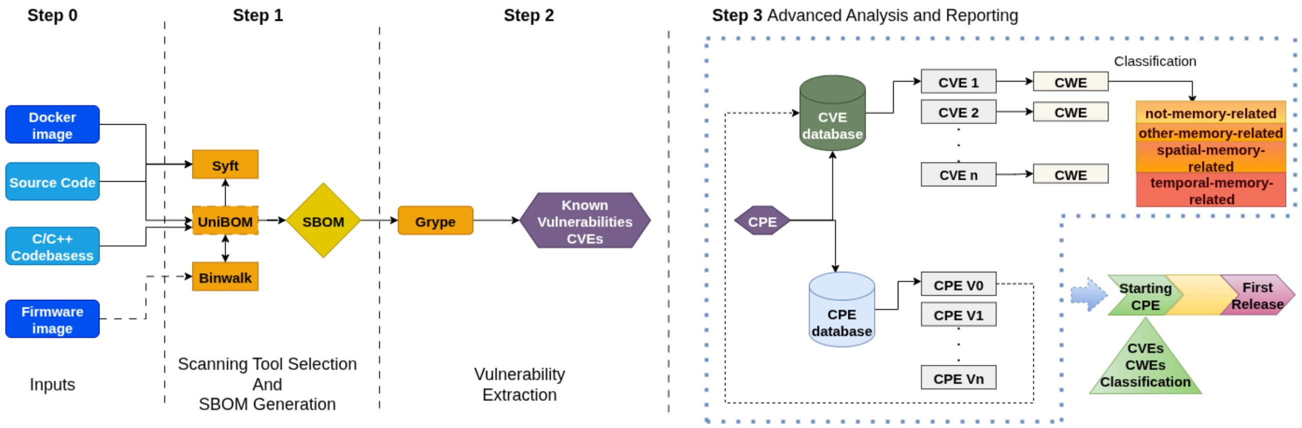


Figure 2: UniBOM architectural pipeline.

`unibom -binwalk "$(pwd)" "[-Me]" image.bin`. Binwalk efficiently identifies and unpacks embedded file systems, libraries, executables, and other components, including recognising compression and encryption methods to uncover hidden or obfuscated data. It also extracts metadata, such as OS versions and build environments, facilitating detailed dependency mapping and source code analysis. These capabilities enable the generation of fine-grained SBOMs and advanced security assessments, such as identifying firmware-specific vulnerabilities and CWEs.

4.1.2 SBOM Generation. SBOM generation in UniBOM is supported through two distinct methods, tailored to the characteristics of the input data:

`unibom -generateSbom <project_path> <project_name>` The integrated Anchore/Syft generator creates an SBOM by analysing extracted filesystems and source code, which can be retrieved directly from a repository or extracted using Binwalk. Syft identifies dependencies managed by supported package managers, such as package.json for Node.js or Conan for C/C++. A detailed list of supported package managers and programming languages is available in Syft's official documentation [5]. To assess vulnerabilities, the SBOM is analysed using Grype, which produces a software vulnerability report based on the generated SBOM.

`unibom -generateCCPPReport <path_to_c/cpp_project> <project_name>` This method addresses the limitations of existing tools in generating SBOMs for C/C++ projects lacking standard package management files, such as *.vcxproj or conan.lock. Leveraging insights from large-scale differential analyses of SBOM generators — including Syft, Trivy, GitHub, and sbom-tool — UniBOM integrates CCScanner, a research-driven tool designed to unify and standardise dependency analysis for C/C++ environments. CCScanner excels in parsing complex build systems, such as Makefiles, CMake, and Bazel, and extracting dependency information directly from source code, linking headers, and linked libraries. It maps these dependencies to standardised identifiers, such as Common Platform Enumeration (CPE) entries, and facilitates the inclusion of third-party libraries often overlooked due to disjointed package management in C/C++ projects. This integration enables UniBOM to produce more fine-grained SBOMs that account for both direct and transitive dependencies, enhancing the visibility of software components and improving vulnerability assessment in firmware with the majority of C/C++ sourcecode.

4.1.3 Extended Analysis and Comparison Features. UniBOM's advanced SBOM analysis features offer detailed insights into the history of CPE releases and their associated vulnerability analysis, categorising security weaknesses for enhanced clarity. These features also enable direct SBOM comparisons to track changes and progress over time, such as comparing neighbouring firmware updates. The following advanced analysis functionality is designed to provide UniBOM users with a comprehensive understanding of the security state of their firmware, enabling them to make well-informed decisions on further targeted improvements and their prioritisation.

`unibom -getHistory <CPE>` This parameter enables the historical analysis of CPE components. For a specified CPE, the tool retrieves and examines all prior versions of that CPE. For each version, it displays relevant information, such as associated CVEs and, for each CVE, its corresponding CWE. It further categorises each CWE based on memory-related classifications (e.g., spatial, temporal, or other memory types). The structure, as illustrated in Table 1, provides a detailed security taxonomy for every version, enabling users to identify the vulnerabilities present in their code and understand their impact on memory usage. This functionality is particularly valuable in environments coded in C/C++ languages, where memory management is a critical consideration. The ability to monitor such intricate interconnections over time is advantageous for proactive risk management. It allows developers and security teams to adaptively assess the risk levels of software components and prioritise mitigation efforts accordingly. This comprehensive perspective facilitates more informed decisions about securing dependencies, especially when supporting legacy versions or managing multiple library versions with varying levels of known vulnerabilities.

`unibom -classifyCwe <CWE-ID>` UniBOM leverages a GPT-based model to classify vulnerabilities into four categories: not memory-related, spatial memory-related, temporal memory-related, and other memory-related. This is achieved by sending a structured

Table 1: CPE historical analysis.

CPE	CVE	CWE	Memory class
…(multiple versions)			
cpe:2.3:a:openssl openssl:0.9.2b	CVE-2014-8176	CWE-119	spatial
…(multiple versions)			
cpe:2.3:a:openssl openssl:0.9.6d	CVE-2016-2106	CWE-189	spatial
…(multiple versions)			
cpe:2.3:a:openssl openssl:1.1.1	CVE-2021-3712	CWE-125	spatial
	CVE-2022-4450	CWE-415	temporal
	CVE-2021-3449	CWE-476	spatial

prompt with instructions for classifying the vulnerability based on its textual description. If the CWE text description is unavailable, the corresponding CVE description can be used for classification. Understanding and quantifying memory-related vulnerabilities is essential for making informed decisions, such as adopting hardware-based memory-protection solutions like the CHERI-based Morello architecture [18] or transitioning to a memory-safe programming language like Rust. The prevalence of memory vulnerabilities in IoT devices, as highlighted in a recent study on IoT firmware analysis [37], emphasises the importance of classifying these risks. IoT devices are particularly susceptible to a range of memory-safety threats, including spatial vulnerabilities such as buffer overflows and temporal vulnerabilities like use-after-free errors. Without quantifying and categorising these vulnerabilities, it is difficult to assess their full impact on system security. By identifying and classifying these risks, stakeholders can gain a clearer understanding of the scale and severity of memory-related issues within the analysed firmware, enabling more informed and effective decision-making.

`unibom -compare <sbom1> <sbom2>` The SBOM compare function provides an in-depth analysis of multiple SBOMs by comparing component versions and their associated CVEs. This feature provides fine-grained visibility into the security state across configurations and environments by isolating component version changes and mapping vulnerabilities to specific versions within each SBOM. It is particularly valuable for vulnerability management and compliance reporting, as it offers a clear understanding of changes in dependencies and their impact on overall security. Automating the SBOM comparison process reduces the manual effort required to identify systems or environments at greater risk, while also highlighting configuration-specific gaps or inconsistencies. This enables security teams to make prompt, informed decisions on prioritising remediation efforts.

Table 2 presents an example of the output generated from comparing components identified in two test SBOMs of the same firmware. The CPEs are categorised into "Component", "Version Information", and "Vulnerabilities", highlighting differences between the two SBOMs in terms of component presence, version consistency, and potential security risks.

- *Key Components*: indicates the presence or absence of specific components, such as OpenSSL, kernel, and SQLite, in each SBOM. For example, SBOM-1 contains OpenSSL, whereas SBOM-2 does not. Conversely, SQLite is unique to SBOM-2, while the kernel is present in both SBOMs.
- *Version Information*: details the components included in each SBOM along with their respective versions. For instance, SBOM-1 lists OpenSSL as version 3.0.0, but SBOM-2 does not include it. The kernel is version 2.24.2 in both SBOMs, while SBOM-2 lists SQLite as version 3.5.9.
- *Vulnerabilities*: lists the known vulnerabilities associated with each SBOM component. For example, OpenSSL in SBOM-1 has the vulnerability CVE-2009-1390, but OpenSSL is absent from SBOM-2, and therefore no vulnerabilities are listed for it. The kernel in SBOM-1 has CVE-2014-9114, while SBOM-2 contains CVE-2016-2779 for the same component. Additionally, SQLite vulnerabilities unique to SBOM-2 include CVE-2015-3414.

Table 2: SBOM comparison for key CPEs.

Component	SBOM-1	SBOM-2
Version Information		
openssl	3.0.0	none
kernel	2.24.2	2.24.2
sqlite	none	3.5.9
Vulnerabilities		
openssl	CVE-2009-1390, ...	none
kernel	CVE-2014-9114, ...	CVE-2016-2779, ...
sqlite	none	CVE-2015-3414, ...

5 Evaluation

In order to demonstrate UniBOM's effectiveness in handling the most commonly used input formats of IoT system firmware — binary images, filesystems, and source code — its functionality was compared against popular SBOM tools, including Microsoft's SBOM-tool (referred to hereafter as sbom-tool) [29], Trivy [2], Syft [5], and CCScanner [42]. Two experiments were conducted: (a) SBOM generation followed by vulnerability scanning of real IoT firmware binaries from a recent dataset used in prior CVE analyses of IoT firmware images [50], and (b) source code analysis of popular IoT operating systems.

5.1 IoT Binary Firmware Analysis

The evaluation involved 258 Linux-based firmware binaries from leading wireless gateway manufacturers: 54 from D-Link, 104 from OpenWrt, and 100 from TRENDnet.

To ensure a fair comparison with other tools, all binary images were extracted prior to analysis. The extraction was performed using Binwalk [26], and for each extracted firmware binary, SBOMs were generated using Syft, Trivy, and sbom-tool. These SBOMs were subsequently parsed with the Grype vulnerability scanner [4] to identify CVEs and analyse their severity occurrences.

UniBOM was the only tool that streamlined the entire process of SBOM generation and vulnerability analysis by integrating Binwalk, Syft, CCScanner, and Grype into a unified workflow, providing seamless binary analysis and vulnerability scanning.

Table 3: Vulnerability scanning results for IoT firmware binaries.

Tools	D-Link	OpenWrt	TRENDnet	Total	Total Severities
sbom-tool	0	0	0	0	-
Trivy	0	0	0	0	-
Syft	1885	1371	1326	4582	526 Crit 2300 High 1642 Med 114 Low
CCScanner	0	0	0	0	-
UniBOM	1885	1371	1326	4582	526 Crit 2300 High 1642 Med 114 Low

Table 3 shows that UniBOM and Syft were the only tools capable of successfully analysing the extracted filesystems, whereas other tools failed to identify the majority of dependencies within the extracted filesystem data. Both UniBOM and Syft detected a total of 4582 CVE occurrences, with over 60% classified as high or critical severity. This highlights the advantage of UniBOM's seamless integration with Binwalk, Syft, CCScanner, and Grype, where Binwalk handles binary extraction, Syft effectively analyses the resulting filesystems (with CCScanner for source code analysis, if required), and Grype performs further vulnerability assessment.

5.2 IoT OS Source Code Analysis

To demonstrate UniBOM's capabilities beyond binary analysis, its SBOM generation and analysis functionalities were tested on source code from widely used IoT operating systems, including the Raspberry Pi Linux Kernel (RPi) [17], Zephyr [36], Nuttx [16], OpenWrt 23.05.5 [44], and the latest version of OpenWrt [34]. The evaluation focused on assessing UniBOM's effectiveness in identifying vulnerabilities through source code analysis. Table 4 summarises the vulnerabilities detected in the IoT OS source code by the evaluated SBOM analysis tools.

UniBOM significantly outperformed all other standalone tools by integrating Syft's metadata extraction with CCScanner's advanced source code analysis techniques. This combination allowed UniBOM to detect a broader range of dependencies and uncover more vulnerabilities, identifying the highest number of CVE occurrences, with a total of 56. Compared to CCScanner, UniBOM identified 2 additional critical, 5 medium, and 2 high-severity CVEs. Against Syft, UniBOM detected 2 more critical, 15 high-severity, and 21 medium-severity CVEs. The performance gap was even more evident when compared to Trivy and SBOM-tool, which detected only 5 and 0 CVEs, respectively.

6 Web GUI Overview

The UniBOM GUI, powered by its SBOM generation and analysis engine as detailed in Section 4, is designed to provide convenience in analysis visualisation, enhancing user understanding of the security state and trends of analysed firmware. Users can upload SBOM files for in-depth analysis, with results presented through interactive dashboards and charts. Key features include listing all components within the SBOM file, identifying vulnerabilities associated with each component, and categorising these vulnerabilities based on severity and relevance to memory-safety issues, which are prevalent in IoT systems. To make multidimensional analysis data accessible, the UniBOM GUI employs dynamic bar charts to illustrate the distribution of vulnerabilities by severity, pie charts to classify vulnerabilities based on memory safety, and tables to offer detailed information on each vulnerability, including severity, affected systems, and links to external National Vulnerability Database (NVD) references. The GUI includes a historical analysis feature, offering insights into CVE appearance and mitigation trends across selected CPEs and their previous versions.

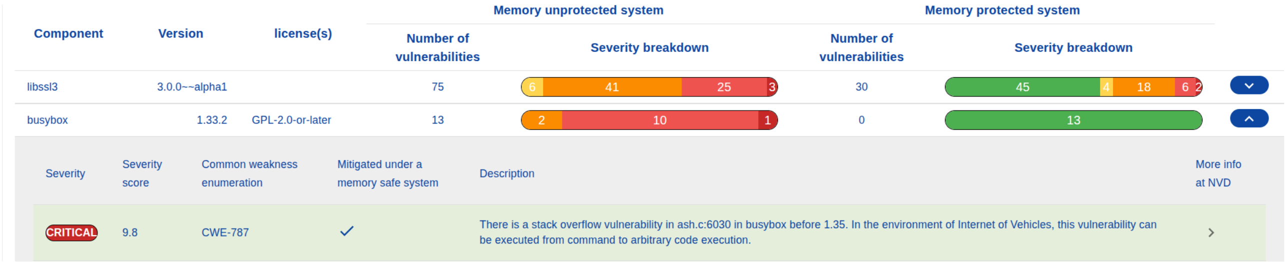
Table 4: Vulnerability scanning results for IoT OS source code.

Tools	RPi	Zephyr	Nuttx	OpenWrt 23.05	OpenWrt Latest	Total Sevs
sbom-tool	0	0	0	0	0	-
Trivy	0	5	0	0	0	2 Crit 2 Med 1 Low
Syft	0	5	0	0	4	2 Crit 2 High 5 Med
CCScanner	2	0	38	1	0	9 Crit 15 High 21 Med 1 Low 1 Unkn
UniBOM	2	5	38	1	4	11 Crit 17 High 26 Med 1 Low 1 Unkn

6.1 Visualising Firmware Vulnerability Status

When an SBOM file is uploaded, UniBOM enumerates all its CPEs and groups the associated CVEs by severity level (Low, Medium, High, or Critical) and memory-safety classification (memory-related or non-memory-related). These classifications are visualised through pie charts, as shown in Figure 3, providing users with an overview of the risk distribution within the analysed firmware.

Figure 3 also highlights memory classification and an analysis of the potential impact of transitioning selected firmware to secure-by-design memory protection solutions, such as CHERI [46], CHERIoT [3], Rust [28] and other potential memory-protection approaches discussed in recent works [25, 39, 48]. The screenshot in Figure 3 showcases an analysis of example firmware, demonstrating that adopting a memory protection solution could eliminate 42 medium, high, and critical CVE occurrences, significantly reducing the vulnerability surface. This feature is particularly critical for IoT systems, where memory-safety threats are notably widespread.



Component	Version	license(s)	Memory unprotected system: Number of vulnerabilities	Memory unprotected system: Severity breakdown	Memory protected system: Number of vulnerabilities	Memory protected system: Severity breakdown
libssl3	3.0.0~~alpha1		75	6 41 25 3	30	45 4 18 6 2
busybox	1.33.2	GPL-2.0-or-later	13	2 10 1	0	13

Severity	Severity score	Common weakness enumeration	Mitigated under a memory safe system	Description	More info at NVD
CRITICAL	9.8	CWE-787	✓	There is a stack overflow vulnerability in ash.c:6030 in busybox before 1.35. In the environment of Internet of Vehicles, this vulnerability can be executed from command to arbitrary code execution.	›

Figure 3: UniBOM GUI screenshot: firmware vulnerability status.

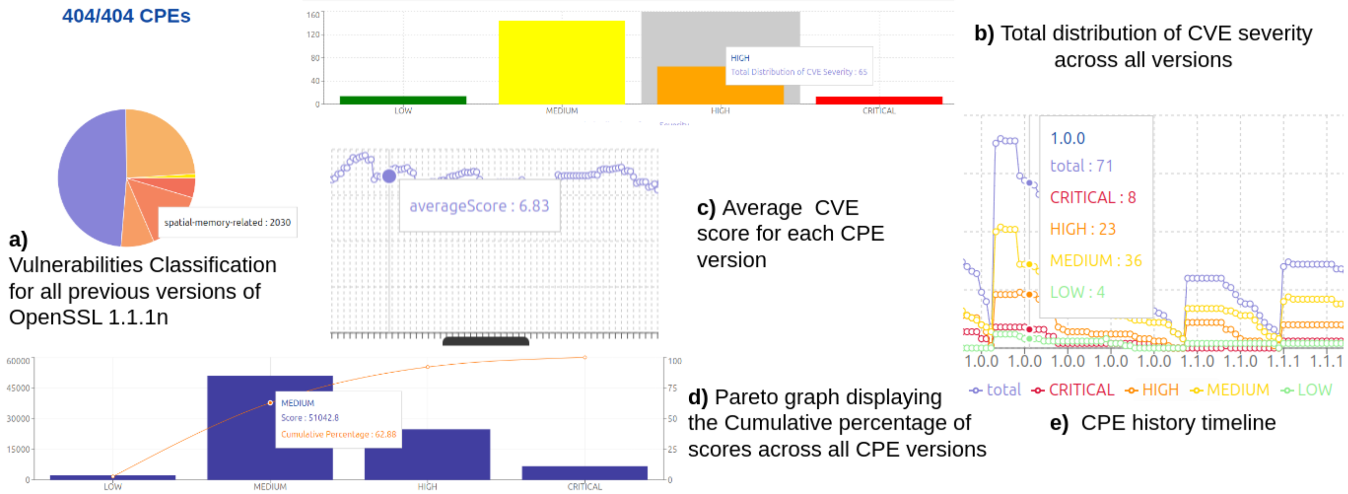


Figure 4: UniBOM GUI screenshot: historical analysis for OpenSSL 1.1.1n.

6.2 Historical Analysis: Understanding Vulnerability Trends

UniBOM's historical analysis capability is one of its most distinctive features, enabling users to examine the vulnerability trajectory of specific components over time (Figure 4). A user can select a specific CPE, e.g. OpenSSL 1.1.1n, and retrieve all previously identified CPE versions of that component along with their associated vulnerabilities (Figure 4b, c, and e).

The severity distribution graphic uses intuitive colour coding to help users quickly identify components with the highest risk levels. Similarly, the memory vulnerability breakdown chart highlights whether the CPE weaknesses are due to spatial or temporal memory issues (Figure 4a).

Time-series charts display the number of vulnerabilities for each component version over time. For instance, the OpenSSL analysis highlights a spike in vulnerabilities for specific versions (Figure 4e), enabling users to identify when and where security issues were introduced. Additionally, Figure 4c shows the average severity score for each previous OpenSSL versions, allowing for deeper insights into version-specific risks.

Pareto charts aggregate vulnerabilities by severity, helping users focus on the most critical issues that account for the majority of vulnerabilities (Figure 4d).

The historical analysis feature not only provides insights into past security trends but, through extrapolation, can also help anticipate and predict potential vulnerabilities in future updates.

7 Conclusion

This paper introduced UniBOM, a comprehensive tool for SBOM generation, analysis, and visualisation, designed to address the challenges of ensuring the security and accountability of IoT systems and beyond. Its web application enhances usability with interactive dashboards, historical trend analysis, and detailed risk visualisation.

UniBOM addresses the limitations of existing SBOM tools by integrating binary, filesystem, and source code analysis into a unified framework, enabling a more fine-grained approach to SBOM creation and seamless vulnerability analysis. Its ability to categorise vulnerabilities by severity and memory-safety relevance enables the effective prioritisation of protective actions, particularly in systems utilising memory-unsafe languages such as C/C++, which remain prevalent in critical systems. A case study involving 258 wireless gateway firmware binary images and the source code of four popular IoT operating systems demonstrated UniBOM's superior detection capabilities through more precise SBOM generation compared to existing solutions.

Although the demonstration examples focus on IoT networked systems security, UniBOM is a general-purpose SBOM tool, and thus applies to diverse software environments. Future work will prioritise embedding UniBOM into operational workflows and introducing further enhancements to improve its effectiveness and strengthen its impact on the security accountability of real-world software systems. In this direction, we plan to integrate UniBOM with AI Bill of Materials (AIBOM) approaches, such as TAIBOM [38], to strengthen security, accountability, and trust in both modern and next-generation AI-enabled software systems.

Acknowledgments

This work was supported by the Innovate UK-funded Secure Networking by Design (SNbD) project, grant number 10028034.

References

[1] [n. d.]. Conan - The open source C and C++ package manager. https://conan.io/. Accessed: 2024-11-28.

[2] 2024. Trivy: Open Source Vulnerability Scanner. https://trivy.dev/v0.33/ Accessed: 2024-11-28.

[3] Saar Amar, Tony Chen, David Chisnall, Felix Domke, Nathaniel Filardo, Kunyan Liu, Robert Norton-Wright, Yucong Tao, Robert N. M. Watson, and Hongyan Xia. 2023. *CHERIoT: Rethinking security for low-cost embedded systems*. Technical Report MSR-TR-2023-6. Microsoft. https://www.microsoft.com/en-us/research/publication/cheriot-rethinking-security-for-low-cost-embedded-systems/

[4] Anchore. 2024. Grype - A vulnerability scanner for container images and filesystems. https://github.com/anchore/grype. Accessed: 2024-11-28.

[5] Anchore. n.d.. Syft: A CLI tool and library for generating SBOMs from container images and filesystems. https://github.com/anchore/syft Accessed: 2024-11-28.

[6] Manos Antonakakis, Tim April, Michael Bailey, Matt Bernhard, Elie Bursztein, Jaime Cochran, Zakir Durumeric, J. Alex Halderman, Luca Invernizzi, Michalis Kallitsis, Deepak Kumar, Chaz Lever, Zane Ma, Joshua Mason, Damian Menscher, Chad Seaman, Nick Sullivan, Kurt Thomas, and Yi Zhou. 2017. Understanding the Mirai Botnet. In *26th USENIX Security Symposium (USENIX Security 17)*. USENIX Association, Vancouver, BC, 1093–1110. https://www.usenix.org/conference/usenixsecurity17/technical-sessions/presentation/antonakakis

[7] J. Biden. 2021. Executive Order on Improving the Nation's Cybersecurity. https://www.whitehouse.gov/briefing-room/presidential-actions/2021/05/12/executive-order-on-improving-the-nations-cybersecurity/ [Accessed: Jun. 20, 2024].

[8] MITRE Corporation. 2024. About Common Weakness Enumeration. https://cwe.mitre.org/about/index.html [Accessed: Jun. 19, 2024].

[9] NIST CPE. 2024. NVD - Common Platform Enumeration (CPE). https://nvd.nist.gov/products/cpe [Accessed: Jun. 19, 2024].

[10] NIST CVE. 2024. NIST's CVE Process. https://nvd.nist.gov/general/cve-process [Accessed: Jun. 19, 2024].

[11] Cybersecurity and Infrastructure Security Agency (CISA). n.d.. The Urgent Need for Memory Safety in Software Products. https://www.cisa.gov/news-events/news/urgent-need-memory-safety-software-products [Accessed: Nov. 28, 2024].

[12] CycloneDX. 2024. CDxgen - Generate SBOMs with CycloneDX. https://github.com/CycloneDX/cdxgen Accessed: 2024-12-03.

[13] Ruian Duan, Ashish Bijlani, Meng Xu, Taesoo Kim, and Wenke Lee. 2017. Identifying Open-Source License Violation and 1-day Security Risk at Large Scale. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security* (Dallas, Texas, USA) *(CCS '17)*. Association for Computing Machinery, New York, NY, USA, 2169–2185. doi:10.1145/3133956.3134048

[14] Muyue Feng, Zimu Yuan, Feng Li, Gu Ban, Yang Xiao, Shiyang Wang, Qian Tang, He Su, Chendong Yu, Jiahuan Xu, Aihua Piao, Jingling Xue, and Wei Huo. 2020. B2SFinder: detecting open-source software reuse in COTS software. In *Proceedings of the 34th IEEE/ACM International Conference on Automated Software Engineering* (San Diego, California) *(ASE '19)*. IEEE Press, 1038–1049. doi:10.1109/ASE.2019.00100

[15] FOSSA. n.d.. Learn SBOMs. https://fossa.com/learn/sboms Accessed: 2024-12-03.

[16] Apache Software Foundation. n.d.. Apache NuttX GitHub Repository. https://github.com/apache/nuttx.git Accessed: 2024-11-29.

[17] Raspberry Pi Foundation. n.d.. Linux GitHub Repository. https://github.com/raspberrypi/linux.git Accessed: 2024-11-29.

[18] R. Grisenthwaite, G. Barnes, R. N. M. Watson, S. W. Moore, P. Sewell, and J. Woodruff. 2023. The Arm Morello Evaluation Platform—Validating CHERI-Based Security in a High-Performance System. *IEEE Micro* 43, 3 (2023), 50–57. http://dx.doi.org/10.1109/MM.2023.3264676

[19] Mozilla Hacks. 2019. Rewriting a Browser Component in Rust. https://hacks.mozilla.org/2019/02/rewriting-a-browser-component-in-rust/ [Accessed: Nov. 28, 2024].

[20] Armijn Hemel, Karl Trygve Kalleberg, Rob Vermaas, and Eelco Dolstra. 2011. Finding software license violations through binary code clone detection. In *Proceedings of the 8th Working Conference on Mining Software Repositories* (Waikiki, Honolulu, HI, USA) *(MSR '11)*. Association for Computing Machinery, New York, NY, USA, 63–72. doi:10.1145/1985441.1985453

[21] Alex Hern. 2016. DDoS attack that disrupted internet was largest of its kind in history, experts say. https://www.theguardian.com/technology/2016/oct/26/ddos-attack-dyn-mirai-botnet Accessed: 2024-11-28.

[22] Tiancheng Hu, Zijing Xu, Yilin Fang, Yueming Wu, Bin Yuan, Deqing Zou, and Hai Jin. 2023. Fine-Grained Code Clone Detection with Block-Based Splitting of Abstract Syntax Tree. In *Proceedings of the 32nd ACM SIGSOFT International Symposium on Software Testing and Analysis* (Seattle, WA, USA) *(ISSTA 2023)*. Association for Computing Machinery, New York, NY, USA, 89–100. doi:10.1145/3597926.3598040

[23] Intel. n.d.. CVE Binary Tool: A tool to scan for known vulnerabilities in software binaries. https://github.com/intel/cve-bin-tool Accessed: 2024-11-28.

[24] R. Jiao, M. Tseng, Q. Ma, and Y. Zou. 2000. Generic Bill-of-Materials-and-Operations for High-Variety Production Management. *Concurrent Engineering: Research and Applications* 8, 4 (Dec. 2000), 297–321. https://doi.org/10.1177/1063293X0000800404

[25] Michael Johnson and Patricia Wilson. 2023. Penetrating Shields: A Systematic Analysis of Memory Corruption Mitigations in the Spectre Era. *arXiv preprint arXiv:2309.04119* (2023).

[26] ReFirm Labs. 2024. Binwalk. https://github.com/ReFirmLabs/binwalk [Accessed: Aug. 5, 2024].

[27] J. Martínez and J.M. Durán. 2021. Software supply chain attacks, a threat to global cybersecurity: SolarWinds' case study. *International Journal of Safety and Security Engineering* 11, 5 (2021), 537–545. doi:10.18280/ijsse.110505

[28] Nicholas D. Matsakis and Felix S. Klock II. 2014. The Rust Language. *ACM SIGAda Ada Letters* 34, 3 (2014), 103–104. doi:10.1145/2663171.2663188

[29] Microsoft. n.d.. SBOM Tool: Generate Software Bill of Materials (SBOMs). https://github.com/microsoft/sbom-tool Accessed: 2024-11-28.

[30] MITRE. 2024. CPE Version 2.3 Specifications. https://cpe.mitre.org/specification/ [Accessed: Jun. 19, 2024].

[31] Microsoft Security Response Center (MSRC). 2019. A Proactive Approach to More Secure Code. https://msrc.microsoft.com/blog/2019/07/a-proactive-approach-to-more-secure-code/ [Accessed: Nov. 28, 2024].

[32] National Security Agency of the United States. 2023. Recommendations for Software Bill of Materials (SBOM) Management. https://media.defense.gov/2023/Dec/14/2003359097/-1/-1/0/CSI-SCRM-SBOM-Management-v1.1.PDF Accessed: 2024-11-28.

[33] NIST IoT. n.d.. Trusted IoT Device Network-Layer Onboarding and Lifecycle Management. https://www.nccoe.nist.gov/projects/trusted-iot-device-network-layer-onboarding-and-lifecycle-management Accessed: 2024-11-28.

[34] OpenWRT Project. n.d.. OpenWRT GitHub Repository - latest release. https://github.com/openwrt/openwrt.git Accessed: 2024-11-29.

[35] The Chromium Project. n.d.. Memory Safety - Chromium Security. https://www.chromium.org/Home/chromium-security/memory-safety/ [Accessed: Nov. 28, 2024].

[36] Zephyr Project. n.d.. Zephyr RTOS GitHub Repository. https://github.com/zephyrproject-rtos/zephyr.git Accessed: 2024-11-29.

[37] Vadim Safronov, Ionut Bostan, Nicholas Allott, and Andrew Martin. 2025. How Memory-Safe is IoT? Assessing the Impact of Memory-Protection Solutions for Securing Wireless Gateways. In *Proceedings of the 14th International Conference on the Internet of Things (IoT '24)*. Association for Computing Machinery, New York, NY, USA, 261–266. doi:10.1145/3703790.3703820

[38] Vadim Safronov, Anthony McCaigue, Nicholas Allott, and Andrew Martin. 2025. TAIBOM: Bringing Trustworthiness to AI-Enabled Systems. arXiv:2510.02169 [cs.SE] https://arxiv.org/abs/2510.02169

[39] John Smith and Jane Doe. 2022. A Comprehensive Survey of Tagged Memory-Protection Techniques. *Comput. Surveys* 53, 4 (2022), 1–30. doi:10.1145/3533704

[40] Shreyas Srinivasa, Jens Myrup Pedersen, and Emmanouil Vasilomanolakis. 2022. Deceptive directories and "vulnerable" logs: a honeypot study of the LDAP and log4j attack landscape. In *2022 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)*. 442–447. doi:10.1109/EuroSPW55150.2022.00052

[41] W. Tang, Z. Xu, C. Liu, J. Wu, S. Yang, Y. Li, P. Luo, and Y. Liu. 2022. Towards Understanding Third-party Library Dependency in C/C++ Ecosystem. In *Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering (ASE '22)*. 1–12. http://dx.doi.org/10.1145/3551349.3560432

[42] W. Tang, Z. Xu, C. Liu, J. Wu, S. Yang, Y. Li, P. Luo, and Y. Liu. 2022. Towards Understanding Third-party Library Dependency in C/C++ Ecosystem. (Oct. 2022). https://doi.org/10.1145/3551349.3560432

[43] DeepSCA Team. n.d.. DeepSCA: SBOM Analysis Tool. https://tools.deepbits.com/ Accessed: 2024-12-03.

[44] OpenWRT Team. n.d.. OpenWRT Version 23.05.5 Archive. https://github.com/openwrt/openwrt/archive/refs/tags/v23.05.5.zip Accessed: 2024-11-29.

[45] Seunghoon Woo, Sunghan Park, Seulbae Kim, Heejo Lee, and Hakjoo Oh. 2021. Centris: A Precise and Scalable Approach for Identifying Modified Open-Source Software Reuse. In *Proceedings of the 43rd International Conference on Software Engineering* (Madrid, Spain) *(ICSE '21)*. IEEE Press, 860–872. doi:10.1109/ICSE43902.2021.00083

[46] Jonathan Woodruff, Robert N.M. Watson, David Chisnall, Simon W. Moore, Jonathan Anderson, Brooks Davis, Ben Laurie, Peter G. Neumann, Robert Norton, and Michael Roe. 2014. The CHERI capability model: revisiting RISC in an age of risk. In *Proceeding of the 41st Annual International Symposium on Computer Architecuture* (Minneapolis, Minnesota, USA) *(ISCA '14)*. IEEE Press, 457–468.

[47] B. Xia, T. Bi, Z. Xing, Q. Lu, and L. Zhu. 2023. An Empirical Study on Software Bill of Materials: Where We Stand and the Road Ahead. *arXiv* (2023). https://arxiv.org/abs/2301.05362

[48] Shengjie Xu, Wei Huang, and David Lie. 2021. In-fat pointer: hardware-assisted tagged-pointer spatial memory safety defense with subobject granularity protection. In *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems* (Virtual, USA) *(ASPLOS '21)*. Association for Computing Machinery, New York, NY, USA, 224–240. doi:10.1145/3445814.3446761

[49] Sheng Yu, Wei Song, Xunchao Hu, and Heng Yin. 2024. On the Correctness of Metadata-Based SBOM Generation: A Differential Analysis Approach. In *2024 54th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. 29–36. doi:10.1109/DSN58291.2024.00018

[50] B. Zhao, S. Ji, J. Xu, Y. Tian, Q. Wei, Q. Wang, C. Lyu, X. Zhang, C. Lin, J. Wu, and R. Beyah. 2022. A Large-Scale Empirical Analysis of the Vulnerabilities Introduced by Third-Party Components in IoT Firmware. In *Proceedings of the 31st ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA)*. 442–454. https://doi.org/10.1145/3533767.3534366