

UNIVERSIDAD DEL VALLE DE GUATEMALA

Departamento de Computación

Cifrado de Información

Sección 10



Proyecto 1: Cifrados de Flujo – Desafíos en Seguridad

Brandon Ronaldo Sicay Cumes - 21757

Guatemala, 24 de abril del 2025

Descripción general del proyecto

En el presente proyecto se desarrolló un conjunto de retos prácticos orientados a la comprensión profunda de los algoritmos de cifrado de flujo, sus vulnerabilidades y su implementación segura. Los desafíos se abordaron en un entorno de tipo Capture The Flag (CTF), donde se utilizaron distintas técnicas para recuperar mensajes ocultos (flags) y fragmentos narrativos representados como poneglyphs en archivos de imagen.

Los algoritmos abordados fueron:

- XOR clásico – un cifrado elemental pero ampliamente usado en diversos sistemas embebidos.
- RC4 – un algoritmo de cifrado de flujo ampliamente utilizado en el pasado, hoy considerado inseguro.
- Cifrado personalizado con PRNG – simulando un sistema mal diseñado basado en generadores pseudoaleatorios débiles.
- ChaCha20 – un algoritmo moderno, seguro y eficiente, utilizado en sistemas reales como TLS, SSH y WireGuard.

Cada reto exigió el análisis de estructuras de archivos, el descifrado de mensajes con claves derivadas del carné del estudiante, y la extracción de información oculta en metadatos de imágenes. A lo largo del proyecto se desarrollaron scripts en Python para automatizar procesos, aplicar fuerza bruta de manera eficiente, manipular metadatos EXIF, y trabajar con cifrados estándar y personalizados.

Se estará trabajando sobre la base de este repositorio https://github.com/locano-uvg/ctf_onepice_symmetric_cipher en donde se encontrarán todos los pasos necesarios para generar los challenges necesarios mediante un contenedor de docker.

Contenedores creados:

0.08% / 800% (8 CPUs available)		4.06MB / 7.47GB	
Search		Only show running containers	
Name	Container ID	Image	Port(s)
proyecto1_cifrados	-	-	0.08% 1 hour ago
nami_challenge	3a6e8330ec25	proyecto1_cifrados-nami_image	2203:22 ↗ Show all ports (2)
usopp_challenge	04660b78e9f2	proyecto1_cifrados-usopp_image	2204:22 ↗ Show all ports (2)
zoro_challenge	20b5e9b51bed	proyecto1_cifrados-zoro_image	2202:22 ↗ Show all ports (2)
luffy_challenge	cd3083b267b3	proyecto1_cifrados-luffy_image	2201:22 ↗ Show all ports (2)

Reto 1: Luffy - XOR

Este reto consistió en aplicar conocimientos sobre el cifrado XOR para descifrar un mensaje oculto dentro de archivos distribuidos en un entorno tipo Capture the Flag (CTF). A través del análisis de archivos y metadatos, se buscaba obtener tanto una flag como un texto relacionado con el lore de One Piece (el “poneglyph”).

Proceso

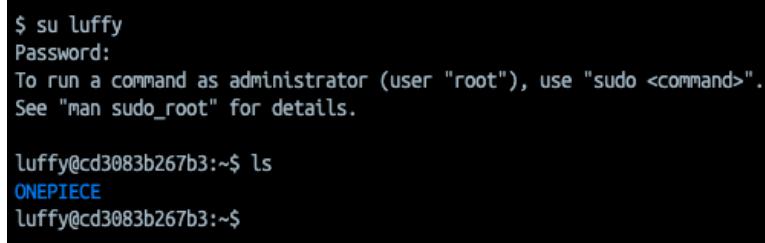
- Como primer paso, y siguiendo instrucciones del repositorio, se inicio sesión en el challenge en cuestión:

Comandos ejecutados:



```
1 su luffy
2 password: onepiece
```

Consola:



```
$ su luffy
Password:
To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.

luffy@cd3083b267b3:~$ ls
ONEPIECE
luffy@cd3083b267b3:~$
```

- Búsqueda de archivos flag.txt en el sistema:

- Para encontrar los archivos distribuidos por el entorno CTF, se utilizó el comando find desde el contenedor Docker. Este comando permite buscar archivos en una estructura de carpetas.



```
1 sudo find / -type f -name flag.txt 2>/dev/null
2
```

- Este comando:

- i. Busca archivos llamados flag.txt en todo el sistema de archivos (/).
- ii. Ignora los errores de permisos (2>/dev/null).
- iii. Esto retornó una lista de rutas que contenían archivos flag.txt repartidos por distintos lugares como Skypiea, Zou, Alabasta, entre otros.

```

Logs   Inspect   Bind mounts   Exec   Files   Stats
luffy@cd3083b267b3:~$ sudo find / -type f -name flag.txt 2>/dev/null
[sudo] password for luffy:
/home/luffy/ONEPIECE/Skypiea/Upper_Yard/Casa_de_Enel/flag.txt
/home/luffy/ONEPIECE/Skypiea/Shandoria/Casa_de_Montblanc_Norland/flag.txt
/home/luffy/ONEPIECE/Zou/Left_Fore_Leg/Casa_de_Kawamatsu/flag.txt
/home/luffy/ONEPIECE/Zou/Right_Hind_Leg/Casa_de_Inuarashi/flag.txt
/home/luffy/ONEPIECE/Zou/Right_Hind_Leg/Casa_de_Kawamatsu/flag.txt
/home/luffy/ONEPIECE/Zou/Right_Fore_Leg/Casa_de_Nekomamushi/flag.txt
/home/luffy/ONEPIECE/Pirate_Island/Pirates_Tavern/Casa_de_Shanks/flag.txt
/home/luffy/ONEPIECE/Alabasta/Katorea/Casa_de_Kohza/flag.txt
/home/luffy/ONEPIECE/Alabasta/Rainbase/Casa_de_Igram/flag.txt
/home/luffy/ONEPIECE/Alabasta/Alubarna/Casa_de_Crocodile/flag.txt
luffy@cd3083b267b3:~$
```

3. Visualización del contenido de todos los flag.txt

- a. Para visualizar el contenido de todos los archivos encontrados, junto con su ruta respectiva, se ejecutó el siguiente script:

```

1
2 sudo find /home/luffy/ONEPIECE -type f -name flag.txt 2>/dev/null | while read filepath; do
3   echo "----- $filepath -----"
4   cat "$filepath"
5   echo
6 done
7
```

- b. Este script:

- i. Itera sobre cada archivo encontrado.
- ii. Muestra su ruta y contenido con cat.
- iii. Como resultado, se encontró una secuencia hexadecimal:

```

----- /home/luffy/ONEPIECE/Zou/Right_Hind_Leg/Casa_de_Inuarashi/flag.txt -----
Has encontrado un obstáculo y hay que superarlo
----- /home/luffy/ONEPIECE/Zou/Right_Hind_Leg/Casa_de_Kawamatsu/flag.txt -----
Has encontrado un lugar peligroso
----- /home/luffy/ONEPIECE/Zou/Right_Hind_Leg/Casa_de_Inuarashi/flag.txt -----
747d767268560904050200540406540707525052005451000300540e00030b0702560f5154
----- /home/luffy/ONEPIECE/Zou/Right_Fore_Leg/Casa_de_Nekomamushi/flag.txt -----
Has encontrado un enemigo y ha tenido que huir
----- /home/luffy/ONEPIECE/Pirate_Island/Pirates_Tavern/Casa_de_Shanks/flag.txt -----
Has encontrado un lugar peligroso
```

4. Aplicación del cifrado XOR para descifrar la flag

- a. Se utilizó un script en Python para descifrar esta secuencia. Se probó una interpretación donde la clave era el carné del estudiante 21757, tratado como cadena ASCII (archivo [xorHex.py](#) en utils)

```
● ● ●
1 import binascii
2
3 # Mensaje cifrado en hexadecimal
4 cipher_hex = "747d767268560904050200540406540707525052005451000300540e00030b0702560f5154"
5 cipher_bytes = bytes.fromhex(cipher_hex)
6
7 # Interpretación 1: clave como caracteres ASCII
8 key_ascii = b"21757"
9 decoded_ascii = bytes([b ^ key_ascii[i % len(key_ascii)] for i, b in enumerate(cipher_bytes)])
10
11 print(" Descifrado usando clave ASCII '21757':")
12 print(decoded_ascii.decode('utf-8', errors='replace'))
13
14
15
16 RESULTADOS: -----
17 Descifrado usando clave ASCII '21757':
18 FLAG_d83052e33c56eee2ef542e954965c8ce
```

b. Esto nos dio la flag

FLAG_d83052e33c56eee2ef542e954965c8ce

5. Búsqueda de archivos ZIP o imágenes (poneglyphs)

- Se utilizó find para buscar archivos con extensiones de imagen o comprimidos:

```
● ● ●
1 sudo find /home/luffy/ONEPIECE -type
2 f \(-iname "*.png" -o -iname "*.jpg" -o -iname "*.jpeg" -o -iname "*.gif" -o -iname "*.bmp" -o -iname "*.webp" -o -iname "*.zip" \) 2>/dev/null
3
```

b. Esto arrojó como resultado:

```
luffy@cd3083b26/b3:~/ONEPIECE$ sudo find /home/luffy/ONEPIECE -type 1
zip" \) 2>/dev/null
[sudo] password for luffy:
/home/luffy/ONEPIECE/Alabasta/Alubarna/Casa_de_Igaram/poneglyph.zip
```

6. Extracción del ZIP dentro del contenedor

- El archivo .zip fue extraído con:

```
● ● ●
1 unzip /home/luffy/ONEPIECE/Alabasta/Alubarna/Casa_de_Igaram/poneglyph.zip
2 -d /home/luffy/extracted_poneglyph
3
```

b. Este comando:

- i. Descomprime el contenido del ZIP en el directorio /home/luffy/extracted_poneglyph.
- ii. Dentro del archivo extraído se encontró una imagen jpeg con el nombre poneglyph-1.jpeg.
- c. Luego del archivero de docker se guardó hacia mi máquina local.



- 7. Extracción del texto cifrado en los metadatos de la imagen
 - a. Se utilizó el código del archivo *extract_text_from_image.py* proporcionado en el repositorio base.
 - b. El mensaje extraído fue cifrado con el mismo esquema XOR, y se descifró usando el mismo carné como clave (21757).

```

 1 # Uso del código para descifrar la imagen
 2 # Ejemplo de uso
 3 image_path = "Proyecto1_Cifrados/extracted_poneglyph/poneglyph-1.jpeg"
 4 student_id = input("Introduce tu carné para descifrar el mensaje: ")
 5 texto_cifrado = extraer_texto_metadata(image_path)
 6 decrypted_text = xor_cipher(texto_cifrado, student_id)
 7 print(decrypted_text)
 8
 9
10
11

```

- c. El resultado fue:

```

 1 b'Crocodile targeted the Arabasta Kingdom because of '
 2 'its Poneglyph, which contained information on the whereabouts of Pluton, '

```

En resumen:

- Flag encontrada:
 - FLAG_d83052e33c56eee2ef542e954965c8ce
- Texto del poneglyph
 - Crocodile targeted the Arabasta Kingdom because of its Poneglyph, which contained information on the whereabouts of Pluton,
 -

Reto 2: Zoro - Rompiendo RC4

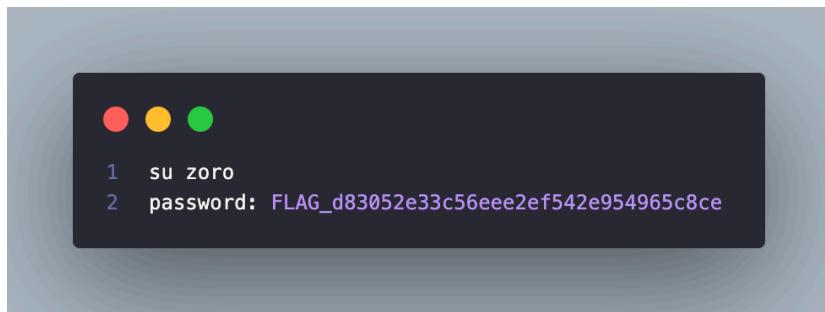
Este reto consistió en aplicar técnicas de análisis y descifrado sobre un flujo de datos cifrado utilizando el algoritmo RC4, una técnica de cifrado de flujo que fue ampliamente utilizada pero que hoy se considera insegura debido a varias vulnerabilidades conocidas.

El propósito era identificar una flag y extraer un mensaje oculto en una imagen (poneglyph), empleando una clave proporcionada (el carné del estudiante) y comprendiendo el funcionamiento interno del algoritmo RC4.

Proceso

1. Como primer paso, se inicio sesión en el challenge en cuestión, y como fue costumbre en previos ejercicios de CTF, como password se utilizó la flag encontrada dle reto pasado:

Comandos ejecutados:



Consola:

```
$ su zoro
Password:
To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.

zoro@20b5e9b51bed:~$
```

2. Búsqueda de archivos flag.txt y visualización en el sistema:

- d. Para visualizar el contenido de todos los archivos encontrados, junto con su ruta respectiva, se ejecutó el mismo script del challenge pasado:

```

1  sudo find /home/zoro/ONEPIECE -type f -name flag.txt 2>/dev/null | while read filepath; do
2    echo "---- $filepath ----"
3    cat "$filepath"
4    echo
5  done

```

- e. Este script:
- Itera sobre cada archivo encontrado.
 - Muestra su ruta y contenido con cat.
 - se identificó un archivo importante: /home/zoro/ONEPIECE/Zou/Right_Fore_Leg/Casa_de_Inuarashhi(flag.txt)
 - Como resultado, se encontró una secuencia hexadecimal:

```

cat "$filepath"
echo
done
----- /home/zoro/ONEPIECE/Skypiea/Angel_Beach/Casa_de_Gan_Fall/flag.txt -----
Has encontrado un mapa que apunta a Right_Fore_Leg
----- /home/zoro/ONEPIECE/Zou/Left_Belly_Fortress/Casa_de_Kawamatsu/flag.txt -----
Has encontrado un amigo y han decidido viajar juntos
----- /home/zoro/ONEPIECE/Zou/Left_Belly_Fortress/Casa_de_Inuarashi/flag.txt -----
Has encontrado un lugar lleno de secretos y misterios que apuntan a Right_Fore_Leg
----- /home/zoro/ONEPIECE/Zou/Left_Hind_Leg/Casa_de_Kawamatsu/flag.txt -----
Te cansaste de buscar y te has quedado dormido
----- /home/zoro/ONEPIECE/Zou/Right_Fore_Leg/Casa_de_Inuarashi/flag.txt -----
7a0e44502ee5c8f505a4fe70550b99005b25037af1ff22171847385360962e0ca9d4d718d
----- /home/zoro/ONEPIECE/Whole_Cake_Island/Liqueur_Island/Casa_de_Katakuri/flag.txt -----
Has encontrado un enemigo y ha tenido que huir
----- /home/zoro/ONEPIECE/Whole_Cake_Island/Cacao_Island/Casa_de_Pudding/flag.txt -----
Has encontrado una pista que apunta a Right_Fore_Leg
----- /home/zoro/ONEPIECE/Whole_Cake_Island/Caramel_Mountain/Casa_de_Big_Mom/flag.txt -----
Has encontrado un enemigo y ha tenido que luchar
----- /home/zoro/ONEPIECE/Pirate_Island/Pirates_Ship/Casa_de_Shanks/flag.txt -----
Has encontrado un enemigo y ha tenido que luchar
----- /home/zoro/ONEPIECE/Alabasta/Spiders_Cafe/Casa_de_Nico_Robin/flag.txt -----
Has encontrado un enemigo y ha tenido que luchar
----- /home/zoro/ONEPIECE/Alabasta/Rainbase/Casa_de_Vivi/flag.txt -----
Has encontrado un enemigo y ha tenido que luchar
zoro@20b5e9b51bed:~$ 

```

3. Descifrado del mensaje usando RC4

- f. Conociendo que el mensaje estaba cifrado con RC4 y la clave de mi carné "21757", se implementó un script en Python con una implementación del algoritmo RC4 (Key Scheduling Algorithm + PRGA) para descifrar el texto: (archivo [xorHexrc4_decrypt.py](#) en utils)

```

1 def rc4(key, data):
2     S = list(range(256))
3     j = 0
4     out = []
5
6     # Key Scheduling Algorithm (KSA)
7     for i in range(256):
8         j = (j + S[i] + key[i % len(key)]) % 256
9         S[i], S[j] = S[j], S[i]
10
11    # Pseudo-Random Generation Algorithm (PRGA)
12    i = j = 0
13    for char in data:
14        i = (i + 1) % 256
15        j = (j + S[i]) % 256
16        S[i], S[j] = S[j], S[i]
17        K = S[(S[i] + S[j]) % 256]
18        out.append(char ^ K)
19
20    return bytes(out)
21
22 if __name__ == "__main__":
23     # Mensaje cifrado en hex
24     cipher_hex = "7a0e44502ee5c8f505a4fe70550b99005b2b5037af1ff22171847385360962e0ca9d4d718d"
25
26     # Convertir a bytes
27     cipher_bytes = bytes.fromhex(cipher_hex)
28
29     # Clave como bytes
30     key = b"21757"
31
32     # Descifrado
33     plain = rc4(key, cipher_bytes)
34
35     print("■ Mensaje descifrado:")
36     print(plain.decode("utf-8", errors="replace"))

```

g. Esto nos dio la flag

FLAG_dc0a0346fb30f3eb96b18577a9e5296f

4. Búsqueda de archivos ZIP o imágenes (poneglyphs)

h. Tal como en el reto anterior, se utilizó find para localizar archivos de imagen y .zip que pudieran contener los poneglyphs:

```

1 zoro@20b5e9b51bed:~$ 
2 sudo find /home/zoro/ONEPIECE -type f \(
3   -iname "*.png" -o -iname "*.jpg" -o -iname "*.jpeg" -o -iname "*.gif" -o -iname "*.bmp" -o -iname "*.webp" -o -iname "*.zip" \)
4 >/dev/null

```

i. Esto arrojó como resultado:

```

zoro@20b5e9b51bed:~$ 
sudo find /home/zoro/ONEPIECE -type f \(
  -iname "*.png" -o -iname "*.jpg" -o -iname "*.jpeg" 
  -iname "*.gif" -o -iname "*.bmp" -o -iname "*.webp" -o -iname "*.zip" \
>/dev/null

```

/home/zoro/ONEPIECE/Whole_Cake_Island/Cacao_Island/Casa_de_Katakuri/poneglyph.zip

zoro@20b5e9b51bed:~\$

5. Extracción del ZIP dentro del contenedor

- j. El archivo .zip fue extraído con:

```
● ● ●
1 unzip /home/zoro/ONEPIECE/Whole_Cake_Island/
2 Cacao_Island/Casa_de_Katakuri/poneglyph.zip -d /home/zoro/extracted_poneglyph
3
```

- k. Este comando:

- i. Esto generó un archivo de imagen poneglyph-2.jpeg, el cual contenía el mensaje oculto en los metadatos EXIF.
- l. Luego del archivero de docker se guardó hacia mi máquina local.

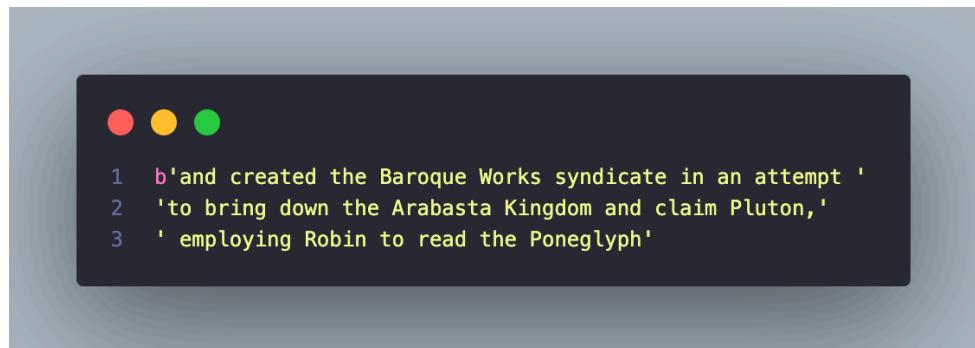


6. Extracción del texto cifrado en los metadatos de la imagen

- m. Se utilizó el código del archivo *extract_text_from_image.py* proporcionado en el repositorio base.
- n. El mensaje extraído fue cifrado con el mismo esquema XOR, y se descifró usando el mismo carné como clave (21757).



o. El resultado fue:



```
1 b'and created the Baroque Works syndicate in an attempt '
2 'to bring down the Arabasta Kingdom and claim PlutoN,'
3 ' employing Robin to read the Poneglyph'
```

En resumen:

- Flag encontrada:
 - FLAG_dc0a0346fb30f3eb96b18577a9e5296f
- Texto del poneglyph
 - and created the Baroque Works syndicate in an attempt to bring down the Arabasta Kingdom and claim PlutoN, employing Robin to read the Poneglyph

Reto 3: Usopp - XOR Cipher Custom

En este reto se presenta un sistema de cifrado de flujo personalizado basado en un generador de números pseudoaleatorios débil (PRNG) en Python. El objetivo fue analizar cómo el uso de un generador predecible permite romper el cifrado y recuperar un mensaje secreto (la flag), así como un texto oculto en los metadatos de una imagen, tal como en los retos anteriores.

A diferencia de los retos anteriores, este desafío implicó el uso de fuerza bruta sobre la semilla del generador para romper el sistema y demostrar lo inseguro que puede ser usar funciones como random.seed en criptografía.

Proceso

1. Como primer paso, se inicio sesión en el challenge en cuestión, y como fue costumbre en previos ejercicios de CTF, como password se utilizó la flag encontrada del reto pasado:

Comandos ejecutados:

```
1 su ussop
2 password: FLAG_dc0a0346fb30f3eb96b18577a9e5296f
```

Consola:

```
$ FLAG_d83052e33c56eee2ef542e954965c8ce
/bin/sh: 6: FLAG_d83052e33c56eee2ef542e954965c8ce: not found
$ su ussop
su: user ussop does not exist or the user entry does not contain all the required fields
$ su usopp
Password:
To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.

usopp@04660b78e9f2:~$
```

2. Búsqueda de archivos flag.txt y visualización en el sistema:

- p. Para visualizar el contenido de todos los archivos encontrados, junto con su ruta respectiva, se ejecutó el mismo script de los dos challenge pasado:

```
1 sudo find /home/usopp/ONEPIECE/ -type f -name flag.txt 2>/dev/null | while read filepath; do
2 echo "----- $filepath -----"
3 cat "$filepath"
4 echo
5 done
6
```

- q. Este script:
- Itera sobre cada archivo encontrado.
 - Muestra su ruta y contenido con cat.
 - se identificó un archivo importante: /home/usopp/ONEPIECE/Pirate_Island/Pirates_Den/Casa_de_Whitebeard/flag.txt
 - Como resultado, se encontró una secuencia hexadecimal:

```

done
----- /home/usopp/ONEPIECE/Zou/Right_Belly_Fortress/Casa_de_Inuarashi/flag.txt -----
Has encontrado un enemigo y ha tenido que luchar
----- /home/usopp/ONEPIECE/Zou/Left_Belly_Fortress/Casa_de_Nekomamushi/flag.txt -----
Te cansaste de buscar y te has quedado dormido
----- /home/usopp/ONEPIECE/Zou/Left_Fore_Leg/Casa_de_Nekomamushi/flag.txt -----
Has encontrado un lugar peligroso
----- /home/usopp/ONEPIECE/Zou/Right_Hind_Leg/Casa_de_Nekomamushi/flag.txt -----
Has encontrado un lugar abandonado
----- /home/usopp/ONEPIECE/Whole_Cake_Island/Caramel_Mountain/Casa_de_Big_Mom/flag.txt -----
Te cansaste de buscar y te has quedado dormido
----- /home/usopp/ONEPIECE/Pirate_Island/Pirates_Ship/Casa_de_Whitebeard/flag.txt -----
Has encontrado un enemigo y ha tenido que luchar
----- /home/usopp/ONEPIECE/Pirate_Island/Pirates_Den/Casa_de_Gol_Roger/flag.txt -----
Has encontrado un amigo y han decidido viajar juntos
----- /home/usopp/ONEPIECE/Pirate_Island/Pirates_Den/Casa_de_Whitebeard/flag.txt -----
a77742694e1d538c1c6d3c30d4c4df294c6c02379a1b138a411f26ec12fbc58cee1b028c39
----- /home/usopp/ONEPIECE/Pirate_Island/Pirates_Hideout/Casa_de_Shanks/flag.txt -----
Has encontrado un amigo y han decidido viajar juntos
----- /home/usopp/ONEPIECE/Alabasta/Katorea/Casa_de_Kohza/flag.txt -----
Has encontrado un lugar misterioso
usopp@04660b78e9f2:~$
```

3. Análisis y descifrado del sistema de cifrado personalizado

- r. El repositorio del proyecto incluía un archivo de utilidades con funciones para generar el flujo de claves (keystream) y cifrar/descifrar mensajes utilizando un generador de claves muy débil basado en random.seed.



```

● ● ●

1 def decrypt(ciphertext):
2     cipherbytes = bytes.fromhex(ciphertext)
3     for seed in range(100000): # Intentar semillas del 0 al 99999
4         keystream = generate_keystream(seed, len(ciphertext))
5         plaintext = bytes([c ^ k for c, k in zip(cipherbytes, keystream)])
6         if plaintext.startswith(b"FLAG_"):
7             print(f"Semilla encontrada: {seed}")
8             print(f"Texto descifrado: {plaintext.decode()}")
9             break
10
11 def usopp_cipher(flag, secret_seed):
12     ciphertext = encrypt(flag, secret_seed)
13     return ciphertext
14
15
16
17 if __name__ == "__main__":
18     decrypt('a77742694e1d538c1c6d3c30d4c4df294c6c02379a1b138a411f26ec12fbc58cee1b028c39')
19
20
21
```

- s. Esto nos dio la flag

FLAG_7a9ee38e32eb180cd2db21f2a39fddd0

4. Búsqueda de archivos ZIP o imágenes (poneglyphs)

- t. Tal como en el reto anterior, se utilizó find para localizar archivos de imagen y .zip que pudieran contener los poneglyphs:

```
● ● ●
1 sudo find /home/usopp/ONEPIECE/
2 -type f \(-iname "*.png"-o -iname "*.jpg"-o -iname "*.gif"-o -iname "*.bmp"-o -iname "*.webp"-o -iname "*.zip"\) 2>/dev/null
3
```

u. Esto arrojó como resultado:

```
Has encontrado un lugar misterioso
usopp@04660b78e9f2:~$ sudo find /home/usopp/ONEPIECE/ -type f \(-iname "*.png"-o -iname "*.jpg"-o -iname "*.gif"-o -iname "*.bmp"-o -iname "*.webp"-o -iname "*.zip"\) 2>/dev/null
/home/usopp/ONEPIECE/Skypiea/Angel_Beach/Casa_de_Conis/poneglyph.zip
usopp@04660b78e9f2:~$
```

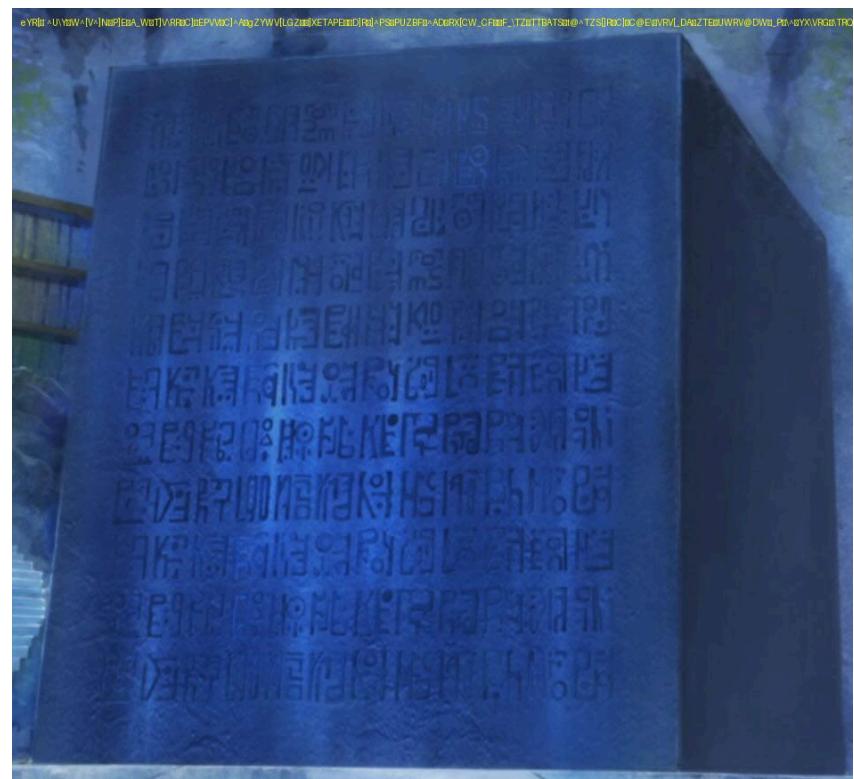
5. Extracción del ZIP dentro del contenedor

v. El archivo .zip fue extraído con:

```
● ● ●
1 unzip /home/usopp/ONEPIECE/Skypiea
2 /Angel_Beach/Casa_de_Conis/poneglyph.zip -d /home/usopp/extracted_poneglyph
3
```

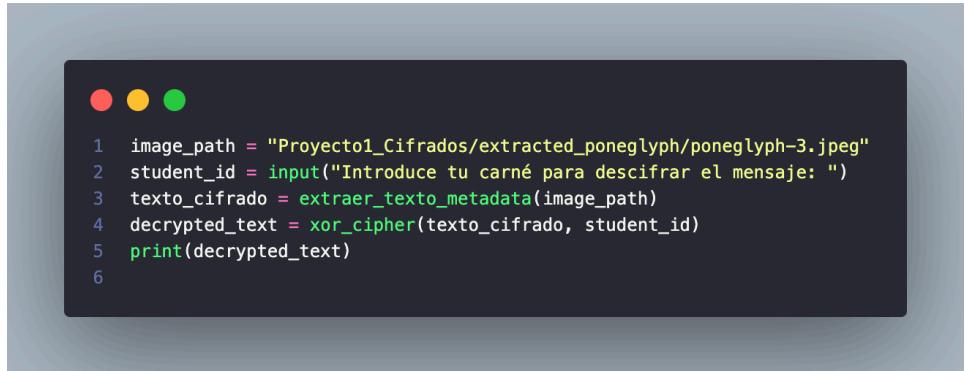
w. Este comando:

- i. Esto generó un archivo de imagen poneglyph-3.jpeg, el cual contenía el mensaje oculto en los metadatos EXIF.
- x. Luego del archivero de docker se guardó hacia mi máquina local.



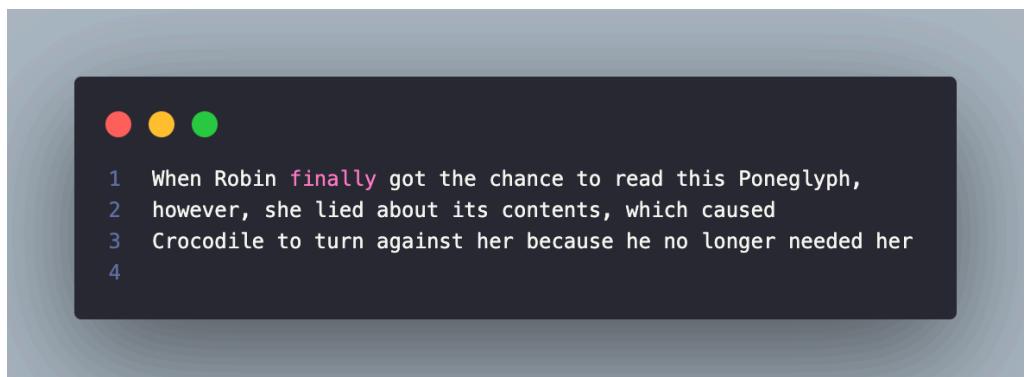
6. Extracción del texto cifrado en los metadatos de la imagen

- y. Se reutilizó el mismo script de los retos anteriores para obtener el contenido del metadato Artist en la imagen .jpeg. Luego, se aplicó un XOR con el carné "21757" para descifrar el mensaje.



```
● ● ●
1 image_path = "Proyecto1_Cifrados/extracted_poneglyph/poneglyph-3.jpeg"
2 student_id = input("Introduce tu carné para descifrar el mensaje: ")
3 texto_cifrado = extraer_texto_metadata(image_path)
4 decrypted_text = xor_cipher(texto_cifrado, student_id)
5 print(decrypted_text)
6
```

- z. El resultado fue:



```
● ● ●
1 When Robin finally got the chance to read this Poneglyph,
2 however, she lied about its contents, which caused
3 Crocodile to turn against her because he no longer needed her
4
```

En resumen:

- Flag encontrada:
 - FLAG_7a9ee38e32eb180cd2db21f2a39fddd0
- Texto del poneglyph
 - When Robin finally got the chance to read this Poneglyph, however, she lied about its contents, which caused Crocodile to turn against her because he no longer needed her

Reto 4: Nami - Chacha20

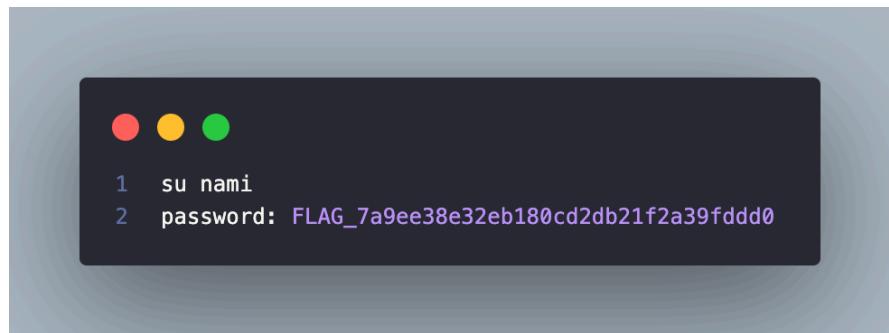
Este reto consistió en descifrar un mensaje cifrado con el algoritmo de cifrado de flujo moderno ChaCha20, utilizando una clave y un nonce derivado del carné del estudiante. Además, se evaluó la importancia de no reutilizar un nonce con la misma clave, ya que hacerlo puede comprometer gravemente la seguridad del sistema criptográfico.

Como en los desafíos anteriores, el reto también incluía la extracción de un fragmento de texto oculto en una imagen (poneglyph), cuya interpretación estaba cifrada mediante una operación XOR simple.

Proceso

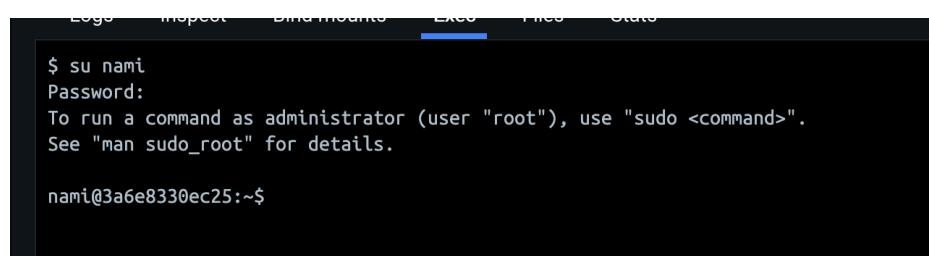
- Como primer paso, se inicio sesión en el challenge en cuestión, y como fue costumbre en previos ejercicios de CTF, como password se utilizó la flag encontrada del reto pasado:

Comandos ejecutados:



```
1 su nami
2 password: FLAG_7a9ee38e32eb180cd2db21f2a39fddd0
```

Consola:



```
$ su nami
Password:
To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.

nami@3a6e8330ec25:~$
```

- Búsqueda de archivos flag.txt y visualización en el sistema:

- Para visualizar el contenido de todos los archivos encontrados, junto con su ruta respectiva, se ejecutó el mismo script de los TRES challenge pasados:



```
1 sudo find /home/nami/ONEPIECE/ -type f -name flag.txt 2>/dev/null | while read filepath; do
2 echo "----- $filepath -----"
3 cat "$filepath"
4 echo
5 done
6
```

bb. Este script:

- i. se identificó un archivo importante:
/home/nami/ONEPIECE/Zou/Right_Belly_Fortress/Casa_de_Kawamatsu(flag.txt)
- ii. Como resultado, se encontró una secuencia hexadecimal:

```
password incorrect--reenter:  
  inflating: /home/nami/extracted_poneglyph/challenges/nami/poneglyph.jpeg  
nami@3a6e8330ec25:~$ sudo find /home/nami/ONEPIECE/ -type f -name flag.txt 2>/dev/null | while read filepath;  
  echo "----- $filepath -----"  
  cat "$filepath"  
  echo  
done  
----- /home/nami/ONEPIECE/Skypiea/Upper_Yard/Casa_de_Aisa/flag.txt -----  
Has encontrado un objeto misterioso y no sabes qué es  
----- /home/nami/ONEPIECE/Skypiea/Upper_Yard/Casa_de_Enel/flag.txt -----  
Has encontrado un lugar lleno de secretos y misterios que apuntan a Right_Belly_Fortress  
----- /home/nami/ONEPIECE/Skypiea/Shandora/Casa_de_Montblanc_Norland/flag.txt -----  
Has encontrado un enemigo y ha tenido que huir  
----- /home/nami/ONEPIECE/Zou/Right_Belly_Fortress/Casa_de_Nekomamushi/flag.txt -----  
Has encontrado una pista que apunta a Right_Belly_Fortress  
----- /home/nami/ONEPIECE/Zou/Right_Belly_Fortress/Casa_de_Kawamatsu/flag.txt -----  
08f08738905e1b0a058052a7499a86a36945d63c73c2994195f214a335fefc160b1266bc27  
----- /home/nami/ONEPIECE/Whole_Cake_Island/Sweet_City/Casa_de_Big_Mon/flag.txt -----  
Has encontrado una pista que apunta a Right_Belly_Fortress  
----- /home/nami/ONEPIECE/Whole_Cake_Island/Casa_de_Rio_Mon/flag.txt -----
```

3. Análisis y descifrado del sistema de cifrado personalizado

cc. El repositorio proporcionaba una función chacha20_decrypt para realizar el descifrado, la cual utilizaba la librería pycryptodome. La clave y el nonce eran derivados automáticamente desde el carné.



```
1  def chacha20_decrypt(ciphertext, user_id):  
2      key, nonce = generate_key_nonce(user_id=user_id)  
3      cipher = ChaCha20.new(key=key, nonce=nonce)  
4      plaintext = cipher.decrypt(ciphertext)  
5      return plaintext.decode()  
6  
7  
8  def nami_cipher(plaintext, user_id):  
9      ciphertext = chacha20_encrypt(plaintext, user_id)  
10     return ciphertext  
11  
12  
13  
14 if __name__ == "__main__":  
15     print(chacha20_decrypt( bytes.fromhex('08f08738905e1b0a058052a7499a86a36945d63c73c2994195f214a335fefc160b1266bc27') ), '21757')
```

dd. Esto nos dio la flag

FLAG_0d4e96b8ab746237b7300fcfee747755

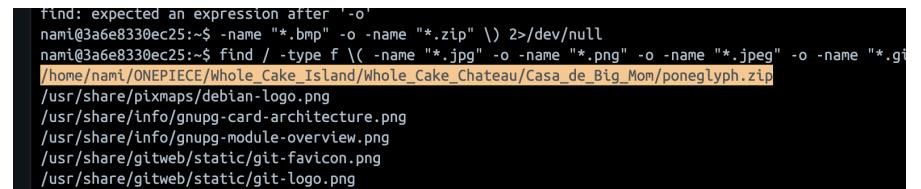
4. Búsqueda de archivos ZIP o imágenes (poneglyphs)

ee. Tal como en el reto anterior, se utilizó find para localizar archivos de imagen y .zip que pudieran contener los poneglyphs:



```
1 sudo find /home/nami/ONEPIECE/
2 -type f \(-name "*.png" -o -name "*.jpg" -o -name "*.gif" -o -name "*.bmp" -o -name "*.webp" -o -name "*.zip"\) 2>/dev/null
3
4
```

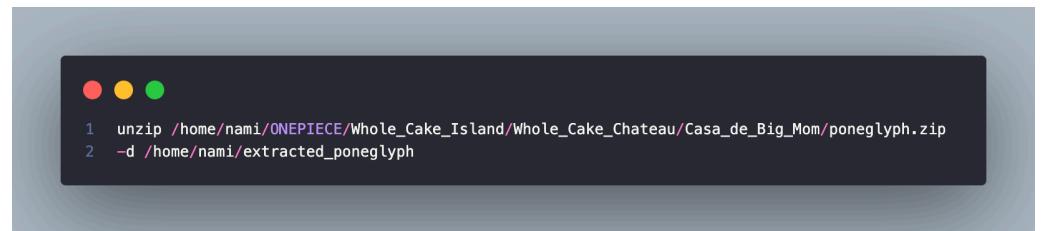
ff. Esto arrojó como resultado:



```
find: expected an expression after '-o'
nami@3a6e8330ec25:~$ -name "*.bmp" -o -name "*.zip" \) 2>/dev/null
nami@3a6e8330ec25:~$ find / -type f \(-name "*.jpg" -o -name "*.png" -o -name "*.jpeg" -o -name "*.gi
/home/nami/ONEPIECE/Whole_Cake_Island/Whole_Cake_Chateau/Casa_de_Big_Mom/poneglyph.zip
/usr/share/pixmaps/debian-logo.png
/usr/share/info/gnupg-card-architecture.png
/usr/share/info/gnupg-module-overview.png
/usr/share/gitweb/static/git-favicon.png
/usr/share/gitweb/static/git-logo.png
```

5. Extracción del ZIP dentro del contenedor

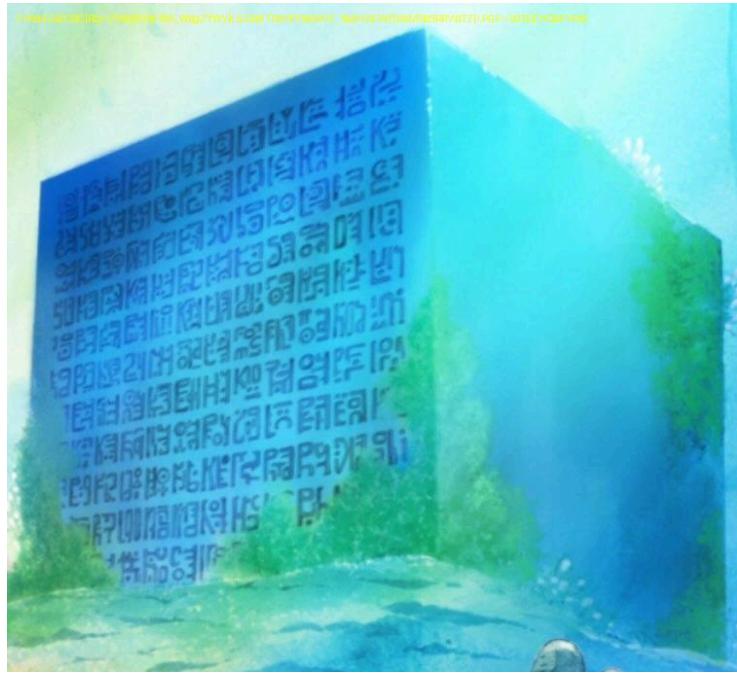
gg. El archivo .zip fue extraído con:



```
1 unzip /home/nami/ONEPIECE/Whole_Cake_Island/Whole_Cake_Chateau/Casa_de_Big_Mom/poneglyph.zip
2 -d /home/nami/extracted_poneglyph
```

hh. Este comando:

- i. Al solicitar la contraseña del ZIP, se utilizó la flag encontrada anteriormente (FLAG_0d4e96b8ab746237b7300fcfee747755), revelando el archivo:
- ii. Esto generó un archivo de imagen poneglyph-4.jpeg, el cual contenía el mensaje oculto en los metadatos EXIF.
- iii. Luego del archivero de docker se guardó hacia mi máquina local.



6. Extracción del texto cifrado en los metadatos de la imagen

jj. Se reutilizó el mismo script de los retos anteriores para obtener el contenido del metadato Artist en la imagen .jpeg. Luego, se aplicó un XOR con el carné "21757" para descifrar el mensaje.

```
● ● ●
1 image_path = "/Users/brand/Documents/UVG/Proyecto1_Cifrados/extracted_poneglyph/poneglyph-4.jpeg"
2 student_id = input("Introduce tu carné para descifrar el mensaje: ")
3 texto_cifrado = extraer_texto_metadata(image_path)
4 decrypted_text = xor_cipher(texto_cifrado, student_id)
5 print(decrypted_text)
6
```

kk. El resultado fue:

```
● ● ●
1 the Tomb of the Kings where the Poneglyph was held became unstable
2 and began collapsing around them.
3
```

En resumen:

- Flag encontrada:
 - FLAG_0d4e96b8ab746237b7300fcfee747755
- Texto del poneglyph
 - the Tomb of the Kings where the Poneglyph was held became unstable and began collapsing around them.

Reflexión final

Este proyecto representó una valiosa experiencia de aprendizaje en el campo de la criptografía práctica. A través de los retos propuestos, se pudo evidenciar la fragilidad de muchos sistemas criptográficos mal implementados y la importancia de seguir buenas prácticas en el diseño de sistemas de seguridad.

Entre los aprendizajes más importantes se podría destacar:

- Comprender cómo una operación tan simple como el XOR, si se utiliza de forma incorrecta o con claves repetidas, puede comprometer la seguridad de un sistema.
- Romper el cifrado RC4 permitió visualizar las consecuencias del mal diseño en el algoritmo y su sesgo en la generación del flujo de claves.
- El reto con el PRNG personalizado reveló cómo el uso de generadores de números pseudoaleatorios inseguros y claves débiles permite romper sistemas cifrados en cuestión de segundos con técnicas de fuerza bruta.
- Finalmente, trabajar con ChaCha20, un estándar moderno, resaltó la relevancia del uso correcto del nonce y la clave, y cómo una mínima mala práctica puede llevar a vulnerabilidades críticas.

Además, el uso de metadatos EXIF en imágenes para ocultar texto fue una lección valiosa e interesante sobre esteganografía digital y su papel en escenarios reales de ocultamiento de datos.

Referencias

Shotts, W. (2019). *The Linux command line: a complete introduction*. No Starch Press.