Jurnal Infosecure Vol.6 No.1, April 2025
p-ISSN: 2087-1023
MTI-UNLA

Vulnerability Assessment of EV Roaming Protocol OCPI Based on OWASP API Security Risks

Reyza Permana Saputra[1], Arief Zulianto[2], Toto Suharto[3]

Postgraduate Program in Computer Science, Langlangbuana University[1,2,3]

reyzaps@gmail.com[1]

madzul@unla@ac.id[2]

tsuharto@gmail.com[3]

Abstract— The increasing use of electric vehicles (EV) is driven by supported by the rapid growth of EV charging infrastructure. This has driven widespread adoption of the OCPI protocol, which removes barriers to charging services and facilitates roaming between service providers. However, due to high market demand and the accelerated deployment of charging stations, OCPI protocol implementations often focus solely on functionality, with little attention paid to cybersecurity aspects. In this research, we present a vulnerability assessment of OCPI protocol implementation based on OWASP Top Ten - API Security Risks to minimize risks. Our method involves threat modeling to identify threats, attacks, vulnerabilities, and preventive measures that can impact OCPI implementation. We identified 27 potential threats in OCPI implementation and categorized them according to the risks listed in OWASP Top Ten. This assessment provides a basis for improving security standards in OCPI implementation for a safer future.

Keywords— Vulnerability Assessment, OCPI Protocol, API Security, OWASP API Security Risks, Electric Vehicles

I. Introduction

The trend of electric vehicle (EV) charging continues to increase year by year. Public charging and the interoperability of charging infrastructure are key factors in expanding the adoption of EVs [1]. One of the main challenges in this industry is the need for an integrated system for charging. The Open Charge Point Interface (OCPI) protocol has emerged as a solution that bridges this gap [2]. As the number of EV users grows, so the demand for roaming services increase, even though private charging may reduce the demand for roaming services, public charging infrastructure remains critical for: (1) people who do not have the capability to charge at home or at work, (2) EV users traveling long distances, (3) EV users with limited battery capacity, and (4) EV users traveling on vacation[3].

With the increased adoption of the OCPI protocol in charging infrastructure, communication security and data integrity have become increasingly critical issues. The complex combination of actors, devices, networks, and protocols in the ecosystem still lacks a clear focus on security aspects, making interconnected vulnerabilities occurs[4]. A report from D6.3 (*Design principles for an ideal EV roaming protocol*) one of interviews states that the current OCPI protocol is not sufficiently secure end-to-end. If the OCPI protocol is implemented without adequate security considerations, there are risks of cyber-attacks that could lead to data breaches, transaction manipulation, or even operational disruptions to the charging network.

In this study, we analyze potential security and mitigation issues related to the implementation of the OCPI protocol. In section 2, we describe the research methods used, including literature review, analysis, testing and reporting. In section 3, we conduct threat modeling, describing how roaming services are applied in the EV charging ecosystem, identifying assets, actors, and potential threats with mitigates that are classified into the OWASP Top Ten API security risks, and performing risk assessments using the OWASP risk rating methodology. Section 4 presents the conclusions of the study

II. Method

This research employs an experimental method with a pre-experimental One-Group Pretest-Posttest design, as pre-tests and post-tests are used to compare conditions before and after the mitigation [7].

A. *Research Flow*

Figure 1 shows the research flow conducted. The first step involves literature review and data collection, followed by threat modeling, vulnerability identification, risk assessment, and simulations of attacks and mitigations.

B. *Data Collection Method*

1. Literature Study

   Studying and gathering data by reviewing documentation, literature, notes, and journals related to cybersecurity, OCPI protocol, and OWASP Security Risks.
2. Field Study

   We collected data directly using observation techniques by observing the activities related to the use of the OCPI protocol and identifying the issues that arise during the implementation of the system.

C. Analysis Method

The analysis method used in this stage is threat modeling, which involves the following steps:

1. Describing System

This stage involves analyzing the OCPI protocol and how the protocol is implemented for the roaming charging process, represented using a sequence diagram

2.Decompose System

At this stage, the OCPI protocol is decomposed in detail and represented using a sequence diagram to illustrate the interaction between objects.

3. Threat Analysis

Threat analysis at this stage involves the identification of assets, actors, potential threats, and mitigations. Potential threats and mitigations are derived from the OWASP Top Ten Security Risks 2023 and simulated on the OCPI protocol by illustrating an abuse case diagram that depicts misactors and their actions [8]. The technique used is probability sampling proportionate stratified random sampling [7] for identification is selected at one of the endpoints of the OCPI protocol module that has a proportionate stratum with high risk and impact.

4. Risk Assessment

The Risk Assessment process at this stage is used to evaluate the risks identified during the threat analysis stage. The evaluation of these risks is conducted based on the OWASP Risk Rating methodology [6].

D. Testing Method

The testing method employed in this research involves several stages, which include the following:

1. Preparation

The preparation stage involves setting up the necessary materials such as software, hardware, and tools. The devices and tools used for testing consist of both software and hardware components. The testing target is the implementation of the OCPI protocol using a multiple peer-to-peer topology, where each peer has a different party id and country code.

2. Attack Simulation

The attack simulation at this stage is conducted by testing the vulnerabilities and mitigations identified through the threat analysis process. The testing process utilizes the technique of probability sampling proportionate stratified random sampling [7], where one of the endpoint threats is tested (pre-test) and re-tested after it has been handled or mitigated (post-test).

III. Results and Discussion

A. Describing & Decomposing the System

This section describes how the OCPI protocol is implemented in the EV Charging ecosystem. In this scenario, a user registered with operator A can perform charging with operator B, provided both operators have established a roaming process. Operator A is commonly referred to as the e-mobility service provider (e-MSP), while operator B is known as the charge point operator (CPO). Figure 2 depicts the architecture when roaming services are implemented, where dashed lines represent data flow and solid lines represent electricity flow.

The scenario detailing the flow of the charging process using roaming services can be described as follows:

1. The user, who is registered with an EMSP, connects the charging connector to their vehicle.
2. The user starts/stops charging through an application provided by the EMSP.
3. The application sends a start/stop charging signal to the EMSP central system via a REST API.
4. The EMSP central system receives the request and forwards it with the Command Start/StopSession request via the OCPI protocol to the CPO Central System.
5. The CPO sends a start/stop charging signal with the RemoteStartTransaction/RemoteStopTransaction command to the Charge Station via WebSocket using the OCPP protocol.
6. The Charge Station receives the request and responds by sending a confirmation response (Start/StopTransaction) to the Central System.
7. The charge points either delivers or stops the electricity to the electric vehicle using the ISO 15118 protocol.
8. The CPO central system creates a session for the EMSP, and the EMSP creates a session for the user.
9. The CPO central system generates a charging detail record (CDR) for the EMSP, and the EMSP generates an invoice for the user based on the electricity received. Figure 3 depics the detailed scenario described.

Based on the charging system described above, it can be concluded that the charging process involving roaming or interoperability between operators involves at least four actors:

1. **EMSP** - The actor responsible for providing charging services to the EV user, bridging the EV user with the CPO for charging.
2. **CPO** - The actor operating the charging station. The CPO is connected with the EMSP to allow roaming services.
3. **Charge Point** - The actor responsible for delivering electricity to the electric vehicle.
4. **EV User** - The actor performing the charging process via the EMSP [9].

The OCPI protocol includes several modules that enable roaming or interoperability between platforms. These modules are divided into four categories:

1. **Credential Handshake** - A process where the CPO or EMSP platforms perform pairing, exchanging

credential tokens so that the CPO and EMSP can engage in transactions, such as pushing or pulling data.

2. **Discovery** - A process where the CPO platform provides data on charging locations, including charger point details and tariffs, to the EMSP.
3. **Charging** - A process where the EMSP platform intermediates between the CPO and the user to provide charging services. To implement the charging feature within the OCPI protocol, at least three modules are required: token, command, and session.
4. **Billing** - A process where the CPO platform sends the Charging Detail Record (CDR) to the EMSP, detailing the charging process that has been completed [9].

B. Threat Analysis

Threat analysis is carried out by identifying assets within the implementation of the OCPI protocol. These assets may include physical devices, software, sensitive data such as credentials, tokens, sessions, and CDRs (Charging Detail Records), as well as private data like locations and tariffs.

We categorize the actors who may pose threats in the implementation of the OCPI protocol into three categories:

1. **Authorized Partner**: A partner, such as an e-MSP, that possesses authentication and has access to specific resources.
2. **Unauthorized Partner**: A partner, such as a CPO, that has authentication but does not have access to specific resources.
3. **Attacker**: A user or competitor who have no authentication and attempts to cause harm.

The next step is to identify, analyze, and manage potential threats to the implementation of the OCPI protocol. This will help in understanding the threats that the system may face, how these threats can be exploited and the appropriate mitigation steps that can be taken to protect the assets. The analysis and testing are conducted following the OWASP Top Ten sequence.

1. Broken Object Level Authorization (BOLA)

BOLA is a vulnerability when an unauthorized party exploits API endpoint with failed authorization by manipulating the object identifier in the parameters sent as a request [10]. The impact of exploiting BOLA can lead to data exposure, modification, or deletion.

BOLA can occur in OCPI protocol implementations when the implementer fails to apply unpredictable values to object IDs and fails to perform authorization checks. For example, an unauthorized partner can manipulate `location_id` or `tariff_uid` in the endpoint `/ocpi/cpo/2.2.1/locations/{location_id}` or `/ocpi/cpo/2.2.1/tariffs/{tariff_uid}`.

Mitigation for BOLA involves implementers applying unpredictable values such as UUIDs for all object identifiers and applying authorization checks based on the OCPI documentation [9].

2. Broken Authentication

Broken authentication is a vulnerability that occurs when authentication mechanisms are improperly implemented, allowing attackers to exploit these weaknesses to take over other users' identities. This flaw can lead to unauthorized access, data breaches, and other harmful effects [11].

The OCPI protocol uses an authorization header with a credential token to authenticate each request and ensure that the request is made by an authenticated party [9]. Broken authentication in OCPI protocol implementations can happen when attackers exploit authentication weaknesses, such as brute-force attacks, predictable credential tokens (e.g., plain text or weakly hashed), or stolen tokens from data breaches to access the system.

Mitigating broken authentication involves (1) implementing rate-limiting mechanisms to limit and handle attacker requests, (2) locking the IP address after failed authentication attempts, (3) hashing or encrypting data used for authentication, such as credential tokens and `token_uid` in the database, (4) validating easily predictable tokens, and (5) renewing credential tokens at least once a month.

3. Broken Object Propery Level Authorization (BOPLA)

BOPLA is a security weakness similar to BOLA but focuses on specific properties or attributes of an object. BOPLA risks arise when an endpoint exposes excessive data (Excessive Data Exposure) and allows adding or deleting values from an object (Mass Assignment), which may be private or sensitive [9].

BOPLA is prone to occur at OCPI implementation endpoints that display sensitive data (Excessive Data Exposure), such as session and CDR endpoints. For example, the system may fail to authorize or filter a CDR object, causing the CDR response to reveal all data requested by an EMSP. BOPLA impacts can expose sensitive data to an EMSP that should not have access to it.

Mitigation involves adhering to the OCPI technical documentation, which states: "A CPO is not required to return all known CDRs; the CPO is allowed to return only the CDRs relevant to the requesting EMSP"[9]. The CPO should only return session or CDR data relevant to the EMSP by authorizing the CDR data based on `country_code` and `party_id` from the `cdr_token` object and the `CredentialRoles` possessed by the EMSP.

4.Unrestricted Resource Consumption

Authorized parties can make excessive requests, and the system fails to limit or monitor resource usage properly, leading to reduced performance and server overload, which can cause a Denial of Service (DOS) due to resource exhaustion [10].

In OCPI protocol implementations, two main scenarios contribute to Unrestricted Resource Consumption: (1) failing to limit pull and push requests from partners, particularly with dynamic APIs like the location API, where the charger point status constantly changes. A pull request may occur when the EMSP wants the latest status from the CPO, and a push request may occur when the CPO bulk-pushes location data to the EMSP after the CPO server experiences downtime. (2) Failing to limit payloads from partners, particularly in the session API, where one session may have multiple charging periods.

Mitigations are provided by the OCPI standard, including (1) limiting excessive pull requests from the EMSP within intervals measured in minutes or even seconds, as advised in the OCPI documentation: "It is therefore advised to clients pulling lists from a server to do this at a relatively low polling interval: think in hours, not minutes, and to introduce some splay (randomize the length of the poll interface a bit)"[10], (2) using message queues to handle data updates, and (3) ensuring the charging period payload aligns with tariff changes, as recommended in the OCPI technical documentation: "It is also recommended to add Charging Periods for all moments relevant to Tariff changes"[10].

5.Broken Function Level Authorization (BFLA)

Complex access control policies with hierarchies, groups, and different roles, as well as unclear separation between administrative and regular functions, tend to cause authorization weaknesses [11].

BFLA can occur in OCPI protocol implementations when a partner only assigned the role of an EMSP is able to manage CPO data through CPO API endpoints, and vice versa.

Mitigations include (1) ensuring that CPOs and EMSPs are only provided with a list of relevant endpoints during the initial pairing process, and (2) implementing authorization checks for each request made by the CPO or EMSP by verifying the credentials associated with the EMSP.

6.Unrestricted Access to Sensitive Business Flow

Unrestricted Access to Sensitive Business Flow occurs when a system or application lacks adequate restrictions or controls to regulate who can access and execute critical and sensitive business processes [10].

This vulnerability can occur in OCPI protocol implementations when an EMSP with token data performs charging at multiple charge points simultaneously, and the system fails to limit this charging business flow, preventing other users from accessing the charging point.

Mitigations include validating that a single token owned by the EMSP can only reserve and/or charge at one charge point at a time.

7.Server-Side Request Forgery (SSRF)

SSRF is a security vulnerability where an attacker can manipulate a server to make HTTP requests to domains or IP addresses chosen by the attacker [10].

SSRF can occur in OCPI protocol implementations due to a lack of validation for input URLs and endpoints or features that allow users or other systems to submit URLs or addresses for the server to access. Attackers can perform port scanning through input URLs and HTTP redirection, such as callback command results, by exploiting the absence of URL validation.

Mitigations include validating and sanitizing all input data from the client, particularly input URLs, and avoiding redirects to insecure connections such as HTTP or rejecting `response_url` if the HTTP protocol is used.

8.Security Misconfiguration

Security Misconfiguration is a vulnerability where attackers exploit weaknesses caused by misconfigurations in the security settings of systems, applications, or infrastructure, potentially exposing security holes [10]. Security Misconfiguration can occur in OCPI protocol implementations due to inadequate security headers, exposing sensitive user data and system details, which may lead to full server compromise [10].

Potential threats include: (1) Attackers exploit outdated TLS versions that may use insecure encryption and hash algorithms, (2) Attackers exploit detailed error messages from error codes that often reveal system structure, server configuration, or application implementation details. This information can give attackers insight into potential vulnerabilities, such as database names, tables, or queries used[12], (3) Attackers exploit details of the software version used, disclosing server configuration or components, providing attackers with clues to potential vulnerabilities, (4) Attackers exploit CORS misconfigurations, allowing unauthorized third parties to call the API from other domains, (5) Attackers exploit the lack of security headers, making the API vulnerable to several types of attacks, (6) The OCPI protocol is a REST API, which implementers may access through web browsers like Swagger or OpenAPI, allowing attackers to exploit default accounts to access the implemented API, (7) Attackers exploit the lack of suspicious request logging and monitoring, which is fundamental in nearly all incidents[10].

Mitigations include (1) Ensuring that the TLS version used is up-to-date, (2) Ensuring that the response complies with OCPI standards, (3) Ensuring that all response headers and bodies in the OCPI API implementation do not expose details of the technology used, (4) Ensuring proper CORS policies are in place, (5) Ensuring that each endpoint includes and properly configures essential security headers, (6) Ensuring that strong usernames and passwords are used to access documentation, and (7) Implementing audit logs and effective monitoring and alert systems for all requests.

9.Improper Inventory Management

Unsafe Consumption is a risk where an application using an API fails to validate or verify the data received from an external or internal API correctly. This can lead to various types of attacks, such as injection, man-in-the-middle, r data misuse.

This risk is particularly likely in OCPI protocol communications (1) partners interact with the system through unencrypted channels, such as providing callback URLs with command results using the HTTP protocol, which can expose credential tokens and sensitive data, (2) attackers exploit the absence of validation and sanitization of inputs from each request, potentially allowing SQL Injection and (3)Cross-Site Scripting (XSS) attacks.

Mitigation include (1) use encrypted channels such as HTTPS or TLS. As stated in the OCPI technical documentation: "The interfaces are protected on the HTTP transport level, with SSL and token-based authentication. Please note that this mechanism does not require client-side certificates for authentication, only server-side certificates to set up a secure SSL connection." Therefore, all communication within OCPI implementations should use HTTPS, ensuring that input URLs used for callbacks are validated, (2) apply sanitization to user-provided inputs to avoid potential injection flaws, such as SQL Injection and XSS.

B. Testing

At this stage, testing is conducted by simulating attacks based on the potential risks and mitigations analyzed against the OCPI protocol implementation. These attack simulations will be performed according to the OWASP Top Ten 2023 - API Security Risks.

1. BOLA

In our BOLA testing, we used the A-B Testing method by creating two different accounts, where one account attempts to access data from the other account [12]. Our testing results revealed that every endpoint in the OCPI protocol with object identifiers such as location/{country_code}/{party_id}/location_id/, /tariff_uid and /token_uid failed to perform authorization checks, leading to exposure of information and modification of data by unauthorized partners.

The mitigation measures implemented have successfully addressed the BOLA risk by applying unpredictable UUID values and performing authorization checks with a response code of 2001.

2. Broken Authentication

Broken Authentication testing was conducted using brute force and fuzzing techniques with the ZAP tool, utilizing easily predictable credential tokens based on combinations of country code, party ID, and timestamp, and simulating data theft scenarios through data leakage.

Our testing results identified the absence of rate limiting, validation against weak tokens, and the use of plain text for tokens as factors leading to broken authentication.

The mitigation measures implemented include applying rate limits and locking IP addresses after 3 failed authentication attempts by blocking the IP, rejecting weak tokens by returning a response code of 2002, hashing tokens used by partners to access the system and using AES-128 encryption for system-to-partner access and renewing tokens every month. These measures have proven successful in addressing the risks associated with broken authentication. However, the use of hashing for tokens has a drawback: in OCPI, there is no identifier like a username or similar to indicate who is making the request, so token verification must be done individually, which burdens the memory.

3. BOPLA

BOPLA testing is quite similar to BOLA testing, using a multiple peer-to-peer topology but with different scenarios. BOPLA testing was conducted under the condition that one platform has performed a charging process and generated a session or CDR.

Our testing results revealed that the CPO failed to filter data correctly, returning all session and CDR data to EMSP, including irrelevant information, through GET requests to /ocpi/sessions or /ocpi/cdrs.

The mitigation measures successfully eliminated the risk associated with BOPLA by filtering data based on country_code and party_id on the session or CDR objects owned by the CPO, and filtering country_code and party_id on the cdr_token object related to session or CDR objects with the credential token held by the EMSP.

4. Unrestricted Resource Consumption

Unrestricted resource consumption testing was conducted on a target server with the specifications of 2 CPUs and 8 GB of memory. The testing scenarios included (1) excessive pull requests that making numerous requests to the version, location, tariff, session, and CDR modules simultaneously with high polling intervals. (2) bulk push that sending data with 1000 objects. (3) excessive limit with requesting data from the location endpoint (pull location) with a query parameter limit set to 1000, where the system failed to enforce this limit. (4) send excessive charging period with sending too many charging periods requests to EMSP.

The results of the testing revealed that the server became high CPU loads, and sometimes users were unable to access the server due to Gateway Time-outs.

The mitigation measures implemented included: (1) applying rate limits to pull requests. (2) limiting the payload size for charging periods. (3) Enforcing a default return limit of 100 data items when the query parameter limit is set to 1000. (4) Implementing a queue for bulk data requests. These measures successfully mitigated the risk of unrestricted resource consumption, as observed from the server’s condition after testing.

5. BLFA

BFLA testing was conducted using a multiple peer-to-peer topology with the scenario where platform EFG acts as EMSP using endpoints related to EMSP on platform ABC, and platform XYZ acts as CPO using endpoints related to CPO on platform ABC.

Our testing revealed that platforms with roles limited to either EMSP or CPO could make unauthorized requests. For instance, an EMSP platform could add or modify locations or tariffs on the CPO platform.

Mitigation measures implemented include (1) providing a list of endpoints appropriate to the partner's role. (2) Performing authorization checks to verify that the country_code and party_id of the requesting platform match those of the country_code and party_id in the object being created or modified. For example, returning a response code of 2001 when a platform with only the EMSP role attempts to create or modify location data.

6. Unrestricted Access to Sensitive Business Flow

Testing for unrestricted access to sensitive business flow was conducted with the scenario where a single token owned by e-MSP could initiate a session or charging more than once.

Our testing revealed that a single token owned by EMSP (authorized partner) could initiate charging multiple times, leading to unused charge points being marked as occupied.

Mitigation for unrestricted access to sensitive business flow successfully addressed this risk by adding validation to ensure that a single token cannot initiate charging more than once when it already has an active session. The system now responds with a rejection when EMSP attempts to start a session with a token that has already been used.

7. SSRF

SSRF testing was conducted by experimenting with input data for port scanning through image URLs in the OCPI location module and attempting to input response URLs pointing to malicious URLs.

The results from both tests showed that the response URL from localhost:5432 was accepted, and the system blindly followed redirections to https://attacker.com. This allowed an attacker to exploit the redirection to direct the application to internal or malicious endpoints.

Mitigation for SSRF risk was implemented with two measures with validating each URL against a whitelist of allowed URLs and rejecting response URLs if they do not match permitted endpoints. These mitigations effectively reduce the risk associated with SSRF.

8. Security misconfiguration

Security misconfiguration testing was conducted on a target server with IP xx.xx.xxx.xxx for the OCPI protocol implementation using ZAP and Postman for scanning.

Our testing revealed the following issues:

- Verbose Errors: Displaying detailed error messages that reveal the programming technologies used.
- Server Technology Version: Revealing the server technology version through response headers.
- Lack of Security Headers: Missing security headers such as Content Security Policy (CSP), Server Leak Version Information, Strict-Transport-Security, X-Content-Type-Options, and CORS policy.
- Default Account Exposure: Default accounts were used to access OCPI protocol API documentation through Swagger or OpenAPI.
- Insufficient Logging and Monitoring: Inadequate logging and monitoring practices.

Mitigation for security misconfiguration was implemented as follows:

- Remove Verbose Errors: Ensure responses adhere to OCPI protocol standards without revealing detailed error messages.
- Remove Server Version from Response Headers: Hide server technology version information in response headers.
- Add Security Headers: Implement Content Security Policy (CSP) with default-src 'self', Strict-Transport-Security, and X-Content-Type-Options with nosniff in response headers.
- Replace Default Accounts: Change default accounts to strong combinations of usernames and passwords.
- Enhance Logging and Monitoring: Implement alerts or notifications via email for attacks such as brute force attempts.

9. Improper Inventory

Improper inventory management testing was conducted by examining whether the OCPI implementation was documented properly.

The testing results revealed that while the OCPI protocol API implementation was documented, but it was not managed correctly. Issues included a lack of clear versioning and differentiation between environments for the implemented API. The mitigation measure implemented was to add versioning information (e.g., V1) for the running API.

10. Unsafe Consumption of APIs

Unsafe consumption of APIs was tested using the following methods: Inputting URLs with the HTTP protocol, SQL injection. XXS (Cross-Site Scripting) injection.

The results were as follows:

- HTTP Protocol: The response_url in the CommandResult module still accepted HTTP protocol inputs.
- SQL Injection: SQL injection was possible through the authorization header with the input test OR 1 = 1, which allowed bypassing the login for unauthorized access.
- XXS Injection: XXS injection vulnerabilities were found in Swagger or OpenAPI used for OCPI protocol documentation, particularly with inputs at the URL https://api.ma.la/tmp/cors/swi/.

Mitigation measures successfully addressed these issues by:

- Validating response_url to ensure only secure protocols are accepted.
- Sanitizing the authorization header to prevent SQL injection attacks.

• Removing URL inputs in Swagger or OpenAPI documentation to prevent XXS injection.

Table 1 summarizes the potential vulnerabilities that may occur in the implementation of the OCPI protocol.

TABEL I. POTENTIAL THREATS IN OCPI IMPLEMENTATION

No	Potential Threat	Description
1	*Manipulate Object Identifier*	Unauthorized Partners manipulate by guessing object identifiers in request parameters to obtain unauthorized data by exploiting authorization check failures.
2	*Access Unauthorized Data*	Unauthorized Partners access unauthorized location data by exploiting authorization failures in the system.
3	*Create or Update Unauthorized Data*	Unauthorized CPOs create or modify unauthorized location data that does not align with their CredentialRoles and exploit authorization failures in the system.
4	*Brute Force Attack*	Attackers perform brute force attacks to exploit weaknesses in the authentication mechanism.
5	*Exploit Week or Predictable Token*	Attackers exploit weak and predictable tokens to gain access to the system.
6	*Exploit Token Theft*	Attackers exploit tokens following data breaches.
7	*Access Unauthorized Data Sensitif*	Unauthorized EMSPs access CDR data that is not relevant to their role.
8	*Excessive Pull Data*	EMSP performs excessive pull requests.
9	*Pull Data in one Request*	EMSP makes requests to the API with a large limit.
10	*Bulk Push Data*	CPO performs excessive bulk data operations.
11	*Send Excessive Payload*	CPO sends excessively large payloads.
12	*Manipulate Data CPO*	Unauthorized EMSPs manipulate CPO data, such as locations and tariffs, which should be managed by the CPO.
13	*Manipulate Data EMSP*	Unauthorized CPOs manipulate EMSP data, such as tokens and other information, which should be managed by the EMSP.
14	*Request Multiple StartSession with one Token*	EMSP makes multiple charging requests with the same token.
15	*Port Scanning over URL*	Attacker performs port scanning through input URLs.
16	*Expolit HTTP Redirection*	Attackers exploit HTTP redirections to obtain credential token data.
17	*Expolit Deprecated TLS*	Attackers exploit vulnerabilities in outdated or insecure versions of the TLS (Transport Layer Security) protocol to gain unauthorized access to sensitive data transmitted between clients and servers.
18	*Exploit Verbose Error Message*	Attackers exploit overly detailed error messages to obtain sensitive information or internal application configurations that can be used to launch further attacks.
19	*Exploit Disclosure Server Information*	Attackers exploit inadvertently exposed server information through headers response or other public resources to gain details about server and software configurations that can be used to plan further attacks.
20	*Exploit Missing CORS Policy*	Attackers exploit the absence of proper CORS (Cross-Origin Resource Sharing) policies on the server, allowing them to access and manipulate sensitive resources from different domains.
21	*Exploit Missing Security Headers*	Attackers exploit deficiencies in the implementation of security headers on the API
22	*Exploit Default Account*	Attackers exploit default accounts in OCPI documentation that can be accessed through a web browser.
23	*Exploit Insufficient Logging & Monitoring*	Attackers exploit default accounts in OCPI documentation that can be accessed through a web browser.
24	*Exploit Old Version of API OCPI*	Attackers exploit vulnerabilities in outdated versions that have not been properly maintained
25	*Input Callback URL without HTTP*	Partners input callback URLs without using HTTPS, causing sensitive data sent to these URLs to be intercepted and modified by unauthorized parties.
26	*Input SQL Injection*	Attackers attempt to exploit the system by injecting malicious SQL commands through improperly validated input.
27	*Input XXS Injection*	Attackers exploit the system by injecting malicious scripts through improperly validated input, aiming to execute malicious code.

Table II summarizes the findings from the research conducted on the implementation of the OCPI protocol, along with the severity levels determined through the risk assessment process using the OWASP Risk Rating.

Threats are classified into severity levels based on their likelihood and impact. A threat is considered critical when it has both a high likelihood of occurrence and a high impact on the system, resulting in severe consequences if exploited. It is deemed high when there is a medium likelihood and medium impact, potentially causing significant issues. medium threats are those with low likelihood and low impact, leading to moderate consequences. Finally,low threats are characterized by both low likelihood and low impact, resulting in minimal consequences if exploited.

TABEL II. SEVERITY LEVEL OF THREAT

OWASP TOP TEN	Threat	Severity
API1	*Manipulate Object Identifier*	*Medium*
	Access Unauthorized Data	*Low*
	Create or Update Unauthorized Data	*High*
API2	*Brute Force Attack*	*Critical*
	Exploit Weak or Predictable Token	*Critical*
	Exploit Token Theft	*Critical*
API3	*Access Unauthorized Data*	*Low*
API4	*Excessive Pull Data*	*High*
	Pull Data in one Request	*Low*
	Bulk Push Data	*Medium*
	Send Excessive Payload	*Low*
API5	*Manipulate Data CPO*	*Medium*
	Manipulate Data EMSP	*Medium*
API6	*Request Multiple StartSession per Token*	*Low*
API7	*Port Scanning over URL*	*Medium*
	Exploit HTTP Redirection	*Low*
API8	*Exploit Deprecated TLS*	*Medium*
	Exploit Verbose Error Message	*Low*
	Exploit Disclosure Server Information	*Low*
	Exploit Missing CORS Policy	*Medium*
	Exploit Missing Security Headers	*Low*
	Exploit Default Account	*Medium*
	Exploit Insufficient Logging & Monitoring	*Low*
API9	*Exploit Old Version of API OCPI*	*High*
API10	*Input Callback URL without HTTP*	*High*
	Input SQL Injection	*Critical*
	Input XXS Injection	*Low*

After identifying the vulnerabilities present in each endpoint of the OCPI protocol, Table III provides the mitigation measures for the aforementioned risks.

OWASP TOP TEN	**Mitigates**
API1	Implementation of GUID/UUID for records related to IDs
	Implementation of authorization checks by filtering data based on party_id and country_code, with credential roles held by the CPO as the data owner.
API2	Implementation of Rate Limiting Mechanism
	Block IP after 3 failed authentication attempts
	Encrypt credential tokens using strong cryptographic algorithms, such as HMAC-SHA256 or HMAC-SHA512.
	Renew credential tokens at least once a month.
API3	Implementation of Authorization Check by ensuring that the returned objects are relevant to the EMSP making the request, by filtering based on country_code and party_id.
API4	Implementation of Rate Limiting Mechanism
	Display default data with a maximum of 100 records per page
	Limit the payload sent by the CPO
API5	Display a list of endpoints that are relevant based on the role
	Implementation of authorization checks for endpoint functions based on the role
API6	Implementation of validation with a policy that allows each EMSP token to initiate only one charging session.

API7	Validate and sanitize all input data from clients, especially URL inputs.
	Avoid redirections using insecure connections such as HTTP.
API8	Return all responses according to OCPI standards.
	Do not expose the version of the technologies used by the server.
	Set proper *Security Header.*
	Do not use default accounts for accessing OCPI documentation.
	Provide alerts for any suspicious requests, such as brute force and SQL injection.
API9	Use versioning for the implementation of the OCPI API.
API10	Validate Input URLs to enforce HTTPS
	Sanitize all inputs provided by partners.
	Do not display input URLs in OpenAPI or Swagger documentation.

TABEL II. MITIGATES OF THREAT

IV. CONCLUSION

Based on the research conducted, the use of threat modeling and automation scanning methods with OWASP Risk Rating assessment has successfully identified 27 potential threats along with their mitigations. The severity levels are classified as 4 Critical, 3 High, 8 Medium, and 11 Low for the OCPI protocol implementation.

The results from threat modeling and automation scanning of the OCPI protocol implementation, based on OWASP Top Ten API Security Risks, also indicate that the implementation is still vulnerable to several common attacks that can be exploited by attackers. The discovery of 9 out of 10 risks in the OWASP Top Ten API Security Risks list suggests significant security gaps, particularly in the areas of authorization, authentication, data exposure, and inadequate resource management.

REFERENCE

[1] (2023) *Trends in electric vehicle charging* [Online]. Available: https://www.iea.org/reports/global-ev-outlook-2024/trends-in-electric-vehicle-charging.

[2] (2023) *Open Charge Point Interface (OCPI): A Gateway To Seamless EV Charging* [Online]. Available: https://bacancysystems.com/blog/open-charge-point-interface-ocpi.

[3] Van der Kam, M., dan Bekkers, R. (2020): *Design principle for an 'ideal' Roaming Protocol, Report* D6.3 *for the* evRoaming4EU *project*.

[4] Van Aubel, P., dan Poll, E. (2022): *Security of EV-Charging Protocols, Digital Security Group*, Institute for Computing and Information Sciences, Radboud University.

[5] Gough, J, Bryant, D, Auburn, M : *Mastering API Architecture*, Buku, O'reilly, 2023.

[6] (Jeff Williams) OWASP *Risk Rating Methodology* [Online], Available: https://owasp.org/www-community/OWASP_Risk_Rating_Methodology.

[7] Sugiono: Metode Penelitian Kuantitatif, Kualitatif dan R & D, Alfabeta, 2018.

[8] Islam, G., Qureshi, M.A: *A Framework for Security Requirment Elicitation*, Thesis, Blekinge Institute of Technology, 95,2012.

[9] EVRoaming Foundation: *Open Charge Point Interface* (OCPI) 2.2.1, Dokumentasi Teknis OCPI, EVRoaming Foundation, 2021.

[10] (2023) OWASP TOP 10 API *Security Risks* - 2023 [Online], Available: https://owasp.org/API-Security/editions/2023/en/0x11-t10/.

[11] Cybelium: Mastering OWASP (A Comprehensive Guide To Master Owasp), 2023.

[12] Ball., J. Corey. *Hacking* APIs, *Breaking Web Application Programming Interface.* No Starch Press, 2023.