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

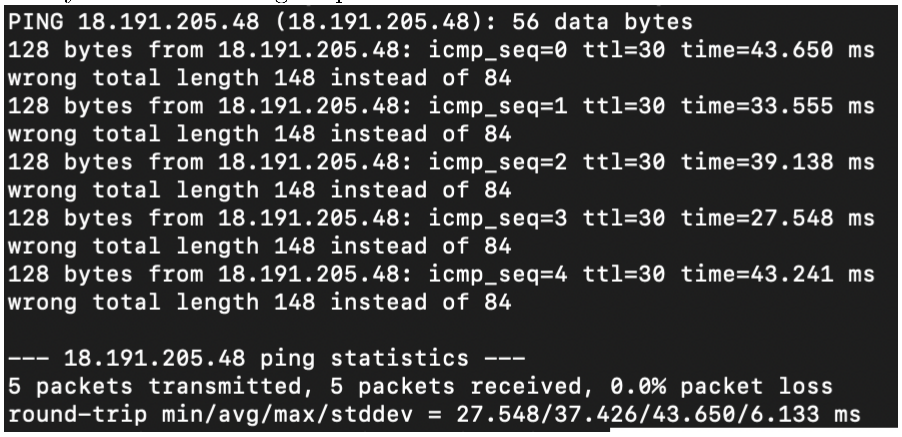
Miscellaneous challenges can often be hard to approach as the category is very broad and there may not be much information to go off of. For this particular challenge only two pieces of information were given: the name of the challenge ”pong” and the description ”18.191.205.48”. The first step was to 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:



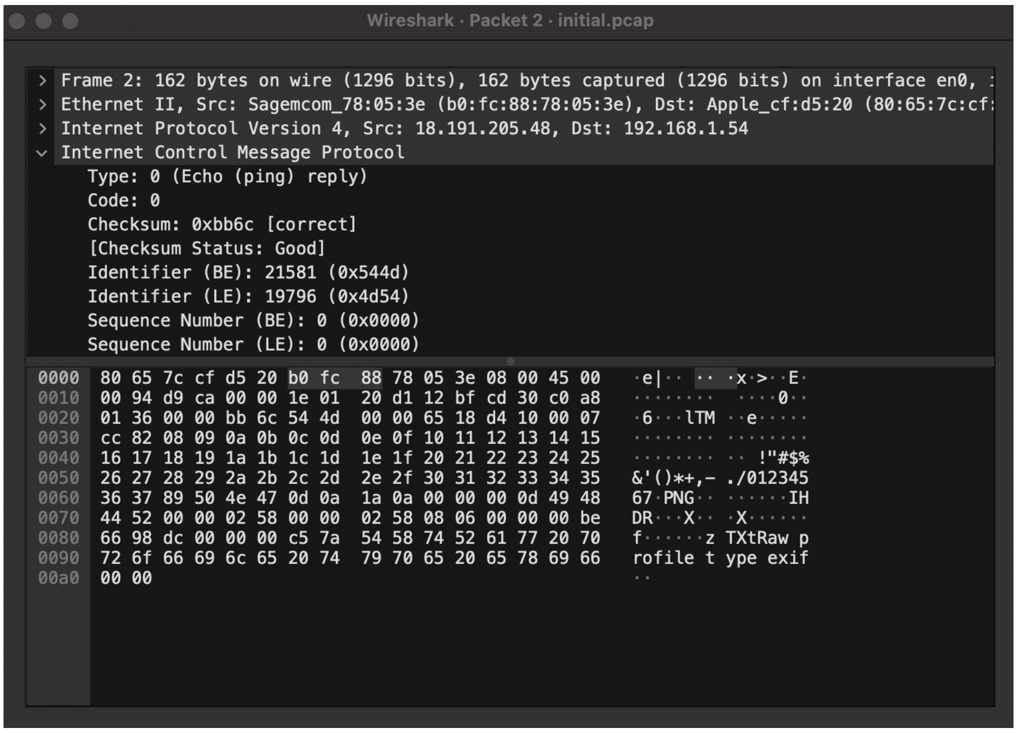
```
PING 18.191.205.48 (18.191.205.48): 56 data bytes
128 bytes from 18.191.205.48: icmp_seq=0 ttl=30 time=43.650 ms
wrong total length 148 instead of 84
128 bytes from 18.191.205.48: icmp_seq=1 ttl=30 time=33.555 ms
wrong total length 148 instead of 84
128 bytes from 18.191.205.48: icmp_seq=2 ttl=30 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=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
v 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)

0000  80 65 7c cf d5 20 b0 fc  88 78 05 3e 08 00 45 00   ·e|·· ·· ·x·>··E·
0010  00 94 d9 ca 00 00 1e 01  20 d1 12 bf cd 30 c0 a8   ········ ····0··
0020  01 36 00 00 bb 6c 54 4d  00 00 65 18 d4 10 00 07   ·6···lTM ··e·····
0030  cc 82 08 09 0a 0b 0c 0d  0e 0f 10 11 12 13 14 15   ········ ········
0040  16 17 18 19 1a 1b 1c 1d  1e 1f 20 21 22 23 24 25   ········ ·· !"#$%
0050  26 27 28 29 2a 2b 2c 2d  2e 2f 30 31 32 33 34 35   &'()*+,- ./012345
0060  36 37 89 50 4e 47 0d 0a  1a 0a 00 00 00 0d 49 48   67·PNG·· ······IH
0070  44 52 00 00 02 58 00 00  02 58 08 06 00 00 00 be   DR···X·· ·X······
0080  66 98 dc 00 00 00 c5 7a  54 58 74 52 61 77 20 70   f······z TXtRaw p
0090  72 6f 66 69 6c 65 20 74  79 70 65 20 65 78 69 66   rofile t ype exif
00a0  00 00                                              ··
```

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
```

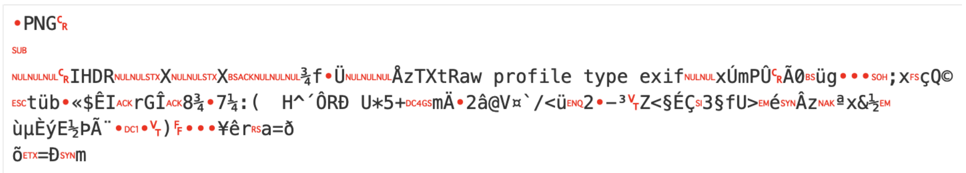
Note: this ping and packet capture can take quite a long time to run.

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:

```
08090a0b0c0d0e0f101112131415161718191a1b1c1d1e1f202122232425262728292a2b2c2d2e2f3031
323334353637
89504e470d0a1a0a0000000d4948445200000258000002580806000000be6698dc000000c57a54587452
61772070726f66696c65207479706520657869660000

08090a0b0c0d0e0f101112131415161718191a1b1c1d1e1f202122232425262728292a2b2c2d2e2f3031
323334353637
78da6d50db0dc33008fc678a8e80013b781ce751a91b74fc6283ab24ca49067247ce0638be9f37bc3a28
09485eb4d452d020552a352b141d6dc48432e24056a4

08090a0b0c0d0e0f101112131415161718191a1b1c1d1e1f202122232425262728292a2b2c2d2e2f3031
323334353637
602f3cfc05328a2db30b5a3ca7c9c70f33a766553e19e916c27a15aa7826bd19f9b5c8fd45bddec3a886
11930b290c9a8f85a5ea721e613df00af5033dd0166d
```

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:



```
•PNG CR
SUB
NULNULNUL CR IHDRNULNULSTXXNULNULSTXXBSACKNULNULNUL¾f•ÜNULNULNULÅzTXtRaw profile type exifNULNULxÚmPÛ CR Ã0BSüg•••SOH;xFSçQ©
ESCtüb•«$ÊIACKrGÎACK8¾•7¼:(  H^´ÔRÐ U*5+DC4GSmÄ•2â@V¤`/<üENQ2•–³ VT Z<§ÉÇSI3§fU>EMéSYNÂzNAKªx&½EM
ùµÈýE½ÞÃ¨•DC1• VT ) FF •••¥êrRSa=ð
õETX=ÐSYNm
```

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.

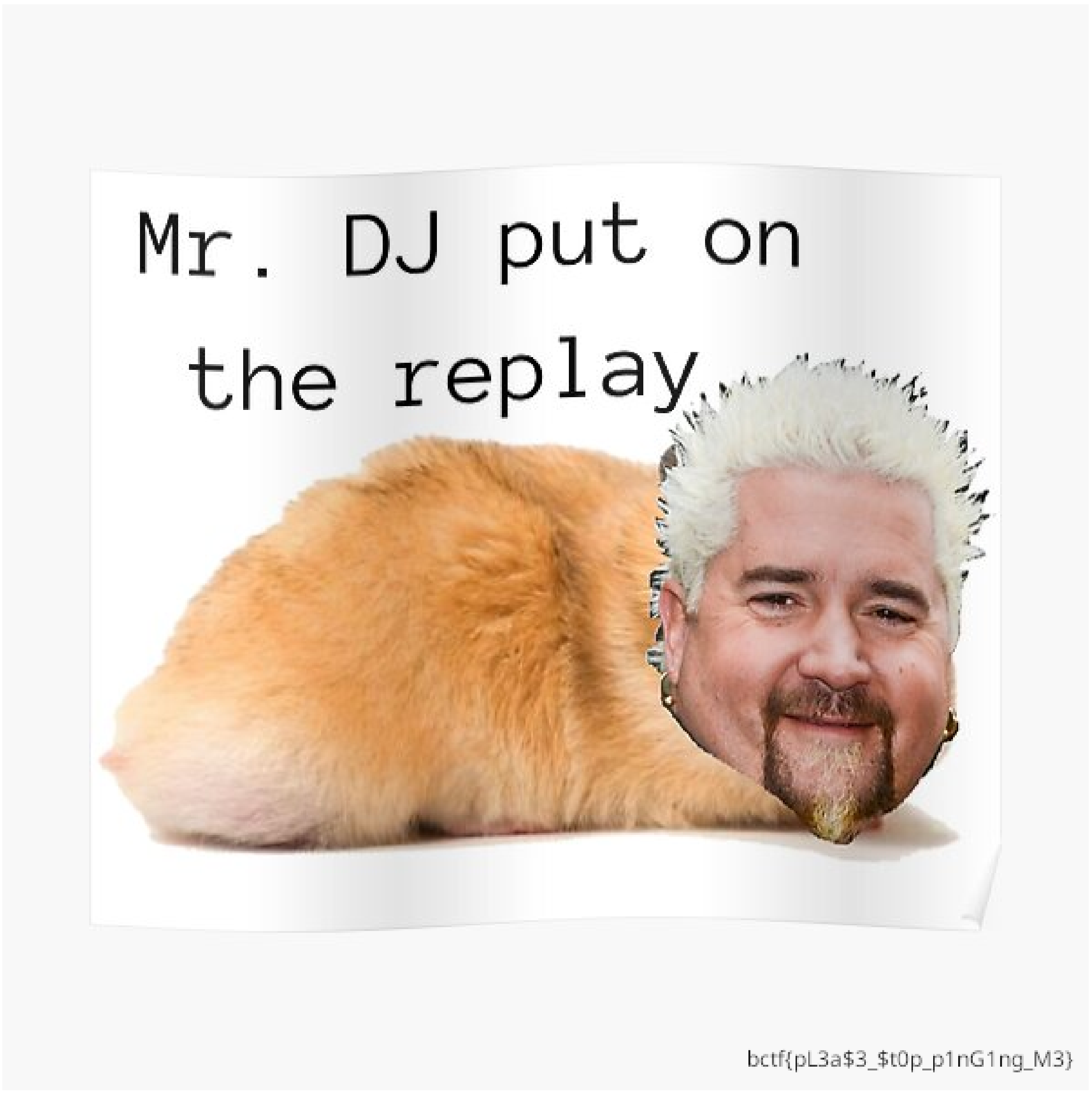
6 Automating Construction

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.

```python
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
        if "data" in x["_source"]["layers"]["icmp"]:
        #    print(x["_source"]["layers"]["icmp"]["data"]["data.data"])
            s += x["_source"]["layers"]["icmp"]["data"]["data.data"][144:]
        else:
            print(x["_source"]["layers"]["frame"]["frame.number"])

f = open("out.txt","a")
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:



8