#### 1 Introduction

This is a writeup for the challenge "pong" that was part of the BuckeyeCTF 2023 Competition. Through this writeup I hope to share not only my solution but also the logical steps I took to find and reach the solution and acquire the flag.

### 2 First Steps

Miscellaneous challenges can often be hard to approach as the category is very broad and there may not be much information to go off of. For this particular challenge only two pieces of information were given: the name of the challenge "pong" and the description "18.191.205.48". The first step was to identify the description as an ip address, this is clear because it has 4 integers of range 0-255 separated by periods. Additionally, an ip address is a very good entry point because it allows us to connect somehow to a server. This information needed to be pieced together with the play on words of the title "ping pong" to identify that maybe we should try sending a ping to this ip address. So with this idea as a starting place it was time to experiment.

### 3 Experimentation

The first step in the experimentation was simply to send a few ping messages to the server which can be accomplished with the following command:

ping -c 5 18.191.205.48

This yields the following output:

```
PING 18.191.205.48 (18.191.205.48): 56 data bytes
128 bytes from 18.191.205.48: icmp_seq=0 ttl=30 time=43.650 ms
wrong total length 148 instead of 84
128 bytes from 18.191.205.48: icmp_seq=1 ttl=30 time=33.555 ms
wrong total length 148 instead of 84
128 bytes from 18.191.205.48: icmp_seq=2 ttl=30 time=39.138 ms
wrong total length 148 instead of 84
128 bytes from 18.191.205.48: icmp_seq=3 ttl=30 time=27.548 ms
wrong total length 148 instead of 84
128 bytes from 18.191.205.48: icmp_seq=3 ttl=30 time=43.241 ms
wrong total length 148 instead of 84
128 bytes from 18.191.205.48: icmp_seq=4 ttl=30 time=43.241 ms
wrong total length 148 instead of 84
--- 18.191.205.48 ping statistics ---
5 packets transmitted, 5 packets received, 0.0% packet loss
round-trip min/avg/max/stddev = 27.548/37.426/43.650/6.133 ms
```

We can notice here that something looks off, for each ping we sent the reply was length 148 instead of the expected length 84 for a standard ICMP reply packet. This indicates that the server is sending back 64 additional bytes of data. Since this seems oddly suspicious, I thought we should probably capture these packets and have a look. To do this we can run the following commands in order in separate terminal tabs, this will capture the ICMP packets sent and received by our computer and write them into a pcap file that we can analyze in wireshark.

```
\\First command to run
sudo tcpdump -c 10 -vvv -XX -i any icmp -w initial.pcap
\\Then run ping again in a separate terminal tab
ping -c 5 18.191.205.48
```

## 4 Honing our Focus

Now that we have identified that the server is sending additional data back with each ICMP packet they send back in response to our ping and we have captured this packets in a pcap file we can start to take a look at the contents of this extra data. My preferred way to view packets in pcap files is to open them with the wireshark application. When I open my initial pcap file in wireshark I can the look at each packet. Here is the first reply packet:

```
Frame 2: 162 bytes on wire (1296 bits), 162 bytes captured (1296 bits) on interface en0,
Ethernet II, Src: Sagemcom_78:05:3e (b0:fc:88:78:05:3e), Dst: Apple_cf:d5:20 (80:65:7c:cf:
Internet Protocol Version 4, Src: 18.191.205.48, Dst: 192.168.1.54
Internet Control Message Protocol
     Type: 0 (Echo (ping) reply)
     Code: 0
     Checksum: 0xbb6c [correct]
      [Checksum Status: Good]
     Identifier (BE): 21581 (0x544d)
     Identifier (LE): 19796 (0x4d54)
     Sequence Number (BE): 0 (0x0000)
     Sequence Number (LE): 0 (0x0000)
                                                                   3e 08 00 45 00
bf cd 30 c0 a8
18 d4 10 00 07
11 12 13 14 15
21 22 23 24 25
31 32 33 34 35
00 00 00 49 48
06 00 00 00 be
52 61 77 20 70
20 65 78 69 66
                                                              05
12
65
10
20
30
00
8
74
65
            65 7c cf d5 20 b0 fc
94 d9 ca 00 00 1e 01
36 00 00 bb 6c 54 4d
82 08 09 0a 0b 0c 0d
17 18 19 1a 1b 1c 1d
27 28 29 2a 2b 2c 2d
37 89 50 4e 47 0d 0a
52 00 00 02 58 00 00
98 dc 00 00 00 c5 7a
6f 66 69 6c 65 20 74
                                                   88 78
20 d1
00 00
0e 0f
1e 1f
2e 2f
1a 0a
02 58
54 58
79 70
       00
01
cc
16
26
36
44
66
72
00
                                                                                                    6···l™
                                                                                                   f z TXtRaw profile t ype exif
```

As you may notice, the data in this packet looks strange, and quite suspicious. We can see just in this packet the strings "PNG", "IHDR", and "TXtRaw profile type exif". This makes me think that this is the beginning of a PNG header. Indeed if we continue looking at the reply packets we start to see various strings and characters that we might find in a PNG header. This tells us that the embedded data is a PNG image file that it seems we must reconstruct. However, PNG files can be quite large so we likely need to get much more than the first 5 ICMP reply packets. To get more packets we can simply run our commands from before updating the counts denoted by the -c flag.

```
\\First command to run

sudo tcpdump -c 5000 -vvv -XX -i any icmp -w initial.pcap

\\Then run ping again in a separate terminal tab

ping -c 10000 18.191.205.48 -i .1
```

#### 5 Reconstructing an Image

We can start playing with piecing the data together by hand to get a sense for how it might be done, to do this I simply copied the data for the first few reply packets into a text editor to look at them:

```
08090 a 0b0 c 0 d 0 e 0 f 101112131415161718191 a 1b1 c 1d1e1f 202122232425262728292 a 2b2 c 2d2e2f 3031323334353637
```

89504e470d0a1a0a000000d49484452000002580000002580806000000be6698dc0000000c57a5458745261772070726f66696c65207479706520657869660000

# 08090a0b0c0d0e0f101112131415161718191a1b1c1d1e1f202122232425262728292a2b2c2d2e2f3031 323334353637

78da6d50db0dc33008fc678a8e80013b781ce751a91b74fc6283ab24ca49067247ce0638be9f37bc3a28 09485eb4d452d020552a352b141d6dc48432e24056a4

# 08090a0b0c0d0e0f101112131415161718191a1b1c1d1e1f202122232425262728292a2b2c2d2e2f3031323334353637

602 f3 c fc 05328 a 2 db 3 0 b 5 a 3 ca 7 c 9 c 7 0 f 33 a 7 6655 3 e 19 e 9 16 c 27 a 15 a a 7 8 2 6 b d 19 f 9 b 5 c 8 f d 45 b d dec 3 a 8 8 6 1193 0 b 29 0 c 9 a 8 f 8 5 a 5 e a 7 2 1 e 6 1 3 d f 0 0 a f 5 0 3 3 d d 0 1 6 d d

We can then notice that the first 48 bytes in the data repeat every time. This is likely because these 48 bytes are part of the expected data in an ICMP reply packet. As such, we will need to cleanse the data of these 48 bytes every time as it is not part of the image data. We can do this for the data in the first three packets and then when we convert the data from hex to ascii we get:

This is the start of a PNG header so we can assume the the data will go in order of the packets. However, PNG files are large so the data needed will be too many packets to analyze and reconstruct manually. Clearly we will need to automate doing this data cleansing and construction for a large number of packets. This is our next step.

## 6 Automating Construction

Now that we have 5000 ping request and 5000 ping reply packets and we have discovered the scheme for cleansing and piecing the data together it is time to write a python script to do this for us. But first we need a way to interact with the data via python. Luckily, wireshark gives us the option to export a pcap file as JSON data, which can easily be handled in python. To export a pcap file as JSON in wireshark go to File->Export Packet Dissections->As JSON and select the All Packets option. The following extract.py script iterates over all of the JSON objects, each corresponding to one packet, and extracts and cleanses the data if the packet is a reply from the server. The code then appends the data to the collected data and writes this data to a text file when it is done.

```
import json
   with open("data.json", "r") as read_file:
       data = json.load(read_file)
   s = ""
   for x in data:
       # Only read data from ping replys
       if("icmp.no_resp" not in x["_source"]["layers"]["icmp"]):
           # Truncate first 144 characters of response data as this corresponds
           # to the first 48 bytes + 48 ':' characters between that are part of
           # the ICMP Ping data and not part of the image
12
           if "data" in x["_source"]["layers"]["icmp"]:
                print(x["_source"]["layers"]["icmp"]["data"]["data.data"])
14
                += x["_source"]["layers"]["icmp"]["data"]["data.data"][144:]
           else:
              print(x["_source"]["layers"]["frame"]["frame.number"])
17
18
   f = open("out.txt", "a")
19
   f.write(s)
   f.close()
```

#### 7 Conclusion

After pinging the server 5000 times and extracting the data from the response packets into a text file it is finally time to convert this file into an image. To do this I used the online tool cyberchef to convert this text file of bytes to a png image. To do this I used the input as file option with the render image filter with input format set to raw. This renders the following image and we can see in the flag that the server is fed up with all the pings:



bctf{pL3a\$3\_\$t0p\_p1nG1ng\_M3}

8