USENIX

THE ADVANCED COMPUTING
SYSTEMS ASSOCIATION

EmuOCPP: Effective and Scalable OCPP Security and Privacy Testing

Soumaya Boussaha, *SAP, EURECOM;* Victor Fresno Gómez, *EURECOM, UPM;*
Thomas Barber, *SAP SE;* Daniele Antonioli, *EURECOM*

https://www.usenix.org/conference/vehiclesec25/presentation/boussaha

This paper is included in the Proceedings of the
3rd USENIX Symposium on Vehicle Security and Privacy.

August 11–12, 2025 • Seattle, WA, USA

978-1-939133-49-6

Open access to the Proceedings of the 3rd USENIX Symposium
on Vehicle Security and Privacy is sponsored by USENIX.

EmuOCPP: Effective and Scalable OCPP Security and Privacy Testing

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

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

Thomas Barber
SAP SE, Walldorf, Germany
thomas.barber@sap.com

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

Abstract

The Open Charge Point Protocol (OCPP) is the de facto standard for communication between electric vehicle charging stations (CS) and charging station management systems (CSMS). However, its security and privacy have been only partially explored, mainly due to the lack of an adequate testing framework. To this end, we introduce EmuOCPP, a new OCPP security and privacy testing framework. The framework is based on container emulation to reproduce real-world OCPP networks with high fidelity and low cost. We discuss our implementation of EmuOCPP, using open-source software (IPMininet) and low-cost hardware.

Using EmuOCPP, we uncover five attacks on OCPP 1.6, 2.0, and 2.0.1. These include man-in-the-middle attacks exploiting OCPP security profile upgrades and downgrades. And CS impersonation attacks leveraging undefined behaviors in the CS boot notification process. We successfully evaluate the attacks across nine targets, including open- and closed-source OCPP implementations, a real CS, and a production network operated by a major company. We discuss the attacks' root causes, including new OCPP design and implementation vulnerabilities. We present effective mitigations to address the discovered vulnerabilities and attacks. We responsibly disclosed our findings with the OCPP consortium and will open source EmuOCPP once the disclosure is completed.

1 Introduction

The Open Charge Point Protocol (OCPP) [15] has become the industry standard communication protocol for interactions between Charging Points (CPs) and Charging Station Management Systems (CSMSs). OCPP has undergone multiple iterations, each improving upon previous versions to address evolving industry requirements. Key versions include OCPP 1.6 [5], OCPP 2.0, and OCPP 2.0.1 [6]. OCPP offers three security profiles to provide some security guarantees, like confidentiality, integrity, and authenticity.

Numerous studies have examined the security and privacy of OCPP, primarily focusing on 1.6 [2, 14, 29, 30, 32, 36, 47]. These works have highlighted critical security concerns, including man-in-the-middle (MitM) attacks, denial-of-service (DoS) threats, and protocol implementation weaknesses. Additionally, privacy-related risks such as tracking attacks and data leakage have been explored [2, 22, 49]. In contrast, research on OCPP 2.0 and 2.0.1 remains significantly limited, with only a few studies addressing some security aspects [4, 31].

The limited research on more recent OCPP versions is due to the lack of a comprehensive OCPP security and privacy testing framework. Existing tools are fragmented, restricting researchers from conducting holistic security analysis. For instance, the Mobility House provides a Python-based OCPP library that supports 1.6 and 2.0.1, but it lacks a full-fledged implementation, requiring additional development effort [34].

Other solutions, such as SteVe, function exclusively as OCPP CSMS for 1.6, with no support for 2.0 or 2.0.1 [51]. OpenEVSE, an open-source EV charging platform, does not support OCPP 2.0.1 or security profiles, limiting its utility in security testing [43]. SAP's OCPP simulator is a Node.js-based application that only supports OCPP1.6 [28]. While OCPPStorm [21] provides a fuzzing framework to test CSMSs running 1.6 and 2.0, it does not cover 2.0.1 and cannot test charging stations. This lack of coverage presents a significant risk: OCPP is a standard protocol that could be exploited to compromise the security and privacy of millions of devices.

To address these issues, we present *EmuOCPP*, a comprehensive security and privacy testing framework for OCPP. The framework supports OCPP *1.6, 2.0, 2.0.1* and its three security profiles. EmuOCPP enables *large-scale emulation* of OCPP networks via containers, including hundreds of CSs controlled by one or more CSMSs. It is low-cost and easy to reproduce, requiring a single Linux kernel and open-source software.

EmuOCPP facilitates discovering and testing OCPP design and implementation vulnerabilities and attacks. It allows the creation of emulated network attackers with arbitrary capabilities, like impersonation, MitM, eavesdropping, or DoS.

The emulated attacker can run real-world tools, like Nmap or Scapy, and target real-world OCPP client and server stacks, including proprietary and open-source ones.

EmuOCPP is implemented using *IPMininet* [50] for network emulation, enabling large-scale testing of OCPP deployments while reducing computational overhead. Unlike prior testbeds relying on multiple virtual machines [47], our approach allows deploying hundreds of charging stations efficiently. A *DNS server* enhances realism by managing CS registration. The *Messages* module extends the *Mobility House* Python OCPP library to support both OCPP 1.6 and 2.0. EmuOCPP implements seventeen OCPP messages (Table 1) and custom functionalities, including CSO-to-CSMS interactions. Configurations are defined via a YAML-based *Config* module, specifying network topology, security settings, OCPP versions, and providing a convenient graphical user interface.

Using EmuOCPP, we uncover *five attacks* against OCPP and test two known ones. Among the new findings, some attacks exploit weaknesses in the protocol's security profile upgrade and downgrade mechanisms, including one that leverages an implementation vulnerability to force a security profile downgrade. We also identify impersonation attacks that exploit ambiguities in charging station identifier management, allowing us to impersonate CSs. These attacks affect all OCPP security profiles regardless of the OCPP version. Some uncovered attacks have been disclosed to and acknowledged by the Open Charge Alliance (OCA), and may be addressed in future protocol updates.

We evaluate the attacks and the performance of EmuOCPP empirically. We confirm that the attacks are effective on nine OCPP targets implementing 1.6, 2.0, 2.0.1 and providing SP1, SP2, SP3, or no SP. The target set includes open-source and closed-source OCPP implementations, an open-source charger, and a production OCPP network. We experimentally confirm that EmuOCPP satisfies its design requirements. Moreover, we conducted a performance evaluation proving the lightweight nature of EmuOCPP, enabling the simultaneous deployment of hundreds of devices. Finally, we isolate the attacks' root causes and discuss effective mitigations.

We summarize our contributions as follows:

- We present EmuOCPP, a configurable, lightweight OCPP security and privacy testing framework. It supports OCPP 1.6, 2.0, and 2.0.1 and the three OCPP security profiles and enables testing real-world and heterogeneous OCPP implementations. We implement EmuOCPP with open-source software and low-cost hardware.
- We uncover five attacks exploiting issues in the OCPP standard, including weak SP upgrade procedures and undefined downgrade and duplicate CS ID behaviors.
- We successfully conducted the attacks against nine OCPP targets, including open and proprietary implementations spanning all SPs and relevant OCPP versions. We isolate the vulnerabilities enabling the attacks and discuss effective defenses.

Ethics, Disclosure, and Availability. We conducted our experiments ethically in a controlled environment. We tested virtual devices, physical devices we own, and an OCPP production network with explicit authorization from its owner. We responsibly disclosed our findings to the OCPP standard body and open-source our tool once the disclosure is completed. We provide our toolkit in a public Github repository at: `https://github.com/vfg27/EmuOCPP`.

2 OCPP Preliminaries

OCPP is the de facto standard for communication between Charging Points (CPs) and Charging Station Management Systems (CSMS) [42], which provide the components needed to monitor, control, and optimize charging station performance. OCPP also considers the charging station operator (CSO) role, which oversees the charging infrastructure's deployment, maintenance, and operation. It enables convenient CS functionalities, such as allowing users to reserve a CS or securing certificates into a CS. OCPP is specified in an open standard maintained by the *Open Charge Alliance (OCA)*, a large consortium of leading EV companies [9]. Next, we discuss OCPP's most relevant versions (1.6, 2.0, and 2.0.1) messages and security profiles.

2.1 Versions

OCPP 1.6, released in 2015, marked a significant step forward by standardizing the core communication functionalities required for effective EV charging infrastructure. It introduced features like smart charging and enhanced support for different transaction types.

OCPP 2.0 was launched in 2018, bringing substantial enhancements, including support for new functionalities like improved security features. However, it was fastly replaced by 2.0.1 in just 2 years due to functional limitations.

OCPP 2.0.1 was released in 2020 to address issues identified in 2.0 and to introduce additional features to support advanced use cases, such as vehicle-to-grid (V2G) technology.

2.2 Messages

OCPP enables communication via a standardized set of messages consisting of several fields [26]. The messages can be implemented as JavaScript Object Notation (JSON) objects (1.6, 2.0), denoted OCPP-J, or SOAP (1.6 only).

An OCPP message is either a request or a response. In OCPP 1.6, requests are denoted with `.req`, and responses with `.conf`. There is no dot notation in OCPP

2.0 and 2.0.1, but this information is encoded in the message name, e.g., `BootNotificationRequest` other than `BootNotification.req`. *Core* messages manage essential functions required for CS and CSMS operations and transactions, such as starting a CS, sending heartbeats, or notifying device status. *Security* related implemented messages deal with the handling and installation of CS and CSMS certificates. *Configuration and management* messages retrieve or modify configuration data in the CS or the CSMS. Similarly, *remote control* messages trigger actions on the CS, such as sending a heartbeat or requesting a BootNotification. Next, we describe common OCPP messages (complete list in [25]).

`BootNotification`. When a CS initializes or reboots, it sends a BootNotification to inform the CSMS. This message includes details such as the CS model, vendor, and firmware version, enabling the CSMS to recognize and register the CS appropriately.

`Authorize`. Before initiating a charging session, the CS sends an authorization request to the CSMS to verify the user asking to charge is authorized. The CSMS responds with acceptance or rejection, ensuring that only authorized users can commence charging.

`StartTransaction`. Upon successful authorization and connection to the EV, the CS issues a StartTransaction message to notify the CSMS that a charging session has begun. This message contains information such as the connector ID, user ID tag, and the initial meter reading, establishing the context for the session.

`TriggerMessage`. Certain actions performed by the CS can only be executed upon receiving a command from the CSMS. The TriggerMessage allows the CSMS to initiate various actions, from triggering a BootNotification to generating a new CS certificate.

2.3 Security Profiles

OCPP supports *three security profiles (SP)*. The SPs were introduced with 2.0, refined in 2.0.1, and backported to 1.6. A secure OCPP network is typically configured with SP1, which might be *upgraded* to SP2, which in turn might be upgraded to SP3. The OCPP standard specifies some upgrade logic, while some details are left to vendors, like how certificate provisioning and validation are handled.

SP1: Password-based CS Auth. SP1 requires password authentication for the CS boot notification process. When it comes online, a CS must present a password to authenticate to the CSMS. However, SP1 does not protect data in transit, as there is no secure channel between the CS and CSMS, and it does not authenticate the CSMS.

SP2: TLS and CSMS Cert Auth. SP2 enhances SP1 by adding TLS and certificate-based CSMS (server) authentication. During the boot process, the CSMS provides a TLS certificate to the CS, signed by a CA, that can be the CSO. The CS (client) authenticates the CSMS, establishes a TLS session with the CSMS, and then authenticates itself using a password.

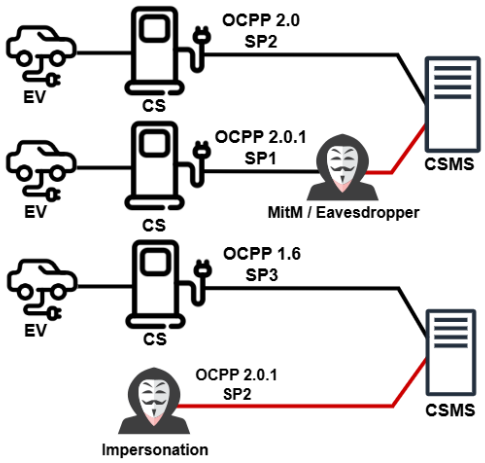

Figure 1: EmuOCPP design overview. The tool emulates an OCPP network, including OCPP clients and servers, using different OCPP stacks and configurations and attackers like eavesdroppers, spoofers, and MitM.

SP3: TLS and Mutual Cert Auth. SP3 extends SP2 with certificate-based CS (client) authentication. It is the strongest OCPP SP. During the boot process, the CS and CSMS mutually authenticate a TLS session, providing trusted certificates. SP3 gets rid of passwords, so it is not affected by effective password attacks.

3 Threat Model

System model. We consider an OCPP charging network composed of CSs and CSMSs like the one presented in Figure 1. Each network host is configured to run as an OCPP client or server. Multiple OCPP versions and SPs can coexist in the network. CPs and CMSMs can run open-source or proprietary OCPP implementations, like the SteVe CSMS [51].

The CSMS manages and monitors the CS state, including charging activities, firmware updates, and charging reservations. The CS can authenticate to CSMS through a boot notification message, send heartbeats to maintain connection and start the transaction to enable the EV user to charge their car.

Attacker model. We consider an attacker who can access the charging network and send OCPP packets to CSs and CSMSs. We focus on *four* attacker models and related threats: i) *eavesdropper* sniffing (encrypted) OCPP packets, ii) *spoofer* impersonating a CS to a CSMS or vice-versa, iii) *MitM* positioning between a CS and a CSMS, and iv) *DoS* performing DoS attack on a CS or a CSMS.

The attacker wants to compromise the security and privacy of the OCPP network. They try to violate confidentiality, integrity, availability, and authenticity. Moreover, they attempt

to compromise unlinkability, anonymity, and non-trackability privacy guarantees.

The active attacker models are a Dolev-Yao adversary [27] who can modify, drop, replay, reassemble, and create OCPP packets. The attacker can also gather public information about a host, like a CS ID printed as a QR code on the device or using open-source intelligence resources.

The attacker tries to exploit OCPP *design* and *implementation* flaws. The former includes protocol-level vulnerabilities affecting one or more OCPP versions and SPs. The latter involves mis-implementation of OCPP, potentially leading to undefined behavior.

The attacker can exploit misconfigured OCPP networks where CSs and CMSMs are exposed to the internet and remotely compromise them using known or zero-day vulnerabilities like the ones presented in [47]. Or, they can connect to an OCPP local area network, e.g., by plugging an Ethernet cable or compromising a weak Wi-Fi password, and then observe the (unprotected) OCPP traffic and inject packets.

The attacker cannot physically damage or tamper with the CS or interact with a device, such as plugging into the charging ports or accessing internal hardware components. This constraint is significant because it means the attacker must rely on non-physical methods to exploit the system. Moreover, they cannot break standard cryptographic mechanisms, including TLS and its PKI.

4 EmuOCPP

Next, we motivate the need for EmuOCPP, introduce its six requirements, and explain how we address them with our design.

4.1 Motivation

Numerous papers have explored the *security* and *privacy* of OCPP 1.6. These studies have examined vulnerabilities, security challenges, and potential mitigation strategies [2, 14, 29, 30, 32, 36, 47]. They highlight security concerns such as MitM and DoS attacks and implementation weaknesses. Other research focused on OCPP privacy concerns [2, 22, 49].

Only a few research studies have investigated OCPP 2.0 and 2.0.1 security and privacy, such as [3, 31]. Alcaraz et al. [3] conducted a threat analysis of OCPP 2.0.1 using the STRIDE [37] and DREAD [40] threat modeling methodologies, identifying potential vulnerabilities. Similarly, Garofalaki et al. [31] provided an overview of security issues in OCPP-based smart charging scenarios, highlighting areas requiring further research. However, these studies do not offer tools for testing and validating the security of OCPP 2.0.1.

A notable study that presents an OCPP testing tool is OCPP-Storm [21], which introduces a black-box fuzzer to test CSMS implementations up to version 2.0.1. However, in the study, the authors mentioned their inability to test real CS implementations for 2.0.1, as none were open-source. Additionally, the study was able to test only against OCPP.Core, a CSMS that partially supports OCPP 2.0.1 [19]. Hence, a security and privacy testing framework for OCPP is needed and should cover the latest OCPP versions and all security profiles.

Several open-source OCPP software and hardware can be building blocks for an OCPP security and privacy framework. An open-source OCPP Python library from The Mobility House [34] supports OCPP 1.6, 2.0, and 2.0.1 and provides a CS and CSMS skeleton code. SteVe [51], an open-source CSMS, functions only for OCPP 1.6. OpenEVSE is an open-source hardware and software platform for EV charging stations [43], but it lacks support for OCPP 2.0.1 and security profiles. SAP's OCPP simulator [28] is a Node.js application designed to simulate CSs running OCPP 1.6. OCPP.Core [20] is a CSMS that supports deploying different OCPP 1.6, 2.0, and 2.0.1.

4.2 Design

This work presents **EmuOCPP**, a new and comprehensive security and privacy testing framework for OCPP. The framework has *six* requirements:

R1: Supports OCPP 1.6, 2.0 and 2.0.1.

R2: Supports OCPP SP1, SP2 and SP3.

R3: Emulates realistic OCPP networks where hosts run different OCPP implementations.

R4: Tests OCPP security and privacy at the design and implementation level, including eavesdropping, spoofing, MitM, DoS, and tracking issues.

R5: Scalable to complex topologies and thousands of hosts.

R6: Reproducible, open-source, and low-cost.

Figure 1 provides an overview of EmuOCPP's design. The design integrates a new OCPP emulator supporting OCPP network components, like CS and CSMS, topologies, and traffic shapes. The emulator allows setting bandwidth and packet drop rates to reproduce real-world network conditions where packets might be dropped or experience latency (**R3**).

The emulated hosts can run any OCPP version and SP (**R1**, **R2**). Hence, we can reproduce insecure and secure networks, including their OCPP SP upgrades. For example, in Figure 1, we show an emulated network where CSs and CSMSs operate with OCPP 1.6, 2.0, 2.0.1, and SP1, SP2, and SP3.

EmuOCPP emulation strategy is *container-based*, an effective and lightweight alternative to emulation based on virtual machines (VMs) [33]. Containers allow the emulation of thousands of hosts, leading to higher performance while keeping a realistic environment and supporting hardware in the loop. Unlike VMs, which require separate operating systems,

containers share the host system's kernel, allowing multiple isolated user-space instances to run simultaneously.

EmuOCPP enables extensive security and privacy testing by configuring emulated hosts as attackers (**R4**). As shown in Figure 1, an emulated attacker can operate as a MitM, Eavesdropper, or Impersonator, targeting vulnerabilities across different OCPP versions and security profiles. The attacker can leverage real-world security and privacy tools such as Nmap, Scapy, Mitmproxy, and OCPPStorm to enumerate, manipulate traffic, and exploit protocol weaknesses.

For example, EmuOCPP can automatically establish a MitM position by attacking the IPv6 Neighbor Discovery Protocol (NDP) at the link layer, then escalate to tampering with OCPP packets at the application layer, leading to session hijacking, leaked private data and packet manipulation. These capabilities allow programmatic, systematic, and reproducible security and privacy testing of OCPP.

To achieve scalability (**R5**), EmuOCPP employs a file-based configuration strategy using Yet Another Markup Language (YAML). The config files set the network, including the topology, hosts, CSs, CSMSs, switches, routers, and links. Moreover, they configure which OCPP client and server implementations are running in each host, their OCPP version, and supported SP. This allows the testing of open-source and closed-source OCPP implementations programmatically.

Moreover, EmuOCPP is based on open-source software and low-cost hardware; hence, it can be reproduced at low cost (**R6**). For example, it can be installed on any Linux-based distribution or subsystem.

5 Implementation

In this section, we discuss the implementation of EmuOCPP which is based on four modules: i) *IPMininet* for network emulation, ii) *Tests* for providing attacks tooling iii) *Messages* providing automated OCPP messages parsing and validation, iv) *Config* a configuration engine based on YAML files and a GUI.

5.1 IPMininet

To evaluate the security of OCPP implementations and explore potential attack scenarios, we developed an OCPP emulator using *IPMininet* [50], an extension of Mininet [38] that supports IPv6. IPMininet enables the creation of a virtual network that closely mimics real-world OCPP deployments, facilitating large-scale and low-cost emulation of OCPP communication between CS, CSMS, and other hosts.

Unlike prior OCPP testbeds that relied on multiple virtual machines [47], our approach leverages IPMininet's lightweight architecture to deploy *hundreds of charging stations* with minimal resource consumption. This makes our testbed suitable for large-scale testing and stress analysis. Additionally, we incorporate a *DNS server* into the emulator to enhance realism and expand potential attack vectors. The DNS server manages CS registration, resolves CSMS addresses dynamically, and enables testing of attacks.

EmuOCPP also provides a script for generating certificates. When deploying a CS using SP3, a CS certificate is automatically created and installed on the device. Additionally, this script allows implementers to test their own CSMS certificates. These certificates are signed by a fictional CSMS, for which a separate script is available to generate its corresponding certificate.

5.2 Test

EmuOCPP's Test module implements two MitM attacks. It provides IPv6 NDP MitM using `parasite6` and `flood_router26` from the THC toolkit [52]. The `parasite6` tool acts as an ICMPv6 Neighbor Solicitation/Advertisement spoofer, allowing attacks similar to ARP spoofing on IPv4 networks. Meanwhile, `flood_router26` generates random router advertisements to disrupt the target's routing table.

Moreover, it includes a script using `mitmdump` [41] to MitM a connection protected by SP1, and maintains this access for SP2 and SP3. During execution, it listens to incoming messages and displays the captured information. In addition, it monitors specific messages and waits for opportunities to modify them, such as the first message sent from the CSMS to the CS. The script allows the installation of a certificate on the target CS or retrieving a valid CS certificate from the CSMS using the information sent from the CS.

5.3 Messages

This section discusses the custom OCPP implementation we built and its protocol coverage. We also discuss implementing the CSO and other network components necessary to emulate our test scenarios. Please note that EmuOCPP can run open-source or proprietary programs as described in Section 5.1.

To develop the functional behavior of CPs and CSMSs, we extended the *Mobility House* Python OCPP library. This library provides predefined schemas for OCPP messages and validation tools. However, since more recent library versions removed support for OCPP 2.0, we used an older version to maintain compatibility with OCPP 1.6 and OCPP 2.0. When the project was initiated, this was the most recent version available.

We developed two Python scripts to test the interactions between CS and CSMS. These scripts support key security operations, like certificate installation or password modification Of the 64 messages in OCPP 2.0 and 2.0.1, and the 39 in OCPP 1.6, EmuOCPP implements seventeen of them covering the Core, Security, Config&Mgmt, Remote control, and Reservation categories. The list of messages is shown in Table 1.

Table 1: EmuOCPP implements seventeen messages spanning all OCPP versions and five categories.

Message	Version	Category
Heartbeat	1.6, 2.0, 2.0.1	Core
BootNotification	1.6, 2.0, 2.0.1	Core
Authorize	1.6, 2.0, 2.0.1	Core
StatusNotification	1.6, 2.0, 2.0.1	Core
TransactionEvent	2.0, 2.0.1	Core
Reset	2.0, 2.0.1	Core
SignCertificate	1.6, 2.0, 2.0.1	Security
InstallCertificate	1.6, 2.0, 2.0.1	Security
CertificateSigned	1.6, 2.0, 2.0.1	Security
GetVariables	2.0, 2.0.1	Config&Mgmt
SetVariables	2.0, 2.0.1	Config&Mgmt
SetNetworkProfile	2.0, 2.0.1	Config&Mgmt
GetConfiguration	1.6	Config&Mgmt
ChangeConfiguration	1.6	Config&Mgmt
TriggerMessage	2.0, 2.0.1	Remote control
ExtendedTriggerMessage	1.6	Remote control
ReserveNow	1.6, 2.0, 2.0.1	Reservation

Reservation messages reserve a CS for a vehicle. In this scenario, an EV owner submits a reservation request to book a CS through a dedicated mobile application. The CSMS, in turn, processes it and sends the `reservationNow` message to the corresponding CS with the user token who booked the CS for a given time window.

We also extended the Mobility House implementation to allow communication between a CSO and CSMS (which is not covered by the OCPP standard), emulation of an EV reservation made by a user, and better handling of DNS. Next, we describe these three new features. In EmuOCPP the CSO establishes a WebSocket connection to the CSMS and can issue specific commands to initiate various processes, such as configuration changes or certificate installation requests.

We developed an emulated EV reservation process via a dedicated HTTP API, allowing clients to retrieve and post to a related database. Via the API, a client communicates with the CSMS to replicate the reservation made by an EV user, e.g., a reservation via a mobile application. The CSMS listens for reservation requests and, upon registering an EV, sends a reservation message to the CS. CSs and CSMSs must interact with a DNS server before they can operate. CSMSs register by sending their domain, IP, and port. CSs query the DNS to resolve the CSMS address and obtain the necessary connection details. The DNS server balances its connections if multiple CSMSs register with the same domain name.

5.4 Config

We build a configuration engine based on a *network configuration file*. As shown below, the file configures CS clients (`client0`, ...), each client with a custom OCPP configuration (`SecProfile`, `version`). It configures one or more CSMS servers and attacker nodes. It sets the number of routers, the link shape, and the network topology.

```
clients:
  client0:
    SecProfile: '3'
    priority: [0, 1]
    profiles:
      0: { SP: 2, OCPP_version: OCPP201 }
      1: ...
    version: 2.0.1
    ...
routers:
  router0: { name: R0 }
servers:
  server0: { multiple: 1, ... }
links:
  - [CLI0, R0]
    ...
```

Once the network topology is defined, additional configurations are automatically managed to streamline the process. These include generating and assigning serial numbers, passwords, and security certificates. However, users can manually adjust these settings before execution to customize the emulation to their needs.

To further simplify the creation and testing of network topologies, we developed a EmuOCPP GUI using *Tkinter* [45] This interface allows users to: i) add, delete, and reposition CSs, CSMSs, routers, switches, DNS servers, and regular hosts, ii) define key configuration parameters for the CS, including OCPP version and security settings, through a visual interface and iii) generate configuration files for automated deployment, reducing the need for manual scripting. A GUI screenshot is shown in Figure 13.

6 Attacks

We present *eight attacks* we discovered with the help of EmuOCPP. As explained in Section 3, they require an attacker who can access the OCPP network. Notably, *five attacks are novel*: M2, M3, and M4 enable MitM any SP by exploiting SP upgrades and downgrades. I1 and I2 allow the impersonation of arbitrary CSs by exploiting vulnerabilities in the BootNotification process.

6.1 Summary

Table 2 summarizes the eight attacks discovered by EmuOCPP. They include eavesdropping (E1), impersonation

Table 2: We discover eight attacks on OCPP involving design and implementation vulnerabilities across all OCPP versions and SPs. The M2, M3, M4, I1, and I2 attacks are novel. Des: Design, Imp: Implementation.

ID	Name	New	Type	OCPP SP	OCPP Ver.	Impact	Vulnerability
M1	MitM SP1	✗	Des	SP1	1.6, 2.0, 2.0.1	Sec, Pri	Weak SP1
M2	MitM SP2 Upgrade	✓	Des	SP2	1.6, 2.0, 2.0.1	Sec, Pri	Weak SP2 upgrade
M3	MitM SP3 Upgrade	✓	Des	SP3	1.6, 2.0, 2.0.1	Sec, Pri	Weak SP3 upgrade
M4	MitM SP Downgrade	✓	Imp	SP2, SP3	1.6, 2.0, 2.0.1	Sec, Pri	No SP down prot
E1	Eavesdrop	✗	Des	SP1	1.6, 2.0, 2.0.1	Sec, Pri	Weak SP1
D1	CSMS (D)DoS	✗	Des	SP1, SP2, SP3	1.6, 2.0, 2.0.1	Sec	Unauth CS boot
I1	CS Impersonation	✓	Des, Imp	SP1, SP2, SP3	1.6, 2.0, 2.0.1	Sec, Pri	CS ID undefined behaviour
I2	CS Impersonation	✓	Des, Imp	SP1, SP2, SP3	1.6, 2.0, 2.0.1	Sec, Pri	CS status trackable by anyone

(I1), MitM (M1–M4), and DoS (D1) threats. They isolate OCPP design and implementation vulnerabilities, including OCPP SP downgrades and undefined behaviors with duplicate OCPP IDs. They result in severe security and privacy issues, like stealing private credentials, charging fraud, and distributed DoS. They affect all OCPP versions and SPs.

We discovered them by studying the OCPP specification [5, 6], a public OCPP test suite [7, 8], and using EmuOCPP as a fast and configurable testing framework. Next, we describe the attacks and their root causes.

6.2 MitM SP1 (M1)

M1 is a MitM attack against OCPP networks using SP1 or no SP. In both scenarios, there is no confidentiality or authentication (i.e., no TLS). The attacker gets a MitM position by exploiting unprotected network layer or link layer protocol vulnerabilities according to its position in the OCPP network, including IPv4 ARP or IPv6 NDP spoofing.

For example, in Figure 7, the adversary positions in the middle by sending unauthenticated IPv6 Neighbor Advertisement (NA) and Router Advertisement (RA) packets to manipulate the network routing. Then, they can observe all OCPP packets exchanged by the victims, block them, and force the CS and CSMS to re-establish an insecure OCPP connection while the attacker is in the middle.

The M1 attack root cause is a design flaw in SP1, which neither protects data in transit nor authenticates the CSMS. SP1 is not recommended for production systems, but related work has demonstrated that it is widely deployed [47].

6.3 MitM SP2 Upgrade (M2)

An attacker who conducted M1 can *keep* the MitM position even if the victims upgrade to SP2 (TLS with server certificate). Before the CSMS starts the upgrade to SP2 with `ChangeConfiguration` (OCPP 1.6) or `SetVariables` (OCPP 2.0 and 2.0.1), the attacker sends a `InstallCertificate` message, as shown in Figure 2, to install a malicious root certificate on the CS. Once SP2 is in

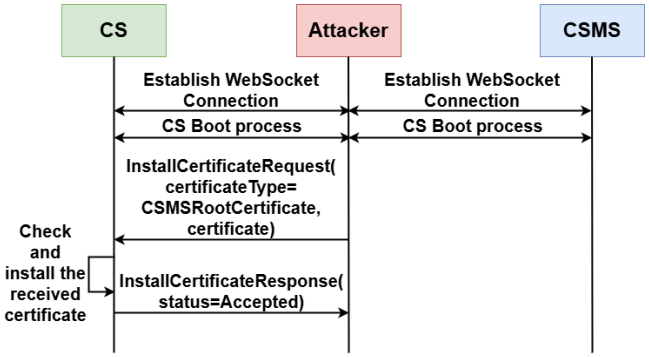

Figure 2: M2 MitM SP2 via malicious CSMS certificate upgrade.

place, the attacker can authenticate to the CS using the malicious certificate and MitM the TLS connection.

For this to work, the CS needs to have *AdditionalRootCertificateCheck* set to False. If the flag is True, the CS allows only one CSMS Root Certificate (plus a temporary fallback) at a time. Any new certificate must replace the existing one and be signed by it, ensuring a secure and verifiable chain of trust for certificate management.

For SP1, the use of *AdditionalRootCertificateCheck* is optional, and there is no requirement for its configuration. However, in SP2 and SP3, this configuration variable is mandatory. Despite its required presence, the OCPP standard does not mandate the implementation of the feature associated with this variable. Instead, it remains an optional feature that CS can choose to support or not and would still be compliant.

The M2 attack root cause is a design flaw in the OCPP SP2 upgrade process, which allows the installation of *self-signed* root certificates when *AdditionalRootCertificateCheck* is disabled. This design vulnerability impacts any OCPP implementation, even those with an OCPP security certification.

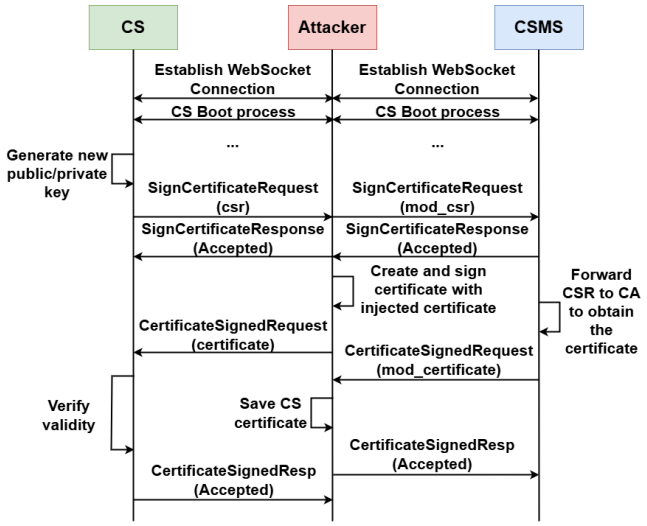

Figure 3: M3 MitM SP3 via client certificate upgrade manipulation.

6.4 MitM SP3 Upgrade (M3)

An attacker who conducted M2 can *maintain* the MitM even if the victims upgrade to SP3 (TLS with client and server certificates). Before the CSMS starts upgrading to SP3 with the same message as the upgrade to SP2, the attacker obtains a valid client certificate from the CSMS by completing a *client certificate enrollment process* as shown in Figure 3. The attacker intercepts the `SignCertificateRequest` message from the CS and substitutes the victim Certificate Signing Request (`csr`) with their own (`mod_csr`).

Then, the CSMS creates and sends the attacker a client authentication certificate (`mod_certificate`). Moreover, the attacker creates and sends to the victim a client certificate (`certificate`) that the victim employs to authenticate to the attacker. After manipulating the client certificate, the attacker can establish two parallel and mutually authenticated TLS sessions with the CS and CSMS using SP3 and keep the MitM position.

The M3 attack root cause is a design issue in the OCPP SP3 upgrade process, which allows any CS to obtain valid TLS client certificates from the CSMS rather than from a trusted third-party CA. This design vulnerability impacts any OCPP implementation, like the one discussed for M2.

6.5 MITM SP Downgrade (M4)

An attacker can downgrade SP3 to SP2 or SP2 to SP1 by selectively dropping OCPP packets, even if they are encrypted. The attacker requires a link-layer MitM position between the CS and CSMS, which can be achieved using an IPv6 ND spoofing attack. Per our threat model, the attacker does not need to compromise any TLS session. Then, as shown in Figure 4, the attacker can observe the SP in use via the WebSocket header and programmatically block secure connection until a weaker security profile is in use, e.g., force SP1 other than SP2 or SP3. The SP downgrade attack has critical real-world consequences. For example, it enables the downgrade of a connection encrypted with SP3 to one unencrypted using SP1 to steal user credentials and other (sensitive) data.

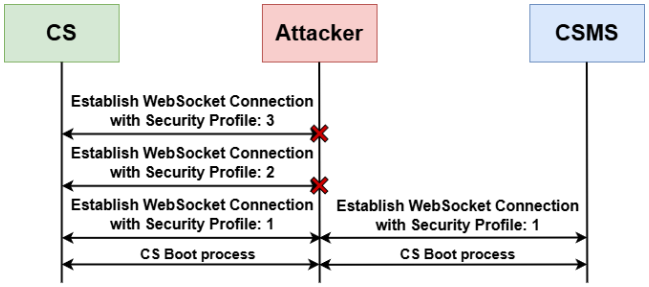

Figure 4: M4 SP3 or SP2 downgrade to SP1 MitM attack.

The downgrade effectiveness depends on the CS configuration of its Connection Profiles. A Connection Profile defines how a CS connects to a CSMS and has a priority indicated in the `NetworkConfigurationPriority` variable. Each Connection Profile uses a specific SP. The Connection Profiles are used by priority and can have a failure count. Hence, it is expected that CSs will support three Connection Profiles with SP1, SP2, and SP3 to provide maximal backward compatibility regardless of the CSMS.

M4 stems from an OCPP implementation vulnerability. OCPP mandates deleting pre-configured network profiles with lower SPs after an upgrade to a higher security profile. Failure to do so introduces inconsistencies that can be exploited to downgrade a connection, as demonstrated in M4. During the responsible disclosure process, the discussion with OCA concluded with a recommendation of potentially adding specific test use cases for this scenario while testing for compliance.

6.6 Eavesdrop (E1)

The attacker can eavesdrop on OCPP plaintext packets when SP1 is used, or there is no SP. The attacker can sniff OCPP packets in several ways, including compromising a network switch or router, adding a wiretap, or performing a MitM attack at the link or network layer against IPv6. For example, they can perform a *Neighbor Advertisement (NA) spoofing* attack against IPv6's *Neighbor Discovery Protocol (NDP)*, which is the equivalent of Address Resolution Protocol (ARP) spoofing for IPv4.

With E1, the attacker can access security and privacy-critical OCPP data, including user authentication tokens, CS passwords, and charging event details. We note that with SP1, it is also trivial to impersonate a CS, as the attacker can sniff its password. The E1 attack root cause is insecure SP1 design as for M1.

6.7 CSMS DoS (D1)

As depicted in Figure 6, the attacker can perform a DoS attack on a CSMS by presenting themselves as a CS to a victim CSMS and opening multiple OCPP connections *without* having to authenticate. They start multiple connections to the victim CSMS using *unauthenticated* `BootNotification`. The attacker crafts fake authentication credentials or certificates to flood the CSMS resources with authentication attempts. Thus, D1 is effective regardless of the target CSMS SP as the attacker does *not* need the CS password or certificate to perform the DoS. Moreover, D1 can be scaled to a *Distributed DoS (DDoS)* by an attacker controlling multiple hosts and using them to target one or more CSMSs (e.g., via a CS botnet).

The root cause of D1 is a deployment implementation vulnerability stemming from the lack of sufficient rate limiting and resource protection. This vulnerability allows an attacker to overwhelm the CSMS with an unbounded number of connection and authentication attempts. DoS and DDoS attacks are relevant for OCPP as they can prevent people from charging their vehicles and cause damage to the charging network. Related work has demonstrated that many production systems still suffer from these vulnerabilities [47].

6.8 CS Impersonation (I1)

I1 allows an attacker to impersonate a CS to a CSMS, sometimes even if the impersonated CS communicates with the CSMS. The attacker needs the impersonated CS vendor name, model, and charging point serial numbers. This information is usually public, e.g., printed on the CS as shown in Figure 14 or stored in an Internet-accessible database [23]. Then, the attacker sends a `BootNotification` request to the CSMS while spoofing a CS while the impersonated CS is already connected to the CSMS.

Our experiments (discussed in Section 7) show that a CSMS can: i) accept the attacker connection while keeping the one with the impersonated CS, ii) terminate the connection with the impersonated CS, and start a new one with the attacker, or iii) reject the connection with the attacker. We assume the attacker can connect with the CSMS, abusing the serial number flaw.

When no SP is used, impersonation is trivial, as there is no authentication of the CS. Under SP1, impersonation remains feasible since credentials are transmitted in plaintext. While SP2 and SP3 offer stronger protections, they are not entirely foolproof: attackers could exploit implementation vulnerabilities on the CSMS, such as SQL injection, to bypass authentication and impersonate a CS. As such, the I1 attack remains viable across all SP levels, as SPs only mitigate but do not address the underlying design flaw in how the protocol manages CS identity.

Impersonating a CS via I1 leads to critical security and privacy breaches in real-world deployments. For instance, an attacker can receive a `ReserveNowRequest` from a victim user, which includes the user's authentication token, as shown in Figure 8. The attacker could then recharge their vehicle using the victim's token, effectively committing fraud by shifting the charging cost to the victim user.

The root cause of I1 lies in a limitation of the OCPP standard regarding the management of CS IDs, for instance the protocol do not specify whether multiple CS can have the same ID or not.This has led most implementers to use static identifiers that are publicly accessible, for example, through physical labels attached to the CS as in Figure 14. Furthermore, the standard is light on providing guidelines on how authentication should be tied to CS IDs. An implementer would not know how to handle parallel CS connection requests using the same serial ID. As a result, different CSMS implementations react inconsistently to such impersonation attempts. The OCA acknowledged this issue during our disclosure.

6.9 CS Impersonation (I2)

I2 is a CS impersonation attack that improves upon I1 in that it is effective even when the CSMS accepts only one connection per CS serial ID. The attacker monitors the target CS, e.g., by periodically pinging it, to detect when the CS goes offline. This can occur, for example, if the CS reboots for a firmware update. Only once the attacker has detected that the CS is disconnected from the CSMS, they exploit this time window to send a `BootNotification` request, impersonating the target CS.

Since the CSMS registers the first connection it receives for that serial ID, it accepts the attacker's fake CS as legitimate, thinking it is the legitimate charger that has finished rebooting. When the victim CS attempts to reconnect, the connection has already been stolen, leaving the legitimate CS unable to re-establish communication with the CSMS.

The root cause of I2 is twofold: First, as in I1, the OCPP protocol lacks clear guidance on how CS IDs should be securely managed. This leads implementers to rely on static and publicly accessible identifiers, which attackers can easily obtain. Second, CS status is trackable by any host in the OCPP network regardless of the security profile in use. Many CSs respond to unsolicited requests such as pings, enabling attackers to infer when the CS is rebooting and time their impersonation accordingly.

7 Evaluation

We describe our evaluation setup and the attacks and EmuOCPP performance evaluation results. In our experiments, we successfully exploit nine OCPP targets and introduce them in Appendix A. The scripts for executing the evaluated attacks are at `https://github.com/vfg27/EmuOCPP` in the `charging/scenarios` folder.

Table 3: Emulated network specs. B: bandwidth, D: delay.

Link	B (Mbps)	D (ms)
Switch-Router	1000	1
Router-Router	100	15
Host-Switch	1000	-

7.1 Setup

We set up EmuOCPP in different configurations to evaluate our attacks. Figure 9 shows the topology we used to test M1–M4 and E1, while Figure 10 illustrates the one used to test I1 and D1. In both topologies, the target CS and CSMS are separated by two routers. This choice is motivated to emulate a realistic deployment where the CS and CSMS are in 2 different LANs separated by a WAN.

Table 3 shows the emulated network bandwidth and delay, which align with typical network configurations. A Local Area Network (LAN) connection often features bandwidths of 1000 Mbps with latencies under 1 ms. While a Wide Area Network (WAN) generally offers lower bandwidths, such as 100 Mbps, with higher latencies around 15 ms [1].

This emulated topology reflects realistic LAN and WAN conditions, ensuring that the security evaluations and attack scenarios are not limited to a specific environment. As a result, the attacks tested within this setup can be generalized to real-world deployments, as they accurately capture the network characteristics and constraints commonly encountered in operational OCPP networks.

We purchased an *OpenEVSE charger* [43] that supports OCPP 1.6. We configured a network where an Ubuntu machine acts as the attacker and is connected to the same OCPP network as the charger. While configuring the device, we realized that it does *not* support any SPs.

We also deployed three VMs running *SteVe*, *Open E-Mobility* and *OCPP.Core*, three popular open-source CSMS implementations. SteVe and E-mobility [28, 51] support OCPP 1.6 while OCPP.Core supports both 1.6 and 2.0. With additional engineering effort, we plan to extend EmuOCPP in future versions to support the automated deployment of these CSMS implementations as customizable CSMS in the topologies that can be built through EmuOCPP.

Finally, we tested the I1 and D1 attacks on a production network running OCPP 1.6 and SP2. For confidentiality reasons, we do not disclose the name of the company managing the network. We got explicit permission from them to conduct our tests.

7.2 Attacks Results

Summary. Table 4 summarizes our attack evaluation results against *nine* OCPP implementations. All the tested attacks are effective against the targets, except for OCPP.Core 1.6 and 2.0 that are not vulnerable to I1. The exploited targets cover open-source (e.g., SteVe, OpenE-Mob) and closed-source (e.g., Anon) OCPP stacks, the OpenEVSE EV charger, the most recent OCPP versions (1.6, 2.0, 2.0.1), and all OCPP SPs (SP1, SP2, and SP3). These results demonstrate the need and effectiveness of EmuOCPP and that OCPP needs further security and privacy studies. Next, we describe the result for each attack target.

Mobilty House. We evaluated the eight attacks against our OCPP implementation, built on top of the Mobility House library (see Section 5.3), using dedicated EmuOCPP emulated networks. All attacks were successfully executed, regardless of the OCPP version. For each attack, we developed a corresponding OCPP implementation tailored to its scope with the help of EmuOCPP.

The first tested CSMS implementation allowed duplicate sessions, keeping legitimate and fake CS connected. Thus, I1 was successful, resulting in both the legitimate and fake CS being connected. The second implementation accepted the latest connection as legitimate and terminated the old one, allowing I1 to succeed. The third implementation preserved the first connection as legitimate and rejected any new connection attempts with the same ID, effectively preventing I1.

However, I2 was successful against all CSMS implementations, including the third one that rejects duplicate IDs. By waiting for the legitimate CS to reboot, the attacker could exploit the temporary disconnection window to impersonate the CS and establish a connection before the actual device could reconnect, effectively taking over the session and rendering the legitimate CS inoperative.

SteVe. SteVe is vulnerable to all tested attacks as it implements no SP. While analyzing I1, we found that SteVe provides two options for handling duplicate BootNotification messages. The first allows the server to maintain old and new connections, leading to I1. The second (default) ensures only one active connection per CS ID by terminating the old session when a new one is established, which is still vulnerable to I1. I2 also succeeded against these two configuration options, and the outcome was similar to I1. We could not test M2–M4 because there is no SP. M1 and E1 were both successful since no SP was supported. D1 was successful, as no rate limitation mitigation is in place.

Open E-Mob. All attacks are effective as Open E-Mob does not support SPs (like Steve). When investigating the outcome of I1, we found that Open E-Mobiltiy considers the last authenticated CS with a valid ID legitimate, and this logic enables both I1 and I2. We could not test M2–M4 because there is no SP. M1 and E1 were both successful since no SP was supported. D1 was successful, as no rate limitation mitigation is in place.

OpenEVSE. The OpenEVSE charger does *not* support any SP. However, it introduces a custom CS authentication mechanism. You can authenticate with a username and a password for the CS with the CSMS, where either one can

Table 4: Evaluation results of eight attacks against nine OCPP implementations. ✓ means that an attack is successful and ✗ that it is not successful. NA: Not Applicable due to lack of SP.

Target	Ver	SP	M1	M2	M3	M4	E1	D1	I1	I2
Mobility House	1.6	All	✓	✓	✓	✓	✓	✓	✓	✓
Mobility House	2.0	All	✓	✓	✓	✓	✓	✓	✓	✓
Mobility House	2.0.1	All	✓	✓	✓	✓	✓	✓	✓	✓
SteVe	1.6	None	✓	NA	NA	NA	✓	✓	✓	✓
Open E-Mob	1.6	None	✓	NA	NA	NA	✓	✓	✓	✓
OpenEVSE	1.6	None	✓	NA	NA	NA	✓	NA	✓	✓
OCPP.Core	1.6	None	✓	NA	NA	NA	✓	✓	✗	✓
OCPP.Core	2.0	None	✓	NA	NA	NA	✓	✓	✗	✓
Prod Network	1.6	SP2	NA	NA	NA	NA	NA	✓	✓	✓

be anything and is not restrained by the formats enforced by OCPP standards. Adapting to the custom authentication mechanism, we could still run E1, M1, I1, and I2, where we found the CSMS to consider the last CS to authenticate with the same ID to be the legitimate charger; the server terminates the old connection.

OCPP.Core. We successfully tested OCPP.Core 1.6 and 2.0 implementations. We connected our Mobility House CSs successfully with SP1. E1 and M1 are viable attacks for this implementation and were successfully tested. Because OCPP.Core does not support `InstallCertificate`, `SignCertificate`, and `CertificateSigned` messages. We could not test M2 and M3 as the relevant SPs are unsupported.

For I1, if a new connection attempts to use a serial number already employed, the CSMS rejects it. Thus, the OCPP.Core implementations are not vulnerable to classical I1. I2, on the other hand, worked against OCPP.core and effectively took over the connection of the targeted CS.

Prod Network A major company permitted us to run the I1 and D1 attacks against their productively deployed OCPP network. We confirmed the effectiveness of I1 and D1 on a target device in their quality assurance (QA) environment. We found the CSMS vulnerable to a D1 attack and did not implement any rate limiting or protection against DoS attacks. We validated that the production deployment suffered from the same limitation.

Furthermore, while testing I1, we discovered that the CSMS handles duplicate CS IDs by accepting the new connection as legitimate after successful authentication with valid credentials. The CSMS does not terminate the old connection at the TCP layer with the victim CS. This results in a bug where the victim CS continues to send heartbeat messages to the CSMS and is not terminated. In other words, the attacked CS remains unaware it has been disconnected.

7.3 Performance Results

Table 4 showcases that EmuOCPP satisfies in practice the six requirements we set in Secion 4.2. It is capable of testing the newest OCPP versions (**R1**) and all OCCP SPs (**R2**). It emulates OCPP networks with heterogeneous hosts (**R3**). It facilitates discovering new and old security and privacy issues (**R4**). It scales to complex topologies with many hosts (**R5**). It is reproducible, open-source, and low-cost (**R6**).

Table 5: RAM usage per number of devices.

CS	CSMS	RAM (MB)
1	1	210
10	1	590
100	1	4195
150	2	6240

We stress-tested EmuOCPP to evaluate its performance and scalability using VMware 17 Player on a machine running Ubuntu 24.04.1 LTS with 10 GB DDR5 RAM, 8 CPU cores, and 40 GB SSD storage. Our stress-test script is available at https://github.com/vfg27/EmuOCPP. It includes a configurable parameter to set the number of emulated CS. Each CS is configured dynamically with random but valid security profiles and OCPP versions.

During the stress tests, the network topology adapts dynamically based on the number of deployed CS. For instance, when only two CS are deployed, the network follows the topology illustrated in Figure 11. As the number of CS increases, the topology scales accordingly. When deploying 20 charging stations, the expanded topology is shown in Figure 12.

A key observation is that since CS passwords, certificates, and serial numbers are generated during execution, initial startup time increases as the CS number grows. On average, launching each CS takes between 1 to 3 seconds. We monitored memory consumption using the `free -m` command, reporting RAM usage in megabytes (MB). The results are in Tables 5 and 5.

Table 5 demonstrates the efficient scalability of EmuOCPP under increasing load. Even with 150 CSs and 2 CSMSs, the total memory usage remains within 6.2 GB, showing

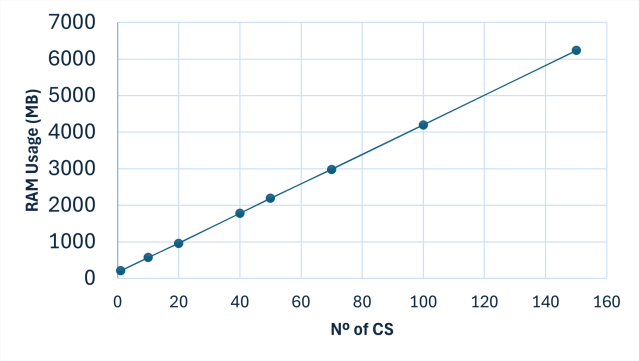

Figure 5: RAM usage with the increasing number of CSs.

that the emulator can support large-scale deployments on standard hardware. The near-linear RAM consumption suggests that each CS operates with minimal overhead (40 MB per CS) while the core emulation framework remains lightweight. Additionally, system initialization times remain manageable, with CS boot-up times averaging 1-3 seconds, making EmuOCPP practical for high-density OCPP network simulations without excessive resource demands.

8 Countermeasures

We discuss how to fix or mitigate our eight attacks and their root causes summarized in the last column of Table 2.

M1, E1. To defend against M1 and E1, implementers must use either SP2 or SP3. Furthermore, they can deploy protections such as Secure Neighbor Discovery (SEND) for IPv6 [11] and Dynamic ARP Inspection (DAI) [17] for IPv4 to prevent link-layer MitM attacks.

M2, M3. M2 can be mitigated by mandating the CS to use `AdditionalRootCertificateCheck` with trusted certificates. The latter provides certificate pinning and would require the attacker to leak the CSMS certificate. Moreover, OCPP should ban the usage of *self-signed* certificates, adopt a trusted CA architecture like the web PKI, and use certificate pinning. Additional network-level defenses, such as SEND, provide defense in depth.

M4. To fix M4, the OCPP standard should define how to handle SP downgrades and related Connection Profile configurations. OCPP 2.0 and 2.0.1 must ensure that once a security profile update occurs for a CS, all network connection profiles are upgraded to the same SP value, and legacy profiles are erased. Properly testing the network profile rollback mechanism after an update is recommended to ensure OCPP compliance and protection against this attack.

D1. CSMSs must be protected against DoS and DDoS attacks, as they operate as web applications and face threats similar to traditional web servers. Failure to implement appropriate defenses can result in multiple CPs losing connection to the CSMS and unable to authenticate EV users properly. To mitigate D1, CSMS deployments should apply standard web security measures, including connection rate limiting [10], SYN flood protection [10], IP reputation filtering [48], connection timeouts [10], and early-stage traffic filtering using Web Application Firewalls (WAFs) [18, 35]. Load balancing and resource scaling [10] can also reduce the impact of volumetric attacks targeting the CSMS.

I1. To mitigate the boot notification concurrent ID undefined behavior, we recommend enforcing a whitelist of valid CS IDs on the CSMS to prevent unauthorized or duplicate CS connections. Additionally, CSMS implementations should allow only one active connection per CS ID. Furthermore, deployments should use SP2 or SP3 when feasible, and CPs and CSMSs must be hardened against implementation vulnerabilities, such as those listed in the OWASP Top 10 [44], which could be leveraged to bypass SP2 or SP3.

I2. It is challenging to fix I2 as the attacker can use different techniques to check if a CS is online, including traffic analysis on encrypted data and side-channel attacks. As a mitigation, we recommend that a CS not communicate its status to unauthenticated network hosts. In addition, CSMSs should monitor for anomalous reconnect patterns, such as serial ID reuse from unexpected network origins.

9 Related work

MiniV2G is an open-source emulator combining *Mininet* and *RiseV2G* to simulate *Vehicle-to-Grid (V2G)* communications. It enables security research, including attack simulations like *message modification* and *denial-of-service* [12]. While it provides a virtualized testing environment, it lacks *physical-layer* simulations, such as real-time voltage analysis. Despite this, MiniV2G is a valuable tool for V2G security research.

The "MiniV2G" emulator by Attanasio [13] and our work share a common approach to building a testing framework on Mininet [38]. In contrast, we used IPMininet [50] for IPV6 support. While "MiniV2G" focuses on the Vehicle-to-Grid (V2G) end to explore security vulnerabilities in the communication between the EV and the charging point [13], our research emphasizes the Open Charging Point Protocol (OCPP), simulating various security challenges such as handling Boot Notifications and potential Denial of Service (DoS) attacks. Both projects provide a lightweight testing framework and can be integrated.

Alcaraz and Lopez analyzed the protocol's security threats and challenges, identifying vulnerabilities that could compromise the system [2]. Garofalaki et al. conducted a comprehensive survey on the security issues and challenges of OCPP [30]. Johnson and Smith investigated how specific attacks on OCPP 1.6 could lead to privacy breaches, particularly concerning user identity and charging habits [36]. The Swiss Federal Institute of Technology's report also sheds light on privacy

vulnerabilities within the protocol, suggesting measures to mitigate potential data leaks [49].

Dalamagkas et al. proposed a federated learning approach to detect cyberattacks on OCPP 1.6, aiming to enhance privacy preservation in intrusion detection systems [22]. Sarieddine et al. reveal significant security weaknesses in OCPP back-ends through a detailed analysis and simulation study. The authors designed a test bed requiring multiple virtual machines to validate their findings and extended their investigation by implementing proof-of-concept attacks on real-world OCPP systems [47].

In contrast to existing OCPP testing frameworks, EmuOCPP is lightweight, capable of running hundreds to thousands of CPs and CSMS instances on a commodity laptop, as demonstrated in Section 7.3. Conversely, Sarieddine et al. [47] managed to run only a few dozen CPs, relying on multiple virtual machines, each representing a CP or CSMS. Furthermore, EmuOCPP offers a configurable testbed supporting all OCPP versions, including the latest (2.0 and 2.0.1, whereas only 2 out of 6 surveyed open-source implementations provide partial support for 2.0, and none support 2.0.1 [21, 28, 46, 51]. In addition, most existing frameworks are limited to CSMS or CP roles, while EmuOCPP enables concurrent testing of both ends in a single testbed.

Moreover, EmuOCPP allows running various OCPP implementations, such as those in [19, 46, 51]. Related work [47] might also be able to use their setup to support various setups, but they don't allow for the same performance as EmuOCPP. Future work may extend EmuOCPP to support OCPP with V2G implementations by integrating, for example, some parts of MiniV2G [12].

Lastly, OCPP penetration testing tools, including the OCPPStorm fuzzer [21], and automated attack generation tools based on model checking [39] can be integrated with EmuOCPP to test potential vulnerabilities and attacks in an emulator running actual OCPP implementations. Hence, our tool is complementary to effective security testing methodologies.

10 Conclusion

The paper presents *EmuOCPP*, a new OCPP security and privacy testing framework. EmuOCPP covers all major OCPP versions (1.6, 2.0, and 2.0.1) and all OCPP SPs (SP1–SP3). It emulates an OCPP network using containers and reproduces complex topologies running hundreds of hosts and different OCPP configurations and implementations. It is configurable via a YAML file to support diverse testing scenarios. It allows the creation of multiple attacker nodes with arbitrary capabilities, like eavesdroppers, spoofers, and MitM. EmuOCPP is low-cost and easy to reproduce. We implement the tool using open-source software, including IPMininet and the THC toolkit.

Using EmuOCPP, we uncover five attacks affecting OCPP. The attacks include MitM, impersonation, DoS, and eavesdropping. They cover the major OCPP versions and all SPs. They exploit novel design issues, including undefined behaviors with CS serial IDs and SP downgrade or upgrade logic. We confirm the effectiveness of the attacks by testing them on nine OCPP targets, including open and closed source implementations, some of which were running on EmuOCPP. Specifically, we test Mobility House clients and servers, a SteVe server, an Open E-Mobility server, an OpenEVSE charger client, OCPP.Core servers, clients, and servers in a production network run by an Anonymous company. To address the risks posed by our findings, we discuss effective mitigations. We responsibly disclosed our findings to the OCPP consortium. Moreover, we empirically test the performance of EmuOCPP by stress testing it and confirm that it can run 150 hosts on a mid-range Linux laptop.

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 for them. Moreover, it has been supported by the French National Research Agency under the France 2030 label (NF-HiSec ANR-22-PEFT-0009). This work has also been supported by the Apricot/ENCOPIA ANR MESRI-BMBF project (ANR-20-CYAL-0001).

References

[1] What are good latency & ping speeds? https://www.pingplotter.com/wisdom/article/is-my-connection-good/, n.d. Accessed: 2025-02-21.

[2] C. Alcaraz, J. López, and S. Wolthusen. OCPP Protocol: Security Threats and Challenges. *International Journal of Critical Infrastructure Protection*, 19:47–58, 2017.

[3] Cristina Alcaraz, Jesús Cumplido, and Alicia Triviño. OCPP in the spotlight: threats and countermeasures for electric vehicle charging infrastructures 4.0. *International Journal of Information Security*, 22(5):1395–1421, 2023.

[4] Cristina Alcaraz, Javier Lopez, and Stephen Wolthusen. OCPP Protocol: Security Threats and Challenges. *IEEE Transactions on Smart Grid*, PP(99):1–1, 2017.

[5] Open Charge Alliance. Open Charge Point Protocol v1.6. https://openchargealliance.org/wp-content/uploads/2024/01/02.-Test-Procedure-T

est-Plans_v1.1.4.pdf, 2015. Accessed: 2025-02-20.

[6] Open Charge Alliance. Open Charge Point Protocol v2.0.1. https://openchargealliance.org/wp-content/uploads/2023/12/02.Test-Procedure-Test-Plans_v2.0.1_v11.pdf, 2020. Accessed: 2025-02-20.

[7] Open Charge Alliance. OCPP 2.0.1 Test Procedure & Test Plans. https://openchargealliance.org/wp-content/uploads/2023/12/02.Test-Procedure-Test-Plans_v2.0.1_v11.pdf, 2023. Accessed: 2025-02-20.

[8] Open Charge Alliance. OCPP 1.6 Test Procedure & Test Plans. https://openchargealliance.org/wp-content/uploads/2024/09/02.-Test-Procedure-Test-Plans_v1.3.1.pdf, 2024. Accessed: 2025-02-20.

[9] Open Charge Alliance. Open Charge Alliance (OCA) website, 2025. https://openchargealliance.org/.

[10] Amazon Web Services. Aws best practices for ddos resiliency. https://docs.aws.amazon.com/whitepapers/latest/aws-best-practices-ddos-resiliency/, 2024. Accessed: 2025-04-29.

[11] Jari Arkko, James Kempf, B. Zill, and P. Nikander. SEcure Neighbor Discovery (SEND), March 2005.

[12] Luca Attanasio. MiniV2G: An Electric Vehicle Charging Emulator. https://arxiv.org/abs/2101.11720, 2021. arxiv.

[13] Luca Attanasio, Mauro Conti, Denis Donadel, and Federico Turrin. MiniV2G: An Electric Vehicle Charging Emulator. In *Proceedings of the 7th ACM on Cyber-Physical System Security Workshop*, pages 65–73, 2021.

[14] Thomas E. Carroll, Michael J. Mylrea, and Trevor C. Moore. Motivation and Design of the OCPP Security Service. Technical report, Pacific Northwest National Laboratory, 2024.

[15] Chargelab. What is OCPP? https://chargelab.co/industry-advocacy/ocpp, 2024. ChargeLab.

[16] ChargeTimeEU. Java-OCA-OCPP. https://github.com/ChargeTimeEU/Java-OCA-OCPP. Accessed: 2025-02-14.

[17] Cisco Systems. Dynamic arp inspection configuration guide. https://www.cisco.com/c/en/us/td/docs/switches/lan/catalyst6500/ios/12-2SX/layer2/configuration/guide/l2_guide/dynarp.html, 2023. Accessed: 2025-04-29.

[18] Cloudflare. What is a web application firewall (waf)? https://www.cloudflare.com/learning/ddos/glossary/web-application-firewall-waf/, 2024. Accessed: 2025-04-29.

[19] Dallmann Consulting. OCPP.Core, 2024. https://github.com/dallmann-consulting/OCPP.Core.

[20] Dallmann Consulting. Ocpp.core: Ocpp server and management ui written in .net-core, 2025. Accessed: 2025-02-20.

[21] Gaetano Coppoletta, Kaur Amanjot, Rigel Gjomemo, and V. Venkatakrishnan. OCPPStorm: A Comprehensive Fuzzing Tool for OCPP Implementations. In *2024 VehicleSec (Co-located with NDSS)*, pages 1–12, 2024.

[22] Christos Dalamagkas, Panagiotis Radoglou-Grammatikis, Pavlos Bouzinis, Ioannis Papadopoulos, Thomas Lagkas, Vasileios Argyriou, Sotirios Goudos, Dimitrios Margounakis, Eleftherios Fountoukidis, and Panagiotis Sarigiannidis. Federated Detection of Open Charge Point Protocol 1.6 Cyberattacks. *arXiv preprint arXiv:2502.01569*, 2025.

[23] data.gouv.fr. Interparking. https://www.data.gouv.fr/fr/datasets/interparking/, 2023. data.gouv.fr.

[24] Developer. OpenEVSE WiFi for ESP32. https://github.com/OpenEVSE/ESP32_WiFi_V4.x. Accessed: 2025-02-14.

[25] ABB Developer. ABB Terra AC Charger OCPP 1.6 Implementation Overview. https://library.abb.com. Accessed: 2025-02-14.

[26] OCPP Developer. What's MessageTypeId in OCPP request payload. https://stackoverflow.com/questions/72362226/whats-messagetypeid-in-ocpp-request-payload. Accessed: 2025-02-14.

[27] Danny Dolev and Andrew C. Yao. On the Security of Public Key Protocols. *IEEE Transactions on Information Theory*, 29(2):198–208, 1983.

[28] SAP e Mobility Simulator. SAP e-Mobility Charging Simulator, n.d. https://github.com/SAP/e-mobility-charging-stations-simulator.

[29] David Elmo II. The Open Charge Point Protocol (OCPP) Version 1.6 Cyber Range: A Training and Testing Platform. Master's thesis, Wright State University, 2023.

[30] 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). *Computer Networks*, 203:108614, 2022.

[31] 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). *arXiv preprint arXiv:2207.01950*, 2022.

[32] L. Gebauer, Henning Trsek, and G. Lukas. Evil SteVe: An Approach to Simplify Penetration Testing of OCPP Charge Points. In *2022 IEEE 27th International Conference on Emerging Technologies and Factory Automation (ETFA)*, pages 1–4, 2022.

[33] Nikhil Handigol, Brandon Heller, Vimalkumar Jeyakumar, Bob Lantz, and Nick McKeown. Reproducible network experiments using container-based emulation. In *Proceedings of the 8th international conference on Emerging networking experiments and technologies*, pages 253–264, 2012.

[34] The Mobility House. Mobility House OCPP Python Library, 2025. https://github.com/mobilityhouse/ocpp.

[35] Indusface. Top 10 cybersecurity threats wafs prevent. https://www.indusface.com/blog/top-cyber-threats-waf-prevents/, 2024. Accessed: 2025-04-29.

[36] Michael Johnson and Emily Smith. Disrupting EV Charging Sessions and Gaining Remote Code Execution with DoS, MITM, and Code Injection Exploits using OCPP 1.6. Technical report, Idaho National Laboratory, 2023.

[37] Loren Kohnfelder and Praerit Garg. The threats to our products. Microsoft Interface, 1999. https://www.microsoft.com/en-us/securityengineering/sdl/practices#threatmodeling.

[38] Bob Lantz, Brandon Heller, and Nick 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*, pages 1–6. ACM, 2010.

[39] Will Marrero, Edmund Clarke, and Somesh Jha. A model checker for authentication protocols. In *Proceedings of the DIMACS workshop on design and formal verification of security protocols*, volume 9, 1997.

[40] John Meier, Michael Dunner, Srinath Vasireddy, Shanmugam Manoharan, and Anandha Gopalan. Improving web application security: Threats and countermeasures. Microsoft Patterns & Practices, 2003. https://docs.microsoft.com/en-us/previous-versions/msp-n-p/ff648641(v=pandp.10).

[41] mitmproxy. mitmproxy Documentation. https://docs.mitmproxy.org/stable/. Accessed: 2025-02-27.

[42] OCA. Open Charge Point Protocol Specification. https://openchargealliance.org/protocols/open-charge-point-protocol/, 2024. OCPP.

[43] OpenEVSE. OpenEVSE Electric Vehicle Charging Solutions, n.d. https://www.openevse.com/.

[44] OWASP Foundation. Owasp top 10: The ten most critical web application security risks. https://owasp.org/Top10/, 2023. Accessed: 2025-04-29.

[45] Python. tkinter — Python interface to Tcl/Tk, 2025. https://docs.python.org/3/library/tkinter.html.

[46] SAP. SAP e-Mobility Smart Charging. https://github.com/SAP/emobility-smart-charging. Accessed: 2025-02-14.

[47] Khaled Sarieddine, Mohammad Ali Sayed, Sadegh Torabi, Ribal Atallah, Danial Jafarigiv, Chadi Assi, and Mourad Debbabi. Uncovering Covert Attacks on EV Charging Infrastructure: How OCPP Backend Vulnerabilities Could Compromise Your System. In *Proceedings of the 2024 ACM Asia Conference on Computer and Communications Security (AsiaCCS)*, pages 1–14, 2024.

[48] Fidelis Security. Guide to safeguard your network from ddos attacks. https://fidelissecurity.com/threatgeek/network-security/prevent-ddos-attacks-on-network/, 2024. Accessed: 2025-04-29.

[49] National Test Center Switzerland. Security Analysis of the Swiss Charging Infrastructure. Technical report, National Test Center Switzerland, 2023.

[50] Olivier Tilmans. IPMininet: Mininet extension to support complex IP network emulation. https://github.com/cnp3/ipmininet, 2019. Accessed: 2025-02-15.

[51] RWTH Aachen University. SteVe - Open Charge Point Protocol (OCPP) Server, n.d. https://github.com/RWTH-i5-IDSG/steve.

[52] van Hauser and THC. THC-IPv6: Attack Toolkit for the IPv6 Protocol. https://github.com/vanhauser-thc/thc-ipv6, 2024. Accessed: February 2025.

A OCPP Implementations

A.1 OpenEVSE

OpenEVSE [43] and its derivative, *EmonEVSE* [24], are open-source platforms offering customizable charging solutions.

EmonEVSE is a physical charging station designed for European installations, providing up to 22kW charging power and compliance with IEC 60947-5 and BS EN 61851-1:2011 standards. Both platforms currently support OCPP 1.6 (beta); however, they lack implementations for security profiles and do not support versions beyond OCPP 1.6.

A.2 SteVe

SteVe [51], developed at RWTH Aachen University, is an OCPP server supporting versions 1.2 to 1.6. While it provides basic functionality like RFID authentication, it lacks support for OCPP 2.0 and security profiles.

A.3 SAP EV Simulator

SAP EV Simulator [28] is a Node.js tool designed to simulate and scale a set of charging stations based on the OCPP-J protocol. While it facilitates testing and development, it currently supports only OCPP 1.6 and lacks support for the latest OCPP versions and the OCPP security profiles.

A.4 Open E-Mobility

Open E-Mobility [46] is an open-source project that provides a comprehensive solution for managing electric vehicle charging infrastructure. It includes charge point management, user administration, and billing integration features. The platform supports OCPP 1.6, enabling communication between charging stations and the central system. However, like other implementations, it lacks support for newer OCPP versions and advanced security profiles, which are essential for ensuring secure and reliable operations in evolving EV charging ecosystems.

A.5 Java-OCA-OCPP

Java-OCA-OCPP, [16] maintained by ChargeTimeEU, is an open-source client and server library for OCPP. This Java-based library supports OCPP versions 1.6 and 2.0, providing a framework for developers to implement OCPP-compliant charging stations and central systems. While it facilitates the development of OCPP solutions, users must ensure the implementation of necessary security measures, as the library's primary focus is on protocol compliance rather than security features.

A.6 Mobility House

Mobility House [34] offers an open-source Python library that implements the JSON version of the Open Charge Point Protocol (OCPP), supporting versions 1.6, 2.0, and 2.0.1. This library provides developers with the foundational tools to construct charging stations (charge points) and central management systems (CSMS). While it facilitates the development of OCPP-compliant applications, developers must build upon its framework to create complete solutions tailored to specific use cases.

A.7 OCPP.Core

OCPP.Core [19] is an OCPP server written in .NET 8. It provides a CSMS capable of managing charge points and tokens (RFID-Token) through a Web UI. The system supports WebSocket-based communication and implements essential OCPP functionalities, including charging point registration, message exchange, and basic authentication mechanisms. It currently supports OCPP1.6J and 2.0(JSON/REST). OCPP.Core is currently used with 6 KEBA P30c/x charge points operating in load management and OCPP1.6J.

B Figures

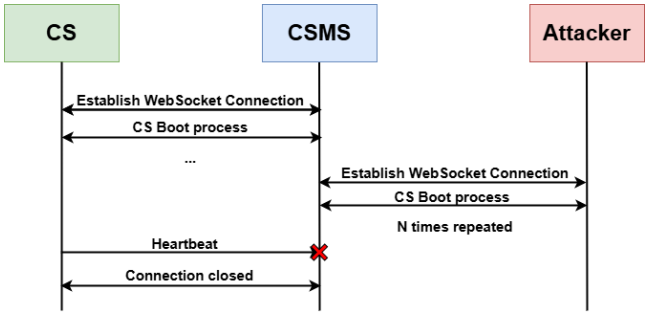

Figure 6: CSMS DoS Attack (D1).

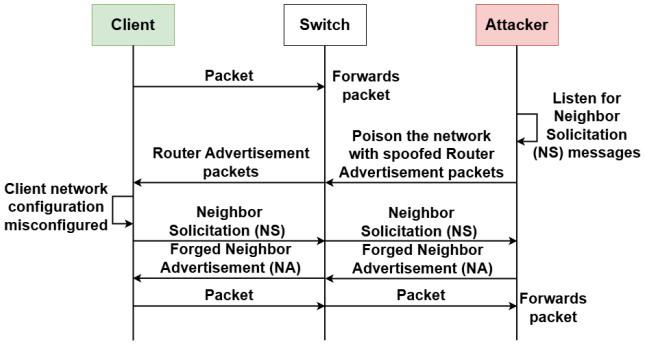

Figure 7: IPv6 link-layer NDP Spoofing.

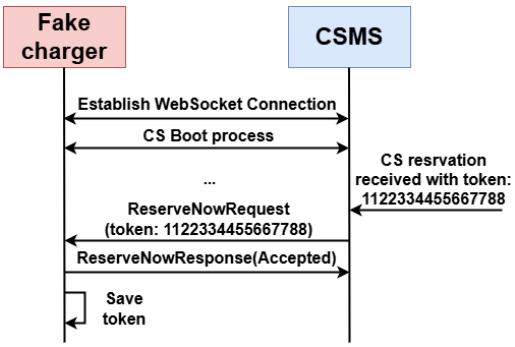

Figure 8: I1 user authentication token theft.

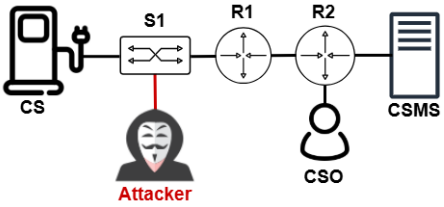

Figure 9: Topology for E1, M1, M2, M3 and M4.

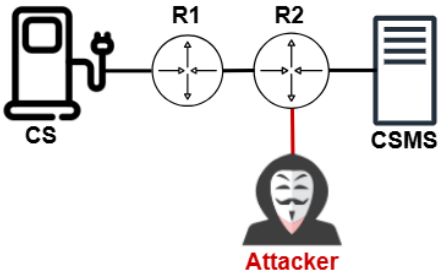

Figure 10: Topology for I1 and D1.

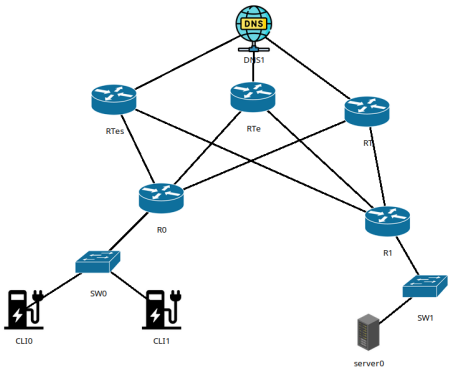

Figure 11: Topology with two CSs.

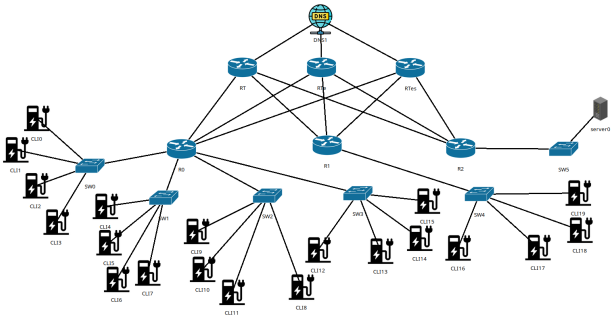

Figure 12: Topology with twenty CSs.

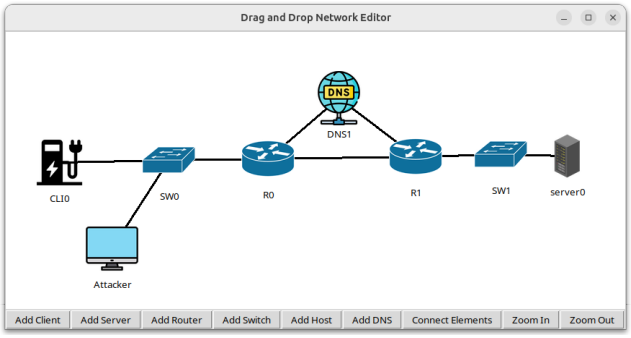

Figure 13: EmuOCPP's graphical user interface with an example of an emulated topology.

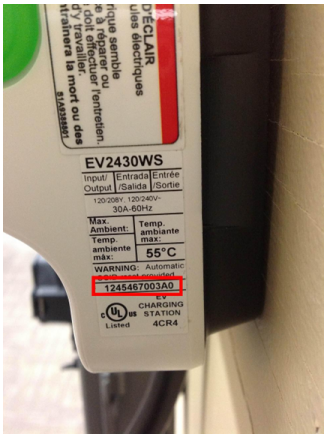

Figure 14: Public information on a production CS, as serial number and manufacturer.