Disrupting EV Charging Sessions and Gaining Remote Code Execution with DoS, MITM, and Code Injection Exploits using OCPP 1.6

David Elmo II
Department of Computer Science and Engineering
Wright State University
Dayton, USA
davidelmo2@gmail.com

George Fragkos
Sandia National Laboratories
Albuquerque, USA
gfragko@sandia.gov

Jay Johnson
Sandia National Laboratories
Albuquerque, USA
jjohns2@sandia.gov

Kenneth Rohde
Idaho National Laboratory
Idaho Falls, USA
kenneth.rohde@inl.gov

Sean Salinas
Idaho National Laboratory
Idaho Falls, USA
sean.salinas@inl.gov

Junjie Zhang
Department of Computer Science and Engineering
Wright State University
Dayton, USA
junjie.zhang@wright.edu

***Abstract*—Open Charge Point Protocol (OCPP) 1.6 is widely used in the electric vehicle (EV) charging industry to communicate between Charging System Management Services (CSMSs) and Electric Vehicle Supply Equipment (EVSE). Unlike OCPP 2.0.1, OCPP 1.6 uses unencrypted websocket communications to exchange information between EVSE devices and an on-premise or cloud-based CSMS. In this work, we demonstrate two machine-in-the-middle attacks on OCPP sessions to terminate charging sessions and gain root access to the EVSE equipment via remote code execution. Second, we demonstrate a malicious firmware update with a code injection payload to compromise an EVSE. Lastly, we demonstrate two methods to prevent availability of the EVSE or CSMS. One of these, originally reported by SaiFlow, prevents traffic to legitimate EVSE equipment using a DoS-like attack on CSMSs by repeatedly connecting and authenticating several CPs with the same identities as the legitimate CP. These vulnerabilities were demonstrated with proof-of-concept exploits in a virtualized Cyber Range at Wright State University and/or with a 350 kW Direct Current Fast Charger at Idaho National Laboratory. The team found that OCPP 1.6 could be protected from these attacks by adding secure shell tunnels to the protocol, if upgrading to OCPP 2.0.1 was not an option.**

***Index Terms*—Electric vehicle charging, cybersecurity, OCPP, cyberattack, cyber-resilience**

I. Introduction

The rapid expansion of electric vehicles (EV) globally creates a need for charging stations. In the US, the 2021 Infrastructure Investment and Jobs Act (IIJA) allocated $7.5 billion in EV charging infrastructure which is being rolled out through the National Electric Vehicle Infrastructure, (NEVI) Formula Program [1]. The US Inflation Reduction Act further provides tax credits for the purchase of new and used EVs and California plans to eliminate the sale of new internal combustion engine-powered vehicles by 2035 [2]. In a March 2023 White House press release, the Biden administration reaffirmed its intent to move to 100 percent zero-carbon emissions vehicle (ZEV) light-vehicle fleet by 2027 and medium-vehicle fleet by 2035 [3]. Continued legislation to promote EVs is a clear indication of political will to rapidly transition to a zero-carbon emission transportation infrastructure. Charging stations in the US are poised to expand to 35 million by 2035 [2]. The swift growth of charging infrastructure connects two sectors not previously connected with fossil-fuel powered engines: transportation and the electrical-grid [4]. This large shift to EVs and associated charging infrastructure creates new cybersecurity threats to the transportation and energy sectors [5].

The attacks described in this research explore Electric Vehicle Supply Equipment (EVSE) vulnerabilities that present threats categorized as consumer, commercial, and strategic risks. In the consumer category, personally identifiable information (PII) and financial information are vulnerable to interception due to machine-in-the-middle (MITM) attacks against the link between the EVSE and Charging System Management Services (CSMS). These attacks can result in credit card fraud and identity theft. Commercial impacts include risks of energy-theft and system availability due to MITM, Log4Shell, and denial-of-service (DoS) attacks [6], [7]. The long term impact is loss in profits and damage to the reputation of a company, both short- and long-term. All of these attacks can have a strategic impact. The consequences of adversaries achieving root access to EVSE and CSMSs can result in disruption to power-grid configurations, delays to federal emergency response, and disruption to EV-powered transportation supply chains of goods and services nation-wide [8].

There is currently no industry requirement for this com-

This research was partly funded by the Department of Energy (DOE) Vehicle Technologies Office (VTO) EVs@SCALE Consortium and the DOE Cybersecurity, Energy Security, and Emergency Response (CESER) "EV Secure Architecture Laboratory Demonstrations (EV SALaD)" project.

2023 Resilience Week (RWS) | 979-8-3503-4747-0/23/$31.00 ©2023 IEEE | DOI: 10.1109/RWS58133.2023.10284654

munications link, but one protocol gaining popularity is the Open Charge Point Protocol (OCPP). OCPP is a global open-source protocol that controls the communication between a Charge Point (CP), which is the EVSE, and a Central System (CS), which is the CSMS [9]. The protocol is supported by the Open Charge Alliance (OCA) with more than 220 member-companies active in the electric mobility area [10]. Currently OCPP is being used in 148 countries and has over 65,000 installed and operating charging stations [2]. OCPP has two primary versions in use, i.e., OCPP 1.6 and 2.0.1, which communicate using either SOAP (Simple Object Access Protocol) or JSON (JavaScript Object Notation). The JSON version is indicated with a “j” in the version string, e.g., OCPP 1.6j, and the SOAP version is indicated with an “s” in the version. While 2.0.1 is gaining ground, OCPP 1.6j is still the most widely used version.

Vulnerabilities for OCPP 1.6 are continuously examined in the current literature [11]. Friedland discussed issues in the event of an adversary gaining OCPP 1.6 network access [12]. Alcaraz et al. [13] and Rubio et al. [14] discussed MITM vulnerabilities in OCPP, which would allow fraudulent charging or disruption of power system operations at the aggregate level. SaiFlow found weak authentication policies with OCPP and connections between CPs and CSs could be disrupted by falsifying additional connections to the CS [15].

In this work, we show several OCPP 1.6j proof-of-concept exploits in a virtual Cyber Range and on a 350 kW Direct Current Fast Charger (DCFC) EVSE to gain root access to the EVSE, lock out legitimate users, and terminate charging sessions. Then, we demonstrate a method of preventing these attacks by protecting OCPP 1.6j communication channels using Secure Shell Tunnels (SSH) tunnels. The remainder of the paper is structured as follows: Section 2 describes the test environments; Section 3 covers the development of and results from exploits used within the OCPP Cyber Range and 350 kW EVSE at Idaho National Laboratory (INL); Section 4 presents the results; and Section 5 concludes the paper.

II. Testing Environments

A. OCPP Cyber Range

The OCPP Cyber Range is a Python-based client-server environment built from the MobilityHouse code base [16]. Both the OCPP client and server programs are operating system (OS) agnostic. For this research, the environment was deployed on multiple VirtualBox Ubuntu virtual machines (VMs) that represented the CP and CS server, along with a Kali Linux VM that represented an adversary on the same network, as shown in Figure 1.

The OCPP Cyber Range configuration consists of four main components that can be divided into two main subsections, the OCPP client and OCPP server. In the current system configuration the OCPP client’s IP address is 192.168.50.220 and the OCPP server’s IP address is 192.168.50.151. The client and server each have two parts to their communications process, the websocket, enabling the communication between the OCPP client and OCPP server, and an asynchronous

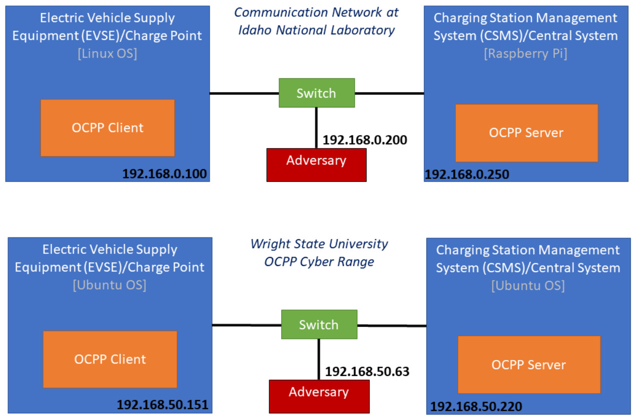

Electric Vehicle Supply Equipment (EVSE)/Charge Point
[Linux OS]
OCPP Client
192.168.0.100
Communication Network at Idaho National Laboratory
Switch
192.168.0.200
Adversary
Charging Station Management System (CSMS)/Central System
[Raspberry Pi]
OCPP Server
192.168.0.250
Electric Vehicle Supply Equipment (EVSE)/Charge Point
[Ubuntu OS]
OCPP Client
192.168.50.151
Wright State University OCPP Cyber Range
Switch
192.168.50.63
Adversary
Charging Station Management System (CSMS)/Central System
[Ubuntu OS]
OCPP Server
192.168.50.220

Fig. 1. The OCPP communication networks use within the Wright State University OCPP Cyber Range and at located at INL.

HTTP server, i.e., aiohttp server [17], [18], used to handle asynchronous communication between an external system operation and the OCPP entities, i.e., client and server requests between servers and communication between the external operator and the OCPP client or server. The aiohttp server enables this by creating a single object that can be re-used for multiple individual requests and by default can make connections with up to 100 different servers at a time. Each system (OCPP client and server) is hosted in a separate Ubuntu VM. The adversary machine is a Kali Linux VM hosted on the same laptop as the OCPP client and server with an IP address of 192.168.50.63.

In the interest of testing the Log4Shell attack several additions were added to the base system in order to provide the infrastructure required for the attack. The Log4j version 2.14 [19] was used in the client’s logging framework in order to enable the Log4Shell attack. On the adversary machine, Light-weight Directory Access Protocol (LDAP), HTTP, and File Transfer Protocol (FTP) servers were added.

B. EV Charging Laboratory at INL

The testbed at INL Electric Vehicle Infrastructure Laboratory (EVIL) includes a 350 kW DCFC EVSE connected to a Java-based SteVe OCPP 1.6j CSMS server [20] running on a Raspberry Pi 3B. The EVSE included one Combo Charging System (CCS) [21] and one CHAdeMO charge port [22], RFID card swipe, and a front panel display for EV drivers to interact with the charger. It was possible to configure a URL in this EVSE to connect to an OCPP server, which in this case was the SteVe server located on the same local subnet as the simulated adversary machine. This communication network configuration is also shown in Figure 1.

III. Exploit Designs and Results

A total of five proof-of-concept (PoC) exploits were created and tested in the OCPP environments. The environments where each PoC was executed is shown in Table I. Two types of attacks were tested utilizing the *OCPP Cyber Range*:

- PoC #1: MITM used to modify firmware update process in order to exploit Log4Shell to gain root access to the EVSE
- PoC #4: Denial-of-service (DoS) vulnerability on the CS

Another set of PoC exploits were demonstrated on the EVSE at *INL*:

- PoC #2: MITM used to terminate the charging session
- PoC #3: A malicious firmware update with code injection to add a new user
- PoC #5: The SaiFlow Denial-of-service (DoS) vulnerability where several falsified CPs connect to a CS to prevent legitimate communications

A. PoC #1: MITM Malicious Firmware Update with Log4Shell

As discussed by TrendMicro [23], there is a risk of Log4Shell (CVE-2021-44228 [24]) exploits reaching Java software running in either the electric vehicle or the EVSE. The risk is that remote code execution from this deserialization attack would allow an adversary to install new accounts, SSH keys, or establish a remote, persistent connection with the EVSE equipment. To demonstrate this vulnerability, a simple Java program processed the *Update Firmware* payload in the OCPP Cyber Range to allow superuser remote code execution on the EVSE.

The Log4Shell vulnerability is caused by a flaw in the way information is retrieved by the Java Naming and Directory Interface (JNDI) API and logged by the Log4j library. Instead of information being logged, the system takes the retrieved information and attempts to execute it as a command. This creates a situation where attackers can cause remote code execution (RCE) or escalation or privileges by forcing the target machine to receive messages from a malicious LDAP server controlled by the attacker. JNDI is a an Application Programming Interface (API) of the Log4j library, which was added in 2013 and its purpose is to assist with data storage and retrieval [25] in terms of producing the naming and directory functionality to software written in Java. It is noted that in order to use JNDI the system must have at least one service provider such as LDAP.

As far as the LDAP is concerned, it is an open, vendor neutral application for retrieving data such as printer connections, user names, passwords, or other static data within directories [26]. The LDAP server allows for multiple databases to be stored in one location and gives users the ability to find information about organizations, persons, etc.

For this attack, a MITM technique was used where the attacker positioned themselves between two legitimate parties in order to hijack the communication link between the OCPP entities [27]. First, we assume that the attacker has achieved access to the local network that the OCPP client and the OCPP server belong to. Afterwards, an ARP (Address Resolution Protocol) spoofing (or ARP cache poisoning) attack is performed in order to intercept and modify network traffic between the client and the server [28]. The attack involves sending falsified ARP messages to the local network. By doing so, the attacker associates their own MAC address with the IP address of the OCPP client's device, causing the OCPP server to send traffic meant for the OCPP client to the attacker's machine instead. Therefore, the attacker is able to eavesdrop on the communication, inject malicious packets, and/or modify the content of the communication. Last, there is a Log4j-based vulnerable Java application processing the OCPP websocket payloads on the EVSE.

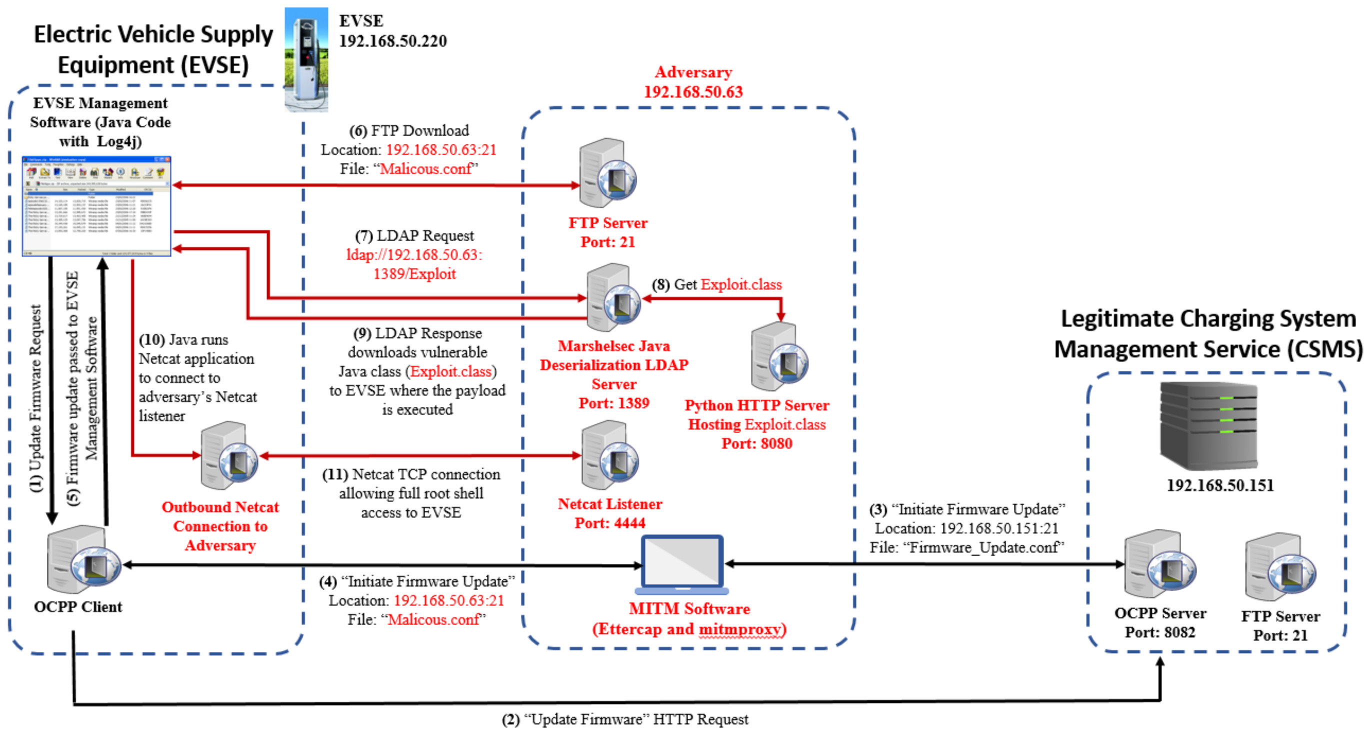
TABLE I
POC EXPLOIT DEMONSTRATION LOCATIONS

PoC	*Cyber Range*	*350 kW EVSE*
#1 MITM Log4Shell Firmware Update	x	
#2 MITM Stop Charging Session		x
#3 Code Injection Firmware Update		x
#4 DoS the CSMS	x	
#5 DoS with Unauthenticated CPs		x

Figure 2 depicts the overall sequence for the Log4Shell attack. At a high level, there is a "Firmware Update" request from the OCPP client to the server. The server response is intercepted and a new malicious file is downloaded to the EVSE. This file includes the LDAP request that pulls down a malicious Java class, Exploit.class, from the adversary that establishes a Netcat connection to the adversary.

Walking through this process in more detail, here are the individual steps:

1) The Java-based EVSE Management Software sends a request to the OCPP client to "request firmware update" from the OCPP server.
2) The OCPP client sends an HTTP request to the OCPP server.
3) The OCPP server sends the FTP location and file back to the EVSE through a websocket connection. In this case, the location to retrieve the firmware update is 192.168.50.151 and the filename is *firmwareUpdate.conf*.
4) The firmware update information is intercepted by the "mitmdump" [29] on the attack machine, which is monitoring the websocket connection between the OCPP server and the OCPP client. Using a websocket injection script, "mitmdump" scans each packets for the *"UpdateFirmware"* command. When it finds this packet it modifies the location and filename of the update file prior to relaying the message to the OCPP client. These modifications change the IP address of the firmware update file to the malicious FTP server on the attacker's machine (IP address: 192.160.50.63) and the filename to the name of the malicious file (malicious.conf) on the attacker's FTP server.
5) The OCPP client receives the "initiate firmware update" command and sends it back to the Java management software for processing.
6) The Java code sends a request to the FTP server to retrieve the "firmware update" file. Utilizing the maliciously changed address and file name, received in the previous step, the Log4j extension receives the "malicious.conf" file from the attacker's



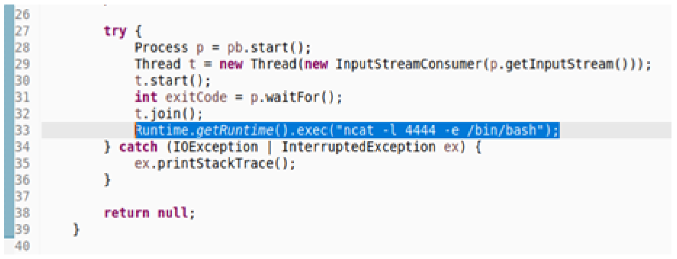
Electric Vehicle Supply Equipment (EVSE)
EVSE 192.168.50.220
EVSE Management Software (Java Code with Log4j)
Adversary 192.168.50.63
(6) FTP Download Location: 192.168.50.63:21 File: "Malicous.conf"
FTP Server Port: 21
(7) LDAP Request ldap://192.168.50.63:1389/Exploit
(8) Get Exploit.class
Marshelsec Java Deserialization LDAP Server Port: 1389
Python HTTP Server Hosting Exploit.class Port: 8080
(9) LDAP Response downloads vulnerable Java class (Exploit.class) to EVSE where the payload is executed
(10) Java runs Netcat application to connect to adversary's Netcat listener
(1) Update Firmware Request
(5) Firmware update passed to EVSE Management Software
Outbound Netcat Connection to Adversary
(11) Netcat TCP connection allowing full root shell access to EVSE
Netcat Listener Port: 4444
OCPP Client
(4) "Initiate Firmware Update" Location: 192.168.50.63:21 File: "Malicous.conf"
MITM Software (Ettercap and mitmproxy)
Legitimate Charging System Management Service (CSMS)
192.168.50.151
(3) "Initiate Firmware Update" Location: 192.168.50.151:21 File: "Firmware_Update.conf"
OCPP Server Port: 8082
FTP Server Port: 21
(2) "Update Firmware" HTTP Request

Fig. 2. Sequence diagram of the Log4Shell attack on the OCPP session using the "Update Firmware" request

FTP server. The "malicious.conf" file consists of a single line of code {*"FirmwarePath": "$ {jndi:ldap://192.168.50.63:1389/Exploit"*} {*"RetrieveDate": "19 April 2023"*}. The first part of the code tells the Log4j extension to utilize the JNDI API to contact the LDAP server. The next part of the code is the IP address and port number to the attacker's LDAP server and finally the malicious Java class, i.e., Exploit.class, and date to retrieve the file.

7) Log4j in the EVSE's Java code processes the *malicious.conf* which causes Log4j to contact the adversary's LDAP server and request the *Exploit.class* per the described Log4shell vulnerability.
8) The LDAP server receives the request for *Exploit.class* and it then forwards that request to the adversary's HTTP server which returns *Exploit.class* with the the Netcat payload. The payload is shown in Figure 3.
9) *Exploit.class* is downloaded by the Java tool and executed.
10) A Netcat session from the EVSE is initiated with the adversary, who had previously created a Netcat listener.
11) The adversary has now persistent root shell access on the EVSE equipment.

The impact of this PoC exploit occurs when processing the "Exploit.class" by Log4j. Rather than simply logging the information (e.g., the "Exploit.class" name), Log4j facilitates the processing of the malicious code, including the "Exploit.class" file, as an executable command, leading to its unintended execution (see Step 11 in Fig. 2). When the "Exploit.class" is executed on the EVSE, it starts a Netcat utility on the OCPP client machine with root access to the system. Once obtaining access to the operating system, the adversary can examine all running processes, locate credentials, establish alternative persistence mechanisms, install kernel-mode rootkits, and search for any upstream connections to the EVSE vendors cloud services – potentially allowing the adversary to pivot to additional EVSE equipment.



```
        try {
            Process p = pb.start();
            Thread t = new Thread(new InputStreamConsumer(p.getInputStream()));
            t.start();
            int exitCode = p.waitFor();
            t.join();
            Runtime.getRuntime().exec("ncat -l 4444 -e /bin/bash");
        } catch (IOException | InterruptedException ex) {
            ex.printStackTrace();
        }

        return null;
    }
```

Fig. 3. Code snippet in the *Exploit.class* that opens a bash terminal and starts the Netcat utility

This attack will only work for EVSE devices that have vulnerable Java-based management code, but it shows the risk of having unencrypted OCPP communications. Any of the data running between the OCPP client and server could be modified to trigger these types of sophisticated cyber kill chains.

B. PoC #2: MITM Stop Charging Session

This MITM attack was performed on the websocket channel between the EVSE and CSMS server at INL. An adversary on a Linux machine was on the same subnet as these devices. To carry out this PoC, the adversary first used the Ettercap tool [30] to perform an Address Resolution Protocol (ARP) spoofing attack on the websocket communication channel between the EVSE and the CSMS server. Specifically, the adversary sends fake ARP messages to associate his MAC address with the IP address of the EVSE, redirecting all traffic to his machine. This allows the adversary to intercept and read all messages exchanged between the EVSE and the CSMS server on the fly.

Once the ARP spoofing is successful, the adversary sets up port forwarding from the Ettercap listening port to a mitmproxy [29] port, where the mitmproxy is an open-source HTTP(S) proxy used to inspect, decode, modify, and replay websocket traffic. The mitmproxy server operates as a reverse proxy between the DCFC and CSMS server. The adversary can then use the mitmdump tool command, similarly to the previous PoC, to intercept and manipulate websocket messages between them in real-time.

To demonstrate the PoC, the adversary utilizes a Python live script running on the mitmproxy server that injects packets in an automated manner. The live script waits for and captures the *StartTransaction* message from the EVSE to the CSMS server, which indicates the start of a charging session and the associated transaction ID. Then, the adversary maliciously transforms a subsequent websocket message, such as a MeterValues message, to a *RemoteStopTransaction* message. Upon successful transmission of the *RemoteStopTransaction* websocket message to the targeted OCPP client, the charging session is terminated. This termination can be sent at a time of the adversary's choosing.

Notably, with this level of access the adversary could also change all other parameters being exchanged between the EVSE and CSMS. For instance, meter data could be modified to defraud the CSMS company. The EVSE availability could be falsified to indicate an inoperative charger as a means to redirect potential customers and financially impact the CSMS company. The charging profile could be modified to reduce the EV charge rate, potentially disrupting the transportation plans of the customer. Reservations could be modified or the EV user could have their authorization status blocked using a *SendLocalListRequest* message. This is just a start of potential impacts from a compromised OCPP session. Several other adversarial objectives could be achieved with full access to the OCPP exchanges.

The MITM attack resulted in the immediate termination of the charging session between the DCFC and EV. The DCFC trusts the websocket RemoteStopTransaction message that appears to originate from the CSMS server. The successful execution of the RemoteStopTransaction highlights the critical importance of integrating robust cybersecurity measures into OCPP 1.6j, in order to effectively mitigate the dangers associated with the MITM attacks.

C. PoC #3: Malicious Firmware Update with Code Injection

OCPP provides several commands that are issued from the CSMS (SteVe) to configure and control the connected EVSE. One of those commands is an *UpdateFirmware* request. This message is usually sent to an EVSE to notify the station that an updated software package is available for download and the location of the firmware with an FTP Uniform Resource Identifier. The update is then downloaded via FTP and installed by the EVSE. The implementation and security of the software update process is directly handled by the vendor of the EVSE, which means that the OCPP protocol does not define how the update is processed by the EVSE. In the case of the 350 kW DCFC EVSE, there was a cybersecurity vulnerability in the handling of the firmware update package. The EVSE did not properly verify firmware updates before execution and there was no sanitation of untrusted firmware data. Therefore, once an adversary gained access to the OCPP network or could issue a Firmware Update Request from the CSMS, the EVSE would download this package and, if packaged correctly, would execute arbitrary code contained in this file. This is a form of code injection via firmware file that enabled remote arbitrary code execution. From this point, the adversary could establish remote shells as described in PoC #1, create new users, or disrupt the operations of the EVSE with a shutdown or other command. In the case of experiments at INL, the team created a firmware update to generate a new user and then connected directly to the equipment via SSH.

This vulnerability was reported to the EVSE vendor and it has since been patched with a firmware update.

D. PoC #4: Denial-of-Service (DoS) on the CSMS Server

DoS attacks are caused by the flooding of traffic to the target machine or network to such a degree that legitimate traffic is impeded causing the target network or system access and usage to be degraded or denied [31]. In this case, the adversary performed a SYN flood attack [32], which exploits the three-way handshake process used in the Transmission Control Protocol (TCP) to establish a connection between the EVSE and CSMS server in the Cyber Range. During the handshake process, the EVSE sends a SYN packet to initiate the connection, the CSMS server responds with a SYN-ACK packet to acknowledge the request, and the client sends an ACK packet to confirm the connection.

Our simulated adversary quickly sent thousands of SYN packets to the CSMS server without responding to the latter's SYN-ACK packets, effectively leaving half-open connections. The CSMS server keeps these connections open in its memory until the connection is established or times out, which causes the server to become overwhelmed and eventually stop responding to legitimate requests from the EVSE.

The SYN DoS attack to the CSMS server was performed using the *hping3* software [33]. It produced heavy resource utilization on the CSMS server. In particular, the target CSMS server was a VM equipped with a dual core 2.3 GHz processor

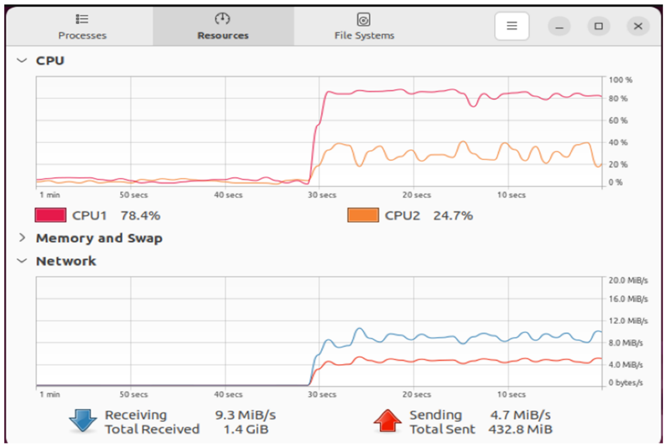


Fig. 4. Resource utilization during DoS attack

and 2 GB of RAM, whereas the attack machine was a 2.3 GHz dual core processor with 8 GB of RAM. Resource monitoring on the target machine identified resource utilization before, during, and after the DoS attack (see Fig.4). We observed the CPU utilization loads of the CSMS server CPU and network traffic to determine if the system was vulnerable to a DoS attack. During the attack a single OCPP client was connected to the CSMS server. We observed initially the utilization in both CPUs is approximately 15%-20% and the network traffic never exceeds 15 kB/s.

At the start of the SYN DoS attack the CSMS server's *CPU 1* utilization increased to 78%-85%, *CPU 2* utilization increases to 20%-40%, and network traffic flow is increased to approximately 10 MiB/s. Figure 4 depicts the spike in resource utilization during the attack. When the attack was terminated all resource utilization levels returned to normal operating ranges.

This DoS PoC is not enough to crash the server – however a noticeable degradation in server response was noticed. Important to note, the attack machine is a VM that is hosted on the same laptop as the server VM. The increase in resource utilization demonstrates that switching to a distributed denial-of-service attack would crash the server or deny legitimate traffic. A good mitigation to this DoS attack would be to create firewall rules that reject unknown connections to the CSMS.

E. PoC #5: SaiFlow Denial-of-Service (DoS)

OCPP suffers from another vulnerability that exposes the system to a form of DoS attacks [15]. The weakness stems from the fact that the standard does not specify how to handle multiple websocket connections from a single CP to the CS/CSMS. Adversaries can exploit this ambiguity by establishing additional "new" connections to the CSMS on behalf of the CP, resulting in potential DoS attacks that could compromise the integrity of the EVSE network. According to Saposnik and Porat, the response of the CSMS depends on the manufacturer: some close the original session while others direct their traffic to the CP that most recently authenticated/communicated with the server.

This vulnerability affects OCPP 1.6 and could also impact the newer OCPP 2.0.1 version if the necessary authentication measures are not implemented. Because the CP requires a continuous connection with the CSMS for charging authorizations, charging sessions, payments, and others, a malicious actor can disrupt this connection and lead to rejected charging transactions. This situation can result in severe financial and reputation damages, as well as the dangerous exposure of the CSMS infrastructure.

Our research team designed an experiment to test this vulnerability in the OCPP standard for managing EVSE. Specifically, the experiment involved mimicking the CP already connected to the CSMS server and creating a new Python-based CP instance connected to the same server. The server's insufficient OCPP 1.6j authorization measures allowed the incoming websocket connection from the new CP instance, leading the CSMS server to assume this new session was the legitimate CP. This design enabled us to bypass the original connection and establish a new communication channel, resulting in a potential DoS attack.

To test the vulnerability, we created and connected an identical CP instance to the SteVe CSMS server during a charging session between the original CP and CSMS server. We observed that even if the new CP instance was a simple Python-based script mimicking the original CP based on the MobilityHouse client [16], the new websocket communication channel was successfully created because of the insufficient authorization measures of the OCPP 1.6j protocol. To simulate a DoS attack, we configured the new "fake" CP to send heartbeats to the CSMS server every millisecond. During testing in a real environment, we observed that the CSMS server was "confused" by the new CP instance and considered it a new safe connection originating from the 350 kW charger. As a result, the CSMS server kept both connections open but accepted the flooding heartbeat messages from the new CP instance.

Consequently, it rejected all the messages/requests that came from the original CP, e.g., *MeterValues* messages from a charging session with an EV. Furthermore, another important observation is that when the CSMS server sent a *RemoteStop-Transaction* request to the original CP, this message was never received by the latter because of the DoS attack and the charging session did not stop. Only when we disabled the DoS attack and disconnected the new CP, was the server able to communicate with the original CP normally again and remotely stop the charging session. The above results reveal the urgent need for identifying countermeasures for the mitigation of the multiple connections handling DoS vulnerability in the OCPP protocol.

IV. Mitigation

In the situation where the EVSE equipment or CSMS server does not support OCPP 2.0.1, there are still options for preventing MITM attacks on the OCPP network. One

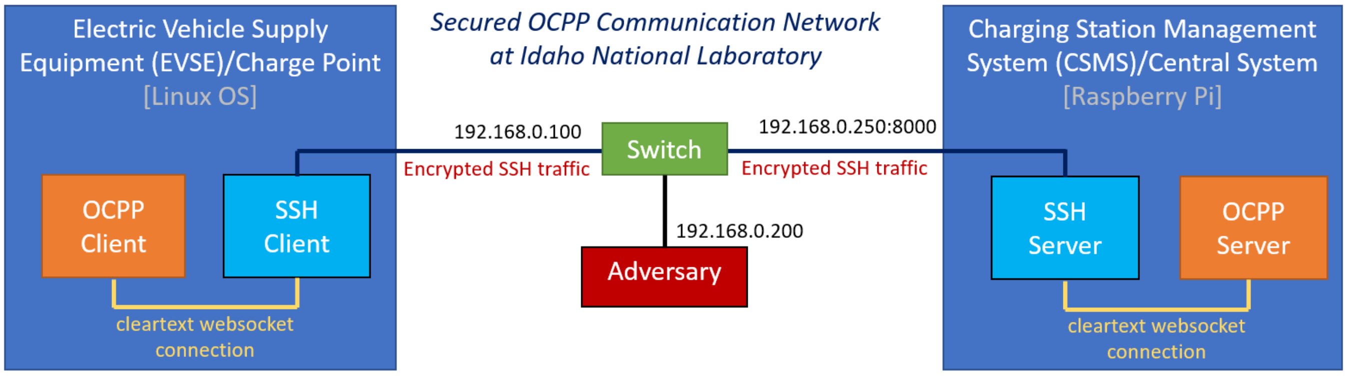

Secured OCPP Communication Network
at Idaho National Laboratory
Electric Vehicle Supply Equipment (EVSE)/Charge Point
[Linux OS]
OCPP Client
SSH Client
cleartext websocket connection
192.168.0.100
Encrypted SSH traffic
Switch
192.168.0.250:8000
Encrypted SSH traffic
192.168.0.200
Adversary
Charging Station Management System (CSMS)/Central System
[Raspberry Pi]
SSH Server
OCPP Server
cleartext websocket connection

Fig. 5. Routing OCPP 1.6 through SSH session.

way this can be done is using SSH tunneling. This was implemented in an INL laboratory to successfully prevent the MITM PoCs presented above. To implement the solution, SSH public-private keys were generated on the EVSE and CSMS. The public keys were saved when the SSH tunnel was first connected and the keys were authenticated to the saved versions when establishing a new connection. This provided mutual authentication of both the EVSE and CSMS. Once authenticated, the hosts generate and exchange a session key that is used for future encrypted traffic. After initial tunnel setup, the EVSE connected to a locally-bound TCP port that automatically encapsulates the traffic and transmits it over the SSH tunnel. The CSMS was set up to regenerate the SSH-encapsulated traffic on its local interface. This encrypts the traffic and provides confidentiality and integrity of the encapsulated payload, effectively preventing MITM attacks as well as payload inspection. A representation of this configuration is shown in Figure 5.

The additional authentication provided by the SSH tunnel also eliminates the risk of the SaiFlow DoS attack because the adversary does not have the CP/EVSE SSH private key. Without this cryptographic material, the CSMS knows the new CP attempting to connect is not legitimate. The SYN DoS attack on the other hand is best prevented by creating an allow list at the CSMS firewall to prevent unknown devices from reaching the OCPP CS server.

V. Conclusion

OCPP 1.6 is vulnerable to several cybersecurity attacks because it lacks appropriate confidentiality and CP authentication. In this work, we demonstrate the use of machine-in-the-middle, malicious firmware updates, and DoS PoC exploits to remotely terminate charging sessions, prevent charging visibility and control, and gain remote access to EVSE equipment. In order to prevent these attacks in production environments, OCPP sessions need to be protected with encrypted channels by either upgrading to OCPP 2.0.1 or wrapping the protocol in an encrypted tunnel with SSH, IPSec, or some other technique. One mechanism for directing OCPP traffic through an encrypted SSH tunnel is presented in this paper based on experimental results at INL.

Acknowledgment

Sandia National Laboratories is a multimission laboratory operated by National Technology and Engineering Solutions of Sandia LLC, a wholly owned subsidiary of Honeywell International Inc., for the U.S. Department of Energy's National Nuclear Security Administration.

Appendix A: Nomenclature

Acronym	***Meaning***
ARP	Address Resolution Protocol
CP	Charge Point
CS	Central System
CSMS	Charging System Management Service
DCFC	Direct Current Fast Charger
DoS	Denial-Of-Service
EV	Electric Vehicle
EVSE	Electric Vehicle Supply Equipment
FTP	File Transfer Protocol
HTTP	Hypertext Transfer Protocol
JNDI	Java Naming and Directory Interface
LDAP	Light-weight Directory Access Protocol
MITM	Machine-in-the-middle
OCPP	Open Charge Point Protocol
PoC	Proof-Of-Concept
RCE	Remote Code Execution
SSH	Secure Shell
VM	Virtual Machine

References

[1] "Bipartisan Infrastructure Law - National Electric Vehicle Infrastructure (NEVI) Formula Program Fact Sheet — Federal Highway Administration." [Online]. Available: https://www.fhwa.dot.gov/bipartisan-infrastructure-law/nevi_formula_program.cfm
[2] S. Acharya, Y. Dvorkin, H. Pandžić, and R. Karri, "Cybersecurity of smart electric vehicle charging: A power grid perspective," *IEEE Access*, vol. 8, pp. 214 434–214 453, 2020.

[3] "FACT SHEET: Biden-Harris Administration Announces New Private and Public Sector Investments for Affordable Electric Vehicles — The White House." [Online]. Available: https://www.whitehouse.gov/briefing-room/statements-releases/2023/03/30/fact-sheet-biden-harris-administration-announces-new-private-and-public-sector-investments-for-affordable-electric-vehicles/

[4] S. Lightman and T. Brewer, *Symposium on federally funded research on cybersecurity of electric vehicle supply equipment (evse)*. US Department of Commerce, National Institute of Standards and Technology, 2020.

[5] J. Johnson, B. Anderson, B. Wright, J. Quiroz, T. Berg, R. Graves, J. Daley, K. Phan, M. Kunz, R. Pratt *et al.*, "Cybersecurity for electric vehicle charging infrastructure." Sandia National Lab.(SNL-NM), Albuquerque, NM (United States), Tech. Rep., 2022.

[6] Z. Pourmirza and S. Walker, "Electric vehicle charging station: cyber security challenges and perspective," in *2021 IEEE 9th International Conference on Smart Energy Grid Engineering (SEGE)*. IEEE, 2021, pp. 111–116.

[7] J. Johnson, T. Berg, B. Anderson, and B. Wright, "Review of electric vehicle charger cybersecurity vulnerabilities, potential impacts, and defenses," *Energies*, vol. 15, no. 11, p. 3931, 2022.

[8] Z. Garofalaki, D. Kosmanos, S. Moschoyiannis, D. Kallergis, and C. Douligeris, "Electric vehicle charging: A survey on the security issues and challenges of the open charge point protocol (ocpp)," *IEEE Communications Surveys & Tutorials*, 2022.

[9] T. Chen, X.-P. Zhang, J. Wang, J. Li, C. Wu, M. Hu, and H. Bian, "A review on electric vehicle charging infrastructure development in the uk," *Journal of Modern Power Systems and Clean Energy*, vol. 8, no. 2, pp. 193–205, 2020.

[10] P. B. Andersen, S. H. Toghroljerdi, T. M. Sørensen, B. E. Christensen, J. C. Morell, L. Høj, A. Zecchino, O. J. Olesen, and A. Due, "The Parker Project Final Report Type: Project final report," 1 2019. [Online]. Available: https://parker-project.com/wp-content/uploads/2019/03/Parker_Final-report_v1.1_2019.pdf

[11] A. Sanghvi and T. Markel, "Cybersecurity for electric vehicle fast-charging infrastructure," in *2021 IEEE Transportation Electrification Conference & Expo (ITEC)*. IEEE, 2021, pp. 573–576.

[12] A. Friedland, "Security and privacy in the current e-mobility charging infrastructure," *Proceedings of the DeepSec, Vienna, Austria*, vol. 31, 2016.

[13] 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, 2017.

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

[15] L. R. Saposnik and D. Porat, "Hijacking ev charge points to cause DOS," Feb 2023. [Online]. Available: https://www.saiflow.com/hijacking-chargers-identifier-to-cause-dos/

[16] "mobilityhouse/ocpp: Python implementation of the Open Charge Point Protocol (OCPP)." 2019. [Online]. Available: https://github.com/mobilityhouse/ocpp

[17] C. Hattingh, *Using Asyncio in Python: understanding Python's asynchronous programming features*. " O'Reilly Media, Inc.", 2020.

[18] "Web Server Quickstart — aiohttp 3.8.4 documentation." [Online]. Available: https://docs.aiohttp.org/en/stable/web_quickstart.html#websockets

[19] "Log4j – Apache Log4j™ 2." [Online]. Available: https://logging.apache.org/log4j/2.x/

[20] Steve-Community, "Steve-community/steve: Steve - ocpp server implementation in java." [Online]. Available: https://github.com/steve-community/steve

[21] G. R. C. Mouli, J. Kaptein, P. Bauer, and M. Zeman, "Implementation of dynamic charging and v2g using chademo and ccs/combo dc charging standard," in *2016 IEEE Transportation Electrification Conference and Expo (ITEC)*. IEEE, 2016, pp. 1–6.

[22] "Chademo protocolthe ev fast charging standard that is the same wherever you go." [Online]. Available: https://www.chademo.com/

[23] S. Dudek, "Examining Log4j Vulnerabilities in Connected Cars and Charging Stations," 12 2021. [Online]. Available: https://www.trendmicro.com/en_us/research/21/l/examining-log4j-vulnerabilities-in-connected-cars.html

[24] H. Gupta, A. Chaudhary, and A. Kumar, "Identification and analysis of log4j vulnerability," in *2022 11th International Conference on System Modeling & Advancement in Research Trends (SMART)*. IEEE, 2022, pp. 1580–1583.

[25] O. Tunde-Onadele, Y. Lin, X. Gu, and J. He, "Understanding software security vulnerabilities in cloud server systems," in *2022 IEEE International Conference on Cloud Engineering (IC2E)*. IEEE, 2022, pp. 245–252.

[26] E. Daniel and N. Vasanthi, "Ldap: a lightweight deduplication and auditing protocol for secure data storage in cloud environment," *Cluster Computing*, vol. 22, pp. 1247–1258, 2019.

[27] J. Antoun, M. E. Kabir, B. Moussa, R. Atallah, and C. Assi, "A detailed security assessment of the ev charging ecosystem," *IEEE Network*, vol. 34, no. 3, pp. 200–207, 2020.

[28] S. Hijazi and M. S. Obaidat, "Address resolution protocol spoofing attacks and security approaches: A survey," *Security and Privacy*, vol. 2, no. 1, p. e49, 2019.

[29] "Mitmproxy is a free and open source interactive https proxy." [Online]. Available: https://mitmproxy.org/

[30] [Online]. Available: https://www.ettercap-project.org/

[31] A. Cetinkaya, H. Ishii, and T. Hayakawa, "An overview on denial-of-service attacks in control systems: Attack models and security analyses," *Entropy*, vol. 21, no. 2, p. 210, 2019.

[32] M. Bogdanoski, T. Suminoski, and A. Risteski, "Analysis of the syn flood dos attack," *International Journal of Computer Network and Information Security (IJCNIS)*, vol. 5, no. 8, pp. 1–11, 2013.

[33] D. Adams, Jun 1969. [Online]. Available: https://linuxhint.com/hping3/