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.

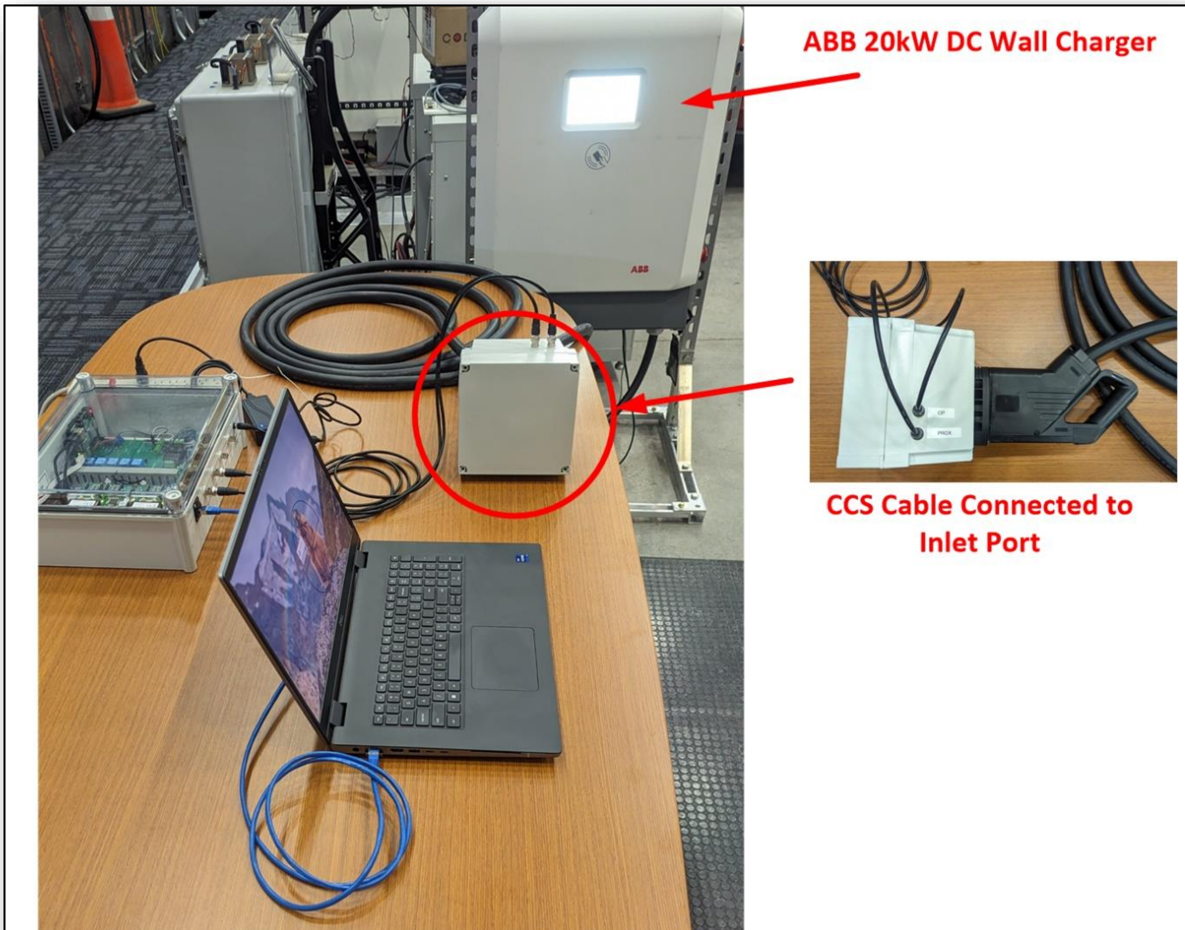


Figure 1: Testing the DC EVSE Wallbox

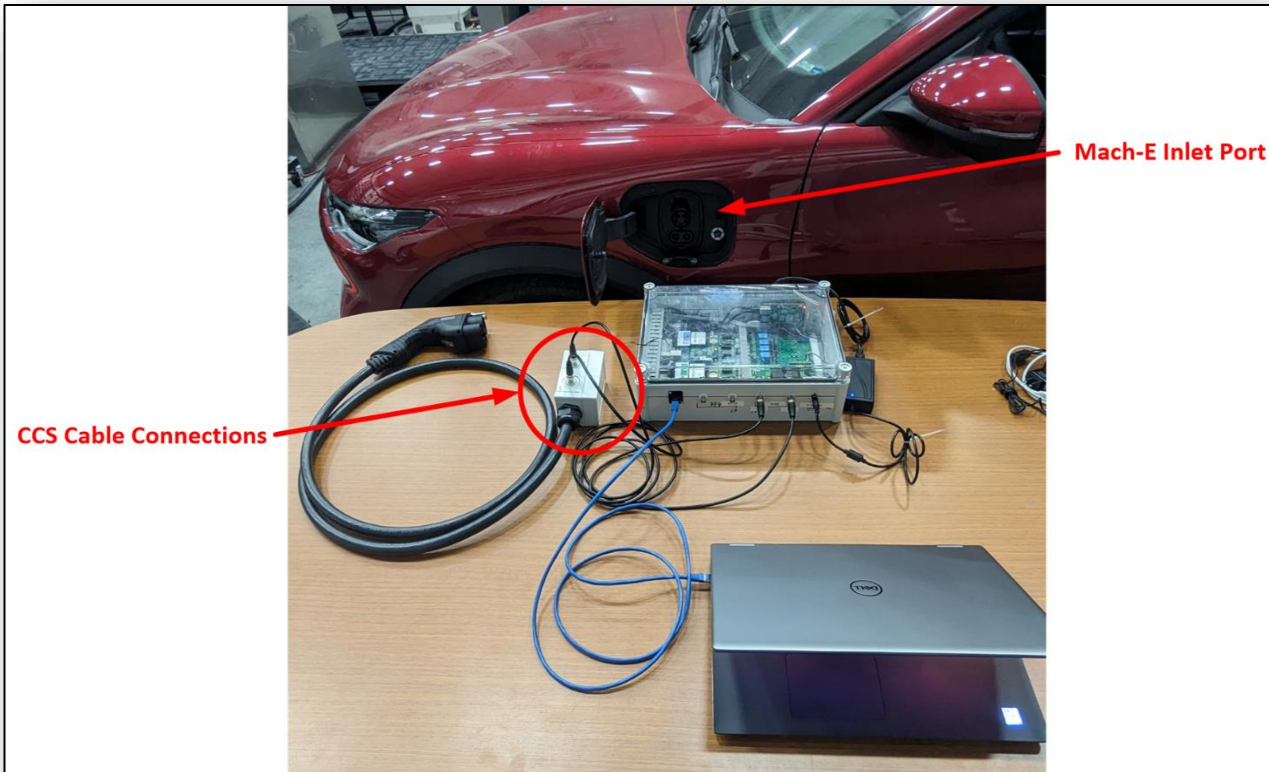


Figure 2: Testing the Mach-E PEV

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.

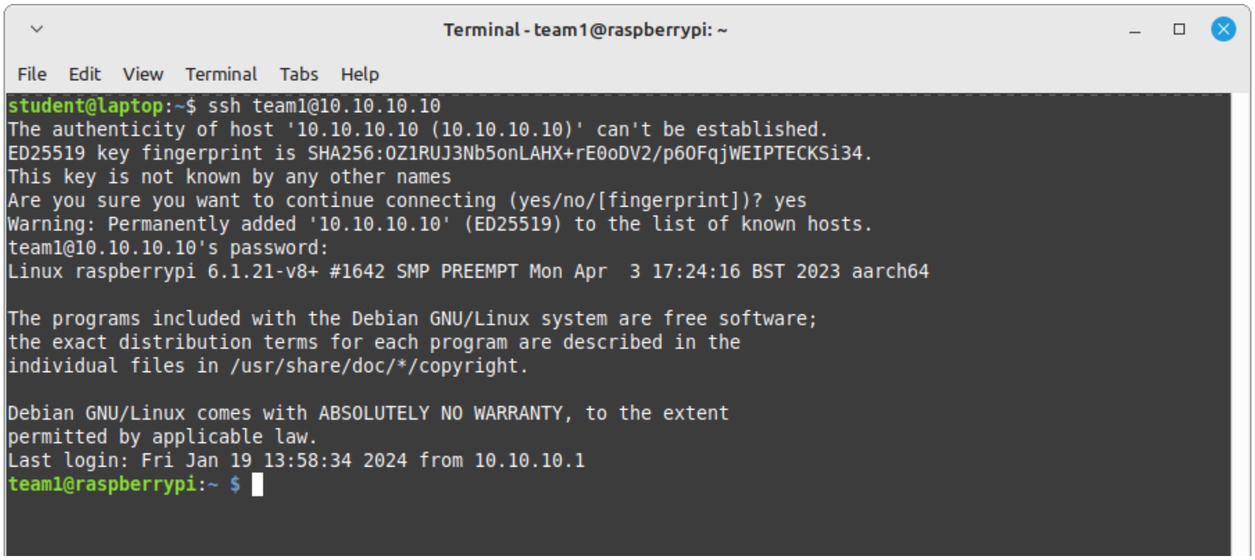
Terminal - student@laptop: ~

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).



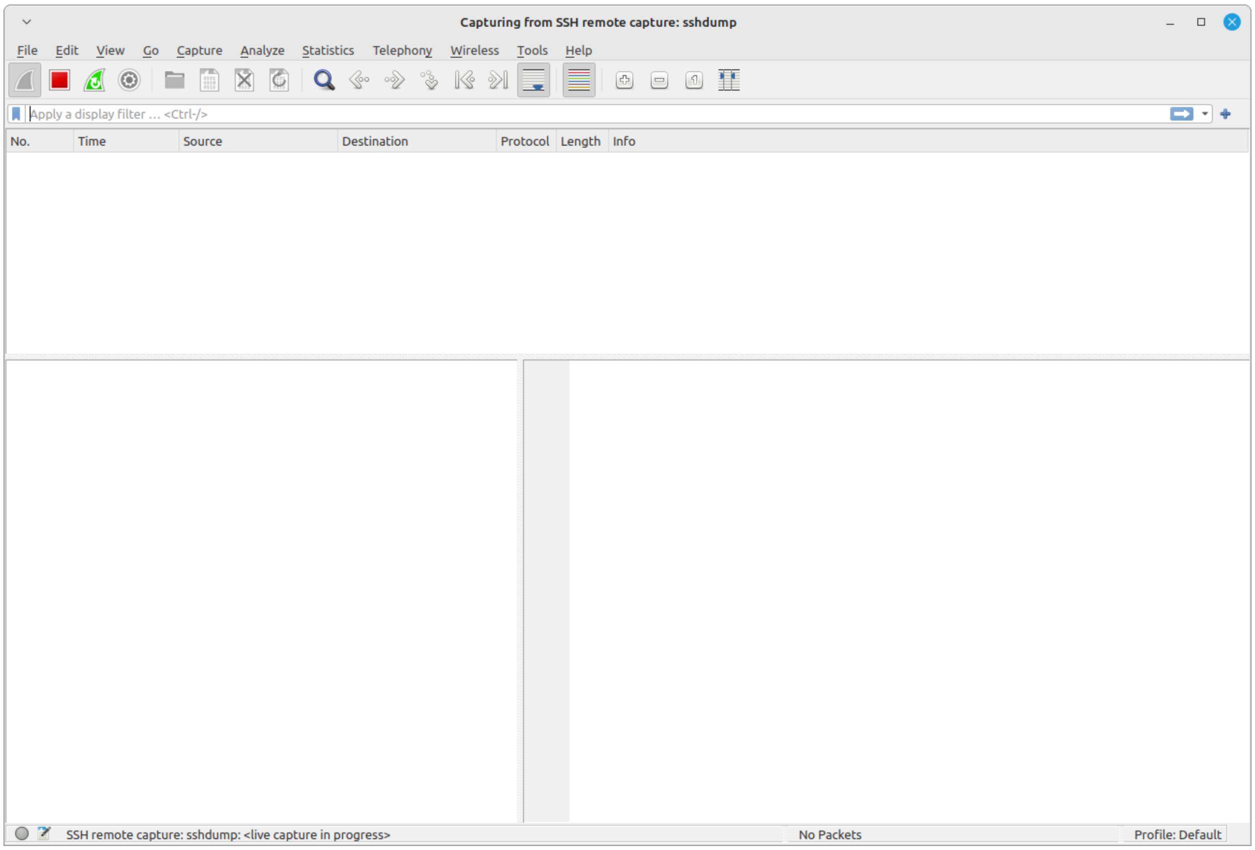
Note: If you receive a message asking you to confirm the SSH connection based on the SHA256 fingerprint, type ‘yes’ in the terminal to continue.

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:

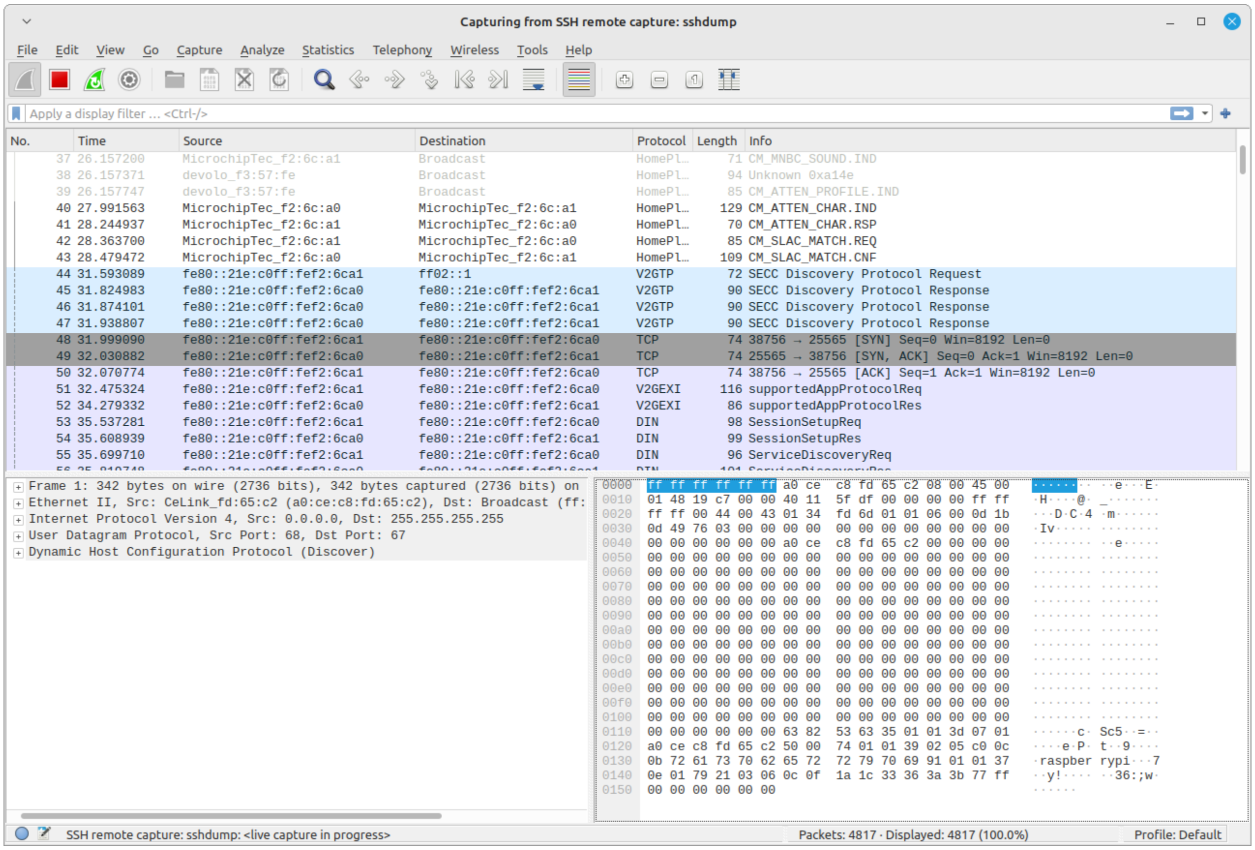
```
usage: PEV.py [-h] [-M MODE] [-I INTERFACE] [--source-mac SOURCE_MAC] [--source-ip SOURCE_IP]
[--source-port SOURCE_PORT] [-p PROTOCOL] [--nmap-mac NMAP_MAC] [--nmap-ip NMAP_IP]
[--nmap-ports NMAP_PORTS]

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.



What are some of the most interesting packets? Which packets do you recognize? Which packets have you never seen before? Make note of some of your most interesting discoveries.

____

____

____

____

____

What differences do you see between this traffic with the actual EVSE compared to running AcCCS in Exercise 1?

____

____

____

____

____

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?

____
____
____
____
____

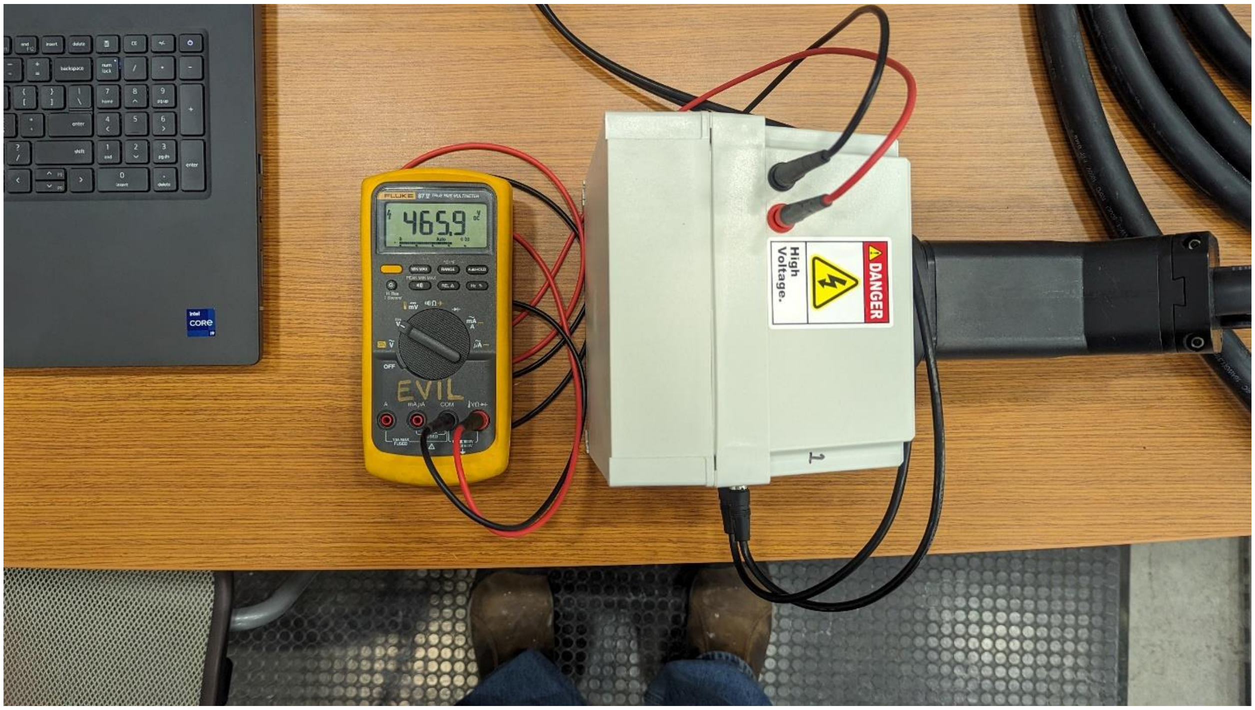
What additional protocol(s) are available for EVSE to PEV communications?

____
____
____
____
____

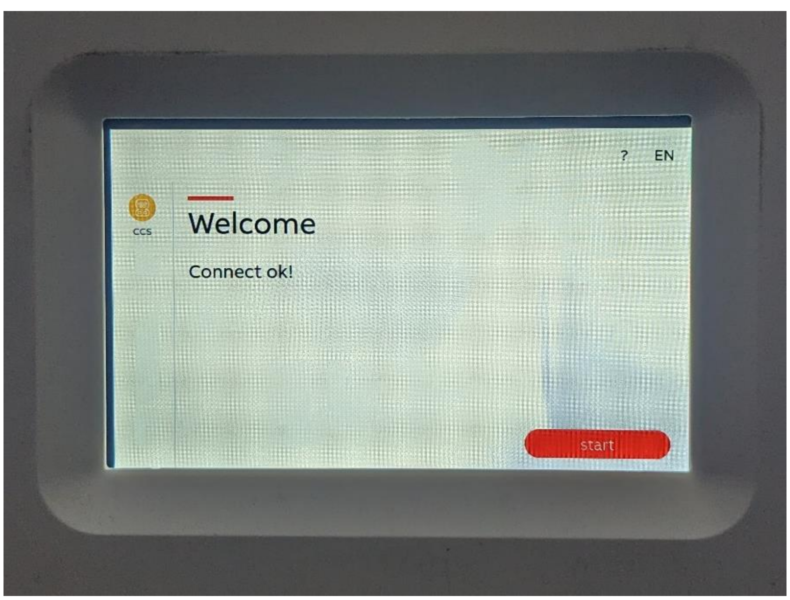
What message(s) are used to stall the conversation and keep the PEV and EVSE in a waiting state?

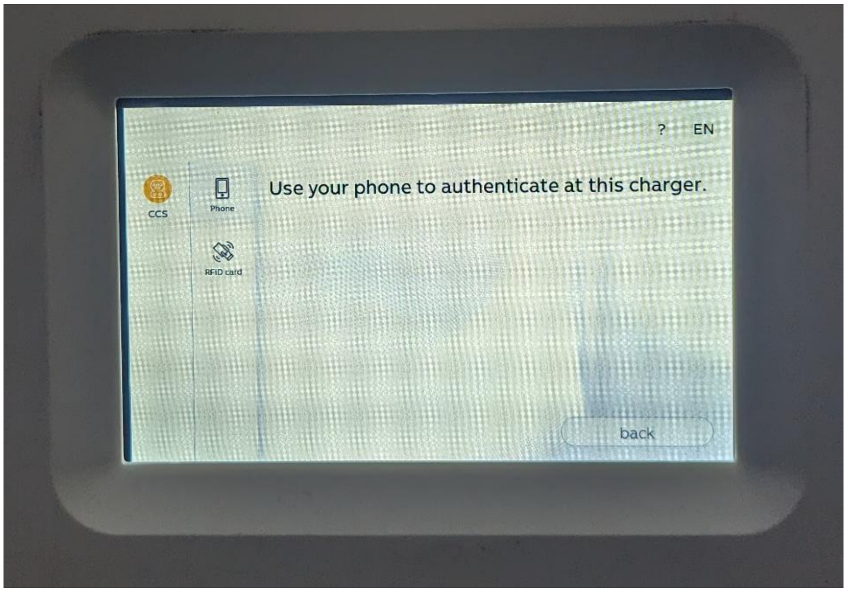
____
____
____
____
____

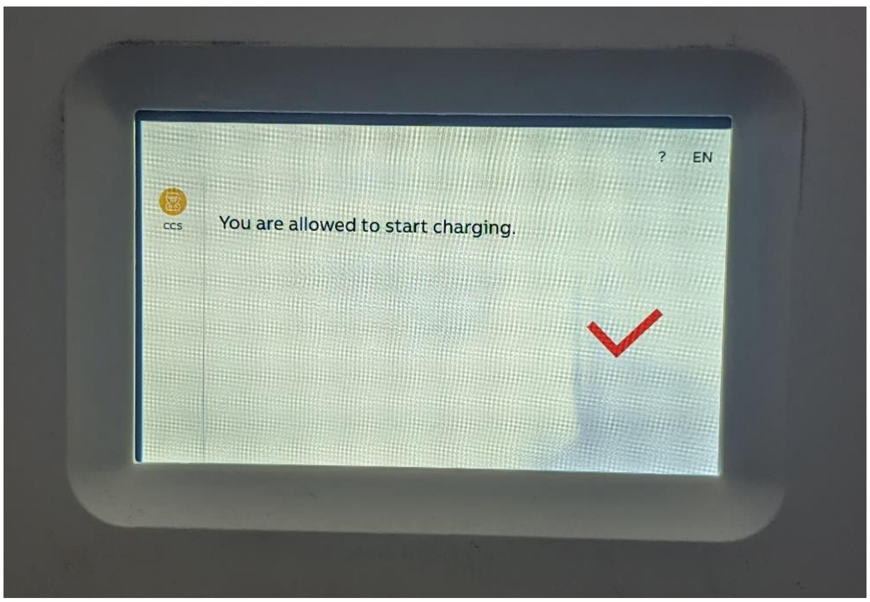
Step 5: Stop the PEV emulator script by pressing Ctrl+C. This will interrupt the script and terminate after a few seconds. Connect a digital multi-meter to the inlet port of the AcCCS box as shown in the following picture.

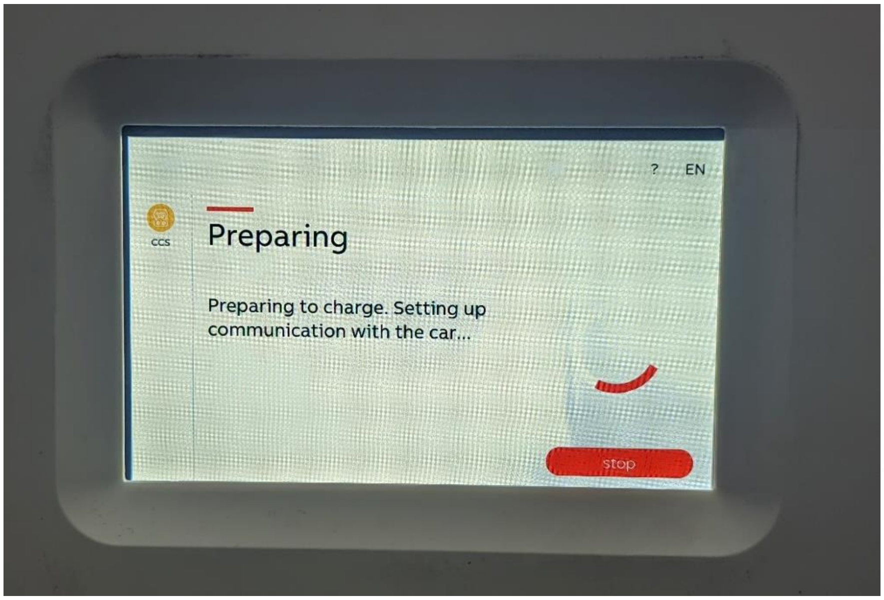


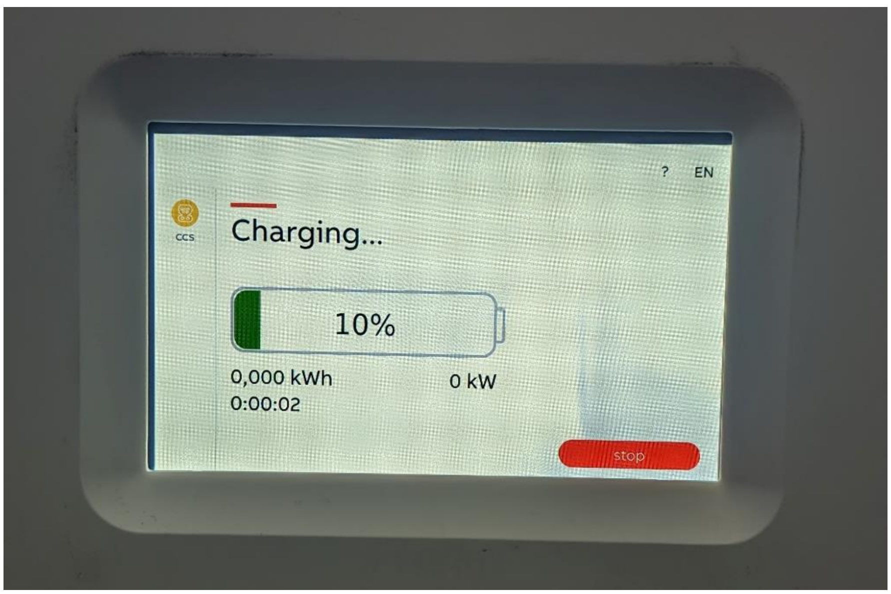
Step 6: Restart the PEV emulator script and monitor the traffic in Wireshark. Once the communications are established, press the Start button on the front of the EVSE. When prompted to authenticate with your phone, scan the provided RFID card on the front of the EVSE.




? EN
CCS
Phone
RFID card
Use your phone to authenticate at this charger.
back


? EN
CCS
You are allowed to start charging.





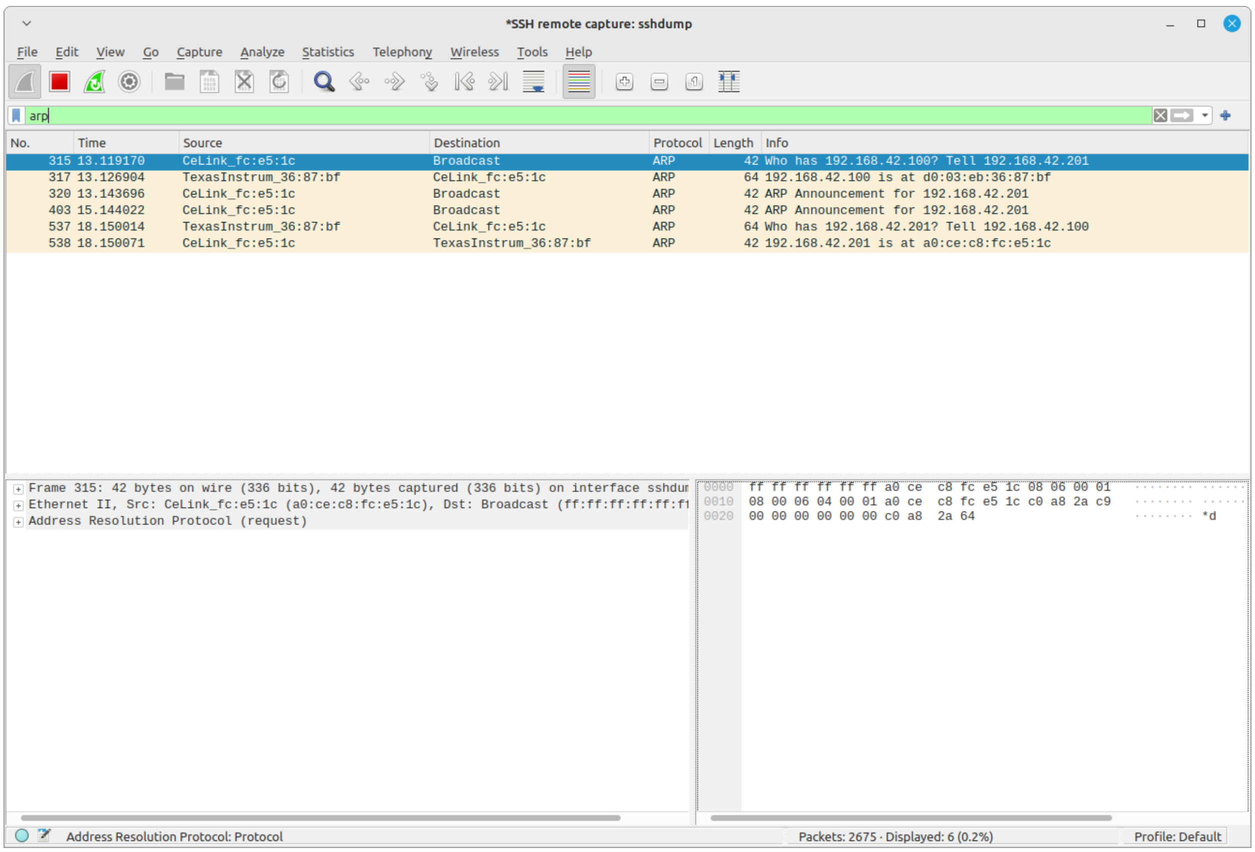
Step 7: Monitor Wireshark and look for what messages change after the Start button was pressed. Also watch the multi-meter and see if the EVSE presents DC voltage over the CCS cable. What messages in Wireshark are new? What actions and messages happened to cause this change? What happens to the voltage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

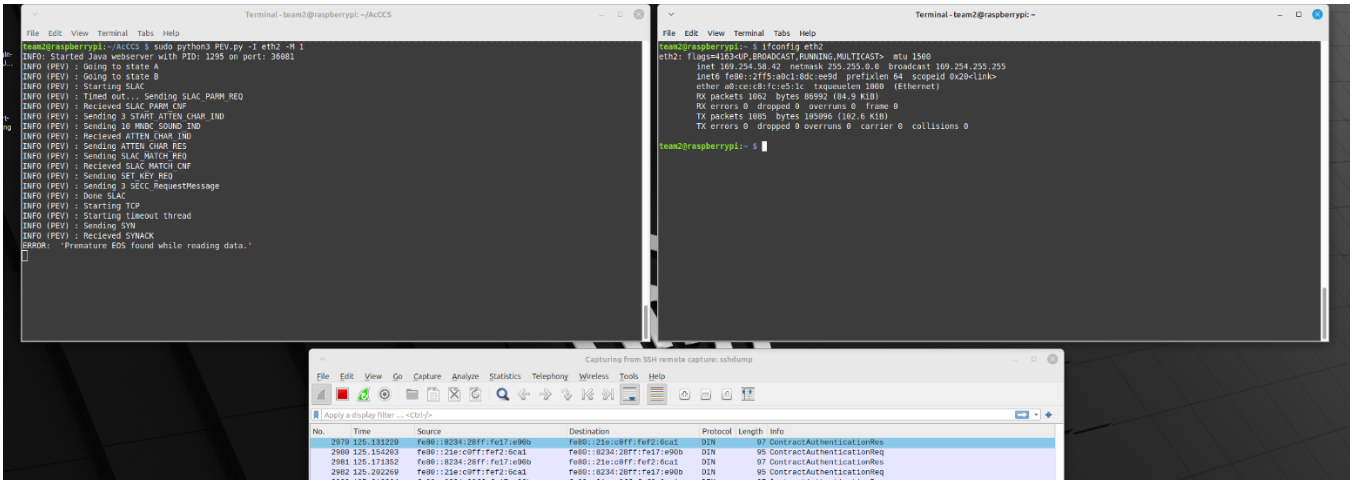
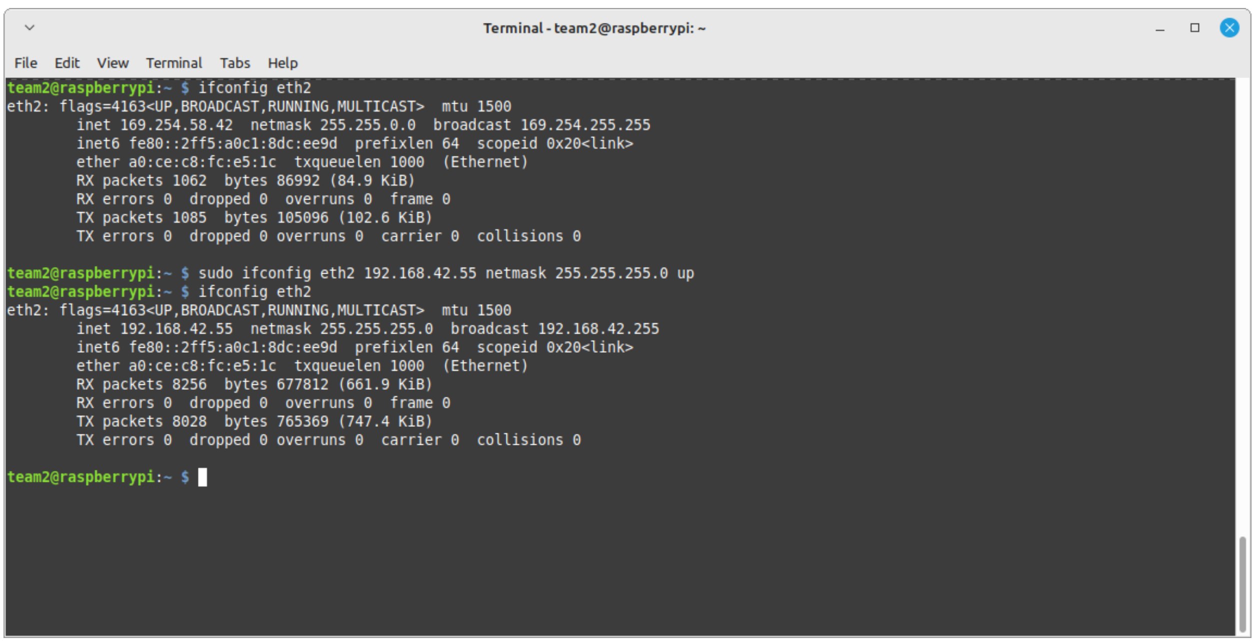


Figure 3: Two terminal windows. The first running the PEV emulator and the second running NMap commands.

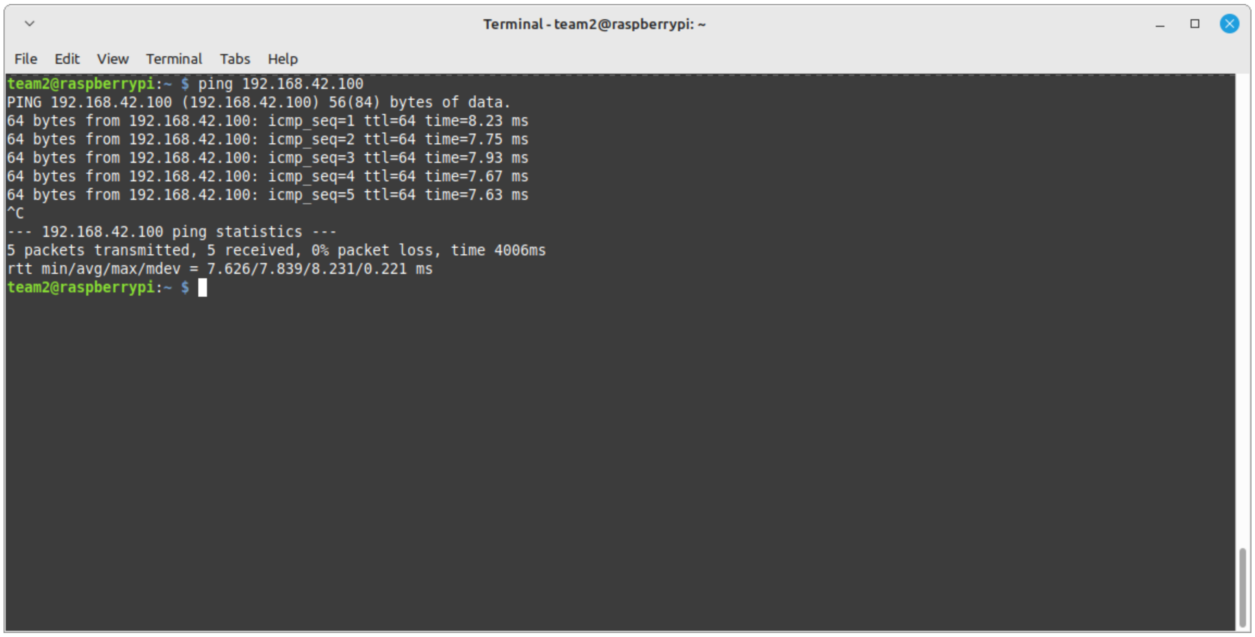
Step 12: In the second terminal, configure the **`eth2`** interface on the AcCCS Raspberry Pi to use an IPv4 address in the same address range as the one you identified earlier. You can do this by running the following command.

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



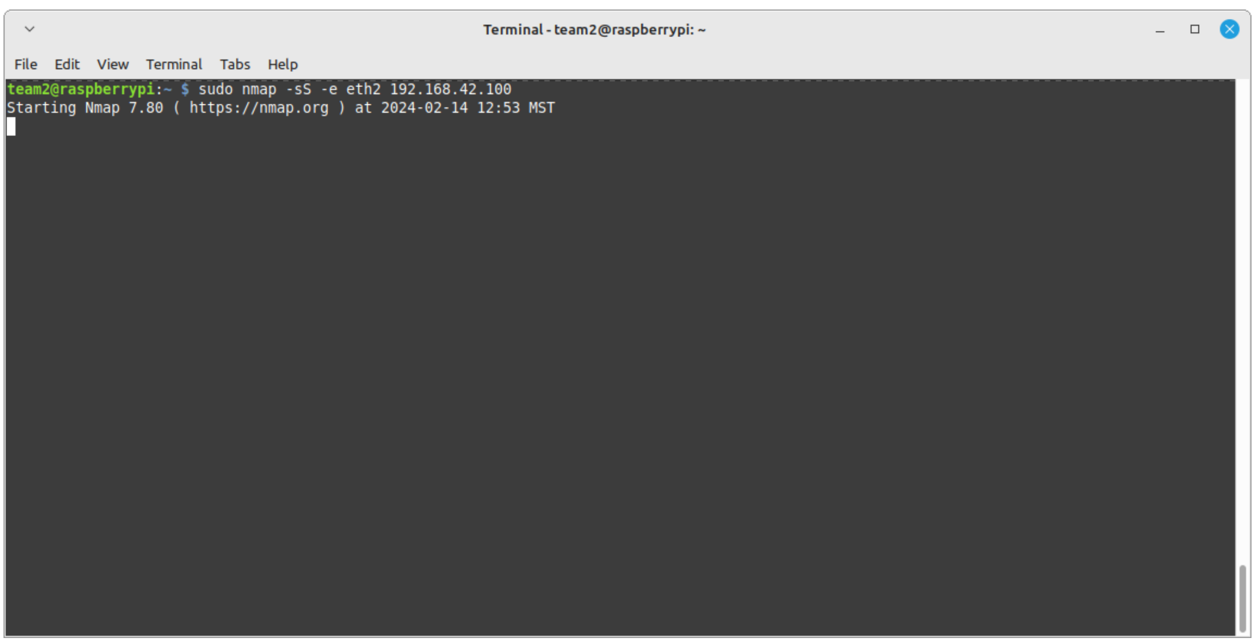
Note: The broadcast ARP packets you identified earlier should have come from addresses in the 192.168.42.0/24 address space. Specifically one address was responding to the ARP requests, so we will target that IP (192.168.42.100). We are configuring **`eth2`** to use the IP address 192.168.42.55 so that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

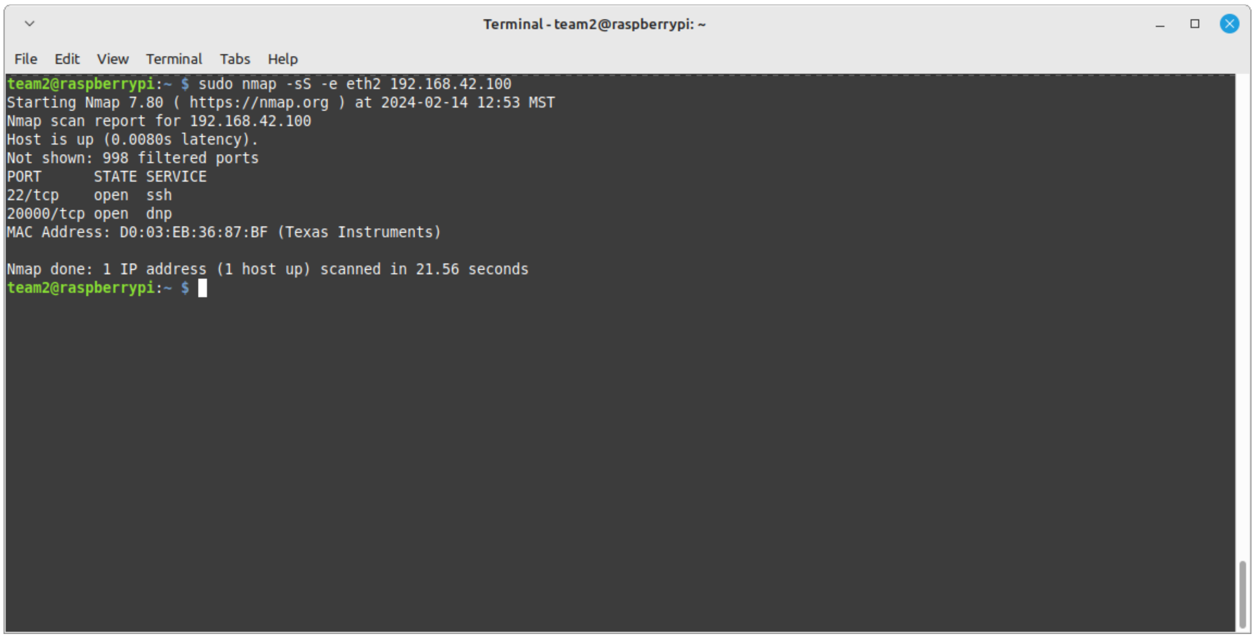


Step 14: In the second terminal window, run a basic NMap scan targeting 192.168.42.100. Make sure the PEV emulator in the first terminal window continues to run and that you can still monitor the conversation in Wireshark.

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



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 to you think is using this IP address?



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.

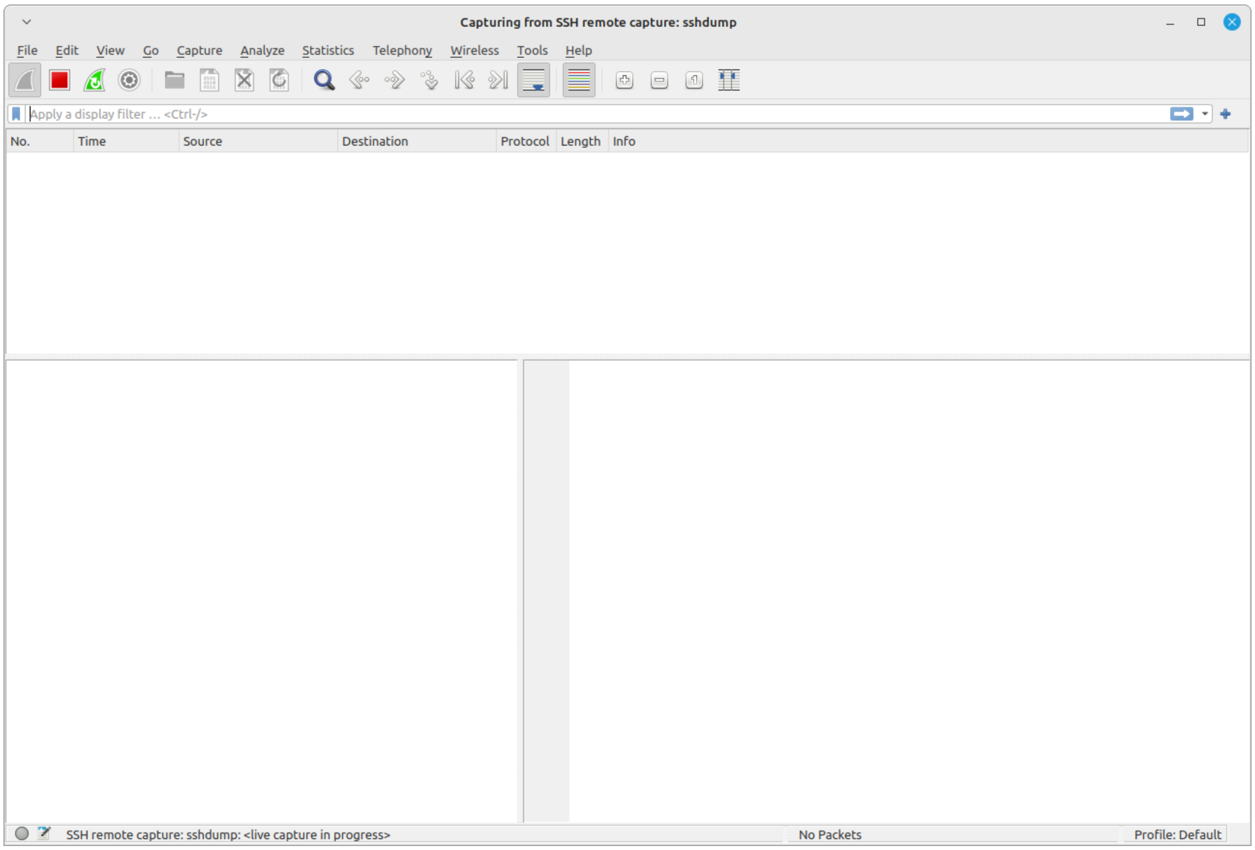
Step 16: What other steps would you take to further investigate these findings? What other NMap scans might you run?

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:

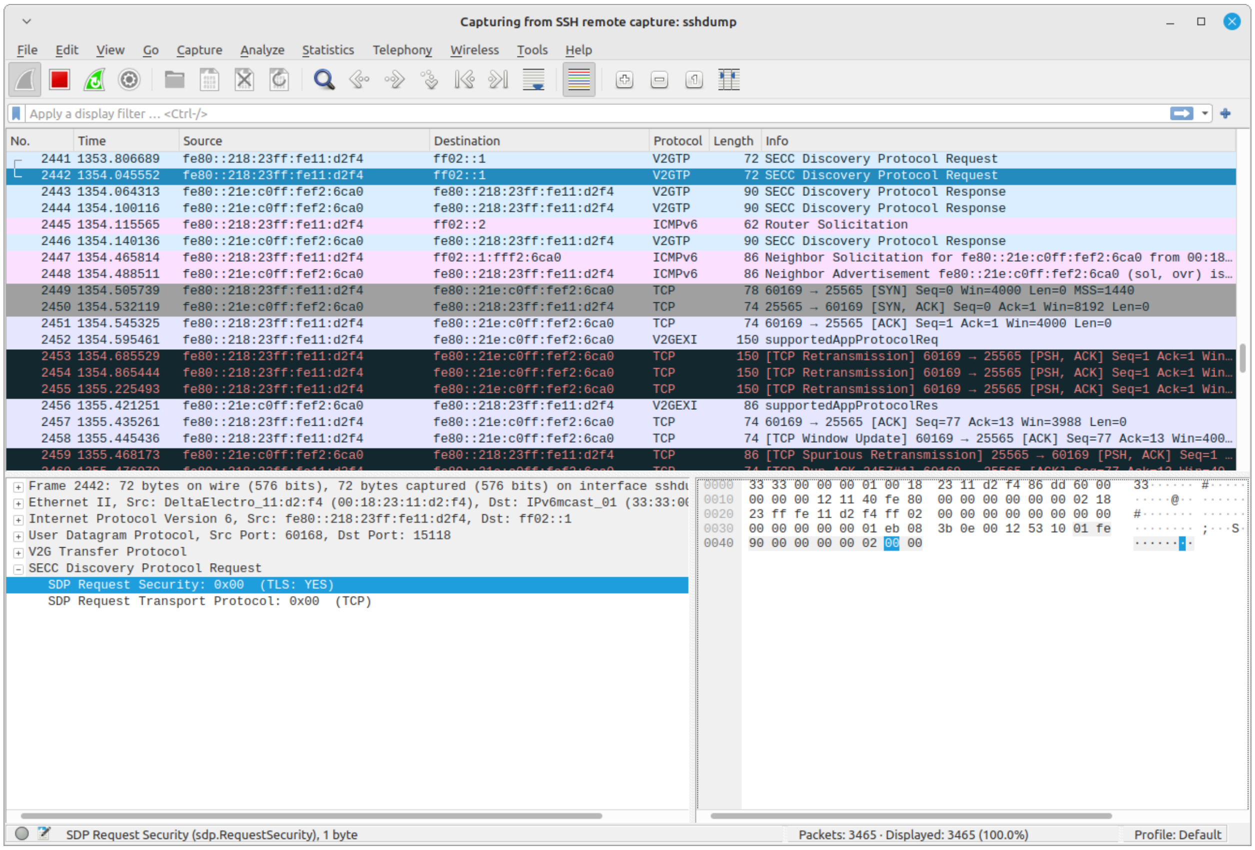
```
usage: EVSE.py [-h] [-M MODE] [-I INTERFACE] [--source-mac SOURCE_MAC] [--source-ip SOURCE_IP]
[--source-port SOURCE_PORT] [--NID NID] [--NMK NMK] [-p PROTOCOL] [--nmap-mac NMAP_MAC]
[--nmap-ip NMAP_IP] [--nmap-ports NMAP_PORTS]

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:



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

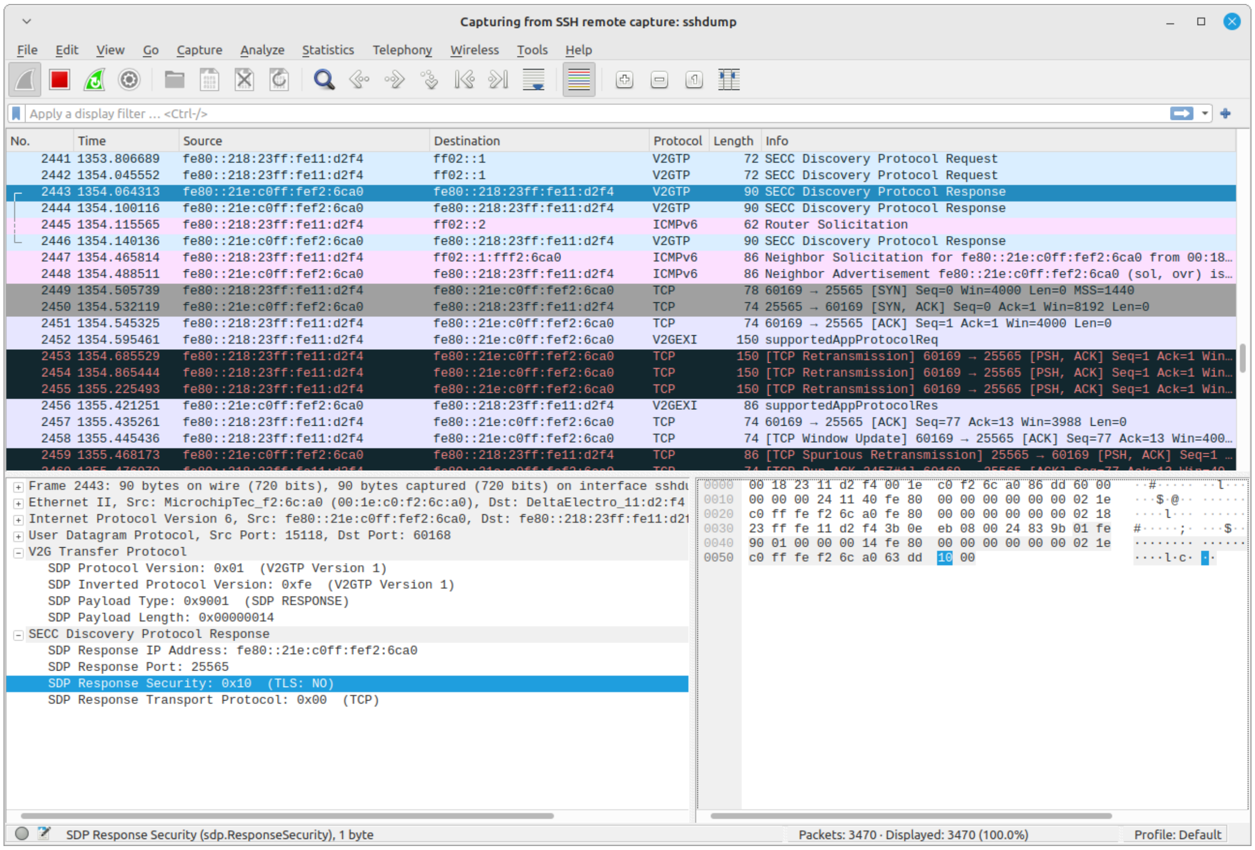
____

____

____

____

____



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?

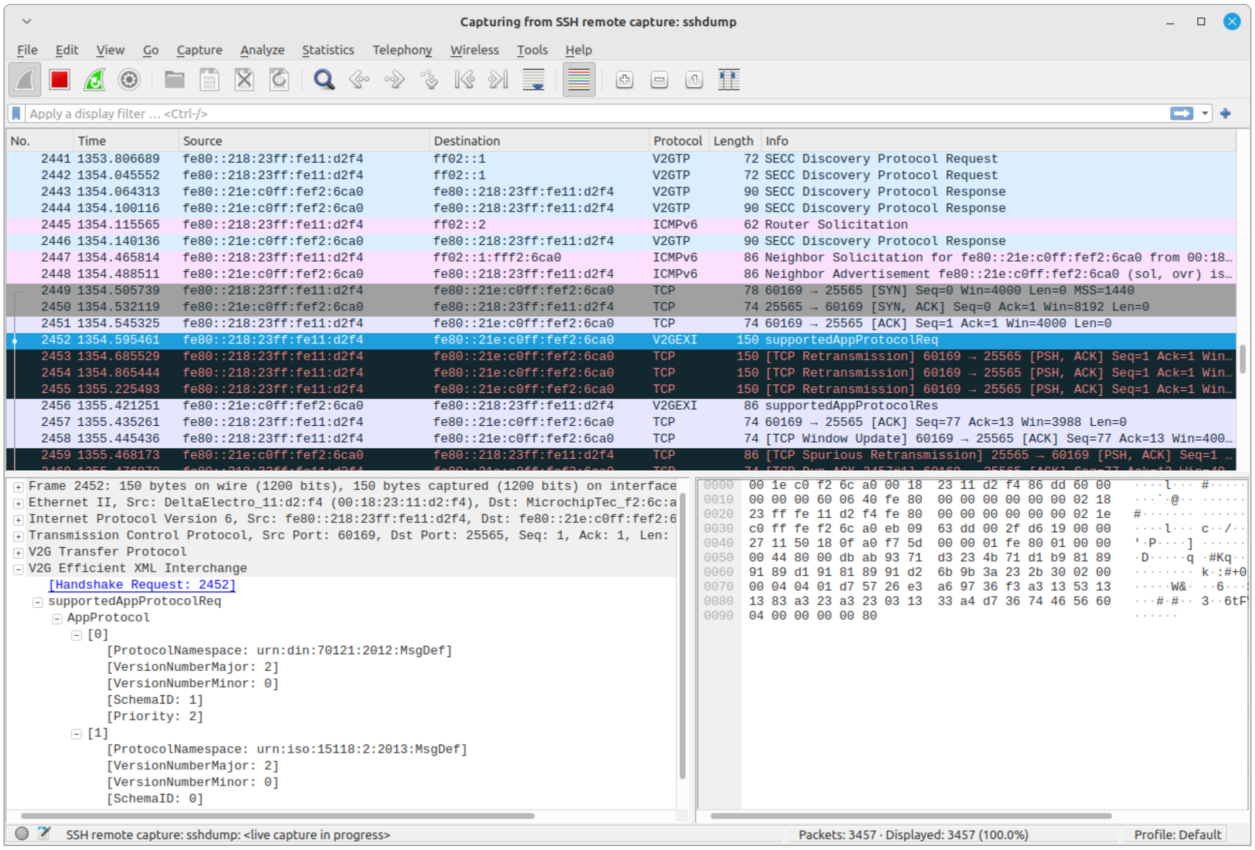
____

____

____

____

____



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?

____

____

____

____

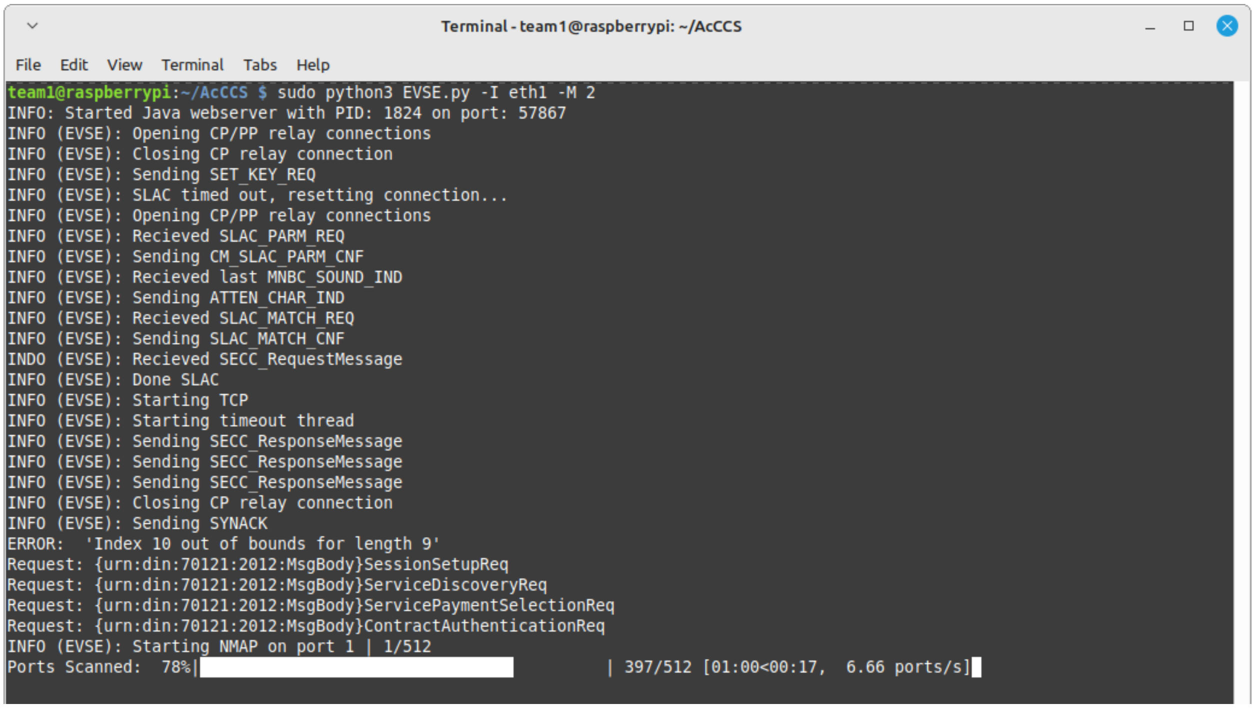
____

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

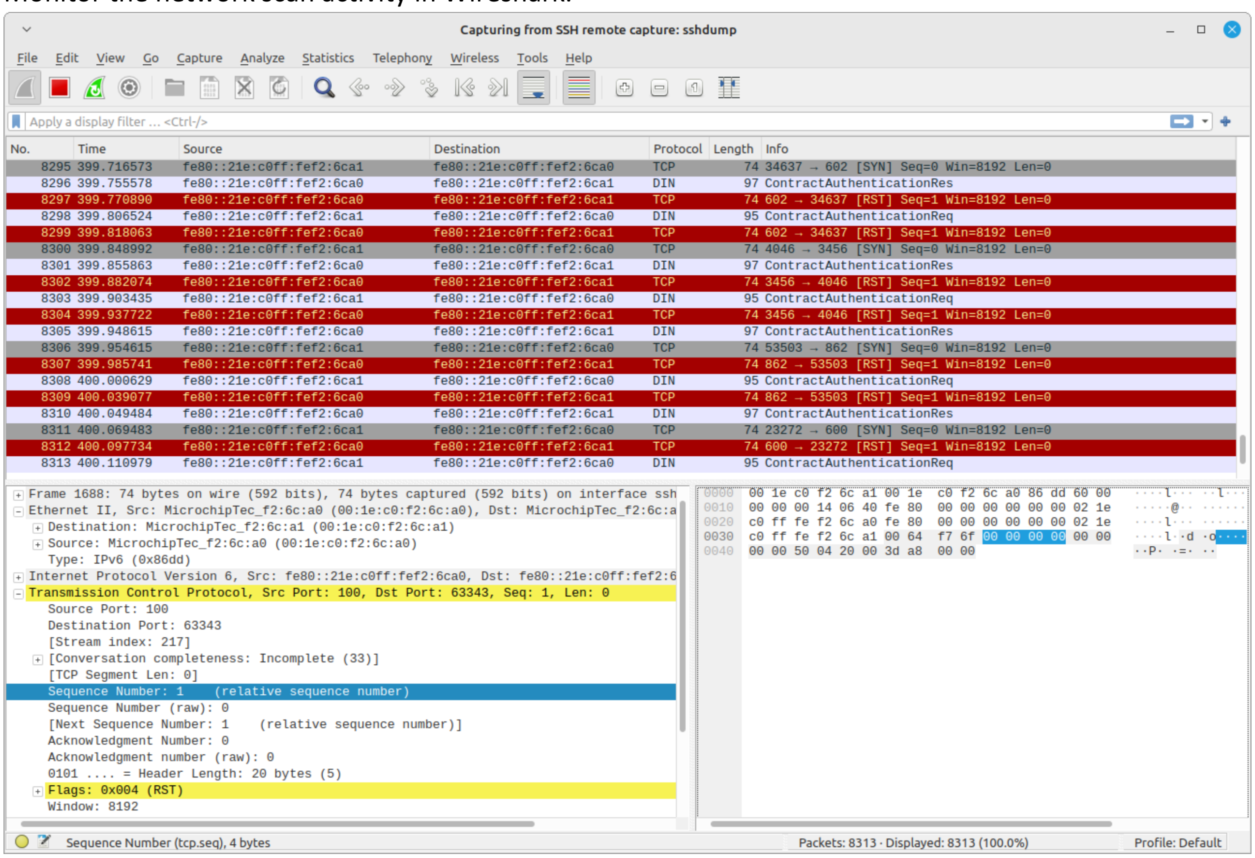
PEV Scanning: Start the EVSE emulator in your terminal window with the following command. **Note**: the **–M** option has changed to include port scanning.

```
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.



Monitor the network scan activity in Wireshark.



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.