BOMBA BINARIA

MEMORIA

-Stage 1:

En primer lugar, para averiguar la contraseña de esta etapa, nos introducimos mediante F11 en el código de la instrucción:

```
00405510 55                  push        ebp    ≤ 1 ms transcurridos
00405511 8B EC               mov         ebp,esp
00405513 81 EC EC 03 00 00   sub         esp,3ECh
00405519 C7 45 FC E8 03 00 00 mov        dword ptr [ebp-4],3E8h
00405520 6A 00               push        0
00405522 68 E8 03 00 00      push        3E8h
00405527 8D 85 14 FC FF FF   lea         eax,[ebp-3ECh]
0040552D 50                  push        eax
0040552E B9 B0 6B 4F 00      mov         ecx,offset std::cin (04F6BB0h)
00405533 E8 28 47 00 00      call        std::basic_istream<char,std::char_traits<char> >::getline (0409C60h)
00405538 68 14 5A 4C 00      push        4C5A14h
0040553D 8D 8D 14 FC FF FF   lea         ecx,[ebp-3ECh]
00405543 51                  push        ecx
00405544 E8 77 DF 07 00      call        strcmp (04834C0h)
00405549 83 C4 08            add         esp,8
0040554C 85 C0               test        eax,eax
0040554E 74 0C               je          Stage1+4Ch (040555Ch)
00405550 6A 01               push        1
00405552 E8 F9 FE FF FF      call        Explode (0405450h)
00405557 83 C4 04            add         esp,4
0040555A EB 0F               jmp         Stage1+5Bh (040556Bh)
0040555C 68 20 5A 4C 00      push        4C5A20h
00405561 6A 01               push        1
00405563 E8 58 FF FF FF      call        Defuse (04054C0h)
00405568 83 C4 08            add         esp,8
0040556B 8B E5               mov         esp,ebp
0040556D 5D                  pop         ebp
0040556E C3                  ret
```

Podemos observar que se hace un **push** del registro **ebp** y se copia en él el contenido de la pila (registro **esp**). Posteriormente se moverá a **ebp-4** el número 3E8h y a **eax** el contenido de **ebp-3EC**h. Es aquí cuando el programa nos pedirá introducir la contraseña de esta fase:

```
00405533 E8 28 47 00 00      call        std::basic_istream<char,std::char_traits<char> >::getline (0409C60h)
```

Una vez hecho esto, el programa apilará la contraseña a través de la siguiente instrucción:

```
00405538 68 14 5A 4C 00      push        4C5A14h
```

Será entonces cuando se llamará a la función **strcmp()** y se comparará el registro **ecx** (en el cual se almacenó la contraseña que introdujimos) junto con el contenido que hay en la dirección de memoria **0x004C5A14**, apilada anteriormente:

```
0x004C5A14  2e 52 72 73 72 68 46 4b 6c 00 00 00 38 38 32 61  .RrsrhFKl...882a
```

He aquí donde encontramos la **primera de las contraseñas**, que se corresponde con: **.RrsrhFKl**. Se coge únicamente hasta antes del primer byte a 0, lo que se corresponde con los bytes en hexadecimal: **2E 52 72 73 72 68 46 4B 6C**:

```
Microsoft Windows [Versión 10.0.19042.928]
(c) Microsoft Corporation. Todos los derechos reservados.

C:\Users\nicol\OneDrive\Escritorio\Universidad\FCR\Trabajo en grupo\Segunda fase>main.exe
.RrsrhFKl
Stage 1 disabled
```

¿Cómo hacer para que cualquier contraseña sea válida?

La respuesta es sencilla, hemos de volver a fijarnos en el código inicial y hacer un mínimo cambio. Si bien nos fijamos en dicho código, justo después de hacer la comparación del registro **ecx** y el contenido de la memoria en la que está guardada la contraseña, en el caso de que sean o no sean iguales se hace un **je** (jump if equals). Pues bien, la manera más cómoda para responder a esta pregunta es modificar mediante el uso de **HxD** dicho salto para que en lugar de hacer un **je** haga simplemente un **jmp** a la instrucción posterior al **Explode()**. El código de dicha instrucción se corresponde con **EB 0F**, según la arquitectura x86 de Intel. Entonces hacemos el cambio e introduciendo cualquier combinación de 32 bits habríamos desactivado esta etapa:

```
00004940  FC FF FF 51 E8 77 DF 07 00 83 C4 08 85 C0 74 0C  üÿÿQèwß..ƒÄ..ÀÀt.
00004940  FC FF FF 51 E8 77 DF 07 00 83 C4 08 85 C0 EB 0F  üÿÿQèwß..ƒÄ..ÀÀë.
```

```
C:\Windows\System32\cmd.exe - main.exe
Microsoft Windows [Versión 10.0.19042.928]
(c) Microsoft Corporation. Todos los derechos reservados.

C:\Users\blanc\OneDrive - Universidad de Oviedo\Documentos\FCR\Trabajo grupo\Trabajo 2\secondPhase(1)>main.exe
PATATA
Stage 1 disabled
```

-Stage 2:

Pasamos ahora a la segunda fase de la desactivación de la bomba, la fase 2. De nuevo, nos volvemos a introducir en el código de la función pulsando F11:

```
00405570 55                 push        ebp
00405571 8B EC              mov         ebp,esp
00405573 83 EC 18           sub         esp,18h
00405576 53                 push        ebx
00405577 C7 45 F4 03 00 00 00 mov         dword ptr [ebp-0Ch],3
0040557E C7 45 FC 00 00 00 00 mov         dword ptr [ebp-4],0
00405585 EB 09              jmp         Stage2+20h (0405590h)
00405587 8B 45 FC           mov         eax,dword ptr [ebp-4]
0040558A 83 C0 01           add         eax,1
0040558D 89 45 FC           mov         dword ptr [ebp-4],eax
00405590 83 7D FC 03        cmp         dword ptr [ebp-4],3
00405594 7D 14              jge         Stage2+3Ah (04055AAh)
00405596 8B 4D FC           mov         ecx,dword ptr [ebp-4]
00405599 8D 54 8D E8        lea         edx,[ebp+ecx*4-18h]
0040559D 52                 push        edx
0040559E B9 B0 6B 4F 00     mov         ecx,offset std::cin (04F6BB0h)
004055A3 E8 C8 0F 00 00     call        std::basic_istream<char,std::char_traits<char> >::operator>> (0406570h)
004055A8 EB DD              jmp         Stage2+17h (0405587h)
004055AA C7 45 F8 01 00 00 00 mov         dword ptr [ebp-8],1
004055B1 8D 5D E8           lea         ebx,[ebp-18h]
004055B4 8B 03              mov         eax,dword ptr [ebx]
004055B6 03 43 04           add         eax,dword ptr [ebx+4]
004055B9 83 F8 FC           cmp         eax,0FFFFFFFCh
004055BC 75 07              jne         Stage2+55h (04055C5h)
004055BE C7 45 F8 00 00 00 00 mov         dword ptr [ebp-8],0
004055C5 83 7D F8 00        cmp         dword ptr [ebp-8],0
004055C9 74 0C              je          Stage2+67h (04055D7h)
004055CB 6A 02              push        2
004055CD E8 7E FE FF FF     call        Explode (0405450h)
004055D2 83 C4 04           add         esp,4
004055D5 EB 0F              jmp         Stage2+76h (04055E6h)
004055D7 68 48 5A 4C 00     push        4C5A48h
004055DC 6A 02              push        2
004055DE E8 DD FE FF FF     call        Defuse (04054C0h)
```

```
004055E3 83 C4 08           add         esp,8
004055E6 5B                 pop         ebx
004055E7 8B E5              mov         esp,ebp
004055E9 5D                 pop         ebp
004055EA C3                 ret
```

Vamos a destacar las **operaciones más importantes:**
Justo después de mover a **ebp** el contenido de la pila (**esp**) se mueve a **ebx** el número exacto de números o contraseñas que pedirá el programa, que más adelante se comparará con el número de iteraciones:

```
00405577 C7 45 F4 03 00 00 00 mov         dword ptr [ebp-0Ch],3
```

Una vez hecho esto, en el **jmp** el programa salta a la instrucción alojada en **0405590**h, donde se comparará el registro **ebx** y el número de iteraciones realizadas. En el caso de que no sean iguales, el programa continuará hasta pedir un número:

```
004055A3 E8 C8 0F 00 00     call        std::basic_istream<char,std::char_traits<char> >::operator>> (0406570h)
```

Entonces, saltará de nuevo hacia atrás a la etiqueta **04055C5**h. Este bucle se repetirá un total de 3 veces. A partir de aquí, el programa saltará a la etiqueta **04055AA**h, donde comparará lo que haya en el registro **eax** y el número en complemento a 2 **0FFFFFFFC**h. En el caso de que no sean iguales, la bomba explotará. En caso contrario se desactivará.
Pues bien, para la deducción de esta contraseña numérica, nos hemos ido fijando en lo que sucedía y donde se almacenaban los números que introducíamos. LLegamos a la conclusión de que la suma entre el primer y el segundo número tenía que ser igual a -4, por lo que en realidad hay un gran número de combinaciones posibles, siendo el tercer número irrelevante. En nuestro caso, en primer lugar introdujimos el **1**, después el **-5** y por último el **0**. De esta forma, la etapa es desactivada:

```
Microsoft Windows [Versión 10.0.19042.928]
(c) Microsoft Corporation. Todos los derechos reservados.

C:\Users\nicol\OneDrive\Escritorio\Universidad\FCR\Trabajo en grupo\Segunda fase>main.exe
.RrsrhFKl
Stage 1 disabled
1
-5
0
Stage 2 disabled
```

```
Microsoft Windows [Versión 10.0.19042.928]
(c) Microsoft Corporation. Todos los derechos reservados.

C:\Users\nicol\OneDrive\Escritorio\Universidad\FCR\Trabajo en grupo\Segunda fase>main.exe
.RrsrhFKl
Stage 1 disabled
-2
-2
125
Stage 2 disabled
```

¿Cómo hacer para que cualquier combinación numérica sea válida?

La respuesta es sencilla y muy parecida por no decir idéntica a la de la primera fase.
Hemos de revisar de nuevo el código y hacer una mínima modificación. Si nos fijamos, justo después de hacer la comparación entre el contenido del registro **eax** y el número **-4** en complemento a dos, hay un **jne** (jump not equals). He aquí donde tenemos que modificar el código fuente. Para ello, sustituimos ese **jne** por un **jmp** que salte hasta la función **Defuse**().
Para ello tomamos los datos de codificación de la arquitectura x86 de Intel y codificamos el jump para que salte un total de 25 bytes (**EB 19**h):

```
000049B0  00 8D 5D E8 8B 03 03 43 04 83 F8 FC 75 07 C7 45  ..]è‹..C.ƒøüu.ÇE
000049B0  00 8D 5D E8 8B 03 03 43 04 83 F8 FC EB 19 C7 45  ..]è‹..C.ƒøüë.ÇE
```

```
Microsoft Windows [Versión 10.0.19042.928]
(c) Microsoft Corporation. Todos los derechos reservados.

C:\Users\blanc\OneDrive - Universidad de Oviedo\Documentos\FCR\Trabajo grupo\Trabajo 2\secondPhase(1)>main.exe
hola
Stage 1 disabled
1
2
3
Stage 2 disabled
```

-Stage 3:

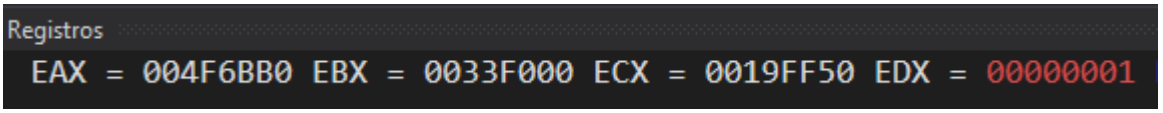
Y por último nos centramos en la última fase de la desactivación de la bomba. Para esta última etapa pulsamos de nuevo y por última vez F11:

```
004055F0 55                push        ebp     ≤ 1 ms transcurridos
004055F1 8B EC             mov         ebp,esp
004055F3 83 EC 14          sub         esp,14h
004055F6 8D 45 F8          lea         eax,[ebp-8]
004055F9 50                push        eax
004055FA 8D 4D FC          lea         ecx,[ebp-4]
004055FD 51                push        ecx
004055FE B9 B0 6B 4F 00    mov         ecx,offset std::cin (04F6BB0h)
00405603 E8 68 0F 00 00    call        std::basic_istream<char,std::char_traits<char> >::operator>> (0406570h)
00405608 8B C8             mov         ecx,eax
0040560A E8 61 0F 00 00    call        std::basic_istream<char,std::char_traits<char> >::operator>> (0406570h)
0040560F 8B 55 F8          mov         edx,dword ptr [ebp-8]
00405612 81 E2 00 00 00 08 and         edx,8000000h
00405618 C1 FA 1B          sar         edx,1Bh
0040561B 89 55 F4          mov         dword ptr [ebp-0Ch],edx
0040561E 8B 45 FC          mov         eax,dword ptr [ebp-4]
00405621 25 00 00 00 01    and         eax,1000000h
00405626 C1 F8 18          sar         eax,18h
00405629 89 45 F0          mov         dword ptr [ebp-10h],eax
0040562C 8B 4D FC          mov         ecx,dword ptr [ebp-4]
0040562F 83 E1 02          and         ecx,2
00405632 D1 F9             sar         ecx,1
00405634 89 4D EC          mov         dword ptr [ebp-14h],ecx
00405637 8B 55 F4          mov         edx,dword ptr [ebp-0Ch]
0040563A 3B 55 F0          cmp         edx,dword ptr [ebp-10h]
0040563D 74 06             je          Stage3+55h (0405645h)
0040563F 83 7D EC 01       cmp         dword ptr [ebp-14h],1
00405643 75 0C             jne         Stage3+61h (0405651h)
```

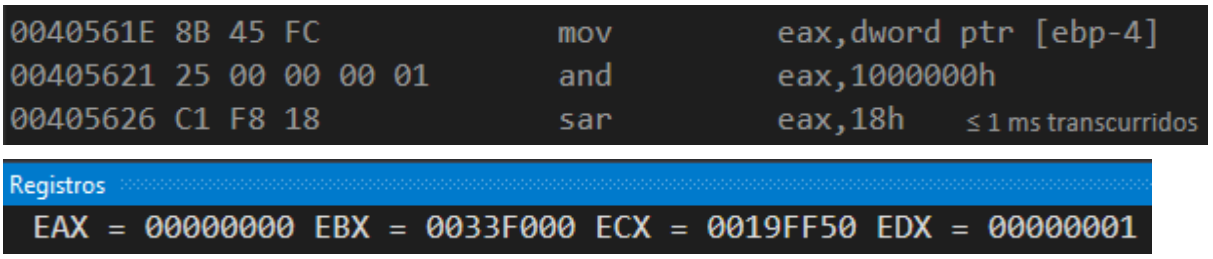
```
00405645 6A 03             push        3
00405647 E8 04 FE FF FF    call        Explode (0405450h)
0040564C 83 C4 04          add         esp,4
0040564F EB 0F             jmp         Stage3+70h (0405660h)
00405651 68 70 5A 4C 00    push        4C5A70h
00405656 6A 03             push        3
00405658 E8 63 FE FF FF    call        Defuse (04054C0h)
0040565D 83 C4 08          add         esp,8
00405660 8B E5             mov         esp,ebp
00405662 5D                pop         ebp
00405663 C3                ret
```

Vamos a comentar los **aspectos más destacables**:
Una vez se ha llamado a la función que permite introducir una clave por teclado, se mueve al registro **ecx**, el contenido del registro **eax**. Posterior a esto, se vuelve a pedir otro número por teclado, el cual se almacenará en **edx**. A continuación se realiza una operación **and** entre el registro **edx** y el número **80000000**h, y en el siguiente paso se realiza una operación **sar**, que mueve el resultado de la operación **and** un total de 27 bits hacia la derecha, dejando un 00000001h en dicho registro, siempre que hayamos introducido como segundo número uno negativo:

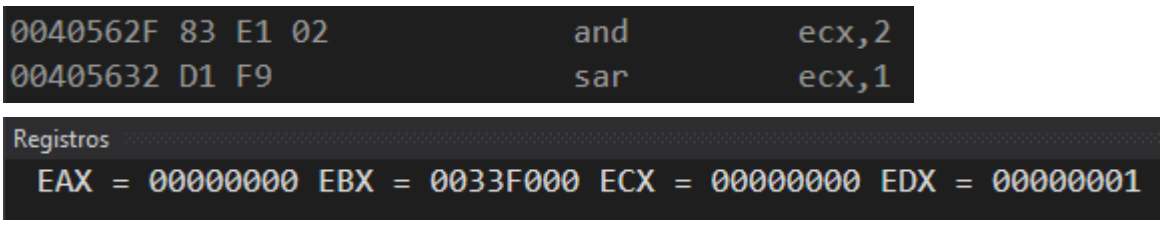
```
00405603 E8 68 0F 00 00    call        std::basic_istream<char,std::char_traits<char> >::operator>> (0406570h)
00405608 8B C8             mov         ecx,eax
0040560A E8 61 0F 00 00    call        std::basic_istream<char,std::char_traits<char> >::operator>> (0406570h)
0040560F 8B 55 F8          mov         edx,dword ptr [ebp-8]
00405612 81 E2 00 00 00 08 and         edx,8000000h
00405618 C1 FA 1B          sar         edx,1Bh
```



A partir de aquí se vuelve a hacer la operación **and** pero con el registro **eax** (el primer número que introdujimos) y el número **10000000**h, para después, mediante la operación **sar** se mueva el bit resultante 24 posiciones hacia la derecha (**18**h) . De esta forma, conseguiremos, introduciendo como primer número un 0 o una potencia de 2, que el registro **eax** quede a 0:

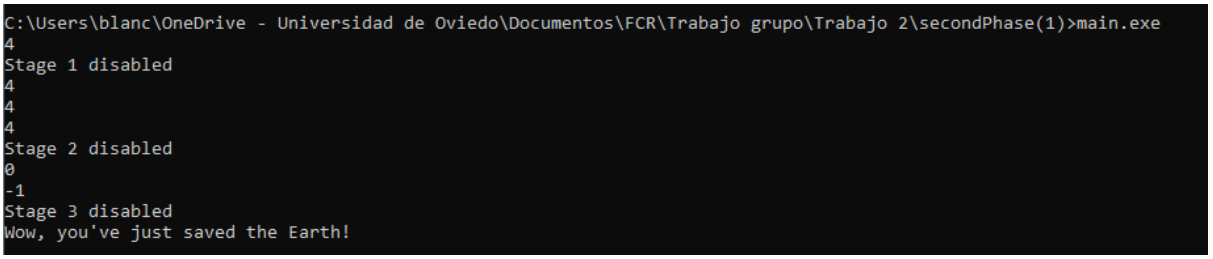


LLegados a este punto, se volverá a hacer una operación **and** entre el registro **ecx** y el número 2, y posteriormente se realizará la operación **sar**, que moverá el byte resultante 1 posición, lo que nos deja un 1 a la derecha:



En este punto, se comparará lo que haya en el registro **ecx** y **edx**. En el caso de que sean iguales, el código nos llevaría a la explosión de la bomba. Si los registros contienen números distintos, el programa comparará el registro **[ebp - 14h]** y el número **1**h. Si son iguales la bomba explotará, si son distintos, se llamará a la función **Defuse**() y habremos desactivado la bomba con éxito.

Al igual que en la segunda etapa, en esta fase también hay un gran número de combinaciones posibles, pero todas tienen el mismo patrón:
El primer número introducido ha de ser o bien 0 o una potencia de 2, y el segundo número ha de ser siempre negativo. En nuestro caso, nosotros hemos probado dos **combinaciones funcionales**: **0** y **-1**, **16** y **-4**:



```
C:\Users\blanc\OneDrive - Universidad de Oviedo\Documentos\FCR\Trabajo grupo\Trabajo 2\secondPhase(1)>main.exe
1
Stage 1 disabled
1
1
1
Stage 2 disabled
16
-4
Stage 3 disabled
Wow, you've just saved the Earth!
```

¿Cómo hacer para que cualquier combinación numérica sea válida?

La respuesta es idéntica a la anterior. Tenemos que evitar todo el código que haya por debajo de la instrucción **cmp edx, dword ptr [ebp - 10h]**. Para ello, tenemos que modificar el código fuente, cambiando el **je** (jump equals) siguiente por un simple **jmp**, que salte hasta justo antes de la operación **Defuse()**, es decir, un total de 25 bytes (**19**h). Para ello, abrimos de nuevo el **HxD** y buscamos la instrucción **74 06** h, que se corresponde con el **je**, y la sustituímos por **EB 19**h (codificación del **jmp +19**h):

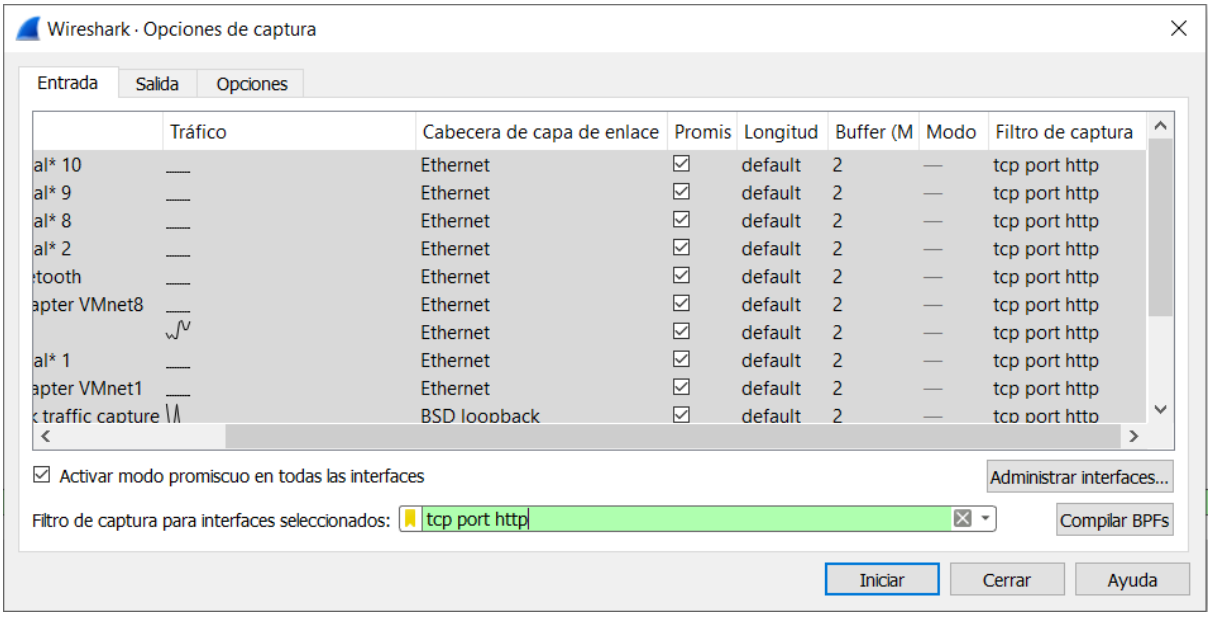
```
00004A30  E1 02 D1 F9 89 4D EC 8B 55 F4 3B 55 F0 74 06 83  á.Ñù‰Mì‹Uô;Uðt.ƒ
00004A30  E1 02 D1 F9 89 4D EC 8B 55 F4 3B 55 F0 EB 19 83  á.Ñù‰Mì‹Uô;Uðë.ƒ
```

```
Microsoft Windows [Versión 10.0.19042.928]
(c) Microsoft Corporation. Todos los derechos reservados.

C:\Users\blanc\OneDrive - Universidad de Oviedo\Documentos\FCR\Trabajo grupo\Trabajo 2\secondPhase(1)>"main

1
Stage 1 disabled
1
1
1
Stage 2 disabled
9999
9999
Stage 3 disabled
Wow, you've just saved the Earth!

C:\Users\blanc\OneDrive - Universidad de Oviedo\Documentos\FCR\Trabajo grupo\Trabajo 2\secondPhase(1)>
```

¡Y ya hemos salvado el mundo!. Ahora solo falta averiguar el nombre de quienes han intentado destruirlo…

-Nombre de los hackers:

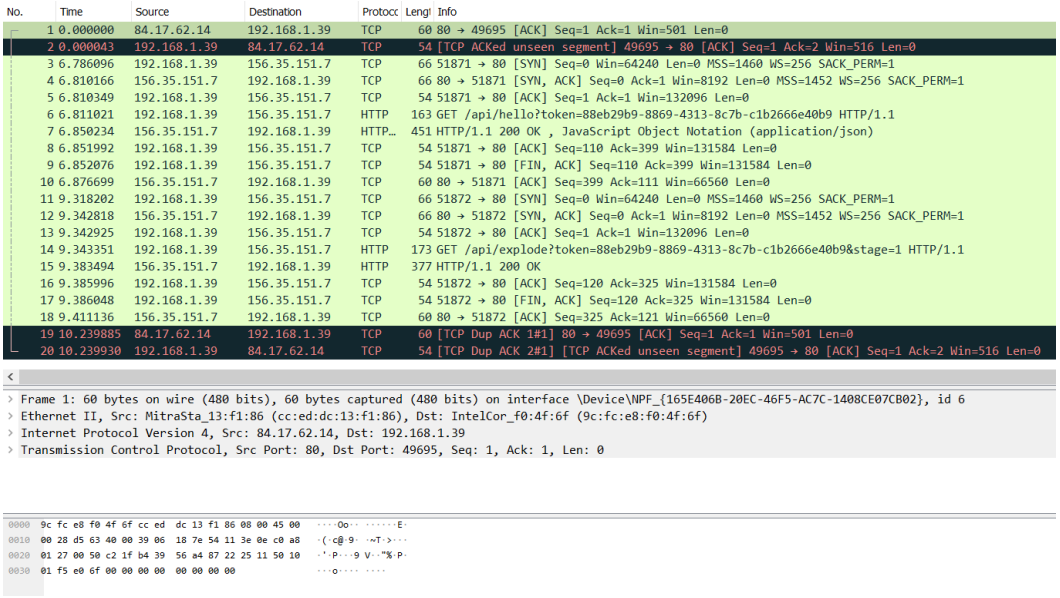
En primer lugar abrimos el Wireshark y seleccionamos el filtro más adecuado para averiguar con quién o qué se conecta nuestro ordenador al ejecutar el .exe . Aplicaremos el filtro **tcp port http** (visto en clase) ya que es el que nos permite observar las comunicaciones con un servidor web:



Una vez aplicamos el filtro, abrimos el ejecutable y probamos una contraseña cualquiera:

```
Microsoft Windows [Versión 10.0.19042.928]
(c) Microsoft Corporation. Todos los derechos reservados.

C:\Users\blanc\OneDrive - Universidad de Oviedo\Documentos\FCR\Trabajo grupo\Trabajo 2>original.exe
patata
Oh, no, the world is over! BOOM!

C:\Users\blanc\OneDrive - Universidad de Oviedo\Documentos\FCR\Trabajo grupo\Trabajo 2>
```

Es ahora cuando Wireshark ha capturado todas las comunicaciones realizadas:



No.	Time	Source	Destination	Protocc	Lengt	Info
1	0.000000	84.17.62.14	192.168.1.39	TCP	60	80 → 49695 [ACK] Seq=1 Ack=1 Win=501 Len=0
2	0.000043	192.168.1.39	84.17.62.14	TCP	54	[TCP ACKed unseen segment] 49695 → 80 [ACK] Seq=1 Ack=2 Win=516 Len=0
3	6.786096	192.168.1.39	156.35.151.7	TCP	66	51871 → 80 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM=1
4	6.810166	156.35.151.7	192.168.1.39	TCP	66	80 → 51871 [SYN, ACK] Seq=0 Ack=1 Win=8192 Len=0 MSS=1452 WS=256 SACK_PERM=1
5	6.810349	192.168.1.39	156.35.151.7	TCP	54	51871 → 80 [ACK] Seq=1 Ack=1 Win=132096 Len=0
6	6.811021	192.168.1.39	156.35.151.7	HTTP	163	GET /api/hello?token=88eb29b9-8869-4313-8c7b-c1b2666e40b9 HTTP/1.1
7	6.850234	156.35.151.7	192.168.1.39	HTTP…	451	HTTP/1.1 200 OK , JavaScript Object Notation (application/json)
8	6.851992	192.168.1.39	156.35.151.7	TCP	54	51871 → 80 [ACK] Seq=110 Ack=399 Win=131584 Len=0
9	6.852076	192.168.1.39	156.35.151.7	TCP	54	51871 → 80 [FIN, ACK] Seq=110 Ack=399 Win=131584 Len=0
10	6.876699	156.35.151.7	192.168.1.39	TCP	60	80 → 51871 [ACK] Seq=399 Ack=111 Win=66560 Len=0
11	9.318202	192.168.1.39	156.35.151.7	TCP	66	51872 → 80 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM=1
12	9.342818	156.35.151.7	192.168.1.39	TCP	66	80 → 51872 [SYN, ACK] Seq=0 Ack=1 Win=8192 Len=0 MSS=1452 WS=256 SACK_PERM=1
13	9.342925	192.168.1.39	156.35.151.7	TCP	54	51872 → 80 [ACK] Seq=1 Ack=1 Win=132096 Len=0
14	9.343351	192.168.1.39	156.35.151.7	HTTP	173	GET /api/explode?token=88eb29b9-8869-4313-8c7b-c1b2666e40b9&stage=1 HTTP/1.1
15	9.383494	156.35.151.7	192.168.1.39	HTTP	377	HTTP/1.1 200 OK
16	9.385996	192.168.1.39	156.35.151.7	TCP	54	51872 → 80 [ACK] Seq=120 Ack=325 Win=131584 Len=0
17	9.386048	192.168.1.39	156.35.151.7	TCP	54	51872 → 80 [FIN, ACK] Seq=120 Ack=325 Win=131584 Len=0
18	9.411136	156.35.151.7	192.168.1.39	TCP	60	80 → 51872 [ACK] Seq=325 Ack=121 Win=66560 Len=0
19	10.239885	84.17.62.14	192.168.1.39	TCP	60	[TCP Dup ACK 1#1] 80 → 49695 [ACK] Seq=1 Ack=1 Win=501 Len=0
20	10.239930	192.168.1.39	84.17.62.14	TCP	54	[TCP Dup ACK 2#1] [TCP ACKed unseen segment] 49695 → 80 [ACK] Seq=1 Ack=2 Win=516 Len=0

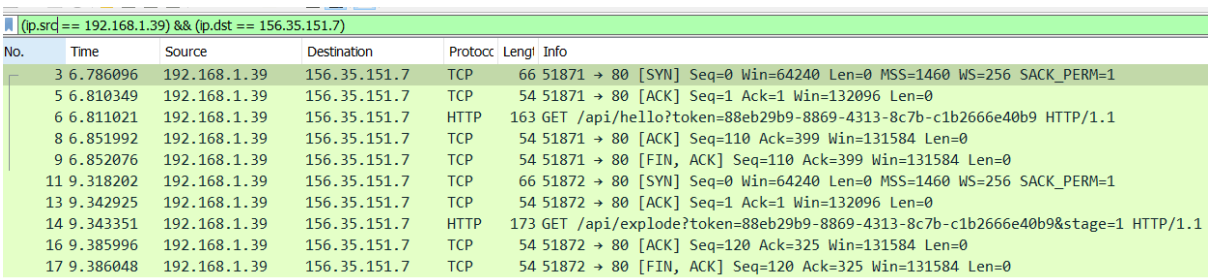
```
> Frame 1: 60 bytes on wire (480 bits), 60 bytes captured (480 bits) on interface \Device\NPF_{165E406B-20EC-46F5-AC7C-1408CE07CB02}, id 6
> Ethernet II, Src: MitraSta_13:f1:86 (cc:ed:dc:13:f1:86), Dst: IntelCor_f0:4f:6f (9c:fc:e8:f0:4f:6f)
> Internet Protocol Version 4, Src: 84.17.62.14, Dst: 192.168.1.39
> Transmission Control Protocol, Src Port: 80, Dst Port: 49695, Seq: 1, Ack: 1, Len: 0
```

```
0000  9c fc e8 f0 4f 6f cc ed  dc 13 f1 86 08 00 45 00   ····Oo·· ·····E·
0010  00 28 d5 63 40 00 39 06  18 7e 54 11 3e 0e c0 a8   ·(·c@·9· ·~T·>···
0020  01 27 00 50 c2 1f b4 39  56 a4 87 22 25 11 50 10   ·'·P···9 V··"%·P·
0030  01 f5 e0 6f 00 00 00 00  00 00 00 00              ···o···· ····
```

Llegados a este punto, ¿quién es nuestro ordenador y quién es el servidor de los hackers?. Esta pregunta tiene fácil solución. Para resolverla, hemos de abrir nuestro cmd e introducir el comando "**ipconfig**":

```
Adaptador de LAN inalámbrica Wi-Fi:

   Sufijo DNS específico para la conexión. . :
   Vínculo: dirección IPv6 local. . . : fe80::555f:81c2:654c:d3dc%4
   Dirección IPv4. . . . . . . . . . . . : 192.168.1.39
   Máscara de subred . . . . . . . . . . : 255.255.255.0
   Puerta de enlace predeterminada . . . . . : 192.168.1.1
```

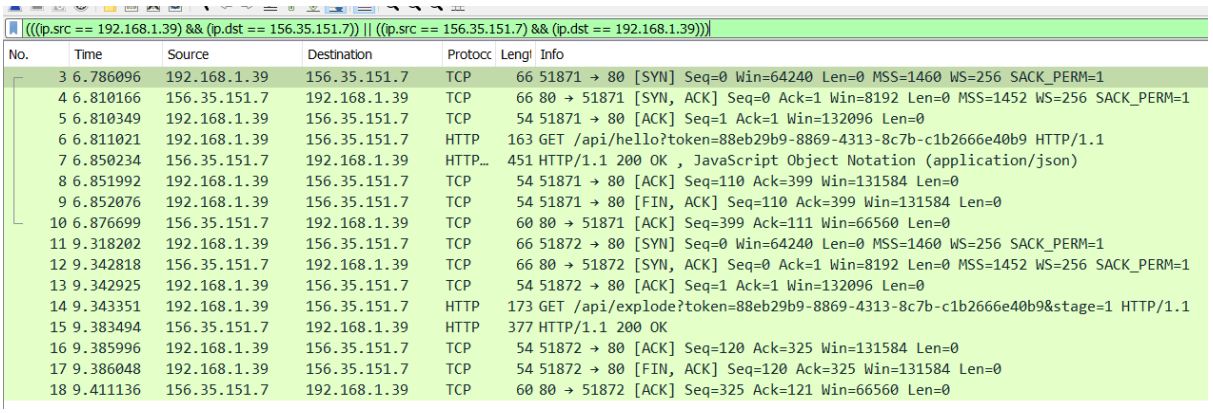
De aquí averiguamos que la dirección ip de nuestro dispositivo es **192.168.1.39**, por lo que ya tenemos parte de uno de los filtros a aplicar. Como bien pudimos observar y anotar en nuestras clases de laboratorio, averiguamos que la dirección ip del servidor de Oviedo al que se comunica la bomba es **156.35.151.7** (también es deducible puesto que la información que se muestra en uno de los mensajes es un hello?). Hemos completado el primer filtro completo:



(ip.src == 192.168.1.39) && (ip.dst == 156.35.151.7)

No.	Time	Source	Destination	Protocc	Lengt	Info
3	6.786096	192.168.1.39	156.35.151.7	TCP	66	51871 → 80 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM=1
5	6.810349	192.168.1.39	156.35.151.7	TCP	54	51871 → 80 [ACK] Seq=1 Ack=1 Win=132096 Len=0
6	6.811021	192.168.1.39	156.35.151.7	HTTP	163	GET /api/hello?token=88eb29b9-8869-4313-8c7b-c1b2666e40b9 HTTP/1.1
8	6.851992	192.168.1.39	156.35.151.7	TCP	54	51871 → 80 [ACK] Seq=110 Ack=399 Win=131584 Len=0
9	6.852076	192.168.1.39	156.35.151.7	TCP	54	51871 → 80 [FIN, ACK] Seq=110 Ack=399 Win=131584 Len=0
11	9.318202	192.168.1.39	156.35.151.7	TCP	66	51872 → 80 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM=1
13	9.342925	192.168.1.39	156.35.151.7	TCP	54	51872 → 80 [ACK] Seq=1 Ack=1 Win=132096 Len=0
14	9.343351	192.168.1.39	156.35.151.7	HTTP	173	GET /api/explode?token=88eb29b9-8869-4313-8c7b-c1b2666e40b9&stage=1 HTTP/1.1
16	9.385996	192.168.1.39	156.35.151.7	TCP	54	51872 → 80 [ACK] Seq=120 Ack=325 Win=131584 Len=0
17	9.386048	192.168.1.39	156.35.151.7	TCP	54	51872 → 80 [FIN, ACK] Seq=120 Ack=325 Win=131584 Len=0

¿Pero qué sucede con la comunicación Servidor-nuestro Pc?. Como no lo sabemos, aplicaremos también el filtro inverso:



(((ip.src == 192.168.1.39) && (ip.dst == 156.35.151.7)) || ((ip.src == 156.35.151.7) && (ip.dst == 192.168.1.39)))

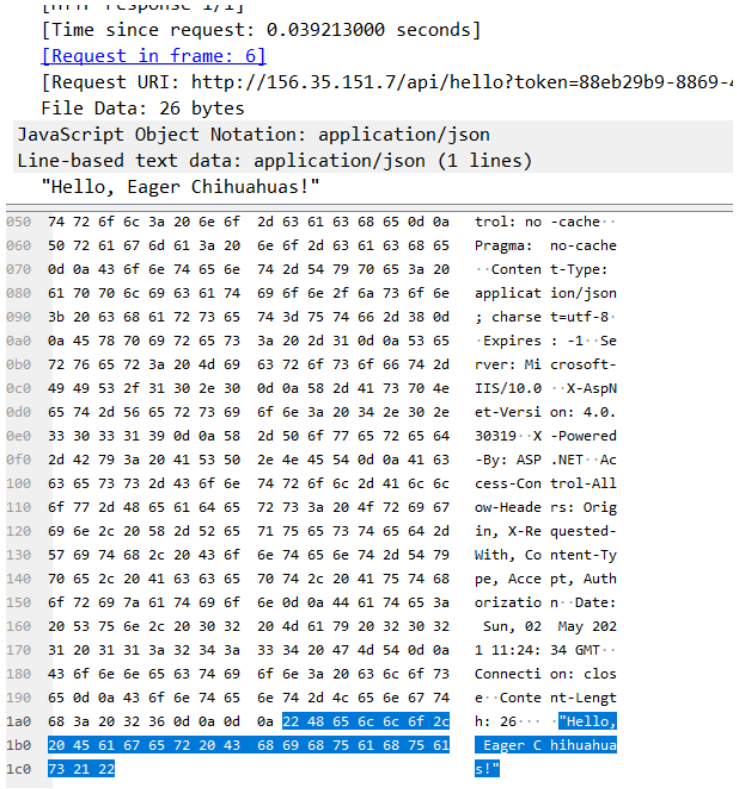
No.	Time	Source	Destination	Protocc	Lengt	Info
3	6.786096	192.168.1.39	156.35.151.7	TCP	66	51871 → 80 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM=1
4	6.810166	156.35.151.7	192.168.1.39	TCP	66	80 → 51871 [SYN, ACK] Seq=0 Ack=1 Win=8192 Len=0 MSS=1452 WS=256 SACK_PERM=1
5	6.810349	192.168.1.39	156.35.151.7	TCP	54	51871 → 80 [ACK] Seq=1 Ack=1 Win=132096 Len=0
6	6.811021	192.168.1.39	156.35.151.7	HTTP	163	GET /api/hello?token=88eb29b9-8869-4313-8c7b-c1b2666e40b9 HTTP/1.1
7	6.850234	156.35.151.7	192.168.1.39	HTTP…	451	HTTP/1.1 200 OK , JavaScript Object Notation (application/json)
8	6.851992	192.168.1.39	156.35.151.7	TCP	54	51871 → 80 [ACK] Seq=110 Ack=399 Win=131584 Len=0
9	6.852076	192.168.1.39	156.35.151.7	TCP	54	51871 → 80 [FIN, ACK] Seq=110 Ack=399 Win=131584 Len=0
10	6.876699	156.35.151.7	192.168.1.39	TCP	60	80 → 51871 [ACK] Seq=399 Ack=111 Win=66560 Len=0
11	9.318202	192.168.1.39	156.35.151.7	TCP	66	51872 → 80 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM=1
12	9.342818	156.35.151.7	192.168.1.39	TCP	66	80 → 51872 [SYN, ACK] Seq=0 Ack=1 Win=8192 Len=0 MSS=1452 WS=256 SACK_PERM=1
13	9.342925	192.168.1.39	156.35.151.7	TCP	54	51872 → 80 [ACK] Seq=1 Ack=1 Win=132096 Len=0
14	9.343351	192.168.1.39	156.35.151.7	HTTP	173	GET /api/explode?token=88eb29b9-8869-4313-8c7b-c1b2666e40b9&stage=1 HTTP/1.1
15	9.383494	156.35.151.7	192.168.1.39	HTTP	377	HTTP/1.1 200 OK
16	9.385996	192.168.1.39	156.35.151.7	TCP	54	51872 → 80 [ACK] Seq=120 Ack=325 Win=131584 Len=0
17	9.386048	192.168.1.39	156.35.151.7	TCP	54	51872 → 80 [FIN, ACK] Seq=120 Ack=325 Win=131584 Len=0
18	9.411136	156.35.151.7	192.168.1.39	TCP	60	80 → 51872 [ACK] Seq=325 Ack=121 Win=66560 Len=0

Es ahora cuando ya tenemos todas las comunicaciones entre Pc-Oviedo/Oviedo-Pc. Si vamos observando la información de cada una de las llamadas, hay una cuya info nos muestra un **hello?**,enviado por nosotros, como si quisiéramos entablar una conversación. Es entonces cuando marcamos la opción de seguir flujo tcp, y llegamos a esta conclusión:

```
GET /api/hello?token=88eb29b9-8869-4313-8c7b-c1b2666e40b9 HTTP/1.1
Host: 156.35.151.7
Connection: close

HTTP/1.1 200 OK
Cache-Control: no-cache
Pragma: no-cache
Content-Type: application/json; charset=utf-8
Expires: -1
Server: Microsoft-IIS/10.0
X-AspNet-Version: 4.0.30319
X-Powered-By: ASP.NET
Access-Control-Allow-Headers: Origin, X-Requested-With, Content-Type, Accept, Authorization
Date: Sun, 02 May 2021 11:24:34 GMT
Connection: close
Content-Length: 26

"Hello, Eager Chihuahuas!"
```

El servidor ha respondido a los hackers por su nombre, **Eager Chihuahuas**.

También se puede comprobar observando la respuesta que nos da el servidor una vez hemos hecho la llamada comprobando el apartado **Hypertext Transfer Protocol**:



-Reparto del trabajo:

Nicolás Montiel Melendi: **18h**
Hugo Besteiro Gutiérrez: **7h**
Eduardo Blanco Bielsa: **30h**