

RETO UNA AL MES 20180515 – EPISODIO1

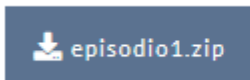
Jose Ignacio de Miguel González. Usuario Nachinho3

Hemos conseguido entrar en la Fábrica Nacional de Moneda y Timbre. Pero una vez dentro, la lanza térmica que usaríamos para abrir la caja fuerte se ha roto. Debes descubrir los códigos para abrirla, y con ello conseguirás la contraseña para el zip del programa que genera la flag y el dinero ;).

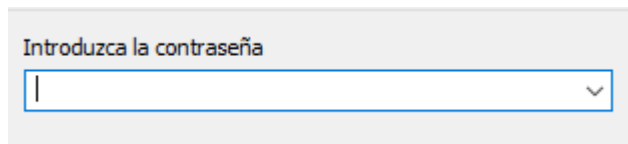
Caja fuerte: <http://34.253.233.243/lacasadepapel/episodio1/puerta.php>

Info: La flag tiene el formato UAM{md5}

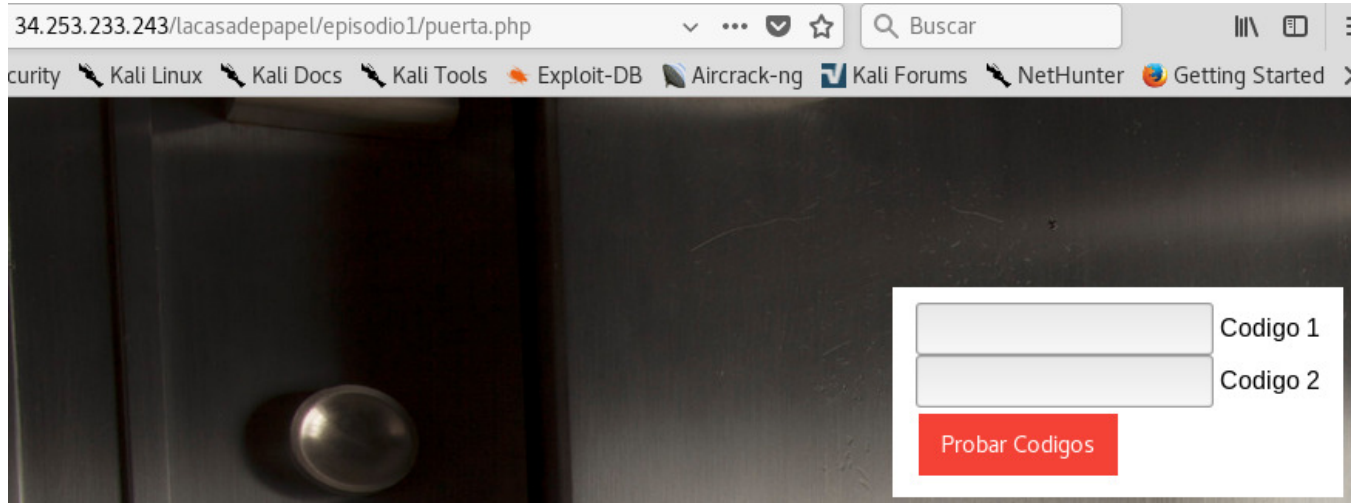
En primer lugar descargamos el fichero que viene en el reto:



Lo descomprimos y lleva un ejecutable dentro, pero al descomprimirlo nos pide un password que no conocemos:

A light gray rectangular box containing the text "Introduzca la contraseña" in a small font. Below the text is a white input field with a blue border and a small downward arrow on the right side.

Vamos a la Web de la caja fuerte a ver si podemos averiguar la contraseña. En dicha Web nos pide dos códigos:



Al mirar el código HTML de la página, vemos que tiene un scroll vertical muy grande, lo que nos da una pista de que podría tener más código más abajo. Conseguimos ver que abajo del todo tiene un comentario:

Cabeceras	Cookies	Parámetros	Respuesta	Tiempos	Traza de la pila
222					
223					
224					
225					
226					
227					
228					
229					
230					
231					
232					
233					
234					
235					
236					
237					
238			<! --		
239			Soy mas de 1234/1234 que de admin/admin		
240			-->		
241			</html>		

Sin embargo, al probar estas combinaciones no conseguimos ningún código, y nos muestra un mensaje de:

No todo es lo que parece... Hay cosas que pueden llevar a confusion. Debes fijarte en los pequeños detalles

Así que tenemos que seguir buscando. Analizamos el código HTML de la página y vemos que incluye un javascript, que además parece propio y no de ninguna librería, pues se llama login.js:

```
<html>
  <head>
    <title>Apertura con codigo</title>
    <script language="JavaScript" src="login.js"></script>
```

Si lo abrimos, lleva comentada una función javascript. Dicha función podría ser una copia olvidada de la función del servidor que valida los códigos. Vamos a probar.

```
/*
function conexion(){
    var Password = "unescape%28String.fromCharCode%252880%252C%2520108%252C%252097%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];
        }
    }
}
*/
```

La función vemos que tiene un string principal, con una cierta ofuscación no demasiado compleja, y después lo que hace es cortarla en dos variables con lo que hay antes y después del carácter ":". Según la función, esos dos valores serán el code1 y el code2.

Vamos a eliminar la ofuscación de manera manual. Primero sacamos el valor original de la cadena:

```
unescape%28String.fromCharCode%252880%252C%2520108%252C%252097%252C%2520110%2529%29:KZQWYZLOMNUWC===
```

Vemos que lleva una codificación del tipo "URL Encoding". En este caso, incluso doble, ya que además del típico "%XX", lleva doble codificación pues el "%" a su vez lo hace como "%25". Por tanto, lo primero convertimos los caracteres "%25", según su código hexadecimal, a su vez, en el carácter "%":

```
unescape%28String.fromCharCode%2880%2C%20108%2C%2097%2C%20110%29%29:KZQWYZLOMNUWC===
```

Ahora ya tenemos una codificación simple, pasamos a convertir cada uno según su hexadecimal:

```
unescape(String.fromCharCode(80, 108, 97, 110)):KZQWYZLOMNUWC===
```

Ahora ya decodificamos la primera parte a partir de los caracteres numéricos, y la segunda parte es un Base32, que decodificamos con cualquier herramienta online, en este caso con https://emn178.github.io/online-tools/base32_decode.html. La cadena definitiva es:

Plan:Valencia|

Por tanto, ya parece que tenemos el code1 y el code2, vamos a insertarlos en la página y efectivamente nos devuelve el password del fichero:

**El código para descomprimir
el zip es:
PR0F3S0R&R10**

Ahora pasamos a la segunda parte, descomprimos el fichero ZIP, y obtenemos el fichero ejecutable. Si lo ejecutamos en nuestra máquina virtual, nos muestra este mensaje:

```
C:\Practicas\Retos\UnaAlMes>episodio1.exe
System_Date: 05/15/18
Wrong date R3m0!

-----HINT-----
'La persistencia de la memoria...'
Presione una tecla para continuar . . .
```

Nos muestra la fecha del sistema y nos dice que la fecha es incorrecta. Por tanto, lo más probable es que solo funcione bajo un rango determinado de fechas. Vamos a realizar un análisis estático del binario, a ver qué vemos. Lo abrimos con IDA, y nos indica que es un binario de 64 bits, así que lanzamos el IDA 64 bits:

```
48 8D 45 20      lea     rax, [rbp+40h+var_20]
49 89 D0         mov     r8, rdx
48 8D 15 DC 7C 08+ lea     rdx, a012389 ; "01/23/89"
48 89 C1         mov     rcx, rax
                  ; try {
E8 34 E8 04 00   call    _ZNStC1EPKcRKSAIcE ; std::string::string(char co
                  ; } // starts at 401567
```

Al abrirlo y consultar el Entry Point, vemos como lee de memoria una cadena que contiene la fecha "01/23/89". Esto tiene toda la pinta de ser la fecha que busca en el sistema, así que vamos a cambiar la fecha del Windows al 23 de enero de 1989, y volvemos a lanzar el ejecutable:

```
C:\Practicas\Retos\UnaAlMes>episodio1.exe
Congratulation!?, Stealing Money $$$...
Stolen: 1.000.000.000 $
Flag: e30f35ad8d9cb6efc0778539a669fa85
Presione una tecla para continuar . . .
```

Efectivamente, el binario ya cumple la condición que necesitaba, y nos muestra la flag. Por tanto la respuesta al reto será **UAM{e30f35ad8d9cb6efc0778539a669fa85}**