EasyChair Preprint

№ 15790

V2G Fuzzer : Fuzzing Tool for Implementing Electric Vehicle Charger V2G Communication

Yu-Bin Kim, Dong-Hyuk Shin, Jae-Jun Ha and Ieck-Chae Euom

February 3, 2025

V2G Fuzzer : Fuzzing Tool for Implementing Electric Vehicle Charger V2G Communication

Yu-Bin Kim[1*], Dong-Hyuk Shin[2], Jae-Jun Ha[2] and Ieck-Chae Euom[3†]
[1, 2, 3] System Security Research Center, Chonnam National University
kingyoubin97@jnu.ac.kr, shindh2@jnu.ac.kr, wowns1106@jnu.ac.kr
iceoum@jnu.ac.kr

Abstract

The rise in electric vehicles(EVs) has led to a rapid increase in the number of electric vehicle charging stations(EVCSs) with approximately 5 million installed worldwide by 2023. These EVCSs are operated according to various standards and protocols. ISO 15118, which is used for communication between EVs and EVCS, lacks security guidelines, and this absence can result in numerous vulnerabilities due to improper implementation. This study introduces the V2G Fuzzer, a security testing tool designed to prevent vulnerabilities caused by incorrect implementations in EV CSs. The tool is designed as a black-box testing solution capable of handling various implementations, regardless of the EVCS platform or programming language used. The fuzzing technique is applied to identify errors and discover vulnerabilities in the application layer where messages are processed. To validate the effectiveness of this approach, fuzzing tests were conducted on open-source EVCS implementations. The results confirmed that the tool is effective in determining whether the open-source projects correctly implement the ISO 15118 standard and in detecting potential vulnerabilities.

1 Introduction

The rapid adoption of electric vehicles (EVs) is transforming the automotive ecosystem, with projections indicating that over 50% of all vehicles will be electric in the near future [1]. This surge has led to the installation of more than 5 million electric vehicle charging stations (EVCS) globally as of 2023, creating a large need for charging infrastructure [2]. A comprehensive infrastructure is essential to provide effective EV charging services, comprising various components and standards. Key components include EVCS, Central Management Systems (CMS), and Electric Mobility Service

[*] Masterminded EasyChair and created the first stable version of this document
[†] Created the first draft of this document

Providers (e-MSP). The ISO 15118 standard plays a crucial role by governing communication between EV and EVCS among the standards facilitating this ecosystem.

ISO 15118 not only manages the communication protocols but also controls the charging process and handles the transmission of critical information such as payment details. Encryption via Transport Layer Security (TLS) is supported by ISO 15118 however, its use is not mandatory. Approximately 90% of EVCSs do not implement TLS encryption, as observed in practice [3]. The ISO 15118 standard lacks specific security guidelines and has insufficient implementation requirements. This deficiency creates opportunities for attackers to exploit the communication interface between EVs and EVCSs, potentially leading to hacks of charging stations or broader attacks on the charging infrastructure. Indeed, various threats have been identified targeting the interfaces that communicate with EVs [4].

This study proposes the design of a black-box fuzzing tool aimed at evaluating whether EVCSs have properly implemented the ISO 15118 standard and at identifying additional vulnerabilities at the application layer. This approach allows for the assessment of system functionality without examining internal code structures, logic, or implementation details. Fuzzing techniques employing various strategies are utilized to explore and exploit security vulnerabilities deeply embedded within protocol interactions. The main contributions of this paper are as follows:

(i) We propose a black-box fuzzing methodology that enables the evaluation of multiple ISO 15118 implementations without prior knowledge of the system's internal workings. 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 ISO 15118 protocol interactions. (iii) Using our proposed fuzzing tool, we identify several vulnerabilities in EVCS implementations. Based on these findings, we suggest measures to enhance the security of the ISO 15118 standard.

2 Background and Related Work

2.1 V2G Communication

First published in 2013, ISO 15118 enabled communication between EV and EVCS, known as Vehicle-to-Grid (V2G) communication. ISO 15118 defines the requirements for implementing communication from the physical layer up to the application layer within the OSI seven-layer model [5]. It also provides test cases to verify proper compliance with the standard. Figure 1 illustrates the relationship between the ISO 15118 standard and the OSI seven-layer model. ISO 15118 operates based on IEC 61851, which is used for conventional EV charging. Power Line Communication (PLC) is performed through the CP (Control Pilot) line of the EVCS cable, and digital communication based on ISO 15118 can be selected via the duty cycle of the CP line [6].

The main standards for communication are ISO 15118-2[7] and ISO 15118-3[8]. ISO 15118-3 defines the requirements for the physical and data link layers for communication between EVs and charging stations and specifies how to transmit data using PLC communication. ISO 15118-2 defines the requirements related to the network layer up to the application layer. ISO 15118-2 specifies the messages exchanged between EVs and charging stations, outlining the flow from IP identification of the EV and EVCS to the termination of charging. Additionally, data is defined to be transmitted in the EXI (Efficient XML Interchange) format, which encodes XML (Extensible Markup Language) data.

EVs and charging stations that satisfy the requirements defined in these standards can perform data communication based on ISO 15118.

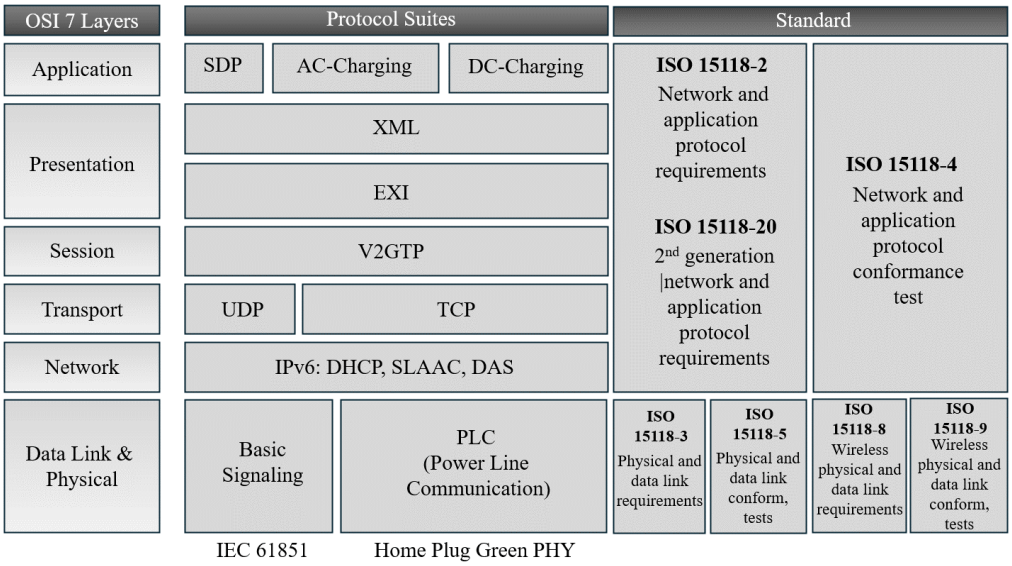

Figure 1: ISO 15118 Protocol Stack and Relationship to OSI Layer 7

2.2 Fuzzing Test

Fuzz testing, commonly known as fuzzing, is a software testing technique used to discover coding errors and security vulnerabilities in software. A fuzzer is a type of software that automates this testing process by providing unexpected or random data inputs to a program. The primary goal of fuzzing is not only to test software functionality but also to explore and identify bugs and vulnerabilities such as coding errors, buffer overflow vulnerabilities, and possibilities of denial-of-service attacks. This can be achieved by inducing unexpected behavior through the use of malformed or random data as program inputs. Fuzzing discovers defects more easily by attempting various combinations of input values to cause crashes or abnormal behavior in software. Traditional fuzzing techniques can be broadly classified into three categories: black-box fuzzing, white-box fuzzing, and gray-box fuzzing.

Black-Box Fuzzing: Black-box fuzzers like zzuf[9] do not require access to the source code or knowledge of the internal implementation details of the target software. Performing fuzzing by generating test cases randomly in a black-box manner is relatively straightforward, but this approach can be inefficient. Randomly generated input values may be rejected as invalid by the software or may fail to reach deeper parts of the code, limiting the effectiveness of the testing. However, if information about the protocol used by the target or the format of the messages is known, black-box-based fuzzing can still perform effective testing. This approach can be highly practical and enhance testing efficiency.

White-Box Fuzzing: White-box fuzzing techniques optimize test case generation by utilizing the source code of the target software. These techniques analyze the code to systematically explore execution paths using dynamic symbolic execution and other advanced methods. This allows for a more thorough examination of the software to discover bugs and vulnerabilities. However, white-box fuzzing requires access to the source code and may not be practical in situations where the source code is unavailable or resources are limited.

Gray-Box Fuzzing: Gray-box fuzzing is more practical for real-world applications where source code access is not available because it does not require source code access. Gray-box fuzzers like AFL[10]

instrument the target binary to collect useful runtime information such as code coverage data. This feedback guides the fuzzer to dynamically generate new test cases that are more likely to explore unexplored code paths, improving the efficiency and effectiveness of the fuzzing process.

In summary, fuzz testing is a powerful technique for discovering software vulnerabilities by injecting unexpected inputs and monitoring the resulting behavior. The choice among black-box, white-box, and gray-box fuzzing depends on specific testing requirements, the availability of source code, and resources. Testers can effectively identify and mitigate potential security risks in software systems by leveraging the strengths of each approach.

2.3 Network Protocol Fuzzing

Network protocol fuzzing is a software testing technique used to detect vulnerabilities in network protocols [11]. It works by supplying the system with randomly generated or intentionally malformed input data to trigger abnormal behavior. Network protocol fuzzing is primarily employed to identify vulnerabilities in protocol implementations by inducing errors in message structures or sequences, revealing potential security flaws.

Two main types of network protocol fuzzing exist. The first targets protocols with publicly available specifications, as seen in research like L2Fuzz[12] and Z-Fuzzer[13]. These approaches create test cases based on official protocol documentation to find vulnerabilities. The second type focuses on protocols without publicly available specifications. Research such as Snipuzz[14] uses response values from messages to infer message formats, allowing fuzzing in protocols where the specifications are not disclosed.

The complexity of network protocol fuzzing goes beyond simple data exchange. It also involves handling functions like transmission error detection, timeout and retry management, and flow control, which makes it more difficult than general software fuzzing. A well-designed communication model that manages the protocol’s communication flow and states is essential for effective fuzzing. Incorrect or incomplete communication models hinder successful fuzzing efforts.

2.4 Related Work

The fuzz testing research on the ISO 15118 protocol stack focused on identifying security vulnerabilities in the communication interface between vehicles and the grid. In particular, the study targeted the TLS protocol, manipulating message fields to uncover various vulnerabilities during the TLS handshake process [15]. By using fuzzing techniques to inject malformed certificate data and manipulate message fields, the research demonstrated that the ISO 15118 protocol is vulnerable to different forms of invalid data inputs. This study contributed to a deeper analysis of potential risks in secure communications within the electric vehicle charging infrastructure. However, since the research was limited to the TLS layer, it was difficult to identify vulnerabilities at higher layers, such as the application layer.

In addition, similar research was conducted targeting CMS. This research performed fuzzing tests based on the Open Charge Point Protocol (OCPP), which governs the communication between EVCS and CMS [16]. OCPP, as a key component of the EV charging infrastructure, ensures interoperability between EVCS and CMS. The study generated two types of test cases: one that adhered to the protocol's constraints and another that deliberately violated them, to verify how well the CMS implemented the OCPP protocol. Fuzzing, performed with consideration of OCPP's state machine transitions, uncovered vulnerabilities arising from complex interactions. Among the vulnerabilities found, 5 were confirmed as Common Vulnerabilities and Exposures (CVEs), with 7 more under review.

3 V2G Fuzzer Design

This section examines the design of the V2G fuzzer and analyzes the specific functions of each component. First, it describes the design of the communication model used to perform network protocol fuzzing on EVCS. Next, it provides a detailed explanation of each component of the V2G fuzzer. Figure 2 shows the architecture of the V2G fuzzer.

The communication model for fuzzing the EVCS through the interaction between the EV and EVCS is designed and implemented to enable seamless communication with the EVCS. The EV and EVCS communicate using the ISO 15118 standard, an open protocol. Based on this standard, a top-down approach was employed to design the communication model. The focus of this study is to perform fuzzing tests targeting the application layer within the EV charging communication flow. Therefore, the model is designed to enable fuzzing at specific points in the communication flow. Figure 3 illustrates the V2G communication process following the ISO 15118 standard, and the model ensures that fuzzing can be applied to the application layer during this process. This fuzzing tool is designed as a black-box approach to be performed directly on actual EVCS. This allows fuzzing to be conducted without dependency on the language or platform in which the EV CS is implemented. Additionally, while white-box or gray-box approaches require knowledge of the code or techniques for emulating the EVCS firmware, the black-box approach does not require such techniques, making it a more practical solution.

This tool establishes a session with the EVCS, allowing communication to input messages—something that conventional network fuzzing tools find challenging. While it similarly inputs data through packets, it follows the specific communication sequence of the EV charger, enabling fuzzing that is specialized for EVCS During the design process, a major challenge was the lack of suitable existing fuzzing tools to reference for EVCS. Consequently, it was necessary to develop a dedicated fuzzing tool for EVCS from the ground up. To address this, a communication model was designed to simulate the network protocol between chargers and electric vehicles based on EVCS simulators. This design established a foundation for more effectively analyzing potential vulnerabilities in V2G communication. This fuzzing tool establishes a session with the EVCS and generates messages to perform fuzzing on the target EVCS. During the fuzzing process, the Message Generator creates messages, and each fuzzing module manages these messages and transmits values accordingly. The

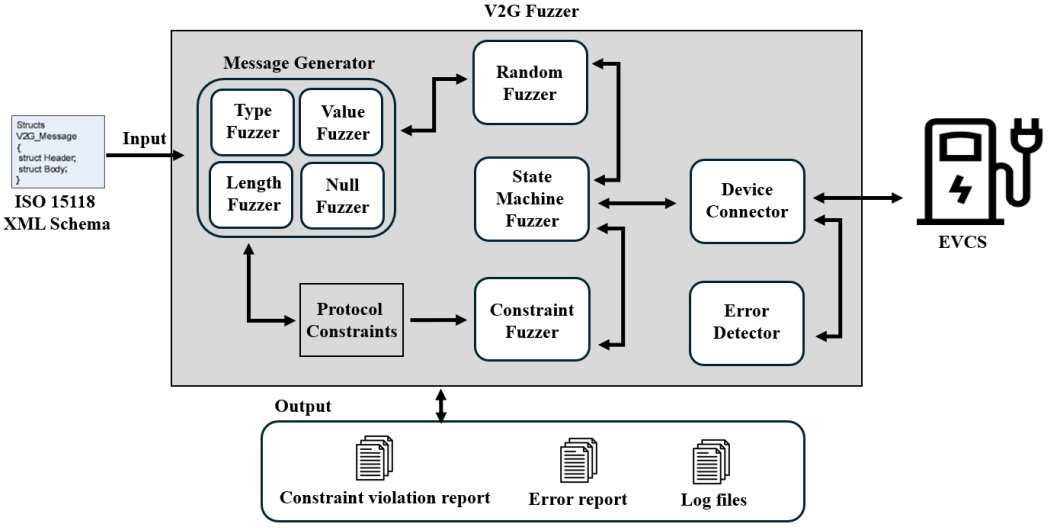

Figure 2: V2G Fuzzer Architecture

Error Detector then checks the responses. The EVCS uses TCP communication, and if the firmware or system of the charger crashes, no TCP response is received. This setup is designed to identify whether an error occurred based on the input by detecting the absence of a response.

3.1 Device Connector

The Device Connector facilitates the establishment of a V2G (Vehicle-to-Grid) session between an EV and an EVCS. This session enables communication when the EV and the EVCS are connected via a charging cable. Upon connection, the EV initiates the Signal Level Attenuation Characterization (SLAC) protocol, which verifies the physical link between the EV and EVCS and exchanges necessary MAC addresses for communication.

Following SLAC, the SECC Discovery Protocol (SDP) is executed, allowing the exchange of IP addresses between the EV and EVCS. Once IP addresses are exchanged, TCP session is established, enabling message transmission. Given that communication depends on the TCP protocol, it is crucial to manage the Seq and Ack numbers accurately when transmitting fuzzing messages. Incorrect Seq and Ack number management may disrupt communication or result in message rejection by the EVCS. The Device Connector ensures this precise management of Seq and Ack numbers, thereby maintaining reliable V2G communication.

3.2 Fuzzing Module

This study utilizes three fuzzing mechanisms—Random Fuzzer, State Machine Fuzzer, and Constraint Fuzzer—to test the security of EVCSs using the ISO 15118 protocol.

Random Fuzzer: The Random Fuzzer generates XML data based on the XML schema defined in ISO 15118, mutating the values of XML elements without violating the schema. The fuzzing is performed by altering the data's type, value, length, and by using null values. This approach allows testing of multiple scenarios where vulnerabilities might occur.

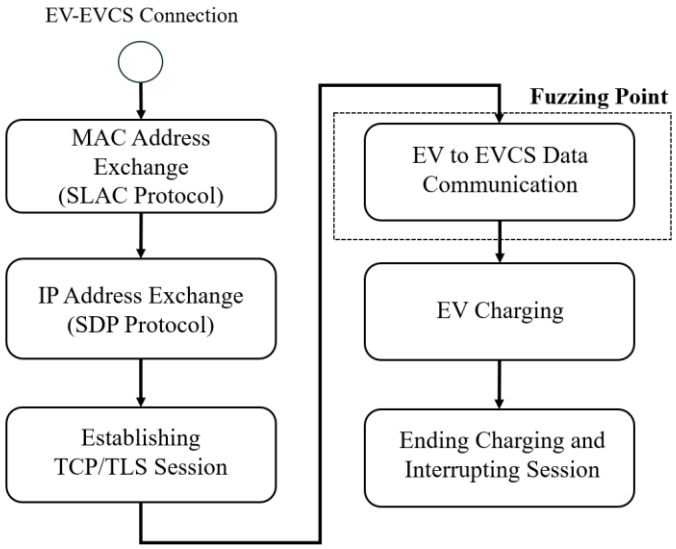

Figure 3: V2G Communicaion Sequence

State Machine Fuzzer: The State Machine Fuzzer simulates valid communication paths between the EV and EVCS before introducing fuzzing at specific states, providing deeper protocol coverage. The fuzzer follows valid ISO 15118 message sequences, such as those for starting a charging session, and performs fuzzing from the state reached. ISO 15118 defines 14 states for the EVCS, and the state machine fuzzer facilitates fuzzing at specific states. For example, in the Wait for supportedAppProtocolReq state, the EVCS waits for the SupportedAppProtocolReq message, and the fuzzer generates messages based on this to conduct effective fuzzing. The fuzzer ensures that all states are covered.

Constraint Fuzzer: The Constraint Fuzzer operates by applying specific constraints to XML attributes when generating messages. For example, the ServiceScope element in the ServiceDiscoveryReq message is a string type with a length limit of 32 bytes. The fuzzer ensures that even within these constraints, vulnerabilities can be discovered. Additionally, the fuzzer tests violations of such constraints to identify whether the EVCS correctly handles exceptions. This process ultimately verifies whether the EVCSs is properly implementing the ISO 15118 standard.

3.3 Message Generator

The message generator is finely tuned to perform effective fuzzing on EVCS, considering the complexity of ISO 15118. It is designed to conduct fuzzing tests on EVCS by utilizing the XML schema provided by ISO 15118. Data mutations occur in four ways: type mutation, value mutation, data length mutation, and the insertion of null values when generating messages. Message generation is divided into two main approaches. The first approach generates data that adheres to the constraints, based on the 14 messages exchanged during communication between the EV and EVCS. This method is used to verify whether the EVCS is correctly implemented and to identify any potential vulnerabilities under ISO 15118-compliant conditions. The second approach violates these constraints, testing how robustly the EVCS responds when given values that do not conform. This method assesses the system's resilience and security by analyzing the effects of exceptional or invalid inputs on the EVCS.

3.4 Error Detector

The error detector plays a role in detecting errors or bugs that occur in EVCS, identifying exceptional situations. Since EVCS and EV are fundamentally connected via TCP communication, any bugs in the EVCS software can be detected immediately. When a software bug occurs, the software is unable to respond to TCP requests due to the exceptional situation. For this reason, after sending a fuzzing message, the TCP response is checked before the next fuzzing message is transmitted. However, ISO 15118 does not define specific responses to exceptional input values, making it difficult to identify the exact error that has occurred in many cases. This study addresses this challenge by utilizing software debug outputs to analyze errors in more detail. As a result, it becomes possible to accurately identify which fuzzing message triggered the error.

Additionally, the responses are analyzed to verify that correct responses are returned for input values conforming to the standard. For example, according to the ISO 15118 standard, when the EV charger receives a valid input, it is defined to return the value "OK" in the Response Code field. By verifying that the expected response is received for standard-compliant input values, compliance with the standard can be assessed.

4 Implementation and Analysis

This section presents the experimental environment and the analysis of the fuzzing test results. In this study, only the Constraint Fuzzer and Random Fuzzer were utilized, while the State Machine Fuzzer

was not used, and fuzzing was performed on a single state. The fuzzing process targeted the state of the electric EVCS that waits for the supportedAppProtocolReq message.

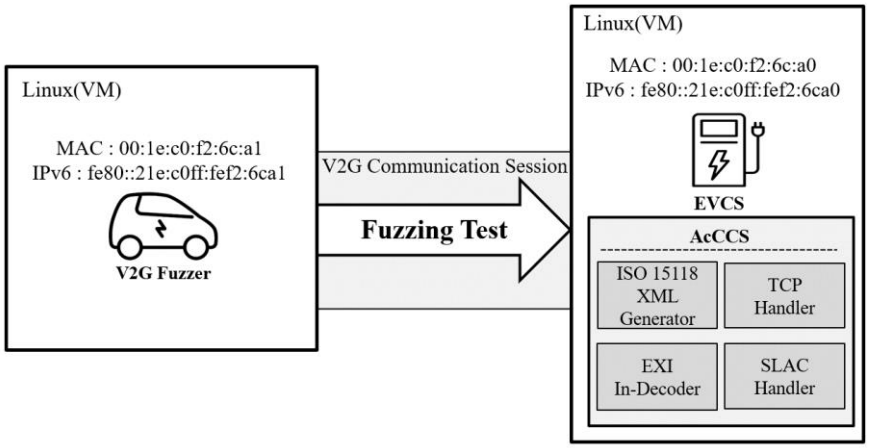

Figure 4: Experiment environment

4.1 Experimental Environment

This study conducted fuzzing tests on EVCS using the open-source software AcCCS[17], which mimics the communication of real EVCS rather than using actual EVCSs. AcCCS was developed by Idaho National Laboratory in the United States to sniff data and perform man-in-the-middle attacks on communication between electric vehicles and EVCS. The experimental environment, as shown in Figure 4, utilized two virtual environments—one running the fuzzing tool and the other running the AcCCS software. The communication was established over IPv6, simulating a real EVCS environment. This setup was designed to facilitate network fuzzing on actual EVCS in future tests.

4.2 Constraint-Based Analysis

AcCCS was fuzzed to verify whether it processes data in compliance with the specified constraints. ISO 15118 provides the constraints for the SupportedAppReq message, which are listed in Table 1. The

Elment	Type	Length	Data Description
ProtocolNameSpace	Stirng	100	Specific protocols supported by EVCS
VersionNumberMajor	UnsingnedInt	-	Major version number of the protocol
VersionNumberMinor	UnsingnedInt	-	Manor version number of the protocol
SchemaID	UnsingnedByte	-	SchemaID assigned to the protocol
Priority	UnsingnedByte	1~20	Protocol priority

Table 1: SupportedAppProtocolReq Message Elements

fuzzing process, based on these constraints, revealed three software errors, as shown in Table 2. These errors include Out Of Memory Error, Index Error, and ParseError. Figure 5 is an Error Massage of AcCCS in action Each of the three errors caused AcCCS to halt, which could potentially lead to Denial of Service (DoS) attacks. The ProtocolNameSpace element in the SupportedAppProtocolReq message was the source of the errors, while other elements remained unaffected. ProtocolNameSpace is a string-

type element limited to 100 characters. AcCCS, however, failed to handle special characters in the string value, resulting in a ParseError. When AcCCS parsed the XML data, characters like <, >, and . caused ParseErrors, and due to the lack of exception handling, the program terminated unexpectedly.

An Index Error occurred when the value of the ProtocolNameSpace element exceeded 30 characters. Although AcCCS should accept up to 100 characters, as defined by the standard constraints, it failed to do so. The Out Of Memory Error occurred when continuous data input overwhelmed the memory. The EXI decoding module in AcCCS could not handle the sustained input, leading to Out of Memory Error

Error Type	Occurred Element	Cause
Index of Error	ProtocolNameSpace	Element value greater than or equal to 30 characters
Parse Error		Special symbols
Out of Memory		Mass Data Input

Table 2: List of errors discovered

These three errors occurred despite AcCCS following the standard constraints, mainly due to poor exception handling and flawed implementation. Fuzzing without following the constraints produced similar results to those obtained while adhering to the constraints. This analysis confirmed that errors can be identified in software implementing electric vehicle charging communication. Testing a single state revealed three errors, suggesting that further fuzzing across multiple states may uncover additional issues. The discovery of three vulnerabilities in a single state suggests that additional vulnerabilities could emerge when fuzzing is performed on other states. Furthermore, identifying these vulnerabilities requires initiating communication prior to handling application-layer messages, which can be challenging for conventional fuzzing tools to detect. Thus, the effectiveness of the proposed fuzzing tool was confirmed.

Figure 5: AcCCS error message

5 Discussion

This study raises two key discussion points the issue of validity verification and the actual design of charging hardware. We developed a methodology to fuzz EVCS and performed fuzzing on EVCS

simulations. However, since we did not conduct fuzzing on the actual firmware and software installed in real EVCS, further fuzzing on real-world EVCS is necessary for proper validity verification. Additionally, the introduction of new evaluation metrics is required to ensure thorough verification.

While the experiments were conducted in a virtual environment, it is necessary to implement the PLC used by EVCSs to enable testing on actual chargers. There are existing studies that have established communication testbeds using real power line modems, and based on this, it is essential to incorporate hardware elements into future work.

6 Conclusions and Future Research

The increasing number of electric vehicles (EVs) has led to a rise in the number of EVCSs, and the infrastructure for EV charging services is being actively developed. Standards and protocols exist to support the implementation of EVCS infrastructure, and among them, ISO 15118 has gained significant attention for enabling V2G communication. However, implementing ISO 15118 presents challenges, as the standard includes limited security guidelines, leading to errors and vulnerabilities due to improper implementation. To address this, we propose a fuzzing tool to test whether EVCS correctly implement the ISO 15118 standard. The fuzzing tool incorporates a Constraint Fuzzer, a State Machine Fuzzer, and a Random Fuzzer, which interact to manage the complex communication processes and state machines of EVCS. These fuzzers assess compliance with constraints and identify potential vulnerabilities.

We conducted fuzzing on an EVCS communication simulator and identified three vulnerabilities. However, since this fuzzing was limited to a single state, future research will expand fuzzing to all state machines to discover more errors and vulnerabilities. Additionally, we plan to continue our research by testing EVCS firmware and other open-source projects related to EVCS communication.

Acknowledgment

"This work was supported by Institute for Information & communications Technology Planning & Evaluation(IITP) grant funded by the Korea government(MSIT)(No.IITP-RS-2022-II221203, 50% contribution, Regional strategic Industry convergence security core talent training business) and This work was supported by a grant from the Korea Electric Power Corporation (R24XO01-4, 50% contribution) for basic research and development projects starting in 2024.

References

National Geographic. (2017). Electric cars may rule the world’s roads by 2040. Retrieved from https://www.nationalgeographic.com/news/2017/09/electriccars-replace-gasoline-engines-2040(Accessed: September 20, 2024).

BloombergNEF. (2023). Electric vehicle outlook 2023.

Szakály, M., Köhler, S., & Martinovic, I. (2024). Current affairs: A measurement study of deployment and security trends in EV charging infrastructure. arXiv preprint arXiv:2404.06635.

A threat analysis of the vehicle-to-grid charging protocol ISO 15118. (n.d.). Retrieved from https://link.springer.com/article/10.1007/s00450-017-0342-y (Accessed: September 20, 2024).

International Organization for Standardization. (n.d.). ISO 15118-1: Road vehicles — Vehicle to grid communication interface — Part 1: General information and use-case definition. Retrieved from https://www.iso.org/standard/55365.html (Accessed: September 20, 2024).

International Electrotechnical Commission. (n.d.). IEC 61851: Electric vehicle conductive charging system. Retrieved from https://www.iec.ch/standards/iec61851 (Accessed: September 20, 2024).

International Organization for Standardization. (n.d.). ISO 15118-2: Road vehicles — Vehicle to grid communication interface — Part 2: Network and application protocol requirements. Retrieved from https://www.iso.org/standard/66975.html (Accessed: September 20, 2024).

International Organization for Standardization. (n.d.). ISO 15118-3: Road vehicles — Vehicle to grid communication interface — Part 3: Physical and data link layer requirements. Retrieved from https://www.iso.org/standard/69156.html (Accessed: September 20, 2024).

Samhocevar. (n.d.). Retrieved from https://github.com/samhocevar/zzuf (Accessed: September 20, 2024).

American fuzzy lop. (n.d.). Retrieved from https://lcamtuf.coredump.cx/afl/ (Accessed: September 20, 2024).

Zhang, X., He, L., Li, Y., Wang, J., & Liu, C. (2024). A survey of protocol fuzzing. *arXiv preprint arXiv:2401.01568*.

Park, H., Kim, S., Kim, J., Lee, H., Kim, T., & Oh, H. (2022). L2Fuzz: Discovering Bluetooth L2CAP vulnerabilities using stateful fuzz testing. In 2022 52nd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN) (pp. X-Y). IEEE.

Ren, M., He, L., Zhang, H., Huang, Y., & Li, C. (2021). Z-Fuzzer: Device-agnostic fuzzing of Zigbee protocol implementation. In Proceedings of the 14th ACM Conference on Security and Privacy in Wireless and Mobile Networks (pp. X-Y).

Feng, X., Zhang, J., Yu, Z., & Dong, M. (2021). Snipuzz: Black-box fuzzing of IoT firmware via message snippet inference. In Proceedings of the 2021 ACM SIGSAC conference on computer and communications security (pp. X-Y).

Schoneberger, T. (2023). FUZZ testing the ISO 15118 protocol stack. Vector Informatik GmbH. Retrieved from https://cdn.vector.com/cms/content/events/2021/vSES21/vSES21Slides07SchoenebergerVector.pdf (Accessed: December 15, 2023).

Coppoletta, G. (2024). OCPPStorm: A comprehensive fuzzing tool for OCPP implementations (Master's thesis). University of Illinois at Chicago.

IdahoLabResearch. (n.d.). Retrieved from https://github.com/IdahoLabResearch/AcCCS(Accessed: September 20, 2024).