2025 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW) | 979-8-3315-9546-3/25/$31.00 ©2025 IEEE | DOI: 10.1109/EUROSPW67616.2025.00044

CheckOCPP: Automatic OCPP Packet Dissection and Compliance Check

Soumaya Boussaha
SAP, EURECOM
Biot, France
soumaya.boussaha@sap.com

Victor Fresno Gómez
EURECOM, UPM
Madrid, Spain
victorfresno@live.com

Thomas Barber
SAP
Baden-Wurtemberg, Germany
thomas.barber@sap.com

Daniele Antonioli
EURECOM
Biot, France
daniele.antonioli@eurecom.fr

***Abstract*—As the adoption of electric vehicles (EVs) grows, ensuring compliance and security in EV charging infrastructure is critical. The Open Charge Point Protocol (OCPP) is the de facto standard for communication between EV charging stations and central management systems. However, verifying real-world implementations for protocol adherence and security remains challenging. We introduce CheckOCPP, an OCPP dissector integrated with Wireshark, designed to detect OCPP versions (1.6, 2.0, and 2.0.1), validate message structures against predefined compliant schemas, and flag non-compliant or malformed packets in real-time. CheckOCPP is built using Lua and leverages the Mobility House Python OCPP open-source library. As a dissector, CheckOCPP can be used for compliance verification and security analysis. Our evaluation demonstrates its effectiveness in dissecting and validating OCPP 1.6, 2.0, and 2.0.1 traffic, including detecting non-compliant behaviors against simulated malformed packets and EmonEVSE, an actual charging station.**

1. Introduction

The *Open Charge Point Protocol (OCPP)* is the de facto standard for communication between electric vehicles (EVs), charging stations (CSs), often referred to as Charge Points (CPs), and central management systems (CSMSs) [1]. OCPP enables remote monitoring, session control, firmware updates, and billing integration. It allows operators to manage access to CPs, track energy consumption, and apply pricing models. Since its inception in 2009, OCPP has been updated with features and security mechanisms. Most notably, OCPP 1.6 and OCPP 2.0.1 accommodate smart charging and security profiles.

Despite OCPP's widespread adoption, practical challenges remain in achieving and verifying *OCPP compliance*. Inconsistent or incomplete implementations can lead to security vulnerabilities and unexpected behaviors. Researchers, security analysts, and compliance auditors struggle to pinpoint these issues, especially in production environments, due to the lack of dedicated tools. The existing OCPP Compliance Test Tool (OCTT) [2] is closed-source, accessible behind a cost barrier (€3,000–€18,000 per OCPP version), and requires conducting predefined active tests against the CS or CSMS. This approach is not best suited for all use cases, including post-deployment audits.

Multiple studies have examined the security landscape of OCPP on production systems, uncovering various threats, including man-in-the-middle (MitM) attacks, replay attacks, message tampering, authentication flaws, and denial-of-service (DoS) [3]–[7]. However, no paper has provided dedicated tooling for OCPP traffic dissection, unlike similar protocols with dedicated dissectors (e.g., MQTT [8] and Modbus [9]). OCPP lacks a valuable tool for security assessments, allowing OCPP compliance checks. Ideally, the tool should be *passive*, i.e., observe OCPP packets flowing in the network or from a pcap, and *black-box*, i.e., work without knowing the internals of the CS or CSMS under test.

To address these concerns, we propose **CheckOCPP**, an OCPP toolkit designed for dissection, compliance auditing, testing, and security analysis of OCPP implementations. Our tool works in a black-box manner without the need to access CP or CSMS implementations. By dissecting OCPP traffic, CheckOCPP automatically isolates the OCPP versions and packet components, including sensitive data, and evaluates their compliance against a schema.

We tested CheckOCPP against all OCPP versions and popular OCPP implementations. We evaluated a Mobility House [10] based simulation for OCPP 2.0 and 2.0.1. The Mobility House provides an open-source OCPP library for simulating charge point and central system behavior. We tested it against the EmonEVSE [11] CS for OCPP1.6, which is used in production systems. EmonEVSE is an open-source charge controller for electric vehicle supply equipment (EVSE) hardware.

CheckOCPP has proven to accurately dissect and analyze OCPP packets across versions 1.6, 2.0, and 2.0.1. It identified four key issues, including three non-compliant behaviors across different implementations. In the EmonEVSE device [11], a physical charging point used in production environments and implementing OCPP 1.6, CheckOCPP detected a violation where a `GetConfiguration` request containing a key exceeding the 50-character limit was improperly accepted.

In an OCPP 2.0 setup using a simulated charging station and the Mobility House library [10], it flagged a malformed `BootNotification` message with a 25-character CS model name, exceeding the specification's 20-character limit. A similar simulation environment for OCPP 2.0.1 revealed another compliance issue, where an undefined certificate type was included in an `InstallCertificate` request.

Beyond compliance testing, CheckOCPP also exposed a security concern by successfully extracting sensitive authentication data (`idToken`) in plaintext from a

DOI 10.1109/EuroSPW67616.2025.00044

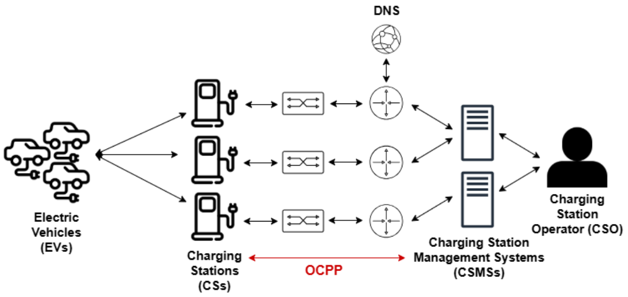

DNS
Electric Vehicles (EVs)
Charging Stations (CSs)
OCPP
Charging Station Management Systems (CSMSs)
Charging Station Operator (CSO)

Figure 1. EV charging backend eco-system.

`ReserveNow` message, emphasizing the risks of deploying OCPP systems without TLS encryption.

We summarize our contributions as follows:

- We present CheckOCPP, a novel OCPP toolkit that integrates with Wireshark to dissect, parse, and analyze all OCPP versions.
- CheckOCPP checks message payloads against compliant schema definitions, automatically detecting non-compliant or malformed OCPP packets.
- We validated CheckOCPP in an evalaution against Mobility House (OCPP 2.0 & 2.0.1) [10] and EmonEVSE (OCPP 1.6) [11]. We open-source our tool at https://github.com/vfg27/CheckOCPP.

2. Background

This section presents background information about OCPP, Wireshark, OpenEVSE, and IPmininet.

2.1. OCPP

The Open Charge Point Protocol (OCPP) [12] is an open communication standard to ensure interoperability between CS and centralized management entities. Centralized management entities typically include charging station operators (CSOs), as in Figure 1, which oversee the deployment, maintenance, and operation of the charging infrastructure. Charging station management systems (CSMSs) provide the software platforms to monitor, control, and optimize charging station performance.

OCPP has *three* main versions: 1.6, 2.0, and 2.0.1, maintained by the *Open Charge Alliance (OCA). OCPP 1.6.* [1], [12], introduced in 2015, marked a milestone for OCPP by improving functionality and expanding its applicability across the electric vehicle charging ecosystem. Widely adopted by CS manufacturers and CSMSs, this release introduced key features such as SOAP and JSON support, intelligent charging capabilities for load balancing and profile management, enhanced status updates, and local list management.

In 2020, the whitepaper publication *Improved Security for OCPP 1.6-J* standardized the implementation of advanced security measures inspired by more modern versions of the OCPP. These enhancements include secure connection establishment, security event logging, and secure firmware updates, enabling developers to create robust and secure implementations of OCPP 1.6-J.

OCPP 2.0 [1], released in 2018, was the first in the 2.x series, developed with industry collaboration. For functional specification issues, it was quickly superseded by OCPP 2.0.1 [1], released in 2020. The latter resolves issues in OCPP 2.0, such as incompatible machine-readable schema definition files. Moreover, OCPP 2.0.1 introduces better device and transaction management and strengthened security measures. It also supports ISO 15118 [1], a vehicle-to-grid charging standard.

OCPP defines *three security profiles* to ensure secure communication [13], [14]. *Security Profile 1* represents the basic level, which mandates client (CS) authentication using a password but does not require authenticating the server (CSMS) and encrypting the traffic [13], [14]. *Security Profile 2* improves over the prior one by introducing Transport Layer Security (TLS) for channel encryption and server certificate validation for server authentication while using a password for client authentication over a TLS channel [13], [14].

Security Profile 3 employs TLS with client and server certificates to achieve mutual authentication [13], [14]. The CS and CSMS must use TLS v1.2 or higher. ECDHE key agreement is recommended over RSA because it provides forward secrecy. Deprecated or insecure cipher suites shall not be used. TLS compression methods are prohibited to prevent side-channel attacks.

OCPP is based on WebSocket [1], [12], an application layer protocol standardized by the IETF as RFC 6455 [15] in 2011. WebSocket enables full-duplex communication between clients and servers over a single TCP connection. Unlike traditional HTTP, which follows a request-response model, WebSocket allows for bidirectional interactions, facilitating real-time data exchange with reduced latency and overhead. This protocol is particularly beneficial for applications such as electric charging and billing, where timely data transmission is crucial.

An OCPP packet has the following four fields:

- Type: message type (2=Request, 3=Response, 4=Error).
- ID: a unique identifier, typically a random UUID.
- Name: present only for requests.
- Payload: the message data.

2.2. Wireshark

Wireshark [16] is a popular open-source network protocol analyzer. It enables real-time packet capture, dissection, and inspection. It supports several protocols, such as TCP, UDP, HTTP, and TLS, making it an essential tool for network security research and troubleshooting. By applying filters and analyzing traffic patterns, users can detect anomalies, manage problems, and examine encrypted communications if they have the necessary keys. Wireshark allows the integration of custom dissectors using Lua and currently does not have an OCPP dissector.

In network traffic analysis, *dissection* refers to systematically breaking down network packets and flows to extract meaningful insights regarding communication patterns, protocols, and potential security threats. A *dissector* is a tool or module to interpret and analyze network protocols, often integrated into traffic analysis frameworks such as Wireshark [8], [9]. These dissectors parse protocol

Figure 2. EmonEVSE WiFi Connected EV Charging Station.

headers, payloads, and metadata, facilitating deep packet inspection (DPI) and anomaly detection.

Wireshark supports dissectors written in Lua [17], a programming language and lightweight scripting engine designed to integrate into applications. Lua's core features include dynamic typing, first-class functions, and versatile table-based data structures. Lua dissectors are more adaptable than those written in C because they are interpreted (other than compiled) and have simpler semantics (comparable to Python).

2.3. OpenEVSE

OpenEVSE [11] is an open-source platform offering a CS and mobile application in Figure 2. Developed initially to generate the SAE J1772 pilot signal, it has become a widely used technology in charging stations worldwide, offering scalability and customization for hardware and software. Thanks to its open architecture, OpenEVSE allows manufacturers and developers to create custom charging solutions, from standard products (e.g., EmonEVSE [18]) to DIY kits. Its features include a WiFi module, which facilitates monitoring, control, and integration with home automation systems, HTTP, and MQTT.

2.4. IPmininet

IPMininet [19] is an extension of the Mininet [20] network emulator. IPMininet and Mininet are based on Linux containers and allow the emulation of complex (Internet-like) networks using a single Linux kernel. A virtual network includes hosts, switches, controllers, routers, and links. Mininet is compatible with IPv4, while IPMininet extends the compatibility to IPv6.

3. Design

In this section, we motivate our work and present the design of CheckOCPP.

3.1. Motivation

Several studies have explored the security landscape of OCPP, identifying Vulnerabilities and attacks (e.g., man-in-the-middle (MitM), replay attacks, message tampering, authentication weaknesses, and DoS) [3]–[7]. Despite the research on OCPP, there is *no* OCPP toolkit to dissect, parse, analyze, and check the compliance of OCPP packets.

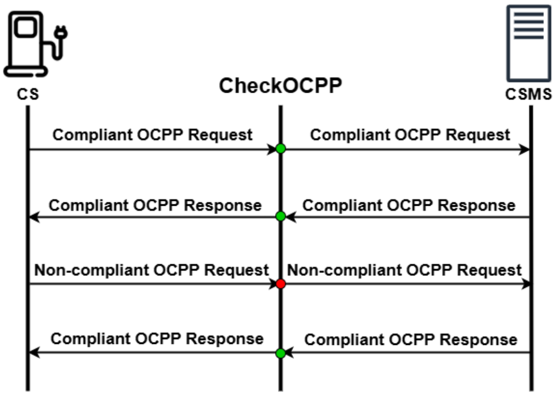


Figure 3. OCPP Communication and CheckOCPP traffic dissection.

A toolkit is needed because deployed OCPP versions (1.6, 2.0, and 2.0.1) introduce new packets, schemas, and security features, and they can even co-exist in the same network. Moreover, OCPP is fragmented as not all vendors upgrade their infrastructure uniformly. Vendors also customize OCPP deployment, e.g., by introducing proprietary authentication mechanisms [3], further increasing the presence of non-compliant and vulnerable OCPP deployments.

OCPP packet dissection would enhance security assessments by isolating sensitive information, such as *authentication tokens*, *charging point passwords*, and facilitating security testing. Furthermore, packet dissection is already used for other protocols like MQTT [8] and Modbus [9] and has been proven helpful for security and traffic analysis. Thus, adopting an OCPP dedicated dissector for each OCPP version would help tackle the challenges of validating OCPP implementations' compliance and testing their security robustness.

Despite OCPP's popularity, there are *no* open-source compliance checkers for OCPP. When it comes to verifying OCPP compliance, auditors have the choice of using the OCPP Compliance Test Tool (OCTT) [2], which requires *active* testing of a set of pre-defined scenarios against CPs and CSMS. However, OCTT is closed-source [2] and comes with a significant paywall of 3000-18000 € per single protocol version. Furthermore, it is unsuited for *passive audits*, where compliance is assessed without direct interaction with the system under test. A passive audit is beneficial in scenarios where interference with a system is undesirable, such as in production environments.

To address these gaps, we introduce CheckOCPP, a toolkit capable of dissecting OCPP packets regardless of their version and checking OCPP compliance. The toolkit can work in passive and active modes and does not require prior knowledge of the tested OCPP clients (CSs) and servers (CSMSs).

3.2. CheckOCPP

CheckOCPP is an OCCP toolkit with dissection and compliance checking capabilities. The tool can sniff the traffic in a (production) OCPP network and analyze the OCPP packets as shown in Figure 3. To dissect OCPP packets when TLS is in place (security levels two or three [13]), the TLS key is needed first to decrypt the packets before parsing and compliance checking.

CheckOCPP supports the JSON implementation of OCPP, while SOAP support can be added with minimal engineering effort by integrating the XML-based SOAP packet schemas. This choice reflects the industry's preference for JSON, which offers better compatibility with modern frameworks than SOAP.

Dissection. The dissection process is the following: after capturing traffic, CheckOCPP looks first for WebSocket packets, then filters packets with the desired protocol (1.6, 2.0, or 2.0.1) in the WebSocket protocol information. CheckOCPP matches the observed data structure against OCPP-compliant schemas tables, each associated with an OCPP version. The dissector infers the specific OCPP version associated with each packet through this process. Furthermore, it organizes the packet content based on schema in an intuitively accessible way.

Compliance Check. CheckOCPP automatically validates the dissected packets against the OCPP specification. In cases where the payload does not align with the predefined schema definitions, CheckOCPP flags the message as non-compliant. It highlights the non-compliant packet and the error concerning the expected compliant schema. To verify compliance, CheckOCPP validates the dissected fields of the packet against the expected up-to-date compliant schema. The compliance verification through the CheckOCPP can be conducted in a passive setup, making it a suitable choice for use cases where no intervention in the traffic is desired.

Security Analysis. The tool can be used for active security analysis on the target OCPP network. Once the traffic is dissected, CheckOCPP can change a packet payload on the fly, facilitating attack scenarios such as man-in-the-middle, replay, or injection. This highlights its relevance as a diagnostic tool and a means to assess the impact of insecure deployments.

4. Implementation

Next, we describe how we implemented CheckOCPP's parsing and compliance check logic.

4.1. Parsing

We implemented CheckOCPP using Lua [17] and Wireshark's WebSocket dissector [16]. As shown in Figure 4, the tool loads the OCPP message schemas from the Mobility House library [10]. These schemas are used to check the structure of the dissected packages. The schemas are organized into three distinct tables, corresponding to OCPP versions 1.6, 2.0, and 2.0.1. we support all the messages in Table 1.

We use the open-source cjson plugin [21] to parse OCPP JSON payloads into Lua tables. A Lua table is a structure that represents arrays or dictionaries. The parsed packets include their message type, identifier, message name (if applicable), and the OCPP payload.

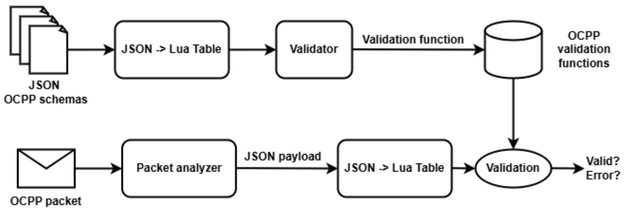


Figure 4. CheckOCPP diagram.

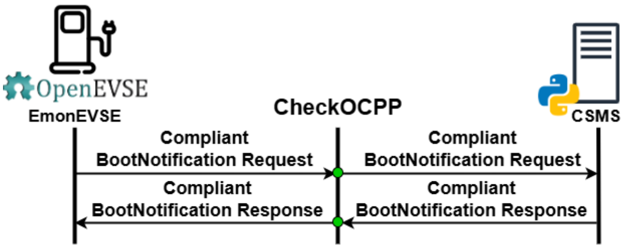

OpenEVSE
EmonEVSE
CheckOCPP
CSMS
Compliant BootNotification Request
Compliant BootNotification Request
Compliant BootNotification Response
Compliant BootNotification Response

Figure 5. CheckOCPP deployment for OCPP 1.6 evaluation against the EmonEVSE device.

4.2. Compliance check

CheckOCPP passes the tables containing the parsed OCPP packets to jsonchema [22], an open-source Lua plugin we use to generate OCPP validation functions stored in a separate table. The parsed OCPP payloads are passed through the corresponding validation function, determining whether they adhere to the expected schema.

As shown in Figure 4, the validation function returns two outputs: Valid or Error. If the payload is Valid, the dissector adds the information to the Wireshark tree and marks the packet compliant. If there is an Error, the error is added to the tree, and the packet is flagged as non-compliant in red as malformed packets using the Expert information dialog [23], a Wireshark-compatible solution to highlight packets for errors and warnings.

5. Evaluation

We present our evaluation setup and results.

5.1. Setup

OCPP 1.6. We tested an EmonEVSE CS (client) based on OpenEVSE in a lab environment (Figure 5). The CSMS (server) was implemented using the Python-based Mobility House library for OCPP 1.6. This setup evaluated CheckOCPP's ability to dissect OCPP 1.6 traffic and verify the compliance of the client and server implementations.

OCPP 2.0, 2.0.1. We developed an OCPP emulation scenario using IP-Mininet [19]. The virtual OCPP CSs and CSMSs use the OCPP 2.0 and 2.0.1 implementations from the Mobility House Python library [10] (Figure 6). This setup enabled controlled message exchanges and deliberate injection of malformed packets to test CheckOCPP's compliance-checking capabilities. It also validated the tool's ability to isolate security-sensitive OCPP data.

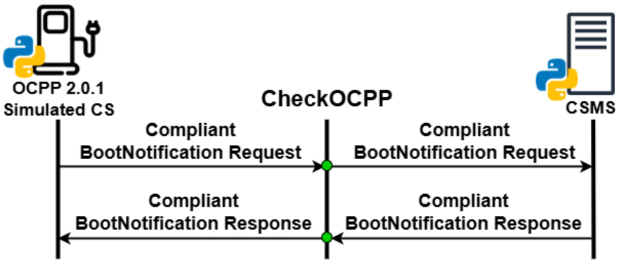
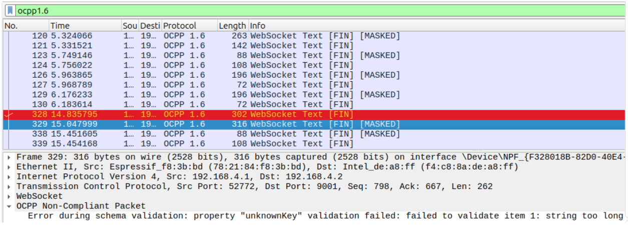


Figure 6. CheckOCPP deployment for OCPP 2.0.1 evaluation using Mobility House and IP-Mininet.



```
ocpp1.6
No.       Time       Sou Desti Protocol  Length Info
    120 5.324066     1…  19…  OCPP 1.6     263 WebSocket Text [FIN] [MASKED]
    121 5.331521     1…  19…  OCPP 1.6     142 WebSocket Text [FIN]
    123 5.749146     1…  19…  OCPP 1.6      88 WebSocket Text [FIN] [MASKED]
    124 5.756022     1…  19…  OCPP 1.6     108 WebSocket Text [FIN]
    126 5.963865     1…  19…  OCPP 1.6     196 WebSocket Text [FIN] [MASKED]
    127 5.968789     1…  19…  OCPP 1.6      72 WebSocket Text [FIN]
    129 6.176233     1…  19…  OCPP 1.6     196 WebSocket Text [FIN] [MASKED]
    130 6.183614     1…  19…  OCPP 1.6      72 WebSocket Text [FIN]
    328 14.835795    1…  19…  OCPP 1.6     302 WebSocket Text [FIN]
    329 15.047999    1…  19…  OCPP 1.6     316 WebSocket Text [FIN] [MASKED]
    338 15.451605    1…  19…  OCPP 1.6      88 WebSocket Text [FIN] [MASKED]
    339 15.454168    1…  19…  OCPP 1.6     108 WebSocket Text [FIN]
› Frame 329: 316 bytes on wire (2528 bits), 316 bytes captured (2528 bits) on interface \Device\NPF_{F328018B-82D0-40E4-
› Ethernet II, Src: Espressif_f8:3b:bd (78:21:84:f8:3b:bd), Dst: Intel_de:a8:ff (f4:c8:8a:de:a8:ff)
› Internet Protocol Version 4, Src: 192.168.4.1, Dst: 192.168.4.2
› Transmission Control Protocol, Src Port: 52772, Dst Port: 9001, Seq: 798, Ack: 667, Len: 262
› WebSocket
⌄ OCPP Non-Compliant Packet
    Error during schema validation: property "unknownKey" validation failed: failed to validate item 1: string too long
```

Figure 7. Non-compliant OCPP 1.6 packet detected in EmonEVSE traffic.

5.2. Results

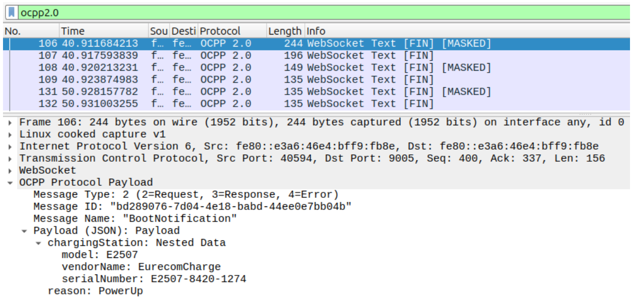
CheckOCPP successfully identified *three non-compliant messages*, including an improperly formatted GetConfiguration response in OCPP 1.6 for the EmonEVSE device [11]. Next, we describe the experimental results in more detail.

OCPP 1.6. Using CheckOCPP, we identified non-compliant OCPP 1.6 behavior in the EmonEVSE charging point. For example, the GetConfiguration message, used to retrieve parameters from the CS, requires parameters to adhere to predefined length restrictions under OCPP 1.6. As shown in Figure 7, we queried an artificially crafted variable with an excessively long name exceeding 50 characters. A compliant CS should reject such requests, but the EmonEVSE responded to this invalid query. CheckOCPP captured this exchange and flagged the non-compliant packets.

OCPP 2.0 and 2.0.1. We also tested CheckOCPP with OCPP 2.0 traffic. As shown in Figure 8, CheckOCPP correctly identified and parsed OCPP 2.0 packets. We then evaluated CheckOCPP with OCPP 2.0.1, including mixed traffic containing both OCPP 2.0 and 2.0.1 packets (Figure 9). CheckOCPP accurately distinguished between the two protocol versions during analysis.

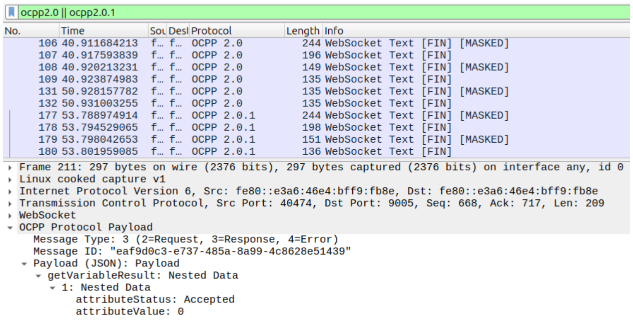
To test compliance checking, we injected a non-compliant BootNotification message with a 25-character CS model name (exceeding the 20-character limit). As shown in Figure 10, CheckOCPP flagged this violation by highlighting the packet in red and displaying an error message.

For OCPP 2.0.1, we tested certificate installation using an undefined certificate type. Figure 11 demonstrates CheckOCPP's ability to detect this non-compliance by flagging the malformed packet in red and displaying the error message.



```
ocpp2.0
No.       Time           Sou Desti Protocol   Length Info
    106 40.911684213     f…  fe…  OCPP 2.0      244 WebSocket Text [FIN] [MASKED]
    107 40.917593839     f…  fe…  OCPP 2.0      196 WebSocket Text [FIN]
    108 40.920213231     f…  fe…  OCPP 2.0      149 WebSocket Text [FIN] [MASKED]
    109 40.923874983     f…  fe…  OCPP 2.0      135 WebSocket Text [FIN]
    131 50.928157782     f…  fe…  OCPP 2.0      135 WebSocket Text [FIN] [MASKED]
    132 50.931003255     f…  fe…  OCPP 2.0      135 WebSocket Text [FIN]
› Frame 106: 244 bytes on wire (1952 bits), 244 bytes captured (1952 bits) on interface any, id 0
› Linux cooked capture v1
› Internet Protocol Version 6, Src: fe80::e3a6:46e4:bff9:fb8e, Dst: fe80::e3a6:46e4:bff9:fb8e
› Transmission Control Protocol, Src Port: 40594, Dst Port: 9005, Seq: 400, Ack: 337, Len: 156
› WebSocket
⌄ OCPP Protocol Payload
    Message Type: 2 (2=Request, 3=Response, 4=Error)
    Message ID: "bd289076-7d04-4e18-babd-44ee0e7bb04b"
    Message Name: "BootNotification"
  ⌄ Payload (JSON): Payload
    ⌄ chargingStation: Nested Data
        model: E2507
        vendorName: EurecomCharge
        serialNumber: E2507-8420-1274
      reason: PowerUp
```

Figure 8. CheckOCPP analyzing OCPP 2.0 traffic.

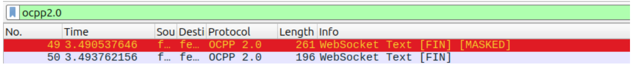


```
ocpp2.0 || ocpp2.0.1
No.       Time           Sou Desl Protocol      Length Info
    106 40.911684213     f…  f…  OCPP 2.0         244 WebSocket Text [FIN] [MASKED]
    107 40.917593839     f…  f…  OCPP 2.0         196 WebSocket Text [FIN]
    108 40.920213231     f…  f…  OCPP 2.0         149 WebSocket Text [FIN] [MASKED]
    109 40.923874983     f…  f…  OCPP 2.0         135 WebSocket Text [FIN]
    131 50.928157782     f…  f…  OCPP 2.0         135 WebSocket Text [FIN] [MASKED]
    132 50.931003255     f…  f…  OCPP 2.0         135 WebSocket Text [FIN]
    177 53.788974914     f…  f…  OCPP 2.0.1       244 WebSocket Text [FIN] [MASKED]
    178 53.794529065     f…  f…  OCPP 2.0.1       198 WebSocket Text [FIN]
    179 53.798042653     f…  f…  OCPP 2.0.1       151 WebSocket Text [FIN] [MASKED]
    180 53.801959085     f…  f…  OCPP 2.0.1       136 WebSocket Text [FIN]
› Frame 211: 297 bytes on wire (2376 bits), 297 bytes captured (2376 bits) on interface any, id 0
› Linux cooked capture v1
› Internet Protocol Version 6, Src: fe80::e3a6:46e4:bff9:fb8e, Dst: fe80::e3a6:46e4:bff9:fb8e
› Transmission Control Protocol, Src Port: 40474, Dst Port: 9005, Seq: 668, Ack: 717, Len: 209
› WebSocket
⌄ OCPP Protocol Payload
    Message Type: 3 (2=Request, 3=Response, 4=Error)
    Message ID: "eaf9d0c3-e737-485a-8a99-4c8628e51439"
  ⌄ Payload (JSON): Payload
    ⌄ getVariableResult: Nested Data
      ⌄ 1: Nested Data
          attributeStatus: Accepted
          attributeValue: 0
```

Figure 9. Mixed OCPP 2.0/2.0.1 traffic analysis using CheckOCPP.

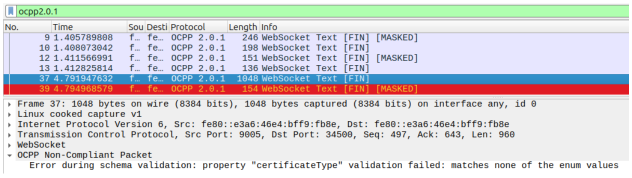
Figure 12 showcases a dissected OCPP ReserveNow message sent when a user authenticates to start charging their EV. The figure highlights CheckOCPP's ability to examine and display all application-layer components of the OCPP packet, including sensitive data such as the idToken (highlighted in red). This token, often an RFID identifier or credit card number, is used by the CSMS to authorize charging sessions.

This last example highlights CheckOCPP's dissection capability that can facilitate detecting sensitive information in OCPP communication. Malicious authors can use the idToken to commit fraud by replicating it. The OCPP alliance recommends using no encryption for secure networks such as home deployment. However, related work [6] has highlighted that many production charging points do not implement TLS.



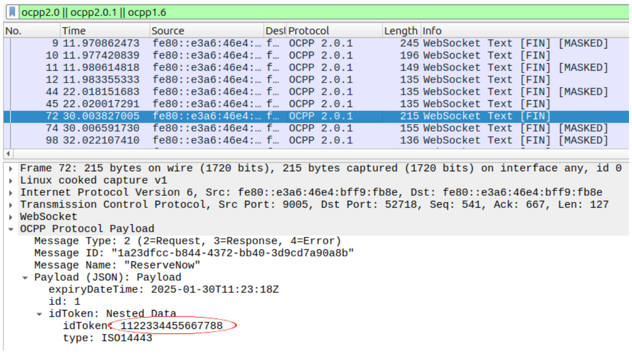
```
ocpp2.0
No.       Time          Sou Desti Protocol   Length Info
     49 3.490537646     f…  fe…  OCPP 2.0      261 WebSocket Text [FIN] [MASKED]
     50 3.493762156     f…  fe…  OCPP 2.0      196 WebSocket Text [FIN]
```

Figure 10. CheckOCPP detecting non-compliant OCPP 2.0 BootNotification message.



```
ocpp2.0.1
No.       Time          Sou Desti Protocol     Length Info
      9 1.405789808     f…  fe…  OCPP 2.0.1      246 WebSocket Text [FIN] [MASKED]
     10 1.408073042     f…  fe…  OCPP 2.0.1      198 WebSocket Text [FIN]
     12 1.411566991     f…  fe…  OCPP 2.0.1      151 WebSocket Text [FIN] [MASKED]
     13 1.412825814     f…  fe…  OCPP 2.0.1      136 WebSocket Text [FIN]
     37 4.791947632     f…  fe…  OCPP 2.0.1     1048 WebSocket Text [FIN]
     39 4.794968579     f…  fe…  OCPP 2.0.1      154 WebSocket Text [FIN] [MASKED]
› Frame 37: 1048 bytes on wire (8384 bits), 1048 bytes captured (8384 bits) on interface any, id 0
› Linux cooked capture v1
› Internet Protocol Version 6, Src: fe80::e3a6:46e4:bff9:fb8e, Dst: fe80::e3a6:46e4:bff9:fb8e
› Transmission Control Protocol, Src Port: 9005, Dst Port: 34500, Seq: 497, Ack: 643, Len: 960
› WebSocket
⌄ OCPP Non-Compliant Packet
    Error during schema validation: property "certificateType" validation failed: matches none of the enum values
```

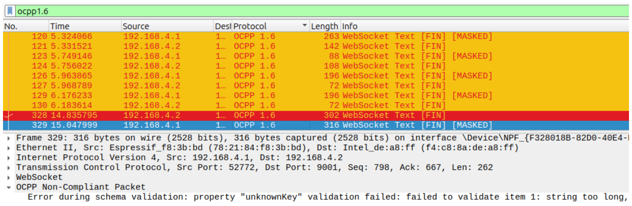
Figure 11. CheckOCPP identifying non-compliant certificate in OCPP 2.0.1 traffic.



```
ocpp2.0 || ocpp2.0.1 || ocpp1.6
No.     Time          Source               Desl Protocol      Length Info
     9 11.970862473  fe80::e3a6:46e4:…  f…   OCPP 2.0.1       245 WebSocket Text [FIN] [MASKED]
    10 11.977420839  fe80::e3a6:46e4:…  f…   OCPP 2.0.1       196 WebSocket Text [FIN]
    11 11.980614818  fe80::e3a6:46e4:…  f…   OCPP 2.0.1       149 WebSocket Text [FIN] [MASKED]
    12 11.983355333  fe80::e3a6:46e4:…  f…   OCPP 2.0.1       135 WebSocket Text [FIN]
    44 22.018151683  fe80::e3a6:46e4:…  f…   OCPP 2.0.1       135 WebSocket Text [FIN] [MASKED]
    45 22.020017291  fe80::e3a6:46e4:…  f…   OCPP 2.0.1       135 WebSocket Text [FIN]
    72 30.003827005  fe80::e3a6:46e4:…  f…   OCPP 2.0.1       215 WebSocket Text [FIN]
    74 30.006591730  fe80::e3a6:46e4:…  f…   OCPP 2.0.1       155 WebSocket Text [FIN] [MASKED]
    98 32.022107410  fe80::e3a6:46e4:…  f…   OCPP 2.0.1       136 WebSocket Text [FIN] [MASKED]

▸ Frame 72: 215 bytes on wire (1720 bits), 215 bytes captured (1720 bits) on interface any, id 0
▸ Linux cooked capture v1
▸ Internet Protocol Version 6, Src: fe80::e3a6:46e4:bff9:fb8e, Dst: fe80::e3a6:46e4:bff9:fb8e
▸ Transmission Control Protocol, Src Port: 9005, Dst Port: 52718, Seq: 541, Ack: 667, Len: 127
▸ WebSocket
▾ OCPP Protocol Payload
    Message Type: 2 (2=Request, 3=Response, 4=Error)
    Message ID: "1a23dfcc-b844-4372-bb40-3d9cd7a90a8b"
    Message Name: "ReserveNow"
  ▾ Payload (JSON): Payload
      expiryDateTime: 2025-01-30T11:23:18Z
      id: 1
    ▾ idToken: Nested Data
        idToken: 1122334455667788
        type: ISO14443
```

Figure 12. Isolating user authentication token using CheckOCPP.



```
ocpp1.6
No.   Time       Source        Desl Protocol   Length Info
  120 5.324066   192.168.4.1   1…   OCPP 1.6      263 WebSocket Text [FIN] [MASKED]
  121 5.331521   192.168.4.2   1…   OCPP 1.6      142 WebSocket Text [FIN]
  123 5.749146   192.168.4.1   1…   OCPP 1.6       88 WebSocket Text [FIN] [MASKED]
  124 5.756022   192.168.4.2   1…   OCPP 1.6      108 WebSocket Text [FIN]
  126 5.963865   192.168.4.1   1…   OCPP 1.6      196 WebSocket Text [FIN] [MASKED]
  127 5.968789   192.168.4.2   1…   OCPP 1.6       72 WebSocket Text [FIN]
  129 6.176233   192.168.4.1   1…   OCPP 1.6      196 WebSocket Text [FIN] [MASKED]
  130 6.183614   192.168.4.2   1…   OCPP 1.6       72 WebSocket Text [FIN]
  328 14.835795  192.168.4.2   1…   OCPP 1.6      302 WebSocket Text [FIN]
  329 15.047999  192.168.4.1   1…   OCPP 1.6      316 WebSocket Text [FIN] [MASKED]
▸ Frame 329: 316 bytes on wire (2528 bits), 316 bytes captured (2528 bits) on interface \Device\NPF_{F328018B-82D0-40E4-
▸ Ethernet II, Src: Espressif_f8:3b:bd (78:21:84:f8:3b:bd), Dst: Intel_de:a8:ff (f4:c8:8a:de:a8:ff)
▸ Internet Protocol Version 4, Src: 192.168.4.1, Dst: 192.168.4.2
▸ Transmission Control Protocol, Src Port: 52772, Dst Port: 9001, Seq: 798, Ack: 667, Len: 262
▸ WebSocket
▾ OCPP Non-Compliant Packet
    Error during schema validation: property "unknownKey" validation failed: failed to validate item 1: string too long,
```

Figure 13. CheckOCPPIPV4 traffic distinction tested against EmonEVSE device.

6. Discussion

Although OCPP does not mandate a specific IP version, IPv6 adoption is growing in the EV charging ecosystem due to its scalability and expanded address space [24]. For example, the IEC 61851-1 standard specifies IPv6-based protocols and power line communication (PLC) for advanced direct current (DC) charging features [24]. Similarly, the IETF documents IPv6 use cases for vehicular networking in Intelligent Transportation Systems (ITS) [25].

To address this trend, we added an IPv4/IPv6 differentiation feature to CheckOCPP. When enabled, IPv4 traffic is flagged in yellow (see Figure 13); this feature can be helpful for systems where IPV6 is required or recommended. Testing CheckOCPP on the OCPP 1.6 setup described in Section 5.2, we observed that the EmonEVSE device uses IPv4 for OCPP 1.6 communications, as shown by the marked packets.

7. Related Work

OCTT [2], provided by the OCA, is a cloud-based service designed to validate CS and CSMS against OCPP specifications. It supports predefined test scenarios for OCPP 1.6 and 2.0.1. However, OCTT requires direct interaction with the device under test (e.g., CS or CSMS) and is limited to predefined test sequences, making it unsuitable for real-world deployments. While OCTT ensures thorough protocol conformance testing, it is not designed for network monitoring or passive analysis. Consequently, it is primarily suited for pre-deployment quality assurance (QA) testing.

In contrast, CheckOCPP operates as a dissector for sniffed traffic between CS and CSMS, requiring no active interference with the tested systems. It flags non-compliant packets and enables passive audits. Another key distinction is that CheckOCPP supports all OCPP versions (at the time of writing) and is open-source, whereas OCTT is closed-source and costs between $3,000 to $18,000 per license for a single OCPP version [2]. While we do not claim CheckOCPP can fully substitute OCTT's complex test cases—developed by the OCPP protocol maintainers, we argue that CheckOCPP serves as a complementary solution for post-deployment analysis and security research.

Mobility House [10] is a Python library that enables developers to create advanced, customized charging systems. The library implements OCPP 1.6 (JSON), 2.0, and 2.0.1. Internal schema validation and typed request/response classes promote standards-compliant development and simplify the creation of custom charging applications. While the library performs internal checks for message consistency, it implements the protocol rather than a diagnostic or dissection tool for OCPP [10].

8. Conclusion

We present CheckOCPP, a new open-source OCPP toolkit compatible with Wireshark, providing dissection and compliance checking for OCPP and enabling passive real-time analysis of OCPP communications. CheckOCPP automates protocol version detection (OCPP 1.6, 2.0, and 2.0.1), validates message structures against schemas, and flags non-compliant packets. Built using Lua and Mobility House, CheckOCPP allows better OCPP traffic analysis, compliance validation, and security analysis. By open-sourcing CheckOCPP, we provide the community with a valuable tool for auditing OCPP deployments, complementing existing compliance frameworks, and enhancing security analysis.

Our evaluation against Mobility House and IP-Mininet (OCPP 2.0/2.0.1) and the production-grade EmonEVSE (OCPP 1.6) demonstrates CheckOCPP's ability to dissect traffic, detect malformed packets, and uncover compliance gaps in real-world implementations. For example, CheckOCPP identified that EmonEVSE accepts `GetConfiguration` requests with key names exceeding the 50-character limit, a deviation from the OCPP 1.6 specification that could be exploited to conduct an overflow attack.

Acknowledgments

Work funded by the European Union under grant agreement no. 101070008 (ORSHIN project). Views and opinions expressed are, however, those of the author(s) only and do not necessarily reflect those of the European Union. Neither the European Union nor the granting authority can be held responsible. Moreover, it has been partially supported by the French National Research Agency under the France 2030 label (NF-HiSec ANR-22-PEFT-0009) and the Apricot/ENCOPIA ANR MESRI-BMBF project (ANR-20-CYAL-0001).

References

[1] OCA, "Open charge point protocol," 2024. [Online]. Available: https://openchargealliance.org/protocols/open-charge-point-protocol/

[2] Open Charge Alliance, "OCPP Test Tools," 2025. [Online]. Available: https://openchargealliance.org/test-tool/

[3] Z. Garofalaki, D. Kosmanos, S. Moschoyiannis, D. Kallergis, and C. Douligeris, "Electric vehicle charging: A survey on the security issues and challenges of the open charge point protocol (OCPP)," *IEEE Communications Surveys & Tutorials*, vol. 24, no. 3, pp. 1504–1533, 2022.

[4] R. Metere, Z. Pourmirza, S. Walker, and M. Neaimeh, "An Overview of Cyber Security and Privacy on the Electric Vehicle Charging Infrastructure," *arXiv preprint arXiv:2209.07842*, 2022. [Online]. Available: https://arxiv.org/pdf/2209.07842

[5] A. Brighente, M. Conti, D. Donadel, R. Poovendran, F. Turrin, and J. Zhou, "Electric Vehicles Security and Privacy: Challenges, Solutions, and Future Needs," *arXiv preprint arXiv:2301.04587*, 2023. [Online]. Available: https://arxiv.org/pdf/2301.04587

[6] K. Sarieddine, M. A. Sayed, S. Torabi, R. Atallah, D. Jafarigiv, C. Assi, and M. Debbabi, "Uncovering Covert Attacks on EV Charging Infrastructure: How OCPP Backend Vulnerabilities Could Compromise Your System," *Preprint*, 2024. [Online]. Available: https://www.researchgate.net/publication/377183224_Uncovering_Covert_Attacks_on_EV_Charging_Infrastructure_How_OCPP_Backend_Vulnerabilities_Could_Compromise_Your_System

[7] S. R. Team. (2023) Hijacking Charger's Identifier to Cause DoS. [Online]. Available: https://www.saiflow.com/blog/hijacking-chargers-identifier-to-cause-dos/

[8] MQTT, "MQTT Dissector," 2023. [Online]. Available: https://github.com/wireshark/wireshark/blob/master/epan/dissectors/packet-mqtt.c

[9] "Modbus Dissector," 2023. [Online]. Available: https://www.wireshark.org/docs/dfref/m/modbus.html

[10] M. House, "Python OCPP: The Mobility House Implementation," 2025. [Online]. Available: https://github.com/mobilityhouse/ocpp

[11] O. authors, "OpenEVSE WiFi Kit," 2024, openEVSE. [Online]. Available: https://store.openevse.com/products/openevse-wifi-kit?_pos=1&_sid=6c639aa62&_ss=r&variant=14554169092

[12] OCA, "What is OCPP?" 2024. [Online]. Available: https://chargelab.co/industry-advocacy/ocpp

[13] W. Energy, "Open Charge Point Protocol (OCPP) Security Explained," 2025. [Online]. Available: https://wevo.energy/white-papers/open-charge-point-protocol-ocpp-security-explained/

[14] Ampeco, "A Complete Guide to OCPP (Open Charge Point Protocol)," 2025. [Online]. Available: https://www.ampeco.com/guides/complete-ocpp-guide/

[15] I. Fette and A. Melnikov, "The WebSocket Protocol," RFC 6455, 2011. [Online]. Available: https://datatracker.ietf.org/doc/html/rfc6455

[16] W. Foundation, "Wireshark Developer's Guide - Part II. Wireshark Development - Chapter 9. Packet Dissection," 2023. [Online]. Available: https://www.wireshark.org/docs/wsdg_html_chunked/ChapterDissection.html

[17] Lua.org, "Lua: Powerful, Fast, Lightweight, Embeddable Scripting Language," 2025, accessed: January 24, 2025. [Online]. Available: https://www.lua.org/

[18] O. authors, "EmonEVSE WiFi Connected EV Charging Station (Type-2)," 2024. [Online]. Available: https://shop.openenergymonitor.com/emonevse-wifi-connected-ev-charging-station-type-2/

[19] O. Tilmans *et al.*, "IPMininet: A Mininet Extension for Emulating Complex IP Networks." [Online]. Available: https://github.com/cnp3/ipmininet

[20] B. Lantz, B. Heller, and N. McKeown, "A Network in a Laptop: Rapid Prototyping for Software-Defined Networks," in *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*. ACM, 2010, pp. 1–6. [Online]. Available: https://conferences.sigcomm.org/hotnets/2010/papers/a19-lantz.pdf

[21] cjson developer, "lua-cjson," 2023. [Online]. Available: https://luarocks.org/modules/openresty/lua-cjson

[22] jsonschema developer, "jsonschema," 2023, luaRocks. [Online]. Available: https://luarocks.org/modules/membphis/jsonschema

[23] W. Foundation, "Wireshark Expert Information," 2025, accessed: 2025-01-31. [Online]. Available: https://www.wireshark.org/docs/wsug_html_chunked/ChAdvExpert.html

[24] Wolfspeed, "What's Under the Hood: EV Chargers - A Tale of Standards and Many Connectors," 2025. [Online]. Available: https://www.wolfspeed.com/knowledge-center/article/whats-under-the-hood-ev-chargers-a-tale-of-standards-and-many-connectors/

[25] IETF, "Problem Statement and Use Cases of IPv6-based Vehicular Networking in Intelligent Transportation Systems," 2025. [Online]. Available: https://datatracker.ietf.org/doc/html/rfc9365

TABLE 1. CheckOCPP OCPP message compatibility

Message	Version
Heartbeat	1.6, 2.0, 2.0.1
BootNotification	1.6, 2.0, 2.0.1
Authorize	1.6, 2.0, 2.0.1
StatusNotification	1.6, 2.0, 2.0.1
TransactionEvent	2.0, 2.0.1
Reset	1.6, 2.0, 2.0.1
MeterValues	1.6, 2.0, 2.0.1
CancelReservation	1.6, 2.0, 2.0.1
ReserveNow	1.6, 2.0, 2.0.1
ClearCache	1.6, 2.0, 2.0.1
ChangeAvailability	1.6, 2.0, 2.0.1
ClearChargingProfile	1.6, 2.0, 2.0.1
DataTransfer	1.6, 2.0, 2.0.1
SendLocalList	1.6, 2.0, 2.0.1
SetChargingProfile	1.6, 2.0, 2.0.1
TriggerMessage	1.6, 2.0, 2.0.1
UnlockConnector	1.6, 2.0, 2.0.1
UpdateFirmware	1.6, 2.0, 2.0.1
SignCertificate	1.6, 2.0, 2.0.1
InstallCertificate	1.6, 2.0, 2.0.1
CertificateSigned	1.6, 2.0, 2.0.1
DeleteCertificate	1.6, 2.0, 2.0.1
GetLog	1.6, 2.0, 2.0.1
LogStatusNotification	1.6, 2.0, 2.0.1
SecurityEventNotification	1.6, 2.0, 2.0.1
GetInstalledCertificateIds	1.6, 2.0, 2.0.1
ChangeConfiguration	1.6
GetConfiguration	1.6
GetDiagnostics	1.6
RemoteStartTransaction	1.6
RemoteStopTransaction	1.6
StartTransaction	1.6
StopTransaction	1.6
DiagnosticsStatusNotification	1.6
ExtendedTriggerMessage	1.6
SignedFirmwareStatusNotification	1.6
SignedUpdateFirmware	1.6
TransactionEvent	2.0, 2.0.1
RequestStartTransaction	2.0, 2.0.1
RequestStopTransaction	2.0, 2.0.1
NotifyChargingLimit	2.0, 2.0.1
NotifyEVChargingSchedule	2.0, 2.0.1
NotifyEVChargingNeeds	2.0, 2.0.1
NotifyDisplayMessages	2.0, 2.0.1
NotifyCustomerInformation	2.0, 2.0.1
NotifyMonitoringReport	2.0, 2.0.1
NotifyReport	2.0, 2.0.1
SetVariables	2.0, 2.0.1
GetVariables	2.0, 2.0.1
SetNetworkProfile	2.0, 2.0.1
GetReport	2.0, 2.0.1
GetBaseReport	2.0, 2.0.1
GetMonitoringReport	2.0, 2.0.1
GetChargingProfiles	2.0, 2.0.1
ReportChargingProfiles	2.0, 2.0.1
PublishFirmware	2.0, 2.0.1
UnpublishFirmware	2.0, 2.0.1
CostUpdated	2.0, 2.0.1
GetCertificateStatus	2.0, 2.0.1
Get15118EVCertificate	2.0, 2.0.1
ClearDisplayMessage	2.0, 2.0.1
GetDisplayMessages	2.0, 2.0.1
SetMonitoringBase	2.0, 2.0.1
PublishFirmwareStatusNotification	2.0, 2.0.1
SetDisplayMessage	2.0, 2.0.1
SetMonitoringLevel	2.0, 2.0.1
SetVariableMonitoring	2.0, 2.0.1
ClearVariableMonitoring	2.0, 2.0.1
ClearedChargingLimit	2.0, 2.0.1
CustomerInformation	2.0, 2.0.1
NotifyEvent	2.0, 2.0.1
GetTransactionStatus	2.0, 2.0.1