HTB x Uni CTF 2020 - Quals

Challenge: ircware (reversing)

by santatecla[UOC]

“During a routine check on our servers we found this suspicious binary, but when analyzing it we couldn't get it to do anything. We assume it's dead malware but maybe something interesting can still be extracted from it?”

OS: “Ubuntu:20.04”
Tools: file, strace, nc, ircd-hybrid, irssi, sed, strings, radare2

En este reto se entrega un .zip que contiene un archivo llamado “ircware”.

```
$ file ircware
ircware: ELF 64-bit LSB executable, x86-64, version 1 (SYSV), dynamically linked, interpreter
/lib64/ld-linux-x86-64.so.2, stripped
```

Lo examino y veo que es un ELF de 64bits linkado dinámicamente y strippeado. Lo ejecuto y muestra el siguiente mensaje:

```
$ ./ircware
EXCEPTION! ABORT%
```

Lo ejecuto con “strace” para ver si hace alguna syscall interesante:

```
$ strace ./ircware
…
connect(3, {sa_family=AF_INET, sin_port=htons(8000), sin_addr=inet_addr("127.0.0.1")}, 16) = -1
ECONNREFUSED (Connection refused)
write(1, "EXCEPTION! ABORT\0", 17EXCEPTION! ABORT)      = 17
exit(1)                                 = ?
+++ exited with 1 +++
```

Parece que intenta conectarse a localhost al puerto 8000, pongo “netcat” a la escucha y lo vuelvo a ejecutar:

```
$ nc -nvlp 8000
Listening on 0.0.0.0 8000
Connection received on 127.0.0.1 35674
NICK ircware_2760
USER ircware 0 * :ircware
JOIN #secret
```

Está intentando conectarse a un servidor irc a la sala “#secret”, instalo el servidor irc “ircd-hybrid” y el cliente “irssi”, edito la configuración del servidor para que escuche por el puerto 8000, lo arranco y me conecto a la sala “#secret”:

```
$ sudo sed -i 's/6665 .. 6669/8000/g' /etc/ircd-hybrid/ircd.conf
$ sudo service ircd-hybrid start
$ irssi
        /connect 127.0.0.1 8000
        /join #secret
```

Si ejecuto el binario, puedo ver en el irc el siguiente mensaje:

```
-!- ircware_7002 [ircware@i.love.debian.org] has joined #secret
```

El programa puede conectarse, uso strings para ver si hay alguna cadena interesante en el binario:

```
$ strings ./ircware
…
PRIVMSG #secret :@exec
PRIVMSG #secret :@flag
RJJ3DSCP
PRIVMSG #secret :@pass
...
```

Me llaman la atención 4 cadenas, después de darle vueltas y probar, me doy cuenta de que “exec”, “flag” y “pass” son ordenes que le puedo pasar al programa a traves del chat:

```
14:32 <@playerRed> @flag
14:32 < ircware_7456> Requires password
14:32 <@playerRed> @pass asd
14:32 < ircware_7456> Rejected
14:32 <@playerRed> @exec asd
14:32 < ircware_7456> Requires password
```

Ejecuto el binario con “radare2”:

```
$ r2 -A ./ircware
        pdf
...
0x0040023c      e84e000000     call fcn.CONNECT
0x00400241      85c0           test eax, eax
0x00400243      0f8873040000   js 0x4006bc
0x00400249      48b818106000.  movabs rax, str.NICK_ircware_0000 ; 0x601018 ; "NICK
ircware_0000"
0x00400253      e8a3000000     call fcn.NICK
0x00400258      48b82a106000.  movabs rax, str.USER_ircware_0___:ircware ; 0x60102a ; "USER
ircware 0 * :ircware"
0x00400262      e894000000     call fcn.USER
0x00400267      48b844106000.  movabs rax, str.JOIN__secret ; 0x601044 ; "JOIN #secret"
0x00400271      e885000000     call fcn.JOIN
0x00400276      e858000000     call fcn.004002d3
0x0040027b      e8c9000000     call fcn.MAIN
0x00400280      ebf4           jmp 0x400276
...
```

He renombrado las funciones más interesantes para que se lea mejor. “fcn.CONNECT” intenta conectarse, si no tiene éxito termina la ejecución, si tiene éxito se ejecuta las funciones “fcn.NICK” que introduce el nick, “fcn.USER” que introduce el username, “fcn.JOIN” que se une a la sala y la función “fcn.MAIN” que ejecuta el bucle que contiene las órdenes que se reciben por el chat.

Entro en la función “fcn.MAIN”, poniendo breackpoints y editando registros consigo que el comando “@flag” me devuelva algo distinto a ”Requires password” pero solo es una cadena sin sentido. Entonces entiendo que para que muestre la flag hay que proporcionar la contraseña válida.

Busco la parte de la función que valida la contraseña:

```
        s fcn.MAIN
        pdf
...
0x004003ed     488d3d5c0d20.     lea rdi, qword str.RJJ3DSCP ; 0x601150 ; "RJJ3DSCP"
0x004003f4     488d1d4c0d20.     lea rbx, qword [0x00601147] ; "RJJ3DSCP"
0x004003fb     4831d2            xor rdx, rdx              ; rdx = 0
0x004003fe     4889f1            mov rcx, rsi
0x00400401     8a06              mov al, byte [rsi]
0x00400403     8803              mov byte [rbx], al
0x00400405     3c00              cmp al, 0
0x00400407     7435              je 0x40043e
0x00400409     3c0a              cmp al, 0xa               ; 10
0x0040040b     7431              je 0x40043e
0x0040040d     3c0d              cmp al, 0xd               ; 13
0x0040040f     742d              je 0x40043e
0x00400411     483b15410d20.     cmp rdx, qword [0x00601159] ; [0x601159:8]=8
0x00400418     774c              ja 0x400466
0x0040041a     3c41              cmp al, 0x41              ; 65
0x0040041c     720e              jb 0x40042c
0x0040041e     3c5a              cmp al, 0x5a              ; 90
0x00400420     770a              ja 0x40042c
0x00400422     0411              add al, 0x11              ; 17
0x00400424     3c5a              cmp al, 0x5a              ; 90
0x00400426     7604              jbe 0x40042c
0x00400428     2c5a              sub al, 0x5a              ; 90
0x0040042a     0440              add al, 0x40              ; 64
0x0040042c     3807              cmp byte [rdi], al
0x0040042e     7536              jne 0x400466
0x00400430     48ffc2            inc rdx
0x00400433     48ffc3            inc rbx
0x00400436     48ffc6            inc rsi
0x00400439     48ffc7            inc rdi
0x0040043c     ebc3              jmp 0x400401
0x0040043e     4889ce            mov rsi, rcx
0x00400441     483b15110d20.     cmp rdx, qword [0x00601159] ; [0x601159:8]=8
0x00400448     751c              jne 0x400466
0x0040044a     48ff05b70b20.     inc qword [0x00601008]
0x00400451     488d353a0c20.     lea rsi, qword str.Accepted ; 0x601092 ; "Accepted"
0x00400458     488b0d3c0c20.     mov rcx, qword [0x0060109b] ; [0x60109b:8]=9 ; "\t"
0x0040045f     e821000000        call fcn.00400485
...
```

He resaltado las instrucciones más relevantes para que sea más legible, este código hace lo siguiente:

1. Se ejecuta "**0x0040045f ; call fcn.00400485**" cuando la contraseña es válida. Esta función se ejecuta si "rdx=8" (**0x00400441 & 0x00400448**).
2. rdx se incrementa a cada iteracción del bucle(**0x00400401 - 0x0040043c**) que comprueba la contraseña caracter a caracter.
3. La contraseña hardcodeada se guarda en [rdi] (**0x004003ed**) y la que se introduce por el chat se guarda en [rsi], de donde se saca caracter a caracter al registro "al"(**0x00400401**).

Si el código solamente hiciera lo explicado, solamente habría que introducir la contraseña hardcodeada, guardada en [rdi](RJJ3DSCP), el problema es que antes de comprobar cada caracter de la contraseña que se introduce por el chat, se le realiza un número variable de cambios, según **2 condiciones**:

1. AL < 0x41 ó AL > 0x5A => No se modifica el carácter y no se hacen más comprobaciones.

Si no se cumple la condición 1, "AL = AL + 0x11" (**0x00400422**) y comprueba la coondición 2.

2. AL <= 0x5A => No se vuelve a modificar el caracter.

Si no se cumple la condición 2, "AL = AL - 0x1A" (**0x00400428 & 0x0040042a**).

Entonces se ve claramente que hay **3 opciones**:

1. Se cumple la condición 1: el caracter hardcodeado es el que hay que introducir.
2. Se cumple la condición 2: hay que introducir el caracter hardcodeado menos 0x11.
3. No se cumple ninguna: hay que introducir el caracter hardcodeado más 0x9.

Una vez comprendido el algoritmo, se puede aplicar al revés a la contraseña hardcodeada:

RJJ3DSCP == 0x 52 4A 4A 33 44 53 43 50

1. **0x52**: Se ve claramente que no cumple la condición 1.
Entonces si tampoco cumpliera la condicion 2, el carácter introducido seria 0x6C (0x52+0x1A) que si cumpliria la condicion 1. Esto es imposible así que sí ha cumpliado la condición 2 y entonces, el carácter introducido debe ser **0x41** (0x52–0x11).

2. **0x4A**: Se ve claramente que no cumple la condición 1.
Si el carácter introducido tampoco cumpliera la condición 2, sería 0x64 (0x4A+0x1A), que no cumple la condición 1 tampoco y sería 0x53 (0x64 – 0x11), esto sí es posible.
Se deduce entonces que no cumple ninguna condición y originalmente era **0x53** (0x4A + 0x9)

3. **0x4A**: es igual que el 2º.

4. **0x33**: Se ve claramente que cumple la condición 1, entonces no se ha modificado.

5. **0x44**: Se ve claramente que no cumple la condición 1.
Si el carácter introducido tampoco cumpliera la condición 2, sería 0x5E (0x44+0x1A), que no cumple la condición 1 tampoco y sería 0x4D (0x5E – 0x11), esto sí es posible.
Se deduce entonces que no cumple ninguna condición y originalmente era **0x4D** (0x44 + 0x9)

6. **0x53**: Se ve claramente que no cumple la condición 1.
Entonces si tampoco cumpliera la condicion 2, el carácter introducido seria 0x6D (0x53+0x1A) que si cumpliria la condicion 1. Esto es imposible así que sí ha cumpliado la condición 2 y entonces, el carácter introducido debe ser **0x42** (0x53–0x11).

7. **0x43**: Se ve claramente que no cumple la condición 1.
Si el carácter introducido tampoco cumpliera la condición 2, sería 0x5D (0x43+0x1A), que no cumple la condición 1 tampoco y sería 0x4C (0x5D – 0x11), esto sí es posible.
Se deduce entonces que no cumple ninguna condición y originalmente era **0x4C** (0x43 + 0x9)

8. **0x50**: Se ve claramente que no cumple la condición 1.
Si el carácter introducido tampoco cumpliera la condición 2, sería 0x6A (0x50+0x1A), que no cumple la condición 1 tampoco y sería 0x59 (0x6A – 0x11), esto sí es posible.
Se deduce entonces que no cumple ninguna condición y originalmente era **0x59** (0x50 + 0x9)

Si le aplicamos el algoritmo al revés a la contraseña hardcodeada:

RJJ3DSCP 52 4A 4A 33 44 53 43 50

Obtenemos la contraseña válida que espera el programa:

ASS3MBLY 41 53 53 33 4D 42 4C 59

Ya puedo introducir la contraseña por el chat y leer la flag :)

```
14:35 <@playerRed> @pass ASS3MBLY
14:35 < ircware_0216> Accepted
14:35 <@playerRed> @flag
14:35 < ircware_0216> HTB{m1N1m411st1C_fL4g_pR0v1d3r_b0T}
```