World Electric Vehicle Journal

MDPI

Article

Cybersecurity Risks in EV Mobile Applications: A Comparative Assessment of OEM and Third-Party Solutions

Bilal Saleem [1], Alishba Rehman [1], Muhammad Ali Hassan [1] and Zia Muhammad [2,*]

[1] Department of Cybersecurity, Air University, Islamabad 44000, Pakistan; 211096@students.au.edu.pk (B.S.); 211120@students.au.edu.pk (A.R.); 211052@students.au.edu.pk (M.A.H.)
[2] Department of Computing, Design, and Communication, University of Jamestown, Jamestown, ND 58405, USA
* Correspondence: zia.muhammad@uj.edu

Abstract

As the world accelerates toward a sustainable future with electric vehicles (EVs), smartphone applications have become an indispensable tool for drivers. These applications, developed by both EV manufacturers and third-party developers, offer functionalities such as remote vehicle control, charging station location, and route planning. However, they also have access to sensitive information, making them potential targets for cyber threats. This paper presents a comprehensive survey of the cybersecurity vulnerabilities, weaknesses, and permissions in these applications. We categorize 20 applications into two groups: those developed by EV manufacturers and those by third parties, and conduct a comparative analysis of their functionalities by performing static and dynamic analysis. Our findings reveal major security flaws such as poor authentication, broken encryption, and insecure communication, among others. The paper also discusses the implications of these vulnerabilities and the risks they pose to users. Furthermore, we analyze 10 permissions and 12 functionalities that are not present in official EV applications and mostly present in third-party apps, leading users to rely on poorly built third-party applications, thereby increasing their attack surface. To address these issues, we propose defensive measures which include 10 CWE AND OWASP top 10 defenses to enhance the security of these applications, ensuring a safe and secure transition to EVs.

Keywords: electric vehicle (EV); cybersecurity; EV mobile application security; Android application vulnerabilities; application permissions; secure coding for Android applications; EV charging application security; OWASP mobile top 10 vulnerabilities; CWE weaknesses; third-party application security; Android manufacturer application vulnerabilities

Academic Editor: Paulo J. G. Pereirinha

Received: 7 May 2025
Revised: 15 June 2025
Accepted: 24 June 2025
Published: 30 June 2025

Citation: Saleem, B.; Rehman, A.; Hassan, M.A.; Muhammad, Z. Cybersecurity Risks in EV Mobile Applications: A Comparative Assessment of OEM and Third-Party Solutions. *World Electr. Veh. J.* **2025**, *16*, 364. https://doi.org/10.3390/wevj16070364

1. Introduction

EVs are gaining traction as an eco-friendly and economical substitute for traditional cars that run on gasoline. Unlike conventional cars that rely on internal combustion engines burning fossil fuels, EVs utilize one or more electric motors for propulsion [1]. Depending on the electricity source and storage, EVs can be classified into various types such as battery electric vehicles (BEVs), plug-in hybrid electric vehicles (PHEVs), and fuel cell electric vehicles (FCEVs) [2]. Compared to their conventional counterparts, EVs offer numerous advantages. They help reduce greenhouse gas emissions, improve air quality, bolster energy security, and create new economic opportunities [3].

The International Energy Agency projects that by 2030, 125 million EVs will be on the road, accounting for nearly a third of all passenger vehicles [4]. Many factors are driving its growth, including rising fuel costs, rising concerns about climate change, and government incentives to adopt electric vehicles EVs to deliver economic benefits in addition to environmental benefits. Because EVs require less fuel and maintenance than gasoline-powered vehicles, they are generally cheaper to maintain. In addition, many countries around the world offer them various subsidies and incentives, such as tax credits and tax exemptions [5]. For example, in the United States, EV buyers can claim a federal tax credit of up to USD 7500 depending [6]. In China, the world's largest EV market, EV buyers can receive a 10% reduction in consumption tax and a subsidy of up to CNY 18,000 [7].

The rapid growth of EVs has led to a parallel increase in the demand for mobile applications that facilitate the management and monitoring of these vehicles. Most EVs are equipped with a proprietary mobile application provided by the manufacturer, explicitly designed to control and monitor certain types of EVs. These applications enable users to efficiently manage and monitor their EV usage [8]. These mobile applications offer a plethora of conveniences to the user. They provide the real-time tracking of charging status, battery levels, and travel range. They also send notifications upon charging completion and facilitate payment at public charging stations. Some manufacturers even allow users to remotely control certain vehicle functions such as door locks, climate control systems, headlights, maintenance schedules, and charging station payment [9].

Apart from the applications offered by EV manufacturers, there are many applications developed and uploaded by third-party developers [10]. Although these applications help consumers, but they also provide developers and manufacturers with useful information on everyday activities, driving patterns, vehicle use, and user behavior [11]. Therefore, the security and integrity of these applications play a crucial role, considering that the data they manage is sensitive. The data collected by these applications can be used for advertising, insurance, and product development. Most applications collect data—which includes car performance metrics, location, infotainment system data, sensor data, driving patterns, personal information, financial details, and user activities [12]. Figure 1 provides an overview of the real-time functionality and important EV application features that are common in most EV applications. It illustrates the key significance and dependency of mobile applications for EV owners and drivers. The image represents that applications can be used for vehicle tracking, remote unlocking, and charging updates, among others [13].

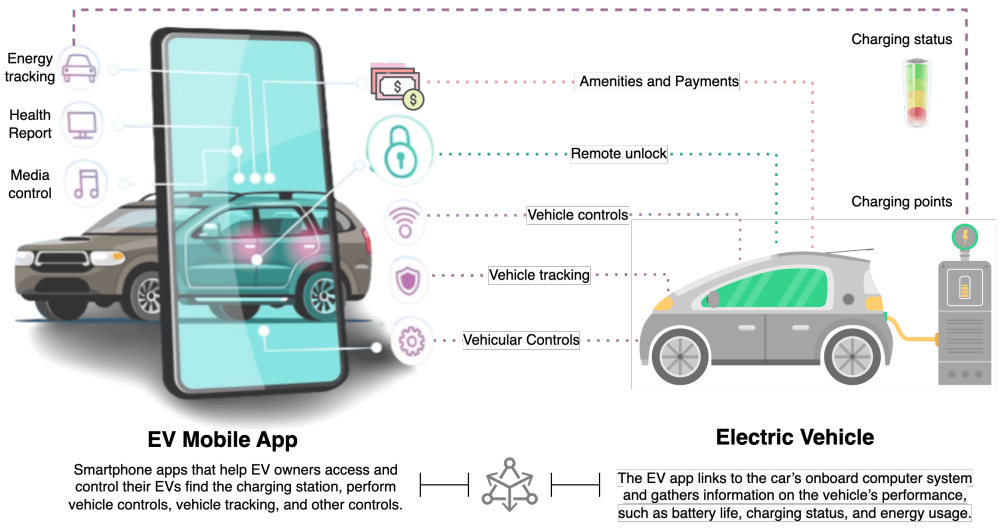


Figure 1. The image represents EV functionality that can be accessed using mobile applications.

To address these cybersecurity challenges and defend against vulnerabilities in mobile application security, we provide a comprehensive cybersecurity assessment of EV mobile applications. We choose these applications from the application distribution platforms and divide them into two groups based on the developers: (1) Applications developed by EV manufacturers and (2) applications developed by independent third-party developers. We then develop a comprehensive methodology to analyze these applications which allows us to perform permission analysis, common weakness enumeration (CWE) [14], and the most critical security vulnerabilities based on the open web-application-security-project (OWASP) security standards [15]. After the identification of vulnerabilities, defensive measures are proposed to enhance the security of these applications. The novelty of our work lies in the comprehensive nature of the assessment, identifying the features that attract users to install third-party applications, and proposing solutions to improve the security.

1. The paper presents a thorough security assessment of EV Android applications. This includes both static and dynamic analysis phases, knowledge building, and a security assessment phase, providing a detailed understanding of the cybersecurity landscape in these applications.
2. The research identifies major security flaws such as poor authentication, broken encryption, insecure communication, and more. It also pinpoints the OWASP top ten vulnerabilities and analyses for CWE, contributing valuable insights to the field of cybersecurity in EV applications.
3. The paper conducts a detailed analysis of the requested permissions in these applications. It also identifies functionalities that are not present in official EV applications, leading users to rely on third-party applications, thereby increasing their attack surface.
4. Finally, the research proposes defensive measures based on OWASP and CWE defenses to mitigate the identified security issues to enhance the security of EV Android applications.

This paper is organized as follows: Section 2 reviews the literature on EVs and identifies research gaps. Section 3 selects applications for security assessment and compares the functionalities and features of the applications. Section 4 describes the methodology and procedure of the assessment. Section 5 presents the results and findings of the analysis. Section 6 provides valuable insights on the results. Section 7 suggests defensive measures to improve the security of the applications. Section 8 concludes the article and discusses future work. Figure 2 shows a paper organization flowchart showing the structure from introduction to conclusion.

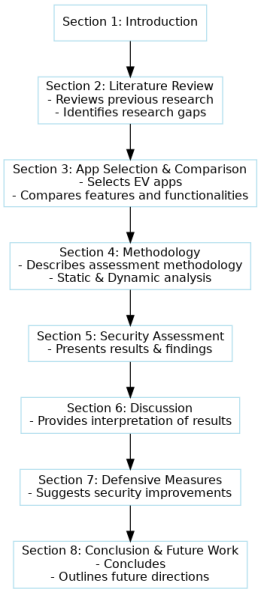


Figure 2. Paper organization flowchart.

2. Literature Review

The increasing popularity and mass production of EVs raise challenges related to cybersecurity concerns. There are five components of EV security profiling: charging station security, information privacy, software security, connected vehicle security, and autonomous driving security [16]. Most vehicles now come with mobile applications that use cloud and fog computing paradigms. These applications communicate with trusted cloud servers through intermediary node servers and possess multiple data entry and exit points where security is needed [17].

Additionally, the growth of electric vehicle adoption has led to increased demand for electric vehicle supply equipment (EVSE) infrastructure and mobile applications. This has also exposed vulnerabilities in EVSE systems, including weak authentication mechanisms and end-to-end communications [18].

EV mobile applications are among the customer-valued areas that may encourage adoption, with security having a significant impact on this [19]. The landscape of mobile applications available to EV drivers is highly fragmented and lacks uniformity and standards and cybersecurity measures [20].

EV applications can misuse sensitive data if they have security flaws, such as overclaimed permissions, obsolete API calls, vulnerabilities, and weaknesses [21]. Among all the mobile applications connected with Android Auto infotainment systems, nearly 80% of the applications are potentially vulnerable, with 25% posing security threats related to the execution of JavaScript [22]. There are security concerns associated with connecting Android devices to car infotainment systems, which can be exploited to read sensitive data and send dangerous commands to the car's hardware [23].

Popular EV applications from manufacturers and third-party developers exhibit security vulnerabilities, potentially compromising sensitive information such as vehicle location, statistics, and personally identifiable information (PII) stored in mobile devices [24]. There is a possibility of remote attacks on mobile applications used for locating EV charging points [25]. There is also the possibility of hacking through existing vulnerabilities in various mobile EV applications, application authentication and insecure communication. Moreover, there are cyber threats to EVs, third-party libraries, and implemented sensors [26]. There are many other security vulnerabilities that exist in Android vehicle-related applications. As these applications use network security configuration (NSC) in enhancing security practices, if not effectively utilized, it can create security vulnerabilities [27].

There is a need for the security assessment of EV mobile applications, as they collect data from EV sensors, such as LiDAR, radar, and cameras, to enhance 3D object detection and visualization, and having vulnerability in these applications can put users on risk [28]. Also, it is equally important to ensure secure vehicle-to-everything (V2X) communications in connected car mobile applications [11]. Safeguards and defensive controls are needed to address the security vulnerabilities in EV applications, particularly focusing on encryption for data transmission and secure data storage to prevent potential security incidents.

Moreover, vehicular technology is also evolving to attain emerging technologies and blockchain-based self-certified key exchange protocols, which propose a highly secure self-certified key exchange protocol that if used in mobile applications, can provide enhanced security, authenticity, and privacy [29].

The existing literature has identified some of the cybersecurity risks and threats that affect EV mobile applications shown in Table 1. However, some identified limitations may include a lack of systematic studies that analyze the security of a wide range of EV mobile applications using various tools and techniques. Also, there is a lack of comparative analysis of the functionality of EV mobile applications, which is the reason to influence users into installing these applications. Therefore, this article aims to provide a detailed cybersecurity

assessment of popular EV mobile applications, identify the features that attract users to install third-party applications, and propose defenses to mitigate the vulnerabilities and build resilience.

Table 1. Comparative analysis of main findings and contributions of the selected articles.

Author	Year	EV Manufacturers	Third-Party Apps	Features Analysis	Permission Analysis	OWASP Mapping	CWE Mapping	Defenses
Zhang et al. [27]	2024	✓	✓	✓	✗	✗	✗	✓
Muhammad et al. [24]	2023	✓	✓	✓	✓	✓	✗	✗
Shirvani et al. [16]	2023	✓	✓	✓	✗	✗	✗	✓
SARIEDDINE et al. [25]	2023	✗	✓	✓	✓	✗	✗	✓
Vailoces et al. [18]	2023	✗	✗	✓	✓	✗	✗	✓
Topman et al. [11]	2022	✓	✓	✗	✗	✗	✗	✗
Chatzoglou et al. [21]	2021	✓	✗	✗	✓	✗	✓	✗
Panarotto et al. [23]	2018	✓	✓	✓	✓	✗	✗	✗
Hanelt et al. [19]	2015	✓	✗	✓	✗	✗	✗	✗
Stillwater et al. [20]	2013	✓	✓	✓	✗	✗	✗	✗
This Article	**2025**	✓	✓	✓	✓	✓	✓	✓

Note: ✓ = Present; ✗ = Not Present. Bold text highlights the current study ("This Article").

3. Application Selection and Functionality Influence

This section describes the selection criteria and functionality of the applications that we used for the underlined article. First, applications were downloaded from the Google Play Store. There are a total of twenty applications which are divided into two groups, Group:1 represents EV manufacturers-developed applications, and Group:2 shows third-party developers' applications.

3.1. Group: 1—EV Manufacturers Applications

1. TESLA: This user-friendly application is designed for managing Tesla vehicles. It offers features like tracking your vehicle, starting or stopping charging remotely, and even locking and unlocking your car. Additionally, it helps users locate charging stations, schedule service appointments, and download the latest updates.
2. Mercedes: This application locks or unlocks the doors, remote starts, tracks the vehicle, records driving habits, and schedules service appointments.
3. BMW: It performs vehicle tracking, allows remote starts, schedules service appointments, and allows maintenance updates.
4. Nissan: This application is all about convenience for Nissan vehicle owners. It enables users to locate and track their vehicles and even offers keyless entry and remote start capabilities. It is compatible with Alexa-enabled devices and Google Assistant, allowing users to use voice commands to start the car, turn on the lights, and interact with their vehicle in various other ways.
5. Volkswagen: The application provides remote access, reserves parking spots, schedules service appointments, and downloads updates.
6. Jaguar InControl: It can schedule maintenance and service appointments, remotely activate the vehicle's horn and lights, access trip history and driving statistics, and remotely start.

7. FordPass: It provides details on fuel levels, allows the lock/unlock of Ford vehicles, checks vehicle health information, finds parking spots, and uses platform assistance.
8. MITSUBISHI: It enables users to remotely control climate settings, manage anti-theft functions, monitor current travel information, and manage charging.
9. Land Rover InControl: It allows users to remotely unlock doors, start the engine, update climate settings, use the vehicle locator, and receive maintenance alerts.
10. MY FERRARI: It allows users to oversee and manage Ferrari vehicles from a distance, review details of the vehicle, access vehicle location tracking, and start or stop the engine, and manage vehicular updates.

3.2. Group: 2—Third-Party Applications

1. EVConnect: This app is a helpful tool for EV drivers. It helps users find EV charging stations, pay for charging services, and send notifications when the charging is complete.
2. EVgo: It allows users to check charging information, locate charging stations, and book a charger. This application also supports loyalty programs and coupons.
3. Plugshare: This app helps users find charging stations, pay for services, and monitor the charging process. It allows users to share their experiences, stories, and photos, and rate their overall experience.
4. ChargeHub: This app is a helpful assistant for electric vehicle owners. It helps users find, compare, and navigate to charging stations. It provides updates on the availability of stations, making it easier for users to plan their charging.
5. EV-Energy: This app is designed to help EV owners manage their home charging needs. It includes features like setting reminders, tracking charging time, and monitoring energy use, making home charging an ease for EV owners.
6. A better route planner (ABRP): It offers trip planning tools for, taking into account weather updates, traffic situations, and charging station amenities.
7. Caura: It allows users to track car-related expenses, track charging costs, update reminders, and track vehicles.
8. Bp Plus: This application is designed for businesses that manage fleets and provide battery-saving insights, find nearby charging stations, and give driving habits.
9. Optiwatt: It allows EV owners to optimize their charging schedule based on energy costs and grid demand. The application also provides analytics and reports on charging behavior and energy consumption.
10. Octopus electroverse: It gives users access to electric vehicle charging stations and uses smart charging features to take advantage of off-peak electricity pricing.

3.3. Comparative Analysis of Functionalities Offered by EV Applications

Application functionality is an important factor that influences users to retain a specific application. The functionalities present in chosen applications are classified into 12 groups, denoted by *F1* to *F12*, where *F* represents functionality. This notation is consistent with the previous scheme of using a letter and a number to represent the applications and optimize the table structures in subsequent sections.

F1. Charging points: These features allow users to search nearby charging point locations.
F2. Vehicle controls: This allows monitor and control different functions of vehicles.
F3. Calls and messages: This helps in making and receiving calls and messages in mobile applications.
F4. Media controls: This controls media, audio, and other entertainment options.
F5. Remote unlock: As the name suggests, it unlocks a car remotely.
F6. Charging status: This feature provides the charging updates of a vehicle.

F7. Vehicle tracking: This will track the location of vehicles using the GPS coordinates.
F8. Car health report: This returns a report on the health and performance of the underlined vehicle.
F9. Payments: This features let a customer pay for various services, like charging.
F10. Amenities: The function helps a user to find hotels and other services.
F11. Energy use tracking: It tracks and analyzes the energy used by a vehicle.
F12. Rewards and coupons: As the name suggests, it is linked to receiving and redeeming rewards and coupons as users use certain services.

Table 2 provides a detailed comparative analysis of the functionalities offered by different EV applications. The first column categorizes the applications into two groups: *Group 1* (applications developed by EV manufacturers) and *Group 2* (applications developed by third-party developers). This categorization is crucial because it allows us to compare the security features and vulnerabilities between manufacturer-developed and third-party applications, which could have different development practices and security standards.

The second column lists the names of the applications, and the remaining columns detail the functionalities present in each application. This information is significant because it helps us understand the scope of each application and the potential security implications associated with each functionality.

The last column lists the total number of functions available in each application. This is an important metric, as it gives us an idea of the complexity of the application. Generally, more functionalities could mean a larger attack surface and potentially more vulnerabilities.

Table 2. Comparative functional features across EV mobile applications.

Group	App Name	F1	F2	F3	F4	F5	F6	F7	F8	F9	F10	F11	F12	Total
Group 1	TESLA [30]	✗	✓	✗	✓	✓	✓	✓	✗	✗	✗	✗	✗	5
	Mercedes [31]	✗	✓	✗	✗	✓	✗	✓	✗	✗	✗	✗	✗	3
	BMW [32]	✓	✓	✗	✗	✓	✗	✗	✗	✗	✗	✗	✗	3
	Nissan [33]	✗	✓	✗	✗	✓	✗	✗	✓	✗	✗	✗	✗	3
	Volkswagen [34]	✗	✓	✗	✗	✓	✗	✓	✗	✗	✗	✗	✗	3
	FERRARI [35]	✓	✗	✗	✓	✓	✓	✗	✓	✓	✓	✗	✗	7
	FordPass [36]	✓	✗	✓	✗	✓	✓	✓	✓	✗	✗	✗	✗	6
	MITSUBISHI [37]	✗	✓	✗	✓	✓	✓	✓	✗	✗	✗	✗	✗	5
	Land Rover [38]	✗	✓	✗	✓	✓	✓	✓	✗	✗	✗	✗	✗	5
	Jaguar [39]	✗	✓	✗	✓	✓	✓	✓	✗	✗	✗	✗	✗	5
Group 2	EV Connect [40]	✓	✗	✗	✗	✗	✓	✗	✗	✓	✗	✓	✗	4
	EVgo [41]	✓	✗	✓	✗	✗	✓	✗	✗	✗	✗	✗	✗	3
	PlugShare [42]	✓	✗	✗	✗	✗	✗	✓	✗	✓	✓	✗	✗	4
	ChargeHub [43]	✓	✗	✗	✗	✗	✓	✗	✗	✗	✓	✗	✗	3
	Optiwatt [44]	✗	✓	✗	✓	✓	✓	✓	✗	✗	✗	✗	✗	5
	Routeplanner [45]	✗	✓	✗	✓	✓	✓	✓	✗	✗	✗	✗	✗	5
	Caura [46]	✗	✓	✗	✓	✓	✓	✓	✗	✗	✗	✗	✗	5
	Bp Plus [47]	✓	✗	✓	✗	✓	✓	✓	✓	✗	✗	✗	✗	6
	Electroverse [48]	✗	✓	✗	✓	✓	✓	✓	✗	✗	✗	✗	✗	5
	EV-Energy [49]	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✓	✓	2

The visual analysis of available functions is represented in Figure 3. The figure shows the count of functionality for each selected application, as well as the corresponding feature categories denoted by F1 to F12. For example, if we look at the leftmost bar, we can see that the Tesla application offers five functionalities, which are F2 (vehicle controls), F4 (media controls), F5 (remote unlock), F6 (charging status) and F7 (vehicle tracking). The figure helps to visualize and compare the functionalities and features of the applications.

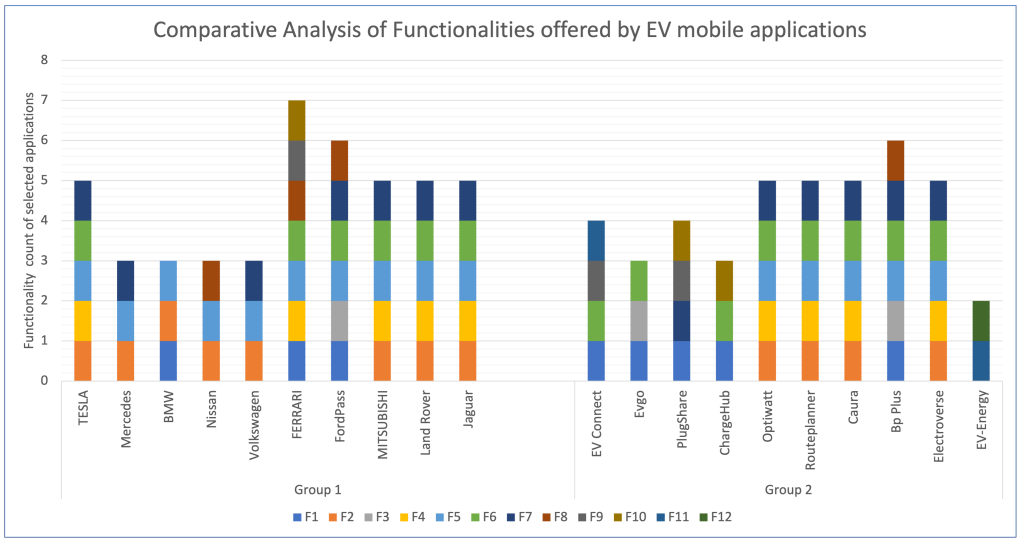


Figure 3. Comparative analysis of functionalities offered by EV mobile applications.

On the other hand, Figure 4 shows the widely available functionality that is present in most applications. This gives insight into the importance of the most necessary functionality for drivers, which must be considered while building new applications. For example, we can see from the graph that the three most common features that are available in almost every application are F5 (remote unlock), F6 (charging status), and F7 (vehicle tracking). This can be used as a fair minimum standard for new applications to survive the market dynamics and fulfill basic user needs.

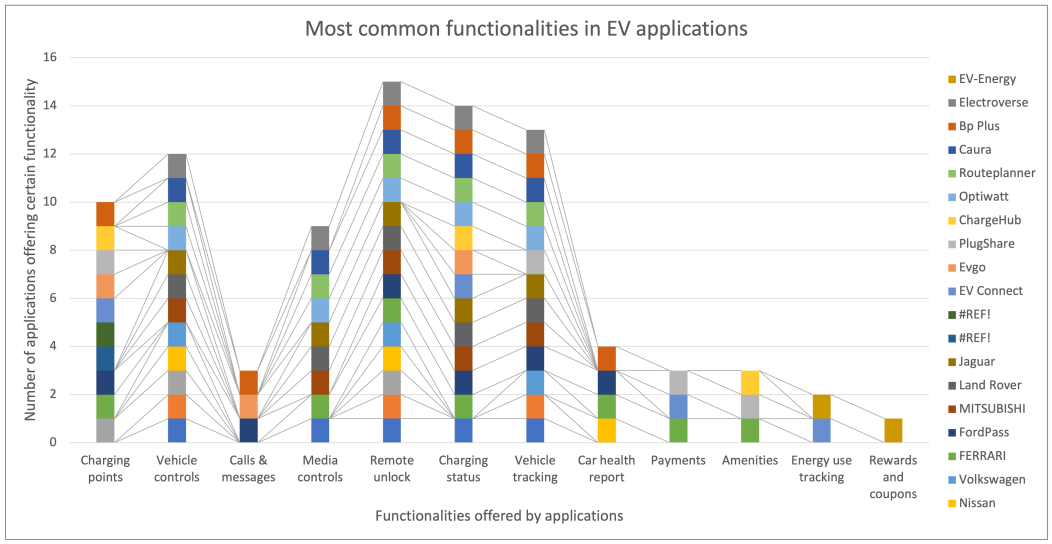


Figure 4. Most common functionalities in EV applications.

4. Security Assessment Methodology

This section describes the methodology, which consists of four phases: (1) the static analysis phase, (2) the dynamic analysis phase, (3) the knowledge-building phase, and (4) the security assessment phase. Figure 5 provides a broad outline of each phase.

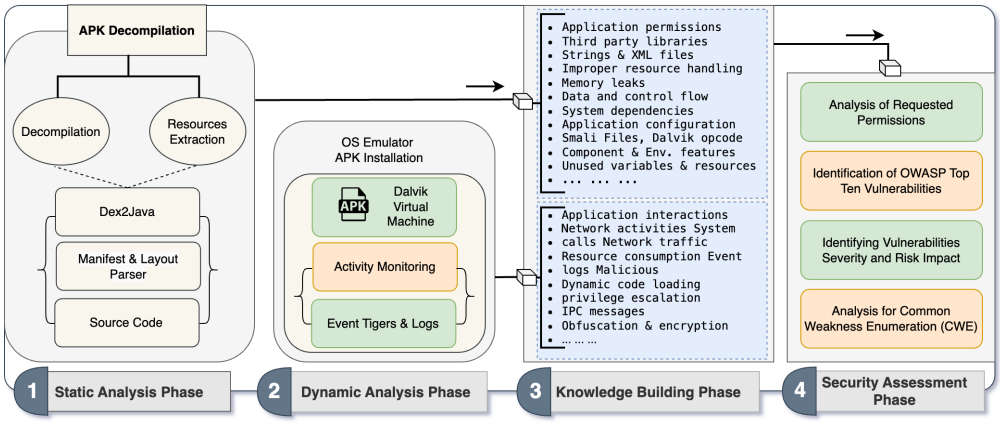


Figure 5. Cybersecurity assessment methodology used to find security vulnerabilities in mobile applications.

To provide a comprehensive understanding of our methodology, it is important to describe the specific timeframe, within which each phase of our cybersecurity assessment was conducted. The static analysis phase started immediately after selecting suitable applications and spanned over a period of two weeks. During this time, applications were decompiled and thoroughly examined for potential security vulnerabilities. Following this, the dynamic analysis phase was initiated, which involved running applications in a controlled virtual environment to monitor real-time operations; this phase lasted three weeks due to its complexity and depth of analysis required.

Subsequently, we allocated one week for the knowledge-building phase, where data from both static and dynamic analyses were synthesized using advanced Python visualization tools (Matplotlib v3.7.1 and Seaborn v0.12.2, created by the Python Software Foundation, Wilmington, DE, USA) to create an insightful knowledge base. Finally, in the security assessment phase, which took another two weeks, we leveraged our comprehensive knowledge base to evaluate all identified security vulnerabilities critically. In total, our methodology was executed over an eight-week period, ensuring meticulous attention to detail at every stage. This systematic approach allowed us to conduct an in-depth and comprehensive security assessment of each application. The details of these phases are discussed in the subsequent sections.

4.1. Static Analysis Phase

In this phase, we performed a static analysis of the selected applications. We started by decompiling the applications using Apktool [50], which allowed us to extract their resources and retrieve valuable information. After the decompilation, we extracted the Android manifest file, opened it with the text editor, and manually extracted different permissions used by applications. This gave us insights into the level of access each application requires. We identified third-party libraries used in the applications by examining the decompiled code. This helped us understand the external dependencies of the applications.

We extracted strings and XML files from the decompiled code using xmlstarlet [51], a toolkit to identify hard-coded secrets, URLs, and other potential security risks. Finally, we examined the AndroidManifest.xml file of each application to understand its configuration, including its components (activities, services, receivers, and intents) and their respective

permissions. Activities, Services, Receivers, and Intents: We analyzed the inter-component communication in the applications by examining their manifest files and decompiled code. This helped us identify potential vulnerabilities related to intent spoofing, unprotected components, and more. This comprehensive approach to static analysis allowed us to gain a deep understanding of the applications' structure, behavior, and potential security vulnerabilities. We retrieved the following information from the applications [52]:

1. Application permissions.
2. Third-party libraries.
3. Strings and XML files.
4. Improper resource handling.
5. Application configuration.
6. Activities, services, receivers, and intents.
7. Unused variables and resources.

4.2. Dynamic Analysis Phase

In the dynamic analysis phase, we used a combination of publicly available tools and custom scripts to monitor the real-time operations of the applications. We used an Android debug bridge (ADB) for communication between the host computer and the emulator [53].

To detect malicious behaviors, such as dynamic code loading, privilege escalation, and data exfiltration, we used MobSF [54], a mobile security framework that provides dynamic analysis and malware detection. For detecting obfuscation, encryption, and antidebugging techniques, we used JEB Decompiler [55]. Finally, Wireshark was used to capture network traffic, allowing us to analyze network activities, protocols, traffic flow, domains, and APIs [56]. We can also use Android Studio, which includes Android 11 emulators; for automated dynamic testing, we can use MobSF; and also to monitor network traffic, we can use Wireshark that is configured with a proxy. These additions aim to clarify the setup and reproducibility of our assessment process. This comprehensive toolkit allowed us to perform a thorough dynamic analysis of the applications, providing valuable insight information about their runtime behavior and potential security vulnerabilities. The following information was obtained from the applications [57]:

1. Application interactions, inputs, outputs, and interprocess communication.
2. Network activities, network protocols, traffic flow, domains, and APIs.
3. System calls, file operations, processes, and socket operations.
4. Resource consumption, CPU usage, memory usage, and battery usage.
5. Event logs, malicious behaviors, dynamic code loading, privilege escalation, and data exfiltration.
6. Obfuscation and encryption, and anti-debugging.

4.3. Knowledge Building Phase

In this phase, we analyzed the data gathered from the previous two phases and built a knowledge base that contains insights about the applications using Python visualization libraries to perform the following tasks:

1. Data cleaning and transformation: This is the process of removing or correcting inaccurate, incomplete, or irrelevant data from the dataset, such as missing values, outliers, and duplicates.
2. Data Integration and Visualization: It is the process of combining or merging data from different sources or formats into a unified dataset and then displaying graphical representations of the data.

3. Data Mining and Interpretation: This process involves discovering and extracting useful patterns and trends from the data.

4.4. Security Assessment Phase

In this phase, we evaluated all security vulnerabilities in every application using the knowledge base created in the previous phase and performed the following tasks [58–60].

1. Analysis of requested permissions, their necessity, legitimacy, and risk level.
2. Identification of OWASP Top Ten vulnerabilities, which serves as a standard reference for identifying critical security risks.
3. Identifying the severity and impact of vulnerabilities, their likelihood of occurrence, and potential consequences.
4. Analysis for CWE [14].

The underlined methodology is adopted to conduct an individual assessment of each application. The results and findings of the analysis are discussed in the subsequent sections.

5. Results and Findings

This section provides details about the results obtained from the analysis of the selected mobile applications using the methodology described in the previous section. The results are based on the data gathered from the static and dynamic analysis phases and the knowledge base created from the knowledge-building phase.

5.1. Analysis of Requested Permissions

Android operating systems enforce application permissions to access device functionalities. However, some permissions are considered dangerous for both the user and the system, as they may expose sensitive information, compromise security, or affect performance. Therefore, we analyze the permissions requested by the EV application and assess their necessity, legitimacy, and level of risk [61]. The permissions are classified into 10 groups, denoted by *P1* to *P10*. This notation is consistent with the previous scheme of using a letter and a number to represent the permissions and optimize the table structures in subsequent sections.

P1 Access precise location.
P2 Access camera.
P3 Read and update contacts.
P4 Manage calendar.
P5 Read/write external storage.
P6 Access call logs/records.
P7 Access accounts and stored credentials.
P8 Connect Bluetooth.
P9 Access network state.
P10 Search and connect WIFI.

Table 3 provides a comparative analysis of the permissions requested by both groups of applications. The first column categorizes the applications into two groups: *Group 1* (applications developed by EV manufacturers) and *Group 2* (applications developed by third-party developers). This categorization is crucial because it allows us to compare the security features and vulnerabilities between manufacturer-developed and third-party applications, which could have different development practices and security standards.

The second column lists the names of the applications, and the third column shows the specific application version that was tested. We mention the version here, as some versions may request extended permissions, which could potentially introduce additional

security risks. The remaining columns detail the permissions that each application requests. This information is significant, as it helps us understand the level of access each application requires to function. Excessive permissions could indicate potential privacy risks or security vulnerabilities. The last column lists the total number of permissions requested by each application. This is an important metric, as it gives us an idea of the scope of access each application requires. Generally, more permissions could mean a larger attack surface and potentially more vulnerabilities.

In our analysis of 20 EV applications, we observed a clear pattern in permission requests between these two groups: Group 1, which includes Manufacturer/OEM apps, requested an average of 6.0 permissions per app, while Group 2, which includes third-party apps, requested an average of 4.9 permissions per app. This shows a 22.5% higher permission request rate among OEM apps. Moreover, 70% of OEM apps requested more than five permissions, compared to only 40% of third-party apps. These results show a tendency toward permission overuse in OEM apps.

Table 3. List of permissions requested by EV mobile applications.

Group	Application Name	Version	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10	Total
Group 1	TESLA [30]	4.15.1	✓	✓	✓	✓	✓	✗	✗	✓	✓	✓	8
	Mercedes [31]	3.30.1	✓	✓	✗	✓	✓	✓	✗	✓	✓	✓	8
	BMW [32]	2.12.3	✓	✓	✓	✓	✓	✗	✗	✗	✓	✓	7
	Nissan [33]	6.0.1	✓	✓	✗	✓	✓	✓	✗	✗	✓	✗	6
	Volkswagen [34]	5.5.0	✓	✓	✗	✗	✓	✓	✗	✗	✓	✗	5
	FERRARI [35]	3.0.5	✓	✗	✗	✓	✓	✓	✗	✓	✓	✓	7
	FordPass [36]	4.25.0	✓	✓	✗	✗	✓	✗	✗	✓	✓	✓	6
	MITSUBISHI [37]	1.0.1	✓	✗	✗	✗	✗	✗	✗	✓	✗	✗	2
	Land Rover [38]	1.90.0	✓	✗	✗	✗	✓	✓	✓	✗	✓	✓	6
	Jaguar [39]	1.90.0	✓	✗	✗	✗	✓	✓	✓	✗	✓	✓	6
Group 2	EV Connect [40]	3.12.3	✓	✓	✗	✗	✓	✗	✗	✗	✗	✗	3
	EVgo [41]	7.12.0	✓	✓	✗	✗	✓	✗	✗	✗	✓	✓	5
	PlugShare [42]	4.0.0	✓	✓	✗	✗	✓	✗	✓	✗	✓	✓	6
	ChargeHub [43]	12.11.0	✓	✓	✗	✗	✓	✗	✗	✗	✓	✓	5
	Optiwatt [44]	1.3.9	✗	✓	✗	✗	✗	✗	✗	✗	✓	✓	3
	Routeplanner [45]	4.4.0	✓	✓	✗	✓	✓	✗	✗	✓	✓	✓	7
	Caura [46]	2.2.7	✗	✓	✗	✗	✓	✗	✗	✗	✓	✗	3
	Bp Plus [47]	2.5.1	✓	✗	✗	✗	✓	✗	✗	✗	✓	✓	4
	Electroverse [48]	56.0	✓	✗	✗	✗	✓	✗	✗	✗	✗	✓	3
	EV-Energy [49]	2.12	✓	✗	✗	✗	✓	✗	✗	✗	✓	✓	4

Figure 6 provides a high-level overview of the number of requested permissions in each application. The chart visualizes the distribution of permissions and resources across applications and compares the applications based on their permission count.

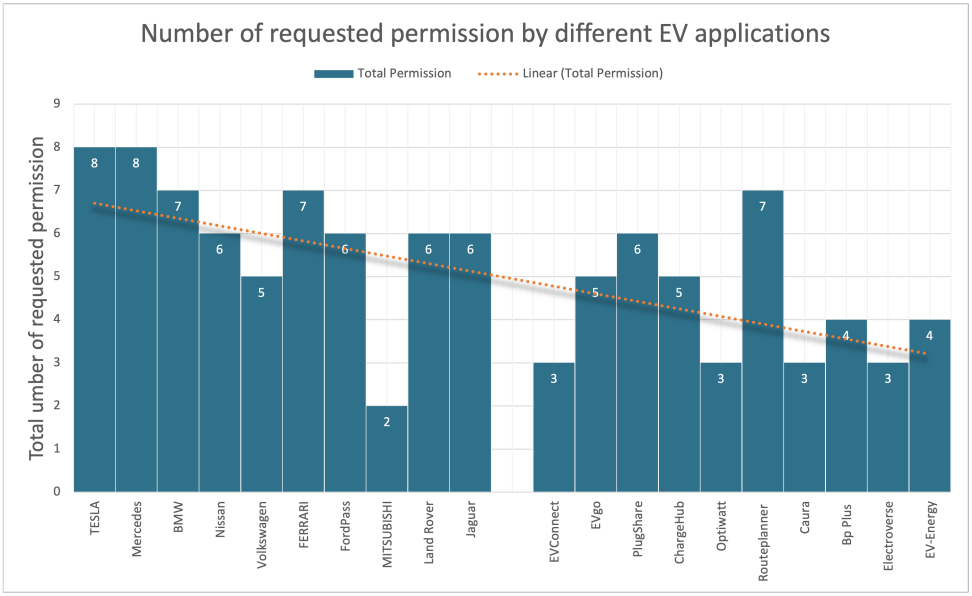


Figure 6. Number of requested permission by different EV applications.

5.2. Identification of OWASP Top Ten Vulnerabilities

OWASP is a non-profit organization dedicated to improving software application security [62,63]. In this subsection, we discuss the vulnerabilities that were identified in the code of the analyzed applications and further managed using textcolorredthe OWASP standards.

The identification and mapping of application vulnerabilities span over two tables. Table 4 covers Group 1 and Table 5 covers Group 2. Both tables on the left provide the first column as the vulnerability name, which is represented as V1–V10, consistent with the previous naming scheme of words and numbers. Column 2 maps the vulnerabilities according to their risk level, such as high, medium, and warning. These classifications are based on different types of vulnerabilities such as hardcoded credentials, improper cryptography, etc., the potential impact on system security, and alignment with OWASP Mobile Top 10 and CWE. Weaknesses such as data leakage or remote exploitation considered critical consequences are assigned as high severity, and vulnerabilities with moderate implications are considered medium. Besides this, those less critical are marked as warnings. This classification method reflects the widely accepted qualitative criteria. We know that there are more validated frameworks available. In the future, we will include more quantitative risk assessment models, such as CVSS, or ordered methodologies such as STRIDE or DREAD to provide consistent scoring. This improvement will enhance the reliability of our security evaluations. The rest of the table shows the applications that match the vulnerabilities. Finally, the last row at the bottom shows the total number of vulnerabilities present in each application.

V1 Improper platform usage, root functionality, hardware manipulation, or bypassing security measures that can result in unauthorized access, privilege escalation, or device compromise.

V2 Insecure data storage, storing data in plain text, using weak encryption, or lacking access controls.

V3 Insecure communication, unauthenticated connections, exposing data to interception, or leaking data to third parties.

V4 Insecure authentication, default or hard-coded credentials, lagging password policies, or relying on single-factor authentication.

V5 Insufficient cryptography, using outdated or broken algorithms, using weak or predictable keys, or implementing cryptography incorrectly.

V6 Insecure authorization, lack of role-based access control, lack of checking permissions, or excessive privileges. This may result in unauthorized access, privilege escalation, or data exposure.
V7 Quality of the client code, using insecure libraries or frameworks, having vulnerabilities or bugs in the code, or lacking code review or testing.
V8 Code tampering, adding malicious code, removing security checks, or hiding vulnerabilities.
V9 Reverse engineering, encryption keys, API keys, or credentials.
V10 Extraneous functionality, backdoors, debug functions, or test functions.

Table 4. Identification of vulnerabilities in *Group 1* applications developed by EV manufacturers.

Vul.	Risk	A1	A2	A3	A4	A5	A6	A7	A8	A9	A10
V1	HIGH	✓	✗	✓	✗	✗	✗	✗	✗	✗	✗
	MEDIUM	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗
	WARNING	✓	✓	✓	✗	✓	✗	✗	✗	✗	✗
V2	HIGH	✗	✗	✗	✓	✗	✗	✗	✗	✗	✓
	MEDIUM	✗	✓	✗	✗	✗	✓	✗	✗	✗	✗
	WARNING	✓	✓	✓	✓	✓	✓	✓	✗	✓	✓
V3	HIGH	✓	✓	✓	✓	✓	✓	✓	✓	✓	✗
	MEDIUM	✗	✗	✗	✗	✗	✗	✗	✗	✗	✓
	WARNING	✗	✓	✗	✗	✗	✗	✗	✗	✗	✗
V4	HIGH	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗
	MEDIUM	✓	✓	✓	✓	✓	✓	✓	✗	✓	✓
	WARNING	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗
V5	HIGH	✓	✓	✓	✓	✗	✗	✗	✗	✓	✗
	MEDIUM	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
	WARNING	✓	✓	✓	✓	✓	✗	✗	✓	✓	✗
V6	HIGH	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗
	MEDIUM	✓	✓	✓	✓	✓	✗	✗	✗	✓	✗
	WARNING	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗
V7	HIGH	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗
	MEDIUM	✗	✓	✓	✓	✓	✗	✗	✗	✗	✗
	WARNING	✓	✓	✓	✓	✓	✗	✗	✗	✓	✗
V8	HIGH	✓	✓	✓	✓	✓	✗	✗	✗	✗	✗
	MEDIUM	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗
	WARNING	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗
V9	HIGH	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗
	MEDIUM	✗	✗	✗	✗	✗	✗	✓	✗	✗	✗
	WARNING	✓	✓	✓	✓	✓	✗	✗	✗	✓	✗
V10	HIGH	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗
	MEDIUM	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗
	WARNING	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗
	Total	12	14	12	11	10	5	5	3	9	5

Note: ✓ = Present; ✗ = Not Present. Bold text highlights the current study ("This Article").

In addition to the OWASP Top Ten vulnerabilities, potential zero-day vulnerabilities could exist in these applications. Zero-day vulnerabilities are previously unknown vulnerabilities that could be exploited by attackers before they are discovered and fixed by the developers. These vulnerabilities pose a significant threat, as they leave the applications exposed to potential attacks until they are identified and patched.

Table 5. Identification of vulnerabilities in *Group 2* applications developed by third-party developers.

Vul.	Risk	A11	A12	A13	A14	A15	A16	A17	A18	A19	A20
V1	HIGH	✗	✓	✗	✓	✗	✗	✗	✗	✗	✗
	MEDIUM	✗	✗	✗	✗	✗	✗	✗	✗	✗	×
	WARNING	✗	✓	✗	✓	✓	✗	✗	✗	✗	×
V2	HIGH	✗	✗	✗	✗	✗	✗	✓	✗	✗	×
	MEDIUM	✗	✗	✗	✗	✗	✓	✓	✗	✓	✗
	WARNING	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
V3	HIGH	✓	✗	✗	✓	✓	✓	✗	✓	✓	✓
	MEDIUM	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗
	WARNING	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗
V4	HIGH	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗
	MEDIUM	✓	✗	✗	✗	✓	✓	✓	✓	✓	✗
	WARNING	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗
V5	HIGH	✓	✓	✓	✓	✓	✗	✗	✗	✗	✗
	MEDIUM	✓	✗	✗	✗	✓	✓	✗	✓	✓	✓
	WARNING	✓	✓	✓	✓	✓	✗	✓	✓	✓	✗
V6	HIGH	✗	✗	✗	✗	✗	✗	✓	✗	✗	✗
	MEDIUM	✗	✗	✗	✗	✗	✗	✗		✗	✗
	WARNING	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗
V7	HIGH	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗
	MEDIUM	✓	✗	✗	✗	✓	✗	✗	✓	✗	✓
	WARNING	✓	✓	✗	✗	✓	✗	✗	✓	✗	✗
V8	HIGH	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗
	MEDIUM	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗
	WARNING	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗
V9	HIGH	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗
	MEDIUM	✗	✗	✗	✗	✗	✓	✗	✗	✓	✗
	WARNING	✓	✓	✓	✓	✓	✗	✓	✓	✗	✗
V10	HIGH	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗
	MEDIUM	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗
	WARNING	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗
	Total	9	7	4	7	10	6	6	8	7	4

Note: ✓ = Present; ✗ = Not Present. Bold text highlights the current study ("This Article").

5.3. Analysis for CWE

CWE is a list of flaws that may result in security breaches and serves as a common language for describing these weaknesses, and provides a standard for measuring and

mitigating them. CWE is industry-endorsed and supported by a community of information security professionals [64]. CWE is free for public use and is maintained by the MITRE Corporation. It is closely related to common attack pattern enumeration and classification (CAPEC) and common vulnerabilities and exposures (CVEs) [65].

This section analyzes the EV smartphone applications for the presence of CWE. Table 6 provides a comparative analysis of CWE vulnerabilities by both groups of applications. The table is structured as follows: The first column on the left shows the application name and the group to which it belongs. The remaining columns show the identified CWE vulnerabilities that each application has. The last column lists the total number of CWEs present in each application. The details of the identified CWE are as follows:

1. CWE-250: Execution with unnecessary privileges.
2. CWE-330: Insufficiently random values are used and vulnerable to spoofing attacks.
3. CWE-276: Incorrect default permissions.
4. CWE-532: Insertion of sensitive data into the log file, passwords, tokens, or keys, into log files by mistake.
5. CWE-312: Cleartext storage of sensitive information.
6. CWE-89: Improper neutralization of special elements used in a SQL command (SQL injection).
7. CWE-327: Use of a compromised or risky cryptographic algorithm.
8. CWE-295: Improper certificate validation.
9. CWE-749: Exposed to dangerous methods or functions.
10. CWE-919: Improper handling of sensitive data, insecure communication, or lack of authentication.

Table 6. CWE present in EV mobile applications

Application	CWE-250	CWE-330	CWE-276	CWE-532	CWE-312	CWE-89	CWE-327	CWE-295	CWE-749	CWE-919	Total
TESLA [30]	✓	✓	✓	✗	✓	✓	✓	✓	✗	✓	8
Mercedes [31]	✗	✓	✓	✗	✓	✓	✗	✓	✓	✗	6
BMW [32]	✓	✓	✓	✗	✓	✓	✓	✗	✗	✓	7
Nissan [33]	✗	✗	✓	✗	✓	✓	✓	✗	✗	✗	4
Volkswagen [34]	✓	✗	✗	✗	✓	✓	✓	✗	✗	✓	5
FERRARI [35]	✗	✓	✓	✗	✓	✗	✓	✗	✗	✓	5
FordPass [36]	✗	✓	✗	✗	✓	✓	✗	✗	✓	✓	5
Mitsubishi [37]	✗	✓	✗	✓	✗	✗	✗	✗	✗	✗	2
Land Rover [38]	✗	✓	✓	✓	✓	✗	✓	✗	✗	✗	5
Jaguar [39]	✗	✗	✓	✓	✗	✗	✗	✗	✗	✗	2
EV Connect [40]	✗	✓	✓	✗	✓	✓	✓	✗	✗	✗	5
EVgo [41]	✗	✓	✓	✓	✓	✓	✓	✗	✓	✓	8
PlugShare [42]	✗	✓	✓	✓	✓	✗	✓	✓	✓	✓	8
ChargeHub [43]	✗	✓	✓	✓	✓	✓	✓	✗	✓	✗	7
Optiwatt [44]	✗	✗	✗	✓	✗	✗	✗	✗	✗	✗	1

Table 6. *Cont.*

Application	CWE-250	CWE-330	CWE-276	CWE-532	CWE-312	CWE-89	CWE-327	CWE-295	CWE-749	CWE-919	Total
Routeplanner [45]	✗	✓	✓		✓	✓	✓	✓	✗	✗	6
Caura [46]	✗	✗	✓	✓	✓	✗	✓	✗	✗	✗	4
BP pulse [47]	✗	✓	✓	✓	✓	✓	✓	✗	✗	✗	6
Electroverse [48]	✗	✓	✓	✗	✓	✓	✓	✗	✗	✗	5
Ev Energy [49]	✗	✓	✓	✗	✓	✓	✓	✗	✗	✗	5
Total	3	15	16	8	17	13	15	4	5	7	

6. Discussion

After a detailed analysis of Table 2, Figures 3 and 4, we conclude that some of the functionalities that are available in most third-party applications are F1 (charging points), F9 (payments), F10 (amenities), F11 (energy use tracking), and F12 (rewards and coupons). These feature sets are possible reasons for the users to rely on third-party applications, as need is not fulfilled by the applications provided by their EV manufacturer.

Table 7 summarizes each security technique along with their specific scenarios and potential performance impacts. This table provides a practical guide for EV app developers when choosing appropriate countermeasures.

Table 7. Security techniques, their use cases, and performance considerations.

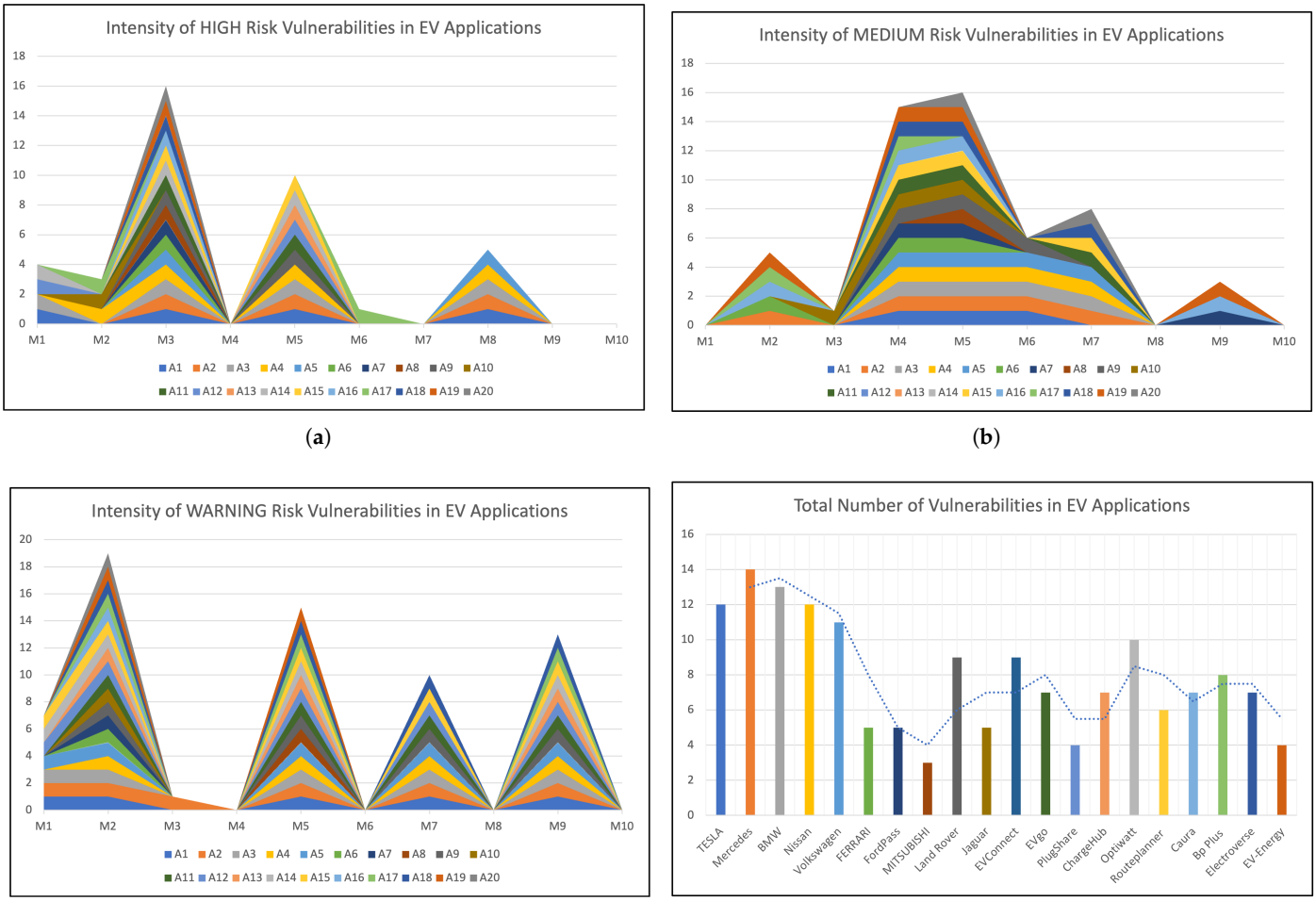
Security Technique	Scenario/Use Case	Performance Consideration
Use Save Cryptographic Storage	Save credentials, tokens, or sensitive user information more securely.	It requires a moderate level of CPU and memory, which may impact low-end devices.
Use input Fuzzing technique, e.g., libFuzzer and Hongfuzz	Find hidden vulnerabilities through malformed input testing	Basically, it is highly computationally exhaustive and is best suited for the testing phase.
Use a dynamic analysis tool like Frida, Xposed,	Identify runtime problems like privilege escalation or unauthorized access	It is based on real-time monitoring. Through this, there is minimal impact if used in dev/testing.
Align with OWASP Top 10 Secure Coding	Remove known web/mobile threats (e.g., M1: Improper Platform Usage)	Negligible runtime cost. It is a design time benefit.
Align with CWE and Mitigation	Address known software weaknesses such as CWE-89: SQL Injection etc	Usually applied at design time. It improves long-term maintainability.
Reduce Permission	By requesting only necessary permissions, we decrease the attack surface.	Improve user trust and privacy. It has no performance cost.
Long-term Update Mechanism	Ensure continued security patching and risk mitigation.	It depends on system capability, and may not be supported on legacy systems.

From Table 3 and Figure 6, we can identify patterns, such as the applications of Group 1 (EV manufacturers) tending to request more permissions and resources than the applications from Group 2 (third-party applications). *Group* 1 applications have more functionalities that require access to devices and data, which may also increase the risk of exposing sensitive information or compromising security, as *Group* 1 applications may request extended sensitive permissions and resources, including calling the phone and obtaining accounts. The most common permissions and resources requested by all applications are P1 (access fine location), P2 (camera), and P5 (external storage). Moreover, the least common permissions and resources requested by all applications are P3 (contacts), P4 (calendar), P6 (call phone), and P7 (get accounts). These permissions and resources may

also create a high risk to the user's privacy, security, or system stability, as they can access or modify the user's contacts, calendar, phone, or accounts.

From Figure 7, we can identify patterns, for example, the applications of *Group* 1 tend to have more vulnerabilities than the applications from *Group* 2, and have more diverse functionalities, which may introduce more security flaws and weaknesses. Also, *Group* 1 applications have more severe or critical vulnerabilities, such as V1 (improper platform usage), V3 (insecure communication), or V5 (insufficient cryptography).

Figure 7b,c show that the applications from *Group* 2 tend to have more medium- or warning-level vulnerabilities. However, Figure 7d shows that applications from *Group* 1 tend to have more high-level vulnerabilities than the applications from *Group* 2. This is because *Group* 1 applications have more specific or unique functionalities and features, which may be subject to more novel or sophisticated vulnerabilities.



(**a**) (**b**)

(**c**) (**d**)

Figure 7. Distribution of vulnerabilities across EV applications mapped with OWASP top ten categories based on risk level. (**a**) Intensity of high-risk vulnerabilities in EV applications. (**b**) Intensity of medium-risk vulnerabilities in EV applications. (**c**) Intensity of warning-risk vulnerabilities in EV applications. (**d**) Total number of vulnerabilities in EV applications.

As for Figure 7a, the most common vulnerabilities identified for all applications are V2 (insecure data storage), V3 (insecure communication), V5 (insufficient cryptography), and V9 (reverse engineering). However, these vulnerabilities can also pose a high risk to user privacy, security, or system stability, as they may allow attackers to access, modify, or decrypt sensitive data or analyze or manipulate the code.

Figure 8 provides a high-level overview of the number of CWE vulnerabilities in each application, using different colors to represent the variety of CWE. This means that we can easily compare applications based on their security weaknesses and identify the most common and severe CWE vulnerabilities among them. For example, we can see from the left that the Tesla application has CWE vulnerabilities, with 8 of 10. Also, CWE-312 (Sensitive information cleartext storage) and CWE-276 (Incorrect default permissions) are the most frequent CWE vulnerabilities in all applications.

Figure 9 shows comparative analysis permissions vs. apps, vulnerability risk levels by group, and CWE frequency. In the figure, you see that the quantitative summaries of *Group* 1 apps averaged total risk score is 6.4, while the *Group* 2 averaged total risk score is 4.8.

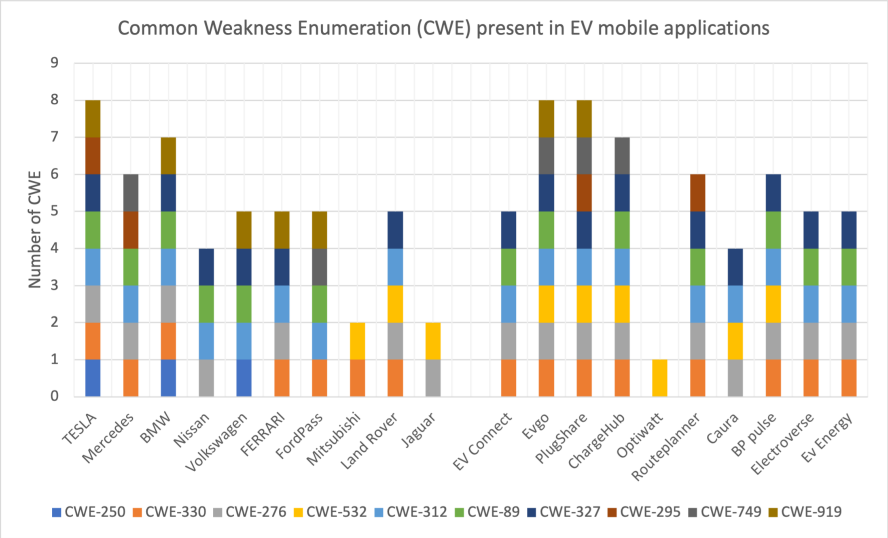


Figure 8. Common weakness enumeration (CWE) present in EV mobile applications.

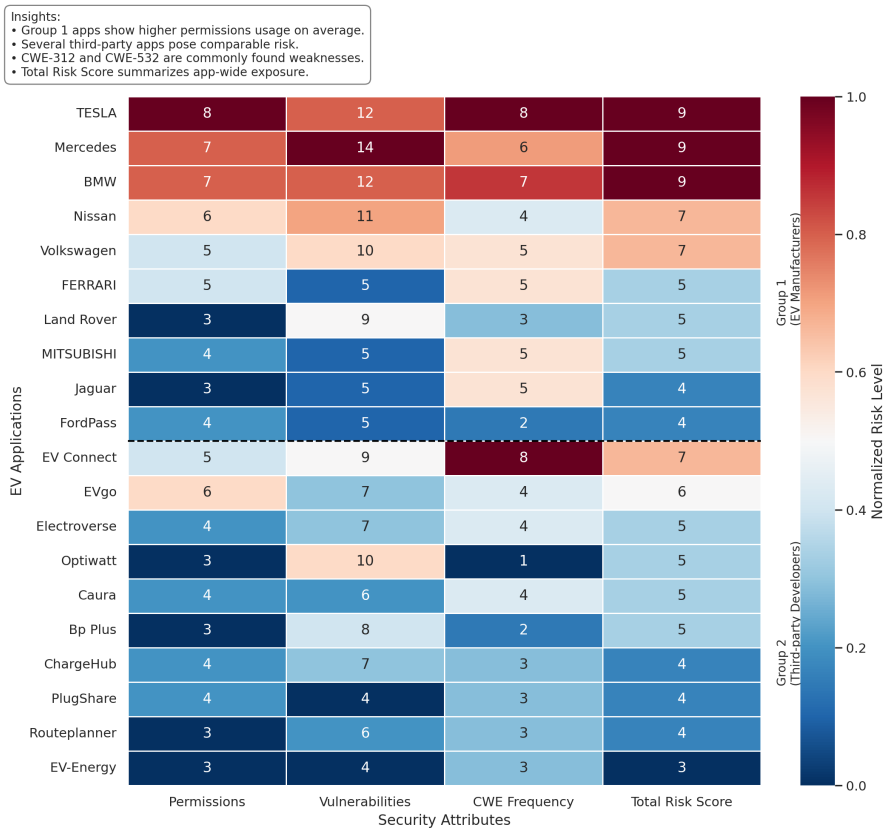


Figure 9. Comparative analysis permissions vs. apps, vulnerability risk levels by group, CWE frequency.

6.1. User Impact Analysis

It is also important to understand the potential real-world consequences for users if these vulnerabilities are exploited. If a malicious actor were to exploit these vulnerabilities, users could face a range of impacts. For instance, insecure data storage and transmission could lead to unauthorized access to sensitive user data, such as location history and payment details. This could result in privacy breaches and financial loss. Similarly, poor authentication mechanisms could allow unauthorized users to gain control of the vehicle's functions, posing significant safety risks.

6.1.1. Case Study: Insecure Data Handling in "PlugShare"

From Table 6, let us consider the "PlugShare" app for our case study. It is a widely used third-party Android app among EV owners. It provides functionalities such as locating charging stations and sharing reviews about them. However, it has a significant vulnerability—CWE-919: Improper handling of sensitive data, insecure communication, or lack of authentication.

The app collects sensitive user data, including location history and user reviews, but does not handle this data securely. This makes it susceptible to various security threats. For example, PlugShare users, use the app to find charging stations and share reviews. The app stores location history and reviews in an insecure manner. A malicious actor, if they manage to breach the app's database and gain access to the users' data, can potentially misuse this information, leading to privacy breaches.

Similarly, if someone uses public Wi-Fi networks to access the app, due to the app's insecure communication, a malicious actor present on the same network can perform a Man-in-the-Middle attack. The attacker can intercept the communication between a user's device and the app's servers, gaining access to the user's location, history, and reviews.

In both scenarios, the users face serious privacy risks due to the insecure data handling in the PlugShare app. This case study highlights the importance of secure coding practices, including secure data handling and communication, in protecting user data. It also underscores the need for robust authentication mechanisms to prevent unauthorized access to user data. These are areas where PlugShare, and similar apps, need to improve to provide a safer user experience.

6.1.2. Case Study: Insecure Personal Credentials in "ChargeHub"

Take another app "ChargeHub," from Table 6 for our next case study. This app is made by a third party that helps users to find charging points for electric vehicles, plan travel routes, and monitor charging sessions. However, it has a significant 516 vulnerability—CWE-522: Insufficiently Protected Credentials.

ChargeHub stores user login credentials and session information locally on the user's device with an unsecured encryption method. In that case, the phone is lost or compromised by malware or a virus. The attacker can easily access this information and gain unauthorized entry. Through this, attackers alter the information even without physical access.

In the given scenario, the user faces serious privacy issues due to an insecure encryption method. This case study highlights the implementation of security checksums on the backend and client side as well which cannot accept any SQL query, irrelevant to the query from the user. We need to implement secure user authentication, which includes two-factor authentication (2FA), biometrics, advanced encrypted methods like RSA, etc.

6.1.3. Case Study: Sensitive Cookie Exposure in HTTPS Session in "EVgo"

Let us consider the next app "EVgo" from Table 6 for this case study. EVgo helps users to find fast charging stations for EVs and manage their accounts. This app also has a

CWE-614 vulnerability, which is known as a sensitive cookie in an HTTPS session without the secure attribute.

During the evaluation, it was revealed that the cookies stored by the app are insecure and vulnerable. It would open doors and case session hijacking, especially when users access it from shared devices. Due to this weak session control, the attacker can exploit or gain access even when the user logs out. The EVgo app should have a secure cookie authentication mechanism like OAuth 2.0 with PKCE to minimize the session-related vulnerability risk.

6.2. Best Practices for Developers

In addition to our detailed analysis of the vulnerabilities in third-party applications, we believe it is key for developers to understand common security practices. While developing and publishing an application to the Google Play Store, a developer should be aware of vulnerabilities and security flaws. Our research indicates that while some developers follow best practices for secure coding, others may lack the necessary knowledge or resources to implement robust security measures. This disparity in security practices can lead to significant variations in the security posture of different third-party applications.

Common security shortcomings in third-party applications often stem from a lack of secure coding practices. For instance, developers may neglect to validate input properly, leading to vulnerabilities such as SQL injection or cross-site scripting. Insecure data storage and transmission are other common issues, often resulting from the improper use of cryptographic APIs or the absence of secure communication protocols.

To address these shortcomings, we suggest several best practices for third-party developers. First, developers should adhere to secure coding guidelines, such as those provided by OWASP or CWE. Regular code reviews and security testing can also help identify and fix potential vulnerabilities. Second, developers should implement robust authentication and authorization mechanisms to protect sensitive user data. Finally, developers should stay updated on the latest security threats and mitigation strategies, ensuring their applications can withstand new and emerging cyber threats.

By following these best practices, third-party developers can significantly improve the security of their EV mobile applications, providing a safer and more reliable experience for users.

Moreover, developers must use static and dynamic code analysis tools to identify and fix vulnerabilities. Also, users should be aware of the permissions and resources requested by the applications and only grant them if they are necessary, legitimate, and risk-free. Users should be cautious of applications that have more vulnerabilities, especially those that have high-level or critical vulnerabilities, as they may compromise the user's privacy and security. A person may not rely on most third-party applications, as they may have more medium- or warning-level vulnerabilities, which may indicate poor quality, insecure communication, code security, or inadequate use of cryptography.

Third-party developers use Secure Development Lifecycle (SDLC). By enabling these security considerations at each phase of development, a developer can identify possible risks before deployment. The first phase includes the requirements and design phase, and integrates security standards such as OWASP Mobile Top 10 and CWE, which guide developers to make the system more secure. The second phase includes the implementation phase: The developer follows secure coding practices, adds different security checksums, enforces permission cleanliness, and handles data after these integrations, which will be enforced to minimize common vulnerabilities. The next phase, known as the testing phase, detects vulnerabilities before deployment using tools such as MobSF, Frida, and fuzzing techniques that implement both static and dynamic analysis. These evaluations help ensure the robustness of the application before release. Finally, the last phase is the deployment

and maintenance phase, which includes continuous monitoring and reassessment using updated threat intelligence and security tools that can help developers maintain the long-term security of an application. Embedding these assessment techniques improves the overall security position of EV applications and aligns development practices with modern cybersecurity standards.

7. Defensive Measures

This section provides defensive measures for the identified vulnerabilities and weaknesses in the previous sections. It provides countermeasures that can be used to secure EV mobile applications and make them resilient against cyberattacks. There are two types of defenses presented as CWE defenses and OWASP defenses. Details of both are as follows:

7.1. CWE Defenses

This subsection is dedicated to providing details on the possible defenses that can be used to mitigate the impact of the CWEs identified in the applications. These mitigations are targeted and intended to use while developing an application, and before deployment.

1. CWE-250: Execution with unnecessary privileges. It is recommended to use the principle of least privilege to limit the privileges or permissions of the application [66]. Also, use user ID and process ID in containerization to isolate the application from the rest of the system or environment, and secure coding and security by design techniques to ensure that the application is safe.
2. CWE-330: Use of insufficiently random values. Developers can use cryptographically secure pseudorandom number generators (CSPRNGs) to generate random values for security purposes [67]. Also, it is recommended to use long, secure random values to resist brute-force or guessing attacks.
3. CWE-276: Incorrect default permissions. It can be essential to use secure file permissions and access control mechanisms and perform input sanitization for chmod, chown, and ACLs, among others.
4. CWE-532: Insertion of sensitive information into log files. Developers may use different data masking or data obfuscation techniques to protect sensitive information in log files and implement secure log libraries to write log files for the application, such as using Logback, SLF4J, or CocoaLumberjack [68].
5. CWE-312: Storage of sensitive information in clear text. It can be important to implement encryption at rest, hashing, and masking while storing sensitive information. Additionally, do not store any information in plain text on the device. Moreover, developers can use data minimization to reduce the amount or type of sensitive information that is stored.
6. CWE-89: Improper neutralization of special elements used in an SQL command ('SQL Injection'). This vulnerability can be mitigated by using parameterized queries or prepared statements to prevent SQL injection [69]. Similarly, the implementation of input validation and output encoding can be used to prevent SQL injection.
7. CWE-327: Use of a compromised or risky cryptographic algorithm. It is recommended to use resilient and reliable cryptographic algorithms or protocols to protect data. Additionally, implement secure key management and distribution mechanisms to protect keys and use mechanisms such as FIPS 140-2, NIST SP 800-57, or PKCS [70].
8. CWE-295: Improper certificate validation. Developers are encouraged to use SSL/TLS certificates from trusted authorities and verify trusted authorities, such as VeriSign, DigiCert, and Let's Encrypt. Similarly, they can use certificate pinning to prevent accepting expired or forged certificates.

9. CWE-749: Exposed dangerous method or function. It is important to implement secure authentication and authorization mechanisms to protect the interface and secure communication to protect the data.
10. CWE-919: Weaknesses in Mobile Applications. The application can be secured using the OWASP MSTG and the OWASP MASVS to test and verify the security of mobile applications.

7.2. OWASP Defenses

This subsection provides the guidelines and mitigation strategies for building applications without security vulnerabilities. The techniques can be used to address the OWASP security vulnerabilities identified in applications.

1. **M1: Improper use of the platform.** This can be addressed by using secure coding approaches [71], limiting sticky intentions, and restricting communication to only consent-based communication. Moreover, developers can limit the use of public intents and libraries, implement access control and authorization mechanisms, and check for possible misconfigurations.
2. **M2: Insecure data storage.** Developers can use standard encryption algorithms such as AES 256-bit and triple DES. They can also implement data security measures and harden the code using data masking, data obfuscation, or data minimization [72]. Finally, they can deploy multiple authorization checks that can grant or deny access to data in storage based on the user's identity.
3. **M3: Insecure communication.** Developers can use the SSL/TLS protocol to send and receive information and protect it from eavesdropping, tampering, and spoofing. They can verify the authenticity of SSL/TLS certificates before sending or receiving sensitive information. Finally, they should avoid sending sensitive information through SMS, email, and phone notifications.
4. **M4: Insecure authentication.** This vulnerability can be significantly addressed by conducting thorough tests, audits, and log data security measures with encryption, hashing, or masking. Moreover, it is required to authenticate on the server side and store data on the mobile device only after verification. Finally, encrypt client-side data when necessary, such as user preferences, session tokens, and keys.
5. **M5: Insufficient cryptography.** It is significantly important to implement strong encryption algorithms to protect sensitive data in transit or at rest, such as AES, RSA, or SHA-256. Developers need to implement long and secure keys for encryption and not store them in device or application code. Also, they need up-to-date encryption protocols. Encryption protocols such as SSL/TLS, HTTPS, or PGP. It is recommended to never use outdated algorithms and encryption protocols that are prone to weaknesses.
6. **M6: Insecure authorization.** It can be significantly helpful to implement multiple authorization checks to verify user permissions or roles. Also, developers can use secure authentication protocols, such as two-factor authentication with OAuth, or OpenID Connect. In addition, they can regularly update user permissions and roles and revoke unnecessary rights.
7. **M7: Poor code quality.** It is beneficial to use a code review and software quality assurance testing (SQA) to fix code security weaknesses or flaws, such as broken encryption, weak hashing, or data leaks [73]. This can be fixed by updating the application regularly and performing bug fixes. Moreover, this issue can be significantly resolved if a developer performs static and dynamic code analysis and fixes the identified issues.

8. **M8: Code tampering.** To get ahead of tampering, it is beneficial to use a code signing technique. This will increase the integrity of the code and it will also prevent code tampering and also it will stop dynamic code loading [74].
9. **M9: Reverse engineering.** This is very common in Android applications, as attackers can reverse an application to see and exploit its vulnerabilities. Therefore, it is required for developers to use different code obfuscation techniques and anti-debugging techniques that will make the code difficult to reverse engineer and to see its initial code [75,76].
10. **M10: Extraneous functionality.** Application developers can use a code-releasing mechanism when developing an application to ensure the quality and security of each interactional release. This will keep track of the introduced functionalities and mitigate the impact of extraneous functionality that becomes obsolete by simply removing it. Finally, developers can use different code monitoring and runtime analytics platforms to observe and improve application crashes and fix them in the next releases.

7.3. Long-Term Security Strategies

In addition to the immediate defensive measures, long-term strategies are very important to maintain the security of EV mobile applications. These strategies focus on the continuous improvement and adaptation to evolving threats.

One of the key strategies is continuous monitoring and updating. It is needed to regularly monitor the applications for any unusual activities or vulnerabilities and keep the applications updated with the latest security patches and improvements. This ensures that the applications are always equipped to handle the latest threats. Another important strategy is user education. Regularly educating users about the best practices for using the applications securely can go a long way in preventing security breaches. This includes guidance on permissions, secure communication, and data storage.

Additionally, regular security audits are also an essential part of maintaining long-term security. These audits help identify and fix any security vulnerabilities in the applications. Similarly, adapting to new security standards and guidelines, such as the updated OWASP Mobile Top 10, is another important strategy. This ensures that the applications are protected against the latest known threats and are following the best practices in the industry. Finally, incorporating security at every stage of the application development lifecycle is important. This includes secure coding practices, thorough testing, and post-deployment monitoring. By following a Secure Development Lifecycle (SDLC), we can ensure that security is not an afterthought but a fundamental part of the development process. By implementing these long-term strategies, application developers can ensure that EV mobile applications remain secure against evolving cyber threats.

8. Conclusions and Future Work

In this research, we performed a cybersecurity assessment of 20 EV applications that are available on the Google Play Store. The chosen applications were divided into two groups: one which is developed by EV manufacturers and the other by third-party developers. We implemented a rigorous methodology based on application permission, features, OWASP, and CWE frameworks to analyze these applications. The findings of our assessment have shown a range of security weaknesses in these applications. Moreover, we mapped these vulnerabilities against the OWASP and CWE lists. Finally, we provided defensive measures and recommendations to help developers improve their security and prevent cyberattacks.

In light of the recent release of the OWASP Mobile Top 10 2024, our future work will focus on updating our research to align with this new standard. We anticipate that this will

require us to conduct new experiments and collect fresh artifacts, thereby enriching our analysis and findings. By doing so, we aim to ensure that our research remains relevant and continues to contribute valuable insights into the cybersecurity landscape of Electric Vehicle Android applications.

To gain a deeper understanding of these risks, we plan to conduct user surveys and case studies in our future work. These surveys could help us understand the users' awareness of these vulnerabilities and their potential impacts.

Some key points are given below:

- This study investigated the vulnerabilities of 20 EV applications (10 Original Equipment Manufacturers (OEMs) and 10 third parties) using both static and dynamic analyses.
- We identified major security risks like weak authentication, inadequate encryption, and insecure communication. These flaws were mapped against OWASP Top 10 and CWE frameworks and showed that more permissions are required for OEM apps, while third-party apps have more vulnerable features due to poor coding practices and weaker communication mechanisms.
- Finally, we proposed a set of defensive measures based upon OWASP and CWE best practices to help mitigate the identified risks.

Our proposed defense measures address a broad spectrum of vulnerabilities identified in EV applications. However, there are challenges in resource-constrained settings, such as low-end electric vehicles. Techniques like secure cryptographic storage require higher processing power and memory, which damages devices that have fewer hardware capabilities. Additionally, long-term strategies may not be workable on a system without reliable update mechanisms. In the future, the focus will be on adopting these defenses into lightweight or integrating hardware security modules that protect our system.

To enhance our technical methodology, future work will explore more advanced dynamic analysis techniques. While MobSF offers important aspects through static and dynamic analysis, there are a lot of limitations, particularly in detecting complex runtime behaviors and vulnerabilities that exist at the root levels. To address these gaps, in future work, we include more refined fuzzing techniques such as libFuzzer and Honggfuzz, which generate abnormal inputs and reveal in-depth vulnerabilities. In addition, dynamic tools like Frida and Xposed can check app behavior on a real-time basis. These tools can show vulnerabilities such as privilege escalation and unauthorized information access, which may remain undetected in MobSF.

Moreover, similar research can be conducted by the Apple app store to identify cybersecurity challenges in EV applications developed for iPhone and other Apple devices. Finally, future work can be carried out to build a fully automated software toolkit that operates with less manual input and uses the underlying methodology to find vulnerabilities and weaknesses. This toolkit was initially developed using Python, and it has a wide range of libraries for automation and analysis. We try to integrate MobSF for static and dynamic analysis, and also use custom scripts to interpret security findings based on OWASP and CWE. In addition, the toolkit also incorporates Frida for runtime analysis and fuzzing frameworks such as libFuzzer or Honggfuzz for advanced vulnerability detection. The goal is to make a simple, lightweight system that is capable of automated APK scanning, finding security flaws, and generating an automated defense report on how to patch identified vulnerabilities.

Author Contributions: Conceptualization, A.R.; methodology, B.S. and M.A.H.; software, B.S.; validation, A.R. and Z.M.; formal analysis, B.S.; investigation, B.S.; resources, M.A.H.; data curation, B.S.; writing—original draft preparation, B.S. and A.R.; writing—review and editing, B.S. and Z.M.; visualization, B.S.; supervision, Z.M.; project administration, Z.M. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: The original contributions presented in this study are included in the article. Further inquiries can be directed to the corresponding author.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Lau, Y.Y.; Wu, A.Y.; Yan, M.W. A way forward for electric vehicle in Greater Bay Area: Challenges and opportunities for the 21st century. *Vehicles* **2022**, *4*, 420–432. [CrossRef]
2. Gelmanova, Z.; Zhabalova, G.; Sivyakova, G.; Lelikova, O.; Onishchenko, O.; Smailova, A.; Kamarova, S. Electric cars. Advantages and disadvantages. *J. Phys. Conf. Ser.* **2018**, *1015*, 052029. [CrossRef]
3. Weiss, M.; Cloos, K.C.; Helmers, E. Energy efficiency trade-offs in small to large electric vehicles. *Environ. Sci. Eur.* **2020**, *32*, 46.
4. Gray, S. 125 Million Electric Vehicles Will Be on the Road by 2030, Agency Says. *Fortune*, 31 May 2018. Available online: https://fortune.com/2018/05/31/electric-vehicles-international-energy-agency/ (accessed on 26 June 2025).
5. Schloter, L. Empirical analysis of the depreciation of electric vehicles compared to gasoline vehicles. *Transp. Policy* **2022**, *126*, 268–279. [CrossRef]
6. House, W. President Biden Announces Steps to Drive American Leadership Forward on Clean Cars and Trucks. *Fact Sheet*, 5 August 2021. Available online: https://bidenwhitehouse.archives.gov/briefing-room/statements-releases/2021/08/05/fact-sheet-president-biden-announces-steps-to-drive-american-leadership-forward-on-clean-cars-and-trucks/ (accessed on 26 June 2025).
7. IEA. Global Sales and Sales Market Share of Electric Cars, 2010–2021. 2023. Available online: https://www.statista.com/statistics/665774/global-sales-of-plug-in-light-vehicles/ (accessed on 15 November 2023).
8. Gnanavendan, S.; Selvaraj, S.K.; Dev, S.J.; Mahato, K.K.; Swathish, R.S.; Sundaramali, G.; Accouche, O.; Azab, M. Challenges, solutions and future trends in EV-technology: A review. *IEEE Access* **2024**, *12*, 17242–17260. [CrossRef]
9. Gong, J.; Tang, Z.; Yu, X.; Yu, L.; Cheng, H. Feasibility analysis of regional electric vehicle market—A case study of Panzhihua city. In Proceedings of the Computational Social Science: 2nd International Conference on New Computational Social Science (ICNCSS 2021), Suzhou, China, 15–17 October 2021; Taylor & Francis: Abingdon, UK, 2022; p. 81.
10. Kotia, N. EV Charging App: Types, Features, Process, Varieties. *Konstant Infosolutions Blog*, 31 August 2021. Available online: https://www.konstantinfo.com/blog/ev-charging-app/ (accessed on 26 June 2025).
11. Topman, N.; Adnane, A. Mobile applications for connected cars: Security analysis and risk assessment. In Proceedings of the NOMS 2022—2022 IEEE/IFIP Network Operations and Management Symposium, Budapest, Hungary, 25–29 April 2022; pp. 1–6.
12. Yang, K.H. Selling consumer data for profit: Optimal market-segmentation design and its consequences. *Am. Econ. Rev.* **2022**, *112*, 1364–1393. [CrossRef]
13. Saleem, B.; Ahmed, M.; Zahra, M.; Hassan, F.; Iqbal, M.A.; Muhammad, Z. A survey of cybersecurity laws, regulations, and policies in technologically advanced nations: A case study of Pakistan to bridge the gap. *Int. Cybersecur. Law Rev.* **2024**, *5*, 533–561. [CrossRef]
14. Martin, R.A.; Barnum, S. Common weakness enumeration (cwe) status update. *ACM SIGAda Ada Lett.* **2008**, *28*, 88–91. [CrossRef]
15. Bach-Nutman, M. Understanding the top 10 owasp vulnerabilities. *arXiv* **2020**, arXiv:2012.09960
16. Shirvani, S.; Baseri, Y.; Ghorbani, A. Evaluation framework for electric vehicle security risk assessment. *IEEE Trans. Intell. Transp. Syst.* **2023**, *25*, 33–56. [CrossRef]
17. Paverd, A.; Martin, A.; Brown, I. Modelling and Automatically Analysing Privacy Properties for Honest-but-Curious Adversaries; Technical Report, University of Oxford, Oxford, UK, 11 April 2014. Available online: https://www.cs.ox.ac.uk/people/andrew.paverd/casper/casper-privacy-report.pdf (accessed on 26 June 2025).
18. Vailoces, G.; Keith, A.; Almehmadi, A.; El-Khatib, K. Securing the Electric Vehicle Charging Infrastructure: An In-Depth Analysis of Vulnerabilities and Countermeasures. In Proceedings of the Int'l ACM Symposium on Design and Analysis of Intelligent Vehicular Networks and Applications, Montreal, QC, Canada, 30 October–3 November 2023; pp. 31–38.
19. Hanelt, A.; Nastjuk. Disruption on the way? The role of mobile applications for electric vehicle diffusion. In Proceedings of the Wirtschaftsinformatik, Osnabrück, Germany, 3 June 2015.

20. Stillwater, T.; Woodjack, J.; Nicholas, M. Mobile app support for electric vehicle drivers: A review of today's marketplace and future directions. In Proceedings of the International Conference on Human-Computer Interaction, Las Vegas, NV, USA, 21–26 July 2013; Springer: Berlin/Heidelberg, Germany, 2013; pp. 640–646.
21. Chatzoglou, E.; Kambourakis, G.; Kouliaridis, V. A Multi-Tier Security Analysis of Official Car Management Apps for Android. *Future Internet* **2021**, *13*, 58. [CrossRef]
22. Mandal, A.K.; Cortesi, A.; Ferrara, P.; Panarotto, F.; Spoto, F. Vulnerability analysis of android auto infotainment apps. In Proceedings of the 15th ACM International Conference on Computing Frontiers, Ischia, Italy, 8–10 May 2018; pp. 183–190.
23. Panarotto, F.; Cortesi, A.; Ferrara, P.; Mandal, A.K.; Spoto, F. Static analysis of android apps interaction with automotive can. In Proceedings of the Smart Computing and Communication: Third International Conference, SmartCom 2018, Tokyo, Japan, 10–12 December 2018; Proceedings 3; Springer: Berlin/Heidelberg, Germany, 2018; pp. 114–123.
24. Muhammad, Z.; Anwar, Z.; Saleem, B. A cybersecurity risk assessment of electric vehicle mobile applications: Findings and recommendations. In Proceedings of the 2023 3rd International Conference on Artificial Intelligence (ICAI), Islamabad, Pakistan, 22–23 February 2023; pp. 45–51.
25. Sarieddine, K.; Sayed, M.A.; Torabi, S.; Atallah, R.; Assi, C. Investigating the security of ev charging mobile applications as an attack surface. *ACM Trans.-Cyber-Phys. Syst.* **2023**, *7*, 1–28. [CrossRef]
26. Muhammad, Z.; Anwar, Z.; Saleem, B.; Shahid, J. Emerging cybersecurity and privacy threats to electric vehicles and their impact on human and environmental sustainability. *Energies* **2023**, *16*, 1113. [CrossRef]
27. Zhang, L.; Ma, D. *Evaluating Network Security Configuration (NSC) Practices in Vehicle-Related Android Applications*; Technical Report, SAE Technical Paper; SAE: Warrendale, PA, USA, 2024.
28. Alaba, S.Y.; Gurbuz, A.C.; Ball, J.E. Emerging Trends in Autonomous Vehicle Perception: Multimodal Fusion for 3D Object Detection. *World Electr. Veh. J.* **2024**, *15*, 20. [CrossRef]
29. Shahidinejad, A.; Abawajy, J. Blockchain-Based Self-Certified Key Exchange Protocol for Hybrid Electric Vehicles. *IEEE Trans. Consum. Electron.* **2023**, *70*, 543–553. [CrossRef]
30. Playstore. Tesla. 2023. Available online: https://play.google.com/store/search?q=tesla&c=apps&hl=en&gl=US (accessed on 26 June 2024).
31. Playstore. Mercedes Me Connect (USA). 2023. Available online: https://play.google.com/store/apps/details?id=com.mbusa.mmusa.android&hl=en&gl=US (accessed on 26 June 2024).
32. Playstore. BMW. 2023. Available online: https://play.google.com/store/search?q=MY%20BMW&c=apps&hl=en&gl=US (accessed on 26 June 2024).
33. Playstore. NissanConnect® Services. 2023. Available online: https://play.google.com/store/search?q=NissanConnect%C2%AE%20Services&c=apps&hl=en&gl=US (accessed on 26 June 2024).
34. Playstore. Volkswagen. 2023. Available online: https://play.google.com/store/search?q=Volkswagen&c=apps&hl=en&gl=US (accessed on 26 June 2024).
35. Playstore. 2023. Available online: https://www.ferrari.com/en-EN/auto/myferrari (accessed on 26 June 2024).
36. Playstore. 2023. Available online: https://play.google.com/store/apps/details?id=com.ford.fordpass&hl=en&gl=US (accessed on 26 June 2024).
37. Playstore. 2023. Available online: https://play.google.com/store/apps/details?id=com.mitsubishi_motors.remote_ps&hl=en_GB (accessed on 26 June 2024).
38. Playstore. 2023. Available online: https://play.google.com/store/apps/details?id=com.bosch.myspin.launcherapp_landrover&hl=en&gl=US (accessed on 26 June 2024).
39. Playstore. 2023. Available online: https://play.google.com/store/apps/details?id=com.bosch.myspin.launcherapp_jaguar&hl=en&gl=US (accessed on 26 June 2024).
40. Playstore. 2023. Available online: https://www.evconnect.com/ (accessed on 26 June 2024).
41. Playstore. EVGO. 2023. Available online: https://play.google.com/store/search?q=EVgo&c=apps&hl=en&gl=US (accessed on 26 June 2024).
42. Playstore. PlugShare—EV & Tesla Map. 2023. Available online: https://play.google.com/store/search?q=PLUGSHARE&c=apps&hl=en&gl=US (accessed on 26 June 2024).
43. Playstore. ChargeHub EV & Tesla Charging. 2023. Available online: https://play.google.com/store/search?q=ChargeHub&c=apps&hl=en&gl=US (accessed on 26 June 2024).
44. Playstore. 2023. Available online: https://play.google.com/store/apps/details?id=com.getoptiwatt.optiwatt&hl=en&gl=US (accessed on 26 June 2024).
45. Playstore. 2023. Available online: https://abetterrouteplanner.com/ (accessed on 26 June 2024).
46. Playstore. 2023. Available online: https://play.google.com/store/apps/details?id=com.caura.android&hl=en_IN&gl=GB (accessed on 26 June 2024).
47. Playstore. 2023. Available online: https://www.bpplus.com.au/ (accessed on 26 June 2024).

48. Playstore. 2023. Available online: https://electroverse.octopus.energy/ (accessed on 26 June 2024).
49. Playstore. 2023. Available online: https://www.ev.energy/ (accessed on 26 June 2024).
50. Rawal, H.; Parekh, C. Android Internal Analysis of APK by Droid_Safe & APK Tool. *Int. J. Adv. Res. Comput. Sci.* **2017**, *8*, 2397–2402.
51. Grushinskiy, M. XmlStarlet Command Line XML Toolkit User's Guide. Available online: https://xmlstar.sourceforge.net/doc/UG/xmlstarlet-ug.html (accessed on 2 March 2024).
52. Almomani, I.; Khayer, A. Android applications scanning: The guide. In Proceedings of the 2019 International Conference on Computer and Information Sciences (ICCIS), Sakaka, Saudi Arabia, 3–4 April 2019; pp. 1–5.
53. Amarante, J.; Barros, J.P. Exploring USB connection vulnerabilities on Android devices breaches using the Android debug bridge. In Proceedings of the 14th International Joint Conference on e-Business and Telecommunications, ICETE 2017, Madrid, Spain, 24–26 July 2017; SciTePress: Setúbal, Portugal, 2017; pp. 572–577.
54. Kusreynada, S.U.; Barkah, A.S. Android Apps Vulnerability Detection with Static and Dynamic Analysis Approach using MOBSF. *J. Comput. Sci. Eng. (JCSE)* **2024**, *5*, 46–63. [CrossRef]
55. Balikcioglu, P.G.; Sirlanci, M.A.; Kucuk, O.; Ulukapi, B.; Turkmen, R.K.; Acarturk, C. Malicious code detection in android: The role of sequence characteristics and disassembling methods. *Int. J. Inf. Secur.* **2023**, *22*, 107–118. [CrossRef]
56. Nath, A. *Packet Analysis with Wireshark*; Packt Publishing Ltd.: Birmingham, UK, 2015.
57. OLIVEIRA, L.C.C.A.d. Comparative Study of Techniques for Detecting Emulators on Android Devices. Bachelor's Thesis, Universidade Federal de Pernambuco (UFPE), Recife, Brazil, 2022.
58. LaMalva, G.; Schmeelk, S. MobSF: Mobile health care Android applications through the lens of open source static analysis. In Proceedings of the 2020 IEEE MIT Undergraduate Research Technology Conference (URTC), Cambridge, MA, USA, 9–11 October 2020; pp. 1–4.
59. Rehman, A.U.; Nadeem, A.; Malik, M.Z. Fair feature subset selection using multiobjective genetic algorithm. In Proceedings of the Genetic and Evolutionary Computation Conference Companion, Boston, MA, USA, 9–13 July 2022; pp. 360–363.
60. Anwar, C.; Hady, S.; Rahayu, N.; Kraugusteeliana, K. The Application of Mobile Security Framework (MOBSF) and Mobile Application Security Testing Guide to Ensure the Security in Mobile Commerce Applications. *J. Sistim Inf. Teknol.* **2023**, *5*, 97–102.
61. Khunt, A.R.; Prabu, P. An Empirical Analysis of Android Permission System Based on User Activities. *J. Comput. Sci.* **2018**, *14*, 324–333. [CrossRef]
62. Li, J. Vulnerabilities mapping based on OWASP-SANS: A survey for static application security testing (SAST). *arXiv* **2020**, arXiv:2004.03216 [CrossRef]
63. Alanda, A.; Satria, D.; Mooduto, H.; Kurniawan, B. Mobile application security penetration testing based on OWASP. *IOP Conf. Ser. Mater. Sci. Eng.* **2020**, *846*, 012036. [CrossRef]
64. Meng, H.; Thing, V.L.; Cheng, Y.; Dai, Z.; Zhang, L. A survey of Android exploits in the wild. *Comput. Secur.* **2018**, *76*, 71–91. [CrossRef]
65. Park, C.; Moon, S.Y.; Kim, R.Y.C. Detecting Common Weakness Enumeration (CWE) Based on the Transfer Learning of CodeBERT Model. *KIPS Trans. Softw. Data Eng.* **2023**, *12*, 431–436.
66. Williams, I. Evaluating a Method to Develop and Rank Abuse Cases Based on Threat Modeling, Attack Patterns and Common Weakness Enumeration. Ph.D. Thesis, North Carolina Agricultural and Technical State University, Greensboro, NC, USA, 2015.
67. Fischer, T. Testing cryptographically secure pseudo random number generators with artificial neural networks. In Proceedings of the 2018 17th IEEE International Conference on Trust, Security and Privacy in Computing and Communications/12th IEEE International Conference on Big Data Science and Engineering (TrustCom/BigDataSE), New York, NY, USA, 1–3 August 2018; pp. 1214–1223.
68. Cheng, F.; Cheng, F. The platform logging api and service. *Exploring Java 9: Build Modularized Applications in Java*; Apress: Berkeley, CA, USA, 2018; pp. 81–86.
69. Samarin, S.D.; Amini, M. Preventing SQL injection attacks by automatic parameterizing of raw queries using lexical and semantic analysis methods. *Sci. Iran. Trans. D Comput. Sci. Eng. Electr.* **2019**, *26*, 3469–3484.
70. Barker, E.B.; Barker, W.C.; Lee, A. *SP 800-21 Second edition. Guideline for Implementing Cryptography in the Federal Government*; National Institute of Standards & Technology: Gaithersburg, MD, USA, 2005.
71. Tran, A.D.; Nguyen, M.Q.; Phan, G.H.; Tran, M.T. Security Issues in Android Application Development and Plug-in for Android Studio to Support Secure Programming. In Proceedings of the Future Data and Security Engineering, Big Data, Security and Privacy, Smart City and Industry 4.0 Applications: 8th International Conference, FDSE 2021, Virtual Event, 24–26 November 2021; Proceedings 8; Springer: Berlin/Heidelberg, Germany, 2021; pp. 105–122.
72. Garcia, J.; Hammad, M.; Malek, S. Lightweight, obfuscation-resilient detection and family identification of android malware. *ACM Trans. Softw. Eng. Methodol. (TOSEM)* **2018**, *26*, 1–29. [CrossRef]
73. Nazir, M. Software Quality Assurance and Android Application Development: A Comparison among Traditional and Agile Methodology. *Lahore Garrison Univ. Res. J. Comput. Sci. Inf. Technol.* **2020**, *4*, 1–29.

74. Yin, Z.; Li, Z.; Cao, Y. A web application runtime application self-protection scheme against script injection attacks. In Proceedings of the Cloud Computing and Security: 4th International Conference, ICCCS 2018, Haikou, China, 8–10 June 2018; Revised Selected Papers, Part II 4; Springer: Berlin/Heidelberg, Germany, 2018; pp. 566–577.
75. Dong, S.; Li, M.; Diao, W.; Liu, X.; Liu, J.; Li, Z.; Xu, F.; Chen, K.; Wang, X.; Zhang, K. Understanding android obfuscation techniques: A large-scale investigation in the wild. In Proceedings of the Security and Privacy in Communication Networks: 14th International Conference, SecureComm 2018, Singapore, 8–10 August 2018; Proceedings, Part I; Springer: Berlin/Heidelberg, Germany, 2018; pp. 172–192.
76. Saad, M.; Taseer, M. The Study of the Anti-Debugging Techniques and their Mitigations. *Int. J. Electron. Crime Investig.* **2022**, *6*, 33–44.