

# 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:

---

```
1 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.

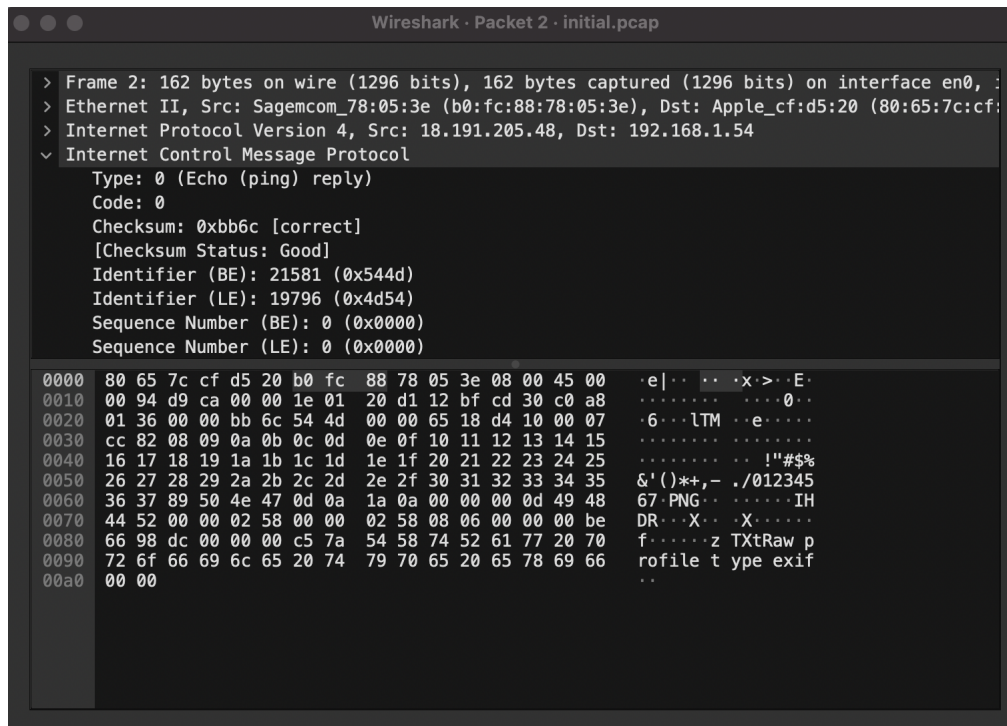
---

```
1 \\First command to run
2 sudo tcpdump -c 10 -vvv -XX -i any icmp -w initial.pcap
3 \\Then run ping again in a separate terminal tab
4 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:



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.

- 
- 1 \\First command to run
  - 2 `sudo tcpdump -c 5000 -vvv -XX -i any icmp -w initial.pcap`
  - 3 \\Then run ping again in a separate terminal tab
  - 4 `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:

```
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:



Mr. DJ put on  
the replay.



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