Instituto Superior de Engenharia de Lisboa

Cibersegurança

Desafios Capture the Flag

Mestrado em Engenharia Informática de Multimédia

Pedro Gonçalves, 45890

Rodrigo Dias, 45881

Rúben Santos, 49063

Semestre de Inverno, 2021/2022

1. Introdução

Este projeto procurará solucionar alguns desafios disponibilizados pela plataforma **picoCTF** (***capture the flag***). Muitos dos desafios encaixam no âmbito de segurança na rede, e por isso utiliza-se recorrentemente software **Wireshark**.

2. Desafios

2.1. Wireshark Doo Dooo Do Doo…

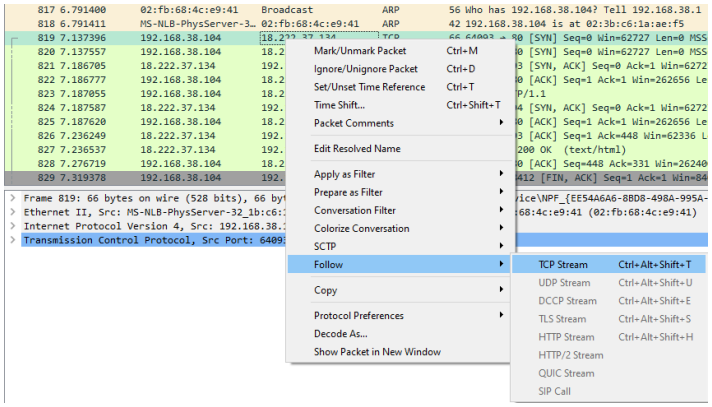
O primeiro desafio fornece um registo de pacotes capturados na rede (o ficheiro **shark1.pcapng**). Para abrir o ficheiro e analisar os pacotes, utilizar-se-á o software **Wireshark**.

806	3.812195	192.168.38.103	192.168.38.104	TCP	1514 5985 → 64029 [ACK] Seq=250753 Ack=1018068 Wi
807	3.812195	192.168.38.103	192.168.38.104	HTTP	553 HTTP/1.1 200 (application/http-kerberos-se
808	3.812226	192.168.38.104	192.168.38.103	TCP	54 64029 → 5985 [ACK] Seq=1018068 Ack=252712 Wi
809	3.816367	192.168.38.104	192.168.38.103	TCP	390 64029 → 5985 [PSH, ACK] Seq=1018068 Ack=2527
810	3.816402	192.168.38.104	192.168.38.103	HTTP	7599 POST /wsman/subscriptions/EB489718-F373-4F7F
811	3.816618	192.168.38.103	192.168.38.104	TCP	54 5985 → 64029 [ACK] Seq=252712 Ack=1025949 Wi
812	3.817286	192.168.38.103	192.168.38.104	TCP	1514 5985 → 64029 [ACK] Seq=252712 Ack=1025949 Wi
813	3.817286	192.168.38.103	192.168.38.104	HTTP	553 HTTP/1.1 200 (application/http-kerberos-se
814	3.817315	192.168.38.104	192.168.38.103	TCP	54 64029 → 5985 [ACK] Seq=1025949 Ack=254671 Wi
815	4.509702	192.168.38.105	192.168.38.104	TLSv1.2	84 Application Data
816	4.552983	192.168.38.104	192.168.38.105	TCP	54 51315 → 9000 [ACK] Seq=1 Ack=31 Win=8367 Len
817	6.791400	02:fb:68:4c:e9:41	Broadcast	ARP	56 Who has 192.168.38.104? Tell 192.168.38.1
818	6.791411	MS-NLB-PhysServer-3…	02:fb:68:4c:e9:41	ARP	42 192.168.38.104 is at 02:3b:c6:1a:ae:f5
819	7.137396	192.168.38.104	18.222.37.134	TCP	66 64093 → 80 [SYN] Seq=0 Win=62727 Len=0 MSS=8
820	7.137557	192.168.38.104	18.222.37.134	TCP	66 64094 → 80 [SYN] Seq=0 Win=62727 Len=0 MSS=8
821	7.186705	18.222.37.134	192.168.38.104	TCP	66 80 → 64093 [SYN, ACK] Seq=0 Ack=1 Win=62727
822	7.186777	192.168.38.104	18.222.37.134	TCP	54 64093 → 80 [ACK] Seq=1 Ack=1 Win=262656 Len=
823	7.187055	192.168.38.104	18.222.37.134	HTTP	501 GET / HTTP/1.1
824	7.187587	18.222.37.134	192.168.38.104	TCP	66 80 → 64094 [SYN, ACK] Seq=0 Ack=1 Win=62727
825	7.187620	192.168.38.104	18.222.37.134	TCP	54 64094 → 80 [ACK] Seq=1 Ack=1 Win=262656 Len=
826	7.236249	18.222.37.134	192.168.38.104	TCP	54 80 → 64093 [ACK] Seq=1 Ack=448 Win=62336 Len
827	7.236537	18.222.37.134	192.168.38.104	HTTP	384 HTTP/1.1 200 OK (text/html)
828	7.276719	192.168.38.104	18.222.37.134	TCP	54 64093 → 80 [ACK] Seq=448 Ack=331 Win=262400
829	7.319378	192.168.38.104	192.168.38.105	TCP	54 64091 → 8412 [FIN, ACK] Seq=1 Ack=1 Win=8400
830	7.319763	192.168.38.105	192.168.38.104	TLSv1.2	78 Application Data
831	7.319763	192.168.38.105	192.168.38.104	TCP	54 8412 → 64091 [FIN, ACK] Seq=25 Ack=2 Win=227

Ao analisar os pacotes (por alto, ainda sem inspecionar os detalhes), imediatamente identificam-se algumas linhas suspeitas, como a **817** e **818**, que representam uma comunicação através do protocolo **ARP**. Tipicamente, encontram-se pacotes **ARP** quando se inicia uma conversação, visto que é através desse protocolo que os endereços são descobertos.

Ao procurar todos os pacotes **ARP**, verifica-se que apenas existem estes dois.

Analisando a ***stream*** (fluxo de pacotes de uma conversação na rede) que vem a seguir (linha **819**), descobre-se uma linha que tem um formato parecido com a flag que se pretende encontrar. Contudo parece protegida por algum tipo de cifra.




```
HTTP/1.1 200 OK
Date: Mon, 10 Aug 2020 01:51:45 GMT
Server: Apache/2.4.29 (Ubuntu)
Last-Modified: Fri, 07 Aug 2020 00:45:02 GMT
ETag: "2f-5ac3eea4fcf01"
Accept-Ranges: bytes
Content-Length: 47
Keep-Alive: timeout=5, max=100
Connection: Keep-Alive
Content-Type: text/html

Gur synt vf cvpbPGS{c33xno00_1_f33_h_qrnqorrs}
```

Descobre-se então que a cifra utilizada é a ROT-13, uma cifra de translação em que o alfabeto de encriptação se desloca 13 posições (portanto, metade do alfabeto latino básico, com 26 letras). Decifrando, a mensagem é a seguinte:

"The flag is picoCTF{p33kab00_1_s33_u_deadbeef}"

Então, a flag é a seguinte:

"picoCTF{p33kab00_1_s33_u_deadbeef}"

2.2. Wireshark Twoo Twooo Two Twoo

O segundo desafio fornece outro registo de pacotes capturados na rede (o ficheiro **shark2.pcapng**). Da mesma forma que o desafio anterior, se se inspecionarem os diferentes streams contidos no registo, verifica-se que existem várias flags escondidas da mesma forma, todas diferentes, e desta vez nem estão encriptadas.

```
GET /flag HTTP/1.1
Host: 18.217.1.57
Connection: keep-alive
Cache-Control: max-age=0
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,i
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9

HTTP/1.0 200 OK
Content-Type: text/html; charset=utf-8
Content-Length: 73
Server: Werkzeug/1.0.1 Python/3.6.9
Date: Mon, 10 Aug 2020 01:39:40 GMT

picoCTF{09082a0313e16fc36f8076ff86e54e83048a8568f5c2294fea5fb3bcd212e7f2}
```

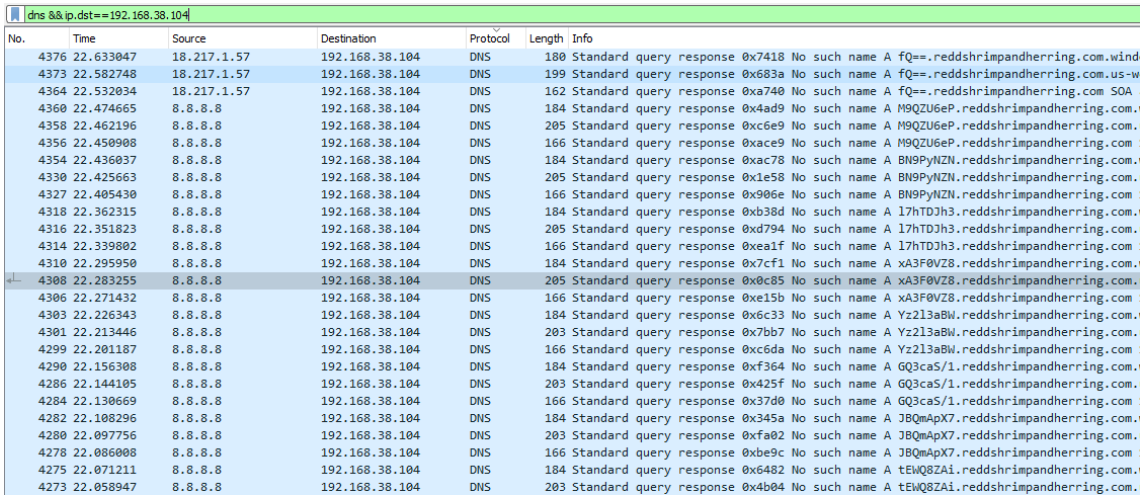
Isto leva a crer que não passam de uma ratoeira. Por essa razão, é preferível investigar os pacotes um pouco mais. Verifica-se que existem vários pacotes DNS, todos com um domínio muito suspeito, com o formato:

"xA3F0VZ8.reddshrimpandherring.com"

Em que "**xA3F0VZ8"** assume diferentes valores.

699	3.647278	216.58.194.196	192.168.38.104	GQUIC	1392 Payload (Encrypted), PKN: 2, CID: 11558175087779634948
698	3.647278	216.58.194.196	192.168.38.104	GQUIC	1392 Rejection, PKN: 1, CID: 11558175087779634948
697	3.631224	192.168.38.104	216.58.194.196	GQUIC	1392 Client Hello, PKN: 1, CID: 11558175087779634948
4376	22.633047	18.217.1.57	192.168.38.104	DNS	180 Standard query response 0x7418 No such name A fQ==.reddshrimpandherring.com.winc
4374	22.583745	192.168.38.104	18.217.1.57	DNS	105 Standard query 0x7418 A fQ==.reddshrimpandherring.com.windomain.local
4373	22.582748	18.217.1.57	192.168.38.104	DNS	199 Standard query response 0x683a No such name A fQ==.reddshrimpandherring.com.us-w
4365	22.533184	192.168.38.104	18.217.1.57	DNS	127 Standard query 0x683a A fQ==.reddshrimpandherring.com.us-west-1.ec2-utilities.am
4364	22.532034	18.217.1.57	192.168.38.104	DNS	162 Standard query response 0xa740 No such name A fQ==.reddshrimpandherring.com SOA
4361	22.481648	192.168.38.104	18.217.1.57	DNS	89 Standard query 0xa740 A fQ==.reddshrimpandherring.com
4360	22.474665	8.8.8.8	192.168.38.104	DNS	184 Standard query response 0x4ad9 No such name A M9QZU6eP.reddshrimpandherring.com.
4359	22.463138	192.168.38.104	8.8.8.8	DNS	109 Standard query 0x4ad9 A M9QZU6eP.reddshrimpandherring.com.windomain.local
4358	22.462196	8.8.8.8	192.168.38.104	DNS	205 Standard query response 0xc6e9 No such name A M9QZU6eP.reddshrimpandherring.com.
4357	22.451860	192.168.38.104	8.8.8.8	DNS	131 Standard query 0xc6e9 A M9QZU6eP.reddshrimpandherring.com.us-west-1.ec2-utilitie
4356	22.450908	8.8.8.8	192.168.38.104	DNS	166 Standard query response 0xace9 No such name A M9QZU6eP.reddshrimpandherring.com
4355	22.437065	192.168.38.104	8.8.8.8	DNS	93 Standard query 0xace9 A M9QZU6eP.reddshrimpandherring.com
4354	22.436037	8.8.8.8	192.168.38.104	DNS	184 Standard query response 0xac78 No such name A BN9PyNZN.reddshrimpandherring.com.
4334	22.426618	192.168.38.104	8.8.8.8	DNS	109 Standard query 0xac78 A BN9PyNZN.reddshrimpandherring.com.windomain.local
4330	22.425663	8.8.8.8	192.168.38.104	DNS	205 Standard query response 0x1e58 No such name A BN9PyNZN.reddshrimpandherring.com.

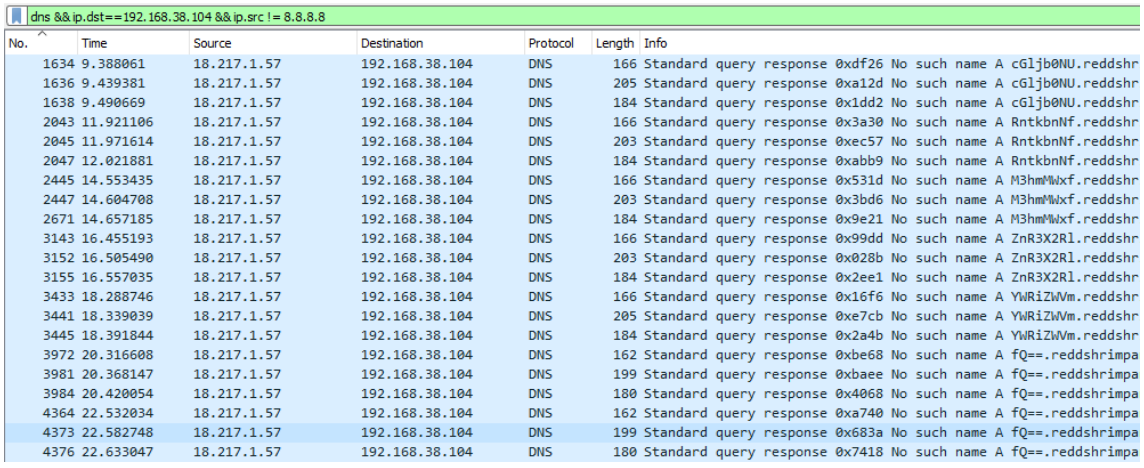
Verifica-se também que estes pacotes **DNS** ocorrem sempre numa conversação entre o endereço **192.168.38.104** e outro endereço. Se se filtrarem os pacotes, de forma a visualizar apenas os DNS e aqueles que contêm o endereço **192.168.38.104** como IP de destino, observa-se o seguinte:



dns && ip.dst==192.168.38.104

No.	Time	Source	Destination	Protocol	Length	Info
4376	22.633047	18.217.1.57	192.168.38.104	DNS	180	Standard query response 0x7418 No such name A fQ==.reddshrimpandherring.com.wind
4373	22.582748	18.217.1.57	192.168.38.104	DNS	199	Standard query response 0x683a No such name A fQ==.reddshrimpandherring.com.us-w
4364	22.532034	18.217.1.57	192.168.38.104	DNS	162	Standard query response 0xa740 No such name A fQ==.reddshrimpandherring.com SOA
4360	22.474665	8.8.8.8	192.168.38.104	DNS	184	Standard query response 0x4ad9 No such name A M9QZU6eP.reddshrimpandherring.com.
4358	22.462196	8.8.8.8	192.168.38.104	DNS	205	Standard query response 0xc6e9 No such name A M9QZU6eP.reddshrimpandherring.com.
4356	22.450908	8.8.8.8	192.168.38.104	DNS	166	Standard query response 0xace9 No such name A M9QZU6eP.reddshrimpandherring.com
4354	22.436037	8.8.8.8	192.168.38.104	DNS	184	Standard query response 0xac78 No such name A BN9PyNZN.reddshrimpandherring.com.
4330	22.425663	8.8.8.8	192.168.38.104	DNS	205	Standard query response 0x1e58 No such name A BN9PyNZN.reddshrimpandherring.com.
4327	22.405430	8.8.8.8	192.168.38.104	DNS	166	Standard query response 0x906e No such name A BN9PyNZN.reddshrimpandherring.com
4318	22.362315	8.8.8.8	192.168.38.104	DNS	184	Standard query response 0xb38d No such name A l7hTDJh3.reddshrimpandherring.com.
4316	22.351823	8.8.8.8	192.168.38.104	DNS	205	Standard query response 0xd794 No such name A l7hTDJh3.reddshrimpandherring.com.
4314	22.339802	8.8.8.8	192.168.38.104	DNS	166	Standard query response 0xea1f No such name A l7hTDJh3.reddshrimpandherring.com
4310	22.295950	8.8.8.8	192.168.38.104	DNS	184	Standard query response 0x7cf1 No such name A xA3F0VZ8.reddshrimpandherring.com.
4308	22.283255	8.8.8.8	192.168.38.104	DNS	205	Standard query response 0x0c85 No such name A xA3F0VZ8.reddshrimpandherring.com.
4306	22.271432	8.8.8.8	192.168.38.104	DNS	166	Standard query response 0xe15b No such name A xA3F0VZ8.reddshrimpandherring.com
4303	22.226343	8.8.8.8	192.168.38.104	DNS	184	Standard query response 0x6c33 No such name A Yz2l3aBW.reddshrimpandherring.com.
4301	22.213446	8.8.8.8	192.168.38.104	DNS	203	Standard query response 0x7bb7 No such name A Yz2l3aBW.reddshrimpandherring.com.
4299	22.201187	8.8.8.8	192.168.38.104	DNS	166	Standard query response 0xc6da No such name A Yz2l3aBW.reddshrimpandherring.com
4290	22.156308	8.8.8.8	192.168.38.104	DNS	184	Standard query response 0xf364 No such name A GQ3caS/1.reddshrimpandherring.com.
4286	22.144105	8.8.8.8	192.168.38.104	DNS	203	Standard query response 0x425f No such name A GQ3caS/1.reddshrimpandherring.com.
4284	22.130669	8.8.8.8	192.168.38.104	DNS	166	Standard query response 0x37d0 No such name A GQ3caS/1.reddshrimpandherring.com
4282	22.108296	8.8.8.8	192.168.38.104	DNS	184	Standard query response 0x345a No such name A JBQmApX7.reddshrimpandherring.com.
4280	22.097756	8.8.8.8	192.168.38.104	DNS	203	Standard query response 0xfa02 No such name A JBQmApX7.reddshrimpandherring.com.
4278	22.086008	8.8.8.8	192.168.38.104	DNS	166	Standard query response 0xbe9c No such name A JBQmApX7.reddshrimpandherring.com
4275	22.071211	8.8.8.8	192.168.38.104	DNS	184	Standard query response 0x6482 No such name A tEWQ8ZAi.reddshrimpandherring.com.
4273	22.058947	8.8.8.8	192.168.38.104	DNS	203	Standard query response 0x4b04 No such name A tEWQ8ZAi.reddshrimpandherring.com.

Existem vários pacotes com o endereço de origem **8.8.8.8**, que é o serviço de **DNS** público disponibilizado pela **Google**. Desprezando esses, e ordenando:



dns && ip.dst==192.168.38.104 && ip.src != 8.8.8.8

No.	Time	Source	Destination	Protocol	Length	Info
1634	9.388061	18.217.1.57	192.168.38.104	DNS	166	Standard query response 0xdf26 No such name A cGljb0NU.reddshr
1636	9.439381	18.217.1.57	192.168.38.104	DNS	205	Standard query response 0xa12d No such name A cGljb0NU.reddshr
1638	9.490669	18.217.1.57	192.168.38.104	DNS	184	Standard query response 0x1dd2 No such name A cGljb0NU.reddshr
2043	11.921106	18.217.1.57	192.168.38.104	DNS	166	Standard query response 0x3a30 No such name A RntkbnNf.reddshr
2045	11.971614	18.217.1.57	192.168.38.104	DNS	203	Standard query response 0xec57 No such name A RntkbnNf.reddshr
2047	12.021881	18.217.1.57	192.168.38.104	DNS	184	Standard query response 0xabb9 No such name A RntkbnNf.reddshr
2445	14.553435	18.217.1.57	192.168.38.104	DNS	166	Standard query response 0x531d No such name A M3hmMWxf.reddshr
2447	14.604708	18.217.1.57	192.168.38.104	DNS	203	Standard query response 0x3bd6 No such name A M3hmMWxf.reddshr
2671	14.657185	18.217.1.57	192.168.38.104	DNS	184	Standard query response 0x9e21 No such name A M3hmMWxf.reddshr
3143	16.455193	18.217.1.57	192.168.38.104	DNS	166	Standard query response 0x99dd No such name A ZnR3X2Rl.reddshr
3152	16.505490	18.217.1.57	192.168.38.104	DNS	203	Standard query response 0x028b No such name A ZnR3X2Rl.reddshr
3155	16.557035	18.217.1.57	192.168.38.104	DNS	184	Standard query response 0x2ee1 No such name A ZnR3X2Rl.reddshr
3433	18.288746	18.217.1.57	192.168.38.104	DNS	166	Standard query response 0x16f6 No such name A YWRiZWVm.reddshr
3441	18.339039	18.217.1.57	192.168.38.104	DNS	205	Standard query response 0xe7cb No such name A YWRiZWVm.reddshr
3445	18.391844	18.217.1.57	192.168.38.104	DNS	184	Standard query response 0x2a4b No such name A YWRiZWVm.reddshr
3972	20.316608	18.217.1.57	192.168.38.104	DNS	162	Standard query response 0xbe68 No such name A fQ==.reddshrimpa
3981	20.368147	18.217.1.57	192.168.38.104	DNS	199	Standard query response 0xbaee No such name A fQ==.reddshrimpa
3984	20.420054	18.217.1.57	192.168.38.104	DNS	180	Standard query response 0x4068 No such name A fQ==.reddshrimpa
4364	22.532034	18.217.1.57	192.168.38.104	DNS	162	Standard query response 0xa740 No such name A fQ==.reddshrimpa
4373	22.582748	18.217.1.57	192.168.38.104	DNS	199	Standard query response 0x683a No such name A fQ==.reddshrimpa
4376	22.633047	18.217.1.57	192.168.38.104	DNS	180	Standard query response 0x7418 No such name A fQ==.reddshrimpa

Obtém-se então uma lista mais limitada. A ideia será concatenar os prefixos do domínio **"reddshrimpandherring.com"** obtendo aquilo que se suspeita tratar de uma mensagem cifrada. O resultado da concatenação é:

"cGljb0NURntkbnNfM3hmMWxfZnR3X2RlYWRiZWVmfQ=="

Ora, é sabido que as mensagens que utilizam o ***encoder* base64** utilizam **"=="** como um sufixo especial. Descodificando:

"picoCTF{dns_3xf1l_ftw_deadbeef}"

2.3. WPA-ing Out

Para este terceiro desafio, sabe-se que a ***flag*** secreta é uma password contida na wordlist **Rockyou.txt**. Utilizando a ferramenta **Aircrack-ng**, torna-se muito fácil:

```
Reading packets, please wait...
Opening wpa-ing_out.pcap
Read 23523 packets.

1 potential targets

                               Aircrack-ng 1.6

      [00:00:00] 1286/10303727 keys tested (3649.60 k/s)

      Time left: 47 minutes, 3 seconds                           0.01%

                          KEY FOUND! [ mickeymouse ]


      Master Key     : 61 64 B9 5E FC 6F 41 70 70 81 F6 40 80 9F AF B1
                       4A 9E C5 C4 E1 67 B8 AB 58 E3 E8 8E E6 66 EB 11

      Transient Key  : 26 85 7B AC DD 2C 44 E6 06 18 03 B0 0F F2 75 A2
                       32 63 F7 35 74 2D 18 10 1C 25 F9 14 BC 41 DA 58
                       52 48 86 B0 D6 14 89 F6 77 00 1E 99 0D 1C 97 00
                       00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

      EAPOL HMAC     : 65 2F 6C 0E 75 F0 49 27 6A AA 6A 06 A7 24 B9 A9
```

A flag é:

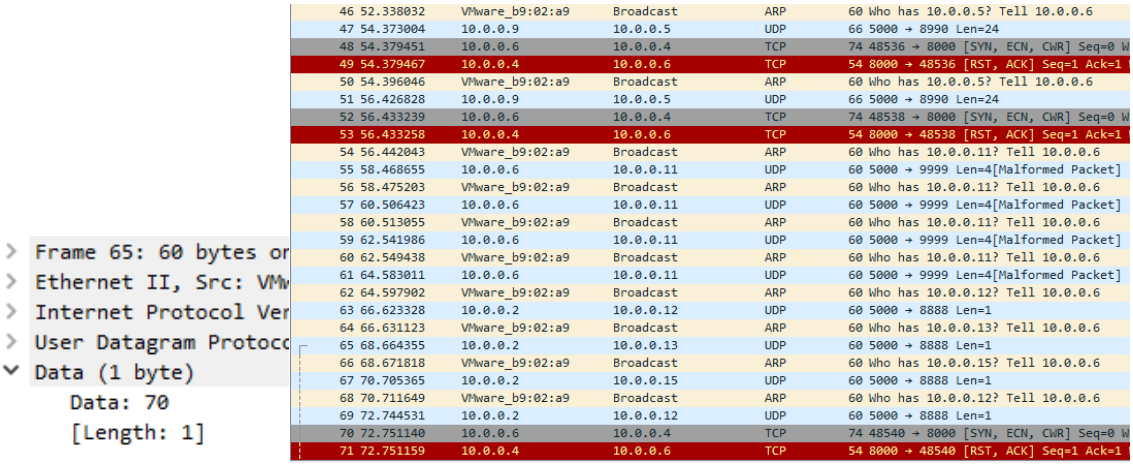
"picoCTF{mickeymouse}"

2.4. Shark on Wire 1

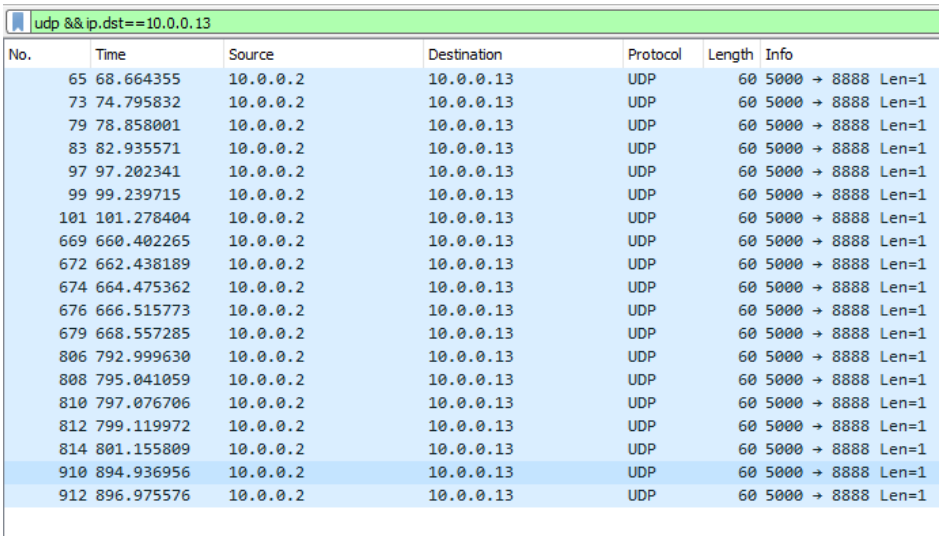
Desta vez, é fornecido um ficheiro **capture.pcap**, que será também analisado no **Wireshark**. Este ficheiro conta com a captura de centenas de pacotes **UDP**, que não parecem conter qualquer informação que aponte na direção da ***flag***.

403	396.982209	10.0.0.2	10.0.0.22	UDP	66 5000 → 8990 Len=24
404	397.001665	VMware_b9:02:a9	Broadcast	ARP	60 Who has 10.0.0.5? Tell 10.0.0.6
405	399.051415	10.0.0.9	10.0.0.5	UDP	60 5000 → 8990 Len=1
406	399.070387	VMware_b9:02:a9	Broadcast	ARP	60 Who has 10.0.0.22? Tell 10.0.0.6
407	401.104745	10.0.0.2	10.0.0.22	UDP	66 5000 → 8990 Len=24
408	401.133303	VMware_b9:02:a9	Broadcast	ARP	60 Who has 10.0.0.5? Tell 10.0.0.6
409	403.175834	10.0.0.9	10.0.0.5	UDP	60 5000 → 8990 Len=1
410	403.186092	VMware_b9:02:a9	Broadcast	ARP	60 Who has 10.0.0.22? Tell 10.0.0.6
411	405.220852	10.0.0.2	10.0.0.22	UDP	66 5000 → 8990 Len=24
412	405.242823	VMware_b9:02:a9	Broadcast	ARP	60 Who has 10.0.0.5? Tell 10.0.0.6
413	407.279123	10.0.0.9	10.0.0.5	UDP	60 5000 → 8990 Len=1
414	407.300614	VMware_b9:02:a9	Broadcast	ARP	60 Who has 10.0.0.22? Tell 10.0.0.6
415	409.328439	10.0.0.2	10.0.0.22	UDP	66 5000 → 8990 Len=24
416	409.337148	VMware_b9:02:a9	Broadcast	ARP	60 Who has 10.0.0.5? Tell 10.0.0.6
417	411.364223	10.0.0.9	10.0.0.5	UDP	60 5000 → 8990 Len=1
418	411.374473	VMware_b9:02:a9	Broadcast	ARP	60 Who has 10.0.0.22? Tell 10.0.0.6
419	413.411100	10.0.0.2	10.0.0.22	UDP	66 5000 → 8990 Len=24
420	413.440129	VMware_b9:02:a9	Broadcast	ARP	60 Who has 10.0.0.5? Tell 10.0.0.6
421	415.462092	10.0.0.9	10.0.0.5	UDP	60 5000 → 8990 Len=1
422	415.468218	VMware_b9:02:a9	Broadcast	ARP	60 Who has 10.0.0.22? Tell 10.0.0.6
423	417.505996	10.0.0.2	10.0.0.22	UDP	66 5000 → 8990 Len=24
424	417.525166	VMware_b9:02:a9	Broadcast	ARP	60 Who has 10.0.0.5? Tell 10.0.0.6
425	419.559490	10.0.0.9	10.0.0.5	UDP	60 5000 → 8990 Len=1
426	419.595014	VMware_b9:02:a9	Broadcast	ARP	60 Who has 10.0.0.22? Tell 10.0.0.6
427	421.623639	10.0.0.2	10.0.0.22	UDP	66 5000 → 8990 Len=24
428	421.646704	VMware_b9:02:a9	Broadcast	ARP	60 Who has 10.0.0.5? Tell 10.0.0.6

Contudo, ao analisar com mais atenção, verifica-se que alguns pacotes suspeitos, nomeadamente os que têm um IP de destino, por exemplo, igual a **10.0.0.13** ou a **10.0.0.12**, apresentam um campo **"data"** com dimensão de **1 byte**.



Ao filtrar os pacotes, de forma a visualizar apenas os que têm endereço de destino **10.0.0.13**, obtém-se a seguinte lista.



Concatenando os valores (em hexadecimal) dos bytes dos vários pacotes, obtém-se:

7069636f4354467b4e30745f615f664c61677d

Convertendo para caracteres **ASCII**:

picoCTF{N0t_a_fLag}

Trata-se de uma ratoeira. Contudo, se o mesmo procedimento for realizado, mas para pacotes com o endereço de destino **10.0.0.12**, obtém-se:

7069636f4354467b5374615433313335355f363336663665366576 57d

Convertendo para caracteres **ASCII**, encontramos finalmente a **flag** secreta:

picoCTF{StaT31355_636f6e6e}

2.5. Shark on Wire 2

Este desafio é semelhante ao anterior. É novamente fornecido um ficheiro **capture.pcap**, que será mais uma vez analisado no **Wireshark**. Da mesma forma que o desafio anterior, este ficheiro conta com a captura de centenas de pacotes **UDP**.

Ao analisar as ***streams* UDP**, encontram-se diversas mensagens muito interessantes, mas nenhuma aponta na direção da verdadeira **flag**.

```
picoCTF Sure is fun!picoCTF Sure is fun!picoCTF Sure is fun!picoCTF Sure
is fun!picoCTF Sure is fun!picoCTF Sure is fun!picoCTF Sure is fun!
picoCTF Sure is fun!picoCTF Sure is fun!aaaaa
```

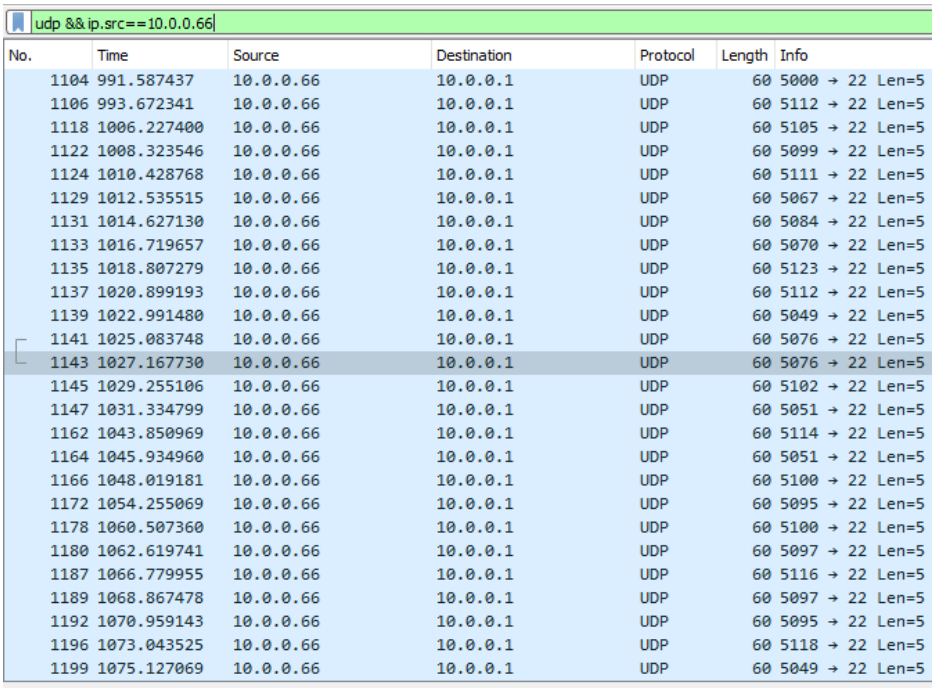
```
I really want to find some picoCTF flagsI really want to find some
picoCTF flagsI really want to find some picoCTF flagsI really want to
find some picoCTF flagsI really want to find some picoCTF flagsI really
want to find some picoCTF flagsI really want to find some picoCTF flagsI
really want to find some picoCTF flagsI really want to find some picoCTF
flags
```

Com exceção de duas mensagens:

```
start
```

```
end
```

Após a primeira mensagem, que parece indicar o começo de algo, e antes da segunda mensagem, que parece indicar o final de algo, existem várias mensagens vindas do **IP** de origem **10.0.0.66**. Filtrando os pacotes com origem nesse endereço, obtém-se:


udp && ip.src==10.0.0.66

No.	Time	Source	Destination	Protocol	Length	Info
1104	991.587437	10.0.0.66	10.0.0.1	UDP	60	5000 → 22 Len=5
1106	993.672341	10.0.0.66	10.0.0.1	UDP	60	5112 → 22 Len=5
1118	1006.227400	10.0.0.66	10.0.0.1	UDP	60	5105 → 22 Len=5
1122	1008.323546	10.0.0.66	10.0.0.1	UDP	60	5099 → 22 Len=5
1124	1010.428768	10.0.0.66	10.0.0.1	UDP	60	5111 → 22 Len=5
1129	1012.535515	10.0.0.66	10.0.0.1	UDP	60	5067 → 22 Len=5
1131	1014.627130	10.0.0.66	10.0.0.1	UDP	60	5084 → 22 Len=5
1133	1016.719657	10.0.0.66	10.0.0.1	UDP	60	5070 → 22 Len=5
1135	1018.807279	10.0.0.66	10.0.0.1	UDP	60	5123 → 22 Len=5
1137	1020.899193	10.0.0.66	10.0.0.1	UDP	60	5112 → 22 Len=5
1139	1022.991480	10.0.0.66	10.0.0.1	UDP	60	5049 → 22 Len=5
1141	1025.083748	10.0.0.66	10.0.0.1	UDP	60	5076 → 22 Len=5
1143	1027.167730	10.0.0.66	10.0.0.1	UDP	60	5076 → 22 Len=5
1145	1029.255106	10.0.0.66	10.0.0.1	UDP	60	5102 → 22 Len=5
1147	1031.334799	10.0.0.66	10.0.0.1	UDP	60	5051 → 22 Len=5
1162	1043.850969	10.0.0.66	10.0.0.1	UDP	60	5114 → 22 Len=5
1164	1045.934960	10.0.0.66	10.0.0.1	UDP	60	5051 → 22 Len=5
1166	1048.019181	10.0.0.66	10.0.0.1	UDP	60	5100 → 22 Len=5
1172	1054.255069	10.0.0.66	10.0.0.1	UDP	60	5095 → 22 Len=5
1178	1060.507360	10.0.0.66	10.0.0.1	UDP	60	5100 → 22 Len=5
1180	1062.619741	10.0.0.66	10.0.0.1	UDP	60	5097 → 22 Len=5
1187	1066.779955	10.0.0.66	10.0.0.1	UDP	60	5116 → 22 Len=5
1189	1068.867478	10.0.0.66	10.0.0.1	UDP	60	5097 → 22 Len=5
1192	1070.959143	10.0.0.66	10.0.0.1	UDP	60	5095 → 22 Len=5
1196	1073.043525	10.0.0.66	10.0.0.1	UDP	60	5118 → 22 Len=5
1199	1075.127069	10.0.0.66	10.0.0.1	UDP	60	5049 → 22 Len=5

Destaca-se imediatamente o facto de todos os pacotes terem diferentes portos associados. Sendo que todos começam com o algarismo **5**, tomar-se-á nota dos três restantes algarismos de cada porto, para, posteriormente, converter tudo para caracteres **ASCII**, tal como foi realizado no desafio anterior.

000 112 105 099 111 067 084 070 123 112 049 076 076 102 051 114 051 100 095 100 097 116 097 095 118 049 097 095 115 116 051 103 048 125

Convertendo para caracteres **ASCII**:

picoCTF{p1LLf3r3d_data_v1a_st3g0}

2.6. Lets Warm Up

Fez-se este desafio para preparar a resolução do desafio **'Nice Netcat...'** e consiste simplesmente em ver qual o símbolo **ASCII** que corresponde ao valor hexadecimal **0x70**. Como tal, visualizando numa tabela **ASCII**, o símbolo correspondente é a letra **p**:

picoCTF{p}

2.7. What's a Netcat?

Este desafio serviu também para preparar o desafio **'Nice Netcat…'** e foi apenas necessário abrir um terminal **Linux** para correr o comando **nc**, abrindo uma conexão com **'jupiter.challenges.picoctf.org'** no porto **41120**, obtendo a ***flag***:

picoCTF{nEtCat_Mast3ry_3214be47}

2.8. Nice Netcat…

Para este desafio foi pedido executar o comando **'nc mercury.picoctf.net 49039'** num terminal Linux e obteve-se a seguinte sequência de números inteiros:

112 105 099 111 067 084 070 123 103 048 048 100 095 107 049 116 116 121 033 095 110 049 099 051 095 107 049 116 116 121 033 095 051 100 056 052 101 100 099 056 125

Assumiu-se que estes números pudessem corresponder a um código **ASCII**, por isso utilizou-se outra fez a tabela **ASCII** para fazer a conversão, mas desta vez converteu-se a partir de **decimal** e não **hexadecimal**, obtendo a ***flag***:

picoCTF{g00d_k1tty!_n1c3_k1tty!_3d84edc8}

2.9. Strings it

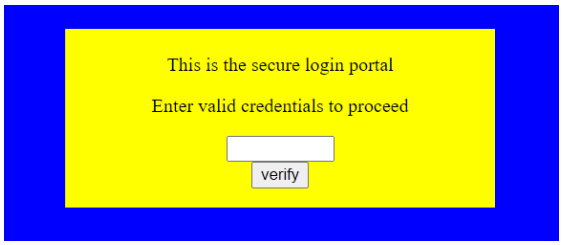
Neste desafio, é fornecido um ficheiro com o nome **strings** e o objetivo é encontrar a ***flag*** escondida. Começa-se por descarregar o respetivo ficheiro e ao abrir observa-se que o ficheiro contém uma enorme lista de ***strings***.

Ao utilizar a documentação do programa **strings** descobre-se a opção **-a** que mostra todo conteúdo do ficheiro, e com o **grep** é possível filtrar esse conteúdo, uma vez que se sabe que a ***flag*** começa sempre pelo prefixo **"pico"**.

```
aluno49063-picoctf@webshell:~$ wget https://jupiter.challenges.picoctf.org/static/5bd86036f013ac3b9c958499adf3e2e2/strings
--2022-01-21 17:29:23--  https://jupiter.challenges.picoctf.org/static/5bd86036f013ac3b9c958499adf3e2e2/strings
Resolving jupiter.challenges.picoctf.org (jupiter.challenges.picoctf.org)... 3.131.60.8
Connecting to jupiter.challenges.picoctf.org (jupiter.challenges.picoctf.org)|3.131.60.8|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 776032 (758K) [application/octet-stream]
Saving to: 'strings.2'

strings.2                     100%[=========================================================>] 757.84K  1.86MB/s    in 0.4s

2022-01-21 17:29:23 (1.86 MB/s) - 'strings.2' saved [776032/776032]

aluno49063-picoctf@webshell:~$ strings -a strings | grep "pico"
picoCTF{5tRIng5_1T_827aee91}
aluno49063-picoctf@webshell:~$
```

2.10. Dont-Use-Client-Side

Neste desafio, observa-se uma página de "***login*** de portal seguro".



Após algumas tentativas de inserção de credenciais, procede-se a inspeção da página. No código de validação das credenciais encontra-se a seguinte função:

```
function verify() {
  checkpass = document.getElementById("pass").value;
  split = 4;
  if (checkpass.substring(0, split) == 'pico') {
    if (checkpass.substring(split*6, split*7) == 'a3c8') {
      if (checkpass.substring(split, split*2) == 'CTF{') {
       if (checkpass.substring(split*4, split*5) == 'ts_p') {
        if (checkpass.substring(split*3, split*4) == 'lien') {
          if (checkpass.substring(split*5, split*6) == 'lz_1') {
            if (checkpass.substring(split*2, split*3) == 'no_c') {
              if (checkpass.substring(split*7, split*8) == '9}') {
                alert("Password Verified")
                }
              }
            }
```

3. Conclusão

Conclui-se que a plataforma **picoCTF** constitui uma forma empolgante e pedagógica de aprender sobre vários conceitos na área da informática, incluindo redes. Muitos deste desafios expoem possíveis superfícies de ataque nesse âmbito.