OCPPStorm: A Comprehensive Fuzzing Tool for OCPP Implementations

Gaetano Coppoletta
University of Illinois Chicago
tano.coppoletta@gmail.com

Amanjot Kaur
and Nima Valizadeh
and Omer Rana
Cardiff University
KaurA7@cardiff.ac.uk
ValizadehN@cardiff.ac.uk
RanaOF@cardiff.ac.uk

Rigel Gjomemo
and V.N. Venkatakrishnan
Discovery Partners Institute
University of Illinois System
rgjome1@uic.edu
venkat@uic.edu

***Abstract*—In the last decade, electric vehicles (EVs) have moved from a niche of the transportation sector to its most innovative, dynamic, and growing sector. The associated EV charging infrastructure is closely following behind. One of the main components of such infrastructure is the Open Charge Point Protocol (OCPP), which defines the messages exchanged between charging stations and central management systems owned by charging companies. This paper presents OCPPStorm, a tool for testing the security of OCPP implementations. OCPPStorm is designed as a black box testing tool, in order to be able to deal with different implementations, independently of their deployment peculiarities, platforms, or languages used. In particular, OCPPStorm applies fuzzing techniques to the OCPP messages to identify errors in the message management and find vulnerabilities among those errors. It's efficacy is demonstrated through extensive testing on two open-source OCPP systems, revealing its proficiency in uncovering critical security flaws, among which 5 confirmed CVEs and 7 under review. OCPPSTorm's goal is to bolster the methodological approach to OCPP security testing, thereby reinforcing the reliability and safety of the EV charging ecosystem.**

***Index Terms*—Cybersecurity, OCPP, Fuzzer, Electric vehicle charging, cyber-resilience**

I. Introduction

The swift embrace of EVs has initiated a fresh wave of research and development, propelling the transportation sector into an era of unparalleled innovation and progress. This shift towards EVs is predominantly fueled by a growing environmental consciousness and a collective endeavor to decrease dependence on traditional fossil fuels. The EV market is undergoing consistent expansion, with numerous companies foreseeing that by the year 2040, EVs will constitute more than 50% of the overall automotive ecosystem [1][2] [3]. However, despite the widespread adoption of EVs, one of the main barriers hindering a broader acceptance is the inadequacy of EV charging infrastructure. Such infrastructure is composed of several components and standards including Electric Vehicle Charging Stations (EVCSs), Central Management Systems (CMS), and e-Mobility Service Providers (e-MSPs) to ensure the optimal functioning of EVs.

The Open Charge Point Protocol (OCPP) is one of the components of the EV charging infrastructure. As an open communication protocol, OCPP facilitates the interoperability between EVCSs and CMS. OCPP has the potential to play a pivotal role in the EV charging ecosystem [4]. It has emerged as a widely adopted protocol, being utilized as the de facto standard in 148 countries across all six continents [5]. The protocol is endorsed by the Open Charge Alliance (OCA), featuring over 220 member companies actively participating in the electric mobility sector [6]. OCPP's widespread implementation is evidenced by its adoption by major charging companies worldwide, detailed in Table I. In practice, OCPP facilitates remote management of charging points from a central management system, allowing centralized operation and monitoring, enabling tasks such as remote diagnostics, firmware updates,starting and stopping of charging sessions, surveillance of power usage, etc.

OCPP has evolved to address growing security concerns, with different versions offering varying levels of security. In OCPP 1.5 and earlier, basic security measures were implemented, establishing a foundational level for secure communications but lacking advanced protections. With the introduction of OCPP 1.6, currently the most used protocol, optional enhanced security features were marked, encompassing encrypted communications and improved authentication mechanisms. The progression continued in OCPP 2.0 and beyond, where advanced security features became standard, incorporating mutual authentication and emphasizing the significance of ongoing security updates.

Recent research into the security of the OCPP includes several works [9] [10] [13]. In general, these works identify scenarios where OCPP protocol may be bypassed to initiate attacks against CMSes or charge points. In this paper, instead, we focus on designing a systematic approach to test the security of *generic OCPP implementations*. The main challenge in designing such approach relies in the fact that

Symposium on Vehicles Security and Privacy (VehicleSec) 2024
26 February 2024, San Diego, CA, USA
ISBN 979-8-9894372-7-6
https://dx.doi.org/10.14722/vehiclesec.2024.23069
www.ndss-symposium.org

OCPP implementations may use different types of technologies, programming languages, and deployment platforms. To address this challenge, we design OCPPStorm, a black box testing tool to test such generic implementations. The key contributions of the paper are as follows:
(i) We propose black box testing as a robust testing methodology that enables the evaluation of multiple security implementations. This approach allows us to assess the functionality of a system without delving into its internal code structure, logic, or implementation details.
(ii) We utilize a range of fuzzing techniques employing various strategies to explore and exploit security vulnerabilities deeply embedded in protocol interactions. This research sheds light on the effectiveness of fuzzing as a proactive security measure, offering valuable insights into identifying and addressing potential weaknesses within the context of protocol interactions.
(iii) Using OCPPStorm, we identify several vulnerabilities in two open source OCPP implementations, leading to 5 confirmed CVEs and 7 more under review at MITRE at the time of this submission.
(iv) The OCPPStorm code base will be released as open source.

The rest of this paper is organized as follows: Section II provides some background and an overview of OCPPStorm. Section III describes OCPPStorm in detail. Section IV covers the implementation and Section V contains the evaluation of OCPPStorm. Section VI reviews related work. Finally, Section VII provides conclusions and outlines future work.

II. Background and Solution Overview

A. Background

OCPP is a protocol that defines the communications between charging stations and central management systems. In particular, OCPP defines a set of messages such as `BootNotification.req` sent by charging stations (or charging points) to central management systems and `BootNotification.conf` sent by central management systems to charging stations. Each message is composed of several fields (e.g., `chargePointVendor`, `messageId`) that convey specific types of information. The protocol specification describes the semantics of the messages and the types of responses associated with each request, as well as the order in which messages are sent. In addition, the protocol specifies the data types for each message field (e.g., `integer`, `string`) and constraints on the length and values of those fields.

A central management system (CMS) is typically responsible for the management of a set of charging stations that are physically distributed over a specific region. The central management system is responsible for storing data about users (e.g., user information, authentication credentials, payment details), charging sessions (e.g., start and end timestamps, power used to recharge an EV), charging stations (e.g., vendor, ids, etc). Usually, a central management system has a user interface, often implemented by a browser, which communicates with a database where all the data about users, stations, and sessions, are stored and updated. From a software architecture point of view, then, a CMS can be thought of as composed of two interconnected parts, a communication component, that implements the OCPP protocol, and a management component that deals with the business logic for authorizing sessions, user management, visual interface for the CMS employees, etc. The main goal of OCPPStorm is testing the security of any CMS whose inputs consist largely of OCPP messages.

Threat Model and Assumptions: We assume that an attacker has the ability to send messages to OCPP CMSes. This could happen in different ways. For instance, an attacker may obtain physical or remote access to a charging station and be able to craft arbitrary OCPP messages that are then encrypted and then sent to the CMS. Different ways to compromise charging points have been demonstrated by recent works [10], [14]. We assume that the attacker is able to authenticate and start a session. This is to be able to test the protocol implementations of the functionality that requires authentication.

Starting from these assumptions, we create OCPPStorm, a tool to test the security of different CMSes or OCPP implementations. In particular, we want to test how well the OCPP communications' component of these CMSes validate the input of incoming OCPP messages. The main challenges in doing so are listed below.

- *Diverse Implementations.* OCPP implementations can vary greatly depending on the manufacturer, version, or specific requirements of the deployment. The heterogeneity of the system complexities precludes the formulation of a universally applicable testing methodology.
- *Language and Platform Independence.* OCPP implementations can be found in a myriad of programming languages and platforms, adding an additional layer of complexity to the testing process.

Addressing these challenges requires a testing methodology that is independent of the platforms, technologies, and languages used to implement CMSes and OCPP communications.

B. Solution Overview

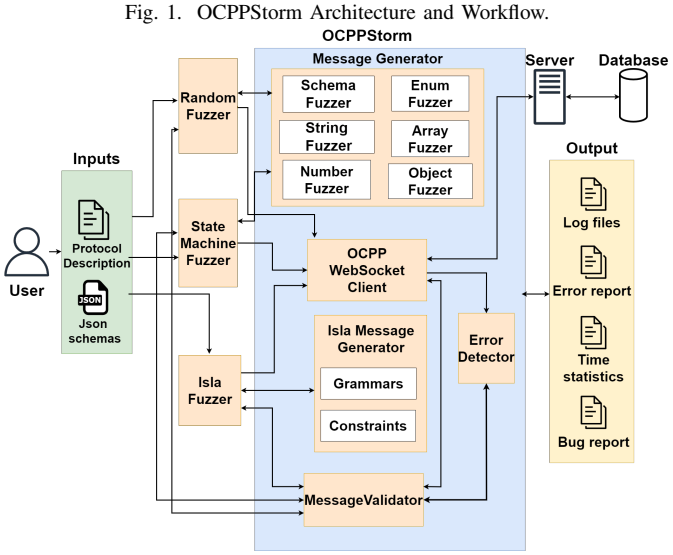
To be platform-independent and language agnostic, our OCPPStorm solution revolves around the principle of *black box testing*. Black box testing is a testing methodology in which the tester is unaware of the internal workings of the item being tested. Instead, the tester is only familiar with the inputs and the expected outcomes. In essence, by employing black box testing for OCPP, we can systematically evaluate various implementations regardless of their internal complexities or specific details. This method provides a comprehensive and robust security assessment that not only identifies vulnerabilities but also ensures the integrity and reliability of critical infrastructure systems.

A special methodology for black box testing is *black box fuzzing*. Fuzzing is a methodology for testing applications that relies on sending a large number of inputs that stress test an application's error handling and logic. The main challenge faced by fuzzers is that of *coverage*. The input sent to the application under test must exercise as much of the application's code as possible. Fuzzers deal with the coverage problem in different

TABLE I
MAJOR CHARGING COMPANIES USING OCPP

Company Name	Region	OCPP Usage	Additional Notes
ChargePoint	US	Yes	One of the largest EV charging networks in the US. Uses OCPP for interoperability.
EVBox	Europe	Yes	A leading EV charging solutions provider in Europe. Adopts OCPP for flexibility and compatibility.
Greenlots	Asia (Singapore-based)	Yes	Offers OCPP-compliant charging solutions, ensuring compatibility across different charging networks.
ABB	Global (Europe, US, Asia)	Yes	A major industrial equipment manufacturer that provides OCPP-compliant EV charging solutions.
Enel X	Europe	Yes	Provides OCPP-compliant charging solutions, allowing for easy integration with various management systems.
NewMotion	Europe	Yes	One of Europe's largest providers of EV charging services. Supports open standards like OCPP.

Fig. 1. OCPPStorm Architecture and Workflow.



ways. Some fuzzers like AFL receive branch information as feedback and mutate their inputs in order to exercise untested branches. Other fuzzers rely on knowledge of the code to solve constraints that exercise untested branches.

In our case, because of our choice to create a general OCPP testing tool as a black box fuzzer for multiple implementations, the only feedback that we receive from the application is in the OCPP messages themselves and eventual error messages. The lack of information about the server-side code execution, like branches and paths executed means that we cannot measure code coverage on the server code. Instead, we introduce the notion of *protocol coverage*. The main idea here is that in order to test the implementation of a specific component of the protocol, which may be located deep in the sequence of messages, we need to send inputs that satisfy the constraints and logic of the protocol up to reaching the component that we want to test. OCPPStorm does this via a *State Machine Fuzzer*, which is described later in this section.

OCPPStorm's architecture is depicted in Figure 1. The input to OCPPStorm is a set of json schema files representiing the messages and a protocol description or representation as a state machine. OCPPStorm integrates three core fuzzing modules—Random Fuzzer, State Machine Fuzzer, and Isla Fuzzer—interfacing with specialized components like the Message Generator and Isla Message Generator. All three components can generate both valid and invalid OCPP messages. Valid messages satisfy the constraints of the OCPP protocol on data types, field lengths, content, and order of messages. Invalid messages instead, violate one or more of those constraints. The random message generator does so in a random manner, the ISLA message generator uses the ISLA library [22], while the state machine fuzzer uses a representation of the OCPP protocol as a state machine to guide the message generation.

The OCPP WebSocket Client manages communication with the external server representing a CMS, and the Message Validator ensures compliance with OCPP standards. Any discrepancies or anomalies detected by these components are flagged by the Error Detector. This collaborative architecture ensures the effective crafting and dispatching of messages, with the goal of uncovering potential vulnerabilities in OCPP implementations. The effectiveness of OCPPStorm is fundamentally tied to the quality and relevance of its inputs, comprising the protocol description and JSON schemas.

We describe in more detail the different components in the next sections.

III. DESIGN

In this section, we delve into the functionalities of each component, highlighting their roles within OCPPStorm and their collective contribution to the overall fuzzing process.

A. Fuzzing Modules in OCPPStorm

OCPPStorm implements three different independent fuzzing mechanisms, 1) a *Random Fuzzer* that generates random messages, 2) a *State Machine Fuzzer* that generates messages compliant with the protocol up to a specific state and then starts generating random messages, and 3) a grammar-based fuzzer called *Isla Fuzzer* based on the ISLA library [22].

Random Fuzzer: The Random Fuzzer, integrated with the Message Generator, operates through iterative random message generation. Its process involves: (a) Random selection of a message type for each iteration, (b) Message generation

and subsequent validation using JSON schema, (c) Prediction of server response, based on the fuzzed message properties, and (d) Transmission of the message and analysis of server feedback by the Error Detector.

State Machine Fuzzer: The State Machine Fuzzer offers an advanced approach to increase the coverage of the fuzzer. This component employs user-defined sequences of valid OCPP messages to simulate paths in a state machine. OCPPStorm can then fuzz messages starting from the state reached in the state machine. For instance, a sequence of messages may include a message to authenticate a user and a message to start a charging session. After these messages are sent, OCPPStorm can start fuzzing the application from the state reached when the charging session is started. At this point, the Error Detector module evaluates the server response, ensuring the transition to the next state in the sequence. We opted for a representation of the state machine as a sequence of messages for usability purposes. In fact, our goal is to make it easier for software developers and testers to use OCPPStorm to test their implementations without having to create complex state machine representations of their implementations.

Isla Fuzzer: The Isla Fuzzer, deriving its functionality from the Isla library, presents two modes of operation:

a) Fuzzing With Constraints: This mode employs constraints-based message generation, targeting specific Json properties for fuzzing. This strategy involves selecting a set of predefined constraints for a particular message type during the fuzzing process. For example, in generating an `Authorize` message, the sole property, `idTag`, is subject to a maximum length constraint of 20 characters. In a constraints-based approach, the Isla library is instructed to create an `idTag` that deliberately exceeds this limit, challenging the protocol's established constraints.

b) Fuzzing Without Constraints: Focuses on generating messages without considering OCPP constraints, enhancing the randomness and coverage of fuzz testing. The integration of the Isla Fuzzer with other standard modules of OCPPStorm, such as the Error Detector and Message Validator, ensures a thorough and efficient fuzzing process.

B. Message Generators

OCPPStorm provides two message generators that follow two different strategies for the generation of the messages.

Message Generator: The Message Generator is a key element, distinguished from the Isla Message Generator, and is intricately tailored to enhance the fuzzing process for the OCPP protocol. Unlike the Isla library's general-purpose approach, our Message Generator is meticulously crafted to address the specific intricacies of the OCPP protocol.Using the Json schemas of the OCPP messages, this component is able to generate both valid and invalid OCPP messages to send to the CMSes. The valid messages are consistent with the various OCPP message types. The invalid messages are created by systematically creating random actions and payload structures. In particular, the randomization operates at the: 1) single data type level (by creating fields with different datatypes, fields with values or lengths outside the ranges specified by OCPP), 2) message level (by creating messages with missing or added fields or fields different from the ones that are expected).

The Message Generator is designed to work cohesively with other modules within OCPPStorm. This collaboration facilitates a synchronized operation of the fuzzing process, with its output feeding into components like the Random Fuzzer and the State Machine Fuzzer.

Isla Message Generator: The Isla Message Generator, operating in conjunction with the Isla Fuzzer, uses a combination of grammars and constraints for message creation, based on the ISLa library [22]. These grammars specify string production rules and constraints that the ISLa library uses to create strings in the language of the grammar. The ISLa message generator in OCPPStorm encodes grammars that based on the Json schemas create strings that represent OCPP messages.

C. Error Detector

Within the architecture of OCPPStorm, the Error Detector is a crucial component tasked with the analysis of server feedback under various test scenarios. It essentially acts as a bridge, aligning the theoretical expectations of the OCPP protocol with the actual server responses. The key functionalities of the Error Detector are outlined as follows:

Analyzing Server Responses: There are two kind of messages that can be received:

a) Valid OCPP Messages: When the server receives legitimate OCPP messages, the Error Detector evaluates whether the responses are appropriate and in line with the required operations. This ensures that the server's behavior remains consistent with the protocol standards.

b) Invalid OCPP Messages: In scenarios where fuzzed messages are sent to the server, the Error Detector assesses the server's reaction to these irregular inputs, identifying any deviations or anomalies that could indicate vulnerabilities or non-compliance with the OCPP protocol.

Predicting Error Responses: The Error Detector also possesses the capability to predict potential errors. This capability is built from the information in the OCPP specification documents, which specify the type of errors that can occur for each OCPP message [25]. This predictive analysis involves comparing the fuzzed message against its corresponding JSON schema and utilizing logical constructs to anticipate potential server errors. This approach is particularly effective in detecting violations of JSON schema constraints and provides insights into the server's adherence to protocol standards.

Gathering Error and Bug Statistics: A critical role of the Error Detector is to compile data on server errors and potential vulnerabilities. This information is crucial for a comprehensive assessment of the server's robustness and security. The collected data, which includes statistics on various error types and bugs, is documented for detailed analysis and future reference.

D. Message Validator

The *Message Validator* within OCPPStorm is critical for ensuring the accuracy and integrity of message exchanges.

It serves as a fundamental component, verifying messages against predefined standards and structures. Its primary responsibilities include:

Timestamp Verification: The validator scrutinizes the 'timestamp' property in messages, validating it against the RFC3339 standard. This ensures format and value consistency.

Schema Compliance: The validator confirms message compliance with corresponding JSON schemas. It checks the structure and values against the schema, marking messages as non-compliant in case of deviations.

Exception Handling: Capable of managing scenarios where messages deviate from expected structures, the validator identifies and reports discrepancies, crucial for handling fuzzed messages.

The *Message Validator* acts as a vital checkpoint within the OCPPStorm fuzzer, as depicted in **??**. It interfaces with multiple components, including the *Random Fuzzer*, *State Machine Fuzzer*, *Isla Fuzzer*, the *OCPP WebSocket Client*, and the *Error Detector*. This ensures that every OCPP message, whether generated or received, undergoes validation, upholding the communication process's integrity and adherence to protocol schemas.

E. OCPPStorm Output Report

OCPPStorm generates detailed output logs, each serving a specific function in documenting and analyzing the fuzzing process. These logs are instrumental in understanding the responses of the OCPP implementation to various test cases. Here is an overview of the output files:

Correct Messages Log: This file documents messages deemed correct by OCPPStorm, including valid OCPP messages and status notifications. It's vital for identifying instances where the server incorrectly rejects valid inputs.

Fuzzed Messages without Errors Log: Captures fuzzed messages that did not elicit errors from the server, suggesting potential gaps in input validation.

Non-Protocol-Compliant Responses Log: Details server responses that are technically valid but do not adhere to the expected OCPP JSON schema.

Statistics Log: Compiles general fuzzing metrics, such as total iterations, rates of correct and incorrect responses, and various detected error types.

Error Type Statistics Log: Tracks specific error types encountered during fuzzing, providing quantitative data on each error category.

Status Notification Statistics Log: Monitors transitions related to the *StatusNotification* message, offering insights into the server's state management.

Fuzzing Duration Log: Records start and end timestamps of the fuzzing process, reflecting the overall duration and efficiency of the tests.

Valid Requests Causing Errors Log: Lists valid requests that unexpectedly trigger server errors, indicating potential issues in server response handling.

Mismatched Error Responses Log: Contains instances where the server's error responses deviated from those anticipated by OCPPStorm, highlighting error handling discrepancies.

These logs are pivotal in evaluating both OCPPStorm's performance and the resilience of the OCPP implementations under test, providing granular insights into the system's behavior against diverse fuzzing inputs.

IV. Evaluation

OCPPStorm is developed in Python with approximately 1915 lines of code. To evaluate it, we searched for available CMS implementations over the Web. The most well-developed available CMS implementations that we were able to find are two open source packages: Steve[1] and OCPP.Core[2]. Steve is a Java-based open-source CMS with a total of 31,432 lines of code. OCPP.Core is a .NET-based server with a total of 21,684 lines of code. This CMS uses a mix of C#, JavaScript, and XML, showcasing the diverse language use within the .NET ecosystem. For our evaluation, we installed both applications on a local Mac M1 laptop, initializing their databases with fictitious users and charging stations to simulate an operational charging networks.

Results Overview. Our evaluation of OCPPStorm over these two CMSes assesses the tool's performance in error detection, security vulnerability identification, resource performance, and scalability. OCPPStorm was able to exercise all OCPP 1.6 messages that have as destination the central CMS. It found that both the CMS implementations deviate from the OCPP protocol significantly in terms of error handling and leniency towards constraints violations. OCPPStorm identified numerous bugs and vulnerabilities, with 6 confirmed CVEs and 6 suspicious behaviors that may result in attacks under specific situations. We describe our results in more detail in the rest of this section.

A. OCPPStorm Evaluation Metrics

In our evaluation of the OCPPStorm, we aimed to cover a broad spectrum of test scenarios, ranging from schema violations and exceeding length constraints to the processing of unexpected input types. To do this, we use the following metrics.

1) Total Iterations: The complete count of messages sent.

2) Correct Responses Rate: The ratio of responses that were accurate, aligning with expectations for both valid and invalid inputs. Specifically, we consider a response as correct in two cases: a) if the fuzzer sends an invalid message and we receive an error response, matching the type of error predicted by the fuzzer, b) if the fuzzer sends a valid message and we receive a valid response. Ideally, this value should be as close to 100% as possible. This rate therefore reflects the system's accuracy in responding correctly according to the expected outcome for each specific input. An example of correct message that falls into this category is:

[1]https://github.com/steve-community/steve

[2]https://github.com/dallmann-consulting/OCPP.Core

```
[2,"1","Authorize","idTag":"abcde12345"]
```

which will receive this response:

```
[3,"1",{"idTagInfo":{"expiryDate
    ":"2024-01-24T00:00:00-06:00","
    status":"Accepted"}}]
```

An example of incorrect message (*idTag* is not a string) that trigger the correct error and so falls into this ratio is:

```
[2,"1","Authorize","idTag": 123]
```

which will receive the predicted error (*TypeConstraintViolation*), here is a possible response:

```
[4,"1","TypeConstraintViolation","idTag
    longer than 20",{}]
```

3) Incorrect Response Rate: This metric is the opposite of the Correct Response Rate. It is the ratio of responses that were different from the expected ones. This is composed of the following subcategories:

a) Wrong Error Rate: The ratio of errors differing from those anticipated by the fuzzer, suggesting potential issues in error handling. An example of message that cause the wrong error is:

```
[2,"1","Authorize","idTag":123]
```

which should trigger a *TypeConstraintViolation* error. If we receive another type of error, this ratio becomes higher. An example of wrong error for this message is:

```
[4,"1","InternalError","",{}]
```

b) Valid Requests Causing Error Rate: The ratio of instances in which valid inputs erroneously triggered errors, indicating improper processing by the system. An example of message that falls into this category is:

```
[2, '15505', 'StatusNotification', {'
    connectorId': 6, 'errorCode': '
    GroundFailure', 'status': '
    Finishing'}]
```

The *StatusNotification* message comply with the constraints defined by its json schema and it should be accepted by the server. If it's not accepted and the server sends an error, this ratio becomes higher. An example of response is:

```
[4,"15505","InternalError","",{}]
```

c) Non-error-Causing Fuzz Rate: The fraction of invalid messages that did not cause an error but ideally should have, indicating possible validation gaps. An example of message that falls into this category is:

```
[2, '84', 'MeterValues', {'
    connectorId': 7}]
```

the json schema of the *MeterValues* message has two required properties: *connectorId* and *meterValue*. The message in the example does not contain the *meterValue* property and so is flagged as fuzzed and should trigger an error. If we do not receive an error from the server this ratio will be higher. An example of response is:

```
[3,"84",{}]
```

d) Responses Not Valid with Protocol: The frequency of responses that did not adhere to the OCPP protocol's constraints. In particular the response is validated against its json schema and if it does not comply with it, the message falls into this category. An example of message for this ratio is:

```
[2, '1', 'Authorize', {}]
```

This message is not valid because the *idTag* property is required and here is missing. The server sends this response:

```
[3,"1",{}]
```

which does not comply with the json schema for the *AutorizeResponse* message, that must contain at least the *idTagInfo* property.

4) Comprehensive Handling Rate: This ratio evaluates the system's overall ability to correctly process inputs, but with a broader scope than the Correct Responses Rate. It includes all well-formed messages that receive an appropriate response and all incorrect messages that successfully trigger any error response, regardless of the specific error type. This rate is indicative of the system's general resilience and effectiveness in handling a wide range of inputs, reflecting its capacity to appropriately respond under varied circumstances, without being restricted to the accuracy of error types. An example of message that falls into this category is:

```
[2,"1","Authorize","idTag":123]
```

the predicted error for this message is *TypeConstraintViolation* but here we are just looking for an error without checking if it's the predicted one. Examples of possible responses for this category are:

```
[4,"1","TypeContraintViolation","idTag
    longer than 20",{}]
[4,"1","InternalError","",{}]
[4,"1","GenericError","",{}]
```

Table II contains the results for the different metrics. With the exception of the first row, each row of the table corresponds to one metric and shows the results of OCPPStorm's components (random fuzzer, ISLA fuzzer, and State Machine fuzzer) for that metric against Steve and OCPP.Core. The first row corresponds to the number of iterations. We report different numbers of iterations for the different variants of OCPPStorm. This is due to the slow performance of the ISLa fuzzer with constraints (see Table IV for the performance results). In this fuzzer, the constraint solution step can significantly slow the message generation.

In Random Fuzzer against Steve, the evaluation shows a Correct Responses Rate of 22.44%, indicating limited alignment with expected outcomes. The Comprehensive Handling Rate stands at 57.31%, suggesting moderate effectiveness. However, a high Incorrect Response Rate of 77.56% points to notable issues in error handling and protocol compliance, underscoring areas for improvement in Steve's implementation.

TABLE II
EVALUATION METRICS FOR OCPPSTORM

Metric	**Random fuzzer against Steve**	**Random fuzzer against OCPP.CORE**	**Isla Fuzzer w/o constraints against Steve**	**Isla Fuzzer w/o constraints against OCPP.CORE**	**Isla Fuzzer w/ Constraints against Steve**	**Isla Fuzzer w/ Constraints against OCPP.CORE**	**State Machine Fuzzer against Steve**	**State Machine Fuzzer against OCPP.Core)**
Total Iterations	500,000	500,000	100,000	100,000	19,950	19,950	1,114,800	1,114,800
Correct Responses Rate	22.44%	22.04%	57.80%	42.07%	31.35%	33.67%	22.74%	21.86%
Incorrect Response Rate	77.56%	77.96%	42.19%	57.93%	68.65%	66.32%	77.24%	78.14%
Wrong Error Rate	51.11%	75.20%	23.18%	36.58%	49.63%	60.57%	50.99%	75.41%
Valid Requests Causing Error Rate	0%	0%	0%	19.13%	0.18%	0.98%	0%	0%
Non-error-Causing Fuzz Rate	20.82%	2.75%	13.26%	2.22%	18.83%	4.77%	20.66%	2.72%
Responses Not Valid with Protocol	5.63%	0%	5.75%	0%	0%	0%	5.58%	0%
Comprehensive Handling Rate	57.31%	86.37%	58.79%	61.99%	69.63%	82.90%	57.08%	86.48%

In Random Fuzzer against OCPP.Core, the results show a Correct Responses Rate of 22.04%, with the Comprehensive Handling Rate at a higher 86.37%. The high wrong error eate of 75.20% suggests issues in error categorization. A low non-error causing Fuzz Rate of 2.75% indicates good detection of malformed inputs.

In Isla Fuzzer w/o Constraints against Steve, the table shows a Correct Responses Rate of 57.80%, with over half of the responses meeting expectations. The Comprehensive Handling Rate is 58.79%, indicating consistent handling of messages. However, incorrect response rate of 42.19% and Wrong Error Rate of 23.18% suggest areas for improvement in error handling and accuracy.

In Isla Fuzzer w/o Constraints against OCPP.Core, the Correct Responses Rate is 42.07% , indicating that under half of the responses were as expected. The Comprehensive Handling Rate stands at 61.99%, showing good system resilience. However, a high incorrect response rate of 57.93% and wrong error rate of 36.58% reveal substantial challenges in error handling and protocol compliance.

In Isla Fuzzer w/ Constraints against Steve, the results exhibit a Correct Responses Rate of 31.35%, showing Steve's struggle with both valid and fuzzed inputs. A high Incorrect Response Rate of 68.65%, mainly due to a Wrong Error Rate of 49.63%, underscores significant error handling issues. The Comprehensive Handling Rate at 69.63% suggests moderate robustness but highlights the need for error categorization and protocol compliance improvement.

In Isla Fuzzer w/ Constraints against OCPP.Core, the results reveal a Correct Responses Rate of 33.67%, reflecting difficulties in protocol-compliant input processing. A significant Incorrect Response Rate of 66.32%, with a Wrong Error Rate of 60.57%, indicates notable error handling and classification issues. Despite these challenges, the Comprehensive Handling Rate stands at 82.90%, underscoring the system's general efficacy in handling diverse inputs but with limitations in error categorization and protocol adherence.

In State Machine Fuzzer against Steve, the results demonstrates a Correct Responses Rate of 22.74%, signifying notable protocol adherence and input processing issues. With a high incorrect Response Rate of 77.24% and a Wrong Error Rate of 50.99%, the results indicate significant error handling and categorization challenges. The Comprehensive Handling Rate at 57.08% suggests moderate effectiveness in input management, yet underscores the need for improvements in protocol compliance and error accuracy.

In State Machine Fuzzer against OCPP.Core, the results exhibits a Correct Responses Rate of 21.86%, reflecting major protocol adherence and response accuracy issues. With a significant incorrect Response Rate of 78.14%, largely due to a 75.41% Wrong Error Rate, the results highlight major error handling inaccuracies. Despite these challenges, a Comprehensive Handling Rate of 86.48% suggests a high level of system resilience in managing diverse inputs, underlining its

overall robustness.

As these results show, OCPPStorm is able to perform black box fuzzing efficiently and to catch several inaccuracies in the responses from the two OCPP CMS implementations in our evaluation. The vulnerabilities that we found in these two implementations were all triggered by the State Machine fuzzer (See Section IV-D for a detailed discussion on the vulnerabilities).

B. Constraint Violation Analysis

A fundamental component of our evaluation is the detailed analysis of constraint violations. This analysis is critical for understanding how effectively the system adheres to specified constraints under diverse testing conditions. Our focus encompasses key areas like Required Field Omission, Length Constraint Breach, and Data Type Discrepancy. These areas are crucial for evaluating the system's ability to accurately process protocol-specific deviations.

For each category, we have conducted an in-depth examination to determine the system's conformity with established protocol standards and its response to atypical input scenarios. Due to the extensive nature of this analysis, the complete results will be uploaded online in the form of a detailed table linked by the paper. In Table III, we provide a condensed excerpt of the findings, illustrating some of the primary constraint violations encountered during testing.

When analyzing the handling of protocol constraints by Steve and OCPP.Core (Table III), distinct patterns in their compliance emerge. Steve's implementation exhibits notable leniency towards protocol specifications. It tends to accept inputs that deviate from the defined constraints, such as allowing overly long fields, disregarding missing required properties, and overlooking data type discrepancies. This pattern is evident across various message properties and types. Although this suggests a degree of flexibility in input handling, it also raises concerns about security and the predictability of system behavior in response to non-standard or malformed inputs. Conversely, OCPP.Core demonstrates a more stringent adherence to protocol constraints. The system shows greater compliance, particularly in managing required fields and data types, and is more restrictive with inputs that do not adhere to the expected format or length. This rigidity indicates a focus on maintaining protocol integrity, potentially enhancing system security against malformed inputs and ensuring consistent behavior.

In essence, Steve and OCPP.Core each adopt different approaches to protocol adherence, leading to varied implications. Steve's approach, while potentially more user-friendly, could introduce security vulnerabilities. In contrast, OCPP.Core's strict adherence might increase security and consistency but at the cost of reduced flexibility.

C. Performance Metrics

Table IV shows the performance metrics of OCPPStorm for STEVE and OCPP.Core.

Table IV reveal notable differences in the performance of the State Machine, Random Fuzzer, and Isla Fuzzers. Both the State Machine and Random Fuzzers process messages at a much faster rate (approximately 521.4 to 595.5 messages per second (MPS) for Steve and 20.59 to 20.69 MPS for OCPP.Core), while the Isla Fuzzers (with and without constraints) operate more slowly (3.99 to 1.36 MPS for Steve and 3.36 to 1.37 MPS for OCPP.Core). This indicates that the isla library is much slower in generating messages compared to the other fuzzers.

Steve vs. OCPP.Core: When comparing the two implementations, Steve generally exhibits a higher message processing speed across all fuzzers, indicating a more efficient handling of incoming messages. However, this does not necessarily reflect the overall robustness or security posture of the implementations. It is essential to consider this in the context of the fuzzing method employed.

D. Vulnerability Analysis in OCPP Implementations

The vulnerabilities identified through OCPPStorm's analysis underscore the importance of strict adherence to the OCPP protocol standards. These vulnerabilities, found in both OCPP.Core and Steve implementations, are not only crucial for these specific systems but may also have implications for other systems using the OCPP protocol. Reproduction, verification, and documentation were conducted for each vulnerability, supported by detailed logs and visual evidence. After the developers were contacted for each found vulnerability, the systems were patched to fix them. In one or two cases, the behavior leading to the vulnerability results from assumptions on the behavior of the charging station. This fact points to the confusion that still exists among developers and researchers about the expected behaviors of the components of EV charging infrastructure. We describe such vulnerabilities next.[3].

1) OCPP.Core Vulnerabilities: ***DoS Vulnerability:*** A critical issue was identified in the handling of the 'chargePointVendor' field in the 'BootNotification' message. The system's failure to validate input length could lead to server instability or DoS under excessive data load. Specifically, we were able to send a message with a very large payload, which crashed the OCPP.Core implementation.

Negative Charging Transactions: The system improperly processes 'StopTransaction' messages, accepting instances where 'meterStop' values are lower than the 'meterStart' from 'StartTransaction' messages, leading to erroneous negative charging logs and potential billing discrepancies.

Unauthorized Transaction Termination: We discovered a vulnerability allowing the termination of active transactions using 'StopTransaction' messages with random 'transactionId's, indicating a lack of stringent validation checks.

Concurrent Transaction Handling: The system erroneously permits multiple transactions with identical 'connectorId' and 'idTag', deviating from the 'ConcurrentTx' status protocol.

[3]In this version of the paper, we omit the confirmed CVE numbers to preserve the submission anonymity. We will add the CVE numbers to the final paper version in case of acceptance

TABLE III
CONSTRAINT VIOLATION RESULTS FOR STEVE & OCPP.CORE

MessageType.Property	Required Field Omitted		Length Constraint Breached		Datatype Discrepancy	
	STEVE	OCPP.CORE	STEVE	OCPP.CORE	STEVE	OCPP.CORE
Authorize.idTag	X	-	X	X	X	X
BootNotification.chargePointVendor	X	-	X	X	X	X
BootNotification.chargePointModel	X	-	X	X	X	X
BootNotification.chargePointSerialNumber	-	-	X	X	X	X
BootNotification.chargeBoxSerialNumber	-	-	X	X	X	X
BootNotification.firmwareVersion	-	-	X	X	X	X
BootNotification.iccid	-	-	X	X	X	X
BootNotification.imsi	-	-	X	X	X	X
BootNotification.meterType	-	-	X	X	X	X
BootNotification.meterSerialNumber	-	-	X	X	X	X
DataTransfer.vendorId	X	-	X	X	X	-
DataTransfer.messageId	-	-	X	X	X	-
DataTransfer.data	-	-	-	-	X	-
DiagnosticsStatusNotification.status	X	-	-	-	X	-
FirmwareStatusNotification.status	X	-	-	X	X	-

TABLE IV
PERFORMANCE METRICS COMPARISON FOR STEVE & OCPP.CORE

Performance Metric	Random Fuzzer		State Machine Fuzzer		Isla Fuzzer(w/o constraints))		Isla Fuzzer(w constraints)	
	STEVE	OCPP.CORE	STEVE	OCPP.CORE	STEVE	OCPP.CORE	STEVE	OCPP.CORE
Total Messages	500000	500000	1,114,800	1,114,800	100,000	100,000	4950	19950
Messages per Second (MPS)	521.4	20.69	595.5	20.59	3.99	3.36	1.36	1.37
Total Execution Time	0h 15m 59s	6h 42m 42s	0h 31m 12s	15h 2m 33s	6h 57m 14s	8h 15m 48s	1h 0m 38s	4h 2m 30s
Average Time per Message (s)	0.0019	0.0483	0.0017	0.0486	0.2503	0.2975	0.7349	0.7293

This flaw could lead to significant transaction management and billing errors.

Handling of Additional and Duplicate Properties: The server inappropriately processes 'StartTransaction' messages with extraneous or duplicate properties without validation. Notably, the acceptance of the last occurrence in duplicate properties poses risks for transaction record manipulation and system stability.

Repeated Use of Message ID: A non-compliance issue with the OCPP specification was found, wherein the system accepts repeated use of the same message ID for different CALL messages on the same WebSocket connection. This could cause confusion in request processing and compromise transaction traceability and integrity.

The findings from OCPPStorm's evaluation of OCPP.Core demonstrate critical areas that need strengthening to enhance protocol compliance, security, and system reliability.

2) STEVE Vulnerabilities: The evaluation of Steve using OCPPStorm uncovered multiple vulnerabilities impacting its transaction processing and compliance with OCPP standards.

Invalid Timestamp Handling: Steve incorrectly processes 'StartTransaction' messages with invalid timestamps, leading to database errors and potential data integrity issues.

Multiple StopTransaction Message Handling: The system accepts multiple 'StopTransaction' messages for a single transaction, causing inconsistencies and errors in transaction recording.

Reprocessing of StartTransaction Messages: Steve fails to reject repeated 'StartTransaction' messages, allowing transaction duplication and data inconsistencies.

Unauthorized Transaction Termination: Predictable transaction IDs in Steve enable unauthorized termination of ongoing transactions, revealing a significant security vulnerability.

Billing Discrepancies from Invalid MeterStop Values: Steve processes 'StopTransaction' messages with 'meterStop' values lower than 'meterStart', leading to incorrect billing calculations.

Repeated Use of Message ID: The system does not enforce unique message IDs for CALL messages on the same WebSocket connection, undermining transaction traceability and integrity.

The discovery of these vulnerabilities demonstrates that OCPPStorm is able to successfully test the security of OCPP CMS implementations in a black box manner.

E. Discussion and Limitations

OCPPStorm is a completely black box fuzzer that, by design, does not use code execution information, like executed branches, to generate its inputs. This design choice has advantages and disadvantages. The main advantage is that OCPPStorm is application- and platform-agnostic and it can be deployed without the overhead of instrumenting the code to be tested and providing feedback about its execution. On the other hand, the lack of coverage information may lead to less accurate results and reduced coverage.

Unlike other fuzzers like AFL, OCPPStorm is focused on using only the information from the protocol specification

to practically test if an implementation conforms with the protocol. In this sense, OCPPStorm can be seen as a dynamic protocol implementation and testing tool that works as a fuzzer.

An additional limitation of OCPPStorm's current version is that it only tests the central servers' implementation of the protocol via requests sent from a charging station to a CMS. In fact, we did not test implementations of the charging point stations because we were not able to find such implementations. However, the implementation can be quickly expanded to include such messages.

V. Related Work

This chapter synthesizes the relevant literature on Electric Vehicle (EV) charging security, with a focus on the Open Charge Point Protocol (OCPP), its security aspects as well as its advanced testing methods. The burgeoning EV market has prompted extensive research on EV charging infrastructure security. Studies like [7] and [8] offer an in-depth analysis of vulnerabilities within EV charging systems, encompassing hardware, network connections, payment systems, and communication protocols like OCPP. These papers highlight the intricate security risks and the necessity for robust protective measures in the EV charging ecosystem. A classic literature in this field was published by Alacaraz et al. in [14] focusing on attacks that interfere with resource reservation originating with the Electric Vehicle (EV), which may also be initiated by a man in the middle, energy theft, or fraud. Even though the OCPP implementation of each manufacturer will have a different method of deployment [15], the security of OCPP implementation itself is also a key focus area. Rubio et al. in [9] presents a novel approach for preserving privacy in smart meter data transmission using a secret sharing scheme as a countermeasure to Man-in-the-Middle (MitM) attacks in OCPP, offering an alternative to traditional encryption-based methods. Elmo et al. in [10] demonstrate two MiTM attack scenarios to terminate charging sessions and gain root access to the EVSE equipment via remote code execution for OCPP-based systems. Furthermore, Gebauer et al. in [13] developed an OCPP server for penetration testing of OCPP charge points, and they devised three different attack scenarios which could be executed by a malicious OCPP central system to attack an OCPP charge point. Furthermore, Sarieddine et al. in their recent work focus on analyzing the OCPP communication for injection vulnerabilities [23]. The authors identify potential entry points lacking input cleansing and validation, using system fuzzing on the WebSocket handshake with various payloads. They performed OCPP backend system fuzzing on Websocket handshake using some composed payloads by monitoring HTTP request-response traffic using Burpsuite[24]. These payloads, available on this GitHub repository [26], are used to test systems for potential exploits, without causing harm [23].

Traditional testing approaches, such as the one outlined in [11], have been fundamental in ensuring OCPP compliance. However, the emergence of fuzzing techniques, as demonstrated by OCPPStorm, marks a significant shift in protocol testing. Fuzzing offers a more dynamic assessment, uncovering vulnerabilities beyond the scope of predefined test cases, and thus enhancing the reliability of EV charging systems. Advancements in automated fuzzing, such as those demonstrated by AutoFuzz[12], have set the stage for specialized tools like OCPPStorm. OCPPStorm's targeted approach, tailored for OCPP, goes beyond generic fuzzing, providing a more nuanced analysis vital for EV charging security.

In another study by Nasr et al. with a broader perspective, considering the whole Electric Vehicle Charging System (EVCS) as a larger system vulnerable to attacks, it was found that these systems are susceptible to manipulations of charging operations that could potentially harm the EV battery by altering charging levels. The research also revealed that attackers could upload malicious firmware, thereby maintaining covert access, and use the compromised EVCS for coordinated network activities, including denial of service attacks [27]. Furthermore, these systems could be locked to deny access to legitimate users, potentially leading to ransomware attacks for financial gain [27].

Another protocol within the EV charging infrastructure is ISO-15118, which has also been a matter of importance in terms of implementation testing and threat analysis due to its connection with the OCPP protocol. Any vulnerabilities or attacks within the ISO-15118 protocol can potentially impact the OCPP as well, thereby emphasizing the need for rigorous testing. Lee et al. focuses on the data used in the communication between electric vehicles and power charging infrastructure. It analyzes the security vulnerabilities of the ISO/IEC-15118 standard-based charging technology and proposes countermeasures for them [20]. Bao et al. introduces adversary-centric threat analysis, which exposes implicit assumptions that need to be fulfilled by the deployment context of a charging point infrastructure to guarantee security. This analysis focuses on the scope boundaries of the communication protocol between electric vehicles and charging stations [21]. Researchers have also explored the challenges implicated by the wireless communication of ISO-15118 and provided a proof of concept prototypical ISO-15118 based implementation that provides a mechanism for achieving EV-EVSE Pairing, next to the Association and Authentication [16]. In addition, the ISO-15118 protocol stack has been subjected to Fuzz Testing, a method that enhances the test for robustness and negative invalid input [18]. Conformance tests for combined communication and power interfaces have also been conducted [17]. Furthermore, a test system according to ISO/IEC-15118 has been presented, using the TTCN-3 testing framework for maximum flexibility to support various testing scenarios [19]. These studies highlight the importance of rigorous testing in ensuring the robustness and security of the ISO-15118 protocol, similar to the OCPP protocol.

VI. Conclusion

In this paper, we describe OCPPStorm, which is able to test the security of different OCPP implementations in a language- and platform-agnostic way. OCPPStorm is evaluated against two available OCPP implementations and is able to find several vulnerabilities and errors in handling OCPP messages. Future work includes integration of white-box fuzzing into OCPP-Storm and incorporation of static code analysis techniques, providing a comprehensive view of potential vulnerabilities and contributing to the development of a more robust security testing framework for OCPP implementations.

Acknowledgment

This work was supported by the National Science Foundation under award 2330565 and the UK Research and Innovation organization (UKRI) under award EP/Y026233/1.

References

[1] BloombergNEF. 2020. “Electric Vehicle Outlook 2020”. https://about.bnef.com/electric-vehicle-outlook/

[2] National Geographic. 2017. “Electric Cars May Rule the World’s Roads by 2040”. https://www.nationalgeographic.com/news/2017/09/electric-cars-replace-gasoline-engines-2040

[3] Geotab. Electric Vehicle Trends in 2020: Top 6 Factors Impacting Fleet Electrification; Geotab: Oakville, ON, Canada, 2020.

[4] ”Open Charge Alliance,” Open Charge Alliance (OCA), Tech. Rep., 2020. [Online]. Available: https://www.openchargealliance.org/

[5] S. Acharya, Y. Dvorkin, H. Pandzi˘ c, and R. Karri, “Cybersecurity of ´ smart electric vehicle charging: A power grid perspective,” IEEE Access, vol. 8, pp. 214 434–214 453, 2020.

[6] “Open Charge Alliance - Our mission,” 2022. [Online]. Available: https://www.openchargealliance.org/about-us/

[7] Antoun, J., Kabir, M.E., Moussa, B., Atallah, R. and Assi, C., 2020. A detailed security assessment of the EV charging ecosystem. IEEE Network, 34(3), pp.200-207.

[8] Johnson, J., Berg, T., Anderson, B. and Wright, B., 2022. Review of electric vehicle charger cybersecurity vulnerabilities, potential impacts, and defenses. Energies, 15(11), p.3931.

[9] Rubio, J.E., Alcaraz, C. and Lopez, J., 2018, February. Addressing security in OCPP: Protection against man-in-the-middle attacks. In 2018 9th IFIP International Conference on New Technologies, Mobility and Security (NTMS) (pp. 1-5). IEEE.

[10] Elmo, D., Fragkos, G., Johnson, J., Rohde, K., Salinas, S. and Zhang, J., 2023, November. Disrupting EV Charging Sessions and Gaining Remote Code Execution with DoS, MITM, and Code Injection Exploits using OCPP 1.6. In 2023 Resilience Week (RWS) (pp. 1-8). IEEE.

[11] Priyasta, D., Septiawan, R., Herminawan, F. and Bayu, H., 2023. Ensuring Compliance and Reliability in EV Charging Station Management Systems: A Novel Testing Tool for OCPP 1.6 Messages Conformance. Journal Européen des Systèmes Automatisés, 56(1).

[12] Gorbunov, S. and Rosenbloom, A., 2010. Autofuzz: Automated network protocol fuzzing framework. Ijcsns, 10(8), p.239.

[13] L. Gebauer, Henning Trsek, and G. Lukas, “Evil SteVe: An Approach to Simplify Penetration Testing of OCPP Charge Points,” 2022 IEEE 27th International Conference on Emerging Technologies and Factory Automation (ETFA), Sep. 2022.

[14] C. Alcaraz, J. Lopez, and S. Wolthusen, “OCPP Protocol: Security Threats and Challenges,” IEEE Transactions on Smart Grid, vol. 8, no. 5, pp. 2452–2459, Sep. 2017.

[15] A. Sanghvi and T. Markel, “Cybersecurity for Electric Vehicle Fast-Charging Infrastructure,” 2021 IEEE Transportation Electrification Conference & Expo (ITEC), Jun. 2021.

[16] N. E. Sayed, ”A Prototypical Implementation of an ISO-15118-Based Wireless Vehicle to Grid Communication for Authentication over Decoupled Technologies,” 2019 AEIT International Conference of Electrical and Electronic Technologies for Automotive (AEIT AUTOMOTIVE), Turin, Italy, 2019, pp. 1-6, doi: 10.23919/EETA.2019.8804545.

[17] K. Hänsch et al., ”An ISO/IEC 15118 conformance testing system architecture,” 2014 IEEE PES General Meeting — Conference & Exposition, National Harbor, MD, USA, 2014, pp. 1-5, doi: 10.1109/PESGM.2014.6938863.

[18] Schöneberger, T. FUZZ testing the ISO 15118 protocol stack - Vector Informatik GmbH. Available at: https://cdn.vector.com/cms/content/events/2021/vSES21/vSES21_Slides_07_Schoeneberger_Vector.pdf (Accessed: 15 December 2023).

[19] Shin, M.; Kim, H.; Kim, H.; Jang, H. Building an Interoperability Test System for Electric Vehicle Chargers Based on ISO/IEC 15118 and IEC 61850 Standards. Appl. Sci. 2016, 6, 165. https://doi.org/10.3390/app6060165

[20] S. Lee, Y. Park, H. Lim and T. Shon, ”Study on Analysis of Security Vulnerabilities and Countermeasures in ISO/IEC 15118 Based Electric Vehicle Charging Technology,” 2014 International Conference on IT Convergence and Security (ICITCS), Beijing, China, 2014, pp. 1-4, doi: 10.1109/ICITCS.2014.7021815.

[21] Bao, K., Valev, H., Wagner, M. et al. A threat analysis of the vehicle-to-grid charging protocol ISO 15118. Comput Sci Res Dev 33, 3–12 (2018). https://doi.org/10.1007/s00450-017-0342-y

[22] Dominic Steinhöfel and Andreas Zeller. ”Input Invariants”. Software Engineering 2023, Fachtagung des GI-Fachbereichs Softwaretechnik. 24 Februar 2023, Paderborn. pp. 113–114, 2023.

[23] K. Sarieddine et al., “Uncovering Covert Attacks on EV Charging Infrastructure: How OCPP Backend Vulnerabilities Could Compromise Your System”.

[24] Burp Suite. [n. d.]. BURP suite - application security testing software. https: //portswigger.net/burp

[25] https://openchargealliance.org/my-oca/ocpp/

[26] 1N3. [n. d.]. 1N3/intruderpayloads: A collection of burpsuite intruder payloads, Burpbounty payloads, Fuzz Lists, malicious file uploads and web pentesting methodologies and checklists. https://github.com/1N3/IntruderPayloads

[27] Nasr, T. et al. (2023) ‘Chargeprint: A framework for internet-scale discovery and security analysis of EV charging management systems’, Proceedings 2023 Network and Distributed System Security Symposium [Preprint]. doi:10.14722/ndss.2023.23084.