

## AcCCS Hands-on Exercise 2 – EVSE and PEV Communications

### Scenario

During this exercise you will use an AcCCS system to connect to an actual EVSE and PEV while simulating the other device. This will allow you to evaluate the capabilities of the EVSE and/or PEV and also examine those systems for potential network issues.

### Objective

This exercise will allow you to use the skills developed in Exercise 1 with the actual EVSE and/or PEV hardware.

### Preparation

Before you work on Exercise 2, ensure you have finished the AcCCS Exercise 1 workbook. You will use the same emulator scripts from Exercise 1 with the actual vehicle and charging station.

## Exercise Setup Steps

**Step 1:** Connect the AcCCS box to the provided 12v power supply. Connect your laptop to the AcCCS box using the provided Ethernet cable. Your exercise setup should look something like the following pictures.

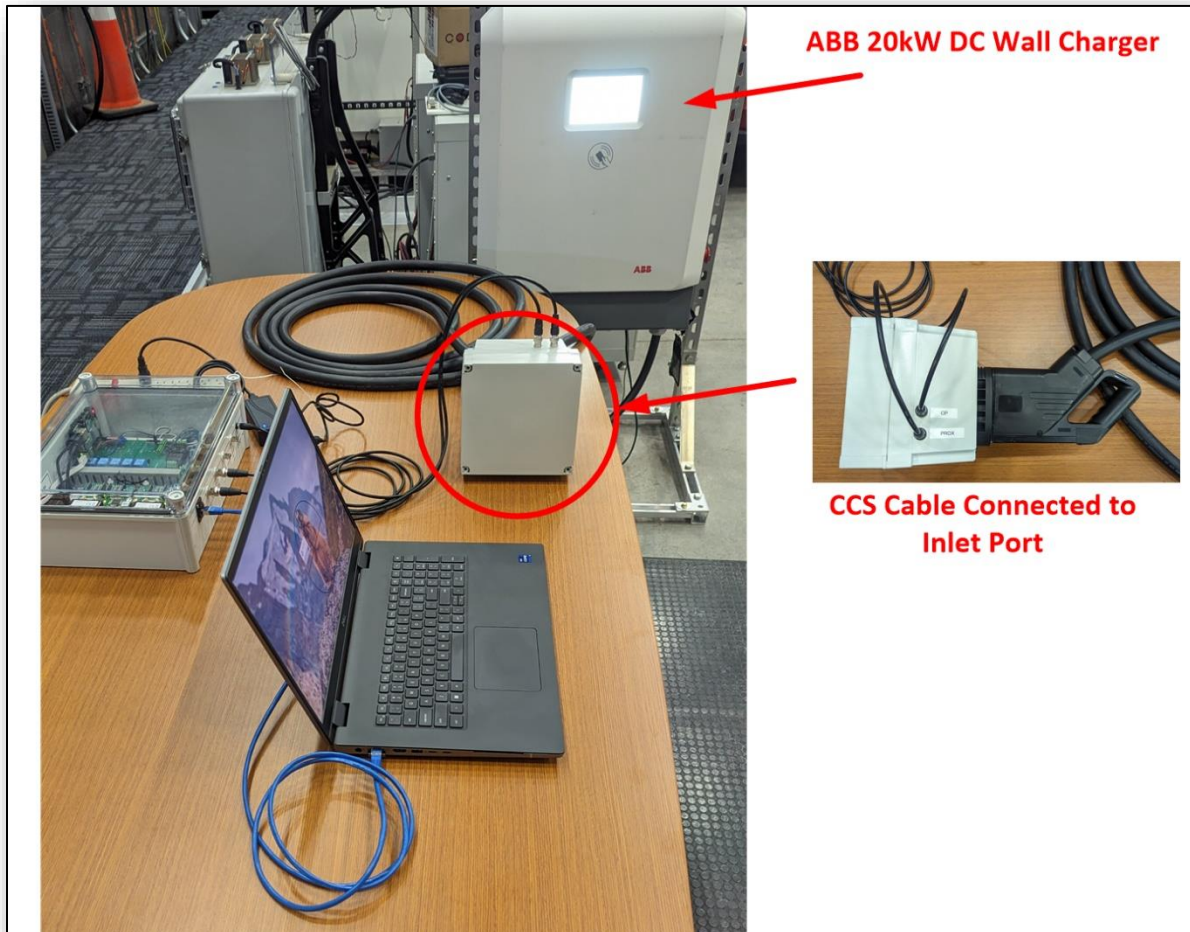


Figure 1: Testing the DC EVSE Wallbox

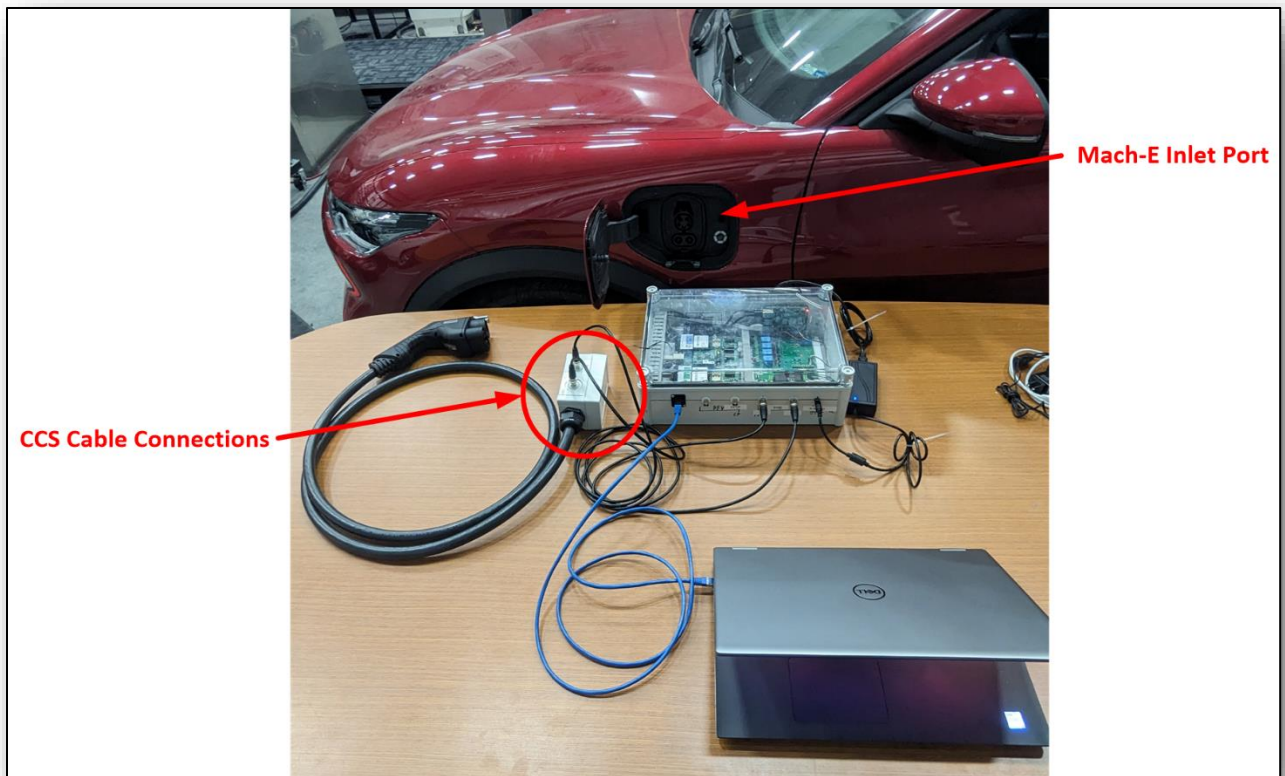


Figure 2: Testing the Mach-E PEV

- Step 2:** Start VMWare Player and power on the provided virtual machine (VM). Log on to the VM using the provided account (user: student, password: password).
- Step 3:** Verify the VM has received an appropriate IP address from the AcCCS box. The AcCCS box provides IPv4 addresses using DHCP, and your address should reside in the 10.10.10.0/24 address space (e.g. 10.10.10.100, 10.10.10.101, etc.). You can do this by starting a terminal window and using the **ifconfig** command or the **ip** command.

```
Terminal - student@laptop: ~
File Edit View Terminal Tabs Help

student@laptop:~$ ifconfig
ens33: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 10.10.10.127 netmask 255.255.255.0 broadcast 10.10.10.255
    inet6 fe80::cbcc:a87e:e043:319a prefixlen 64 scopeid 0x20<link>
    ether 00:0c:29:3c:74:6e txqueuelen 1000 (Ethernet)
    RX packets 25448 bytes 21731341 (21.7 MB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 13663 bytes 2014179 (2.0 MB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
    inet 127.0.0.1 netmask 255.0.0.0
    inet6 ::1 prefixlen 128 scopeid 0x10<host>
    loop txqueuelen 1000 (Local Loopback)
    RX packets 3211 bytes 335009 (335.0 KB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 3211 bytes 335009 (335.0 KB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

student@laptop:~$ ip a list
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
        valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
        valid_lft forever preferred_lft forever
2: ens33: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
    link/ether 00:0c:29:3c:74:6e brd ff:ff:ff:ff:ff:ff
    altname enp2s1
    inet 10.10.10.127/24 brd 10.10.10.255 scope global dynamic noprefixroute ens33
        valid_lft 42938sec preferred_lft 42938sec
    inet6 fe80::cbcc:a87e:e043:319a/64 scope link noprefixroute
        valid_lft forever preferred_lft forever
student@laptop:~$
```

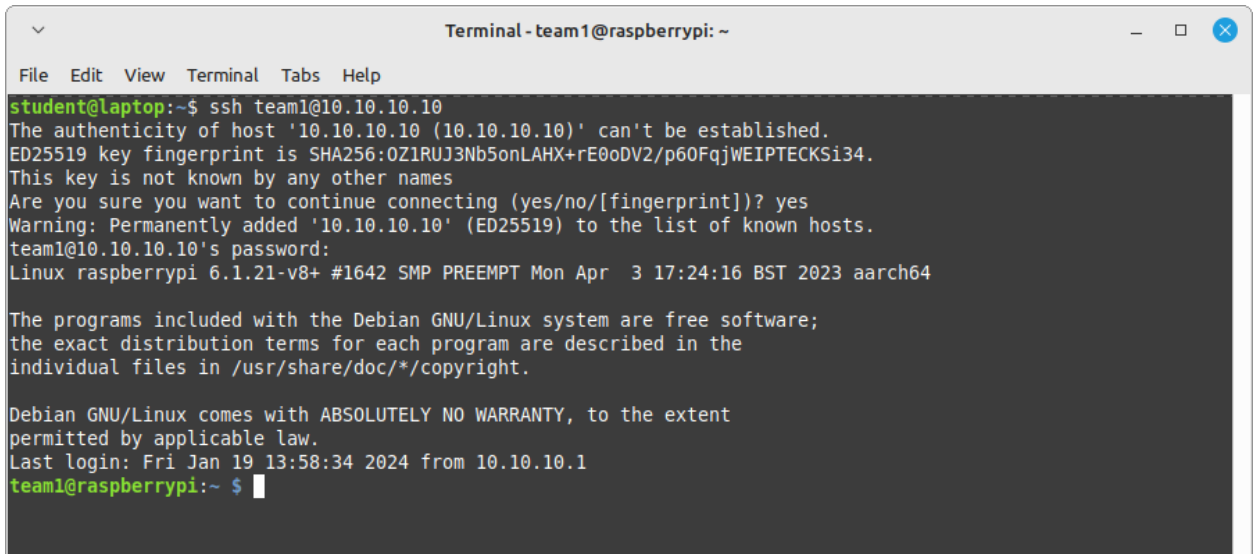
**Step 4:** Ping the Raspberry Pi in the AcCCS box to ensure you have a working network connection.

```
Terminal - student@laptop: ~
File Edit View Terminal Tabs Help

student@laptop:~$ ping 10.10.10.10
PING 10.10.10.10 (10.10.10.10) 56(84) bytes of data.
64 bytes from 10.10.10.10: icmp_seq=1 ttl=64 time=0.618 ms
64 bytes from 10.10.10.10: icmp_seq=2 ttl=64 time=0.611 ms
64 bytes from 10.10.10.10: icmp_seq=3 ttl=64 time=0.719 ms
^C
--- 10.10.10.10 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2042ms
rtt min/avg/max/mdev = 0.611/0.649/0.719/0.049 ms
student@laptop:~$
```

Press Ctrl+C to stop the ping command.

**Step 5:** Connect to the AcCCS Raspberry Pi using the correct account for testing the EVSE or PEV. Make sure you connect using the correct user for the specific exercise you are running (i.e. user: **team2** for the EVSE exercise and user: **team1** for the PEV exercise).

A terminal window titled "Terminal - team1@raspberrypi: ~" is shown. The window has a menu bar with "File", "Edit", "View", "Terminal", "Tabs", and "Help". The terminal output shows a user at a laptop running the command "ssh team1@10.10.10.10". The output includes a warning about the host's authenticity, a confirmation to add the host to the known hosts list, and the password prompt. After the password is entered, the terminal shows the Raspberry Pi's login banner, which includes the system version (6.1.21-v8+), SMP, PREEMPT, and the date and time (Mon Apr 3 17:24:16 BST 2023). The banner also states that the programs are free software and that Debian GNU/Linux comes with absolutely no warranty. The terminal ends with the prompt "team1@raspberrypi:~ \$".

```
student@laptop:~$ ssh team1@10.10.10.10
The authenticity of host '10.10.10.10 (10.10.10.10)' can't be established.
ED25519 key fingerprint is SHA256:0Z1RUJ3Nb5onLAHX+rE0oDV2/p60FqjWEIPTECKS134.
This key is not known by any other names
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added '10.10.10.10' (ED25519) to the list of known hosts.
team1@10.10.10.10's password:
Linux raspberrypi 6.1.21-v8+ #1642 SMP PREEMPT Mon Apr  3 17:24:16 BST 2023 aarch64

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Fri Jan 19 13:58:34 2024 from 10.10.10.1
team1@raspberrypi:~ $
```

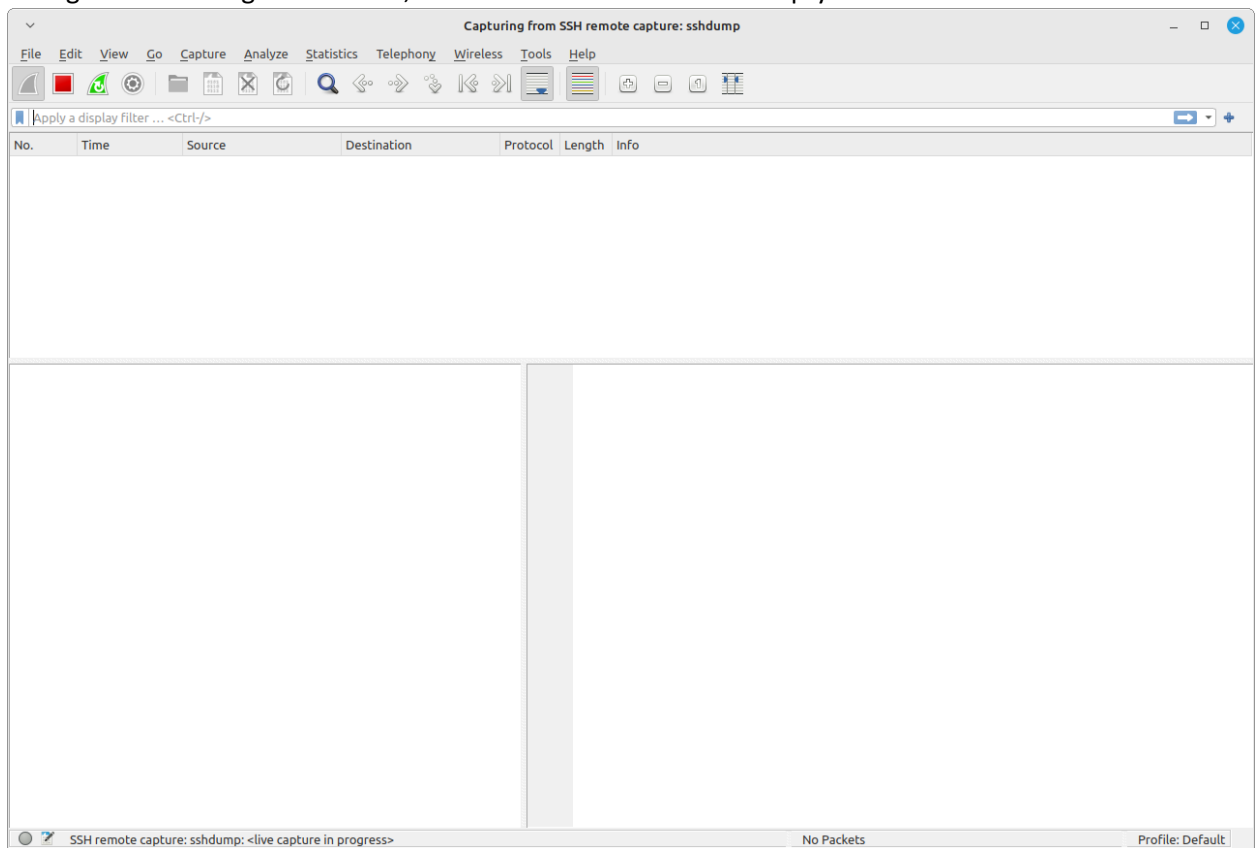
**Note:** If you receive a message asking you to confirm the SSH connection based on the SHA256 fingerprint, type 'yes' in the terminal to continue.

## EVSE Exercise

For this exercise, we will impersonate a PEV and connect to the EVSE. While running the following steps, connect to the AcCCS Raspberry Pi as user **team2**. (user: team2 password: AcCCS).

**Step 1:** Start Wireshark in your VM. We will use Wireshark to remotely monitor the communications from the Raspberry Pi over the CCS cable. Configure the SSH remote capture settings for user **team2** and capture packets on interface **eth2**.

**Step 2:** Start a Wireshark capture by double-clicking the SSH remote capture interface. If your configuration settings are correct, Wireshark will start with an empty screen.



**Step 3:** Start running the PEV emulator.

The PEV emulator script has several command-line options. Below is a description of each of these options:

```
usage: PEV.py [-h] [-M MODE] [-I INTERFACE] [--source-mac SOURCE_MAC] [--source-ip SOURCE_IP]
[--source-port SOURCE_PORT] [-p PROTOCOL] [--nmap-mac NMAP_MAC] [--nmap-ip NMAP_IP]
[--nmap-ports NMAP_PORTS]

PEV emulator for AcCCS

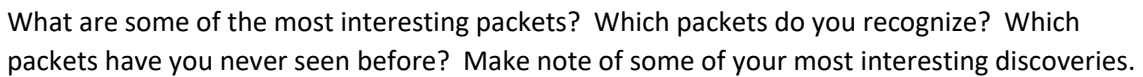
optional arguments:
  -h, --help show this help message and exit
  -M MODE, --mode MODE Mode for emulator to run in: 0 for full conversation, 1 for stalling the
conversation, 2 for portscanning (default: 0)
  -I INTERFACE, --interface INTERFACE Ethernet interface to send/recieve packets on (default: eth1)
  --source-mac SOURCE_MAC Source MAC address of packets (default: 00:1e:c0:f2:6c:a0)
  --source-ip SOURCE_IP Source IP address of packets (default: fe80::21e:c0ff:fef2:72f3)
  --source-port SOURCE_PORT Source port of packets (default: 25565)
  -p PROTOCOL, --protocol PROTOCOL Protocol for EXI encoding/decoding: DIN, ISO-2, ISO-20
(default: DIN)
  --nmap-mac NMAP_MAC The MAC address of the target device to NMAP scan
(default: SECC MAC address)
  --nmap-ip NMAP_IP The IP address of the target device to NMAP scan
(default: SECC IP address)
  --nmap-ports NMAP_PORTS List of ports to scan seperated by commas (ex. 1,2,5-10,19,...)
(default: Top 8000 common ports)
```

We recommend first testing your connection by using “mode 1” (the **-M 1** command-line option). This will establish a connection with the EVSE and keep the conversation running for an indefinite period.

```
sudo python3 PEV.py -I eth2 -M 1
```

**Note:** Running the PEV emulator is possible even without authenticating or authorizing a charge (i.e. there is no need to scan a RFID card or even press any buttons on the EVSE screen)

Monitor Wireshark and take notes about what packets are sent between AcCCS and the EVSE.



---

---

---

---

---

---

---

---

---

---



**Step 4:** Review your captured network traffic from Step 3 and answer the following questions:

What is the Network Membership Key (NMK) used to negotiate and setup the HomePlug Green PHY network? Do you know what this key is used for?

---

---

---

---

What interesting fields are found in the SECC Discovery Protocol Request and Response?

---

---

---

---

What additional protocol(s) are available for EVSE to PEV communications?

---

---

---

---

What message(s) are used to stall the conversation and keep the PEV and EVSE in a waiting state?

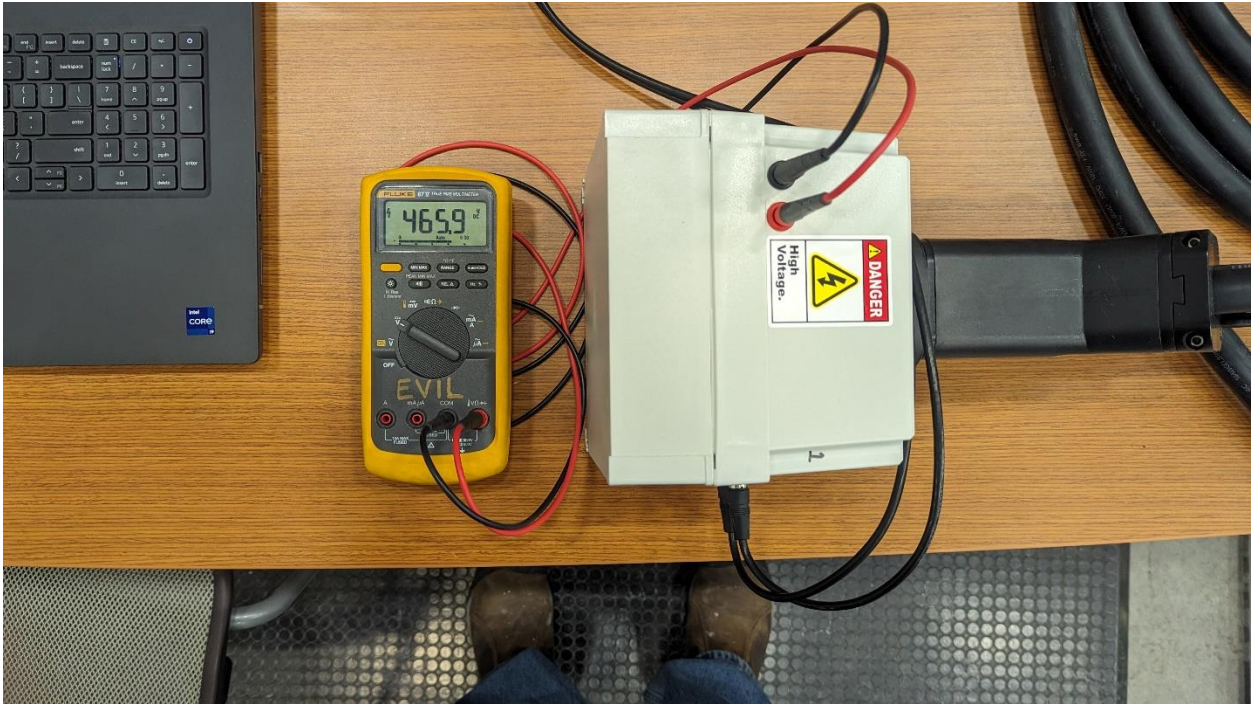
---

---

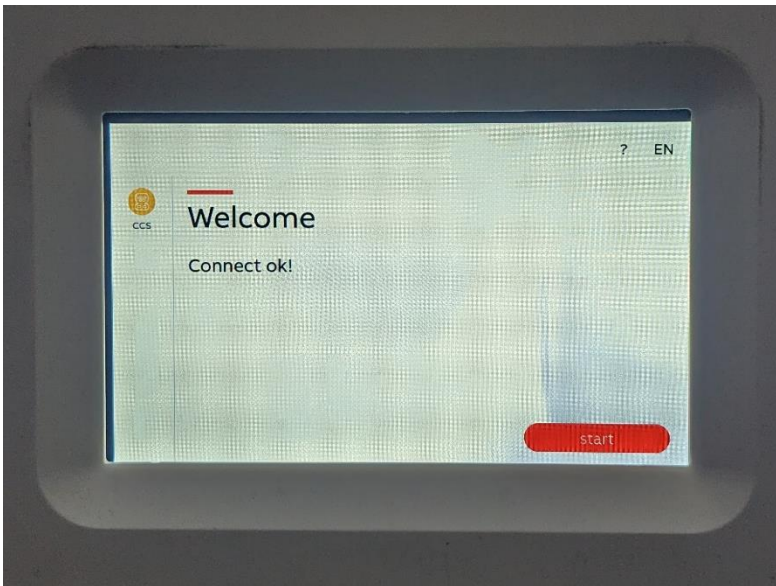
---

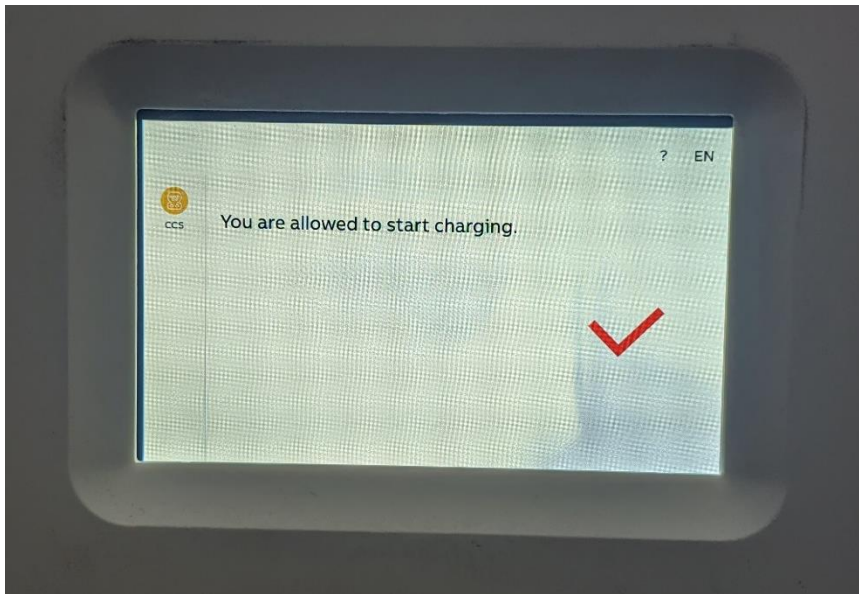
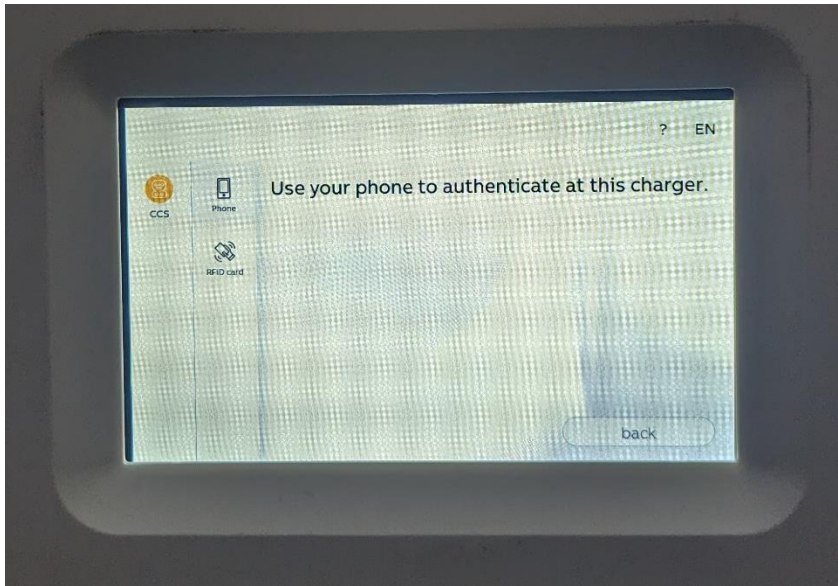
---

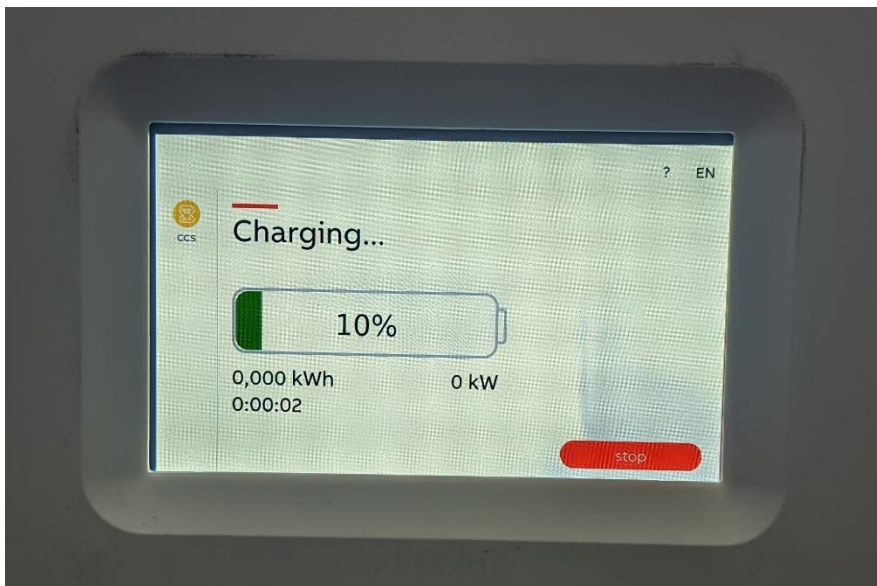
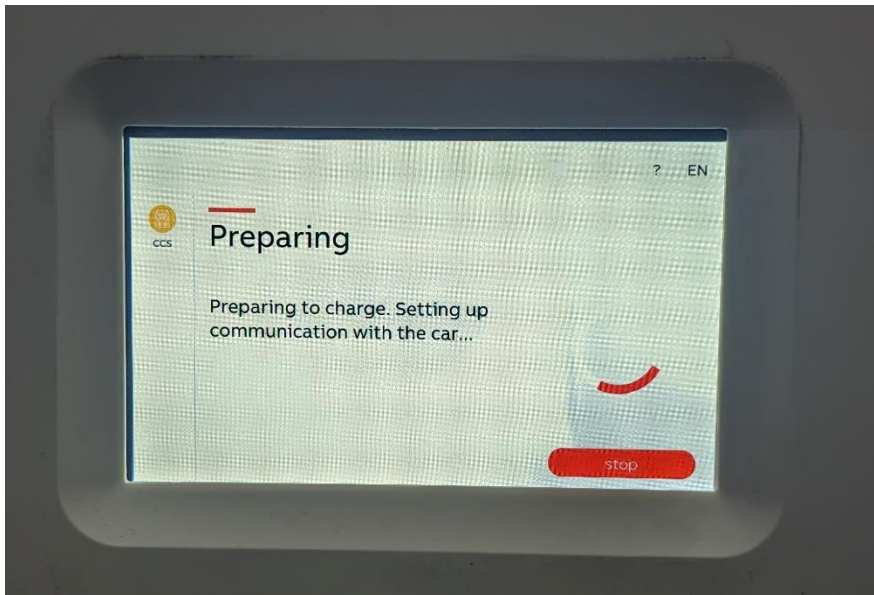
**Step 5:** Stop the PEV emulator script by pressing Ctrl+C. This will interrupt the script and terminate after a few seconds. Connect a digital multi-meter to the inlet port of the AcCCS box as shown in the following picture.



**Step 6:** Restart the PEV emulator script and monitor the traffic in Wireshark. Once the communications are established, press the Start button on the front of the EVSE. When prompted to authenticate with your phone, scan the provided RFID card on the front of the EVSE.







**Step 7:** Monitor Wireshark and look for what messages change after the Start button was pressed. Also watch the multi-meter and see if the EVSE presents DC voltage over the CCS cable. What messages in Wireshark are new? What actions and messages happened to cause this change? What happens to the voltage on the multi-meter after a few minutes? Why?

---

---

---

---

---

What messages are used to control the State of Charge (SOC) value (10%) that is displayed on the screen? What other parameters are found in these messages?

---

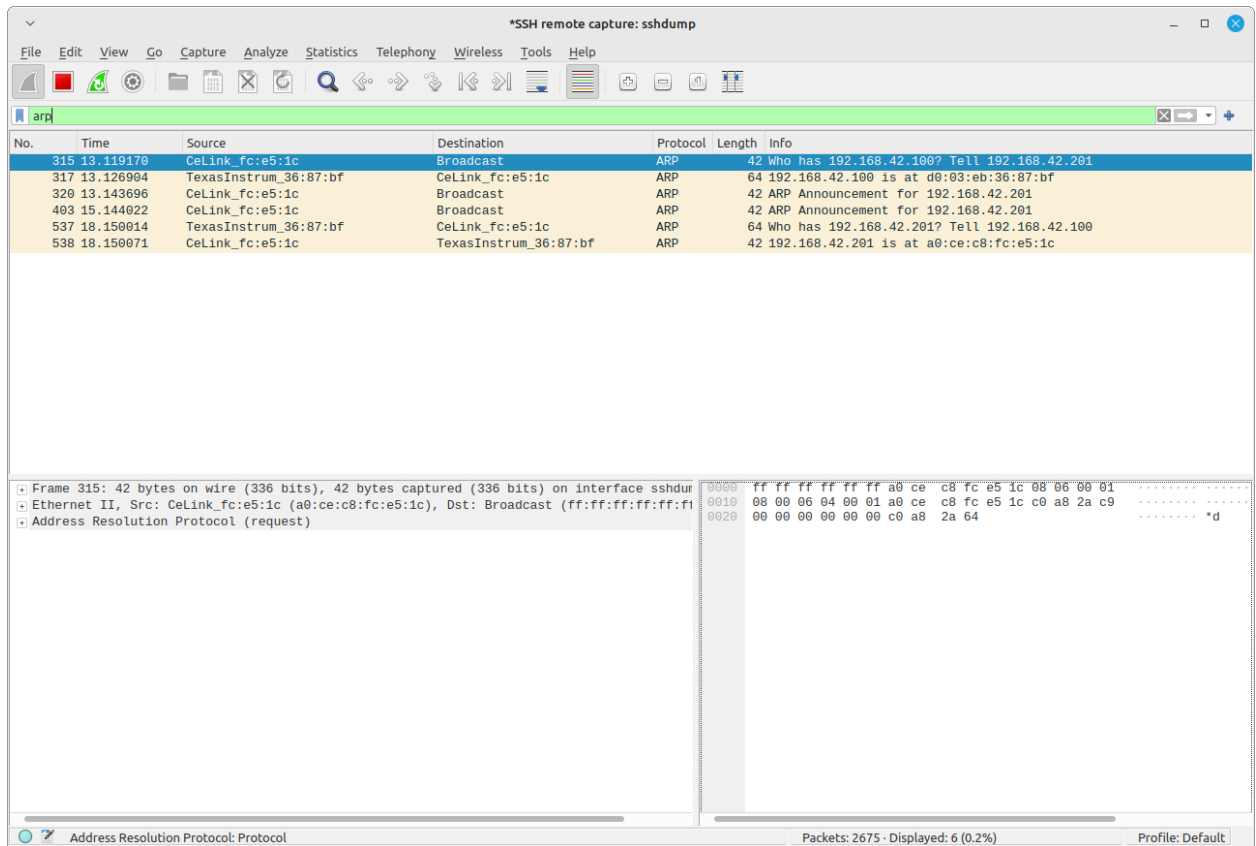
---

---

---

---

**Step 8:** Add a display filter in Wireshark for ARP. You should find a few interesting broadcast packets.



What do you think is happening here? What are these additional IP addresses? Write down the IP addresses and the MAC addresses you located and what you think they might be used for.

---

---

---

---

---



**Step 9:** This EVSE has some interesting network issues. The CCS specification clearly states that only IPv6 addresses are used for communications, and yet you are able to see broadcast IPv4 ARP requests. This is not normal behavior and this is not coming from either of the CCS endpoints (i.e. another device in the EVSE is trying to communicate).

Let's connect again to the EVSE and run the PEV emulator in mode 2 and port scan the EVSE. This port scan will be done using the IPv6 addresses used for normal CCS communications.

```
sudo python3 PEV.py -I eth2 -M 2
```

When the scan is finished, review the results found in the **scan\_results** folder. What open ports were identified? Is this what you expect? What additional steps might you take to investigate this EVSE even further?

---

---

---

---

---

#### Optional Steps (Extra Credit)

**Step 10:** We will now investigate the EVSE even more by trying to determine what is generating the IPv4 ARP broadcast packets. To do this we will establish the CCS communications in mode 1, and then scan the remote IPv4 network using Nmap. Start the PEV emulator again using mode 1.

```
sudo python3 PEV.py -I eth2 -M 1
```

Monitor Wireshark and ensure the CCS communications are running as you expect.

**Step 11:** Open a second terminal in the VM and SSH into the Raspberry Pi. We will use the second terminal window to run the following steps while leaving the PEV emulator running in terminal 1. Log on as user **team2**.

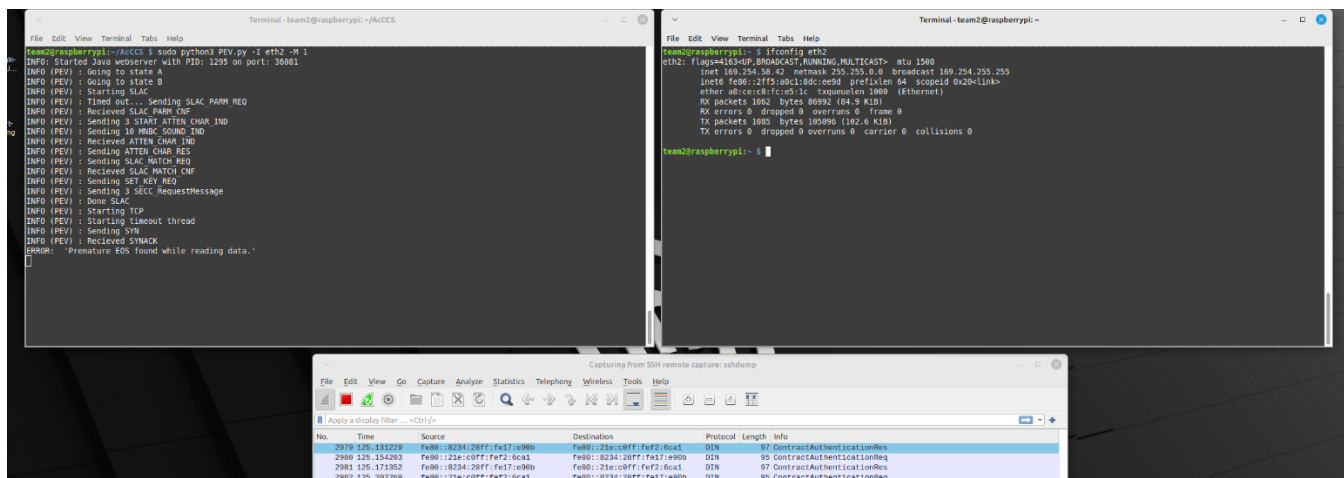
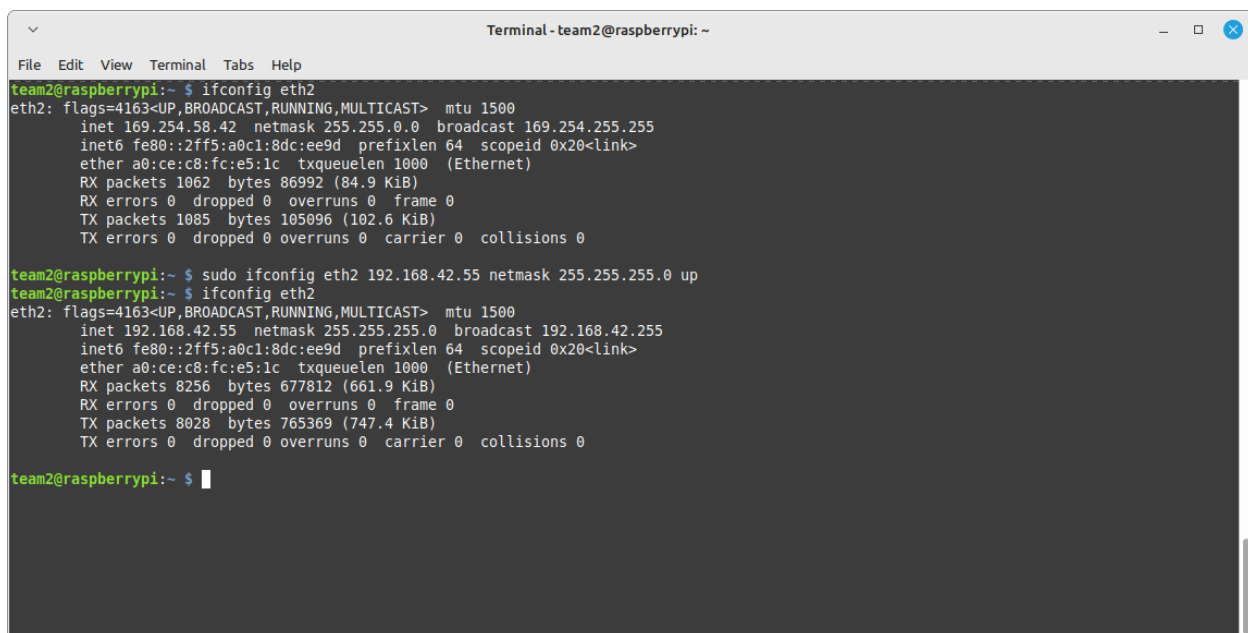


Figure 3: Two terminal windows. The first running the PEV emulator and the second running NMap commands.

**Step 12:** In the second terminal, configure the **eth2** interface on the AcCCS Raspberry Pi to use an IPv4 address in the same address range as the one you identified earlier. You can do this by running the following command.

```
sudo ifconfig eth2 192.168.42.55 netmask 255.255.255.0 up
```



**Note:** The broadcast ARP packets you identified earlier should have come from addresses in the 192.168.42.0/24 address space. Specifically one address was responding to the ARP requests, so we will target that IP (192.168.42.100). We are configuring **eth2** to use the IP address 192.168.42.55 so that we can communicate with 192.168.42.100.

**Step 13:** In the second terminal window, ping 192.168.42.100 and ensure you are able to get responses.

```
Terminal - team2@raspberrypi: ~
File Edit View Terminal Tabs Help
team2@raspberrypi:~$ ping 192.168.42.100
PING 192.168.42.100 (192.168.42.100) 56(84) bytes of data.
64 bytes from 192.168.42.100: icmp_seq=1 ttl=64 time=8.23 ms
64 bytes from 192.168.42.100: icmp_seq=2 ttl=64 time=7.75 ms
64 bytes from 192.168.42.100: icmp_seq=3 ttl=64 time=7.93 ms
64 bytes from 192.168.42.100: icmp_seq=4 ttl=64 time=7.67 ms
64 bytes from 192.168.42.100: icmp_seq=5 ttl=64 time=7.63 ms
^C
--- 192.168.42.100 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4006ms
rtt min/avg/max/mdev = 7.626/7.839/8.231/0.221 ms
team2@raspberrypi:~$
```

**Step 14:** In the second terminal window, run a basic NMap scan targeting 192.168.42.100. Make sure the PEV emulator in the first terminal window continues to run and that you can still monitor the conversation in Wireshark.

```
sudo nmap -sS -n -e eth2 192.168.42.100
```

```
Terminal - team2@raspberrypi: ~
File Edit View Terminal Tabs Help
team2@raspberrypi:~$ sudo nmap -sS -e eth2 192.168.42.100
Starting Nmap 7.80 ( https://nmap.org ) at 2024-02-14 12:53 MST

```

**Step 15:** Wait a few minutes for the NMap scan to complete. Was the scan successful? What ports are open? What is the MAC address of this remote device? What device do you think is using this IP address?



---

---

---

---

---

```
Terminal - team2@raspberrypi: ~
File Edit View Terminal Tabs Help
team2@raspberrypi:~$ sudo nmap -sS -e eth2 192.168.42.100
Starting Nmap 7.80 ( https://nmap.org ) at 2024-02-14 12:53 MST
Nmap scan report for 192.168.42.100
Host is up (0.0080s latency).
Not shown: 998 filtered ports
PORT      STATE SERVICE
22/tcp    open  ssh
20000/tcp  open  dnp
MAC Address: D0:03:EB:36:87:BF (Texas Instruments)

Nmap done: 1 IP address (1 host up) scanned in 21.56 seconds
team2@raspberrypi:~$
```

**Note:** The scan of 192.168.42.100 was possible due to a lack of network segmentation and firewall use inside of this EVSE. The front panel screen (Human Machine Interface or HMI) is using an IPv4 address to connect to other devices in the cabinet as well as remote services (e.g. OCPP). The IPv4 network packets should never be allowed out via the CCS cable. This is a good example of finding an interesting vulnerability using the AcCCS system.

**Step 16:** What other steps would you take to further investigate these findings? What other NMap scans might you run?

---

---

---

---

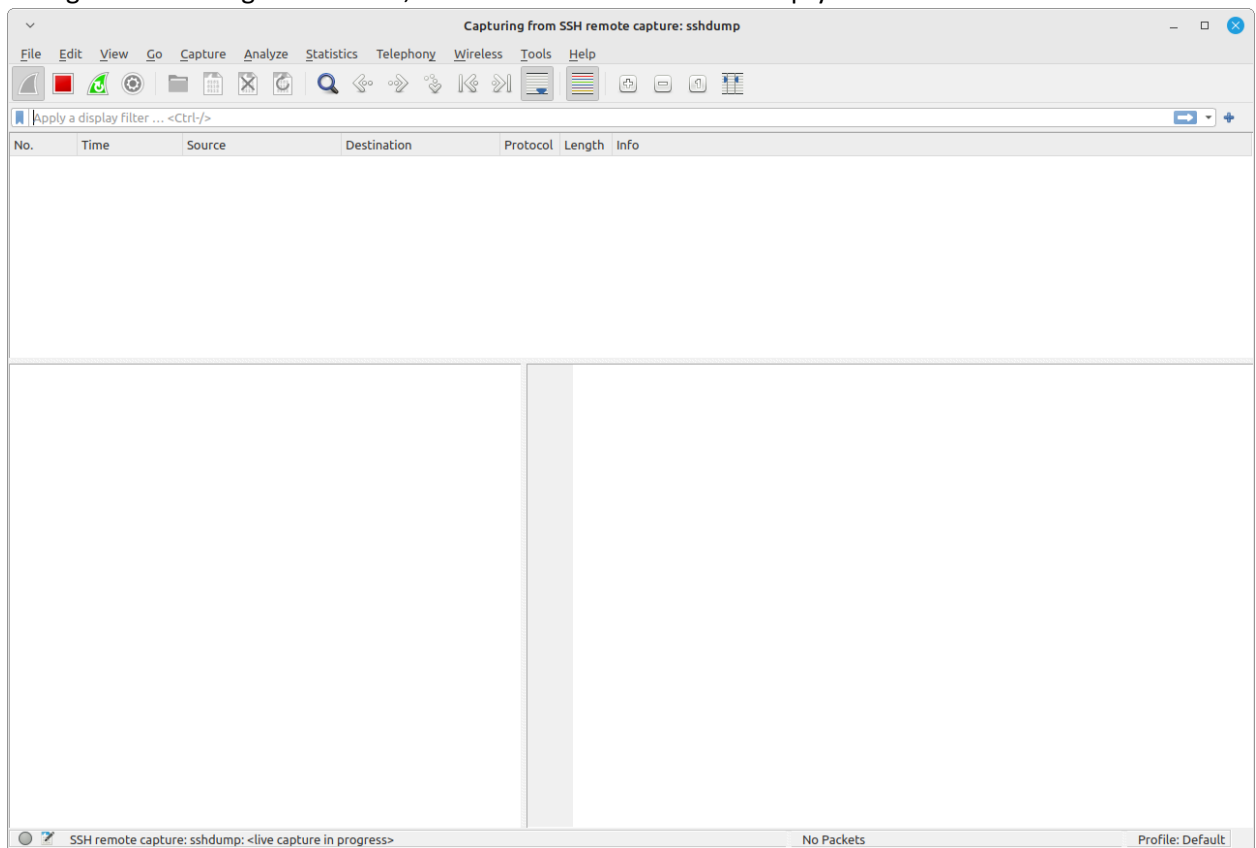
---

## PEV Exercise

For this portion of the exercise, we will impersonate an EVSE and connect to the PEV. While running the following steps, connect to the AcCCS Raspberry Pi as user **team1**. (user: team1 password: AcCCS).

**Step 1:** Start Wireshark in your VM. We will use Wireshark to remotely monitor the communications from the Raspberry Pi over the CCS cable. Configure the SSH remote capture settings for user **team1** and capture packets on interface **eth1**.

**Step 2:** Start a Wireshark capture by double-clicking the SSH remote capture interface. If your configuration settings are correct, Wireshark will start with an empty screen.



**Step 3:** Start running the EVSE emulator.

The EVSE emulator script has several command-line options. Below is a description of each of these options:

```
usage: EVSE.py [-h] [-M MODE] [-I INTERFACE] [--source-mac SOURCE_MAC] [--source-ip SOURCE_IP]
[--source-port SOURCE_PORT] [--NID NID] [--NMK NMK] [-p PROTOCOL] [--nmap-mac NMAP_MAC]
[--nmap-ip NMAP_IP] [--nmap-ports NMAP_PORTS]

EVSE emulator for AcCCS

optional arguments:
  -h, --help            show this help message and exit
  -M MODE, --mode MODE  Mode for emulator to run in: 0 for full conversation, 1 for stalling the
conversation, 2 for portscanning (default: 0)
  -I INTERFACE, --interface INTERFACE Ethernet interface to send/recieve packets on (default: eth1)
  --source-mac SOURCE_MAC Source MAC address of packets (default: 00:1e:c0:f2:6c:a0)
  --source-ip SOURCE_IP Source IP address of packets (default: fe80::21e:c0ff:fef2:72f3)
  --source-port SOURCE_PORT Source port of packets (default: 25565)
  --NID NID Network ID of the HomePlug GreenPHY AVLN (default: \x9c\xb0\xb2\xbb\xf5\x6c\x0e)
  --NMK NMK Network Membership Key of the HomePlug GreenPHY AVLN
  (default: \x48\xfe\x56\x02\xdb\xac\xcd\xe5\x1e\xda\xdc\x3e\x08\x1a\x52\xd1)
  -p PROTOCOL, --protocol PROTOCOL Protocol for EXI encoding/decoding: DIN, ISO-2, ISO-20
  (default: DIN)
  --nmap-mac NMAP_MAC The MAC address of the target device to NMAP scan
  (default: EVCC MAC address)
  --nmap-ip NMAP_IP The IP address of the target device to NMAP scan (default: EVCC IP address)
  --nmap-ports NMAP_PORTS List of ports to scan seperated by commas
  (ex. 1,2,5-10,19,...) (default: Top 8000 common ports)
```

Start the EVSE emulator and monitor Wireshark to validate the connection is stable.

```
sudo python3 EVSE.py -I eth1 -M 1
```

**Note:** Most electric vehicles are finicky about being kept in a wait state while the charge port is connected. It is not uncommon for the connection between the PEV and the AcCCS box to periodically stop. When that happens, unplug the cordset from the vehicle and restart the EVSE emulator.

**Step 4:** Monitor Wireshark and take note of any new packets you see between the EVSE emulator and the PEV. Review your captured network traffic and answer the following questions:

Capturing from SSH remote capture: sshdump

File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

Apply a display filter ... <Ctrl-F>

No.	Time	Source	Destination	Protocol	Length	Info
2441	1353.806689	fe80::218:23ff:fe11:d2f4	ff02::1	V2GTP	72	SECC Discovery Protocol Request
2442	1354.045552	fe80::218:23ff:fe11:d2f4	ff02::1	V2GTP	72	SECC Discovery Protocol Request
2443	1354.064313	fe80::21e:c0ff:fe2:6ca0	fe80::218:23ff:fe11:d2f4	V2GTP	90	SECC Discovery Protocol Response
2444	1354.100116	fe80::21e:c0ff:fe2:6ca0	fe80::218:23ff:fe11:d2f4	V2GTP	90	SECC Discovery Protocol Response
2445	1354.115565	fe80::218:23ff:fe11:d2f4	ff02::2	ICMPv6	62	Router Solicitation
2446	1354.140136	fe80::21e:c0ff:fe2:6ca0	fe80::218:23ff:fe11:d2f4	V2GTP	90	SECC Discovery Protocol Response
2447	1354.465814	fe80::218:23ff:fe11:d2f4	ff02::1:ffff2:6ca0	ICMPv6	86	Neighbor Solicitation for fe80::21e:c0ff:fe2:6ca0 from 00:18...
2448	1354.488511	fe80::21e:c0ff:fe2:6ca0	fe80::218:23ff:fe11:d2f4	ICMPv6	86	Neighbor Advertisement fe80::21e:c0ff:fe2:6ca0 (sol, ovr) is...
2449	1354.505739	fe80::218:23ff:fe11:d2f4	fe80::21e:c0ff:fe2:6ca0	TCP	78	60169 → 25565 [SYN] Seq=0 Win=4000 Len=0 MSS=1440
2450	1354.532119	fe80::21e:c0ff:fe2:6ca0	fe80::218:23ff:fe11:d2f4	TCP	74	25565 → 60169 [SYN, ACK] Seq=0 Ack=1 Win=8192 Len=0
2451	1354.545325	fe80::218:23ff:fe11:d2f4	fe80::21e:c0ff:fe2:6ca0	TCP	74	60169 → 25565 [ACK] Seq=1 Ack=1 Win=4000 Len=0
2452	1354.595461	fe80::218:23ff:fe11:d2f4	fe80::21e:c0ff:fe2:6ca0	V2GEXI	150	supportedAppProtocolReq
2453	1354.685520	fe80::218:23ff:fe11:d2f4	fe80::21e:c0ff:fe2:6ca0	TCP	150	[TCP Retransmission] 60169 → 25565 [PSH, ACK] Seq=1 Ack=1 Win...
2454	1354.865444	fe80::218:23ff:fe11:d2f4	fe80::21e:c0ff:fe2:6ca0	TCP	150	[TCP Retransmission] 60169 → 25565 [PSH, ACK] Seq=1 Ack=1 Win...
2455	1355.225403	fe80::218:23ff:fe11:d2f4	fe80::21e:c0ff:fe2:6ca0	TCP	150	[TCP Retransmission] 60169 → 25565 [PSH, ACK] Seq=1 Ack=1 Win...
2456	1355.421251	fe80::21e:c0ff:fe2:6ca0	fe80::218:23ff:fe11:d2f4	V2GEXI	86	supportedAppProtocolRes
2457	1355.435261	fe80::218:23ff:fe11:d2f4	fe80::21e:c0ff:fe2:6ca0	TCP	74	60169 → 25565 [ACK] Seq=77 Ack=13 Win=3988 Len=0
2458	1355.445436	fe80::218:23ff:fe11:d2f4	fe80::21e:c0ff:fe2:6ca0	TCP	74	[TCP Window Update] 60169 → 25565 [ACK] Seq=77 Ack=13 Win=400...
2459	1355.468173	fe80::21e:c0ff:fe2:6ca0	fe80::218:23ff:fe11:d2f4	TCP	86	[TCP Spurious Retransmission] 25565 → 60169 [PSH, ACK] Seq=1...

Frame 2442: 72 bytes on wire (576 bits), 72 bytes captured (576 bits) on interface sshdi  
 Ethernet II, Src: DeltaElectro\_11:d2:f4 (00:18:23:11:d2:f4), Dst: IPv6mcast\_01 (33:33:00:00:00:01)  
 Internet Protocol Version 6, Src: fe80::218:23ff:fe11:d2f4, Dst: ff02::1  
 User Datagram Protocol, Src Port: 60168, Dst Port: 15118  
 V2G Transfer Protocol  
 SECC Discovery Protocol Request  
 SDP Request Security: 0x00 (TLS: YES)  
 SDP Request Transport Protocol: 0x00 (TCP)

SDP Request Security (sdp.RequestSecurity), 1 byte

Packets: 3465 - Displayed: 3465 (100.0%) Profile: Default

What interesting fields are found in the SECC Discovery Protocol Request?

---



---



---



---



---



---

Capturing from SSH remote capture: sshdump

File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

Apply a display filter ... <Ctrl>F

No.	Time	Source	Destination	Protocol	Length	Info
2441	1353.806689	fe80::218:23ff:fe11:d2f4	ff02::1	V2GTP	72	SECC Discovery Protocol Request
2442	1354.045552	fe80::218:23ff:fe11:d2f4	ff02::1	V2GTP	72	SECC Discovery Protocol Request
2443	1354.064313	fe80::21e:c0ff:fe2:6ca0	fe80::218:23ff:fe11:d2f4	V2GTP	90	SECC Discovery Protocol Response
2444	1354.100116	fe80::21e:c0ff:fe2:6ca0	fe80::218:23ff:fe11:d2f4	V2GTP	90	SECC Discovery Protocol Response
2445	1354.115565	fe80::218:23ff:fe11:d2f4	ff02::2	ICMPv6	62	Router Solicitation
2446	1354.140136	fe80::21e:c0ff:fe2:6ca0	fe80::218:23ff:fe11:d2f4	V2GTP	90	SECC Discovery Protocol Response
2447	1354.465814	fe80::218:23ff:fe11:d2f4	ff02::1:ffff2:6ca0	ICMPv6	86	Neighbor Solicitation for fe80::21e:c0ff:fe2:6ca0 from 00:18...
2448	1354.488511	fe80::21e:c0ff:fe2:6ca0	fe80::218:23ff:fe11:d2f4	ICMPv6	86	Neighbor Advertisement fe80::21e:c0ff:fe2:6ca0 (sol, ovr) is...
2449	1354.505739	fe80::218:23ff:fe11:d2f4	fe80::21e:c0ff:fe2:6ca0	TCP	78	60169 → 25565 [SYN] Seq=0 Win=4000 Len=0 MSS=1440
2450	1354.532119	fe80::21e:c0ff:fe2:6ca0	fe80::218:23ff:fe11:d2f4	TCP	74	25565 → 60169 [SYN, ACK] Seq=0 Ack=1 Win=8192 Len=0
2451	1354.545325	fe80::218:23ff:fe11:d2f4	fe80::21e:c0ff:fe2:6ca0	TCP	74	60169 → 25565 [ACK] Seq=1 Ack=1 Win=4000 Len=0
2452	1354.595461	fe80::218:23ff:fe11:d2f4	fe80::21e:c0ff:fe2:6ca0	V2GEXI	150	supportedAppProtocolReq
2453	1354.685629	fe80::218:23ff:fe11:d2f4	fe80::21e:c0ff:fe2:6ca0	TCP	150	[TCP Retransmission] 60169 → 25565 [PSH, ACK] Seq=1 Ack=1 Win...
2454	1354.865444	fe80::218:23ff:fe11:d2f4	fe80::21e:c0ff:fe2:6ca0	TCP	150	[TCP Retransmission] 60169 → 25565 [PSH, ACK] Seq=1 Ack=1 Win...
2455	1355.225403	fe80::218:23ff:fe11:d2f4	fe80::21e:c0ff:fe2:6ca0	TCP	150	[TCP Retransmission] 60169 → 25565 [PSH, ACK] Seq=1 Ack=1 Win...
2456	1355.421251	fe80::21e:c0ff:fe2:6ca0	fe80::218:23ff:fe11:d2f4	V2GEXI	86	supportedAppProtocolRes
2457	1355.435261	fe80::218:23ff:fe11:d2f4	fe80::21e:c0ff:fe2:6ca0	TCP	74	60169 → 25565 [ACK] Seq=77 Ack=13 Win=3988 Len=0
2458	1355.445436	fe80::218:23ff:fe11:d2f4	fe80::21e:c0ff:fe2:6ca0	TCP	74	[TCP Window Update] 60169 → 25565 [ACK] Seq=77 Ack=13 Win=400...
2459	1355.468173	fe80::21e:c0ff:fe2:6ca0	fe80::218:23ff:fe11:d2f4	TCP	86	[TCP Spurious Retransmission] 25565 → 60169 [PSH, ACK] Seq=1...

Frame 2443: 90 bytes on wire (720 bits), 90 bytes captured (720 bits) on interface sshdi

Ethernet II, Src: MicrochipTec\_f2:6c:a0 (00:1e:c0:f2:6c:a0), Dst: DeltaElectro\_11:d2:f4

Internet Protocol Version 6, Src: fe80::21e:c0ff:fe2:6ca0, Dst: fe80::218:23ff:fe11:d2f4

User Datagram Protocol, Src Port: 15118, Dst Port: 60168

V2G Transfer Protocol

- SDP Protocol Version: 0x01 (V2GTP Version 1)
- SDP Inverted Protocol Version: 0xfe (V2GTP Version 1)
- SDP Payload Type: 0x9001 (SDP RESPONSE)
- SDP Payload Length: 0x00000014

SECC Discovery Protocol Response

- SDP Response IP Address: fe80::21e:c0ff:fe2:6ca0
- SDP Response Port: 25565
- SDP Response Security: 0x10 (TLS: NO)
- SDP Response Transport Protocol: 0x00 (TCP)

SDP Response Security (sdp.ResponseSecurity), 1 byte

Packets: 3470 - Displayed: 3470 (100.0%) Profile: Default

The PEV is requesting TLS (SSL encryption) in the SECC Discovery Protocol Request message, but what does the AcCCS EVSE emulator return in response? Why might this be useful?

---



---



---



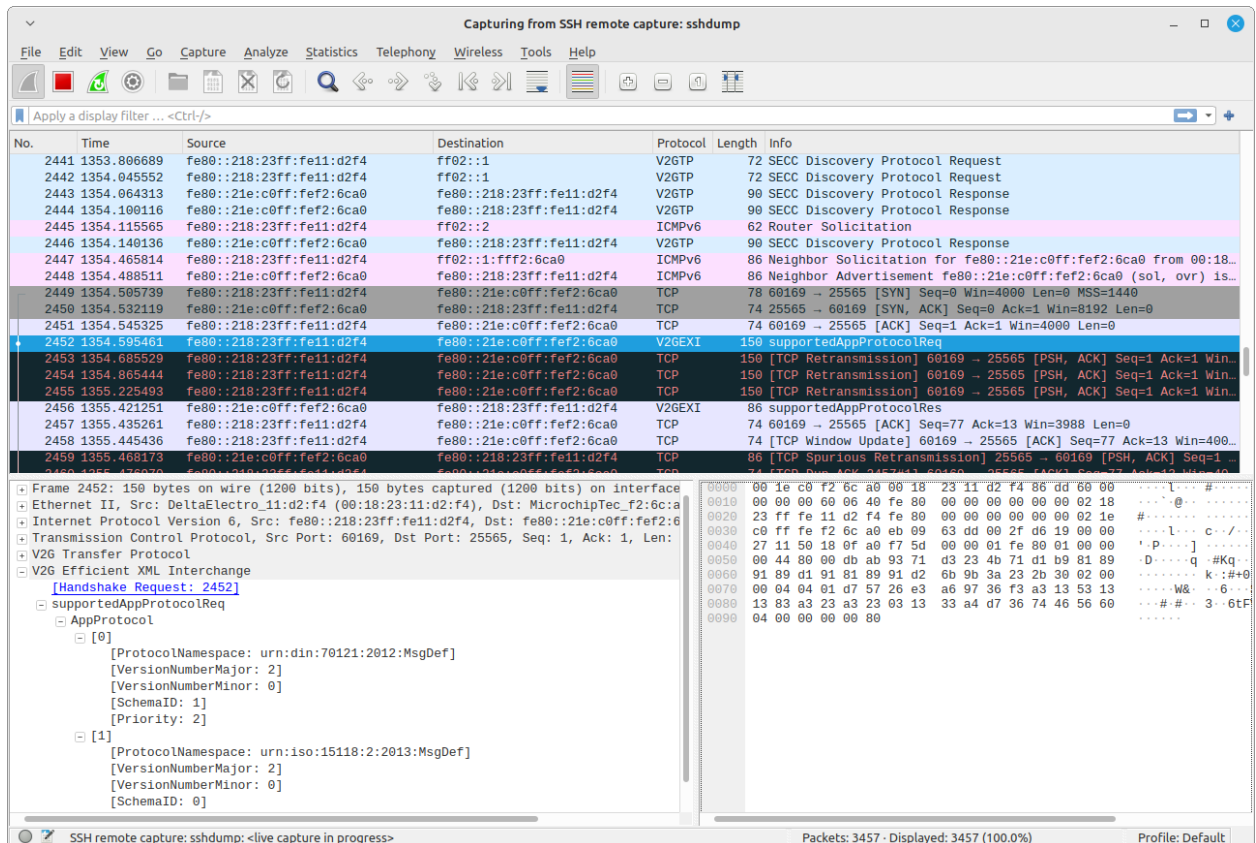
---



---



---



Examine the Supported App Protocol Request message from the PEV. What are the protocol(s) available for EVSE to PEV communications? Examine the response from the EVSE emulator. Which protocol was selected? Why might this be useful?

---



---



---



---



---



---

**Step 5:** Press Ctrl+C to terminate the EVSE emulator or wait for the vehicle to stop communicating.

**PEV Scanning:** Start the EVSE emulator in your terminal window with the following command.

**Note:** the **-M** option has changed to include port scanning.

```
sudo python3 EVSE.py -I eth2 -M 2
```

The network scan will take several minutes to complete, but you should have a progress bar in your terminal window indicating how long the scan has remaining.

```
Terminal - team1@raspberrypi: ~/AcCCS
File Edit View Terminal Tabs Help

team1@raspberrypi:~/AcCCS $ sudo python3 EVSE.py -I eth1 -M 2
INFO: Started Java webserver with PID: 1824 on port: 57867
INFO (EVSE): Opening CP/PP relay connections
INFO (EVSE): Closing CP relay connection
INFO (EVSE): Sending SET_KEY_REQ
INFO (EVSE): SLAC timed out, resetting connection...
INFO (EVSE): Opening CP/PP relay connections
INFO (EVSE): Received SLAC_PARM_REQ
INFO (EVSE): Sending CM_SLAC_PARM_CNF
INFO (EVSE): Received last MNBC_SOUND_IND
INFO (EVSE): Sending ATTN_CHAR_IND
INFO (EVSE): Received SLAC_MATCH_REQ
INFO (EVSE): Sending SLAC_MATCH_CNF
INFO (EVSE): Received SECC_RequestMessage
INFO (EVSE): Done SLAC
INFO (EVSE): Starting TCP
INFO (EVSE): Starting timeout thread
INFO (EVSE): Sending SECC_ResponseMessage
INFO (EVSE): Sending SECC_ResponseMessage
INFO (EVSE): Sending SECC_ResponseMessage
INFO (EVSE): Closing CP relay connection
INFO (EVSE): Sending SYNACK
ERROR: 'Index 10 out of bounds for length 9'
Request: {urn:din:70121:2012:MsgBody}SessionSetupReq
Request: {urn:din:70121:2012:MsgBody}ServiceDiscoveryReq
Request: {urn:din:70121:2012:MsgBody}ServicePaymentSelectionReq
Request: {urn:din:70121:2012:MsgBody}ContractAuthenticationReq
INFO (EVSE): Starting NMAP on port 1 | 1/512
Ports Scanned: 78% | 397/512 [01:00<00:17, 6.66 ports/s]
```

Monitor the network scan activity in Wireshark.

Capturing from SSH remote capture: sshdump

File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

Apply a display filter ... <Ctrl-/>

No.	Time	Source	Destination	Protocol	Length	Info
8295	399.716573	fe80::21e:c0ff:fe2:6ca1	fe80::21e:c0ff:fe2:6ca0	TCP	74	34637 → 602 [SYN] Seq=0 Win=8192 Len=0
8296	399.755578	fe80::21e:c0ff:fe2:6ca0	fe80::21e:c0ff:fe2:6ca1	DIN	97	ContractAuthenticationRes
8297	399.770890	fe80::21e:c0ff:fe2:6ca0	fe80::21e:c0ff:fe2:6ca1	TCP	74	602 → 34637 [RST] Seq=1 Win=8192 Len=0
8298	399.806524	fe80::21e:c0ff:fe2:6ca1	fe80::21e:c0ff:fe2:6ca0	DIN	95	ContractAuthenticationReq
8299	399.818063	fe80::21e:c0ff:fe2:6ca0	fe80::21e:c0ff:fe2:6ca1	TCP	74	602 → 34637 [RST] Seq=1 Win=8192 Len=0
8300	399.848992	fe80::21e:c0ff:fe2:6ca1	fe80::21e:c0ff:fe2:6ca0	TCP	74	4046 → 3456 [SYN] Seq=0 Win=8192 Len=0
8301	399.855863	fe80::21e:c0ff:fe2:6ca0	fe80::21e:c0ff:fe2:6ca1	DIN	97	ContractAuthenticationRes
8302	399.882074	fe80::21e:c0ff:fe2:6ca0	fe80::21e:c0ff:fe2:6ca1	TCP	74	3456 → 4046 [RST] Seq=1 Win=8192 Len=0
8303	399.903435	fe80::21e:c0ff:fe2:6ca1	fe80::21e:c0ff:fe2:6ca0	DIN	95	ContractAuthenticationReq
8304	399.937722	fe80::21e:c0ff:fe2:6ca1	fe80::21e:c0ff:fe2:6ca1	TCP	74	3456 → 4046 [RST] Seq=1 Win=8192 Len=0
8305	399.948615	fe80::21e:c0ff:fe2:6ca0	fe80::21e:c0ff:fe2:6ca1	DIN	97	ContractAuthenticationRes
8306	399.954615	fe80::21e:c0ff:fe2:6ca1	fe80::21e:c0ff:fe2:6ca0	TCP	74	53503 → 862 [SYN] Seq=0 Win=8192 Len=0
8307	399.985741	fe80::21e:c0ff:fe2:6ca0	fe80::21e:c0ff:fe2:6ca1	TCP	74	862 → 53503 [RST] Seq=1 Win=8192 Len=0
8308	400.000629	fe80::21e:c0ff:fe2:6ca1	fe80::21e:c0ff:fe2:6ca0	DIN	95	ContractAuthenticationReq
8309	400.039077	fe80::21e:c0ff:fe2:6ca0	fe80::21e:c0ff:fe2:6ca1	TCP	74	862 → 53503 [RST] Seq=1 Win=8192 Len=0
8310	400.049484	fe80::21e:c0ff:fe2:6ca0	fe80::21e:c0ff:fe2:6ca1	DIN	97	ContractAuthenticationRes
8311	400.069483	fe80::21e:c0ff:fe2:6ca1	fe80::21e:c0ff:fe2:6ca0	TCP	74	23272 → 600 [SYN] Seq=0 Win=8192 Len=0
8312	400.097734	fe80::21e:c0ff:fe2:6ca0	fe80::21e:c0ff:fe2:6ca1	TCP	74	600 → 23272 [RST] Seq=1 Win=8192 Len=0
8313	400.110979	fe80::21e:c0ff:fe2:6ca1	fe80::21e:c0ff:fe2:6ca0	DIN	95	ContractAuthenticationReq

Frame 1688: 74 bytes on wire (592 bits), 74 bytes captured (592 bits) on interface ssh

Ethernet II, Src: MicrochipTec\_f2:6c:a0 (00:1e:c0:f2:6c:a0), Dst: MicrochipTec\_f2:6c:a1 (00:1e:c0:f2:6c:a1)

Destination: MicrochipTec\_f2:6c:a1 (00:1e:c0:f2:6c:a1)

Source: MicrochipTec\_f2:6c:a0 (00:1e:c0:f2:6c:a0)

Type: IPv6 (0x86dd)

Internet Protocol Version 6, Src: fe80::21e:c0ff:fe2:6ca0, Dst: fe80::21e:c0ff:fe2:6ca1

Transmission Control Protocol, Src Port: 100, Dst Port: 63343, Seq: 1, Len: 0

Source Port: 100

Destination Port: 63343

[Stream index: 217]

[Conversation completeness: Incomplete (33)]

[TCP Segment Len: 0]

Sequence Number: 1 (relative sequence number)

Sequence Number (raw): 0

[Next Sequence Number: 1 (relative sequence number)]

Acknowledgment Number: 0

Acknowledgment number (raw): 0

0101 .... = Header Length: 20 bytes (5)

Flags: 0x004 (RST)

Window: 8192

Sequence Number (tcp.seq), 4 bytes

Packets: 8313 · Displayed: 8313 (100.0%)

Profile: Default

When the scan is finished, the emulator script will save a summary of the port scan in the *scan\_results* folder. Terminate the emulator scripts by pressing Ctrl+C.

**Step 6:** Review the scan results using Linux commands like **less**, **more**, and **grep**. List the open ports you identified. What can you determine from these scan results?

```
grep -i "open" scan_results/scan_res_evse_001.txt
```

List the ports you found open on the PEV. What might this mean? Did you expect to find open ports?

---

---

---

---

---

**Note:** It is expected that neither end-point of a CCS connection (i.e. the Electric Vehicle Communication Controller – EVCC or the Supply Equipment Communication Controller – SECC) will have open TCP ports reachable using IPv6. It is not surprising that our scanning of the CCS connection lacks open ports. This is, however, an excellent use case for a system like AcCCS. It is important for OEMs and EVSE manufactures to audit their communications and ensure it is securely implemented.

**Secondary Note:** The Ford Mach-E used in this class uses Automotive Ethernet for internal communications between several of the on-board computers. These networks do utilize IPv4 addresses, but they are also segmented using VLAN technology. It is beyond the scope of this course to attempt to access remote VLANs from the CCS port of the vehicle.