

TRABAJO FUNDAMENTOS DE COMPUTADORES Y REDES: Segunda fase.



Ángela López López UO270318

Juan Mencía Menendez UO264197

Adrián Álvarez Rodríguez

UO265336

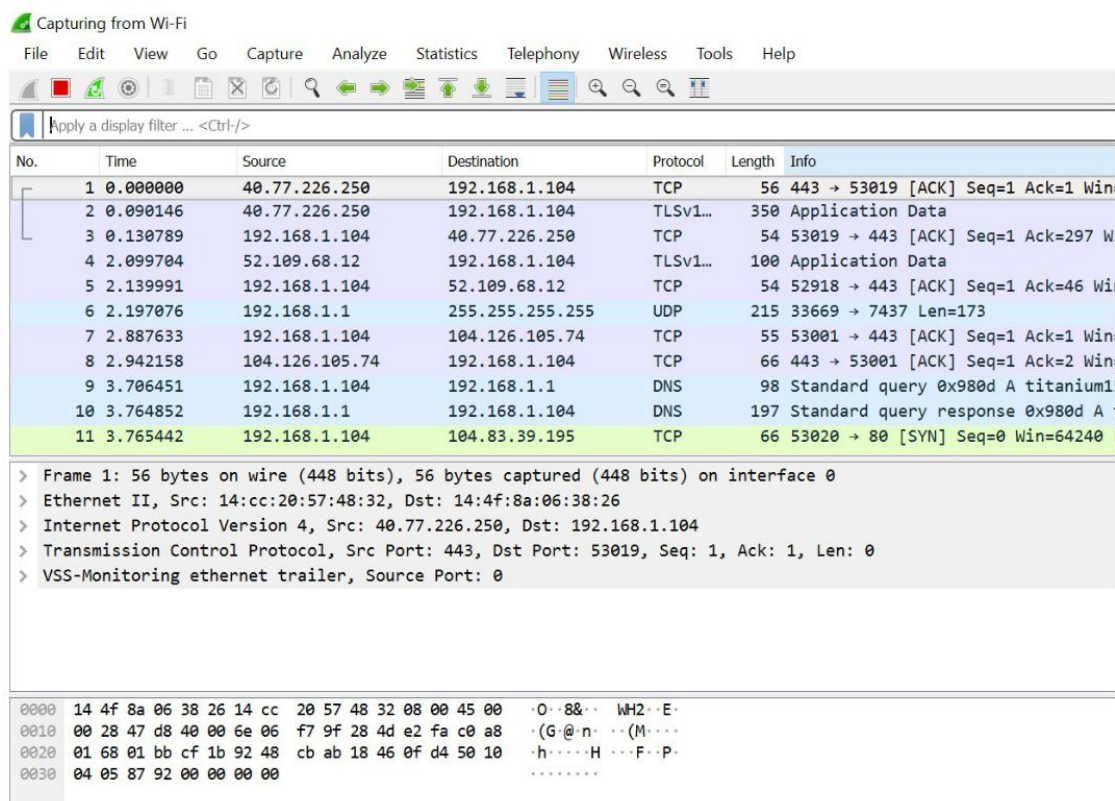
TIEMPO DEDICADO AL TRABAJO:

El trabajo se ha realizado en conjunto de manera presencial, por lo que no hay fases definidas para cada integrante. Sumando todos los encuentros podemos considerar las horas totales de trabajo en unas 6h.

NOMBRE DEL GRUPO CRIMINAL:

Cuando se ejecuta la aplicación esta se conecta con el servidor. La respuesta de este trae un mensaje que muestra el nombre del grupo criminal.

Para poder obtener esos datos es necesario utilizar la captura de datos en Wireshark mientras se ejecuta la aplicación. De esta forma podemos ver los paquetes que se intercambian.



Como lo que nos interesa es la respuesta del servidor, ponemos http en la barra de búsqueda para filtrar los resultados. Ya que queremos la respuesta del servidor, buscamos aquellos mensajes donde responda con OK. Vemos que en uno de ellos devuelve un archivo json a nuestra dirección ip desde una dirección perteneciente a uniovi (lo cual podemos saber a través del comando nslookup).

```

Adaptador de LAN inalámbrica Wi-Fi:

Sufijo DNS específico para la conexión. . . :
Vínculo: dirección IPv6 local. . . : fe80::7882:150b:4569:b529%18
Dirección IPv4. . . . . : 192.168.1.104
Máscara de subred . . . . . : 255.255.255.0
Puerta de enlace predeterminada . . . . : fe80::5489:dff:fefe:2f69%18
                                           192.168.1.1

C:\Users\angel>nslookup 156.35.151.7
Servidor: UnKnown
Address: 192.168.1.1

Nombre: merak.edv.uniovi.es
Address: 156.35.151.7

```

En el apartado Line-based text data podemos ver el mensaje enviado por el servidor: "Hello, Amazing Hackers!".

| http | | | | | | |
|------|------------|-----------------|-----------------|----------|--------|---|
| No. | Time | Source | Destination | Protocol | Length | Info |
| 98 | 10.795927 | 23.214.200.125 | 192.168.1.104 | HTTP | 649 | HTTP/1.1 200 OK (text/html) |
| 102 | 10.797961 | 23.214.200.125 | 192.168.1.104 | HTTP | 649 | HTTP/1.1 200 OK (text/html) |
| 111 | 11.248768 | 192.168.1.104 | 156.35.151.7 | HTTP | 163 | GET /api/hello?token=idb65942-47e9-49c0-8195-70bd8aec7290 |
| 112 | 11.251017 | 192.168.1.104 | 104.126.105.131 | HTTP | 600 | GET /T/172/fBgma2D__Evp9MYmE5CHTeVGrLv |
| 123 | 11.329281 | 156.35.151.7 | 192.168.1.104 | HTTP | 450 | HTTP/1.1 200 OK (application/json) |
| 130 | 11.387537 | 192.168.1.104 | 23.214.200.125 | HTTP | 383 | GET /DCENSUS/64/dea0bc2ee8815bf1b430d61 |
| 131 | 11.388041 | 192.168.1.104 | 104.126.105.131 | HTTP | 600 | GET /T/172/fBgma2D__Evp9MYmE5CHTeVGrLv |
| 136 | 11.578595 | 104.126.105.131 | 192.168.1.104 | HTTP | 554 | HTTP/1.1 200 OK (text/html) |
| 142 | 13.133465 | 23.214.200.125 | 192.168.1.104 | HTTP | 873 | HTTP/1.1 200 OK (text/html) |
| 1520 | 378.789562 | 192.168.1.104 | 23.214.200.125 | HTTP | 448 | GET /CENSUS/192/628a34bf49944a0519fedb1 |
| 1522 | 378.858492 | 23.214.200.125 | 192.168.1.104 | HTTP | 861 | HTTP/1.1 200 OK (text/html) |

| | |
|---|--|
| \r\n | |
| [HTTP response 1/1] | |
| [Time since request: 0.080513000 seconds] | |
| [Request in frame: 111] | |
| [Request URI: http://156.35.151.7/api/hello?token=idb65942-47e9-49c0-8195-70bd8aec7290] | |
| File Data: 25 bytes | |
| JavaScript Object Notation: application/json | |
| Line-based text data: application/json (1 lines) | |
| "Hello, Amazing Hackers!" | |

| | | |
|------|---|--------------------|
| 0150 | 6f 72 69 7a 61 74 69 6f 6e 0d 0a 44 61 74 65 3a | orizatio n...Date: |
| 0160 | 20 4d 6f 6e 2c 20 30 36 20 4d 61 79 20 32 30 31 | Mon, 06 May 201 |
| 0170 | 39 20 31 30 3a 34 39 3a 32 32 20 47 4d 54 0d 0a | 9 10:49: 22 GMT.. |
| 0180 | 43 6f 6e 6e 65 63 74 69 6f 6e 3a 20 63 6c 6f 73 | Connecti on: clos |
| 0190 | 65 0d 0a 43 6f 6e 74 65 6e 74 2d 4c 65 6e 67 74 | e...Conte nt-Lengt |
| 01a0 | 68 3a 20 32 35 0d 0a 0d 0a 22 48 65 6c 6c 6f 2c | h: 25... "Hello, |
| 01b0 | 20 41 6d 61 7a 69 6e 67 20 48 61 63 6b 65 72 73 | Amazing Hackers |
| 01c0 | 21 22 | !" |

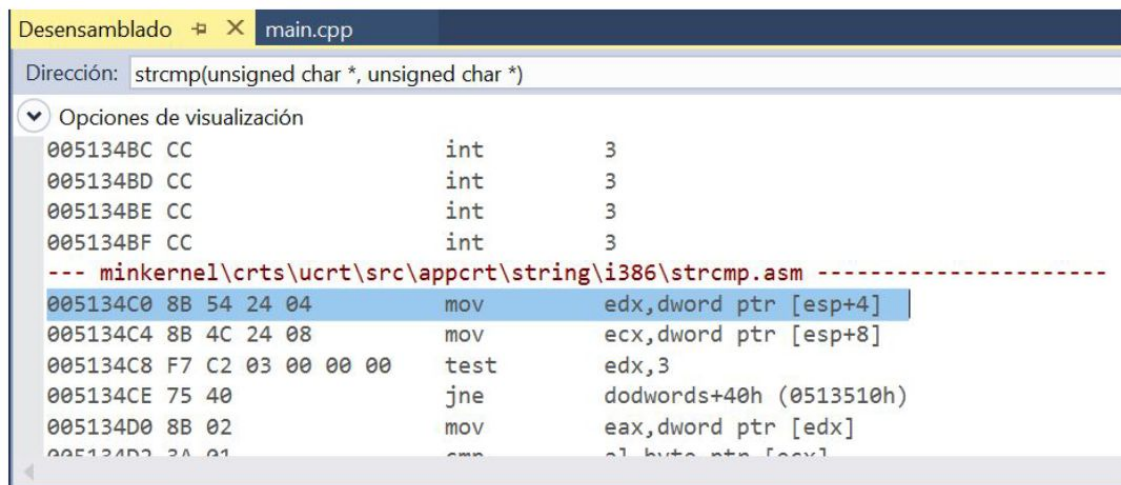
Por lo tanto, el grupo criminal es **Amazing Hackers**.

PRIMERA CONTRASEÑA:

Tenemos que hacer ingeniería inversa para obtener las contraseñas necesarias para desactivar la bomba.

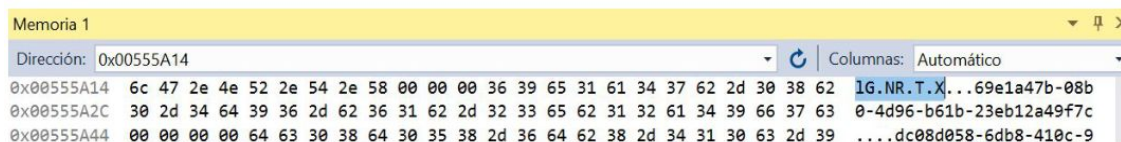
Observando el código vemos que antes del método Explode() hay un salto condicional je Stage1+4Ch (049555Ch) que salta a la dirección de memoria 049555Ch si ZF=1 (el resultado es 0), por lo que nuestra intención es que entre por ese método. Justo antes vemos el método test eax,eax que realiza un AND binario (es decir, eax tiene que ser 0 para que se cumpla la condición del salto).

Antes hay una comparación de strings. Para que el resultado sea 0 ambos strings deben ser iguales. Como call strcmp(05134C0h) apunta a mov edx,dword ptr [esp+4], se compara con el valor de esp+4, es decir, el penúltimo elemento insertado en la pila, en este caso el valor de 555A14h.



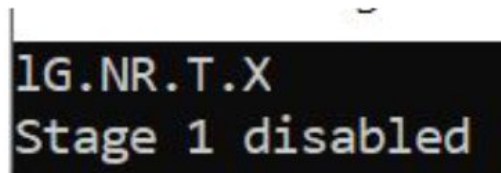
```
Desensamblado  main.cpp
Dirección: strcmp(unsigned char *, unsigned char *)
Opciones de visualización
0051348C CC          int      3
0051348D CC          int      3
0051348E CC          int      3
0051348F CC          int      3
--- minkernel\crt\src\appcrt\string\i386\strcmp.asm ---
005134C0 8B 54 24 04        mov     edx,dword ptr [esp+4]
005134C4 8B 4C 24 08        mov     ecx,dword ptr [esp+8]
005134C8 F7 C2 03 00 00 00  test     edx,3
005134CE 75 40             jne     dodwords+40h (0513510h)
005134D0 8B 02             mov     eax,dword ptr [edx]
005134D2 3A 01             cmp     al,byte ptr [ecx]
```

Por lo tanto, para que el resultado sea 0 debe introducirse el mismo valor como contraseña.



```
Memoria 1
Dirección: 0x00555A14  Columnas: Automático
0x00555A14  6c 47 2e 4e 52 2e 54 2e 58 00 00 00 36 39 65 31 61 34 37 62 2d 30 38 62  1G.NR.T.X...69e1a47b-08b
0x00555A2C  30 2d 34 64 39 36 2d 62 36 31 62 2d 32 33 65 62 31 32 61 34 39 66 37 63  0-4d96-b61b-23eb12a49f7c
0x00555A44  00 00 00 00 64 63 30 38 64 30 35 38 2d 36 64 62 38 2d 34 31 30 63 2d 39  ...dc08d058-6db8-410c-9
```

Es decir, la primera contraseña es **IG.NR.T.X**



```
IG.NR.T.X
Stage 1 disabled
```


SEGUNDA CONTRASEÑA:

En este caso buscamos que no entre por el salto condicional si no son iguales jne Stage2+55h (04955C5h), ya que si pasa por ahí la siguiente condición no se cumple y la bomba explota. Por lo tanto, cmp eax,0FFFFFFF9h tiene que devolver 0 (es decir, **eax tiene que valer 0FFFFFFF9h**).

El valor de eax viene dado por: lea ebx,[ebp-1Ch], mov eax,dword ptr [ebx] - dword ptr [ebx+0Ch].

Para llegar a esos pasos es necesario pasar primero por un bucle que se repite 4 iteraciones (hasta que la comparativa con 4 sea mayor o igual que 0).

Por lo tanto vamos a ir introduciendo las 4 contraseñas y mirando donde se colocan con respecto a ebp:

(Importante recordar que se utiliza el formato Little Endian, donde se ordenan los bytes de menos significativo a más).

| Memoria 1 | |
|------------|---|
| Dirección: | 0x0019FE3E |
| 0x0019FE3E | 58 00 94 fe 19 00 b0 6b 58 00 c4 fe 19 00 a0 fb 54 00 ff ff ff ff 60 fe 19 00 5f 65 49 00 94 fe 19 00 d0 fe 19 00 |
| 0x0019FE64 | fd 66 49 00 f0 f8 4c 00 f0 f8 4c 00 00 00 27 00 58 6b 58 00 01 0a 58 00 58 6b 58 00 00 00 58 00 00 00 01 00 |
| 0x0019FE8A | 49 00 e8 6a 58 00 50 5c 7b 00 b0 6b 58 00 01 6a 58 00 b0 6b 58 00 38 b2 7a 00 58 6b 58 00 8c fe 19 00 8c fe 19 00 |
| 0x0019FEB0 | c8 6b 58 00 01 00 00 00 00 00 00 00 b0 6b 58 00 68 fe 19 00 60 ff 19 00 c0 fb 54 00 ff ff ff ff fc fe 19 00 a8 55 |
| 0x0019FED6 | 49 00 e0 fe 19 00 00 00 27 00 01 00 00 00 4b 21 49 00 f8 fe 19 00 be 32 49 00 04 00 00 00 98 6a 58 00 00 00 00 00 |
| 0x0019FEFC | 04 ff 19 00 60 17 49 00 18 ff 19 00 8e f8 4c 00 01 00 00 00 f8 a6 76 00 d0 b8 7a 00 70 ff 19 00 80 f7 4c 00 a2 c7 |
| 0x0019FF22 | ff e9 f0 f8 4c 00 f0 f8 4c 00 00 00 27 00 |

| Memoria 1 | |
|------------|---|
| Dirección: | 0x0019FE8A |
| 0x0019FE8A | 49 00 e8 6a 58 00 a0 33 78 00 b0 6b 58 00 01 6a 58 00 b0 6b 58 00 70 b3 77 00 58 6b 58 00 8c fe 19 00 8c fe 19 00 |
| 0x0019FEB0 | c8 6b 58 00 01 00 00 00 00 00 00 b0 6b 58 00 68 fe 19 00 60 ff 19 00 c0 fb 54 00 ff ff ff ff fc fe 19 00 a8 55 |
| 0x0019FED6 | 49 00 e4 fe 19 00 00 70 3f 00 01 00 00 00 01 00 00 00 f8 fe 19 00 be 32 49 00 04 00 00 00 98 6a 58 00 01 00 00 00 |
| 0x0019FEFC | 04 ff 19 00 60 17 49 00 18 ff 19 00 8e f8 4c 00 01 00 00 00 20 a1 73 00 28 b9 77 00 70 ff 19 00 80 f7 4c 00 78 72 |
| 0x0019FF22 | a4 7f f0 f8 4c 00 f0 f8 4c 00 00 70 3f 00 |
| 0x0019FF48 | 00 00 00 00 f0 81 58 00 f4 81 58 00 00 00 00 00 20 ff 19 00 00 00 00 00 cc ff 19 00 50 35 4d 00 80 b9 e5 7f 00 00 |

| Memoria 1 | |
|------------|---|
| Dirección: | 0x0019FE8A |
| 0x0019FE8A | 49 00 e8 6a 58 00 a0 33 78 00 b0 6b 58 00 01 6a 58 00 b0 6b 58 00 70 b3 77 00 58 6b 58 00 8c fe 19 00 8c fe 19 00 |
| 0x0019FEB0 | c8 6b 58 00 01 00 00 00 00 00 00 b0 6b 58 00 68 fe 19 00 60 ff 19 00 c0 fb 54 00 ff ff ff ff fc fe 19 00 a8 55 |
| 0x0019FED6 | 49 00 e8 fe 19 00 00 70 3f 00 01 00 00 00 01 00 00 00 01 00 00 00 be 32 49 00 04 00 00 00 98 6a 58 00 02 00 00 00 |
| 0x0019FEFC | 04 ff 19 00 60 17 49 00 18 ff 19 00 8e f8 4c 00 01 00 00 00 20 a1 73 00 28 b9 77 00 70 ff 19 00 80 f7 4c 00 78 72 |
| 0x0019FF22 | a4 7f f0 f8 4c 00 f0 f8 4c 00 00 70 3f 00 |

| Memoria 1 | |
|------------|---|
| Dirección: | 0x0019FE8A |
| 0x0019FE8A | 49 00 e8 6a 58 00 a0 33 78 00 b0 6b 58 00 01 6a 58 00 b0 6b 58 00 70 b3 77 00 58 6b 58 00 8c fe 19 00 8c fe 19 00 |
| 0x0019FEB0 | c8 6b 58 00 08 00 00 00 00 00 00 b0 6b 58 00 68 fe 19 00 60 ff 19 00 c0 fb 54 00 ff ff ff ff fc fe 19 00 a8 55 |
| 0x0019FED6 | 49 00 ec fe 19 00 00 70 3f 00 01 00 00 00 01 00 00 00 01 00 00 00 04 00 00 00 98 6a 58 00 03 00 00 00 00 |
| 0x0019FEFC | 04 ff 19 00 60 17 49 00 18 ff 19 00 8e f8 4c 00 01 00 00 00 20 a1 73 00 28 b9 77 00 70 ff 19 00 80 f7 4c 00 78 72 |
| 0x0019FF22 | a4 7f f0 f8 4c 00 f0 f8 4c 00 00 70 3f 00 |

Partiendo de que ebp está en 0x0019FEDC (valor 04), cómo tenemos que buscar -1Ch (-28) vamos contando 28 posiciones hacia la izquierda.

Memoria 1

Dirección: 0x0019FE18

```

0x0019FE8A  49 00 e8 6a 58 00 70 3b 73 00 b0 6b 58 00 01 6a 58 00 b0 6b 58 00 c
0x0019FEB0  c8 6b 58 00 08 00 00 00 00 00 00 00 b0 6b 58 00 68 fe 19 00 60 ff 1
0x0019FED6  49 00 ec fe 19 00 00 10 33 00 01 00 00 00 01 00 00 00 01 00 00 00 0
0x0019FEFC  04 ff 19 00 60 17 49 00 18 ff 19 00 8e f8 4c 00 01 00 00 00 58 ca 6

```

Desensamblado main.cpp

Dirección: Stage2(void)

Opciones de visualización

| Dirección | Hex | Asm | Comentario |
|-----------|----------------------|-----|---|
| 004955A8 | EB DD | jmp | Stage2+17h (0495587h) |
| 004955AA | C7 45 F8 01 00 00 00 | mov | dword ptr [ebp-8],1 |
| 004955B1 | 8D 5D E4 | lea | ebx,[ebp-1Ch] |
| 004955B4 | 8B 03 | mov | eax,dword ptr [ebx] |
| 004955B6 | 2B 43 0C | sub | eax,dword ptr [ebx+0Ch] ≤ 1 ms transcur |
| 004955B9 | 83 F8 F9 | cmp | eax,0FFFFFFF9h |

Automático

| Nombre | Valor |
|--------|----------|
| EAX | 00000001 |

Como podemos observar, el valor de esa posición equivale a la **parte menos significativa de la primera contraseña introducida**.

Ahora buscamos el valor de dword ptr [ebx+0Ch], realizando el mismo proceso.

Nos dirigimos a la dirección de memoria de ebx 0x0019FEE0 (valor 01) y contamos +0C (+12) posiciones. En este caso contamos hacia la derecha.

Como se puede observar aparecen todas las contraseñas introducidas hasta ahora, y en la posición indicada aparece la **parte menos significativa de la última contraseña introducida**.

Memoria 1

Dirección: 0x0019FEE0

```

0x0019FEE0  01 00 00 00 01 00 00 00 01 00 00 00 08 00 00 00 04 00
0x0019FF06  19 00 8e f8 4c 00 01 00 00 00 58 ca 6e 00 10 6e 6f 00
0x0019FF2C  00 10 33 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x0019FF52  58 00 00 00 00 00 20 ff 19 00 00 00 00 00 cc ff 19 00

```

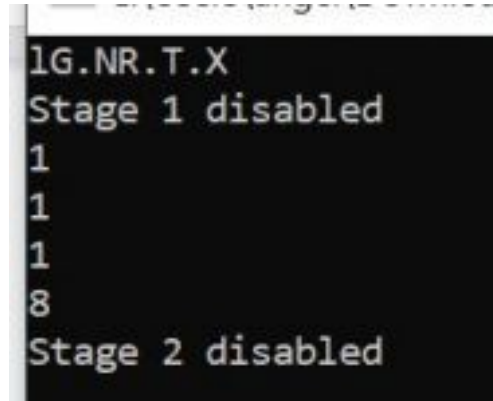
Por lo tanto, tenemos que buscar que la **resta entre la parte menos significativa de la primera contraseña introducida y la parte menos significativa de la segunda** dé como resultado FFFFFFFF9h -> -7 en decimal.

De esa forma eax tomará el valor FFFFFFFF9h y la comparación valdrá 0, por lo que no se saltará en el salto condicional y se obtendrán los valores necesarios para el salto que evita la explosión de la bomba.

| | | | | |
|----------|----------------------|-----|-------------------------|---------------|
| 004955B6 | 2B 43 0C | sub | eax,dword ptr [ebx+0Ch] | |
| 004955B9 | 83 F8 F9 | cmp | eax,0FFFFFFF9h | ≤ 1 ms transc |
| 004955BC | 75 07 | jne | Stage2+55h (04955C5h) | |
| 004955BE | C7 45 F8 00 00 00 00 | mov | dword ptr [ebp-8],0 | |

| Nombre | | Valor |
|--------|--|----------|
| EAX | | FFFFFFF9 |

Una contraseña válida es **1 1 1 8**



TERCERA CONTRASEÑA:

Revisando el código vemos que es necesario que una comparación de un resultado igual y la comparación siguiente de un resultado distinto.

Empecemos estudiando más a fondo la primera comprobación, cmp edx,dword ptr [ebp-10h].

Edx viene dado por un **AND binario** entre la segunda contraseña y un 1 (siguiendo el orden secuencial `and edx,1` -> `mov dword ptr [ebp-0Ch],edx` y finalmente -> `mov edx,dword ptr [ebp-0Ch]`).

| | | | | |
|----------|----------|-----|-------------------------|--|
| 0049560F | 8B 55 F8 | mov | edx,dword ptr [ebp-8] | |
| 00495612 | 83 E2 01 | and | edx,1 | |
| 00495615 | 89 55 F4 | mov | dword ptr [ebp-0Ch],edx | |
| 00495638 | 3B 55 F0 | cmp | edx,dword ptr [ebp-10h] | |

Por otra parte, en **dword ptr [ebp-10h]** tiene el valor de realizar un AND binario entre la primera contraseña y 100h (como vemos secuencialmente en `mov eax,dword ptr [ebp-4],and eax,100h`) y aplicarle posteriormente un reformato, que elimina sus últimos 8 bits y añade 8 ceros por la derecha.


```

00495618 8B 45 FC      mov     eax,dword ptr [ebp-4]
0049561B 25 00 01 00 00  and     eax,100h
00495620 C1 F8 08      sar     eax,8
00495623 89 45 F0      mov     dword ptr [ebp-10h],eax

```

Como tienen que ser iguales, pondremos **1** como segunda contraseña, de forma que **edx valga 1** (and de 1 y 1=1).

```

00495612 83 E2 01      and     edx,1    ≤ 1 ms transcurridos
00495615 89 55 F4      mov     dword ptr [ebp-0Ch],edx
00495618 8B 45 FC      mov     eax,dword ptr [ebp-4]
0049561B 25 00 01 00 00  and     eax,100h
00495620 C1 F8 08      sar     eax,8

```

| Automático | |
|------------|----------|
| Nombre | Valor |
| EAX | 00586BB0 |
| ECX | 0019FF60 |
| EDX | 00000001 |

Ahora vemos el valor binario de 100h (100000000). Tras aplicarle el formateo visto anteriormente nos queda 000000001, es decir, **1**. Al igual que antes, pondremos el mismo valor en el AND binario para que el resultado sea 1.

De esta forma si ponemos como primera contraseña el valor de 100h en decimal (**256**) ambas serán iguales y se cumplirá la primera condición.

```

00495623 89 45 F0      mov     dword ptr [ebp-10h],eax
00495626 8B 4D FC      mov     ecx,dword ptr [ebp-4]
00495629 81 E1 00 40 00 00  and     ecx,4000h
0049562F C1 F9 0E      sar     ecx,0Eh
00495632 89 4D EC      mov     dword ptr [ebp-14h],ecx
00495635 8B 55 F4      mov     edx,dword ptr [ebp-0Ch]

```

| Automático | |
|------------|----------|
| Nombre | Valor |
| EAX | 00000001 |

La segunda condición es cmp dword ptr [ebp-14h],0

```

00495626 8B 4D FC      mov     ecx,dword ptr [ebp-4]
00495629 81 E1 00 40 00 00  and     ecx,4000h
0049562F C1 F9 0E      sar     ecx,0Eh
00495632 89 4D EC      mov     dword ptr [ebp-14h],ecx

```

Mirando el orden secuencial (mov ecx,dword ptr [ebp-4]

-> and ecx,4000h -> sar ecx,0Eh -> mov dword ptr [ebp-14h],ecx)

vemos que se parte de la primera contraseña, se hace un AND binario con 4000h y se formatea igual que en el caso anterior, pero esta vez con 14 bits en lugar de 8. Partimos de 256 como primera contraseña y 1 como segunda.

Como para esta condición no se necesita la segunda contraseña, podemos concluir que **1 es la segunda contraseña definitiva**. Sin embargo, ahora la primera contraseña tiene que ser distinta de 0 al hacer el and binario.

Si continuáramos usando 256, el AND nos devolvería 0, y el formateo lo mismo. En este caso nos interesa que el resultado devuelto sea distinto de 0 ya que el salto es jne.

| | | | | |
|----------|-------------------|-----|-------------------------|----------------------|
| 00495629 | 81 E1 00 40 00 00 | and | ecx,4000h | |
| 0049562F | C1 F9 0E | sar | ecx,0Eh | ≤ 1 ms transcurridos |
| 00495632 | 89 4D EC | mov | dword ptr [ebp-14h],ecx | |
| 00495635 | 8B 55 54 | mov | edx,dword ptr [ebp-0Ch] | |

| Automático | |
|------------|----------|
| Nombre | Valor |
| EAX | 00000001 |
| ECX | 00000000 |

Nos interesa que el resultado del AND y el formateo de 1, por lo que escribimos el mismo valor, 4000h. Sin embargo hay que tener en cuenta que partimos de 100h (ya que sino no entra por la primera comparación) por lo que debemos introducir 4100h como **primera contraseña**, es decir, **16640**.

De esta forma la operación AND devolverá 1000000000000000, que tras el formateo quedará como 0000000000000001. Al ser distinto de 0, es una contraseña válida.

Por lo tanto una contraseña válida es **16640 1**

```
lg.NR.T.X
Stage 1 disabled
1
1
1
8
Stage 2 disabled
16640
1
Stage 3 disabled
Wow, you've just saved the Earth!
```

DESACTIVAR COMPLETAMENTE LA BOMBA:

Como no queremos destruir el mundo por error, creemos que lo más seguro es desactivar completamente la bomba. De esta forma no será necesario validar entradas. Una forma sencilla de hacerlo es eliminar la llamada a Explode().

Si modificamos el ejecutable y cambiamos el código por instrucciones NOP la bomba nunca llegará a estallar. Como se trata de un programa de orden secuencial, las instrucciones se irán ejecutando hasta llegar a Defuse(), que desactiva la bomba.

Para lograr este objetivo tenemos que utilizar el programa HxD, que nos permite modificar el código del ejecutable. De esta forma solo tenemos que buscar la instrucción a modificar y cambiar su código por NOP (importante recordar que estamos trabajando con valores hexadecimales, por que lo que hay que seleccionar esa opción al buscar y reemplazar código)

En la ventana de depuración vemos el siguiente código:

00495552 E8 F9 FE FF FF call Explode (0495450h) que será el que tenemos que reemplazar con instrucciones nop en HxD.

```
00041060 00 8A 45 FC 8B E5 5D C3 CC CC CC CC CC CC CC CC .šEu<āJĀīīīīīīīīīī
00041070 55 8B EC 83 EC 08 89 4D F8 8B 45 F8 8B 08 51 8D U<īfī.ŧMø<Eø<.Q.
00041080 4D FF E8 F9 FE FF FF 8B 55 F8 C7 02 00 00 00 00 Mŷēūpyŷ<UøÇ.....
00041090 8B E5 5D C3 CC CC CC CC CC CC CC CC CC CC CC CC <āJĀīīīīīīīīīīīīīīīī
000410A0 55 8B EC 83 EC 08 8B 45 0C 83 38 00 75 0B 8B 4D U<īfī.<E.f8.u.<M

00041060 00 8A 45 FC 8B E5 5D C3 CC CC CC CC CC CC CC CC .šEu<āJĀīīīīīīīīīī
00041070 55 8B EC 83 EC 08 89 4D F8 8B 45 F8 8B 08 51 8D U<īfī.ŧMø<Eø<.Q.
00041080 4D FF 90 90 90 90 90 8B 55 F8 C7 02 00 00 00 00 Mŷ.....ŷUøÇ.....
00041090 8B E5 5D C3 CC CC CC CC CC CC CC CC CC CC CC CC <āJĀīīīīīīīīīīīīīīīī
000410A0 55 8B EC 83 EC 08 8B 45 0C 83 38 00 75 0B 8B 4D U<īfī.<E.f8.u.<M
```

```
0049554E 74 0C          je          Stage1+4Ch (049555Ch)
00495550 6A 01          push         1
> 00495552 90          nop          ≤ 1 ms transcurridos
00495553 90          nop
00495554 90          nop
00495555 90          nop
00495556 90          nop
00495557 83 C4 04       add          esp,4
0049555A EB 0F       jmp          Stage1+5Bh (049556Bh)
0049555C 68 20 5A 55 00 push         555A20h
```

De esta forma stage1 se desactiva independientemente de su texto de entrada.

```

jejexd
Stage 1 disabled

```

Ahora solo tenemos que repetir lo mismo con las otras dos fases:

004955CD E8 7E FE FF FF call Explode (0495450h)

```

000049B0 00 8D 5D E4 8B 03 2B 43 0C 83 F8 F9 75 07 C7 45 ...]ä<.+C.føùu.ÇE
000049C0 F8 00 00 00 00 83 7D F8 00 74 0C 6A 02 E8 7E FE ø....f}ø.t.j.ë~p
000049D0 FF FF 83 C4 04 EB 0F 68 48 5A 55 00 6A 02 E8 DD fÄ.ë.hHZU.j.èY
000049E0 FE FF FF 83 C4 08 5B 8B E5 5D C3 CC CC CC CC CC pÿÿfÄ.<[ä]Äiiiiii

```

```

000049B0 00 8D 5D E4 8B 03 2B 43 0C 83 F8 F9 75 07 C7 45 ...]ä<.+C.føùu.ÇE
000049C0 F8 00 00 00 00 83 7D F8 00 74 0C 6A 02 90 90 90 ø....f}ø.t.j....
000049D0 90 90 83 C4 04 EB 0F 68 48 5A 55 00 6A 02 E8 DD ..fÄ.ë.hHZU.j.èY
000049E0 FE FF FF 83 C4 08 5B 8B E5 5D C3 CC CC CC CC CC pÿÿfÄ.<[ä]Äiiiiii

```

```

004955C9 74 0C                    je                    Stage2+67h (04955D7h)
004955CB 6A 02                    push                 2
004955CD 90                        nop
004955CE 90                        nop
004955CF 90                        nop
004955D0 90                        nop
004955D1 90                        nop
004955D2 83 C4 04                add                  esp,4
004955D5 EB 0F                   jmp                  Stage2+76h (04955E6h)

```

```

jejexd
Stage 1 disabled
monopoly
Stage 2 disabled

```

00495645 E8 06 FE FF FF call Explode (0495450h)

```

00004A30 F9 0E 89 4D EC 8B 55 F4 3B 55 F0 75 06 83 7D EC ù.%Ml<Uô;Uôu.f}}i
00004A40 00 75 0C 6A 03 E8 06 FE FF FF 83 C4 04 EB 0F 68 .u.j.ë.pÿÿfÄ.ë.h
00004A50 70 5A 55 00 6A 03 E8 65 FE FF FF 83 C4 08 8B E5 pZU.j.èepÿÿfÄ.<[ä
00004A60 5D C3 CC CC CC CC CC CC CC CC CC CC CC CC CC CC jÄiiiiiiiiiiiiiiii

```

```

00004A30 F9 0E 89 4D EC 8B 55 F4 3B 55 F0 75 06 83 7D EC ù.%Ml<Uô;Uôu.f}}i
00004A40 00 75 0C 6A 03 90 90 90 90 90 83 C4 04 EB 0F 68 .u.j.....fÄ.ë.h
00004A50 70 5A 55 00 6A 03 E8 65 FE FF FF 83 C4 08 8B E5 pZU.j.èepÿÿfÄ.<[ä
00004A60 5D C3 CC CC CC CC CC CC CC CC CC CC CC CC CC CC jÄiiiiiiiiiiiiiiii

```

| | | | |
|----------|----------|------|-----------------------|
| 00495641 | 75 0C | jne | Stage3+5Fh (049564Fh) |
| 00495643 | 6A 03 | push | 3 |
| 00495645 | 90 | nop | |
| 00495646 | 90 | nop | |
| 00495647 | 90 | nop | |
| 00495648 | 90 | nop | |
| 00495649 | 90 | nop | |
| 0049564A | 83 C4 04 | add | esp,4 |
| 0049564D | EB 0F | jmp | Stage3+6Eh (049565Eh) |

```

jejexd
Stage 1 disabled
monopoly
Stage 2 disabled
Stage 3 disabled
Now, you've just saved the Earth!

```