



Mapping the landscape of software vulnerabilities in connected vehicles

Downloaded from: <https://research.chalmers.se>, 2026-10-13 02:44 UTC

Citation for the original published paper (version of record):

Basu, S., Staron, M., Almgren, M. et al (2026). Mapping the landscape of software vulnerabilities in connected vehicles. Software Quality Journal, 34(3). <http://dx.doi.org/10.1007/s11219-026-09787-y>

N.B. When citing this work, cite the original published paper.



Mapping the landscape of software vulnerabilities in connected vehicles

Srijita Basu^{1,2} · Mirosław Staron^{1,2} · Magnus Almgren^{1,2} · Jennifer Horkoff^{1,2} · Christian Berger^{1,2}

Received: 2 December 2024 / Accepted: 29 August 2026

© The Author(s) 2026

Abstract

The increased level of connectivity for vehicles leads to sophisticated and challenging software security concerns for automotive Original Equipment Manufacturers (OEMs) and suppliers. With a Connected Vehicle (CV) communicating with multiple entities (road infrastructure, other vehicles, OEMs, cloud, etc.), new attack surfaces are exposed to intruders. For instance, in 2023, ethical hackers demonstrated remote access to critical vehicle controls like steering and braking through vulnerabilities in infotainment and telematics systems. Several OEMs also reported production disruptions due to ransomware targeting software development infrastructure. Additionally, the current state of the art fails to provide a holistic view of a secure software development process for CVs. This paper presents a survey of automotive software vulnerabilities and associated attack vectors and analyzes the significant changes in security trends. Moreover, the vulnerabilities are tracked across the Software Development Life Cycle (SDLC). We include 413 new automotive software vulnerabilities identified within the last 2 years and 9 months in our study. Additionally, 20 automotive software repositories with 827 vulnerable libraries are identified by scanning sources such as repositories, SBOMs (Software Bill of Materials), images, and manifest files. We provide several security recommendations for software engineering teams based on the findings. Our findings are a step forward to support the maintenance of automotive software quality in terms of security across its entire life cycle.

Keywords Attack vector · Automotive software · Connected vehicle · Software vulnerability

1 Introduction

Neglecting software quality in cyber-physical systems (CPS) like Connected Vehicles (CV) can expose the Original Equipment Manufacturer (OEM), supplier, and in the end occupant and other traffic participants to significant risks. We have seen multiple cyber attacks on CVs that control car engines and locks, expose sensitive user data, and even shut down OEM production (PortSwigger, 2024). Therefore, security is necessary (Kim et al., 2022)

Extended author information available on the last page of the article



for maintaining the overall software quality, as intrusions could lead to fatalities. As per our observation, the automotive software development process for CVs lacks a holistic view of software security (SAFe 6.0, 2023) as a means of maintaining its overall quality. Automotive software for CVs contains not only the code base but also several third-party libraries, open source packages, tools, and, most importantly, communication points, which expose many attack surfaces. Therefore, to maintain the overall quality of automotive software and to ensure an end-to-end secured software development process, the software engineering (SE) team should be well aware of the security vulnerabilities across its' entire Software Development Life Cycle (SDLC) and also prioritize the security of the software supply chain. The perspective of SDLC is important because the security team in many cases concentrates just on the development phase leading to unidentified vulnerabilities originating from the other phases (Assal & Chiasson, 2018).

One survey reported 187 CV vulnerabilities and exploits between 2010 and 2019 (Gülsever, 2019). Observing their attack patterns, we see that the number of remote attacks on these entities surpasses the number of local attacks. With time, the count and nature of these vulnerabilities and the attack patterns have significantly changed. For instance, these findings already highlight that attackers need not have physical access to the CV to launch an attack and can relatively easily contaminate and degrade the system's quality using vulnerable software or unprotected network channels. Additionally, malicious content can be planted in the system long before the attack (e.g., at the factory or during Over-The-Air updates). The SE team should be well aware of these facts and consider the necessary measures as a part of the secure automotive software development process.

Despite such risks, the competitive nature of the automotive industry (Morris et al., 2020) compels most software suppliers to focus more on speedy delivery of software components rather than hardening the security of the software across its life cycle and the software supply chain as a whole.

The above problems motivated us to design and perform a study on CV software vulnerabilities and attack vectors/surfaces. The aim is to provide a holistic view of a secured software supply chain across the software development lifecycle, raise awareness about current automotive security trends, and provide recommendations to the SE team for maintaining automotive software quality regarding SE security practices.

In this paper, we have studied in detail the different automotive software vulnerabilities and associated attack vectors for the span of 2022 - September 2024, as our aim was to closely watch the most recent change in trends across a shorter period. Our study contributes with the following:

- Vulnerability identification from the NVD database and open source automotive repositories using keyword search and vulnerability scanning tools.
- Different mappings (attack vector, OEM/Supplier, SDLC and CAPEC) and trend analysis of the automotive vulnerabilities to identify any significant shift in their nature or pattern for the studied timeline
- Recommendations and discussions regarding secured automotive software development processes for Commercial Off-the-Shelf (COTS) and open-source software.

This paper is organized as follows. Section 2 presents the literature background. Section 3 describes the CV ecosystem and related security concepts, focusing on automotive software

components. In Section 4, the methodology for vulnerability identification and various mapping procedures are detailed. Section 5 provides insights into the study's results and their analysis. We discuss the result implications and offer recommendations for the SE team in Section 6. Section 7 describes the validity of our work. Finally, Section 8 concludes the paper.

2 Related work

The academic literature on automotive cybersecurity has made notable advancement in identifying attack vectors, system-level vulnerabilities, and high-level security strategies. However, several critical gaps remain that hinder the development of a comprehensive, secure software engineering process for CVs. Firstly, existing studies lack a systematic trend analysis of vulnerabilities with respect to severity, attack vector, and SDLC phase. Next, prior work has not thoroughly addressed software supply chain vulnerabilities, particularly the role of third-party libraries and packages used in automotive software. Finally, integrated vulnerability scans of open-source automotive repositories, combined with real-world vulnerability databases, have not been explored in existing studies. Our work is designed to fill these specific gaps, providing a data-driven, replicable methodology to support secure software development across the CV software lifecycle.

Earlier pioneering research played a vital role in raising awareness about the risks in automotive software. For example, the 2015 Jeep Cherokee remote hack by Miller and Valasek (2015) demonstrated how attackers could gain control over steering and brakes via the Uconnect infotainment system, prompting major recalls and policy responses. Similar incidents, including the 2022 ransomware attack on Toyota's supplier Kojima Industries (Iwasaki, 2024), which led to the suspension of operations at 14 Toyota factories in Japan, and the 2023 cyberattack on Denso (Dreyer, 2023), a major automotive parts supplier, have intensified industry-wide focus on secure development practices and supply chain hardening. In response, several industry standards have emerged, such as ISO/SAE 21434 (Costantino et al., 2022) for automotive cybersecurity, SAE J3061 (Costantino et al., 2022) for secure lifecycle management, and UNECE WP.29 regulations (Costantino et al., 2022) mandating cybersecurity management systems for automotive software. While some academic works mention these developments, a comprehensive integration of real-world threats, industry practices, and software engineering techniques is still lacking. Our study bridges this gap by combining empirical vulnerability analysis with actionable recommendations mapped to the SDLC, grounded in both academic and industrial realities. The following surveys and studies have helped us further to understand the gaps in automotive software security and triggered us to conduct a more focused study.

In Xiong et al. (2019), the authors explore the security vulnerabilities across 60 OEMs from the National Vulnerability Database (NVD) and analyze the Common Vulnerabilities and Exposures (CVEs) concerning scores of Common Weakness Enumeration (CWE) and Common Vulnerability Scoring System (CVSS). The results mostly cover the severity statistics and the most common weaknesses identified for CVs. The authors suggest some generic mitigation techniques.

The study presented in Gülsever (2019) highlights CV vulnerabilities from 2010-2019. The data was collected from the NVD and Upstream repositories. The authors also produced

the CWE and attack vectors from these resources and gave a total count for each vector. The percentage of vulnerabilities that could be exploited physically, remotely, etc., was also presented with their generic impacts.

Software bugs related to CVs have been reported and analyzed by Garcia et al. in Garcia et al. (2020). They discuss the commits and bugs associated with Baidu Apollo and AutoWare, widely popular open-source CV communities. The analysis digs into the root cause of such bugs and classifies the most commonly affected software components. Finally, the authors provide recommendations regarding the detection, localization, and repairs of these software bugs.

The survey by Pham and Xiong (2021) covers 189 research papers on Connected-Autonomous Vehicle (CAV) attacks and defense strategies. The identified attack models, targeted CAV components and various access requirements associated with such attacks have been analyzed in this work. Furthermore, the authors highlight the disparity between the academic and industry perspectives on CAV security issues. Overall, the paper sheds light on the current security scenario of CAVs and warns about recently found attack models targeted at CAVs.

Moukahal et al. (2021) propose the Secure Vehicle Software Engineering (SVSE) life-cycle that implements a *security by design* framework. Here, the SDLC is adapted to incorporate CV security requirements into all its phases. The authors identify challenges such as complex CV design, security decay across development, maintenance, incident response monitoring, etc. Later, they describe the different phases of SVSE, including *Cybersecurity Planning*, *Cybersecurity Requirements Analysis*, etc.

In Sun et al. (2021), the CAV framework and the different communication technologies involved are discussed. The authors classify the attacks into, *In-Vehicle Network Attacks* (GPS Spoofing attack, Remote Sensor Attacks, etc.), *Vehicle to Everything Network Attacks* (DoS Attacks, Replay Attacks, etc.) and *Other Attacks*. Next, some established defense strategies (e.g. use spatio-temporal challenge response for Remote Sensor Attack) for each of these attacks is detailed. Finally, the importance of cyber security and safety standards for CAV like ISO 26262, SAE J3061, etc. and selected open security challenges are presented.

As mentioned at the beginning, though the existing works portray the current state of CV security from different perspectives, they fail to cover the security of the software supply chain (i.e., security aspects of the third party libraries/packages that are being used in the development of an automotive software). Moreover, such works lack a current security trend analysis in the form of a continuous process for software engineers. This is essential because if the SE team is well aware of the current vulnerability trends (count, severity, common sources in form of attack vectors, CAPECs, etc.) it helps them to take informed decisions and required measures for the identification and mitigation of similar vulnerabilities in their own system.

Table 1 summarizes the distinguishing features of our work compared to the existing literature. It highlights the importance of our work in terms of automotive vulnerability identification and processing. The **P** and **NP** in the table refers to **Present** and **Not Present** respectively. Below, we highlight how our approach directly addresses the specific gaps.

- It identifies the most recent automotive software vulnerabilities for CVs.
- The vulnerability searching method contains 172 relevant keywords which results in a well populated list of vulnerabilities. We can say that our method encompasses many

Table 1 Comparative analysis

Study	Automotive Vulnerability Identification	Timeline	Method	Trend Analysis with analyzed parameters	SDLC- Mapping	Analyzing Open source Automotive Software and Supply Chain Security
Xiong et al. (2019)	Yes	NP	NVD- searched across 60 OEMs	CVSS base score, metrics and CWE	NP	Not Considered
Gülsever (2019)	Yes	2010-19	NVD and Upstream- searched using 104 keywords (CV terminologies, OEMs/Supplier name, etc.)	Count, CWE, attack vector (method of attack vector mapping not clarified)	NP	Not Considered
Garcia et al. (2020)	No (Automotive software bugs are identified)	NP	Apollo and Autoware were manually analyzed for identifying bugs	Components analyzed includes CAN bus, perception modules, sensor calibration software, etc.	NP	Considered 2 repositories (negligible discussion on bugs present in the associated software libraries)
Moukahal et al. (2021)	No (maps the SDLC phases with automotive cyber-security specific tasks)	NP	Considers security planning across the 4 phases, planning, development, testing, and operation	Discussions made on cyber-security incident response, prioritizing security testing, static code analysis, etc.	Though the aspect of SDLC is considered here, but vulnerabilities are not mapped to the SDLC phases in which they could have originated	Not Considered
Sun et al. (2021)	No (Cyber attacks on CV/AVs identified)	2013-20	Identified and discussed the in-vehicle, vehicle to everything and other attacks for CVs	Analysis on defense strategies like software-based encryption mechanisms (e.g., Trillium), sensor fusion etc.	NP	Not Considered
Our Study	Yes	2022-Sept,24	NVD searched using 175 keywords (CV terminologies, OEMs/Supplier name, etc.)	Count, CVSS, attack vector, associated OEM/Supplier, CAPEC	Present	Considered (20 automotive repositories with maximum vulnerable packages/libraries reported across the supply chain)

more cases of automotive vulnerability reports that might get undetected in other methodologies using limited keyword search.

- The trend analysis and mappings (specially SDLC mapping) presented in our paper was not conducted in any other existing work. As mentioned this analysis is beneficial for SE teams to plan their security tactics.
- Although an attack vector mapping can be found in other work, the mapping methodology and analysis is missing. Our work clearly defines these aspects with a clear scope of replicating the mapping procedure if required.
- Analyzing open source automotive repositories using vulnerability scanning tools and reporting these vulnerabilities across the software supply chain (i.e. vulnerabilities found in the associated libraries and packages) is not provided by any other work. Though one of the work analyzes *Apollo* and *Autoware* but it was a manual analysis focusing on software bugs. It should also be mentioned here that we considered these 2 popular automotive repositories for our study as well, but since the number of vulnerabilities found were negligible, we did not include this in our list of top 20 repositories (Table 7).

In a nutshell, given the connected nature of CVs, this paper aims to collect the newest CV vulnerabilities across the software supply chain and analyze them in a way that would facilitate a secure software development process for the entire SDLC of CVs in the long term and thereby contribute to improve the overall software quality. Our findings can assist automotive OEMs and suppliers in prioritizing security checks. Eg., our study highlights that over 70% of vulnerabilities exist in one very popularly used library in vehicle perception modules that can lead to DoS risks. This insight can guide OEMs in avoiding or hardening vulnerable libraries during integration. Other findings prove the fact that some parts of the automotive ecosystem continue to be critical weak points. E.g., infotainment and telematics systems account for over 20% of the vulnerabilities we analyzed, many with high or critical CVSS scores. This confirms that risks first exposed in the 2015 Jeep Cherokee hack are not isolated incidents but persist systemically. A stronger justification for this would be, *CVE-2023-28898* from our study, which revealed that the Škoda MIB3 infotainment system could be remotely crashed via a malformed request, a vulnerability type still appearing nearly a decade later. These trends and insights would allow OEMs to prioritize secure design and validation practices for infotainment software, particularly during OTA update cycles and third-party component integration.

3 Connected vehicle ecosystem and related security concepts

In the first part of this section, we describe the CV ecosystem and its associated software. Later, we introduce the security concepts and terminologies used in this work.

3.1 Automotive software: CV ecosystem

Modern vehicles and their software are no longer standalone embedded systems but are constantly connected with external actors. The CV ecosystem and the identified attack vectors are presented in Fig. 1. Understanding the CV ecosystem is not only useful for classification purposes but also essential for managing and mitigating vulnerabilities. Different compo-

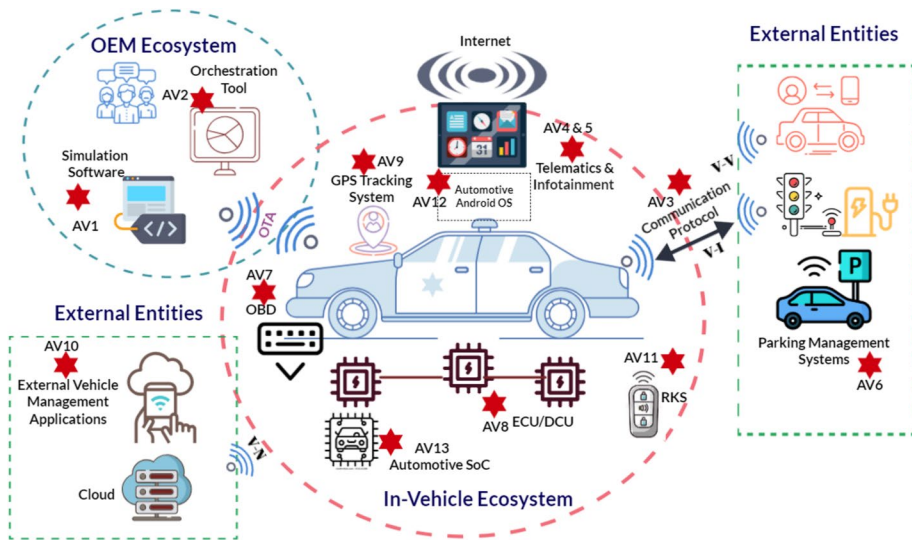


Fig. 1 Connected vehicle ecosystem

nents of the ecosystem are often designed and maintained by different stakeholders, such as OEMs, Tier 1 suppliers, or open-source communities. For instance, certain ECUs may be delivered by Tier 1 suppliers, while SoC (System-On-Chip) firmware or middleware may be sourced from semiconductor vendors. This division of responsibility means that effective vulnerability management requires clarity on which party is accountable for applying security patches, updates, or mitigation. In addition, studying the location of vulnerabilities across different layers of the CV ecosystem allows long-term trends to be observed. For instance, earlier studies (Gülsever, 2019) and our own analysis presented in the upcoming sections indicate that many vulnerabilities historically originated in the in-vehicle ecosystem (e.g., SoC firmware or ECU code). However, in recent years, there is evidence of a shift towards vulnerabilities in external applications, reflecting the growing reliance on cloud based services, OTA update infrastructures, etc. Identifying and visualizing such shifts over time is important because it helps OEMs and suppliers anticipate where future risks are likely to emerge. For example, if vulnerabilities increasingly cluster in external applications, stronger controls in web-facing interfaces, vendor risk management, and secure API design may become higher priorities, while if SoC or ECU-related flaws persist, investment in memory-safe languages, compiler-based defenses, and hardware security features may be more important. Therefore, understanding the CV ecosystem and tracking how vulnerabilities shift across its components over time is crucial for generating actionable intelligence that guides both countermeasures and long-term security investment in automotive systems.

The CV ecosystem can broadly be classified as follows:

- *In-Vehicle Ecosystem – formerly known as embedded vehicle software:* In the context of automotive software, this part of the ecosystem consists of ECU software, software in infotainment and telematics systems of the vehicle, GPS software, Remote Keyless System (RKS), and similar components. Many critical safety functions (e.g., braking, steering, diagnostics) are managed by ECUs, making them high-priority targets for

hardening. Again, infotainment systems, historically exploited in remote attacks (e.g., Jeep Cherokee hack), must be explored for recent vulnerability trends.

- *External Entities*: This entails vehicle monitoring/tracking software, parking management software, traffic monitoring software, various cloud software that stores and processes vehicle data, etc. External systems often handle sensitive user or vehicle data and are frequent entry points for attackers via exposed APIs or misconfigured services. Eg, in 2024, Volkswagen experienced a massive data leak (Darkreading, [n.d.](#)) after attackers accessed unsecured AWS credentials used by a third-party vendor. This led to the exposure of location and contact details of 800,000 customers. The incident highlights the importance of securing cloud-based services and enforcing access controls and credential management when integrating external platforms.
- *OEM Ecosystem/SW development ecosystem*: Different orchestration tools, build tools, container software, and simulation software used for developing, testing, and maintaining the CVs fall under this category. These environments form the backbone of the software supply chain; compromised tooling can introduce malware during build or update processes. Securing this layer helps prevent injection of malicious code at source
- *Communication Platform*: The software existing in the communication protocols like MQTT (Message Queuing Telemetry Transport), Dedicated short-range communications (DSRC), etc. Abdelkader et al. (2021) are steadily rising to form an integral part of the CV ecosystem. These interfaces are vulnerable to spoofing, eavesdropping, and injection attacks. Awareness of their role helps enforce encryption, authentication, and message validation strategies.

Figure 1 illustrates that vulnerabilities in connected vehicles can emerge across all layers of the ecosystem, ranging from in-vehicle ECUs and SoCs to OEM software tools and external cloud services, highlighting the wide and diverse attack surface that must be secured.

3.2 Security concepts & terminologies

Many security terms are used in this study. Each of these and their intended meaning is presented in Table 2

4 Methodology: vulnerability identification, mapping and recommendation

An overview of the method followed in our study along with the associated *Research Question* labeling is presented in Fig. 2. The research questions (RQ1-RQ8) for this study are as follows:

- **RQ1**: What is the trend in the growth of security vulnerabilities for CV automotive software?
- **RQ2**: Which severity patterns have emerged in the security vulnerabilities landscape of automotive software for connected vehicles?
- **RQ3**: What is the vulnerability distribution among the different CV ecosystem components, and how has this distribution changed over time

Table 2 Security concepts/terminologies

Term/Concept	Description in the context of our work
Vulnerability	An inherent weakness in the requirement specification, design, development, testing, configuration, deployment, or maintenance of the automotive software that renders it susceptible to a threat
Threat	Anything that can disrupt automotive software's operation, functioning, integrity, or availability. Threats exploit vulnerabilities
Attack Vector	The source of vulnerability, i.e., the physical or logical entity of the CV ecosystem in which the vulnerability has been identified
Threat Vector	Covers a broader scope, encompassing not only the methods of attack but also the potential sources and motivations behind the
CVE	The Common Vulnerabilities and Exposures (CVE) is a glossary maintained by MITRE Corporation to track and catalog vulnerabilities in consumer software and hardware.
CAPEC	The Common Attack Pattern Enumeration and Classification (CAPEC) by MITRE Corporation provides a publicly available catalog of common attack patterns providing an insight into the threat vectors used by attackers to exploit vulnerabilities
CVSS Score	The Common Vulnerability Scoring System (CVSS) is a standardized framework by NVD for measuring the severity of any information system's security vulnerabilities. The scores vary between 0 and 10, with higher scores meaning more severe issues. The qualitative rating is mapped as 0.0 None, 0.1 – 3.9 Low, 4.0 – 6.9 Medium, 7.0 – 8.9 High, 9.0 – 10.0 Critical
CWE	The Common Weakness Enumeration (CWE) is a list of common software and hardware weaknesses maintained by MITRE Corporation. A "weakness" is a condition in a software, firmware, hardware, or service component that, under certain circumstances, could contribute to introducing vulnerabilities in the system.
CPE	Common Platform Enumeration (CPE) is a standard developed by NIST to identify software applications, operating systems, and hardware components within a company infrastructure.

- **RQ4:** What is the vulnerability distribution among software suppliers/vendors or OEMs?
- **RQ5:** What is the trend of identified vulnerabilities across the SDLC phases of automotive software for connected vehicles?
- **RQ6:** What are the most popular common attack methods (CAPEC) for the identified vulnerabilities of automotive software for connected vehicles?
- **RQ7:** What are some of the most vulnerable open-source automotive software repositories and the underlying vulnerable software libraries used by these repositories?
- **RQ8:** How do the vulnerability severity levels and CWE distributions in automotive software compare to those in adjacent cyber-physical domains such as IoT and Industrial Control Systems (ICS)?

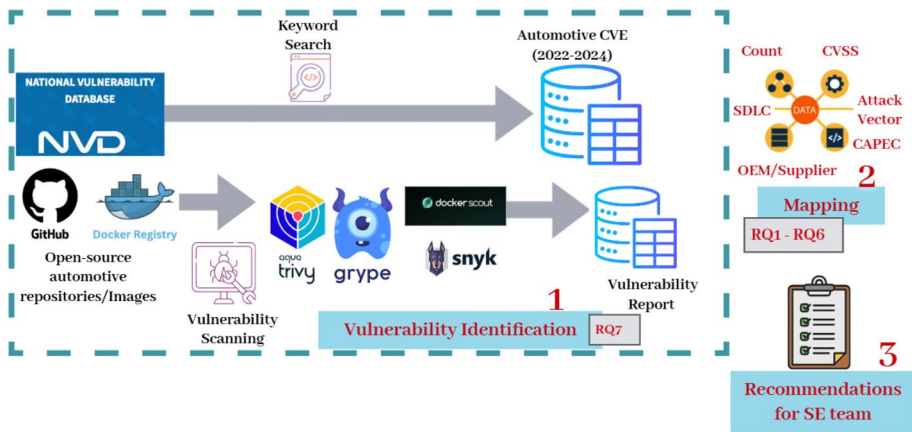


Fig. 2 Study methodology

The upcoming subsections provide a detailed insight of each of the phases described in Fig. 2. All manual analysis and decisions involved a group of two authors, followed by a peer review conducted by three other authors. To ensure the accuracy and reliability of the manual analyses (e.g., keyword selection, repository filtering, attack vector mapping, etc.), we adopted a structured review process. Two authors independently performed each manual labeling task. To assess inter-rater agreement, we calculated Cohen's Kappa score, which resulted in a value of **0.488**, indicating a moderate level of agreement. Disagreements were discussed jointly and resolved through consensus.

4.1 Vulnerability identification and collection

In the first phase, we focused on the identification of potential vulnerabilities by exploring existing vulnerability databases and running some vulnerability scanning tools on selected open-source automotive repositories, in particular:

1. We explored the NVD database and conducted keyword searches to identify vulnerabilities. In total, 175 keywords were used, which could be divided into three categories: i) T-CV: Well-known terms related to CVs like *Infotainment*, *Telematics*, *Simulation*, *ECU*, *OBD*, etc. ii) T-OEM: Terms representing a selection of well-known OEMs, Automotive Software, etc., and iii) T-GH: Terms representing some of top-ranked open source GitHub repository names taken from a study (Kochanthara et al., 2022) which presented the landscape of automotive software in GitHub. The detailed list of keywords, along with their category, is depicted in Table 8. The keywords were selected to ensure broad coverage across the CV ecosystem, targeting commonly affected components (e.g., ECUs, telematics, communication platforms) and high-risk OEMs or technologies referenced in past vulnerability analysis articles (Gülsever, 2019) and cyber-incident reports (Global Automotive Cyber Security Report, 2024) in automotive domain. Selection of keywords, removal of repetitions, and false positives from the search results (i.e., a vulnerability found due to a keyword search that was not a valid automotive vulnerability) were done manually.

2. To mitigate the risk of missing vulnerabilities due to reliance solely on the NVD, we cross-referenced six additional authoritative sources: (i) *VulDB*, (ii) *CISA's Known Exploited Vulnerabilities (KEV) catalog*, (iii) *MITRE*, (iv) *Bugtraq* (v) *JVN (Japanese DB)* and (vi) *ExploitDB*. From these, we identified 51 additional vulnerabilities relevant to the automotive domain: 35 from VulDB, 15 from CISA KEV, and 1 from MITRE. We applied the same keyword-based filtering as shown in Step 1 for these additional sources as well. VulDB surfaced additional vulnerabilities particularly involving automotive SoC software that though were present in NVD were eliminated in the search process. For CISA KEV, we included only those vulnerabilities that were *added to the catalog within our studied timeline* (January 2022 to September 2024), ensuring temporal alignment with our dataset. These additional sources were deduplicated before merging with our primary dataset. Notably, vulnerabilities listed in CISA KEV are confirmed to be actively exploited in the wild, and therefore carry strong security relevance. Including them ensures our dataset captures real-world threat exposure, not just theoretical risks.
3. In addition to Step 1, we selected a group of open-source automotive GitHub repositories. The initial list (more than 100 entries) was taken from Kochanthara et al. (2022) and some industry interactions (e.g., The MQTT broker platforms, automation tools, orchestration tools, virtualization software, etc. used by some popular OEMs). It was further filtered based on its relevance to our concept of automotive software (as explained in Section 3.1), presence and frequency of appearance in the NVD (for the study timeline), and usage in the industry. Finally, we settled for 20 GitHub repositories. The list is provided in Table 7 of the next section.

Next, we have used five popular open-source vulnerability scanning tools (*Snyk*, *Grype*, *Trivy*, *Docker Scout*, and *Vulert*) for inspecting the aforementioned GitHub repositories, their corresponding Docker images, and even the SBOMs to identify and report the CVEs in the underlying libraries/packages used by the repositories. The scanning tools used in this paper were selected based on recommendations provided by the Open Web Application Security Project (OWASP) (Wichers, n.d.), Cybersecurity and Infrastructure Security Agency (CISA) (Grype, n.d.), and after discussions with security personnel from the industry. Moreover they represent a combination of industry-leading solutions widely recommended for software composition analysis (SCA), container security, and SBOM vulnerability scanning. For example, *Snyk* and *Trivy* are among the most widely adopted tools in commercial DevSecOps pipelines (Wichers, n.d.; Snyk, 2023). *Grype* is praised for its speed and offline capabilities (Grype, n.d.). *Docker Scout* integrates directly with container registries and Docker Hub, allowing in-place CVE triage and remediation (Docker Inc., n.d.). *Vulert*, meanwhile, focuses specifically on SBOM and manifest-based scanning (e.g., CycloneDX, SPDX) (Vulert, n.d.). Using a diverse set of tools improved CVE detection coverage and enabled result cross-validation. This was important to understand the existing state of security of the automotive software supply chain. The different modes of execution (commands/web interface), input, and output for these tools are presented in Table 3. We came up with RQ7 after this step.

4.2 Mappings: vulnerability count and CVSS

The vulnerability count per year and the severity metrics i.e. CVSS is furnished from the NVD records directly. No manual mapping is required here. CVSS Version 3.0 scores has

Table 3 Method: vulnerability scanning tools

Tool	Mode of Execution	Input	Output
Snyk	https://app.snyk.io/org/sriabir/add/github-re-adonly (web)	GitHub Repository, Docker Image	Package Name, CVE ID, CVSS, CWE, Introduced & Fixed Versions
Grype	grype (command line)	Container Image, SBOM, Local Directory	Package Name, Type of Package (e.g., python, go, etc.), CVE ID, Severity level, CWE, Installed & Fixed Versions
Trivy	trivy – scanners vuln repo (command line)	Container Image, Filesystem, Git Repository (remote), Virtual Machine Image	Package Name, CVE ID, Description, Severity level, Fixed Version, Status
Docker Scout	docker scout cves (command line)	Container Images, OCI layout directory, Tarball archives, Local Directory or File	Package Name, CVE ID, Severity level, Fixed Version, link to docker scout description page for the vulnerable package
Vulert	https://vulert.com/dashboard/applications/create (web)	Docker Configuration file, Manifest File (SBOM, .lock, .go, .yaml, etc.)	Package Name, CVE ID, Description, CVSS, Patch

been considered in this study as this gave the maximum available scores (using other versions lead to the unavailability of CVSS for some vulnerabilities). A few vulnerabilities that were too recent and for which analysis was yet to be done by NVD, no CVSS could be fetched.

RQ 1 and RQ2 were formulated after this step. We also conducted a comparison of CVSS for our work and the work presented in Gülsever (2019).

4.3 Mapping: attack vector

As per the definition of attack vector mentioned in Table 2, the CVE description of the vulnerabilities is studied manually and mapped to their corresponding attack vectors. The list of attack vectors was inspired from the works presented in Gülsever (2019) and Sun et al. (2021). Figure 1 also reflects the placement of these attack vectors in the CV ecosystem. For example, vulnerability *CVE-2023-28898: The Real-Time Streaming Protocol implementation in the MIB3 infotainment incorrectly handles requests to /logs URI, when the id parameter equals zero. This issue allows an attacker connected to the in-vehicle Wi-Fi network to cause denial-of-service of the infotainment system when certain preconditions are met. Vulnerability discovered on Škoda Superb III (3V3) 2.0 TDI manufactured in 2022*, clearly shows that the source of vulnerability, i.e., the attack vector, is the *Infotainment System Software*.

The complete list of Attack Vectors is presented in Table 4. This mapping led us to the vulnerability distribution among different types of automotive software in the CV ecosystem. RQ3 was formulated after this mapping. Additionally, we also conduct two comparisons here. Firstly, we compare the attack vector distribution obtained in our methodology with the one

Table 4 List: attack vector

Attack Vector	Part of CV Ecosystem	Examples
Communication Platform	Communication Platform	MQTT, Bluetooth, Wi-Fi, etc.
ECU	In-Vehicle Ecosystem	Firmware, Airbag ECU software, Remote Keyless Software, On-board diagnostics ports etc.
External Application	External Entities	Mobile, Web, Cloud Applications interacting with vehicle or handling vehicle data
Sensors	In-Vehicle Ecosystem	GPS, LiDARs etc.
Infotainment Software	In-Vehicle Ecosystem	Car Infotainment software like MIB3, MyLink infotainment system, etc.
Orchestration Tool	OEM Ecosystem	Bazel, Apache AirFlow
Simulation Software	OEM Ecosystem	Simulators used for development, training, and validation of CVs (e.g. Carla)
SoC	In-Vehicle Ecosystem	Automotive System on Chip, Embedded Software
Telematics	In-Vehicle Ecosystem	Software responsible for logging, visualizing & storing vehicle diagnostic data (e.g. TeslaMate)

provided by Gülsever (2019) and secondly we replicate the work of Gülsever (2019) for the recent time period (i.e. 2022– September, 2024) and repeat the comparison with our work.

4.4 Mapping: Vendor/Supplier/OEM

This is again furnished from the NVD records directly and no manual mapping is required. The Hyperlinks, CPEs, and sometimes the vulnerability description itself contains the name of the associated OEM/Supplier/Vendor. We formulated RQ4 after this mapping.

4.5 Mapping: SDLC

The identified vulnerabilities from Section 4.1 are mapped to the particular phase of the SDLC in which they could have been introduced or originated. This helps the SE team to prioritize the security checks and controls for each phase.

The mapping methodology was based on CWE. The CVEs are generally mapped to particular CWEs. CWEs, in turn, have a description that includes the phase in which they could be introduced, e.g., CVE-2023-28895 is mapped to CWE-798, *Use of Hard-coded Credentials*. In the CWE-798 description page (CWE-798, [n.d.](#)), we could find that the introduction phase is mentioned as *Architecture and Design*. This supports the fact that CVE-2023-28895 could be introduced in the design phase of the SDLC.

We found vulnerabilities that could be mapped to multiple phases, and for some of them, mapping was not possible (e.g. *CVE-2024-27000* for which no CWE mapping was given and therefore theSDLC mapping could not be performed). We formulated RQ5 after this step.

4.6 Mapping: CAPEC

In the next analysis step, we map automotive software vulnerabilities to the Mechanisms of Attack according to CAPEC (MITRE). This mapping helps us analyze common attack strategies (threat vectors) and devise common mitigation techniques accordingly. We followed three methods for this mapping:

1. *Direct Mapping from CVE to CAPEC ID*: We used the ASRG (Automotive Security Research Group) repository for direct mapping. For example, CVE-2023-28901 was mapped to CAPEC-116, *Excavation*, as referenced in CVE-2023-28901 (n.d.).
2. *Indirect Mapping using CWE*: We performed indirect mapping through CWE. For instance, CVE-2023-3121 was mapped to CWE-918, *Server-Side Request Forgery (SSRF)* (CVE-2023-3121, n.d.). CWE-918 was subsequently mapped to CAPEC-664, *Server-Side Request Forgery* (CWE-918, n.d.). This process resulted in mapping CVE-2023-3121 to CAPEC-664.
3. *Manual Mapping*: When the above methods could not be applied (e.g., due to unavailable ASRG data or lack of a CWE mapping), we conducted manual mapping. First, we analyzed the vulnerability description to identify keywords (e.g., *Cross-Site Scripting (XSS)*). We then listed the different mechanisms of attack described on the CAPEC page (CWE-918, n.d.) and conducted a keyword search for the identified terms. For example, CAPEC-63, *Cross-Site Scripting (XSS)*, was identified this way. In cases where keyword searches were ineffective, we thoroughly examined the description of the vulnerability and associated it with conceptually related CAPEC mechanisms.

The information was insufficient in several cases, and we could not map some of the CVEs to corresponding CAPECs. RQ6 was formulated at the end of this mapping.

4.7 Mappings: vulnerability details IoT & ICS domain

We also added a study on the CWE and severity details for IoT and ICS domains. The CISA, ICS Advisory was explored for the ICS vulnerabilities in the studied time span. The details of IoT CWEs and severity was referred from the survey by Ryan and Rozier (2024).

The studied automotive vulnerabilities along with the above mentioned mappings are made available as replication package.¹ While the foundational techniques employed in this study, keyword filtering of vulnerability databases and SBOM-based scanning are established in vulnerability analysis, our methodological novelty lies in how these are customized and extended for the automotive software domain. We designed a domain-specific keyword taxonomy (T-CV, T-OEM, T-GH) and applied a multi-phase validation process to ensure contextual relevance to connected vehicle systems. We further enriched the dataset by integrating actively exploited vulnerabilities from CISA KEV, automotive-specific records from MITRE, and vendor-focused entries from VulDB. Importantly, we aligned vulnerabilities not only to CWE and CVSS, but also to SDLC phases and attack vectors specific to the CV ecosystem, creating a layered perspective that goes beyond prior work that typically analyzes CVEs in isolation. The next section presents the results obtained using these methodologies.

¹https://github.com/SriAbir/CV_Vulnerability_22-24.git

5 Results and analysis

Given the methodology and mappings previously described, this section provides the results as answers to our research questions.

RQ1: What is the trend in the growth of security vulnerabilities for CV automotive software?

The first investigation focuses on the changing trend in automotive vulnerabilities from 2022 to 2024 (until September). The results show a maximum number of vulnerabilities (173) during 2023. It was a bit lower (134) for 2022. For 2024, since the data was collected up to and including September, it resulted in a lower number, 106 (by extrapolation, this may result in fewer vulnerabilities in total for 2024 than 2023). Though our study timeline ranges between 2022-2024 we provide an additional statistics showing how the vulnerability trend has changed over a longer period of time (2018-2024). It highlights that the year 2019 experienced a remarkable explosion in the number of automotive vulnerabilities (Fig. 3).

The detailed quarterly statistics are provided in Fig. 4. Two vital observations here are: i) Maximum number (37) of vulnerabilities with **HIGH** CVSS was reported in the third quarter (September-December) of 2023, ii) Maximum number (17) of vulnerabilities with **CRITICAL** CVSS was reported in the second quarter (May-August) of 2022.

Summary: The year 2023 (particularly concentrated on the third quarter) has been identified with a maximum number (142) of CV software vulnerabilities.

RQ2: Which severity patterns have emerged in the security vulnerabilities landscape of automotive software for connected vehicles?

We start by reporting the severity of the **413** automotive vulnerabilities we identified and later compare the results with Gülsever (2019). The CVSS score (**CRITICAL**, **HIGH**, **MEDIUM**, **LOW**, **NOT PROVIDED**, i.e., **NP**) were taken from the NVD database.²

The detailed statistics are provided in Fig. 5 where we can see a significant shift in the severity patterns. In the study conducted by Gülsever (2019), between 2009 and 2016, vulnerabilities with **MEDIUM** CVSS score took the lead with 48.7% whereas, in our study, the majority of the identified vulnerabilities have **HIGH** CVSS severity score (44.8%). Another observation was the increase in the percentage of **CRITICAL** vulnerabilities from 5.1% (Gülsever, 2019) to 16.3%.

The increasing occurrence of **HIGH** and **CRITICAL** vulnerabilities in automotive software is particularly concerning because of the safety-critical nature of connected vehicles. Unlike general IT systems, exploitation of such vulnerabilities may compromise vehicle control functions, expose occupant data, or disrupt fleet and infrastructure operations. For example, a single remotely exploitable vulnerability in a telematics or infotainment system can act as a gateway to safety-critical ECUs, amplifying its impact far beyond the initial

²There were instances when a single vulnerability was associated with multiple CVSS scores, e.g. CVE-2023-29460 has CVSS scores, 9.8 **CRITICAL** (Source: NIST: NVD) and 7.8 **HIGH** (Source: CNA: Rockwell Automation). In all such cases, we considered the higher number, i.e., 9.8 **CRITICAL**, was taken for the above case.

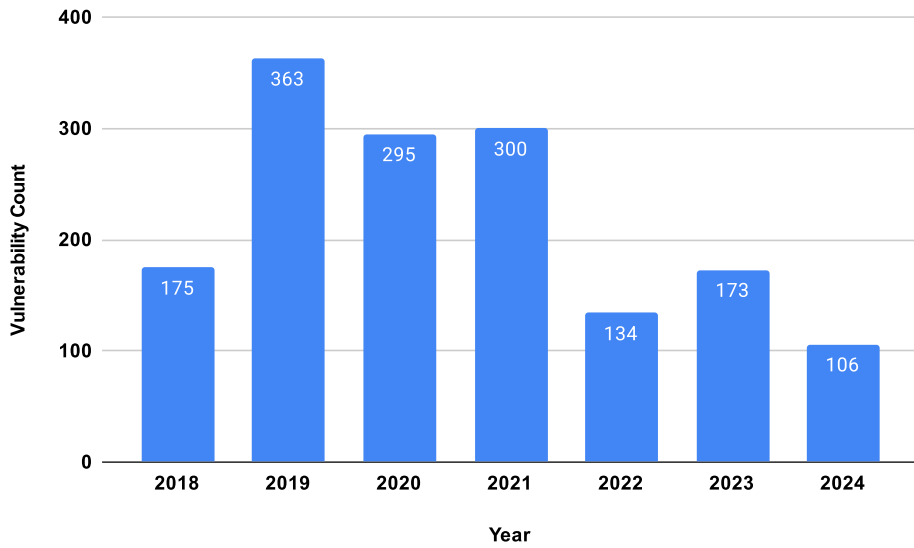


Fig. 3 Trend of automotive vulnerabilities (2018-2024)

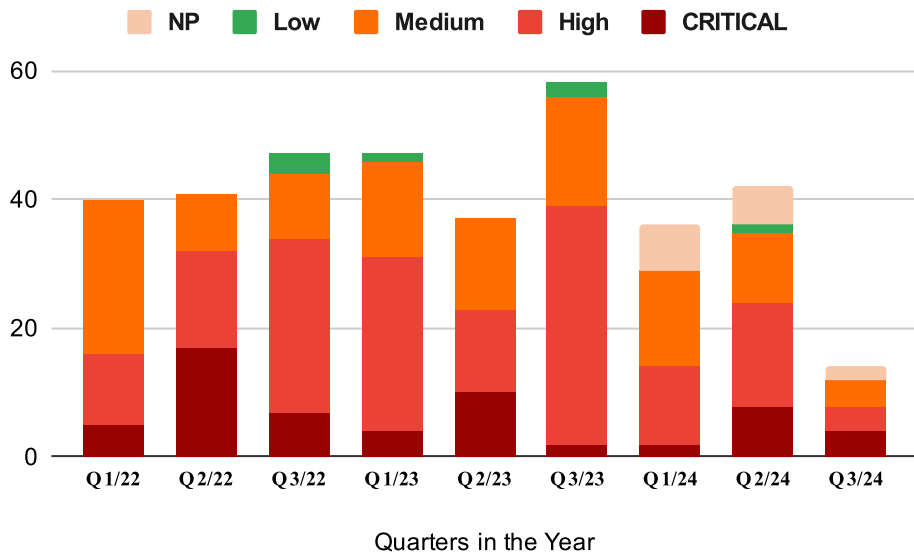


Fig. 4 Trend of automotive vulnerabilities

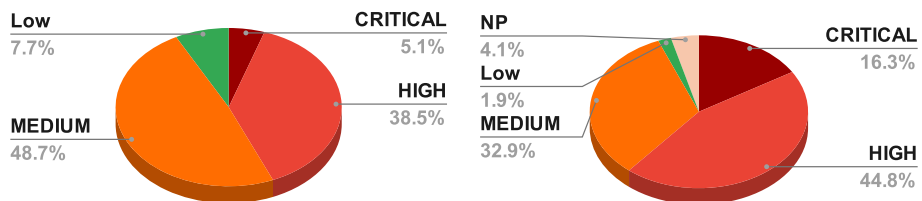


Fig. 5 i) Left Image: Severity Statistics for CV (2010-2019) from Gülsever (2019) ii) Right Image: Severity Statistics for CV from our study

component. The concentration of **CRITICAL** vulnerabilities in 2022 and **HIGH**-severity vulnerabilities in 2023 suggests that attackers increasingly target weak points with the potential for systemic exploitation.

Summary: There has been a noticeable shift in the severity patterns in the last few years with an increasing number of **CRITICAL** vulnerabilities and an overall majority of **HIGH** vulnerabilities. This implies a growing impact on CV security.

RQ3: What is the vulnerability distribution among the different CV ecosystem components, and how has this distribution changed over time?

The 413 vulnerabilities are mapped to 9 attack vectors. This mapping helps us identify the source (software component) of the vulnerabilities in the CV ecosystem. Figure 6 presents the detailed statistics that show that external applications (36.2.3%) are responsible for the maximum number of reported vulnerabilities. It is followed by SoC (23.5%) and Telematics (14.8%).

A larger picture with respect to the CV ecosystem has also been derived. As mentioned in the methodology, we made a threefold comparison. Firstly, we compared the vulnerability distribution across the CV ecosystem from our study (2022- September, 2024) with the vulnerability distribution provided in Gülsever (2019) (2010- 2019). Next, we replicated the method followed in Gülsever (2019) for the current period and compared the results with our results again (this time the study period for both were 2022 - September, 2024). It was important to understand whether the variation in results was due to the different timelines in which the vulnerabilities were studied or the different mechanisms of study (used in Gülsever, 2019 v/s our study)

Figure 7.ii shows that the percentage of *External* vulnerabilities has significantly increased in the recent timeline (2022- September, 2024) as compared to the timeline 2010-2019. A subtle growth in OEM vulnerabilities across the two timelines was also noticed. Figure 7.iii with the replicated study also shows a considerable number of *External* vulnerabilities. This implies a clear trend shift in attack vectors from 2010-2019 to 2022-2024

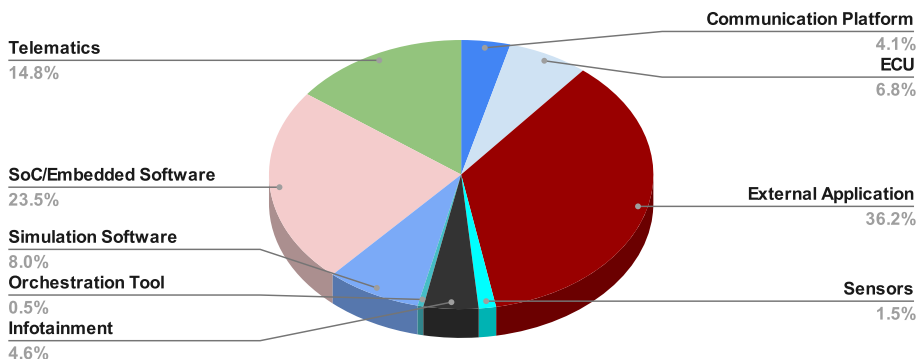


Fig. 6 Statistics for Automotive Vulnerabilities: By attack vector/software type

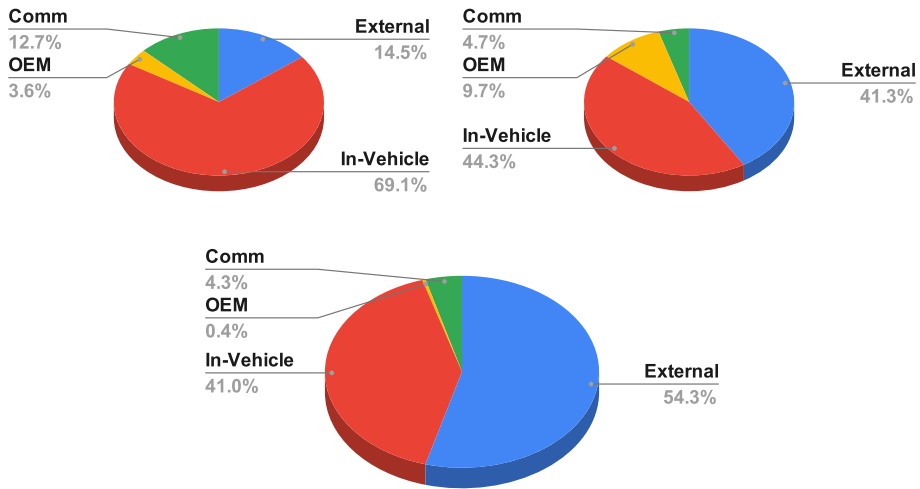


Fig. 7 i) Left Image: Vulnerability Distribution across CV ecosystem in Gülsever (2019) (2010- 2019) ii) Right Image: Vulnerability Distribution across CV ecosystem in our study iii) Bottom Image: Vulnerability Distribution across CV ecosystem in our replication of Gülsever (2019)

(from in-vehicle to external software). Another observation was that the replicated study had not reported *OEM* vulnerabilities because of different study mechanisms used by Gülsever (2019) and our paper.³

Summary: There has been a clear shift in automotive vulnerabilities from in-vehicle to external ecosystem i.e. the external applications act as the major contributing attack vectors.

RQ4: What is the vulnerability distribution among software suppliers/vendors or OEMs?

Statistics of software suppliers and OEMs responsible for developing the vulnerable software for CVs have been presented in Fig. 8 (The ones with single vulnerable products have been ignored here). The Commercial Off-the-Shelf and open-source software providers are denoted with blue and pink bars respectively. *Qualcomm* with a purple bar signifies that some of its vulnerabilities were associated to open-source projects (for which fixes and commits could be found) and for others it was not available. **Qualcomm** tops the list with 58 vulnerabilities spread across several automotive chipset software stack (e.g., Snapdragon Auto 5G Modem-RF, Snapdragon Auto 4G Modem, Snapdragon 820 Automotive Platform, QCA6574AU, QCA6564AU, QCA6595AU, QCA6584AU, etc.). **SourceCodester** occupies the second place with 55 vulnerabilities in the product *Vehicle Service Management System*. This is an open-source PHP project that any vehicle repair/service shop or business can use. The second place is taken by **Grafana** (open-

³It should be noted that there are certain vulnerabilities that can be linked to more than one kind of software component, e.g., *CVE-2023-3028* can be associated with both **telematics** and **communication protocol**. In such cases, we considered the vulnerability of all such identified software components.

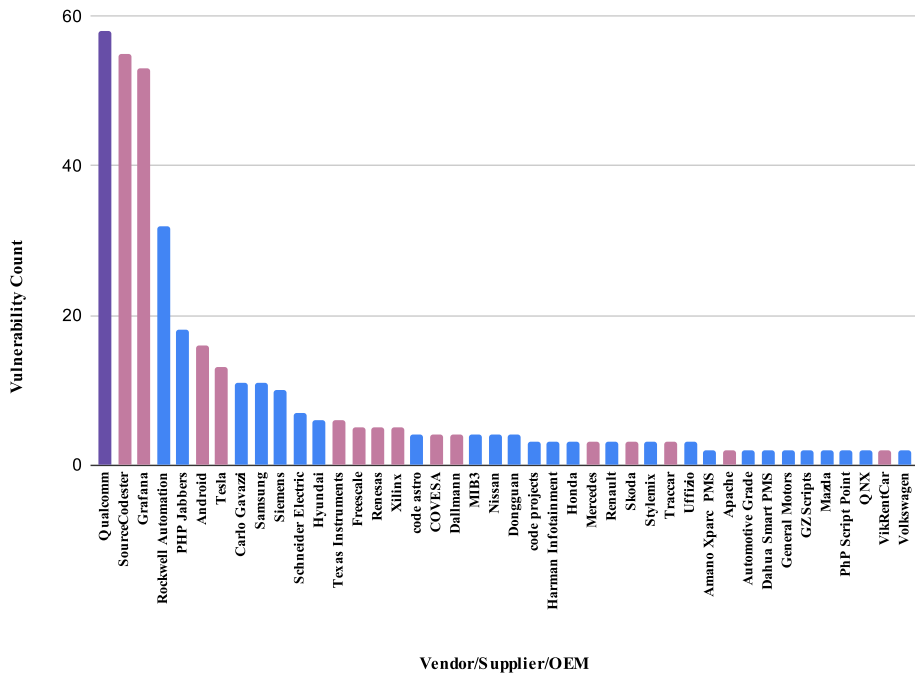


Fig. 8 Statistics for automotive vulnerabilities: by supplier/OEM

source Go and Typescript project) with 53 vulnerabilities spread across different software components like *GeoMap plugin*, *IoTDB Grafana Connector*, *Sourcegraph*, etc. used in the telematics unit of the vehicle.

These vulnerability distributions raise another important question, *why do certain OEMs and suppliers exhibit disproportionately higher vulnerability counts?* Based on the data investigated and the technical context of each supplier, several factors appear to contribute to this pattern.

High-volume, security-critical SoC ecosystem Qualcomm’s dominance in our dataset (58 CVEs) reflects both its extensive footprint in automotive chipsets and its proactive vulnerability reporting processes rather than simply poor software quality. Qualcomm supplies entire automotive platforms, such as Snapdragon Auto 4G/5G Modem-RF, Snapdragon 820 Automotive Platform, and the QCA65xx/95xx Wi-Fi/Bluetooth chipsets, used across multiple OEMs. This makes its software stack a large and attractive attack surface, spanning baseband firmware, drivers, and application frameworks. Because these platforms integrate open-source components (Linux kernels, Android Automotive layers, open cryptographic libraries), Qualcomm’s advisories often include CVEs inherited from upstream open-source code as well as its proprietary patches. Furthermore, Qualcomm participates in coordinated disclosure programs and publishes security bulletins (e.g., “Qualcomm Security Bulletins” Qualcomm Technologies, Inc., 2024) regularly, which increases the visibility of its vulnerabilities compared to more opaque suppliers. This combination of high market share, layered software stacks, and transparent disclosure explains its higher apparent vulnerability count in our dataset.

Technology use and ecosystem exposure OEMs and suppliers that build or integrate modular, plugin-based, or externally connected platforms are more exposed to CVEs by design.

- Grafana, for example, is a telemetry dashboard widely used in telematics units for log visualization and cloud monitoring. It relies on dynamic rendering (JavaScript), open plugins, and integrations with cloud services, making it vulnerable to *privilege escalation* (CAPEC-233) and *XSS* (CAPEC-63), two of the most frequent CAPECs as evident from our results.
- Similarly, SourceCodester's PHP-based vehicle management system, while not specific to OEM production software, is representative of how off-the-shelf service tools, often used in garages or service chains, introduce legacy web vulnerabilities like SQL injection and XSS. This reflects that technology stack complexity, especially reliance on web-facing components or third-party libraries, directly correlates with higher vulnerability incidence.

Development practices and Codebase maintenance policies Projects like Grafana are open-source and community-driven, resulting in a high velocity of commits and broader vulnerability disclosure due to transparency. In contrast, some proprietary OEM tools may under-report vulnerabilities unless required by regulation or coordinated disclosure. Additionally, open-source packages often receive CVEs for transitive dependencies (e.g., Node.js modules in Grafana), which can inflate CVE counts relative to more monolithic proprietary software.

Regulatory visibility and compliance Enforcement Suppliers with stricter code review, integration testing, and static analysis in CI/CD pipelines, such as those aligned with ISO/SAE 21434, tend to report fewer vulnerabilities or resolve them upstream before disclosure. For example, suppliers like Tesla and Samsung report far fewer publicly disclosed CVEs compared to open-source platforms such as Grafana and SourceCodester, which lack formal SDLC controls. This aligns with observations in Costantino et al. (2022), where ISO/SAE 21434 compliance is shown to support proactive risk mitigation throughout the software lifecycle.

Component reuse across brands or platforms This is a common practice in cost-sensitive OEMs which may amplify the apparent vulnerability burden. For example, *CVE-2023-6073* affects the ICAS-3 infotainment system, which is reused across multiple VW Group vehicles. In such cases, a single vulnerability cascades across multiple OEM models, artificially inflating the appearance of supplier-level insecurity when the root cause lies in shared platform architecture.

Taken together, these factors suggest that higher vulnerability counts do not always reflect lower software quality. Instead, they often signal a combination of open-source transparency, aggressive reuse of modular components, diverse regulatory obligations, and the evolving nature of the connected vehicle software stack. Understanding these distinctions is critical for interpreting vulnerability data in a way that supports practical remediation and supplier engagement.

Summary: *Qualcomm* and *SourceCodester* reports to have produced the maximum number of vulnerable software products.

RQ5: What is the trend of identified vulnerabilities across the SDLC phases of automotive software for CVs?

Table 5 presents the attack vectors/automotive software types we identified in this study as the row headers and the different phases of an SDLC as column headers. The numbers in the table represent the number of identified vulnerabilities associated with software that originated in a particular SDLC phase, e.g., the entry **13** depicts that 13 vulnerabilities originated in the *Development Phase* of the different *Infotainment Software* identified in this study. Here, one observation is, unlike the other software, which has the main vulnerability concentration over the development phase, the vulnerability count is quite evenly distributed across the different phases (Software Requirement Specification, Design, Development, and Production) for *Infotainment and Telematics Software*. The columns for *Post-Production* and *Testing* are almost empty as none of the identified vulnerabilities in our study was found to originate in these phases. However, we may encounter vulnerabilities, like malicious software patches in the post-production phase or a vulnerable scanning tool used for scanning software in the testing phase. Therefore, these two phases are included yet left for our future work to study.

In Fig. 9, we consider the total vulnerability counts in each SDLC phase where the Development phase surpasses all the other phases.

Summary: The maximum number of vulnerabilities are found to originate in the *Development* phase followed by the *Design* phase.

RQ6: What are the most popular common attack methods (CAPEC) for the identified vulnerabilities of automotive software for CVs?

We map the identified vulnerabilities to their corresponding attack mechanisms and CAPEC IDs. The results were quite interesting, where 90 CAPEC IDs could be identified

Table 5 Automotive vulnerability trend across SDLC phases

Software Type	SDLC Phases					
	Software Requirement Specification	Design	Development	Testing	Production	Post-Production
Communication Platform	0	11	12	0	4	0
ECU	0	11	20	1	2	0
External Application	11	55	131	0	5	0
Sensor	0	6	3	0	0	0
Infotainment	3	11	13	0	8	0
Orchestration Tool	0	2	2	0	0	0
Simulation Software	2	19	33	0	1	0
SoC/Embedded System	1	9	40	0	0	0
Telematics	4	35	51	0	7	0
Total	21	159	305	1	27	0

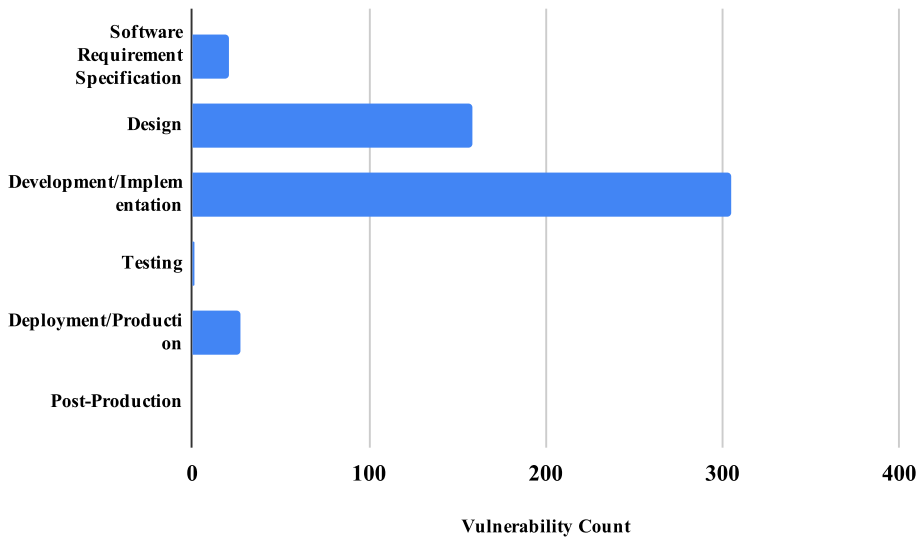


Fig. 9 Statistics for automotive vulnerabilities: by SDLC phase

for the 413 vulnerabilities.⁴ The top 5 (according to their count, i.e., the number of times the CAPEC IDs appeared as a match to a vulnerability) are presented in Table 6. We also analyzed the trend in change of number of vulnerabilities under each CAPEC category. The results for the top 5 are presented in Fig. 10. As evident from the figure, *CAPEC-1: Accessing Functionality Not Properly Constrained by ACLs* and *CAPEC-233: Privilege Escalation* shows an increasing trend across the studied time line. This can be attributed to the growing complexity of permission models in modern CV platforms and a rising reliance on third-party libraries and containerized services, where improper access configuration is a common oversight. In contrast, though *CAPEC-66: SQL Injection* and *CAPEC-100: Overflow Buffers* contribute hugely towards the total number, it show a decreasing trend across years. This could be due to growing adoption of secure coding practices and other tools discussed in Section 6.2. *CAPEC-63: Cross-Site Scripting (XSS)* shows a considerable spike in 2023 followed a drastic drop in numbers in 2024. This pattern implies a possible temporary lapse in input/output sanitization during rapid integration of third-party web components, possibly driven by increased OTA service rollouts. The subsequent drop in 2024 could be the result of improved adoption of secure output encoding libraries and stricter front-end security policies.

Summary: *CAPEC-66, SQL Injection* and *CAPEC-63, Cross-Site Scripting (XSS)* appears to be the most common attack mechanisms for the identified vulnerabilities. This vulnerability only applies to the software “around” the vehicle i.e. the external applications rather than the SoC/Embedded CV software, which again highlights the dominance of external vulnerabilities over the in-vehicle ones over the recent time.

⁴It should be noted that a single vulnerability could be mapped to more than one CAPEC ID.

RQ7: What are some of the most vulnerable open-source automotive software repositories and the underlying vulnerable software libraries used by these repositories?

As a part of our study, we selected 20 open-source repositories that were either automotive software products themselves or related to the latter's development (e.g., build tools). The scanning tools identified **827** vulnerable libraries. Table 7 presents a summary of the scan results. Some of the significant observations from Table 7 are, i) *Tool identifying maximum number of CVEs*: Grype, ii) *Variation in Performance of tools*: Other than a few exceptions, (tramyard/autoexpress, CVE count: 1 given by all tools) the tools perform differently for different repositories, iii) *Hidden vulnerabilities*: For the repository, *PolySync/oscc* none of the tools other than Snyk could identify vulnerabilities, iv) *Most Vulnerable Repository*: grafana/grafana (**5644**), and v) *Highest number of vulnerable Libraries*: erdos-project/pylot (**521**).

Figure 11 presents an overview of some of the most vulnerable software libraries used in developing automotive software in CVs (though these are some of the most popular and generic libraries, they are extensively used in automotive software). The top 5 vulnerable libraries, along with their vulnerability counts turn out to be *tensorflow_gpu*(361), *libmysql-client21*(290), *libhdf5*(84), *openjdk-11-jre*(77), and *qtwebengine5-doc*(74). To evaluate potential “choke points” in the automotive software ecosystem, we examined the 20 open-source repositories analyzed in this study for their most frequently included libraries. The results reveal a high degree of overlap across projects showing *tar*, *ncurses-base*, *bsdutils*, *coreutils*, *gpgv*, *libblkid1*, *libc-bin*, *libc6*, *openssl*, *libgcrypt20* and *libgnutls30* as the most commonly occurring libraries. The presence of multiple security-related libraries (*libgcrypt20*, *libgnutls30*, *gpgv*) among the most commonly shared dependencies is particularly noteworthy. These libraries are responsible for handling cryptographic operations such as TLS handshakes, digital signature verification, and symmetric/asymmetric encryption. A vulnerability in any one of these could propagate across numerous projects, potentially affecting update verification, communication security, or key management systems. For example, a hypothetical zero-day

Table 6 Top CAPEC mechanism of attack for automotive vulnerabilities

Mechanisms of Attack according to CAPEC	Example of mapped CVEs	Count
CAPEC-66:SQL Injection	CVE-2023-1547, CVE-2023-3045, CVE-2023-2046, CVE-2023-23331, CVE-2022-28815, CVE-2022-28813	50
CAPEC-63:Cross-Site Scripting (XSS)	CVE-2023-48837, CVE-2022-39324, CVE-2024-0345, CVE-2024-21550, CVE-2022-43084, CVE-2021-22822	43
CAPEC-233:Privilege Escalation	CVE-2024-8533, CVE-2023-48902, CVE-2022-42431, CVE-2023-32155, CVE-2021-46075, CVE-2022-30495	26
CAPEC-100:Overflow Buffers	CVE-2024-2929, CVE-2024-21912, CVE-2023-33031, CVE-2022-25713, CVE-2022-33282, CVE-2022-33219	24
CAPEC-1:Accessing Functionality Not Properly Constrained by ACLs	CVE-2024-8118, CVE-2024-31214, CVE-2023-2183, CVE-2023-3836, CVE-2022-32019, CVE-2022-38370	20

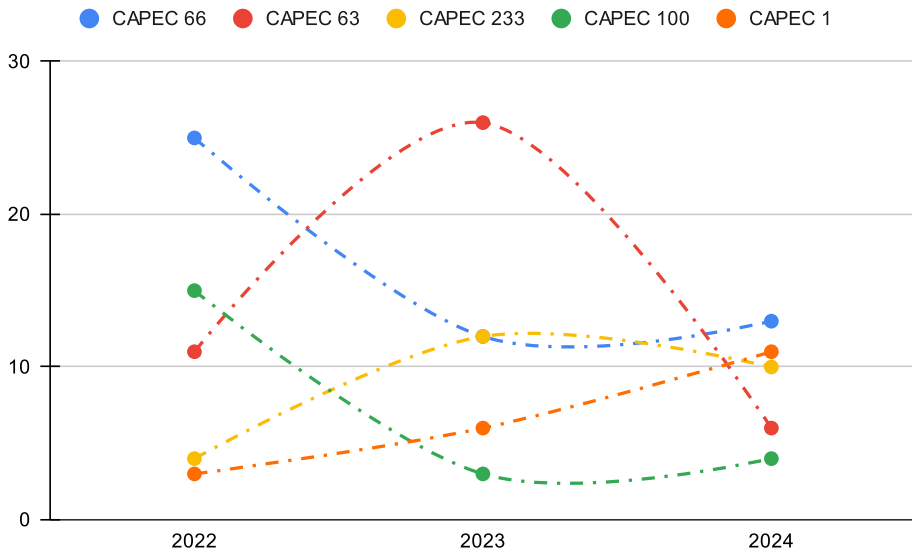


Fig. 10 Vulnerability trend under CAPEC Categories

flaw in `libgnutls30` could expose large portions of the automotive ecosystem to Man-in-the-Middle attacks or firmware tampering, especially in OTA or telematics backends that reuse standard TLS libraries. This categorization emphasizes the need for focused security auditing, patch management, and SBOM-based dependency scanning on these shared components. OEMs and Tier-1 suppliers should prioritize hardening and monitoring these common libraries to reduce systemic risk across vehicle software stacks.

Summary: *grafana/grafana* is the most vulnerable open-source automotive software repository, and *tensorflow_gpu(361)* is the most vulnerable extensively used software library in the automotive industry.

Though the above-mentioned summary highlights *grafana/grafana* and *tensorflow_gpu(361)*, it should be mentioned here that the immense popularity and wide variety of application of these two results into a huge number of contributors (E.g., in the last 1 year *Grafana* had 89 contributors with 137 average commits per week), pull requests, etc., and this might be a reason for their higher ranks in this list. It is evident from the above observations that there is a strong prevalence of vulnerabilities across the automotive software engineering process. The nature and source of these vulnerabilities could be detected to a good extent with the different mappings and scanning tools presented here. Based on these findings, we discuss and make recommendations in the next section.

RQ8: How do the vulnerability severity levels and CWE distributions in automotive software compare to those in adjacent cyber-physical domains such as IoT and Industrial Control Systems (ICS)?

Table 7 Open source automotive software repositories & associated vulnerabilities

Name of Repository	CVE Count: Using Different Vulnerability Scanning Tools (Number of Vulnerable Packages/Libraries detected)				
	Snyk	Trivy	Grype	Docker Scout	Vulert
1. carla-simulator/carla	63(28)	793(136)	1483(155)	334(64)	7(2)
2. apache/airflow	105(28)	223(96)	237(97)	57(32)	2(2)
3. grafana/grafana	5644(123)	26(9)	42(11)	15(6)	24(14)
4. tumgeka/carla-autoware-bridge	0	1633(285)	4108(376)	1068(86)	0
5. carla-simulato/scenario_runner	40(21)	1136 (104)	371(162)	968(53)	1(1)
6. CSS-Electronics/cancloud	540(73)	82(35)	0	0	0
7. gregjohgan/autoecu	119(72)	140(70)	0	0	108(68)
8. Hacksore/bluelinky	8(6)	80(29)	137(30)	58(30)	10(8)
9. jamores/eth-scapy-someip	6(3)	1(1)	1(1)	0	2(1)
10. lgsvl/apollo-5.0	55(28)	10 (6)	142(70)	0	109(69)
11. lgsvl/simulator	62(17)	34(8)	17(5)	0	87(7)
12. openremote/openremote	436(11)	10(8)	11(9)	0	11(9)
13. teslamate-org/teslamate	51(21)	7(7)	74(37)	25(14)	1(1)
14. MaybeShewill-CV/lanenet-lane-detection	11(4)	359(1)	360(1)	0	581(2)
15. tramyardg/autoexpress	1(1)	1(1)	1(1)	0	1(1)
16. erdos-project/pylot	61(11)	216(3)	4056(521)	2846(195)	380(9)
17. PolySync/oscc	94(28)	0	0	0	0
18. firecracker-microvm/firecracker	1(1)	1(1)	1(1)	0	1(1)
19. bazelbuild/bazel-buildfarm	305(61)	63(25)	21(15)	5(2)	1(1)
20. emqx/emqx	1(1)	0	101(38)	27(13)	51(15)
Total	7603	4815	11,163	5403	1377

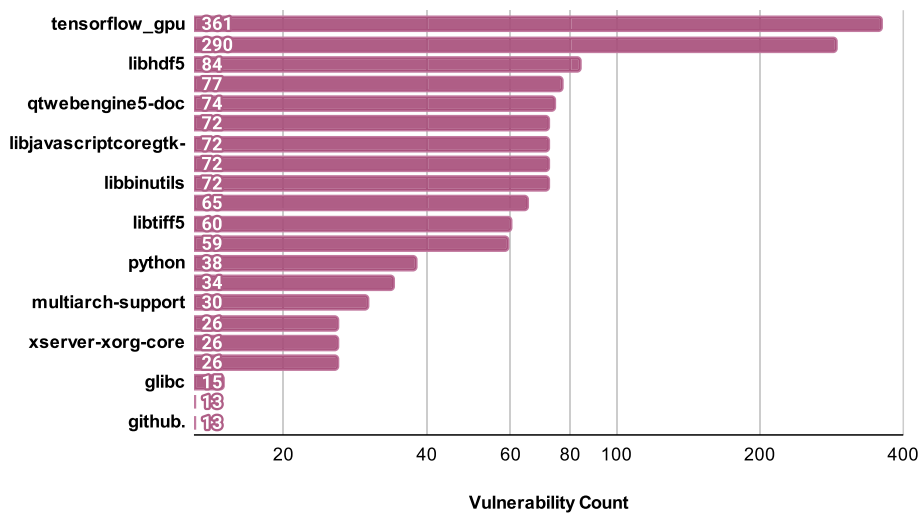
**Fig. 11** Topmost vulnerable software libraries used in automotive industry

Figure 12 presents the CWE distribution across the concerned domains. CWE 20, Improper input validation remains a foundational flaw in connected systems, but it is more prevalent in general-purpose IoT than in safety-critical automotive systems, where stricter validation frameworks may be adopted. In the Automotive domain, CWE-89 (SQL Injection) shows the highest dominance (34.2 %), followed by CWE-79 (XSS) with 24.4%. These two web-oriented vulnerabilities are less common in ICS/IoT, indicating that automotive systems are increasingly dependent on web-connected services, infotainment, and OTA update mechanisms, often reusing general-purpose web stacks. CWE-125 (Out-of-Bounds Read) and CWE-787 (Out-of-Bounds Write) together results into 23.6% of the overall CWEs, highlighting that memory safety issues persist, especially in embedded software stacks. On a contrary, classic memory corruption issues like CWE-787 (Out-of-Bounds Write), CWE-125 (Out-of-Bounds Read), CWE-416 (Use After Free) and CWE-121 (Stack-based Buffer Overflow) are highly prevalent in ICS due to legacy C/C++ codebases, limited update cycles, and real-time system constraints. Although IoT shares memory safety risks with ICS, it has unique issues tied to minimal firmware protections evident from the presence of CWE-190 (Integer Overflow or Wraparound), CWE-284 (Improper Access Control), etc. The 100% stacked bar chart in Fig. 13 presents the proportional distribution of CVE severities (Low, Moderate, High, and Critical) across three major cyber-physical domains: IoT, Industrial Control Systems (ICS), and Automotive software. High severity vulnerabilities dominate all three domains. ICS exhibits the highest proportion of High-severity CVEs (~ 50%), which is notably higher than in IoT and Automotive. Critical severity vulnerabilities are also consistently present across domains. It represents ~15–16 % of each, indicating a sustained presence of high-impact flaws, particularly relevant in safety-critical systems. Moderate severity issues are more common in IoT (~35%) than in ICS and Automotive. This suggests that IoT has a broader mix of vulnerabilities. Low severity vulnerabilities make up the smallest share across all domains. The proportion is slightly higher in IoT, which may reflect broader CVE reporting in consumer-grade products.

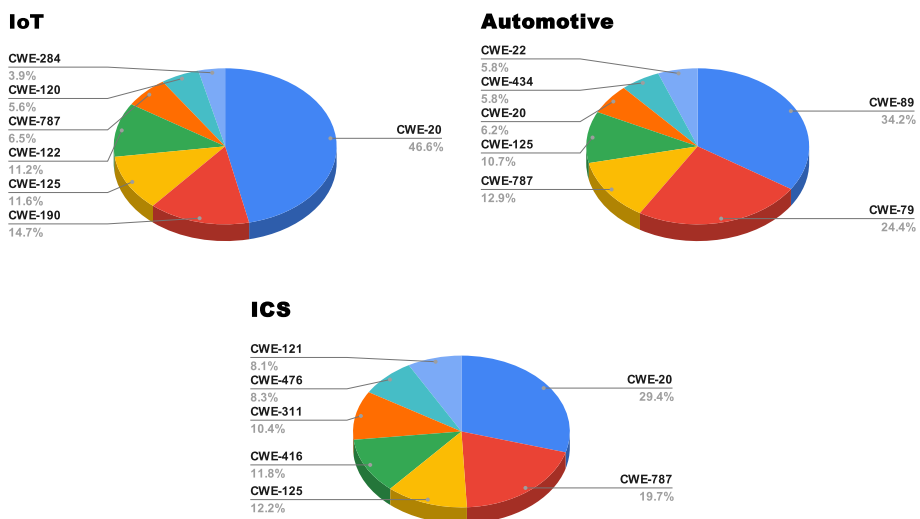


Fig. 12 Top 7 most frequently occurring CWEs in IoT, Automotive & ICS domain

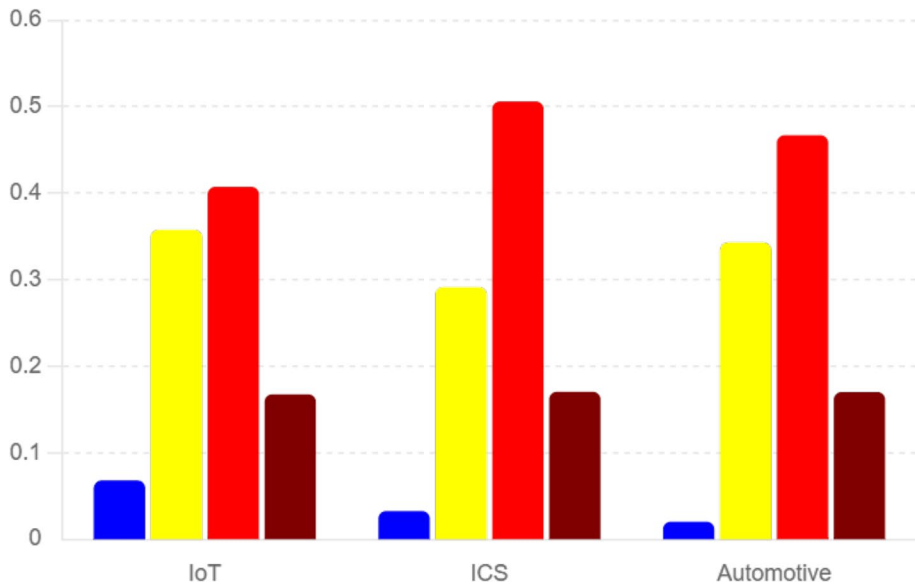


Fig. 13 Relative proportion of vulnerability severity level across the IoT, ICS, and Automotive domains

The comparative analysis in RQ8 offers valuable insights for practitioners in the automotive domain. The high prevalence of CWE-89 (SQL Injection) and CWE-79 (Cross-Site Scripting) in automotive CVEs, unlike ICS or IoT, indicates an increasing integration of web technologies in vehicle systems, such as infotainment units, telematics services, and OTA update portals. This shift requires software engineers to adopt secure web engineering practices traditionally used in enterprise development, including rigorous input validation, parameterized queries, and browser-side content security policies.

Moreover, the comparative severity distribution (Fig. 13) shows that while automotive systems are not disproportionately affected by CRITICAL vulnerabilities, the consistently high proportion of HIGH severity flaws signals the potential for large-scale impact. Given the complex integration of vehicle components and backend systems, even moderately severe issues in externally-facing APIs (e.g., infotainment or mobile app backends) can cascade into safety risks. Security engineers should therefore prioritize threat modeling for web-connected modules and adopt cross-domain mitigation strategies such as software composition analysis (SCA), static code analysis, and interface fuzzing.

In contrast, the relatively lower occurrence of classic memory corruption CWEs in automotive (e.g., CWE-121, CWE-416) compared to ICS suggests that the use of memory-safe languages or tighter build-time checks may be gaining traction in automotive firmware. However, engineers must remain vigilant especially in SoC-based ECUs where performance-critical C/C++ codebases still dominate.

Overall, these findings reinforce the need for a hybrid security mindset in automotive software development, one that combines web security best practices with embedded system hardening and continuous vulnerability monitoring across the SDLC.

Summary: CWE-20 is a widely prevalent issue in all the three domains (IoT, Automotive & ICS) with more occurrences seen in *IoT*. The *Automotive* domain is characterized by more web-oriented flaws such as CWE-89 and CWE-79. *ICS* is affected by classic memory corruption flaws (CWE-787, CWE-125, CWE-416, CWE-121). High-severity vulnerabilities dominate all domains, with ICS showing the largest proportion (~50%).

6 Result implications and recommendations for secure automotive software development

Our study results indicate that software engineers in the automotive industry should use a more holistic approach to security, focusing on their development and full supply chains. In this section, we highlight the importance and probable reasons for some of the significant observations from our results. Later we provide security recommendations i.e. possible actions that can be taken by security practitioners for avoiding the issues. Although it is possible that an SE team already applies one or more of these recommendations, we highlight them here, to reinforce their importance, and to encourage those responsible for SE for CV to follow these recommendations and security controls.

6.1 Result implication & importance

We enumerate the following implications:

- *I1*: It is evident from the results that the third quarter of the year 2023 has experienced the largest number of vulnerabilities with **HIGH** severity score. On studying these vulnerabilities in detail, we found a common problem. Most of these were software associated with SoC/Embedded System and ECUs, and the common problem was either *Overflow Buffers* or *Memory corruptions*.
- *I2*: The second quarter of the year 2022 has experienced the largest number of vulnerabilities with **CRITICAL** severity score. The detailed study revealed that almost all these vulnerabilities were associated with an External Web Application that dealt with car and occupant data, and the common attack mechanism for most of them was *SQL Injection*.
- *I3*: The dominance of vulnerabilities in *external applications* and *telematics systems* has critical implications for developers. While interacting with user-facing features, these components are more often exposed to public networks (e.g., cloud-based monitoring or OTA update infrastructure). The huge number of web-based vulnerabilities like SQL injection and cross-site scripting identified in these applications automatically increases the chance of network based attacks. Infotainment and telematics systems, while traditionally considered non-critical, often act as bridges into more sensitive in-vehicle networks. Historical incidents (e.g., the Jeep Cherokee hack) have shown that these systems, if not properly isolated, can provide attackers with access to ECUs or CAN buses. Our results show that vulnerabilities in these areas remain common and are mostly inherited from third party software usage. Overall, the observed trend shifting from in-vehicle to external components, reflects the expanding attack surface of modern

CVs. Addressing this requires not only secure coding practices but also architectural decisions and supply chain visibility.

- *I4: Grafana* is seen to have produced the most vulnerable products in the last 3 years. Not only did the search result in many vulnerabilities from NVD, but the different vulnerability scanning tools reported the same information. The vulnerabilities in *Grafana* are further studied in the context of automotive software (application in CVs) and found that *Privilege Escalation* is a common attack mechanism here.
- *I5: A significant growth in the vulnerability landscape of External Entities* in the CV ecosystem is already reported in the results. Further investigation shows that most of these vulnerabilities can be associated with *Cross-Site Scripting (XSS)* and *SQL Injection* attack mechanism.
- *I6: The study also reports an increase in the OEM vulnerability percentage over the years. Upon further exploration, we found that the common attack category that can be associated with OEM-vulnerabilities is Inject Unexpected Items* (e.g., *Code Inclusion, Local Execution of Code, Hardware Fault Injection*, etc.)
- *I7: The scanning tools reported tensorflow_gpu to be the most vulnerable library. Investigating further, it was revealed that almost 70% of these vulnerabilities are found to be triggering DoS attacks* (one of the most common security attacks in CV ecosystem) and are caused by the Reachable Assertion problems (cwe.mitre.org, [nd](#)).

6.2 Recommendations

The implications discussed in Section 6.1 highlight recurring vulnerability patterns across different CV ecosystem components. In this section, we provide developer-centric recommendations mapped to these implications and SDLC phases, to support proactive security efforts in both in-house development and supply chain integration. We indicate the associated SDLC phase for each recommendation in parentheses, e.g., (Design Phase). It should be noted that these recommendations are more like widely known security controls (taken from MITRE ([attack.mitre.org](#), [n.d.](#)), OWASP ([owasp.org](#), [n.d.](#)), MISRA C (Bagnara et al., 2018), CERT C (Standard, 2016), etc.), but we try to present these in the light of automotive software vulnerabilities so that the SE team knows the correct time, place, and context to implement such security controls.

1. *R1: The following security measures may help to reduce the problems identified in I1.*

- Programming Languages with their memory management techniques, in-built overflow protection mechanisms (Rust, Go) (Hedgren, 2024), etc. can be selected for developing the ECU/SoC software (*Software Requirement Specification Phase*). Notably, Volkswagen AG has publicly explored the use of Rust for embedded safety-critical functions to eliminate entire classes of memory safety bugs.⁵
- Libraries like Safe C String Library (SafeStr) (Microsoft, [n.d.](#)) and the Strsafe.h library from Microsoft may be used to avoid these issues for string functions (*Design Phase*).
- Avoid string copy operations as much as possible. Use functions like `strlen()`, `strcpy_s()`, `getenv()`, etc., and pre-allocate memory at the beginning to ensure that

⁵ See: <https://github.com/VolkswagenGroup/rust-in-automotive>

storage for strings has sufficient space for character data and the null terminator (*Development Phase*).

- Automatic buffer overflow detection mechanisms offered by some compilers and compiler extensions like the Microsoft Visual Studio/GS flag, Fedora/Red Hat FORTIFY_SOURCE GCC flag, StackGuard, and ProPolice (cwe.mitre.org, [n.d.](#)) (containing canary-based detection and range/index checking) can be used (*Development Phase*).
- The ECU and SoC software can be executed using extensions (Address Space Layout Randomization (ASLR) (Howard, [n.d.](#)) and Position-Independent Executable (PIE) (Murphy, [n.d.](#)) that randomizes the positions of a program's executable and libraries in memory, thereby making the addresses unpredictable. Previously, in case of smaller ECUs randomizing was not very feasible but in today's complex, large and centralized ECUs it can be realized effectively. This can also be a reason of larger number of Embedded/SoC and ECU related vulnerabilities in the span of 2009–2016 (Gülsever, 2019) as compared to recent times (*Development Phase*).
- The tools like Valgrind and Mempatrol (mas.owasp.org, [n.d.](#)) may be used for analyzing memory usage and calls to avoid memory corruption problems (*Development Phase*).
- In general to address the increasing trend of HIGH and CRITICAL vulnerabilities, security teams should adopt targeted measures, such as: (i) incorporating threat modeling and risk assessment early in the design phase to identify attack surfaces most likely to lead to severe impacts, (ii) enforcing rigorous secure coding practices and automated static analysis to prevent memory safety and injection flaws, (iii) prioritizing vulnerability scanning and patch management for components exposed to external interfaces (e.g., infotainment, telematics, OTA update modules), and (iv) leveraging runtime protection mechanisms such as intrusion detection, anomaly detection, and isolation of critical functions from external-facing subsystems. Taken together, these practices can help mitigate the risks associated with the rising number of severe vulnerabilities and ensure that mitigation strategies are proportionate to their potential impact on safety and security.

2. R2: The problem identified in I2 may be minimized by using the following recommendations

- Frameworks and APIs like Hibernate and Enterprise Java Beans may reduce SQL Injection when utilized properly (*Design Phase*).
- Using prepared statements (with parameterized queries), properly constructed stored procedures, and Allow-list Input Validation may reduce the chances of SQL Injection (*Design & Development Phase*).
- Some existing database features like Oracle DBMS_ASSERT package, mysql_real_escape_string() API function, etc., may be used to reduce the chances of SQL Injection (*Development Phase*).
- An error message depicting too much information may reveal the structure of an SQL query, thereby increasing the chances of SQL Injection. Therefore, these messages should contain limited information for intended users only. Additionally, if detailed error messages are stored in logs, proper access control for such log files should be ensured, and sensitive information like passwords, Vehicle Information Numbers (VIN), etc., should never be stored in clear text (*Software Requirement Specification, Development Phase*).

- In 2022, cybersecurity researchers from Yuga Labs (Threat Actors, [n.d.](#)) disclosed a vulnerability in a vehicle telematics portal that allowed unauthenticated SQL injection via a query string parameter containing the VIN. The flaw enabled attackers to extract sensitive data (driver behavior, trip logs) from backend databases. The issue was traced to unsafe string concatenation in the API controller. It was resolved by adopting parameterized queries and applying allow-list filtering on the input fields. This reinforces the need for secure query logic in externally accessible automotive services. Tools such as sqlmap (for detecting injectable parameters), ZAP Proxy (for fuzzing HTTP input), and Snyk or Trivy (for scanning backend dependencies vulnerable to SQL Injection) can be integrated into the CI/CD pipeline to proactively catch injection flaws during development and integration phases.

3. *R3*: The following recommendations are provided for the issue discussed in *I3*

- Applying rigorous *input validation and authentication mechanisms* for all external-facing interfaces. (*Design Phase*)
- Prioritizing *threat modeling and fuzz testing* in telematics, and V2X modules. (*Software Requirement Specification, Design Phase*)
- Ensuring *network isolation* between telematics/infotainment and critical in-vehicle functions (e.g., ECUs, sensors). (*Design Phase*)
- Performing *SBOM-based security analysis* for all third-party packages integrated into these layers. (*Development Phase*).

4. *R4*: The following recommendations are provided for the issue discussed in *I4*

- Using fine-grained access control, creating separate user accounts with the least privileges for each task may reduce the chances of privilege escalation problems (*Design Phase*).
- Functionalities requiring enhanced privileges, such as access to operating system resources, should be centralized, and the corresponding code should be isolated from the rest of the system. The communication channels interacting with this code should also be protected (*Design Phase*).
- The request for raising privileges should be served as late as possible, and dropping them should be done immediately after the requirement is fulfilled (*Development Phase*).
- Using rigorous input validation in a privileged code may reduce the chances of attacks (*Development Phase*).
- Following Tesla's OTA architecture that exhibits strong privilege separation and update integrity by enforcing cryptographic signature verification for each firmware module, isolating critical subsystems through process-level sandboxing, and applying access control frameworks (e.g., AppArmor, seccomp) within its infotainment and gateway environments. These design choices compartmentalize software components, limit lateral movement between subsystems, and mitigate privilege escalation risks during remote update operations (Nie et al., [2017](#); Berard & Dehors, [2023](#)).

5. *R5*: Here we find some recommendations for avoiding the cross-site scripting (XSS) problems discussed in *I5*.

- Parts of the external software (e.g., Data Management and Reporting Modules) communicating directly with the in-vehicle system should be segregated and prevented from having content executable by a web browser. In short, data and code should be separated as much as possible (*Design Phase*).
- Encoding libraries or frameworks like Microsoft's Anti-XSS library, the OWASP ESAPI Encoding module (owasp.org, [n.d.](#)), Apache Wicket, etc. contain methods for output encoding which may reduce the risk of XSS attacks. (*Software Requirement Specification, Design Phase*).
- Validating all parts of the HTTP request, e.g., the VIN (Vehicle Identification Number) should be checked for its length, starting alphabet, the full scope of acceptable alphabets (missing or extra alphabets), syntax, etc. on both client and server side. This should be followed by specific output encoding mechanisms (ISO-8859-1, UTF-7, and UTF-8, etc.) as well (*Development Phase*).
- It is a good practice to maintain a mapping from a set of fixed inputs (such as numeric IDs) to the actual filenames or URLs and ignore all other inputs, e.g., in Parking Access Control modules equipped with ticket dispensers, card readers, etc. such a mapping could be maintained. Again, this is possible only if a limited number of acceptable entities (*Software Requirement Specification, Design Phase*).
- In 2019, Tesla remediated a blind XSS in its "Name Your Vehicle" functionality within 12 hours after disclosure (Leyden, 2019), by pushing a hotfix to its internal dashboards. Based on the issue, a secure fix likely involved output encoding (escaping vehicle names before insertion), restricting allowable characters or patterns, and introducing stronger sanitization. As a best practice, automotive software systems should employ template-based auto-escaping, content security policy (CSP) headers, strict input validation, and HTML/JS encoding to prevent such vulnerabilities in external-to-internal interfaces.

6. *R6*: A few recommendations for avoiding the injection of unexpected items into the OEM software, as discussed in *I6*, are as follows:

- Running a piece of sensitive code in a jail or sandbox environment (e.g., Unix chroot jail, AppArmor, SELinux, etc.) (cwe.mitre.org, [n.d.](#)) restricts boundaries between the process and the operating system. This kind of restriction may reduce chances of Local Code Execution, Code Inclusion, etc. (*Design & Development Phase*).
- Files like a library, include, and utility should be stored outside the web document root or in a separate directory. Additionally, an access control mechanism for the web server may be used to prevent attackers from directly requesting them (*Design Phase*).
- It is important to know all potential mediums through which unreliable inputs can enter the software. E.g., cookies, environment variables, reverse DNS lookups, query results, request headers, URL components, e-mail, files, inputs obtained via API calls, etc. All such entities should be validated properly (*Design Phase*).
- In the Jeep Cherokee Uconnect hack (2015) (Miller & Valasek, 2015), attackers remotely accessed an unauthenticated debug service over the cellular network, exploited a buffer overflow to gain root access on the IVI system, and injected CAN messages by pivoting into the vehicle's internal network. The incident revealed the

need for strict service authentication, privilege separation, and network segmentation between infotainment and safety-critical domains. To defend against similar attack paths, OEMs can implement runtime protection tools such as Runtime Application Self-Protection (RASP), which dynamically inspects input and control flow within APIs to prevent injection attacks. Additionally, Linux-based head units or development environments can benefit from kernel-level auditing tools like Auditd, which logs privileged operations and system calls—helpful for detecting unauthorized access or debugging security incidents (Williams, 2020; Red Hat, n.d.).

7. *R7*: In context of *I7* it is recommended to use validated inputs and make sensitive operation non-reachable by directly user-controlled data. TensorFlow GPU is widely used in perception modules of open-source autonomous driving stacks like Apollo and Autoware. While not specific to a public AV incident, this risk highlights the need for input validation and pipeline isolation in ML-powered automotive software.

This study was conducted to offer a holistic view of software security vulnerabilities across the automotive supply chain. To deepen the practical relevance of this work, we refer to a companion study (Basu et al., 2025) where we analyzed the time taken by vendors to mitigate 180 real-world vulnerabilities in automotive System-On-Chip (SoC) software components. Our findings showed that vulnerabilities associated with CWE-416 (Use After Free) and CWE-126 (Buffer Over-read) accounted for nearly 50% of the total remediation time, with some instances taking up to one year to be patched. Moreover, modules such as Wi-Fi and Inter-Process Communication (IPC) exhibited prolonged patch delays and high variation, likely due to architectural complexity and integration dependencies. These insights underscore the importance of tracking mitigation latency in supply chain components and support our recommendation that future work should incorporate patch timelines and maintenance responsiveness as key indicators for risk-based vulnerability prioritization in connected vehicles. Though currently the analysis is done on a subset i.e., automotive SoC software vulnerabilities, in the near future, we plan to investigate all the identified vulnerabilities in terms of the time taken to fix them and the cost/complexity of patching the vulnerability, etc. to get the broader picture and prioritize the security controls accordingly.

To assess the exploitability of the vulnerabilities in our dataset, we cross-referenced them with CISA's Known Exploited Vulnerabilities (KEV) catalog and ExploitDB, which lists CVEs confirmed to be actively exploited in real life. We found that only 17 automotive-relevant CVEs were added to the KEV catalog between 2022 and 2024 and ExploitDB appeared considerably obsolete (at least in context of automotive vulnerabilities). Out of these 17, 14 (82.4%) were classified as HIGH or CRITICAL severity, reinforcing the notion that severity scores generally align with real-world exploitation potential. However, we did not find most of the vulnerabilities identified in the study in KEV/ExploitDB, suggesting that a large portion of vulnerabilities, while potentially serious, have not yet been exploited in publicly known incidents. This emphasizes the need for improved exploitability tagging and encourages further research into attack observability in vehicle systems. While CVSS provides useful prioritization, understanding which vulnerabilities are actually targeted in real world remains critical. Automotive OEMs and Tier-1s could benefit from incorporating exploit feeds like CISA KEV and ExploitDB into their patch management workflows to improve risk-based remediation.

7 Threats to validity

There may be vulnerabilities, potential attack vectors, and other problems that go undetected using the approach of this paper. When considering the validity of our study, we used the framework by Wohlin et al. (2012) and Staron (2020).

In the *construct validity* category, the main risk is using the CVSS score based on CVSS Version 3.0. Although we acknowledge that the results section could have been different if we had used version 2.0, 3.1, or 4.0, this is a first study, and therefore, we chose this framework as a good start.

Finally, the last *construct validity* threat is using tools for scanning the vulnerabilities. Since the sample of these tools is small, the selection was partly biased from some industry interaction, and the results differ from tool-to-tool, we envision more studies in this area to find the most optimal suite of tools for this type of study.

In the *conclusion validity* area, we see our main risk as our manual analysis. Although we considered using automated tools, e.g., LLMs, we chose to do it manually to be able to validate our results or discuss them if needed.

For the *external validity*, we consider the main risk to be the sample, the 20 GitHub repositories. Following the method depicted in Section 4.1 we procured a categorized list of automotive-related GitHub projects including sub-domains like autonomous driving (e.g., Apollo, Autoware), diagnostics (e.g., CSS-Electronics), simulation (e.g., CARLA), OTA telemetry, or V2X communication. Although some of the selected repositories (e.g., *Grafana*, *Bazel*) are general-purpose or infrastructure-level tools, they are used extensively in automotive software stacks, particularly for telemetry dashboards, perception modules, and CI/CD pipelines in CV development. Their inclusion reflects a realistic view of modern automotive engineering, which increasingly integrates general open-source components into in-vehicle and back-end systems. Still, we acknowledge this introduces a risk of over-generalization. Future work should aim to expand the sample to include OEM-proprietary stacks (when available), domain-specific SDKs, and more diverse Tier-1 supplier codebases. We also recognize that the definition of “automotive software” may evolve and encourage future researchers to develop refined classification guidelines that distinguish core automotive code from general-purpose dependencies reused in the automotive domain.

8 Conclusion and future work

Connected vehicles are highly complex cyber-physical systems, especially from a security perspective. This study demonstrates that incorporating a more holistic and software-aware approach enables a deeper analysis of CV vulnerabilities, moving beyond hardware-focused threat modeling toward actionable insights for the automotive software stack. By analyzing attack surfaces and vulnerabilities across the supply chain, our approach helps software engineers reason about security risks throughout the SDLC.

We analyzed 413 new vulnerabilities across widely used automotive software packages, with nearly 45% classified as HIGH severity. Our scan results exposed a dense overlap of vulnerable libraries and revealed that vulnerabilities increasingly originate not in embedded firmware, but in external-facing applications, such as telematics dashboards, OTA backends, and perception stacks. Furthermore, the CAPEC mapping identified consistent

attack mechanisms (e.g., SQL injection, XSS, memory corruption) tied to specific software domains and lifecycle phases.

This analysis helped us understand security requirements from different dimensions. For example, the vulnerability trend shift from SoC/Embedded vehicle software towards external applications, i.e., software “around” the vehicle is something software developers should be aware of. Additionally we highlight key areas where automotive security engineering must evolve: embracing input validation and output sanitization in web APIs, managing transitive risk in shared libraries, and improving responsiveness in patching critical software modules. We believe our method, mapping and analysis offer practical tools for improving secure software development practices in the connected vehicle domain.

Based on our findings, several future research directions and open questions emerge:

- **Automated SDLC-phase inference:** Our manual mapping of CVEs to software life-cycle phases revealed key security blind spots in early development stages (e.g., architecture and requirement specification). However, this process could be time-consuming and subjective but the question emerges as - *Can machine learning models, particularly LLMs trained on CVE descriptions and CWE taxonomies, accurately infer the SDLC phase of origin for a vulnerability?* Future work could explore the feasibility and precision of such models, with implications for secure design feedback loops and code review prioritization.
- **Dependency risk modeling:** Our analysis found that a small set of libraries (e.g., `lib-gcrypt20`, `libgnutls30`, `libc6`) are reused across multiple projects. This raises systemic risk concerns. Future work can explore the answer to the question- *What is the “blast radius” of a vulnerability in a widely shared automotive dependency, and how can SBOMs be used to trace and contain its spread across ecosystems?* Research could explore dependency graph analytics and diffusion modeling techniques tailored for connected vehicle stacks.
- **Security of ML-enabled automotive components:** As perception and decision-making are increasingly ML-driven, new attack vectors arise (e.g., malformed LiDAR, poisoned datasets, inference-time exploits). A potential research direction could address - *What are the most common vulnerability patterns in deployed ML pipelines within AV and ADAS systems?* This opens a line of research into formal threat modeling, runtime monitoring, and ML-specific fuzzing tailored to sensor-driven automotive environments.
- **Industry-facing DevSecOps integration:** While our results suggest actionable practices (e.g., secure coding for web-facing telematics modules), industry integration remains slow. The direct question that arises is - *How can vulnerability taxonomies, SDLC mappings, and SBOM insights be operationalized in automotive CI/CD pipelines?* This includes research into tooling, metrics, and policy frameworks that align with ISO/SAE 21434 and UNECE WP.29 mandates (Costantino et al., 2022).
- **Distinguishing CV-Critical Libraries from General-Purpose Dependencies:** A valuable extension of this work would be to distinguish between general-purpose infrastructure dependencies and “CV-critical” libraries used in safety or control-related components. This would require detailed inspection of vulnerability descriptions, dependency usage context, and domain-specific integration roles, a task that combines security and automotive engineering expertise.

A: Appendix

Table 8 List of keywords with categorization

Category	Keyword
T-CV	AIRBAG
	APPLIED EV
	AUTO
	AUTOMOBILE
	AUTOMOTIVE
	BLUETOOTH
	CAR
	CARLINK
	CARPLAY
	CONTROLLER AREA NETWORK
	ECU
	ELECTRIC VEHICLE
	FLEXRAY
	FUEL
	GPS
	INFOTAINMENT
	LIDAR
	LIGHTNING SYSTEMS
	MOBILE APP
	OBD
	OTA
	PARKING
	PASSENGER
	RADAR
	SENSOR
	TELEMATICS
	TIRE
	TPMS
	VEHICLE
T-OEM	ACURA
	ADAS
	AIRBIQUITY
	ALFA ROMEO
	ALPINE
	ANDROID AUTO
	ARRIVAL
	ASPARK
	ASTON MARTIN
	AUDI
	AUTOLIV
	BAIC
	BMW
	BOLLINGER
	CADILLAC
	CANOO

Table 8 (continued)

Category	Keyword
	CHANGAN
	CHEVROLET
	CHRYSLER
	CITROËN
	CLEVER
	CRUISE
	DAIMLER
	DELPHI
	DENSO
	DODGE
	DONGFENG
	DRIVESYNC
	EDISON FUTURE
	ENFORM
	EV WEST
	EVERGRANDE EV
	FACTION
	FARADAY FUTURE
	FERRARI
	FIAT
	FISKER
	FORD
	FREESCALE
	GARAGE ITALIA
	GEELY
	GENERAL MOTORS
	GMC
	GOUPIL
	GRAFANA
	GREAT
	GUNSEL ELECTRIC VEHICLES
	GWM GROUP
	HONDA
	HUMBLE
	HYUNDAI
	INDI
	INFINITI
	JAGUAR
	KIA
	KOENIGSEGG
	LAMBORGHINI
	LAND ROVER
	LEMUR
	LEXUS
	LIGHTYEAR SOLAR
	LUCID MOTORS
	LUNAZ

Table 8 (continued)

Category	Keyword
	MASERATI
	MAZDA
	MBRACE
	MCLAREN
	MERCEDES
	MITSUBISHI
	MOBILEYE
	MOMENT MOTORS
	NAEXT MOTORS
	NIKOLA MOTORS
	NIO
	NISSAN
	OLA ELECTRICS
	ONTO
	OPEL
	PAGANI
	PANASONIC
	PEUGOT
	PIONEER
	PORSCHE
	PROTERRA
	PSA
	QNX
	QUANTRON
	RENAULT
	RENESAS
	RIMAC AUTOMOBILI
	RIVIAN
	ROCKWELL AUTOMATION
	ROLLS ROYCE
	SAAB
	SAFETY CONNECT
	SAIC
	SKODA
	SONO MOTORS
	SONY MOBILITY
	SOOORYA
	SSANGYONG
	STEALTH EV
	SUBARU
	SUNNY
	SUZUKI
	TAKATA
	TATA MOTORS
	TESLA
	TEXAS INSTRUMENTS
	THUNDER POWER

Table 8 (continued)

Category	Keyword
T-GH	TOGG
	TOTEM AUTOMOBILI
	TOYOTA
	ULTRASONIC
	UNITI
	VENUCIA
	VEONEER
	VINGROUP
	VOLKSWAGEN
	VOLVO
	VOYAH
	WALL MOST
	WM MOTORS
	WORKHORSE
	XILINX
	XPENG EV
	ZEDRIV EV
	ZELECTRIC
	ZERO LABS
	ZOTYE
	ZUBIE
	AIRFLOW
	AIRSIM
	APOLLO
	ANDROBD
	ARDUINOBD
	AUTOAS
	AUTOWARE
	CARLA
	DDT4ALL
	LANENET-LANE-DETECTION
	MULTITARGET-TRACKER
	UDACITY

Author Contributions S.B and M.S developed the main concept of the paper. S.B wrote the manuscript. M.S, J.H, M.A and C.B reviewed the paper. M.S, J.H, M.A and C.B helped to fine tune and improvise the method and presentation. M.S, J.H, M.A and C.B cross-checked the research data. S.B conducted the implementation.

Funding Open access funding provided by University of Gothenburg. This paper has been partially financed by Software Center, www.software-center.se and SFO Transport/AoA Transport at the University of Gothenburg and Chalmers University of Technology.

Data Availability All the data and mapping relevant to the paper are made available at https://github.com/SriAbir/CV_Vulnerability_22-24.git

Declarations

Competing interests The authors declare no competing interests.

Open Access This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit <http://creativecommons.org/licenses/by/4.0/>.

References

- Abdelkader, G., Elgazzar, K., & Khamis, A. (2021). Connected vehicles: Technology review, state of the art, challenges and opportunities. *Sensors*, 21(22), 7712.
- Assal, H., & Chiasson, S. (2018). Security in the software development lifecycle. In *14th symposium on usable privacy and security (SOUPS 2018)*, (pp. 281–296).
- Bagnara, R., Bagnara, A., & Hill, P. M. (2018). The MISRA C coding standard and its role in the development and analysis of safety-and security-critical embedded software. In *International Static Analysis Symposium*, (pp. 5–23). Cham: Springer International Publishing.
- Basu, S., Haraldsson, B., Staron, M., Berger, C., Horkoff, J., & Almgren, M. (2025). An AUTOSAR-aligned architectural study of vulnerabilities in automotive SoC software. arXiv preprint [arXiv:2510.07941](https://arxiv.org/abs/2510.07941)
- Berard, D., & Dehors, V. (2023). Security of connected vehicles.
- Costantino, G., De Vincenzi, M., & Matteucci, I. (2022). A comparative analysis of unece wp. 29 r155 and ISO/SAE 21434. In *2022 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)*, (pp. 340–347). IEEE.
- Costantino, G., De Vincenzi, M., & Matteucci, I. (2022). A comparative analysis of unece wp. 29 r155 and ISO/SAE 21434. In *2022 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)*, (pp. 340–347). IEEE.
- CVE-2023-28901 (n.d.). Trip Data Disclosure from Backend. <https://asrg.io/security-advisories/cve-2023-28901-trip-data-disclosure-from-backend/>
- CVE-2023-3121 (n.d.). Detail. <https://nvd.nist.gov/vuln/detail/CVE-2023-3121>. Last Accessed 16 May 2024.
- CWE-798 (n.d.). Use of Hard-coded Credentials. <https://cwe.mitre.org/data/definitions/798.html>. Last Accessed 16 July 2024.
- CWE-918 (n.d.). Server-Side Request Forgery (SSRF). <https://cwe.mitre.org/data/definitions/918.html>. Last Accessed 16 July 2024.
- Darkreading (n.d.). Volkswagen Breach Exposes Data of 800K EV Customers. <https://www.darkreading.com/cyberattacks-data-breaches/volkswagen-breach-exposes-data-of-800k-customers>
- Docker Inc. (n.d.). Docker Scout Documentation. <https://docs.docker.com/scout/>. Last Accessed 25/08/09.
- Dreyer, J. T. (2023). From talking past each other to barely talking. *Comparative Connections: A Triannual E-Journal on East Asian Bilateral Relations*, (2), 25.
- Garcia, J., Feng, Y., Shen, J., Almanee, S., Xia, Y., & Chen, A. Q. A. (2020). A comprehensive study of autonomous vehicle bugs. In *Proceedings of the ACM/IEEE 42nd international conference on software engineering*, (pp. 385–396).
- Global Automotive Cyber Security Report (2024) Upstream
- Grype (n.d.). <https://www.cisa.gov/resources-tools/services/grype>. Last Accessed 24/10/29.
- Gülsever, M. (2019). A Study on Vulnerabilities in Connected Cars.
- Hedgren, K. D. (2024). Designing Secure Applications for the Internet of Vehicles. Master's Thesis, Chalmers University of Technology University OF Gothenbur.
- Howard, M. (n.d.). Address Space Layout Randomization in Windows Vista. https://learn.microsoft.com/en-us/archive/blogs/michael_howard/address-space-layout-randomization-in-windows-vista. Last Accessed: 2023/04/07.
- <https://attack.mitre.org/>. Last Accessed 2024/11/13.
- https://cwe.mitre.org/data/definitions/120.html#REF-56_120. Last Accessed 2024/10/21.
- <https://cwe.mitre.org/data/definitions/617.html>. Last Accessed 2024/10/08.
- <https://cwe.mitre.org/data/definitions/829.html>. Last Accessed 2024/10/08.
- <https://mas.owasp.org/MASTG/tests/android/MASVS-CODE/MASTG-TEST-0043/#static-analysis>. Last Accessed 2024/06/24.

- <https://owasp.org/www-project-enterprise-security-api/>. Last Accessed 2024/06/24.
- <https://owasp.org/www-project-secure-coding-practices-quick-reference-guide/>. Last Accessed 2024/10/10.
- <https://portswigger.net/daily-swig/automotive>. Last Accessed 2024/10/26.
- Iwasaki, M. (2024). Fortifying the weakest link: cybersecurity coordination in supply chains and competition rules. *International Cybersecurity Law Review*, 5(1), 21–29.
- Kim, S., Park, K. J., & Lu, C. (2022). A survey on network security for cyber-physical systems: From threats to resilient design. *IEEE Communications Surveys & Tutorials*, 24(3), 1534–1573.
- Kochanthara, S., Dajsuren, Y., Cleophas, L., & van den Brand, M. (2022). Painting the landscape of automotive software in GitHub. In *Proceedings of the 19th International Conference on Mining Software Repositories*, (pp. 215–226).
- Leyden, J. (2019). Tesla security bug uncovered after minor accident. The Daily Swig. <https://portswigger.net/daily-swig/tesla-security-bug-uncovered-after-minor-accident>
- Microsoft (n.d.). Using the Strsafe.h Functions. <https://learn.microsoft.com/en-us/windows/win32/menurc/strsafe-ovw?redirectedfrom=MSDN>. Last Accessed 2023/04/07.
- Miller, C., & Valasek, C. (2015). *Remote exploitation of an unaltered passenger vehicle* (pp. 1–91). Black Hat USA, (S 91), (pp. 1–91).
- Morris, D., Madzudzo, G., & Garcia-Perez, A. (2020). Cybersecurity threats in the auto industry: Tensions in the knowledge environment. *Technological Forecasting and Social Change*, 157, 120102.
- Moukahal, L. J., Zulkernine, M., & Soukup, M. (2021). Towards a secure software lifecycle for autonomous vehicles. In *2021 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)*, (pp. 371–377). IEEE.
- Murphy, G. (n.d.). Position Independent Executables (PIE). Red Hat. 2012–11–28. <https://www.redhat.com/en/blog/position-independent-executables-pie>. Last Accessed: 2023/04/07.
- Nie, S., Liu, L., & Du, Y. (2017). Free-fall: Hacking tesla from wireless to can bus. *Briefing, Black Hat USA*, 25(1), 16.
- Pham, M., & Xiong, K. (2021). A survey on security attacks and defense techniques for connected and autonomous vehicles. *Computers & Security*, 109, 102269.
- Qualcomm Technologies, Inc. (2024). Qualcomm product security bulletins. Qualcomm Technologies, Inc. <https://www.qualcomm.com/company/product-security/bulletins>
- Red Hat (n.d.). Audit system architecture (in Chapter 7. System auditing). In *Security Guide, Red Hat Enterprise Linux 7*. https://docs.redhat.com/en/documentation/red_hat_enterprise_linux/7/html/security_guide/chap-system_auditing#sec-audit_system_architecture
- Ryan, M. K., & Rozier, K. Y. (2024). A survey and analysis of recent IoT device vulnerabilities. SFAE 6.0 (2023). <https://scaledagileframework.com/> released in March 2023.
- Snyk (2023). Snyk named a leader in The Forrester Wave: SCA, Q2 2023. <https://snyk.io/blog/snyk-named-leader-in-strategy-category-forrester-wave-sca-q2-2023/>. Last Accesses 25/08/09.
- Standard, C. C. (2016). SEI CERT.
- Staron, M. (2020). Action research in software engineering (p. 50). Berlin/Heidelberg, Germany: Springer International Publishing.
- Sun, X., Yu, F. R., & Zhang, P. (2021). A survey on cyber-security of connected and autonomous vehicles (CAVs). *IEEE Transactions on Intelligent Transportation Systems*, 23(7), 6240–6259.
- Threat Actors, Car Systems, and Weak API Security. <http://avertium.com/resources/threat-reports/tir-20230213-threat-actors-car-systems-and-weak-api-security>
- Vulert's (n.d.). All-in-One Open Source Security and Compliance Platform. <https://vulert.com/>. Last Accesses 25/08/09.
- Wichers, D. (n.d.). Free for Open Source Application Security Tools. https://owasp.org/www-community/Free_for_Open_Source_Application_Security_Tools. Last Accessed 24/06/22.
- Williams, J. (2020). How to build awesome security instrumentation to automate AppSec testing and protection [Conference presentation]. OWASP. <https://share.google/NqDwzorvIK9CJ9fmM>
- Wohlin, C., Runeson, P., Höst, M., Ohlsson, M. C., Regnell, B., & Wesslén, A. (2012). *Experimentation in software engineering* (Vol. 236). Berlin: Springer.
- Xiong, W., Gülsever, M., Kaya, K.M., & Lagerström, R. (2019). A Study of Security Vulnerabilities and Software Weaknesses in Vehicles. In A. Askarov, R. Hansen, & W. Rafnsson (Eds.) *Secure IT Systems*. NordSec 2019. Lecture Notes in Computer Science, vol 11875. Springer, Cham.



Srijita Basu is a Postdoctoral Researcher in the Software Engineering Unit at Chalmers University of Technology and the University of Gothenburg, Sweden. Her research interests include Agentic AI, Cybersecurity, Automotive Software, Cloud Computing, and Blockchain. She received her Ph.D. in 2023 from the Indian Institute of Information Technology Kalyani. Her current work investigates the use of large language models for software vulnerability detection and correction. She also studies interactions between AI agents, focusing on analyzing multi-agent collaboration patterns and formulating metrics to systematically characterize and evaluate agent interactions and behavior.



Mirosław Staron is a Professor of software engineering with the Department of Computer Science and Engineering, Chalmers, and University of Gothenburg, Sweden. He is the Editor-in-Chief of Information and Software Technology and has a regular column in IEEE Software (Practitioners Digest). He is also on the editorial boards of IEEE SOFTWARE and Elsevier Journal of Systems Architecture. He has been active in national bodies such as AI Sweden, AI Competence for Sweden, and Swedsoft. His work has also been recognized as an Excellent Teacher with the University of Gothenburg. His research interests include software design, metrics, machine learning, and software quality. He has authored five books and over 250 peer-reviewed articles in software engineering.



Magnus Almgren is a professor at the Department of Networks and Systems, Department of Computer and Information Engineering at Chalmers University of Technology. His research focuses on security in computer and network systems, with a particular focus on intrusion detection, machine learning applied to security, and protection of cyber-physical systems and the Internet of Things. His work addresses the resilience and resilience of critical infrastructure such as industrial control systems, smart grids, and in-vehicle networks. He combines methods from anomaly detection, statistical analysis, and spectral analysis to identify cyberattacks against complex systems, and has been involved in building testbeds to evaluate security solutions in operational environments. Related topics include integrity-preserving data analytics, blockchain technology for resource-constrained IoT devices, and cybersecurity in the automotive industry.



Jennifer Horkoff is an Associate Professor (Docent) in the Department of Computer Science and Engineering at Gothenburg University and Chalmers University of Technology. Her research focuses on the intersection of software engineering and artificial intelligence, including requirements engineering for machine learning systems and non-functional requirements for AI-driven software. Her methods combine empirical studies with tool support for modelling complex requirements. Jennifer is on the program and/or organizing committees of several international conferences including the IEEE Requirements Engineering Conference, the International Conference on AI Engineering – Software Engineering for AI (CAIN), the Working Conference on Requirements Engineering: Foundation for Software Quality (REFSQ), and the IFIP WG 8.1 Working Conference on the Practice of Enterprise Modeling (PoEM).



Professor Dr. Christian Berger is Full Professor at the Department of Computer Science and Engineering at University of Gothenburg, Sweden and received his Ph.D. degree from RWTH Aachen University, Germany in 2010. He coordinated the research project for the self-driving vehicle "Caroline", which participated in the 2007 DARPA Urban Challenge Final – the world's first urban robot race. He also co-led the Chalmers Truck Team during the 2016 Grand Cooperative Driving Challenge (GCDC) and is the leading software architect at Chalmers Revere, the laboratory for automotive-related research. His research expertise is on architecting cloud-enabled cyber-physical systems and event identification in multi-modal, large-scale, time-series datasets to support the engineering of safe and trustworthy AI-enabled systems in the era of agentic engineering.

Authors and Affiliations

Srijita Basu^{1,2} · Mirosław Staron^{1,2} · Magnus Almgren^{1,2} · Jennifer Horkoff^{1,2} · Christian Berger^{1,2}

✉ Srijita Basu
srijita.basu@gu.se

Mirosław Staron
miroslaw.staron@gu.se

Magnus Almgren
magnus.almgren@chalmers.se

Jennifer Horkoff
jennifer.horkoff@gu.se

Christian Berger
christian.berger@gu.se

¹ Chalmers University of Technology, 417 56 Göteborg, Sweden

² University of Gothenburg, 417 56 Göteborg, Sweden