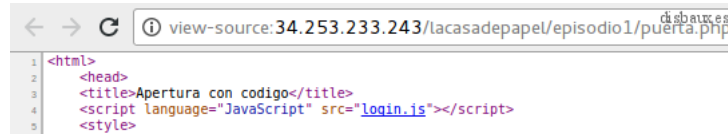


CTF Una-Al-Mes La casa de papel 1ª parte. Write-up

Obtención de los códigos web

Abrimos la página web del CTF: <http://34.253.233.243/lacasadepapel/episodio1/puerta.php>. Observamos un sencillo formulario web con dos cajas de texto, “Código 1” y “Código 2”. Antes de hacer pruebas a lo loco, comprobamos el código fuente de la página:



El código fuente de la página del CTF nos revela un archivo interesante: “login.js”

El archivo JavaScript “login.js” parece ser importante. Lo abrimos y nos encontramos con una única función, llamada **conexion**, que nos dará los dos códigos tras un sencillo proceso de parseado:

```
function conexion(){
    var Password =
        "unescape%28String.fromCharCode%252880%252C%2520108%252C%25
        "2097%252C%2520110%2529%29:KZQWYZLOMNUWC===";
    for (i = 0; i < Password.length; i++)
    {
        if (Password[i].indexOf(code1) == 0)
        {
            var TheSplit = Password[i].split(":");
            var code1 = TheSplit[0];
            var code2 = TheSplit[1];
        }
    }
}
```

El código anterior nos muestra como se generan los dos códigos a partir de la cadena “**Password**”. El terminador que los separa es “:”. Por lo tanto, para el código2 basta coger todos los caracteres después de “:”:

KZQWYZLOMNUWC===

Siendo todo mayúsculas, y con la terminación “===” parecería que estamos ante una codificación [Base32](#). Decodificamos:

```
$ echo -n "KZQWYZLOMNUWC==="|base32 -d
Valencia
```

Para el primer código lo que haremos es coger todos los caracteres antes del “:”. Tenemos esto:

unescape%28String.fromCharCode%252880%252C%2520108%252C%252097%252C%2520110%2529%29

Basta con darse cuenta de que los caracteres que empiezan con % están usando la [codificación hexadecimal en HTML](#). Sabiendo esto, basta con convertirlos a sus correspondientes valores **ASCII** y nos queda:

unescape(String.fromCharCode(80, 108, 97, 110))

Básicamente tenemos una llamada JavaScript simple, que nos devolverá “**Plan**”.

Rellenamos los dos campos del formulario web con “**Plan**” y “**Valencia**”, y obtenemos el password del ZIP:



El password del ZIP tras descubrir como generar Código1 y Código2

Reversing del ejecutable

Descargamos el archivo ZIP y usamos el password "PR0F3S0R&R10" para descomprimirlo. Obtenemos un ejecutable para consola en Windows de 64 bits:

```
$ file episodio1.exe
```

```
episodio1.exe: PE32+ executable (console) x86-64, for MS Windows
```

Lo ejecutamos con **Wine64**:

```
$ wine64 episodio1.exe
```

```
System_Date: 05/16/18
```

```
Wrong date R3m0!
```

```
-----HINT-----
```

```
'La persistencia de la memoria...'
```

```
Press any key to continue...
```

Bien, ahora tenemos 2 maneras diferentes de resolver el reto. Parece evidente que el programa comprueba la fecha del sistema, y si no es la que espera, nos muestra el mensaje de error y el "HINT". La primera manera de resolverlo es, pues, encontrando la fecha correcta a partir del HINT. La segunda es, lógicamente, haciendo reversing. Elegimos la segunda por ser esto un CTF de seguridad informática.

Análisis con radare

Ejecutamos un primer análisis con radare y buscamos la función **main**:

```
r2 -A episodio1.exe
```

```
[x] Analyze all flags starting with sym. and entry0 (aa)
```

```
[x] Analyze len bytes of instructions for references (aar)
```

```
[x] Analyze function calls (aac)
```

```
[x] Use -AA or aaaa to perform additional experimental analysis.
```

```
[x] Constructing a function name for fcn.* and sym.func.* functions (aan)
```

```
-- Setup dbg.fpregs to true to visualize the fpu registers in the debugger view.
```

```
[0x00401500]> afl~main
```

```
0x004011b0 44 834 -> 813 sym.__tmainCRTStartup
```

```
0x00401530 5 595 -> 527 sym.main
```

```
0x0040e960 3 28 -> 24 sym.__main
```

```
0x00419d50 1 6 sym.__getmainargs
```

```
0x00452d40 1 29 sym.std::domain_error::domain_error_std::stringconst__
```

```
0x00452d60 1 36 sym.std::domain_error::_domain_error__
```

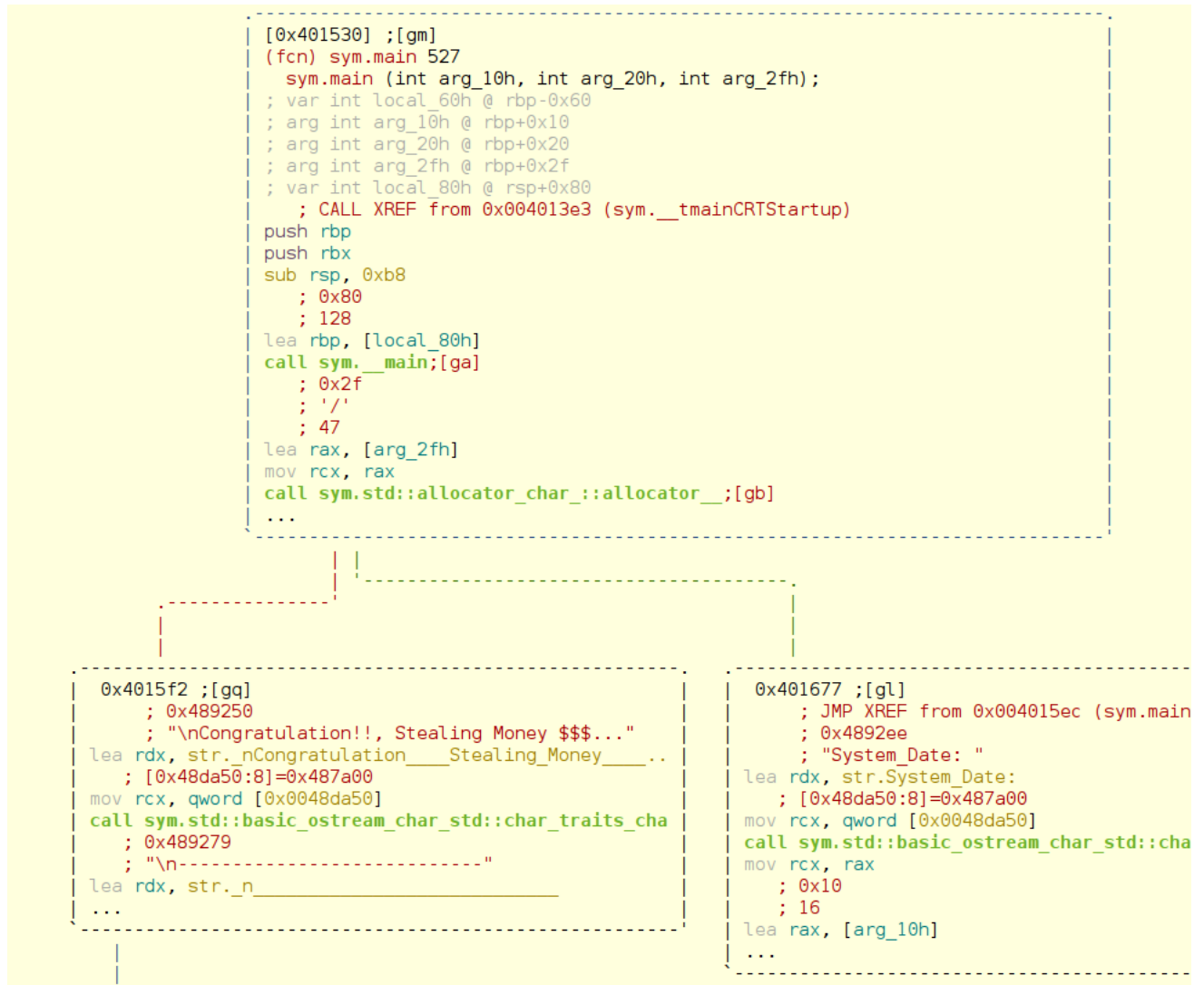
```
0x00452d90 1 15 method.std::domain_error::~domain_error()
```

```
0x0046d410 2 99 -> 139 sub.std::domain_error::domain_error_std::stringconst___410
```

Cuánta porquería, está claro que esto fue escrito en C++ (puag). Vamos directos al **OFFSET** de la función main, pasamos al modo visual, hacemos algo de zoom y observamos el simple flujo del programa:

```
[0x00401500]> s sym.main
```

```
[0x00401530]> VV
```



El flujo del programa. Como se puede apreciar, lo ideal sería "caer" en el nodo de la izquierda.

En la figura anterior, si el flujo del programa cae en el nodo de la izquierda, obtendremos el mensaje de "Congratulations!!, Stealing Money \$\$\$". Es lo que nos interesa, sin duda. ¿Cómo llega aquí? Fácil; se hace **una comparación de fecha**. Si la fecha no es la esperada, saltamos al nodo de la derecha en la figura anterior. ¿Qué fecha es esa? Simplemente recorremos la figura anterior hacia atrás, y observamos que la fecha está hard-codeada dentro del ejecutable:

```
[0x004015f2]> s sym.main
```

```
[0x00401530]> pd 20
```

```

/ (fcn) sym.main 527
sym.main (int arg_10h, int arg_20h, int arg_2fh);
; var int local_60h @ rbp-0x60
; arg int arg_10h @ rbp+0x10
; arg int arg_20h @ rbp+0x20
; arg int arg_2fh @ rbp+0x2f
; var int local_80h @ rsp+0x80
; CALL XREF from 0x004013e3 (sym.__tmainCRTStartup)
0x00401530 55 push rbp
0x00401531 53 push rbx
0x00401532 4881ecb80000 sub rsp, 0xb8
0x00401539 488dac248000 lea rbp, [local_80h] ; 0x80 ; 128
0x00401541 e81ad40000 call sym.__main
0x00401546 488d452f lea rax, [arg_2fh] ; 0x2f ; '/' ; 47
0x0040154a 4889c1 mov rcx, rax
0x0040154d e82e570400 call sym.std::allocator_char::allocator_
0x00401552 488d552f lea rdx, [arg_2fh] ; 0x2f ; '/' ; 47
0x00401556 488d4520 lea rax, [arg_20h] ; 0x20 ; 32
0x0040155a 4989d0 mov r8, rdx
0x0040155d 488d15dc7c08 lea rdx, str.01_23_89 ; 0x489240 ; "01/23/89"

```

La fecha está hard-codeada dentro del binario: 01/23/89".

La fecha se encuentra en el OFFSET apuntado por "str.01_23_89", y es "01/23/89". Esto está en formato americano, con el mes y el día intercambiados. Así que la fecha del sistema debería ser: "23/01/1989".

A partir de aquí, podemos resolver el reto de **al menos de 2 maneras diferentes**:

1. **Cambiamos la fecha** de nuestra VM para que sea 23/01/1989.
2. **Parcheamos la instrucción** de salto tras la comparación de fechas para que el flujo del programa caiga en el nodo de la derecha, sin importar la fecha de nuestro sistema.

Método 1: Cambio de la fecha (pa gandules ;-))

Cambiamos la fecha y ejecutamos de nuevo el binario para obtener la FLAG:

```

root@lud:/home/lud# date --set "01/23/1989"
Mon 23 Jan 00:00:00 GMT 1989
root@lud:/home/lud# date
Mon 23 Jan 00:00:01 GMT 1989
root@lud:/home/lud# exit
lud@lud:~$ wine64 episodiol.exe
fixme:service:schd:database_autostart_services Auto-start service L"MountMgr" failed to start: 2

Congratulation!!, Stealing Money $$$...

Stolen: 1,000,000,000 $

Flag: e30f35ad8d9cb6efc0778539a669fa85
.....
Press any key to continue...

```

Obtención de la flag, método 1: Cambiamos la fecha del sistema.

Método 2: Parcheamos la instrucción de salto (sólo un poco más de trabajo)

Revisamos dónde se hace la comparación de fechas y qué instrucción de salto se ejecuta. Bastará con parchear dicha instrucción de salto para caer siempre en el nodo de la izquierda:

```

call sym.boolstricmp
test al, al
je 0x401677;[gl]

0x4015f2 ;[gq]
; 0x489250
; "\nCongratulation!!, Stealing Money $$$..."
lea rdx, str.nCongratulation____Stealing_Money____...
; [0x48da50:8]=0x487a00

```

La comparación de fechas y el salto; si la fecha no es válida, se ejecutará el salto hacia el OFFSET 0x401677, el nodo de la derecha.

Bastará parchear el **JE** por un **JNE** y pelillos a la mar. Así que obtenemos el **OFFSET** de dicha instrucción y lo cambiamos. Primero debemos salir de radare y ejecutarlo de nuevo con "-w" para poder escribir sobre el binario:

```
$ r2 -w -A episodio1.exe
```

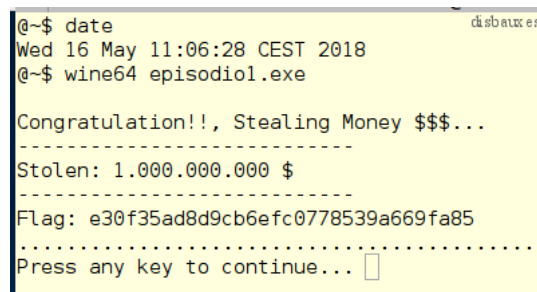
Nos situamos en la instrucción **JE 0x401677**:

```
[0x00000000]> s `c je 0x401677~[0]`  
[0x004015ec]> pd 1  
| ,=< ;-- hit0_0:  
| ,=< 0x004015ec 0f8485000000 je 0x401677
```

Parcheamos el JE por un JNE; comprobamos que el cambio se ha producido, y salimos de Radare:

```
[0x004015ec]> wa jne 0x401677  
Written 6 bytes (jne 0x401677) = wx 0f8585000000  
[0x004015ec]> pd 1  
| ,=< ;-- hit0_0:  
| ,=< 0x004015ec 0f8585000000 jne 0x401677  
[0x004015ec]> q
```

El binario ya está parcheado, así que simplemente lo ejecutamos en nuestro equipo sin cambiar su fecha:



```
@~$ date  
Wed 16 May 11:06:28 CEST 2018  
@~$ wine64 episodio1.exe  
  
Congratulation!!, Stealing Money $$$...  
-----  
Stolen: 1.000.000.000 $  
-----  
Flag: e30f35ad8d9cb6efc0778539a669fa85  
.....  
Press any key to continue... 
```

Obtención de la FLAG mediante parcheo de binario.

La flag, por supuesto, es:

```
UAM{e30f35ad8d9cb6efc0778539a669fa85}
```

Toni Castillo

[@disbauxes](#)