Write-up

Shellmates Mini-CTF 2018 – Eye see Ummm P 2

Amina BALI
SHELLMATES MEMBER
ea_bali@esi.dz

Thanks

Special thanks to KIMOUCHE Mohamed and BOUTHIBA Abderraouf who organized this Mini-CTF and for all the help they gave me and what I learnt from them.

And of course, thanks to all Shellmates members (ntouma haylin 😉).

Challenge description

Title: Eye see Ummm P 2

Category: Networking

Description:

Find the flag in this icmp capture. Not so obvious this time!

Foren4.pcap

Flag format: Shellmates{…}.

Points: 100

Difficulty: Easy to Medium

Author: Raouf

Analysis

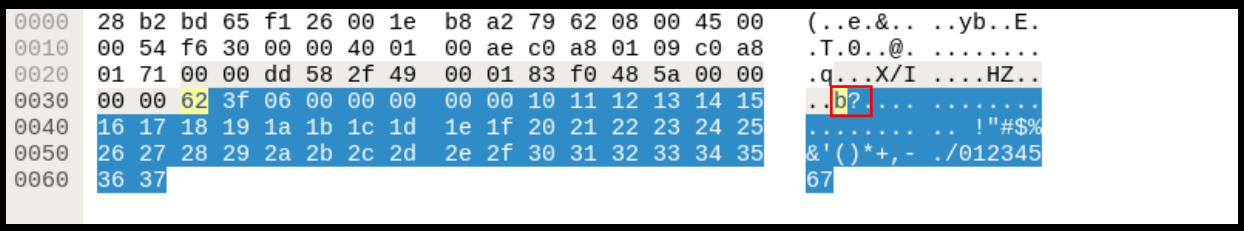
The file in the description is a pcap file that contains a capture of icmp packet exchanged between host 192.168.1.113 and host 192.168.1.9.

When we open the file with Wireshark and we start analyzing the content of the different packets we don't notice anything 'special' in the data section (which is generally the first part we check)

Information:

- ICMP stands for Internet Control Message Protocol, it is a protocol that is used by network devices to send error messages when testing connectivity. For example: an unreachable router or host in the network, an exceeded delay to deliver packets, etc. Two main used commands use ICMP protocol: ping and traceroute commands that serve to test connectivity and delivery packets between two nodes. In addition to that, the traceroute command shows the nodes in the path from source to destination.
- The data section does not import at all in ICMP protocol but if we need it we can modify it to hold data, generally it is used to make the packets larger for fragmentation testing and other network issues).
- Data in echo request and echo reply are generally the same because the reply is as its name shows a 'reply' to the request and icmp does not deal with data section so it will just reply the echo request to prove connectivity between two points in the network!

So, here is an example of the content of one of the icmp echo reply packet:



So, as we said, the data in echo reply and echo request are the same and there for analyzing data section in only request or reply would be sufficient. That's why we ordered packets by Info field in wireshark to see in order all the reply packets and then all the request packets. This will help us analyze them better.

17	39.240654	192.168.1.113	192.168.1.9	ICMP	98	Echo (ping) request id=0x2f56, seq=1/256, ttl=101 (reply in 18)
15	36.741417	192.168.1.113	192.168.1.9	ICMP	98	Echo (ping) request id=0x2f54, seq=1/256, ttl=116 (reply in 16)
13	27.825837	192.168.1.113	192.168.1.9	ICMP	98	Echo (ping) request id=0x2f50, seq=1/256, ttl=97 (reply in 14)
11	24.233772	192.168.1.113	192.168.1.9	ICMP	98	Echo (ping) request id=0x2f4f, seq=1/256, ttl=109 (reply in 12)
9	20.094082	192.168.1.113	192.168.1.9	ICMP	98	Echo (ping) request id=0x2f4e, seq=1/256, ttl=108 (reply in 10)
7	17.923342	192.168.1.113	192.168.1.9	ICMP	98	Echo (ping) request id=0x2f4d, seq=1/256, ttl=108 (reply in 8)
5	14.857570	192.168.1.113	192.168.1.9	ICMP	98	Echo (ping) request id=0x2f4c, seq=1/256, ttl=101 (reply in 6)
3	11.704562	192.168.1.113	192.168.1.9	ICMP	98	Echo (ping) request id=0x2f4b, seq=1/256, ttl=104 (reply in 4)
1	0.000000	192.168.1.113	192.168.1.9	ICMP	98	Echo (ping) request id=0x2f49, seq=1/256, ttl=83 (reply in 2)
72	280.522990	192.168.1.9	192.168.1.113	ICMP	98	Echo (ping) reply id=0x2fc8, seq=1/256, ttl=64 (request in 71)
70	276.937609	192.168.1.9	192.168.1.113	ICMP	98	Echo (ping) reply id=0x2fc7, seq=1/256, ttl=64 (request in 69)
68	272.919968	192.168.1.9	192.168.1.113	ICMP	98	Echo (ping) reply id=0x2fc5, seq=1/256, ttl=64 (request in 67)
66	269.455124	192.168.1.9	192.168.1.113	ICMP	98	Echo (ping) reply id=0x2fc4, seq=1/256, ttl=64 (request in 65)
64	265.072311	192.168.1.9	192.168.1.113	ICMP	98	Echo (ping) reply id=0x2fbf, seq=1/256, ttl=64 (request in 63)
62	252.615023	192.168.1.9	192.168.1.113	ICMP	98	Echo (ping) reply id=0x2fbd, seq=1/256, ttl=64 (request in 61)
60	249.155258	192.168.1.9	192.168.1.113	ICMP	98	Echo (ping) reply id=0x2fbc, seq=1/256, ttl=64 (request in 59)

In our case, the data section is always formed of a constant part: !"$%&'()*+;-./01234567 which has no apparent sense! And a variable part (byte 50 and 51) but this time also with no apparent meaning!

BUT then, when we really focus on the packets content, specially the echo request ones, something attracts our attention! The TTL of the request packets! They different from TTL of reply packets and each one is different from the other! We move to the section and while passing through the packets we notice that the TTL value is a printable character, the first is an "S", second an "h", third "e", etc. Seem to be the three beginning letters of Shellmates! As we continue, we notice that the flag format appears from one TTL value to another! We got it! The flag is hidden in the TTL values of request packets! Amazing, right?

Information:

TTL or Time To Live is a numeric value actually, a timer value that informs nodes in the network about how long they should hold or use a packet before it expires, this is made to avoid L3 loops (network layer loops) so that packets don't float around network forever...In the same region, TTL has the default value of 64.

In ICMP (ping and traceroute commands), TTL reprsesents the number of hops in the path from the source to the destination, whenever a node is crossed, the

TTL is incremented by one! So, with this logic, the first hop will discard a TTL of 1, the second a TTL of 2 and so on!

Resolution

Now that we analyzed the packets and figured out where the flag is hidden, we only have to find a way to extract it without too many efforts! (Lazy people here? :P).

Well, if you're not that lazy, you can just copy the characters hold in TTL of echo request packets, one by one and write them down and you got the flag! BUT, it's not really a proper way to do it! Let's find a better funny way!

What we do first, is export packets content in plaintext with the help of our dear packet analyzer Wireshark. For this, we filter packets by keeping only request packets, this is made by applying this filter: ip.dest == 192.168.1.9 because requests are sent from host 192.168.1.113 to host 192.168.1.9 (we could have used this filter ip.src == 192.168.1.113 too!) and then go Edit and Mark All Displayed, that will select all your packets! After that go export that to plaintext, go to File, Export Packet Dissections and then As Plain_Text.

You'll get a file that contains lines which look like this:

```
No.     Time           Source                Destination           Protocol Length Info
      7 17.923342      192.168.1.113         192.168.1.9           ICMP     98     Echo (ping) request  id=0x2f4d, seq=1/256, ttl=108 (reply in 8)

Frame 7: 98 bytes on wire (784 bits), 98 bytes captured (784 bits)
Ethernet II, Src: IntelCor_65:f1:26 (28:b2:bd:65:f1:26), Dst: Fortis_a2:79:62 (00:1e:b8:a2:79:62)
Internet Protocol Version 4, Src: 192.168.1.113, Dst: 192.168.1.9
Internet Control Message Protocol
```

Here, what matters to us is the TTL value! The idea is to find a way to get all TTL values, concatenate them in one string and display them as characters!

We open our terminal and move to the folder that contains the file we've just saved (packettext), and execute this command line:

```
cat packettext | grep "ttl" | cut -d "=" -f4 | cut -d "(" -f1 | awk '{printf("%c", $1)}' > flag.txt
```

Explanations:

The first command: cat packttext will get and display the content of the text file, the pipe "|" will then follow the result to the second command: grep "ttl" as an input to this one! The grep command will search for all lines that match the criteria which is containing "ttl" string. We follow the result to the third command: cut -d "=" -f4, this command will first divide the previous result into fields, each field will contain a string that is before or after the separator "=", for example:

If the input of the cut -d "=" -f3 command is: string1=string2=string3 string4

This will become first: string1 string2 string3 string4

So, 3 fields as you can see due to the separation by "=" s(-d "="), after that, the displayed result will be: string3 string4, because we told the cut command to give us the 3rd field (-f3) from the result.

Ok, now we can easily figure out that the result of our third command will be a certain number of lines which look like: 180 (reply in 8) (referred to the previous screenshot of our file content). But this is not what we want yet! Our goal is to extract only 180 without what comes after. Here comes the fourth command: cut -d "(" -f1, as you may know it will simply give us the TTL value, in our example 180. Now that we extracted all TTL values, what left to do is to convert ASCII values to text, this is what the last command does: awk '{print("%c", $1)}', for every TTL value returned by the fourth command in $1, this command will display it as character (%c). At the end, we will get all the characters concatenated in one string and stored in our file: flag.txt (the ">" role is to redirect the command result into the specified file).

Finally, we open the flag.txt file and we get our flag:

Shellmates{Th3re_1s_n0_T1me_t0_L1v3}

What we learn from this task

- Through this task, we had an overview of packets analyzing with Wireshark: ordering, filtering and exporting packets.
- We knew more about icmp protocol, ping and traceroute commands and icmp packet format, specially the data and TTL field.
- The use of some linux commands: cat, grep, cut and awk.
- The use of linux pipes and redirections.
- And finally: Pipes are magic *-*.

Thanks for reading 😊