NATIONAL LABORATORY OF THE ROCKIES

Blueprint: Open Worldwide Application Security Project Top 10 Common Weakness Enumerations for Electric Vehicle Supply Equipment

Myungsoo Jun, Jordan Smart, Katherine Amoresano, Erich Feth, Ryan Cryar, and Paul Snyder

National Laboratory of the Rockies

The National Laboratory of the Rockies is a national laboratory of the U.S. Department of Energy, Office of Critical Minerals and Energy Innovation, operated under Contract No. DE-AC36-08GO28308.

This report is available at no cost from the National Laboratory of the Rockies (NLR) at www.nlr.gov/publications.

Technical Report
NLR/TP-5T00-96284
February 2026

NATIONAL LABORATORY OF THE ROCKIES

Blueprint: Open Worldwide Application Security Project Top 10 Common Weakness Enumerations for Electric Vehicle Supply Equipment

Myungsoo Jun, Jordan Smart, Katherine Amoresano, Erich Feth, Ryan Cryar, and Paul Snyder

National Laboratory of the Rockies

Suggested Citation
Jun, Myungsoo, Jordan Smart, Katherine Amoresano, Erich Feth, Ryan Cryar, and Paul Snyder. 2026. *Blueprint: Open Worldwide Application Security Project Top 10 Common Weakness Enumerations for Electric Vehicle Supply Equipment*. Golden, CO: National Laboratory of the Rockies. NLR/TP-5T00-96284.
https://www.nlr.gov/docs/fy26osti/96284.pdf.

The National Laboratory of the Rockies is a national laboratory of the U.S. Department of Energy, Office of Critical Minerals and Energy Innovation, operated under Contract No. DE-AC36-08GO28308.

This report is available at no cost from the National Laboratory of the Rockies (NLR) at www.nlr.gov/publications.

Technical Report
NLR/TP-5T00-96284
February 2026

National Laboratory of the Rockies
15013 Denver West Parkway
Golden, CO 80401
303-275-3000 • www.nlr.gov

NOTICE

This work was authored by the National Laboratory of the Rockies for the U.S. Department of Energy (DOE), operated under Contract No. DE-AC36-08GO28308. Funding provided by the U.S. Department of Energy Office of Critical Minerals and Energy Innovation Vehicle Technologies Office. The views expressed herein do not necessarily represent the views of the DOE or the U.S. Government.

This report is available at no cost from the National Laboratory of the Rockies (NLR) at www.nlr.gov/publications.

U.S. Department of Energy (DOE) reports produced after 1991 and a growing number of pre-1991 documents are available free via www.OSTI.gov.

Cover photos (clockwise from left): Josh Bauer, National Laboratory of the Rockies 61725; Visualization from National Laboratory of the Rockies Insight Center; Getty-181828180; Agata Bogucka, National Laboratory of the Rockies 91683; Dennis Schroeder, National Laboratory of the Rockies 51331; Werner Slocum, National Laboratory of the Rockies 67842.

The National Laboratory of the Rockies prints on paper that contains recycled content.

List of Acronyms

API	application programming interface
CWE	common weakness enumeration
DNS	Domain Name System
EVSE	electric vehicle supply equipment
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
IEC	International Electrotechnical Commission
ISO	International Organization for Standardization
JTAG	Joint Test Action Group
MQTT	Message Queuing Telemetry Transport
NIST	National Institute of Standards and Technology
OCPP	Open Charge Point Protocol
OSCP	Open Smart Charge Protocol
OWASP	Open Worldwide Application Security Project
PHP	Hypertext Preprocessor
PKI	public key infrastructure
PLCC	Plastic Leaded Chip Carrier
PnC	Plug&Charge
RFID	radio-frequency identification
SHA	Secure Hash Algorithm
SQL	Structured Query Language
SSL	Secure Sockets Layer
TCP	Transmission Control Protocol
TLS	Transport Layer Security
UID	unique identifier
URI	Uniform Resource Identifier
USB	Universal Serial Bus
ZAP	Zed Attack Proxy

This report is available at no cost from the National Laboratory of the Rockies (NLR) at www.nlr.gov/publications.

Table of Contents

1 Introduction 1
2 Top 10 Security Weaknesses in EVSE 2
2.1 Communication Network Security 2
2.1.1 Example Exploitations 2
2.1.2 Prevention 3
2.1.3 Related CWEs 3
2.2 Authentication and Authorization 4
2.2.1 Example Exploitations 4
2.2.2 Prevention 4
2.2.3 Related CWEs 5
2.3 Firmware and Software 5
2.3.1 Prevention 6
2.3.2 Example Exploitations 6
2.3.3 Related CWEs 6
2.4 Physical Tampering and Unauthorized Access 7
2.4.1 Prevention 7
2.4.2 Example Exploitations 7
2.4.3 Related CWEs 8
2.5 Backend Cloud Systems 8
2.5.1 Example Exploitations 8
2.5.2 Prevention 9
2.5.3 Related CWEs 9
2.6 Payment Systems 10
2.6.1 Example Exploitations 10
2.6.2 Prevention 10
2.6.3 Related CWEs 11
2.7 Web Applications and API 11
2.7.1 Example Exploitations 11
2.7.2 Prevention 12
2.7.3 Related CWEs 12
2.8 Integration With Power Grid 12
2.8.1 Example Exploitations 13
2.8.2 Prevention 13
2.8.3 Related CWEs 13
2.9 Data Security and Privacy 14
2.9.1 Example Exploitations 14
2.9.2 Prevention 14
2.9.3 Related CWEs 14
2.10 Supply Chain 15
2.10.1 Example Exploitations 15
2.10.2 Prevention 16
2.10.3 Related CWEs 16
3 Recommendations, Best Practices, and Mitigation Strategies to EVSE Industry 17
3.1 CWE-20: Improper Input Validation 17
3.2 CWE-319: Cleartext Transmission of Sensitive Information 18
3.3 CWE-494: Download of Code Without Integrity Check 19
3.4 CWE-798: Use of Hard-Coded Credentials, CWE-1392: Use of Default Credentials 20
3.5 CWE-287: Improper Authentication 21

This report is available at no cost from the National Laboratory of the Rockies (NLR) at www.nlr.gov/publications.

3.6 CWE-312: Cleartext Storage of Sensitive Information 24
3.7 CWE-639: Authorization Bypass Through User-Controlled Key 25
3.8 CWE-77: Improper Neutralization of Special Elements Used in a Command ("Command Injection") 27
3.9 CWE-294: Authentication Bypass by Capture-Replay 28
4 Summary and Conclusions 30
References 31
Appendix: Summary of Related CWEs 32

List of Figures

Figure 1. Example Python code with CWE-20 with no input validation 17
Figure 2. Example remediation of CWE-20 by validating input 18
Figure 3. Example Python code with CWE-319 transmitting sensitive information without encryption 19
Figure 4. Example remediation of CWE-319 by using SSL/TLS 19
Figure 5. Example Python code with CWE-494 downloading code without integrity check 19
Figure 6. Example remediation of CWE-494 by using HTTPS and SHA-256 hash check 20
Figure 7. Example Python code with CWE-798 using hard-coded, default credentials 21
Figure 8. Example remediation of CWE-798 by prompting the user at run time 21
Figure 9. Example Python code with CWE-287 that provides credentials through client-side cookies 22
Figure 10. Example remediation of CWE-287 with server-side session management 24
Figure 11. Example Python code with CWE-312 with cleartext storage of sensitive information 24
Figure 12. Example Python code with CWE-312 remedying cleartext storage of sensitive information 25
Figure 13. Example Python code with CWE-639 that lacks an authentication mechanism 26
Figure 14. Authorization token requirement to protect against CWE-693 27
Figure 15. Example Python code with CWE-77 by using a shell command directly for the user input 28
Figure 16. Example remediation of CWE-77 by using a list of arguments and avoiding invoking a shell 28
Figure 17. Example Python code with CWE-294 by using no encryption and replayable requests 28
Figure 18. Example remediation of CWE-294 by using a time stamp to prevent replay 29

List of Tables

Table 1. Communication Network Security—Related CWEs 3
Table 2. Authentication and Authorization—Related CWEs 5
Table 3. Firmware and Software—Related CWEs 6
Table 4. Physical Tampering and Unauthorized Access—Related CWEs 8
Table 5. Backend Cloud Systems—Related CWEs 9
Table 6. Payment Systems—Related CWEs 11
Table 7. Web Applications and API—Related CWEs 12
Table 8. Integration With Power Grid—Related CWEs 13
Table 9. Data Security and Privacy—Related CWEs 14
Table 10. Supply Chain—Related CWEs 16

This report is available at no cost from the National Laboratory of the Rockies (NLR) at www.nlr.gov/publications.

Introduction

The increasing penetration of electric vehicles (EVs) has prompted a rapid expansion of the EV charging infrastructure across various environments, including residential areas, multiple-dwelling units, public parking spaces, commercial establishments, and dedicated charging hubs. As charging infrastructure is growing at a rapid pace, cybersecurity of these devices are critical and should not be overlooked.

As charging stations evolved from simple power outlets to sophisticated networked devices, they have become increasingly interconnected with vehicles, backend management systems, and even the power grid itself. Although this interconnectedness enables advanced functionalities such as remote monitoring, smart charging, and payment processing, it simultaneously introduces the potential for cybersecurity vulnerabilities. These vulnerabilities not only pose risks to individual users but also present potential threats to critical infrastructure, such as the stability and reliability of the power grid, should these devices be manipulated at scale.

This report categorizes the top 10 security weaknesses in electric vehicle supply equipment (EVSE) using a similar approach by the Open Worldwide Application Security Project (OWASP) Top 10, and it describes example exploitations, methods for prevention, and related common weakness enumerations (CWEs) for each category. The weaknesses listed in this report could lead to security issues or vulnerabilities in EVSE if they are exploited by an attacker to compromise the confidentiality, integrity, or availability of a system. This report also recommends EVSE industry best practices and remediations for the identified weaknesses to use as guidelines for their secure products and systems.

This report is available at no cost from the National Laboratory of the Rockies (NLR) at www.nlr.gov/publications.

1 Top 10 Security Weaknesses in EVSE

OWASP is an open community dedicated to enabling organizations to develop, purchase, and maintain trusted software. OWASP releases the OWASP Top 10 as an awareness document for developers to use to ensure that their software does not contain these weaknesses or vulnerabilities. The list is based on a survey from the community and data from developers. The survey asks which vulnerabilities are believed to be important by the community but might not be reflected in the data yet. The data collected from developers include the time period, the total number of applications tested in the dataset, and the list of CWEs with counts of how many applications contained that CWE.

This report provides the top 10 security weaknesses in EVSE by benchmarking the OWASP Top 10. After a thorough survey of reports and articles regarding EVSE security weaknesses, we produced the top 10 security weaknesses in EVSE based on:

- How frequent the weakness is reported.
- How serious the potential impact of the weakness is if it is exploited.
- How wide the potential impact of the weakness is if it is exploited.

1.1 Communication Network Security

Charging stations are interconnected with many elements throughout the network, including vehicles, backend management systems, and mobile devices. This interconnectedness introduces a wide range of attack surfaces and cybersecurity vulnerabilities. Further, the attacks through these surfaces can spread to other elements throughout the network and could result in significant consequences with large-scale deployments.

Various communication protocols—including Open Charge Point Protocol (OCPP), Open Smart Charging Protocol (OSCP), Message Queuing Telemetry Transport (MQTT), and even Modbus—are used to provide advanced functionalities, such as remote monitoring/control, smart charging management, and payment processing. Many EVSEs, especially older models, might use outdated or unencrypted communication protocols by default, such as unencrypted Hypertext Transfer Protocol (HTTP) or Modbus. Even protocols designed for security, such as OCPP and International Organization for Standardization (ISO) 15118, can have insecure implementations, failing to properly use Transport Layer Security (TLS) or public key infrastructure (PKI). This can lead to sensitive data and control commands being easily captured or intercepted by attackers.

Insecure implementations of communication protocols can also lead to the exploitation of firmware theft and manipulation. Attackers can manipulate proprietary firmware or operational data, leading to the malicious reprogramming of chargers and the injection of malware into the network. The consequences of these protocol-level vulnerabilities include widespread service disruption, significant data theft, compromise of data integrity, unauthorized control over charging sessions, and the potential for broader network intrusions.

1.1.1 Example Exploitations

- **Interception of communication packets:** An attacker intercepts the communications between an EV charger and its backend system, between the EV and the charger, or

This report is available at no cost from the National Laboratory of the Rockies (NLR) at www.nlr.gov/publications.

between a user and a server to attempt a man-in-the-middle attack. If the communications are unencrypted (e.g., using an insecure OCPP profile or unencrypted HTTP), the attacker could read sensitive user data or payment information, or it could manipulate charging commands. Further, these attacks could spoof Domain Name System (DNS) entries to redirect charger traffic to a rogue server. The impacts of these attacks include the theft of sensitive data (payment information, user credentials, vehicle identification numbers), unauthorized control over charging sessions (e.g., free charging, manipulated billing), or the injection of malicious commands.

- **Injection of excessive traffic or malformed request:** An attacker floods an EV charger, a charging site, or the central management system with excessive traffic or malformed requests, overwhelming its resources and interrupting service to attempt a denial-of-service attack. This can be achieved by exploiting vulnerabilities in communication protocols or backend systems. The impacts of these attacks include disrupting charging services (no charging), leading to long queues, stranded vehicles, and lost revenue for operators, as well as undermining public trust. In severe cases, coordinated denial-of-service attacks on multiple chargers could destabilize the power grid, leading to blackouts.

1.1.2 Prevention

EVSE manufacturers should adopt communication protocols with strong security mechanisms, follow proper implementation procedures, and validate implementations with conformance testing. If EVSE manufacturers use open-source libraries, they should validate that the libraries are securely implemented.

Disabling unnecessary network services and implementing strict access control to essential services through firewalls and robust authentication mechanisms are recommended practices to prevent cyberattacks through communication networks.

1.1.3 Related CWEs

Table 1. Communication Network Security—Related CWEs

CWE	Description
CWE-1428: Reliance on HTTP Instead of HTTPS	The product provides or relies on the use of HTTP communications when Hypertext Transfer Protocol Secure (HTTPS) is available.
CWE-319: Cleartext Transmission of Sensitive Information	The product transmits sensitive or security-critical data in cleartext in a communication channel that can be sniffed by unauthorized actors.
CWE-294: Authentication Bypass by Capture-Replay	A capture-replay flaw exists when the design of the product makes it possible for a malicious user to sniff network traffic and bypass authentication by replaying it to the server in question to the same effect as the original message (or with minor changes).
CWE-290: Authentication Bypass by Spoofing	This attack-focused weakness is caused by incorrectly implemented authentication schemes that are subject to spoofing attacks.

This report is available at no cost from the National Laboratory of the Rockies (NLR) at www.nlr.gov/publications.

CWE	Description
CWE-400: Uncontrolled Resource Consumption	The product does not properly control the allocation and maintenance of a limited resource.
CWE-115: Misinterpretation of Input	The product misinterprets an input, whether from an attacker or another product, in a security-relevant fashion.

1.2 Authentication and Authorization

Attackers can leverage several weaknesses to access and control EVSE through improperly configured authentication and authorization mechanisms. Authentication mechanisms are used to validate the identity of a user, whereas authorization mechanisms determine which resources users should have access to. Authentication weaknesses include devices that are shipped with hard-coded credentials that are the same for each installation of the product and usually cannot be changed by equipment administrators, whereas authorization weaknesses include improper access controls where the equipment does not restrict or incorrectly restricts access to a resource from an unauthorized user. Both weaknesses can lead to attacks where an attacker gains unintended access to a system to disrupt services or leak sensitive data.

1.2.1 Example Exploitations

- An EVSE manufacturer uploads code to a public GitHub repository with credentials to a public object storage service. The attacker searches the repository, finds the credentials, and steals all data from this endpoint.
- An EVSE is deployed without changing the default administrative credentials. An attacker can log in with the default credentials and perform administrative functions on the equipment.
- An EVSE is running a web server that is serving a database file containing the credentials of the administrative user. The directory on the web server does not have any permissions, so the attacker can download and decrypt the database file and obtain the administrative password.
- An EVSE is running a web server that uses Hypertext Preprocessor (PHP) scripts to update the administrative credentials. The permissions for these scripts are improperly configured, and an attacker can run these scripts via a direct request and give themselves access to the system.

1.2.2 Prevention

Private and other sensitive information, such as passwords, should be stored in a protected file or an encrypted database. Developers should use the principle of least privilege when determining the access restrictions of users to reduce the attack surface. New EVSEs have default usernames and passwords that are initially configured by the EVSE manufacturers. These default usernames and passwords must be changed to a strong password upon installation to prevent improper access to the chargers.

Developers should divide their applications into different areas for different access levels (e.g., (administrator, registered user, anonymous). Use role-based access control to enforce roles with appropriate boundaries. Industry-standard cryptographic algorithms should be used to protect locally stored credentials. Developers should take extra care to ensure that their applications use

This report is available at no cost from the National Laboratory of the Rockies (NLR) at www.nlr.gov/publications.

proper access control mechanisms on the server side on every page. Caching should be disabled for pages containing sensitive information, and they should be accessible only with an active and authenticated session token.

1.2.3 Related CWEs

Table 2. Authentication and Authorization—Related CWEs

CWE	Description
CWE-1392: Use of Default Credentials	The product uses default credentials (such as passwords or cryptographic keys) for potentially critical functionality.
CWE-327: Use of a Broken or Risky Cryptographic Algorithm	The product uses a broken or risky cryptographic algorithm or protocol.
CWE-284: Improper Access Control	The product does not restrict or incorrectly restricts access to a resource from an unauthorized actor.
CWE-285: Improper Authorization	The product does not perform or incorrectly performs an authorization check when an actor attempts to access a resource or perform an action.
CWE-287: Improper Authentication	When an actor claims to have a given identity, the product does not prove or insufficiently proves that the claim is correct.
CWE-732: Incorrect Permission Assignment for Critical Resource	The product specifies permissions for a security-critical resource in a way that allows that resource to be read or modified by unintended actors.
CWE-862: Missing Authorization	The product does not perform an authorization check when an actor attempts to access a resource or perform an action.
CWE-863: Incorrect Authorization	The product performs an authorization check when an actor attempts to access a resource or perform an action, but it does not correctly perform the check.
CWE-522: Insufficiently Protected Credentials	The product transmits or stores authentication credentials, but it uses an insecure method that is susceptible to unauthorized interception and/or retrieval.
CWE-295: Improper Certificate Validation	The product does not validate, or incorrectly validates, a certificate.
CWE-798: Use of Hard-Coded Credentials	The product contains hard-coded credentials, such as a password or cryptographic key.
CWE-522: Insufficiently Protected Credentials	The product transmits or stores authentication credentials, but it uses an insecure method that is susceptible to unauthorized interception and/or retrieval.

1.3 Firmware and Software

Firmware and software attacks allow adversaries to bypass traditional security measures and potentially compromise entire EV charging systems. Firmware is a low-level software that controls a device's hardware, and software is a type of computer program that tells the computer—in this case, an EV charging point—what to do. Attacks on an EV charger's firmware can disable chargers, alter basic functions, exfiltrate data, or enable remote control. Software can harm computer systems and cause data theft. These vulnerabilities in unison could cause a loss of

This report is available at no cost from the National Laboratory of the Rockies (NLR) at www.nlr.gov/publications.

service for the EV chargers (Delarea 2024). Software and firmware attacks can also occur through supply chain compromise.

1.3.1 Prevention

For these types of attacks, EV charging stakeholders should ensure that firmware and software updates are securely signed and delivered to prevent an attacker from maliciously changing the firmware to a modified one. Implementing a secure boot process ensures the integrity of the charging station's firmware and software. The secure boot process verifies the new firmware and software by checking digital signatures against a database of trusted keys. If a signature does not match a trusted key, the bootloader (a small program that initiates the startup process of a computer, loading the operating system or other firmware into memory) will prevent the new firmware or software from loading. Ultimately, this prevents unauthorized or compromised code from being executed. Additionally, continuous firmware and software updates, secure communication protocols, strong authentication protocols, and ongoing monitoring will aid in keeping EV chargers secure.

1.3.2 Example Exploitations

- **Remote code execution:** Remote unauthenticated users connected to the same local network as the EV charger controller can execute arbitrary commands with elevated privileges on affected devices. This vulnerability applies to cPH2 charging stations with eCharge Salia Plastic Leaded Chip Carrier (PLCC) firmware versions ≤2.0.4 produced by eCharge Hardy Barth GmbH. (Kaiser 2024). The web administrative interface runs on PHP with many of the PHP scripts being makeshift scripts. Lighttpd, a web server for high-traffic websites that is used in eCharge Hardy Barth EV chargers, is configured to enforce authentication, but it does not apply to vulnerable PHP scripts; therefore, unauthenticated remote command execution over the local area network is possible (Kaiser 2024).
- **Heap-based buffer overflow:** If there is an integer overflow vehicle-to-grid transport protocol implementation, this vulnerability leads to a heap overflow. An attacker can run arbitrary code on the Linux process that can lead to bypassing the payment gate for charging, compromise private keys stored in the charging stations, and communicate with the vendor's backend using OCPP by impersonating the compromised charging station.
- **Authorization bypass:** An unauthenticated attacker can obtain the EV charger version and firmware upgrading history by knowing the charger identification. Once this is known, the attacker can use this information to see which current vulnerabilities the EV charger has based on its software or firmware version.

1.3.3 Related CWEs

Table 3. Firmware and Software—Related CWEs

CWE	Description
CWE-77: Improper Neutralization of Special Elements Used in a Command ("Command Injection")	The product constructs all or part of a command using an externally influenced input from an upstream component, but it does not neutralize or it incorrectly neutralizes special elements that could modify the intended command when it is sent to a downstream component.

This report is available at no cost from the National Laboratory of the Rockies (NLR) at www.nlr.gov/publications.

CWE	Description
CWE-122: Heap-Based Buffer Overflow	A heap overflow condition is a buffer overflow, where the buffer that can be overwritten is allocated in the heap portion of the memory, generally meaning that the buffer was allocated using a routine such as malloc().
CWE-190: Integer Overflow or Wraparound	The product performs a calculation that can produce an integer overflow or wraparound when the logic assumes that the resulting value will always be larger than the original value. This occurs when an integer value is incremented to a value that is too large to store in the associated representation. When this occurs, the value could become a very small or negative number.
CWE-639: Authorization Bypass Through User-Controlled Keys	The system's authorization functionality does not prevent one user from gaining access to another user's data or record by modifying the key value identifying the data.
CWE-494: Download of Code Without Integrity Check	The product downloads the source code or an executable from a remote location and executes the code without sufficiently verifying the origin and integrity of the code.
CWE-1277: Firmware Not Updateable	The product does not provide its users with the ability to update or patch its firmware to address any vulnerabilities or weaknesses that might be present.

1.4 Physical Tampering and Unauthorized Access

Even if a charging station is situated behind a firewall, an attacker connecting physically via the charging cable might gain direct access to internal services due to misconfigurations in the charging station controller, underscoring the need for layered security that considers both network and physical access vectors.

1.4.1 Prevention

Securing the ports of EVSE can prevent many physical attacks by narrowing the attack vectors that a bad actor can use. Some ports, such as diagnostic ports or an Ethernet switch's port, must be open to debug problems or use the EVSE entirely. These ports need to be secured by implementing some kind of authentication method or encryption of the traffic flowing through an Ethernet switch. Other ports that are not being used can be disabled entirely to prevent attacks through this vector.

Shielding the EV-EVSE charging cable reduces interference from outside sources. Although most interference would be from unknown sources, leading to minor unintended outcomes, an adversary might exploit unshielded cables to launch targeted attacks on EVs and EVSE.

1.4.2 Example Exploitations

- **Unauthorized access:** In many cases, EVSE are constructed using many connected circuit boards communicating over a shared Ethernet network. By plugging into an unprotected Ethernet switch, bad actors can intercept inter-module traffic, much of which is unencrypted, leading to possible exploits being found or, even worse, payment PKI information being released. Unprotected Universal Serial Bus (USB) ports can be

This report is available at no cost from the National Laboratory of the Rockies (NLR) at www.nlr.gov/publications.

exploited to copy logs of recent users' authentication information and OCPP server log-in credentials.

- **Gaining privileged access via physical connection:** An attacker gains privileged access to EVSE via physical connection—e.g., Joint Test Action Group (JTAG), human-machine interface, USB, local wireless, serial—to disable the EVSE.
- **Connection sniffing:** An attacker gains access via ISO 15118 sniffing, EV-EVSE connection sniffing, or a man-in-the-middle attack.
- **Denial of service:** Attackers can launch attacks wirelessly by exploiting an unshielded charging cable's ability to ack as antennas. The attacker can inject a busy single into the EV and/or EVSE by sending a signal to the charging cable on a specific channel. The EV and/or EVSE will wait for the busy signal to resolve, which in some current protocols does not happen by default. In some cases, the charging session must be restarted, but in most cases, the EV and/or EVSE need to be manually reset, leading to the EVSE being offline until the reset can occur.

1.4.3 Related CWEs

Table 4. Physical Tampering and Unauthorized Access—Related CWEs

CWE	Description
CWE-306: Missing Authentication for Critical Function	The product does not perform any authentication for functionality that requires a provable user identity, or it consumes a significant amount of resources.
CWE-1263: Improper Physical Access Control	The product is designed with access restricted to certain information, but it does not sufficiently protect against an unauthorized actor with physical access to these areas.
CWE-284: Improper Access Control	The product does not restrict or incorrectly restricts access to a resource from an unauthorized actor.

1.5 Backend Cloud Systems

Backend cloud management systems are responsible for processing consumption data, authenticating users, managing billing, supervising EV charging, and controlling the configuration of charging stations. Attacks on these backend systems or the cloud portals used by customers can result in large data breaches and widespread operational disruption and manipulation.

Vulnerabilities in backend systems frequently serve as initial access points for more extensive attacks, allowing attackers to move laterally and escalate privileges within the operator's or manufacturer's broader infrastructure. Attackers often gain this initial access by exploiting traditional cyberattack techniques, such as phishing, leveraging generic software vulnerabilities, or exploiting open port connections. The concentration of control and data in these systems means that a successful attack can have disproportionately massive consequences across the entire EV charging ecosystem.

1.5.1 Example Exploitations

- **Phishing and malware deployment:** An attacker gains initial network access via phishing and use malware to gain access to the backend server, which can result in

This report is available at no cost from the National Laboratory of the Rockies (NLR) at www.nlr.gov/publications.

consequences such as data theft or even system lock down. This attack type can result in widespread service outages, leading to potentially significant financial losses and damage to operator reputation and public trust.

- **Injection of malicious code or compromised firmware:** An attacker gains access to the backend server by injecting malicious code or compromised firmware through an exposed application programming interface (API) endpoint or Structured Query Language (SQL) injection flaw in a web application.

1.5.2 Prevention

A high level of security for backend cloud systems—such as advanced threat intelligence, anomaly detection, stringent access controls, or comprehensive data encryption—could prevent them from attacks. Further, although technical controls are vital, the reliance on traditional attack vectors like phishing indicates that human vulnerabilities remain a persistent entry point; therefore, a comprehensive security strategy should integrate robust human-centric defenses, including continuous security awareness training, strong email security protocols, and mandatory multifactor authentication for all administrative and user access.

1.5.3 Related CWEs

Table 5. Backend Cloud Systems—Related CWEs

CWE	Description
CWE-650: Trusting HTTP Permission Methods on the Server Side	The server contains a protection mechanism that assumes that any Uniform Resource Identifier (URI) that is accessed using HTTP GET will not cause a state change to the associated resource. This might allow attackers to bypass the intended access restrictions and conduct resource modification and deletion attacks because some applications allow GET to modify the state.
CWE-200: Exposure of Sensitive Information to an Unauthorized Actor	The product exposes sensitive information to an actor that is not explicitly authorized to access that information.
CWE-79: Improper Neutralization of Input During Web Page Generation ("Cross-Site Scripting")	The product does not neutralize or incorrectly neutralizes user-controllable input before it is placed in output that is used as a webpage that is served to other users.
CWE-78: Improper Neutralization of Special Elements Used in an OS Command ("OS Command Injection")	The product constructs all or part of an operating system command using externally influenced input from an upstream component, but it does not neutralize or incorrectly neutralizes special elements that could modify the intended operating system command when it is sent to a downstream component.
CWE-400: Uncontrolled Resource Consumption	The product does not properly control the allocation and maintenance of a limited resource.
CWE-115: Misinterpretation of Input	The product misinterprets an input, whether from an attacker or another product, in a security-relevant fashion.
CWE-269: Improper Privilege Management	The product does not properly assign, modify, track, or check privileges for an actor, creating an unintended sphere of control for that actor.

This report is available at no cost from the National Laboratory of the Rockies (NLR) at www.nlr.gov/publications.

1.6 Payment Systems

The ability to exchange payment for electricity to charge an EV is the basic function of an EVSE. Several mechanisms can be used to verify the customer and pass this information to the charge network operator to process the payment, such as plug-and-charge, radio-frequency identification (RFID) cards, or smartphone apps. All these systems rely on some sort of authentication method, such as certificates using a PKI or a unique identifier (UID), such as RFID UID cards. Although most of these methods are seen as secure, in practice there can be some gaps in their security when they are incorrectly implemented.

1.6.1 Example Exploitations

- **Connection sniffing:** Bad actors could steal valid certificates issued by a valid root certificate authority. The EVSE manufacturer would then revoke this certificate to prevent the certificate from being used. Due to inconsistent revocation polices, an EV could still see this certificate as being valid and pass payment and other personally identifiable information to the EVSE using this certificate, and a bad actor could intercept this information.
- **RFID cloning:** A device could be used to copy RFID tags. These copy tags could be used at EVSE to charge an EV while the payment is charged to the customer associated with the original RFID tag.
- **Sensitive information storage:** A bad actor with physical access to an EVSE could copy authorization caches from a microSD on the EVSE controller to bill any transaction to other customers by pulling an RFID UID from the microSD. These RFID UIDs should be scrubbed from the authorization cache.

1.6.2 Prevention

PKI is used for verification, encryption, and authentication throughout the EVSE infrastructure. Although PKI has been used for many years in other protocols, such as HTTPS, the adaption to charging infrastructure could result in implementation gaps on a large scale, especially with conflicting policies and stakeholder interactions. Solutions to gaps include creating and implementing a unified certificate policy for all root hierarchies, improving the certificate revocation policies, and creating life cycle management.

The RFID authentication method typically includes a UID stored on the RFID card or other RFID token. This UID is used by the payment system to bill the customer that is associated with the UID, but these UIDs can be easily copied with an RFID device that is capable of reading an RFID token and writing to a new card or token. Transitioning to plug-and-charge prevents the potential of physical copying, enabling a more secure option of paying.

Some EVSE controllers are equipped with a microSD card to store payment authorization caches for transactions completed with the EVSE. Although this cache has been seen to store information used for logging or debugging, it also stores sensitive payment information, such as information from other EVSE or RFID UIDs, which can be used to repeatedly bill a transaction. These caches should be scrubbed of all payment information to prevent unauthorized billing

This report is available at no cost from the National Laboratory of the Rockies (NLR) at www.nlr.gov/publications.

1.6.3 Related CWEs

Table 6. Payment Systems—Related CWEs

CWE	Description
CWE-299: Improper Check for Certificate Revocation	The software does not correctly verify that the certificate has been revoked before using the certificate for trusted communications, which can allow for sharing sensitive encrypted information.
CWE: Improper Certificate Validation	An application does not validate all portions of a certificate before trusting it, such as the host name, which can lead to trusting malicious certificates.
CWE-294: Authentication Bypass by Capture-Replay	An RFID token can be captured and replayed, allowing for a bad actor to start a charge using an account that does not belong to them.
CWE-312: Cleartext Storage of Sensitive Information	Sensitive information, such as authentication data, is stored without proper sanitation or encryption.

1.7 Web Applications and API

Many charging network operators provide mobile or web services to pay utilities, monitor charging status, check charging station availability, or even remotely manage charging. These services offer convenience and smart functionalities to users but introduce more attack surfaces at the same time. Attacks through weak security of the web and mobile applications associated with charging services may propagate to other systems or services, which can lead to serious and widespread impact on overall charging systems.

Mobile applications used in EV ecosystem are also reported to have security vulnerabilities. Jun, Cryar, and Markel (2024) investigated the security attributes and vulnerabilities of a sample of mobile apps that support key user functions in the EV ecosystem and showed that all analyzed apps have security risks, categorized as high or medium or both. Therefore, even if the charging stations themselves possess some level of security, weak security in the associated web and mobile applications can provide attackers with alternative pathways into the wider EV charging ecosystem.

1.7.1 Example Exploitations

- **Gaining access using default credentials or a weak password policy:** An attacker discovers an EV charger's web interface or a backend server API endpoint that uses default credentials or has a weak password policy. They use publicly available information (such as hard-coded credentials) or brute force techniques to gain administrative access. Once authenticated, they can reconfigure charging station settings, access sensitive operational data, or retrieve proprietary information.
- **SQL injection:** An attacker identifies a web application vulnerability, such as an SQL injection flaw, in the charge point operator's user management portal or an API endpoint that handles user data. They exploit this flaw to gain unauthorized access to the backend database, exfiltrating vast amounts of sensitive customer data, including names, email addresses, payment information, and charging history.

This report is available at no cost from the National Laboratory of the Rockies (NLR) at www.nlr.gov/publications.

1.7.2 Prevention

Developers of web interfaces or API should integrate security from inception by applying secure-by-design principles, such as cyber-informed engineering, from the earliest stages of development for all web applications, APIs, and firmware that interact with EVSE. This includes considering security for all components, software, network configurations, and data management processes from the outset. Strengthening authentication and authorization mechanisms, ensuring secure data handling and storage, implementing robust input validation and error handling, and ensuring secure configuration management are some measures that should be considered in development.

All software, including third-party libraries and open-source components, should be regularly reviewed for potential weaknesses. Automated tools can be used to scan for outdated components with known vulnerabilities.

1.7.3 Related CWEs

Table 7. Web Applications and API—Related CWEs

CWE	Description
CWE-319: Cleartext Transmission of Sensitive Information	The product transmits sensitive or security-critical data in cleartext in a communication channel that can be sniffed by unauthorized actors.
CWE-798: Use of Hard-Coded Credentials	The product contains hard-coded credentials, such as a password or a cryptographic key.
CWE-1188: Initialization of a Resource With an Insecure Default	The product initializes or sets a resource with a default that is intended to be changed by the administrator, but the default is not secure.
CWE-89: Improper Neutralization of Special Elements Used in an SQL Command ("SQL Injection")	The product constructs all or part of an SQL command using externally influenced input from an upstream component, but it does not neutralize or incorrectly neutralizes special elements that could modify the intended SQL command when it is sent to a downstream component. Without sufficient removal or quoting of SQL syntax in user-controllable inputs, the generated SQL query can cause those inputs to be interpreted as SQL instead of ordinary user data.
CWE-1021: Improper Restriction of Rendered UI Layers or Frames	The web application does not restrict or incorrectly restricts frame objects or user interface (UI) layers that belong to another application or domain, which can lead to user confusion about which interface the user is interacting with.
CWE-494: Download of Code Without Integrity Check	The product downloads source code or an executable from a remote location and executes the code without sufficiently verifying the origin and integrity of the code.

1.8 Integration With Power Grid

Connecting EV chargers to the grid creates vulnerabilities surrounding grid stability because of the increased and potentially unpredictable electric load they place on the grid. Uncontrolled charging can cause sudden load changes, creating a significant spike in demand and making it difficult for grid operators to maintain stability. An attack on multiple chargers could lead to

This report is available at no cost from the National Laboratory of the Rockies (NLR) at www.nlr.gov/publications.

potential blackouts by destabilizing the grid. Further, if many vehicles are charging in the same area, it could cause localized stress on the grid and lead to potential outages in that region. Ultimately, integrating charging points with the grid increases the amount of cybersecurity risks that can impact grid stability.

1.8.1 Example Exploitations

- **Oscillations in grid frequency:** By switching a large group of EV chargers on and off from the grid simultaneously, attackers have the potential to induce grid oscillations. Attacks of this nature, if large enough, can destabilize the grid frequency and force generators to trip, which can lead to a power outage.
- **False data injection:** These attacks tamper with the grid's measurement by introducing malicious or false data into the EV charging system. This threatens power grid stability by causing load peaks or drops, which can lead to blackouts, if done at scale.

1.8.2 Prevention

Proactive monitoring and incident response can detect abnormal energy usage, unauthorized charging, or suspicious network activity in real time and promptly and effectively manage cybersecurity incidents, including those that could impact grid stability.

It is a good practice to follow established cybersecurity standards and regulations—such as the National Institute of Standards and Technology (NIST) IR 8473 (Cybersecurity Framework Profile for Electric Vehicle Extreme Fast Charging Infrastructure), ISO 27001, SOC 2 Type II, and International Electrotechnical Commission (IEC) 62443/62351—to prevent power grid vulnerabilities because these frameworks provide a structured approach to managing cybersecurity risks in critical infrastructure.

Additionally, it is good practice to implement smart charging techniques that optimize charging schedules to align with periods of low demand, thus reducing the strain on the grid during peak demand periods. Smart charging consists of load shifting and optimized charging schedules. During load shifting, strain on the grid during peak demand periods is reduced by shifting EV charging to off-peak hours. To optimize charging schedules, algorithms can improve charging schedules to minimize energy costs and maximize the use of renewable energy sources.

1.8.3 Related CWEs

Table 8. Integration With Power Grid—Related CWEs

CWE	Description
CWE-94: Improper Control of Generation of Code ("Code Injection")	The product constructs all or part of a code segment using externally influenced input from an upstream component, but it does not neutralize or incorrectly neutralizes special elements that could modify the syntax or behavior of the intended code segment.
CWE-20: Improper Input Validation	The product receives input or data, but it does not validate or incorrectly validates that the input has the properties that are required to safely and correctly process the data.

This report is available at no cost from the National Laboratory of the Rockies (NLR) at www.nlr.gov/publications.

1.9 Data Security and Privacy

Data security and privacy vulnerabilities in EVSE are primarily caused by weaknesses in how sensitive information is collected, transmitted, stored, and managed across the interconnected components in EV ecosystems. Insecure data transmission or data storage can result in data breaches, leading to identity theft, financial fraud (e.g., unauthorized transactions using stolen payment information), privacy violations, and reputational damage for the charging network operator.

1.9.1 Example Exploitations

- An attacker gains unauthorized access to a charging station or its backend system, potentially through insecure web interfaces or physical tampering. Then they manipulate the consumption data reported by the charger, falsifying meter readings to enable free charging or to generate incorrect invoices for legitimate users. The impacts of this attack include financial losses for operators, violations of legal regulations, and service interruptions affecting billing processes and interoperability with partner charging networks.
- EV charging applications and stations collect precise location data, driving patterns, and charging habits. If these datapoints are exposed due to insecure storage or transmission, attackers can aggregate the data to create detailed profiles of individuals' movements and daily routines. This information can then be used for surveillance, stalking, or other malicious activities.
- An attacker exploits vulnerabilities in poorly secured smartphone applications that are used to manage customer charging sessions. This could involve insecure authentication, weak session management, or data leakage flaws, leading to unauthorized access to a user's account.

1.9.2 Prevention

Data security vulnerabilities caused by insecure data transmission and storage can be prevented by using end-to-end encryption, secure communication protocols, and data encryption. For secure data storage and handling, it is a good measure to collect and store only the data that are strictly necessary for the operation of the charging station and to limit the retention period of the user data to the shortest time possible while still meeting legal and business requirements.

1.9.3 Related CWEs

Table 9. Data Security and Privacy—Related CWEs

CWE	Description
CWE-200: Exposure of Sensitive Information to an Unauthorized Actor	The product exposes sensitive information to an actor that is not explicitly authorized to have access to that information.
CWE-312: Cleartext Storage of Sensitive Information	The product stores sensitive information in cleartext within a resource that might be accessible to another control sphere.
CWE-359: Exposure of Private Information ("Privacy Violation")	The product does not properly prevent a person's private, personal information from being accessed by actors that either (1) are not explicitly authorized to access the

This report is available at no cost from the National Laboratory of the Rockies (NLR) at www.nlr.gov/publications.

CWE	Description
	information or (2) do not have the implicit consent of the person about whom the information is collected.
CWE-778: Insufficient Logging	When a security-critical event occurs, the product either does not record the event or omits important details about the event when logging it.
CWE-639: Authorization Bypass Through User-Controlled Key (Insecure Direct Object Reference)	The system's authorization functionality does not prevent one user from gaining access to another user's data or record by modifying the key value identifying the data.
CWE-522: Insufficiently Protected Credentials	The product transmits or stores authentication credentials, but it uses an insecure method that is susceptible to unauthorized interception and/or retrieval.

1.10 Supply Chain

This is a weakness introduced during the development, manufacturing, delivery, or integration process. The supply chain for EV charging infrastructure involves numerous third-party hardware manufacturers, software developers, and service providers, which allows for multiple ways in which a bad actor could manipulate the supply chain. This complexity creates multiple points where vulnerabilities can be introduced, such as insecure components and manufacturing practices, unpatched third-party software and libraries, and poor vendor support and life cycle issues.

1.10.1 Example Exploitations

- **Back-doored firmware from third-party vendor:** A third-party vendor hides a backdoor in firmware and supplies it to an EVSE manufacturer. This compromised firmware can be used for the remote takeover of chargers, the manipulation of charging behavior, or the theft of data. This may also include unintentional exploitation in which a third-party includes a firmware backdoor for legitimate monitoring purposes but doesn't document this protocol with the manufacturer.
- **Tampered or counterfeit chips with malicious embedded codes:** An attacker compromises or replaces legitimate components, often at the manufacturing or integration stage, with tampered ones containing hidden, malicious logic or embedded code. Malicious chips can stealthily execute unauthorized functions, for example, charging behavior manipulation, payment and authentication compromise, network intrusion, firmware and OTA update alteration, etc.
- **Supply chain compromise of SDK or library:** An attacker compromises a third-party SDK or open-source libraries used in EVSE firmware or backend services by injecting malicious code into the GitHub repository or a package distribution server. If the compromised codes are incorporated into the EVSE SW build, the chargers or backend systems that use it become infected. The injected code can perform stealthy, unauthorized transmission of sensitive information and unauthorized command execution from the central system, resulting in remote manipulation of charging sessions, disabling firmware updates, pushing rogue configurations, or pivoting into backend networks that manage EVSE operations.

This report is available at no cost from the National Laboratory of the Rockies (NLR) at www.nlr.gov/publications.

1.10.2 Prevention

Preventing supply chain vulnerabilities in EVSE requires a comprehensive, multilayered strategy that addresses both physical security and cybersecurity risks while considering broader logistical dependencies and geopolitical factors. This includes:

- Rigorously vetting vendors and performing in-field monitoring.
- Having an understanding of a product's software bill-of-materials (SBOM) and hardware bill-of-materials (HBOM).
- Implementing secure-by-design principles or cyber-informed engineering across the supply chain.
- Ensuring secure software and firmware management by implement strong code signing/verification, continuous auditing codes and logs, etc.
- Complying with industry standards and regulations.

1.10.3 Related CWEs

Table 10. Supply Chain—Related CWEs

CWE	Description
CWE-494: Download of Code Without Integrity Check	The product downloads source code or an executable from a remote location and executes the code without sufficiently verifying the origin and integrity of the code.
CWE-1277: Firmware Not Updateable	The product does not provide its users with the ability to update or patch its firmware to address any vulnerabilities or weaknesses that might be present.
CWE-1104: Use of Unmaintained Third-Party Components	The product relies on third-party components that are not actively supported or maintained by the original developer or a trusted proxy for the original developer.
CWE-913: Improper Control of Dynamically Managed Code Resources	The product does not properly restrict reading from or writing to dynamically managed code resources, such as variables, objects, classes, attributes, functions, or executable instructions or statements.
CWE-912: Hidden Functionality	The product contains functionality that is not documented, not part of the specification, and not accessible through an interface or command sequence that is obvious to the product's users or administrators.

This report is available at no cost from the National Laboratory of the Rockies (NLR) at www.nlr.gov/publications.

2 Recommendations, Best Practices, and Mitigation Strategies to EVSE Industry

This section provides some example codes that have vulnerabilities related to EVSEs, and how to test and remediate these vulnerabilities. Users of this guide can use the example code, testing, and remediations as a guideline when developing secure software and firmware for EVSE or when building backend servers.

Related CWEs in each category of the top 10 security weaknesses in EVSE are listed in the appendix. This list is not complete; there could be more CWEs related to each category. But this list can provide information on what CWEs are related to EVSE security weaknesses and potential remediations/mitigations for the CWEs. The MITRE Corporation. (n.d.) posts information on the complete list of CWEs, descriptions, common consequences, and potential mitigations, and the EVSE industry can refer to this as a security guideline for their products.

2.1 CWE-20: Improper Input Validation

Improper input validation usually occurs when a software application receives input or data but fails to correctly verify that the input possesses the necessary properties for safe and accurate processing. Common consequences of the vulnerability include altered program flow, reading confidential data, unauthorized resource control, or even arbitrary code execution.

Figure 1 shows an example Python script that has improper input validation. The example code assumes that user_id is numeric and does not check whether the value is numeric or not; thus, the code crashes if an attacker sends a non-numeric value for the variable. If the code is connected to a database, this attack could enable SQL injection. This can be remediated by validating inputs, as illustrated in Figure 2.

```
from flask import Flask, request

app = Flask(__name__)

@app.route('/user')
def get_user():
    user_id = request.args.get('id') # No validation!
    return f"Looking up user with ID: {int(user_id)}" # Potential crash or injection

app.run()
```

Figure 1. Example Python code with CWE-20 with no input validation

This report is available at no cost from the National Laboratory of the Rockies (NLR) at www.nlr.gov/publications.

```python
from flask import Flask, request, abort

app = Flask(__name__)

@app.route('/user')
def get_user():
    user_id = request.args.get('id')

    if not user_id or not user_id.isdigit():
        abort(400, description="Invalid user ID.")

    user_id = int(user_id)
    return f"Looking up user with ID: {user_id}"

app.run()
```

Figure 2. Example remediation of CWE-20 by validating input

Testing for improper input validation involves systematically feeding unexpected, malformed, or malicious data into the application's input fields and observing their behavior. Injection testing tools, such as SQLmap, OWASP Zed Attack Proxy (ZAP), or Postman, can be used for automated security testing for improper input validation by feeding unexpected, malformed, or malicious data for specific input validation or brute force random data for broader validation.

2.2 CWE-319: Cleartext Transmission of Sensitive Information

Cleartext transmission of sensitive information happens commonly when sensitive or security-critical data are transmitted over a communication channel without encryption. This can lead to data breaches, identity theft, and financial losses, as attackers can easily read and exploit the exposed information.

An example Python code containing the cleartext transmission of sensitive information vulnerability is shown in Figure 3. The example code represents a client that sends a username and password to a server over a plain Transmission Control Protocol (TCP) socket without any encryption. This vulnerability can be prevented by implementing robust encryption and secure communication practices. Figure 4 illustrates an example of how to fix the vulnerability by using an encrypted connection (Secure Sockets Layer [SSL]).

The vulnerability can be tested using a packet sniffer tool—such as Wireshark, tcpdump, or Bur Suite—to capture TCP traffic and search for the plaintext password string.

This report is available at no cost from the National Laboratory of the Rockies (NLR) at www.nlr.gov/publications.

```python
# Client sending a password over an unencrypted connection (TCP)
import socket

def send_password(password):
    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    s.connect(('example.com', 8000))  # Not using encryption
    s.sendall(password.encode('utf-8'))
    s.close()

send_password("SuperSecret123")
```

Figure 3. Example Python code with CWE-319 transmitting sensitive information without encryption

```python
import ssl
import socket

def send_password_secure(password):
    context = ssl.create_default_context()
    with socket.create_connection(('example.com', 443)) as sock:
        with context.wrap_socket(sock, server_hostname='example.com') as ssock:
            ssock.sendall(password.encode('utf-8'))
```

Figure 4. Example remediation of CWE-319 by using SSL/TLS

2.3 CWE-494: Download of Code Without Integrity Check

This weakness refers to downloading executable code (e.g., Python modules, scripts, or binaries) from a remote source and executing it without verifying its integrity or authenticity. An example code shown in Figure 5 uses HTTP without encryption or authentication and runs downloaded code without checking the integrity or authenticity.

To fix the issues and prevent the vulnerability, the script should download over HTTPS to prevent man-in-the-middle attacks and perform a Secure Hash Algorithm (SHA)-256 hash check to ensure no tampering of the code. A sample remediation of the code is illustrated in Figure 6.

```python
# Downloading and executing remote code without integrity check
import urllib.request

url = "http://example.com/remote_script.py"  # Note: HTTP, no integrity
response = urllib.request.urlopen(url)
code = response.read().decode()

exec(code)  # Executing unverified code
```

Figure 5. Example Python code with CWE-494 downloading code without integrity check

This report is available at no cost from the National Laboratory of the Rockies (NLR) at www.nlr.gov/publications.

```python
import urllib.request
import hashlib

url = "https://example.com/remote_script.py"
expected_sha256 = "8e4d6f2e8e9285c9b9f30e3e6749e2a37ecddf6c43e64c6d13fc71e9b8e78d5d"

# Download code
response = urllib.request.urlopen(url)
code = response.read()

# Validate integrity
sha256 = hashlib.sha256(code).hexdigest()
if sha256 != expected_sha256:
    raise ValueError("Downloaded file integrity check failed!")

# Execute only if hash matches
exec(code.decode())
```

Figure 6. Example remediation of CWE-494 by using HTTPS and SHA-256 hash check

Analyzing how the application handles remote code downloads and verifying the absence of integrity checks can test for this vulnerability. Packet sniffing tools such as Wireshark can be used to capture network traffic to inspect how the application performs while downloading code. Also, a manual code review can be performed to identify the download functions, trace the execution flow, and look for the absence of cryptographic functions between the download and the execution of the code.

2.4 CWE-798: Use of Hard-Coded Credentials, CWE-1392: Use of Default Credentials

This vulnerability is found in the product containing hard-coded, default credentials, such as a password or a cryptographic key. Once an attacker obtains this hard-coded key, they have a permanent backdoor to access and control any EVSE instance that still uses this default credential, bypassing the intended security measures.

A vulnerable example code that uses default, hard-coded credentials is shown in Figure 7. If this gets into production or is reused elsewhere, it creates an easy backdoor for attackers. To prevent this vulnerability, credentials should not be hard-coded in the source code; instead, they could be securely provided by using environment variables, securely prompting for credentials at run time, using secret management tools. A remediation by prompting the user at run time is illustrated in Figure 8.

This report is available at no cost from the National Laboratory of the Rockies (NLR) at www.nlr.gov/publications.

```python
# Insecure: Uses default credentials
import requests

USERNAME = "admin"
PASSWORD = "admin" # Default credential (hard-coded)

def login():
    response = requests.post("https://example.com/api/login", data={
        "username": USERNAME,
        "password": PASSWORD
    })
    print("Login status:", response.status_code)

login()
```

Figure 7. Example Python code with CWE-798 using hard-coded, default credentials

```python
import getpass
import requests

username= input("Enter username: ")
password=getpass.getpass("Enter password: ") # Hidden input

def login():
    response = requests.post("https://example.com/api/login", data={
        "username": username,
        "password": password
    })
    print("Login status:", response.status_code)

login()
```

Figure 8. Example remediation of CWE-798 by prompting the user at run time

Tools like Shodan or Nmap scripts often look for devices running with default credentials. Tools like Burp Suite Intruder can be used to attempt to log in by trying common credentials. And tools such as Hydra, Medusa, or Ncrack can be used for brute force/default credential testing.

2.5 CWE-287: Improper Authentication

Improper authentication occurs when a product improperly or does not at all validate the identity of an actor. The following code in Figure 9 is an example of improper authentication where credentials are provided through client-side cookies, which can be forged. Figure 10 shows the remediated code that avoids trusting the client-side cookies for authentication and instead uses a server-side session file to track authenticated users.

This report is available at no cost from the National Laboratory of the Rockies (NLR) at www.nlr.gov/publications.

```
import cgi
import http.cookies
import os
import sys

def AuthenticateUser(username, password):
    # Replace this with actual authentication logic
    return username == "admin" and password == "secret"

def DoAdministratorTasks():
    # Replace this with actual admin tasks
    print("Content-Type: text/plain\n")
    print("Performing administrator tasks...")

form = cgi.FieldStorage()
cookie = http.cookies.SimpleCookie(os.environ.get("HTTP_COOKIE", ""))

loggedin = cookie.get("loggedin")
user_cookie = cookie.get("user")

if not loggedin or loggedin.value != "true":
    username = form.getfirst("username")
    password = form.getfirst("password")
    if not AuthenticateUser(username, password):
        print("Error: you need to log in first")
        sys.exit(0)
    else:
        # Set cookies
        print("Set-Cookie: loggedin=true")
        print(f"Set-Cookie: user={username}")

# After cookie check, fetch updated user value
user = user_cookie.value if user_cookie else form.getfirst("username", "")

if user == "Administrator":
    DoAdministratorTasks()
```

Figure 9. Example Python code with CWE-287 that provides credentials through client-side cookies

```
import cgi
import http.cookies
import os
import sys
import uuid
import json

SESSION_DIR = "/tmp/cgi_sessions"

def init_session_dir():
    if not os.path.exists(SESSION_DIR):
        os.makedirs(SESSION_DIR, exist_ok=True)
```


This report is available at no cost from the National Laboratory of the Rockies (NLR) at www.nlr.gov/publications.

```
def authenticate_user(username, password):
    return username == "admin" and password == "secret"

def load_session(session_id):
    session_path = os.path.join(SESSION_DIR, session_id)
    if os.path.exists(session_path):
        with open(session_path, 'r') as f:
            return json.load(f)
    return None

def save_session(session_id, data):
    session_path = os.path.join(SESSION_DIR, session_id)
    with open(session_path, 'w') as f:
        json.dump(data, f)

def generate_session(username):
    session_id = str(uuid.uuid4())
    session_data = {"username": username}
    save_session(session_id, session_data)
    return session_id

def do_administrator_tasks():
    print("Content-Type: text/plain\n")
    print("Performing administrator tasks...")

init_session_dir()
form = cgi.FieldStorage()
cookie = http.cookies.SimpleCookie(os.environ.get("HTTP_COOKIE", ""))
session_cookie = cookie.get("sessionid")

session = None
username = None

if session_cookie:
    session = load_session(session_cookie.value)
    if session:
        username = session.get("username")

if not username:
    # Not logged in; try to authenticate
    username = form.getfirst("username")
    password = form.getfirst("password")
    if not (username and password and authenticate_user(username, password)):
        print("Error: you need to log in first")
        sys.exit(0)

    # Create session
    session_id = generate_session(username)
    print("Set-Cookie: sessionid={}; HttpOnly; Path=/".format(session_id))

if username == "Administrator":
    do_administrator_tasks()
else:
```


This report is available at no cost from the National Laboratory of the Rockies (NLR) at www.nlr.gov/publications.

```
    print("Content-Type: text/plain\n")
    print(f"Welcome, {username}")
```

Figure 10. Example remediation of CWE-287 with server-side session management

2.6 CWE-312: Cleartext Storage of Sensitive Information

This vulnerability can occur when systems like EVSE are storing information such as payment authorization caches without encrypting or sanitizing the information. Sensitive information can include various types of information, including, but not limited to, personally identifiable information, biometric data, and authentication data.

An example of CWE-312 storing cleartext sensitive information is shown in Figure 11. The authorization cache, which includes the UID, is stored in the SD card without the sensitive UID being removed. The UID can be used by bad actors to authorize a payment to an account that belongs to the customer that the UID is associated with. Figure 12 shows the sensitive data being sanitized before storage.

```
from datetime import datetime
import json

MICROSD_PATH = "E:/"
authorizationCache = {
    "valid_for_minutes" : 60,
    "status": "active",
    "allowed_kwh": 1000,
    "auth_method": "rfid",
    "auth_method_data": {
        "frid_uid": "8482749847"
    },
    "timestamp": "2023-10-01T12:00:00Z"
    }
auth_cache_file = open(f"{MICROSD_PATH}/auth_cache.json", "w+")

def save_auth_cache_to_sd (authorizationCache):
    record = {
        "timestamp": datetime.now().isoformat(),
        "authorizationCache": authorizationCache
    }
    print(record)
    auth_cache_file.write(json.dumps (record)+"\n")

# Example usage
save_auth_cache_to_sd (authorizationCache)
```

Figure 11. Example Python code with CWE-312 with cleartext storage of sensitive information

This report is available at no cost from the National Laboratory of the Rockies (NLR) at www.nlr.gov/publications.

```
from datetime
import datetime | import json

MICROSD_PATH = "E:/"
authorizationCache = {
   "valid_for_minutes" : 60,
   "status": "active",
   "allowed_kwh": 1000,
   "auth_method": "rfid",
   "auth_method_data": {
      "frid_uid": "8482749847"
   },
   "timestamp": "2023-10-01T12:00:00Z"
   }
auth_cache_file = open(f" {MICROSD_PATH}/auth_cache.json", "w+")

def save_auth_cache_to_sd (authorizationCache):
   del authorizationCache ["auth_method_data"] # Remove sensitive data
   record = {
      "timestamp": datetime.now().isoformat(),
      "authorizationCache": authorizationCache
   }
   print(record)
   auth_cache_file.write(json.dumps (record)+"\n")

# Example usage
save_auth_cache_to_sd (authorizationCache)
```

Figure 12. Example Python code with CWE-312 remedying cleartext storage of sensitive information

2.7 CWE-639: Authorization Bypass Through User-Controlled Key

This vulnerability occurs when a system's authorization functionality does not prevent one user from gaining access to another user's data or record by modifying the key value identifying the data. Using a known charger's identification, an adversary could retrieve EV charger version information and firmware upgrade history. The attacker can use this information to find which current vulnerabilities the EV charger has based on its software or firmware version. The code in Figure 13 lacks an authentication mechanism; thus, an attacker can guess or discover the charger's identification.

This report is available at no cost from the National Laboratory of the Rockies (NLR) at www.nlr.gov/publications.

```python
import requests

# Example target endpoint (simulated)
BASE_URL = "https://evcharger.example.com/api/charger"

# Attacker-controlled charger ID (guessed or discovered via enumeration)
charger_id = "CHARGER12345"

# No authentication headers or tokens are included
url = f"{BASE_URL}/{charger_id}/info"

response = requests.get(url)

if response.status_code == 200:
    print("[+] Charger Info Retrieved: ")
    print(response.json())
else:
    print("[-] Failed to retrieve charger info: {response.status_code}")
```

Figure 13. Example Python code with CWE-639 that lacks an authentication mechanism

There are several ways to mitigate this weakness. Figure 14 shows code that requires an authorization token to access charger information, enforcing access control based on the requester's identity. Implementing access control on the EV charger limits the number of people that can access its upgrade history, therefore minimizing the number of people who know about the charger's vulnerabilities. Other ways to protect against this include using role-based access control and audit logging.

This report is available at no cost from the National Laboratory of the Rockies (NLR) at www.nlr.gov/publications.

```
import requests
# Example target endpoint (simulated)
BASE_URL = "https://evcharger.example.com/api/charger"

# Attacker-controlled charger ID (still assumed known)
charger_id = "CHARGER12345"

# Secure access with an authorization token (e.g., OAuth2, API key)
auth_token = "Bearer eyJhbGcioi..." # Placeholder for a real token

headers = {
"Authorization": auth_token
}

url = f"{BASE_URL}/{charger_id}/info"

response = requests.get(url, headers-headers)

if response.status_code == 200:
    print("[+] Authorized Charger Info Retrieved: ")
    print(response.json())
elif response.status_code == 403:
    print("[-] Access denied: You are not authorized to view this charger info.")
else:
    print("[-] Failed to retrieve charger info: (response.status_code}")
```

Figure 14. Authorization token requirement to protect against CWE-693

2.8 CWE-77: Improper Neutralization of Special Elements Used in a Command ("Command Injection")

The vulnerability occurs when a program constructs and executes system-level commands using untrusted input and fails to sanitize or neutralize special characters (e.g., ;, &&, |, &). This allows attackers to inject arbitrary commands, potentially leading to full system compromise. Tools such as Bandit, Semgrep, Burp Suit, or OWASP ZAP can be used for testing command injection.

An example code with this vulnerability is illustrated in Figure 15. The code gets a user input for the system command execution, and it allows an attacker to run arbitrary system commands. To avoid the vulnerability, shell should not be invoked, command arguments should be kept isolated, and inputs should be sanitized by using allowlists. An example remediation is shown in Figure 16. The remedied code does not invoke a shell and passes arguments as isolated tokens.

This report is available at no cost from the National Laboratory of the Rockies (NLR) at www.nlr.gov/publications.

```
import os

def list_files(directory):
    # User input directly used in shell command
    os.system(f"ls {directory}")

# Example use
user_input = input("Enter directory to list: ")
list_files (user_input)
```

Figure 15. Example Python code with CWE-77 by using a shell command directly for the user input

```
import subprocess

def list_files_safe(directory):
    # Use a list of arguments and avoid shell=True
    subprocess.run([*ls, directory], check=True)

# Example use
user_input = input("Enter directory to list: ")
list_files_safe(user_input)
```

Figure 16. Example remediation of CWE-77 by using a list of arguments and avoiding invoking a shell

2.9 CWE-294: Authentication Bypass by Capture-Replay

This vulnerability happens when an attacker captures a valid authentication message in transit—such as a password, token, or session cookie—and reuses or replays it to gain unauthorized access even without knowing the original credentials. The code in Figure 17 does not use encryption for the log in, and an attacker can capture and resend the same request because the requests are replayable in the code. To prevent replay, best practices include using traffic encryption, token expiration, a unique identification, time window validation, and session binding. The code in Figure 18 uses a time stamp in the token creation and a time window validation.

```
from flask import Flask, request

app = Flask(__name__)

# Simulated login endpoint
@app.route('/login', methods=['POST'])
def login():
    if request.form['username'] == 'admin' and request.form['password'] == 'secret':
        return "Login successful"
    return "Login failed"
```

Figure 17. Example Python code with CWE-294 by using no encryption and replayable requests

This report is available at no cost from the National Laboratory of the Rockies (NLR) at www.nlr.gov/publications.

```python
import hmac, hashlib, time
from flask import Flask, request, abort

app = Flask(__name__)
SECRET_KEY = b'my_secret_key'
TIME_WINDOW = 30  # seconds

def generate_token(username, timestamp):
    message = f"{username}:{timestamp}".encode()
    return hmac.new(SECRET_KEY, message, hashlib.sha256).hexdigest()

@app.route('/login', methods=['POST'])
def login():
    username = request.form.get('username')
    timestamp = request.form.get('timestamp')
    token = request.form.get('token')

    if not username or not timestamp or not token:
        abort(400)

    # Validate time window
    if abs(time.time() - float(timestamp)) > TIME_WINDOW:
        return "Request expired", 401

    # Validate token
    expected_token = generate_token(username, timestamp)
    if hmac.compare_digest(expected_token, token):
        return "Login successful"
    return "Invalid token", 403
```

Figure 18. Example remediation of CWE-294 by using a time stamp to prevent replay

This report is available at no cost from the National Laboratory of the Rockies (NLR) at www.nlr.gov/publications.

3 Summary and Conclusions

Potential impacts of security breaches in the EV charging ecosystem intensify as EV charging stations evolve from stand-alone charging devices to integral parts of a larger network that encompasses the EV itself, the charging network operator's backend systems, and the power grid. Similar to other complex interconnected infrastructures, a complex array of cybersecurity vulnerabilities can pose risks to users, operators, and national infrastructure.

This report provides a list of 10 vulnerability categories for EVSE security based on statistics as well as the severity and scope of the potential impacts. Each category has an example attack scenario, potential impacts, methods for prevention, and related CWEs. This report also provides example codes that have vulnerabilities related to EVSE security along with remediations, test tools, and procedures. This will help increase workforce knowledge and advance the cybersecurity maturity of the industry to recognize insecure coding practices.

For the safe, reliable, and trustworthy expansion of electric mobility, stakeholders across the automotive, energy, and cybersecurity sectors should address the interconnected cybersecurity vulnerabilities in EVSE with a comprehensive, multilayered security approach.

This report is available at no cost from the National Laboratory of the Rockies (NLR) at www.nlr.gov/publications.

References

Delarea, S. 2024. "EV Charging Cyber Security: PlaxidityX Discovers Critical Vulnerability in EVeverest Open-Source EV Charging Firmware Stack (CVE-2024-37310)." PlaxidityX. August 22, 2024. plaxidityx.com/blog/automotive-cyber-security/ev-cyber-security-plaxidityx-discovers-critical-vulnerability-in-everest-open-source-ev-charging-firmware-stack-cve-2024-37310/.

Jun, M., R. Cryar, and A. Markel. 2024. *Investigate the Security of Electric Vehicle (EV) Ecosystem Applications*. Golden, CO: National Renewable Energy Laboratory. NREL/TP-5T00-90388. www.nrel.gov/docs/fy25osti/90388.pdf.

Kaiser, Q. 2024. "Security Advisory: Critical Vulnerabilities in New Charging Stations: Analysis of eCharge Controllers." ONEKEY. November 26, 2024. www.onekey.com/resource/critical-vulnerabilities-in-ev-charging-stations-analysis-of-echarge-controllers.

Mcfarlane, S. 2025. "Rogue Communication Devices Found in Chinese Solar Power Inverters." Reuters. May 14, 2025. www.reuters.com/sustainability/climate-energy/ghost-machine-rogue-communication-devices-found-chinese-inverters-2025-05-14/.

The MITRE Corporation. (n.d.) "CWE." cwe.mitre.org.

This report is available at no cost from the National Laboratory of the Rockies (NLR) at www.nlr.gov/publications.

Appendix: Summary of Related CWEs

CWE	Description
CWE-20	Improper input validation
CWE-77	Improper neutralization of special elements used in a command (“command injection”)
CWE-78	Improper neutralization of special elements used in an operating system command (“operating system command injection”)
CWE-79	Improper neutralization of input during webpage generation (cross-site scripting)
CWE-89	Improper neutralization of special elements used in an SQL command (“SQL injection”)
CWE-94	Improper control of generation of code (“code injection”)
CWE-115	Misinterpretation of input
CWE-122	Heap-based buffer overflow
CWE-190	Integer overflow or wraparound
CWE-200	Exposure of sensitive information to an unauthorized actor
CWE-269	Improper privilege management
CWE-284	Improper access control
CWE-285	Improper authorization
CWE-287	Improper authentication
CWE-290	Authentication bypass by spoofing
CWE-294	Authentication bypass by capture-replay
CWE-295	Improper certificate validation
CWE-299	Improper check for certificate revocation
CWE-306	Missing authentication for critical function
CWE-312	Cleartext storage of sensitive information
CWE-319	Cleartext transmission of sensitive information
CWE-327	Use of a broken or risky cryptographic algorithm
CWE-359	Exposure of private information (“privacy violation”)
CWE-400	Uncontrolled resource consumption
CWE-494	Download of code without integrity check
CWE-522	Insufficiently protected credentials
CWE-639	Authorization bypass through user-controlled keys
CWE-650	Trusting HTTP permission methods on the server side
CWE-732	Incorrect permission assignment for critical resource
CWE-778	Insufficient logging
CWE-798	Use of hard-coded credentials
CWE-862	Missing authorization
CWE-863	Incorrect authorization

This report is available at no cost from the National Laboratory of the Rockies (NLR) at www.nlr.gov/publications.

CWE	Description
CWE-912	Hidden functionality
CWE-913	Improper control of dynamically managed code resources
CWE-965	Insecure session management
CWE-1021	Improper restriction of rendered user interface (UI) layers or frames
CWE-1104	Use of unmaintained third-party components
CWE-1188	Initialization of a resource with an insecure default
CWE-1263	Improper physical access control
CWE-1277	Firmware not updateable
CWE-1392	Use of default credentials
CWE-1428	Reliance on HTTP instead of HTTPS

This report is available at no cost from the National Laboratory of the Rockies (NLR) at www.nlr.gov/publications.