

















































































































































































## Contenido












	Base Encodings: Base64, Base32 y Base85 .....	10
	¿Qué son?.....	10
	Base64 .....	10
	Descripción.....	10
	Ejemplo.....	10
	Uso común en CTF.....	10
	Cómo identificarlo .....	11
	Herramientas para decodificar .....	11
	Base32 .....	11
	Descripción.....	11
	Ejemplo.....	11
	Uso común en CTF.....	11
	Cómo identificarlo .....	11
	Herramientas para decodificar .....	12
	Base85 (a.k.a. ASCII85) .....	12
	Descripción.....	12
	Ejemplo.....	12
	Uso común en CTF.....	12
	Cómo identificarlo .....	12
	Herramientas para decodificar .....	13
	Comparativa rápida.....	13
	ROT Ciphers: ROT13, ROT-n y ROT47.....	14
	¿Qué son?.....	14
	ROT13 .....	14
	Descripción.....	14
	Ejemplo.....	14
	Propiedad importante .....	14
	Uso en CTF.....	14
	Cómo identificarlo .....	15
	Herramientas.....	15
	ROT-n (César con rotación arbitraria).....	15
	Descripción.....	15

	Ejemplo.....	15
	Uso en CTF.....	15
	Cómo identificarlo .....	15
	Herramientas.....	16
	ROT47 .....	16
	Descripción.....	16
	Ejemplo.....	16
	Propiedad importante .....	16
	Uso en CTF.....	16
	Cómo identificarlo .....	16
	Herramientas.....	17
	Comparativa rápida.....	17
	 Vigenère Cipher .....	18
	¿Qué es?.....	18
	¿Cómo funciona?.....	18
	Descifrado.....	19
	Uso en CTF.....	19
	Cómo identificarlo .....	19
	Herramientas útiles .....	19
	Recomendaciones para CTF .....	20
	Resumen rápido.....	20
	 Substitution Cipher (Cifrado por sustitución) .....	21
	¿Qué es?.....	21
	Tipos principales.....	21
	Ejemplo.....	21
	¿Cómo se usa en CTF? .....	22
	¿Cómo identificarlo?.....	22
	¿Cómo resolverlo?.....	22
	Herramientas útiles .....	22
	Script simple de análisis en Python .....	23
	Resumen rápido.....	23
	 Transposition Cipher (Cifrado por transposición) .....	24
	¿Qué es?.....	24

	Tipos comunes de transposición .....	24
	Columnar Transposition Cipher.....	25
	¿Cómo funciona?.....	25
	¿Cómo se usa en CTF? .....	25
	Herramientas útiles .....	26
	Python: Descifrado (Columnar Transposition) .....	26
	Resumen rápido.....	27
	Symbol-Based Encodings: Morse Code y Tap Code .....	28
	1  Morse Code.....	28
	¿Qué es?.....	28
	Ejemplo básico:.....	28
	Decodificación:.....	28
	Identificación en CTF: .....	29
	Herramientas útiles: .....	29
	2  Tap Code (código de golpes) .....	29
	¿Qué es?.....	29
	Estructura.....	29
	Ejemplo:.....	30
	También puede verse como: .....	30
	Identificación en CTF: .....	30
	Herramientas útiles: .....	30
	Comparación rápida.....	30
	Biological-Looking Encoding: DNA Encoding .....	32
	¿Qué es?.....	32
	¿Cómo funciona?.....	32
	Ejemplo.....	32
	Decodificación inversa .....	33
	Variante avanzada.....	33
	¿Cómo identificarlo en un reto? .....	33
	Herramientas útiles .....	34
	Ejemplo en Python.....	34
	Resumen rápido.....	34
	XOR Cipher.....	35




	¿Qué es? .....	35
	¿Cómo funciona el cifrado XOR? .....	35
	Fórmulas .....	35
	Características clave .....	36
	Uso en CTFs .....	36
	Cómo atacar o resolver cifrados XOR .....	36
	Ejemplo sencillo en Python .....	37
	Resumen rápido .....	37
	<b>Affine Cipher (Cifrado Afín) .....</b>	<b>38</b>
	¿Qué es? .....	38
	¿Cómo funciona? .....	38
	Condiciones clave .....	38
	Uso en CTF .....	39
	Cómo romperlo .....	39
	Ejemplo básico en Python .....	39
	Resumen rápido .....	40
	<b>Enigma Cipher .....</b>	<b>41</b>
	¿Qué es? .....	41
	¿Cómo funciona? .....	41
	Detalles técnicos .....	41
	Características clave .....	42
	Uso en CTF .....	42
	Cómo atacar un cifrado Enigma en CTF .....	42
	Herramienta útil .....	42
	Ejemplo básico con Py-enigma (simplificado) .....	43
	Resumen rápido .....	43
	<b>Cifrado César (Caesar Cipher) .....</b>	<b>44</b>
	¿Qué es? .....	44
	¿Cómo funciona? .....	44
	Fórmula .....	44
	Características clave .....	44
	Uso en CTF .....	45
	Cómo romperlo .....	45

	Ejemplo en Python .....	45
	Resumen rápido .....	46
	Cifrado Atbash.....	47
	¿Qué es? .....	47
	¿Cómo funciona? .....	47
	Fórmula .....	47
	Características clave .....	48
	Uso en CTF .....	48
	Ejemplo en Python .....	48
	Resumen rápido .....	49
	 Keyboard Shift Cipher.....	50
	¿Qué es?.....	50
	¿Cómo funciona?.....	50
	¿Dónde se usa?.....	51
	Identificación en CTF .....	51
	Estrategia para resolverlo .....	51
	Ejemplo completo.....	51
	Resumen rápido.....	51
	 Polybius Square Cipher.....	53
	¿Qué es?.....	53
	¿Cómo funciona?.....	53
	Ejemplo: Cifrar la palabra HELLO .....	53
	Para descifrar.....	54
	¿Cómo aparece en CTFs? .....	54
	Variantes.....	54
	Ejemplo con clave.....	54
	Herramientas útiles .....	55
	Python - Descifrado básico.....	55
	Resumen rápido.....	55
	 Rail Fence Cipher.....	56
	¿Qué es?.....	56
	¿Cómo funciona?.....	56
	Ejemplo: mensaje WEAREDISCOVEREDFLEEATONCE .....	56

	Descriptación.....	56
	¿Dónde se usa en CTF? .....	57
	Herramientas útiles .....	57
	Python - Descifrado (3 rails).....	57
	Resumen rápido.....	58
	Brainfuck.....	59
	¿Qué es?.....	59
	Características clave .....	59
	Los comandos de Brainfuck .....	59
	¿Cómo funciona?.....	59
	Uso en CTFs.....	60
	Cómo resolver retos con Brainfuck .....	60
	Ejemplo simple de Brainfuck.....	60
	Resumen rápido.....	60
	JSFuck .....	61
	¿Qué es?.....	61
	¿Cómo funciona?.....	61
	Características clave .....	61
	Uso en CTF y seguridad .....	62
	Cómo resolver retos JSFuck .....	62
	Ejemplo mínimo.....	62
	Resumen rápido.....	62
	Esoteric Languages: Malbolge.....	63
	¿Qué es?.....	63
	Características clave .....	63
	Comandos básicos.....	63
	¿Cómo funciona?.....	63
	Uso en CTFs.....	63
	Cómo resolver retos Malbolge .....	63
	Ejemplo clásico: “Hello, World!”.....	64
	Resumen rápido.....	64
	Whitespace.....	65
	¿Qué es?.....	65

🔍	¿Cómo funciona?	65
✂️	Características clave	65
🔍	Uso en CTFs	65
🔧	Herramientas para Whitespace	66
🐍	Ejemplo (representación visual)	66
✅	Resumen rápido	66
⚡🐰	PikaLang	67
🧠	¿Qué es?	67
🔍	¿Cómo funciona?	67
✂️	Características clave	67
🔍	Uso en CTF	68
🔧	Cómo resolver retos PikaLang	68
🐍	Ejemplo básico (pseudo)	68
✅	Resumen rápido	68
🔒	RSA Attacks and Variants	69
1	Classic RSA	69
	Descripción	69
2	Multi-Prime RSA	69
	Descripción	69
	Ventajas	69
	Riesgos	69
3	Cube Root Attack (Low Exponent Attack)	69
	Descripción	69
	Cómo funciona	70
	Prevención	70
4	Wiener's Attack	70
	Descripción	70
	Idea clave	70
	Cómo funciona	70
	Importancia	70
5	Common Modulus Attack	71
	Descripción	71
	Cómo funciona	71
6	Chinese Remainder Theorem Attack (CRT Attack)	71



Descripción.....	71
Cómo funciona.....	71
Prevención.....	71
 Twin Prime RSA .....	72
Descripción.....	72
Ventajas.....	72
Riesgos.....	72
 Resumen rápido.....	72
 Herramientas y recursos útiles .....	73
Exploiting / Pwn.....	74
Forense.....	74
Misc.....	76
Reversing.....	77
Steganography.....	78
Web.....	80
OSINT.....	81
Privilege Escalation.....	81
Hash Cracking.....	82
Utilidades de red.....	82

## Base Encodings: Base64, Base32 y Base85

### ¿Qué son?

Los **Base Encodings** son métodos de **codificación de datos binarios en texto legible**, usando alfabetos específicos (letras, números y a veces símbolos) para representar bytes. A diferencia de los cifrados, no son **métodos criptográficos**, sino formas de **representar información codificada** que luego se puede decodificar de forma directa.

Son muy comunes en CTFs como medio de **ocultar datos**, archivos, flags o textos.

## Base64

### Descripción

- Codifica datos binarios en texto usando 64 caracteres:
  - A-Z (26)
  - a-z (26)
  - 0-9 (10)
  - +, / (2 símbolos adicionales)
- El resultado termina con = o == para rellenar bloques.

### Ejemplo

- Texto: CTF
- Base64: Q1RGCg==

### Uso común en CTF

- Archivos codificados
- Flags ocultas en formularios, parámetros URL, cookies, JWT, etc.
- Partes de imágenes o binarios

## Cómo identificarlo

- Longitud múltiplo de 4
- Termina con = o == (no siempre obligatorio)
- Solo usa caracteres alfanuméricos, más + y /

## Herramientas para decodificar

- base64 en terminal:
- `echo "Q1RGCg==" | base64 -d`
- [CyberChef](#)
- [dcode.fr/base64](https://dcode.fr/base64)

## Base32

### Descripción

- Usa un alfabeto de **32 caracteres**:
  - A-Z (26)
  - 2-7 (6 dígitos)
- Es **menos compacto** que Base64, pero más robusto en sistemas sensibles a mayúsculas/minúsculas.

### Ejemplo

- Texto: CTF
- Base32: INXW4===

## Uso común en CTF

- Autenticación TOTP/2FA
- Datos en QR o codificados en texto plano
- Protocolos como DNS (resistencia a mayúsculas/minúsculas)

## Cómo identificarlo

- Solo letras mayúsculas y números del 2 al 7
- Longitud múltiplo de 8 (relleno con =)
- Más largo que su equivalente en Base64

## Herramientas para decodificar

- base32 en terminal:
- `echo "INXW4===" | base32 -d`
- [CyberChef](#)
- [dcode.fr/base-32](https://dcode.fr/base-32)

## Base85 (a.k.a. ASCII85)

### Descripción

- Codifica datos usando **85 caracteres imprimibles** (de ASCII 33 a 117).
- Mucho más compacto que Base64 (~25% más eficiente).
- Existen varias variantes:
  - **ASCII85**: Adobe/PostScript
  - **Z85**: ZeroMQ
  - **Base85**: estándar RFC 1924

### Ejemplo

- Texto: CTF
- Base85: GbKC|

### Uso común en CTF

- Archivos binarios (como PDF, PostScript)
- Comunicación binaria sobre texto plano
- Retos que requieren identificar variantes de codificación no evidentes

### Cómo identificarlo

- Caracteres ASCII imprimibles (letras, símbolos, números)
- A menudo **no tiene relleno con =**
- Puede tener delimitadores como `<~` y `~>` (en ASCII85)

## Herramientas para decodificar

- python CON `base64.a85decode()`:
- `import base64`
- `base64.a85decode("GbKC|")`
- CyberChef (tiene todas las variantes)
- [dcode.fr/ascii85](https://dcode.fr/ascii85)

## Comparativa rápida

Característica	Base64	Base32	Base85
Tamaño al codificar	Mayor	Muy alto	Más compacto
Caracteres usados	A-Z, a-z, 0-9, +/	A-Z, 2-7	ASCII 33-117
Relleno (=)	Sí	Sí	No (normalmente)
Común en CTF	Muy común	Moderado	Menos común
Soporte nativo	Muy alto (bash, python)	Medio	Bajo (solo en python/CyberChef)



## ROT Ciphers: ROT13, ROT-n y ROT47

### ¿Qué son?

Los **ROT Ciphers** (del inglés "ROTate") son un tipo de **cifrado César**, en el que se **rota** (desplaza) cada carácter un número fijo de posiciones en el alfabeto o en la tabla ASCII. Son muy simples, pero efectivos para ocultar mensajes a simple vista.

Son **simétricos**: aplicar el mismo cifrado dos veces recupera el original (por ejemplo, ROT13 sobre ROT13 te da el texto original).

## ROT13

### Descripción

- Es una **rotación de 13 posiciones** sobre las letras del alfabeto latino.
- Solo afecta letras A-Z y a-z (mantiene mayúsculas/minúsculas).
- No modifica números ni signos de puntuación.

### Ejemplo

- Original: CTFchallenge
- ROT13: PGSpunyyratr

### Propiedad importante

- Autoinversa:  $\text{ROT13}(\text{ROT13}(\text{texto})) = \text{texto original}$

### Uso en CTF

- Mensajes o flags parcialmente ofuscados
- Descripciones en HTML o páginas web con código ofuscado
- Correos electrónicos (es un truco usado en listas de correo antiguas para ocultar spoilers)

## Cómo identificarlo

- Texto legible, pero ligeramente "distorsionado"
- Solo letras modificadas
- A menudo usado en nombres como "onfr64" → ROT13("onfr") = "base"

## Herramientas

- Terminal:
- `echo "PGSpunyyratr" | tr 'A-Za-z' 'N-ZA-Mn-za-m'`
- CyberChef: "ROT13"
- [dcode.fr](https://dcode.fr/ROT13-Decoder): ROT13 Decoder

## ROT-n (César con rotación arbitraria)

### Descripción

- Variante genérica del cifrado César: rota cada letra **n posiciones** en el alfabeto.
- Ejemplo: ROT1 (A → B), ROT5 (A → F), hasta ROT25.
- No afecta números ni símbolos.

### Ejemplo

- Texto: CTF
- ROT5: HYK
- ROT20: WZN

### Uso en CTF

- Mensajes cifrados con rotaciones no evidentes
- Usado a veces en combinación con Base64
- Flags escondidas que se revelan solo con el ROT correcto

## Cómo identificarlo

- Texto no parece ROT13 pero mantiene estructura alfabética
- Probar todas las rotaciones (fuerza bruta)

## Herramientas

- CyberChef: operación "ROT-n" → "Brute-force Caesar cipher"
- dcode.fr: "César Cipher (ROT-n)"
- Script en Python para fuerza bruta:

```
for i in range(1, 26):  
    print(f"ROT{i}: {''.join(chr((ord(c) - 65 + i) % 26 +  
65) if c.isupper() else c for c in 'CTF')}")
```

## ROT47

### Descripción

- Variante que rota **47 posiciones en la tabla ASCII**.
- Afecta todos los **caracteres imprimibles del 33 ('!') al 126 (~)**.
- Incluye letras, números y símbolos (más amplio que ROT13).

### Ejemplo

- Original: CTF{rot47\_flag}
- ROT47: pt6E?@C6C2=@?xF

### Propiedad importante

- Autoinversa: aplicar ROT47 dos veces = texto original

### Uso en CTF

- Ofuscación en scripts, nombres de archivos, o binarios
- Retos donde ROT13 no revela nada pero ROT47 sí

### Cómo identificarlo

- Texto parece aleatorio pero tiene símbolos ASCII legibles
- No es Base64 ni Base85, y no parece ROT13

## Herramientas

- CyberChef: "ROT47"
- Python:

```
def rot47(text):  
    return ''.join([chr(33 + ((ord(c) - 33 + 47) % 94)) if  
33 <= ord(c) <= 126 else c for c in text])  
  
print(rot47("pt6E?@C6C2=@?xF"))
```

## Comparativa rápida

Cifrado	Afecta	Rango	Simétrico	Común en CTF
ROT13	Solo letras A-Z, a-z	Alfabeto latino	Sí	Muy común
ROT-n	Solo letras A-Z, a-z	Alfabeto latino	No (excepto ROT13)	Común
ROT47	Todo ASCII imprimible	ASCII 33-126	Sí	Media frecuencia

## Vigenère Cipher

### ¿Qué es?

El **Vigenère Cipher** es un cifrado de sustitución **polialfabético**, que utiliza una **palabra clave** (clave secreta) para determinar el desplazamiento de cada letra del texto original (plaintext).

Fue considerado durante mucho tiempo "irrompible" antes del desarrollo de métodos de criptoanálisis más avanzados. Es más seguro que un cifrado César porque **no usa un único desplazamiento**, sino muchos, dependiendo de la clave.

### ¿Cómo funciona?

Cada letra del texto claro se **desplaza según la letra correspondiente de la clave**, utilizando una tabla tipo César.

### Fórmula:

Si tienes:

- Texto: HELLO
- Clave: KEY (repetida: KEYKE)

Aplicamos la siguiente fórmula por letra:


$$C = (P + K) \bmod 26,$$

donde:

- C es el carácter cifrado
- P es la letra del texto plano (convertida a número: A=0...Z=25)
- K es la letra de la clave (también convertida a número)

### Ejemplo:

Letra (P)	H	E	L	L	O
Clave (K)	K	E	Y	K	E
P (num)	7	4	11	11	14
K (num)	10	4	24	10	4
C (num)	17	8	9	21	18
C (letra)	R	I	J	V	S

 Cifrado final: **RIJVS**



## Descifrado

Se hace lo mismo, pero restando la clave:

$$P = (C - K + 26) \bmod 26$$

## Uso en CTF

- Muy frecuente en retos de nivel básico/intermedio
- Puede aparecer como parte de mensajes ocultos en imágenes, scripts o PDFs
- A veces se usa combinado con Base64, ROT o XOR

## Cómo identificarlo

- Texto cifrado que **parece alfabético aleatorio** sin símbolos ni números
- No responde a ROT13 o César simple
- Si se sospecha de Vigenère, hay que deducir la clave (conocida, parcial o adivinada)
- Herramientas de análisis pueden ayudar a **detectar la longitud de la clave** y romper el cifrado

## Herramientas útiles

- [dcode.fr/vigenere-cipher](https://dcode.fr/vigenere-cipher) - Muy completa
- [CyberChef](https://cyberchef.org/) - Tiene cifrado y descifrado
- Terminal con scripts en Python o bash

## Ejemplo en Python:

```
def vigenere_decrypt(ciphertext, key):
    result = ''
    key = key.upper()
    for i, char in enumerate(ciphertext.upper()):
        if char.isalpha():
            shift = ord(key[i % len(key)]) - ord('A')
            decrypted = (ord(char) - ord('A') - shift + 26) % 26
            result += chr(decrypted + ord('A'))
        else:
            result += char
    return result

print(vigenere_decrypt("RIJVS", "KEY")) # Output: HELLO
```



## Recomendaciones para CTF

- Prueba claves obvias: KEY, CTF, FLAG, SECRET, PASSWORD, etc.
- Busca pistas ocultas en los títulos, nombres de archivos o metadatos
- Si la clave no se conoce, intenta con **criptoanálisis**:
  - Método de Kasiski
  - Índice de coincidencia (IC)
  - Ataques de frecuencia por bloques



## Resumen rápido

Característica	Valor
Tipo	Sustitución polialfabética
Necesita clave	Sí (muy importante)
Alfabeto usado	Solo letras (A-Z)
Simétrico	Sí
Dificultad en CTF	Media
Muy común en CTF	Sí
Herramientas online	CyberChef, dCode, scripts Python

## Substitution Cipher (Cifrado por sustitución)

### ¿Qué es?

El **Substitution Cipher** (Cifrado por Sustitución) es uno de los cifrados más antiguos y simples. Consiste en **reemplazar cada letra del mensaje original por otra letra o símbolo**, según un alfabeto o regla específica.

A diferencia de los cifrados como César o Vigenère, donde la sustitución tiene un patrón matemático claro, en un **substitution cipher general**, cada letra puede ser reemplazada **por cualquier otro símbolo o letra**, sin necesidad de seguir un orden alfabético.

### Tipos principales

- **Monoalfabético**: cada letra siempre se reemplaza por la misma letra/símbolo.
- **Polialfabético**: una misma letra puede ser cifrada de diferentes formas (como en Vigenère).

Aquí nos enfocamos en el **monoalfabético**, que es más común en CTFs como reto básico de criptoanálisis.

### Ejemplo

Supongamos este alfabeto de sustitución personalizado:

Claro **A B C D E F G H I J K L M**  
Cifrado **Q W E R T Y U I O P A S D**

Texto original: FLAG  
Texto cifrado: YSQD

Cada letra del texto claro ha sido sustituida por otra según la tabla personalizada.

## ✖ ¿Cómo se usa en CTF?

- Frecuente como texto cifrado en retos de nivel básico o medio.
- Aparece en retos de **criptoanálisis manual**, donde debes deducir el alfabeto cifrado.
- A veces se combina con **frecuencia de letras** para ser resuelto.

## 🕵️ ¿Cómo identificarlo?

- Texto cifrado que mantiene solo letras (sin símbolos ni base64).
- No responde a ROT13, Vigenère ni César.
- La distribución de letras parece **no aleatoria** (ej.: muchas repeticiones de una misma letra).
- Puedes notar patrones como palabras cortas repetidas (podría ser "THE", "AND", "CTF", etc.)

## 🔧 ¿Cómo resolverlo?

1. **Análisis de frecuencia:**
  - En inglés, las letras más comunes son E, T, A, O, I, N, S, H, R.
  - Compara las letras más comunes del texto cifrado con estas.
2. **Reconocimiento de patrones comunes:**
  - Palabras como THE, AND, IS, IN, etc.
  - En español: DE, LA, EL, QUE, ES, etc.
3. **Educación visual:**
  - Fíjate en repeticiones o estructuras gramaticales comunes.
4. **Sustitución progresiva:**
  - Empieza reemplazando letras con mayor probabilidad y prueba distintas combinaciones.

## 🔧 Herramientas útiles

- <https://quipqiup.com> – Descifrador automático de sustitución.
- <https://www.dcode.fr/monoalphabetic-substitution>
- CyberChef: "Substitute" / "Frequency analysis"

- Scripts en Python para analizar frecuencias de letras.

## Script simple de análisis en Python

```
from collections import Counter

ciphertext = "YSQD QTRT QSD YS" # texto cifrado
filtered = ''.join(filter(str.isalpha, ciphertext.upper()))
frequencies = Counter(filtered)

for letter, count in frequencies.most_common():
    print(f"{letter}: {count}")
```

### ✓ Resumen rápido

Característica	Valor
Tipo	Sustitución monoalfabética
Clave necesaria	Tabla personalizada (puede deducirse)
Dificultad en CTF	Media
Común en retos	Sí, especialmente en análisis manual
Criptoanálisis	Frecuencia, patrones, deducción progresiva
Herramientas clave	dCode, Quipqiup, análisis por frecuencia



## Transposition Cipher (Cifrado por transposición)

### ¿Qué es?

El **Transposition Cipher** (Cifrado por Transposición) **no cambia las letras**, sino que **cambia su posición** en el mensaje según una regla o patrón.

A diferencia de los cifrados por sustitución (que cambian una letra por otra), la transposición **reordena** los caracteres, lo que puede dificultar su análisis si no se conoce el sistema o la clave.

### Tipos comunes de transposición

1. **Columnar Transposition** (por columnas - el más común)
2. **Rail Fence Cipher**
3. **Scytale Cipher** (usado por los espartanos)
4. **Route Cipher** (trayectorias en matrices)
5. **Double Transposition Cipher** (dos reordenamientos consecutivos)

## Columnar Transposition Cipher

### ¿Cómo funciona?

Usa una **palabra clave** para organizar el texto en una tabla de columnas y luego reordena las columnas **según el orden alfabético de la clave**.

### Ejemplo

Texto claro:  
WEAREDISCOVEREDFLEEATONCE  
Clave: ZEBRAS

1. Escribes el texto en una tabla bajo la clave (una letra por columna):

```
Z E B R A S
-----
W E A R E D
I I S C O V
E R E D F L
E E A T O N
C E X X X X ← (relleno para completar la tabla)
```

2. Ordenas las columnas **alfabéticamente por la clave**:

```
A B E E N R S Z
```

(Orden real para lectura de columnas según letras de la clave:  
A → 5, B → 2, E → 1, R → 4, S → 6, Z → 0)

3. Lees las columnas en ese orden y escribes el texto resultante.

### ¿Cómo se usa en CTF?

- Texto aparentemente normal con letras mezcladas sin patrón de sustitución.
- Puede aparecer como parte de un reto junto a otro cifrado (ej. Vigenère + Transposición).

- Se oculta entre "ruido" o se combina con esteganografía textual.

## Herramientas útiles

- [dCode Columnar Transposition](#)
- CyberChef → "Column Transposition"
- Scripts Python para cifrar/descifrar con prueba de claves

## Python: Descifrado (Columnar Transposition)

```
from math import ceil

def decrypt_transposition(cipher, key):
    n_cols = len(key)
    n_rows = ceil(len(cipher) / n_cols)
    sorted_key = sorted([(char, i) for i, char in
enumerate(key)])

    # Determinar el número de caracteres en cada columna
    col_lengths = [n_rows] * n_cols
    total_chars = len(cipher)
    extra = n_cols * n_rows - total_chars
    for i in range(extra):
        col_lengths[sorted_key[-(i+1)][1]] -= 1

    # Rellenar las columnas con caracteres del mensaje cifrado
    index = 0
    columns = [''] * n_cols
    for char, original_index in sorted_key:
        l = col_lengths[original_index]
        columns[original_index] = cipher[index:index + l]
        index += l

    # Reconstruir el texto
    result = ''
    for i in range(n_rows):
        for col in columns:
            if i < len(col):
                result += col[i]
    return result

print(decrypt_transposition("EVLNEACDTESEAROFODEECWIREE",
"ZEBRAS"))
```

## ✓ Resumen rápido

Elemento	Valor
Tipo	Cifrado por transposición
Estructura	Reorganiza letras en columnas/filas
Requiere clave	Sí (clave determina el orden de columnas)
Común en CTF	Bastante
Dificultad	Media
Pistas de detección	Letras con distribución normal, sin cifrado visible
Herramientas clave	dCode, CyberChef, scripts

# ●● Symbol-Based Encodings: Morse Code y Tap Code

## 1 📡 Morse Code

### 🧠 ¿Qué es?

El **código Morse** es un sistema de codificación en el que **cada letra o número se representa mediante una secuencia de puntos (.) y rayas (-)**. Fue diseñado para ser transmitido a través de **señales audibles, visuales o eléctricas**.

Es uno de los sistemas de codificación **más reconocibles** y aún se utiliza en aviación, navegación y... en muchos CTFs.

### 📖 Ejemplo básico:

Letra	Código
A	. -
B	- . . .
C	- . - .
E	.
S	. . .
T	-
1	. - - - -
0	- - - - -

### 🔄 Decodificación:

Texto Morse:

.... . .-... .-... --- / .-- --- .-. .-... -..

Separadores:

- Espacios entre letras
- / o (doble espacio) entre palabras

🔍 Traducción: **HELLO WORLD**



## Identificación en CTF:

- Texto hecho con solo . y -
- También visualmente (luces, líneas, símbolos, etc.)
- A veces disfrazado con emojis, señales de audio, flashes o sonidos

## Herramientas útiles:

- [dCode Morse Code](#)
- [CyberChef → Morse Decode](#)
- Morse decoders online: solo pegas .- ...- y listo

## Tap Code (código de golpes)

### ¿Qué es?

El **Tap Code** (o código de golpecitos) es una forma de codificar letras mediante **golpes repetidos** que indican una posición dentro de una matriz 5x5.

Fue muy usado por **prisioneros de guerra** para comunicarse en secreto, golpeando las paredes o tuberías.

### Estructura

Se basa en una **tabla 5x5** con las letras del alfabeto (usualmente se omite la K, que se reemplaza por la C):

	1	2	3	4	5
1	A	B	C	D	E
2	F	G	H	I	J
3	L	M	N	O	P
4	Q	R	S	T	U
5	V	W	X	Y	Z

Cada letra se codifica por **dos series de golpes**:

- La primera indica la **fila**
- La segunda, la **columna**

## Ejemplo:

Letra C → Fila 1, Columna 3 → Golpes: • •• → 1 3

Texto:

1 1 1 2 1 3

☐ A B C

## También puede verse como:

••• •••• → N (fila 3, col 4)

•• ••• → G (fila 2, col 3)

## Identificación en CTF:

- Texto con pares de números (ej. 2 3 3 4 1 5)
- Serie de puntos . o símbolos repetidos por grupos
- A veces representado con emojis repetidos (🐵🐵🐵🐵), sonidos, parpadeos...

## Herramientas útiles:

- [dCode Tap Code](#)
- Conversores online de Tap Code
- Traductores Morse a Tap (en algunos scripts personalizados)

## Comparación rápida

Característica	Morse Code	Tap Code
Tipo	Código de símbolos (sonoro o visual)	Matriz 5x5 (golpes)
Alfabeto	Completo (A-Z, 0-9)	Solo letras (sin K, se usa C)
Longitud por letra	Variable (puntos y rayas)	Fija (dos números o golpes)

Característica	Morse Code	Tap Code
Uso CTF	Muy común	Común en retos de estilo clásico
Complejidad	Baja	Media (requiere conocer la tabla)
Representación	., -, /, 0/1, sonidos	Golpes, puntos repetidos, emojis

## Biological-Looking Encoding: DNA Encoding

### ¿Qué es?

La **codificación ADN (DNA Encoding)** es un método de representación de datos digitales mediante las **cuatro bases nitrogenadas del ADN**:

A (Adenina)  
C (Citosina)  
G (Guanina)  
T (Timina)

En CTFs y criptografía, se usa para **disfrazar texto o binario** como si fuera una secuencia genética. Aunque es una forma de cifrado simple, es muy efectiva para confundir a quien no lo reconoce.

### ¿Cómo funciona?

#### Codificación binaria a ADN

Se suele asignar un par de bits a cada letra del alfabeto del ADN, por ejemplo:

Bits	Base
00	A
01	C
10	G
11	T

Esto quiere decir que cada letra de ADN **representa 2 bits**. Entonces, 1 byte (8 bits) se representa con **4 letras** de ADN.

### Ejemplo

Texto: A

- Paso 1: Convertir A a binario → 01000001
- Paso 2: Dividir en bloques de 2 bits: 01 00 00 01

- Paso 3: Traducir a ADN:  
 01 → C  
 00 → A  
 00 → A  
 01 → C

 Resultado: **CAAC**

## Decodificación inversa

Cadena ADN: GTAC

- G = 10, T = 11, A = 00, C = 01
- Binario: 10110001
- En ASCII → ±

*(Claro, no todos los pares codifican letras imprimibles. Suelen agruparse más caracteres para recuperar texto legible.)*

## Variante avanzada

A veces el texto no se codifica desde ASCII → binario → ADN, sino que se usa **una tabla directa** como:

Letra	ADN
A	GCT
B	CGT
C	AGT
...	...

Pero esto ya depende del reto. Lo más habitual es usar el método **binario → ADN por pares**.

## ¿Cómo identificarlo en un reto?

- Texto compuesto **solo por las letras A, C, G y T**
- Parecer una secuencia genética falsa o sospechosa
- Texto largo con longitud múltiplo de 4
- El nombre del archivo puede dar pistas: dna.txt, gene.log, etc.

## Herramientas útiles

- [CyberChef → DNA Decode](#)
- [dCode DNA Encoding](#)
- Scripts en Python para traducir ADN a binario o ASCII

## Ejemplo en Python

```
def dna_to_text(dna_seq):
    bits = ''
    mapping = {'A': '00', 'C': '01', 'G': '10', 'T': '11'}
    for base in dna_seq:
        bits += mapping[base]

    # Dividir en bytes
    chars = [bits[i:i+8] for i in range(0, len(bits), 8)]
    return ''.join([chr(int(c, 2)) for c in chars if len(c) ==
8])

dna = 'CAAC'
print(dna_to_text(dna))  # → A
```

## Resumen rápido

Elemento	Valor
Nombre	DNA Encoding
Basado en	Secuencia binaria representada con ACGT
Alfabeto usado	A, C, G, T
Relación	1 base = 2 bits → 1 byte = 4 letras ACGT
Uso típico en CTF	Disfrazar texto como datos biológicos
Dificultad	Baja-media
Herramientas	CyberChef, dCode, Python



## XOR Cipher



### ¿Qué es?

El cifrado XOR (o Exclusive OR) es una operación lógica básica usada para cifrar y descifrar datos de manera simple.

- La operación XOR se representa con el símbolo  $\oplus$  o  $\wedge$  en muchos lenguajes.
- Es una operación bit a bit que compara dos bits y da como resultado:
  - 1 si los bits son diferentes
  - 0 si son iguales

A	B	A XOR B
0	0	0
0	1	1
1	0	1
1	1	0



### ¿Cómo funciona el cifrado XOR?

- Para cifrar un mensaje (plaintext) con una clave (key), se hace un XOR bit a bit entre ellos.
- Para descifrar, se aplica la misma operación XOR con la misma clave al texto cifrado.



### Fórmulas

Sea:

- MM = mensaje original en bits
- KK = clave en bits (puede ser una secuencia de bytes)
- CC = ciphertext (mensaje cifrado)

Entonces:

$$C = M \oplus K$$

Y para descifrar:

$$M = C \oplus K \quad M = C \text{ XOR } K$$

## Características clave

- **Simétrico:** La misma operación y clave se usan para cifrar y descifrar.
- **Sencillo y rápido** de implementar.
- **Vulnerable si la clave es corta** o se reutiliza (problema conocido en sistemas como el cifrado de flujo).
- Base para cifrados más complejos (one-time pad, RC4, etc).

## Uso en CTFs

- Muy común en retos de cifrado básicos o para ocultar flags.
- A menudo la clave es corta o repetitiva, lo que facilita ataques de análisis de frecuencia o cribado.
- Puede usarse con claves de un solo byte (XOR byte simple) o claves más largas (XOR de flujo).
- Ejercicios típicos:
  - Encontrar la clave por fuerza bruta.
  - Analizar patrones.
  - Usar herramientas para XOR automático.

## Cómo atacar o resolver cifrados XOR

- **XOR de un solo byte:** probar con las 256 posibles claves.
- **XOR con clave repetida (repeated-key XOR):** usar técnicas de análisis de frecuencia, cribado o la distancia de Hamming para estimar la longitud de la clave.
- **One-Time Pad (OTP):** Si la clave es verdaderamente aleatoria y tan larga como el mensaje, es teóricamente irrompible.



## Ejemplo sencillo en Python

```
def xor_encrypt_decrypt(data: bytes, key: bytes) -> bytes:
    return bytes([b ^ key[i % len(key)] for i, b in
enumerate(data)])
```

```
# Ejemplo:
mensaje = b"hola mundo"
clave = b"key"
cifrado = xor_encrypt_decrypt(mensaje, clave)
descifrado = xor_encrypt_decrypt(cifrado, clave)
```

```
print("Cifrado:", cifrado)
print("Descifrado:", descifrado.decode())
```

## Resumen rápido

Elemento	Valor
Tipo	Cifrado simétrico básico
Operación clave	XOR bit a bit
Uso en CTF	Muy frecuente en retos
Vulnerabilidades	Reutilización o claves cortas
Herramientas útiles	Scripts para fuerza bruta, análisis de frecuencia



## Affine Cipher (Cifrado Afín)



### ¿Qué es?

El cifrado afín es un cifrado por sustitución monoalfabético que combina una transformación lineal y un desplazamiento sobre los índices de las letras del alfabeto.

Matemáticamente, cada letra  $xx$  (convertida a un número, por ejemplo,  $A=0, B=1, \dots, Z=25$ ) se transforma usando la fórmula:

$$E(x) = (ax + b) \bmod m$$

Donde:

- $a$  y  $b$  son claves enteras con  $a$  **coprimo** con  $m$  (el tamaño del alfabeto, normalmente 26)
- $m$  es el tamaño del alfabeto
- $E(x)$  es la letra cifrada

Para descifrar, se usa la inversa modular  $a^{-1}$  de  $a$  módulo  $m$ :

$$D(y) = a^{-1} \times (y - b) \bmod m$$



### ¿Cómo funciona?

1. Convertir letras a números:  $A=0, B=1, \dots, Z=25$
2. Aplicar la fórmula de cifrado para cada letra.
3. Convertir el número resultante a la letra correspondiente.



### Condiciones clave

- $a$  debe ser coprimo con  $m$  (por ejemplo, 26) para que la función sea invertible.
- $b$  es el desplazamiento o "shift".
- Si  $a=1$ , el cifrado afín se reduce al cifrado César clásico.



## Uso en CTF

- Es un cifrado clásico que suele aparecer en retos fáciles o de nivel medio.
- Se puede romper por fuerza bruta (hay pocos valores posibles de aa y bb).
- También es común combinarlo con otros cifrados para mayor dificultad.
- El análisis de frecuencia ayuda a encontrar aa y bb.



## Cómo romperlo

- Probar todas las combinaciones válidas de aa y bb.
- Usar análisis de frecuencia para encontrar las letras más comunes.
- Usar herramientas automáticas o scripts que prueban todas las claves posibles.



## Ejemplo básico en Python

```
def modinv(a, m):
    # Calcula el inverso modular usando Extended Euclidean
    Algorithm
    for x in range(1, m):
        if (a * x) % m == 1:
            return x
    return None

def affine_encrypt(text, a, b):
    result = ""
    m = 26
    for char in text.upper():
        if char.isalpha():
            x = ord(char) - ord('A')
            y = (a * x + b) % m
            result += chr(y + ord('A'))
        else:
            result += char
    return result

def affine_decrypt(cipher, a, b):
    result = ""
    m = 26
```

```

a_inv = modinv(a, m)
if a_inv is None:
    return None # no existe inverso modular
for char in cipher.upper():
    if char.isalpha():
        y = ord(char) - ord('A')
        x = (a_inv * (y - b)) % m
        result += chr(x + ord('A'))
    else:
        result += char
return result

# Ejemplo
clave_a = 5
clave_b = 8
texto = "AFFINE CIPHER"
cifrado = affine_encrypt(texto, clave_a, clave_b)
descifrado = affine_decrypt(cifrado, clave_a, clave_b)

print("Texto:", texto)
print("Cifrado:", cifrado)
print("Descifrado:", descifrado)

```

## ✓ Resumen rápido

Elemento	Valor
Tipo	Cifrado por sustitución monoalfabético
Fórmula cifrado	$E(x) = (ax + b) \bmod m$
Requisito clave	a coprimo con m
Uso en CTF	Común en retos clásicos o combinados
Vulnerabilidades	Fácil de romper por fuerza bruta y análisis
Herramientas útiles	Scripts para probar todas las claves



## Enigma Cipher



### ¿Qué es?

El cifrado **Enigma** es un sistema de cifrado electromecánico utilizado principalmente por Alemania durante la Segunda Guerra Mundial para proteger comunicaciones militares.

Era una máquina que usaba rotores giratorios para crear sustituciones polialfabéticas muy complejas y cambiantes, haciendo el mensaje muy difícil de descifrar sin conocer la configuración exacta.



### ¿Cómo funciona?

- La máquina Enigma tiene varios componentes clave:
  1. **Rotores (Rotors)**: Discos con conexiones internas que sustituyen letras y rotan tras cada pulsación, cambiando la sustitución constantemente.
  2. **Reflector (Reflector)**: Envía la señal de vuelta por los rotores, haciendo que el cifrado sea simétrico (cifrado y descifrado usan la misma configuración).
  3. **Plugboard (Steckerbrett)**: Un panel con cables que intercambian pares de letras antes y después del paso por rotores, añadiendo una capa extra de permutación.
- Cada vez que se presiona una tecla, los rotores giran en un patrón determinado, cambiando la configuración del cifrado para cada letra.



### Detalles técnicos

- Normalmente se usaban 3 rotores seleccionados de un conjunto mayor.
- Cada rotor tiene 26 posiciones, una para cada letra.
- El plugboard permite intercambiar hasta 10 pares de letras.
- La máquina produce un cifrado polialfabético muy difícil de romper con técnicas clásicas.

## Características clave

- **Polialfabético:** El mapeo cambia para cada letra.
- **Simétrico:** Cifrar y descifrar se hacen con la misma configuración.
- **Gran espacio de clave:** Combinaciones de rotores, posiciones iniciales y conexiones del plugboard.
- **Dificultad para romperlo:** Sin configuraciones era casi imposible en la época, pero la labor de los criptoanalistas (como los de Bletchley Park) logró romperlo.

## Uso en CTF

- Retos históricos o simulaciones.
- Cifrar mensajes con configuraciones Enigma.
- Decodificación con herramientas o simuladores.
- Análisis y criptoanálisis básico con configuraciones conocidas.

## Cómo atacar un cifrado Enigma en CTF

- Obtener o deducir la configuración (orden de rotores, posiciones iniciales, plugboard).
- Usar simuladores para probar configuraciones.
- Aplicar cribados (palabras conocidas dentro del mensaje).
- Automatizar búsqueda con fuerza bruta o heurísticas.

## Herramienta útil

- [Py-enigma](#): Simulador de máquina Enigma en Python.
- [Enigma Simulator](#) (web): Simulador online.

## Ejemplo básico con Py-enigma (simplificado)

```
from enigma.machine import EnigmaMachine

# Crear máquina Enigma con configuración
machine = EnigmaMachine.from_key_sheet(
    rotors='I II III',
    reflector='B',
    ring_settings='01 01 01',
    plugboard_settings='AV BS CG DL FU HZ IN KM OW RX'
)

# Establecer posiciones iniciales de rotores
machine.set_display('WXC')

plaintext = 'HELLOWORLD'
ciphertext = machine.process_text(plaintext)
print(f"Cifrado: {ciphertext}")

# Descifrar usando la misma configuración
machine.set_display('WXC')
decrypted = machine.process_text(ciphertext)
print(f"Descifrado: {decrypted}")
```

### Resumen rápido

Elemento	Valor
Tipo	Cifrado mecánico polialfabético
Componentes clave	Rotores, reflector, plugboard
Característica	Sustitución variable para cada letra
Uso histórico	Segunda Guerra Mundial
Uso en CTF	Simulaciones, criptoanálisis, retos históricos
Dificultad	Alta sin configuraciones, baja con ellas



## Cifrado César (Caesar Cipher)



### ¿Qué es?

El cifrado César es uno de los cifrados más antiguos y simples. Consiste en desplazar las letras del alfabeto un número fijo de posiciones.

Por ejemplo, con un desplazamiento de +3, la letra **A** se convierte en **D**, la **B** en **E**, y así sucesivamente.



### ¿Cómo funciona?

1. Se asigna a cada letra un número (ej. A=0, B=1, ..., Z=25).
2. Se suma el desplazamiento (clave)  $k$  al número de la letra.
3. Se aplica módulo 26 para que vuelva al inicio del alfabeto si se pasa del final.
4. Se convierte el número resultante de nuevo a letra.



### Fórmula

$$E(x) = (x + k) \bmod 26$$

Para descifrar:

$$D(y) = (y - k) \bmod 26$$



### Características clave

- **Sustitución monoalfabética:** todas las letras se desplazan igual.
- **Clave sencilla:** un solo número  $k$  (entre 1 y 25).
- **Vulnerable a ataque de fuerza bruta:** solo 25 posibles claves.
- **Muy básico y pedagógico.**





## Uso en CTF

- Retos introductorios.
- A veces usado para ocultar flags de forma muy simple.
- Puede venir combinado con otros cifrados.
- Fácil de romper probando las 25 claves posibles o análisis de frecuencia.



## Cómo romperlo

- Probar todas las claves posibles (fuerza bruta).
- Usar análisis de frecuencia para determinar el desplazamiento.
- Scripts automáticos o incluso a mano en pocos segundos.



## Ejemplo en Python

```
def caesar_encrypt(text, k):
    result = ""
    for char in text.upper():
        if char.isalpha():
            x = ord(char) - ord('A')
            y = (x + k) % 26
            result += chr(y + ord('A'))
        else:
            result += char
    return result

def caesar_decrypt(text, k):
    return caesar_encrypt(text, -k % 26)

# Ejemplo
clave = 3
texto = "Hola Mundo"
cifrado = caesar_encrypt(texto, clave)
descifrado = caesar_decrypt(cifrado, clave)

print("Texto:", texto)
print("Cifrado:", cifrado)
print("Descifrado:", descifrado)
```

## ✓ Resumen rápido

Elemento	Valor
Tipo	Sustitución monoalfabética
Clave	Número entero (1 a 25)
Fórmula	$(x+k) \bmod 26$ $(x + k) \bmod 26$
Seguridad	Muy baja, fácil de romper
Uso en CTF	Retos básicos, introducción

## Cifrado Atbash

### ¿Qué es?

El cifrado Atbash es un cifrado por sustitución monoalfabético que invierte el alfabeto.

Esto significa que:

- La letra **A** se sustituye por la **Z**
- La **B** por la **Y**
- La **C** por la **X**, y así sucesivamente.

Es un cifrado simétrico y muy simple que data de la antigüedad (usado en el hebreo, y también en latín).

### ¿Cómo funciona?

- Se mapea cada letra  $x$  del alfabeto a su “opuesta”, es decir:

$$E(x) = (m-1) - x \quad E(x) = (m - 1) - x$$

donde  $m$  es la longitud del alfabeto (por ejemplo, 26).

Si  $A=0$ , entonces:

- $A \rightarrow Z$  (25)
- $B \rightarrow Y$  (24)
- $C \rightarrow X$  (23), etc.

### Fórmula

$$E(x) = 25 - x \quad E(x) = 25 - x$$

(con alfabeto de 26 letras, con índice desde 0 a 25)

## Características clave

- **Cifrado simétrico:** cifrar y descifrar es la misma operación.
- **No necesita clave.**
- **Fácil de romper y entender.**
- **Sustitución monoalfabética fija.**

## Uso en CTF

- Retos básicos o para esconder texto simple.
- A veces mezclado con otros cifrados para mayor dificultad.
- Útil para practicar sustituciones y patrones.

## Ejemplo en Python

```
def atbash(text):
    result = ""
    for char in text.upper():
        if char.isalpha():
            x = ord(char) - ord('A')
            y = 25 - x
            result += chr(y + ord('A'))
        else:
            result += char
    return result

# Ejemplo
texto = "ATBASH CIPHER"
cifrado = atbash(texto)
descifrado = atbash(cifrado)

print("Texto:", texto)
print("Cifrado:", cifrado)
print("Descifrado:", descifrado)
```

## ✓ Resumen rápido

Elemento	Valor
Tipo	Sustitución monoalfabética
Clave	No tiene
Fórmula	$E(x)=25-x$ $E(x) = 25 - x$
Seguridad	Muy baja
Uso en CTF	Retos básicos o combinados

## Keyboard Shift Cipher

### ¿Qué es?

El **Keyboard Shift Cipher** (también llamado **QWERTY Shift**, **Keyboard Layout Cipher** o **Keyboard Offset**) es una técnica de cifrado por sustitución en la que las letras del texto han sido **desplazadas según su posición en el teclado físico**, no en el alfabeto.

En vez de aplicar una regla como en ROT13 (que actúa sobre el orden alfabético), este cifrado **sustituye cada letra por otra que esté a una determinada distancia** sobre el teclado (por ejemplo: una tecla a la izquierda o arriba).

### ¿Cómo funciona?

Existen varias versiones, pero las más comunes en CTF son:

#### 1. Shift hacia la izquierda o derecha (línea horizontal)

Por ejemplo:

**QWERTY fila Original: q w e r t y u i o p**  
**Shift -1 (izq) Resultado q w e r t y u i o**

Texto cifrado: sdrs

- Teclas desplazadas 1 a la izquierda:  
s → a  
d → s  
r → e  
s → a

 Resultado: asea

#### 2. Shift vertical (arriba o abajo en el teclado)

**QWERTY      1 2 3 4 5 6 7 8 9 0**  
**QWERTY debajo q w e r t y u i o p**

Ejemplo: si el texto usa e, podrías mapearla a la tecla **arriba** (3), o la **de abajo** (d).

Este método es más visual y puede variar mucho, según el teclado.

### ¿Dónde se usa?

- Errores de tipeo sospechosamente consistentes
- Mensajes con letras poco coherentes pero con patrón
- Textos con símbolos del teclado (!@#, [], ;', etc.)

### Identificación en CTF

- Texto sin sentido pero que tiene misma longitud que una palabra clave
- Letras adyacentes al QWERTY real
- Coincidencias con teclas cercanas
- Usualmente sin cifrado extra (ni base64, ni XOR): solo "mal tipeado"

### Estrategia para resolverlo

1. Mira el texto y escribe su posible palabra “correcta” en QWERTY.
  2. Prueba con un teclado físico o virtual:
    - Desplaza hacia izquierda, derecha, arriba o abajo
  3. También puedes usar herramientas online como:
- [dCode Keyboard Shift Cipher](#)
  - [CyberChef → Keyboard Shift \(custom function\)](#)

### Ejemplo completo

Texto: pl,

Parece aleatorio, pero están a la derecha de okm

☑ Teclas una posición a la izquierda:

p → o

l → k

, → m

Resultado: okm

### Resumen rápido

Elemento	Valor
Nombre	Keyboard Shift Cipher
Tipo	Sustitución basada en disposición de teclado físico
Variantes	Izquierda/derecha, arriba/abajo, incluso diagonal
Requiere	Conocer diseño QWERTY (o AZERTY si se indica)
Frecuencia en CTF	Media (usado en desafíos de OSINT, forense, análisis)
Herramientas útiles	Teclado físico/virtual, dCode, CyberChef
Dificultad	Baja-media



## Polybius Square Cipher

### ¿Qué es?

El **Cifrado de la Cuadrícula de Polybius** es un **cifrado por sustitución** que representa cada letra como un par de números, según su posición en una **tabla de 5x5**.

Fue inventado por el historiador griego **Polibio** en el siglo II a.C. y es uno de los métodos más antiguos de codificación.

### ¿Cómo funciona?

Se construye una tabla de 5 filas x 5 columnas con las letras del alfabeto. Como hay **26 letras** y solo caben 25, se combinan normalmente I y J en una sola celda.

**Tabla clásica (con I/J juntas):**

	1	2	3	4	5
1	A	B	C	D	E
2	F	G	H	I/J	K
3	L	M	N	O	P
4	Q	R	S	T	U
5	V	W	X	Y	Z

Para cifrar, se toma la **fila y columna** de cada letra.

### Ejemplo: Cifrar la palabra HELLO

1. H → Fila 2, Columna 3 → 23
2. E → 15
3. L → 31
4. L → 31
5. O → 34

 **Resultado cifrado: 2315313134**

También puede aparecer con espacio entre pares: 23 15 31 31 34

## Para descifrar

Tomas cada par de dígitos y lo buscas en la tabla como coordenadas [fila][columna].

Ejemplo: 23 = fila 2, columna 3 → H

## ¿Cómo aparece en CTFs?

- Como lista de pares numéricos (ej: 11 12 24 34...)
- A veces con letras en minúscula (indicando I/J)
- En forma de imagen con una cuadrícula
- En archivos .txt, .csv, o incluso escondido en mensajes ocultos

## Variantes

- Cuadrículas de 6×6 (para incluir números)
- Uso de letras diferentes (clave personalizada)
- Versiones visuales con símbolos, colores o emojis
- Reordenación de la cuadrícula según una **clave secreta**

## Ejemplo con clave

Clave: ZEBRAS → reorganizas el alfabeto eliminando duplicados y lo llenas en la tabla.

	1	2	3	4	5
1	Z	E	B	R	A
2	S	C	D	F	G
3	H	I/J	K	L	M
4	N	O	P	Q	T
5	U	V	W	X	Y

Este tipo de clave se usa para cifrados más avanzados en CTF.

## Herramientas útiles

- [dCode - Polybius Square Cipher](#)
- CyberChef → “Polybius Square”
- Script Python simple

## Python - Descifrado básico

```
def polybius_decrypt(cipher):
    table = [
        ['A', 'B', 'C', 'D', 'E'],
        ['F', 'G', 'H', 'I/J', 'K'],
        ['L', 'M', 'N', 'O', 'P'],
        ['Q', 'R', 'S', 'T', 'U'],
        ['V', 'W', 'X', 'Y', 'Z']
    ]
    result = ''
    for i in range(0, len(cipher), 2):
        row = int(cipher[i]) - 1
        col = int(cipher[i+1]) - 1
        result += table[row][col][0] # usa solo I en I/J
    return result

print(polybius_decrypt("2315313134")) # → HELLO
```

## Resumen rápido

Elemento	Valor
Tipo	Sustitución (basado en coordenadas numéricas)
Clave necesaria	No (pero puede usarse para cuadrícula personalizada)
Letras codificadas	Como pares de números [fila][columna]
Longitud esperada	Múltiplo de 2
Letras I y J	Suelen compartir celda
Común en CTF	Sí, especialmente con pares numéricos ocultos
Herramientas útiles	dCode, CyberChef, Python
Nivel de dificultad	Bajo - medio

## Rail Fence Cipher

### ¿Qué es?

El **Rail Fence Cipher** (también llamado **zigzag cipher**) es un **cifrado por transposición**, no de sustitución. En lugar de cambiar las letras por otras, **reordena las posiciones de las letras del mensaje original** siguiendo un patrón en zigzag.

Es simple de implementar y rápido de descifrar si sabes el número de "raíles" o niveles.

### ¿Cómo funciona?

1. Eliges un número de "rails" (niveles de riel).
2. Escribes el mensaje en zigzag sobre esos rieles.
3. Luego, lees línea por línea (de arriba hacia abajo) para obtener el texto cifrado.

### Ejemplo: mensaje WEAREDISCOVEREDFLEEATONCE

Usamos 3 "rails":

```
W . . . E . . . C . . . R . . . L . . . T . . . E
. E . R . D . S . O . E . E . F . E . A . O . C .
. . A . . . I . . . V . . . D . . . N . . . E . .
```

Cuando lo lees línea por línea:

```
Primera línea:   WECRLTE
Segunda línea:   ERDSOEEFEAOOC
Tercera línea:   AIVDEN
```

→ Texto cifrado: **\*\*WECRLTEERDSOEEFEAOOCAIVDEN\*\***

### Descriptación

Para descifrar, necesitas:

1. Saber el número de "rails".
2. Volver a dibujar el zigzag con guiones o posiciones vacías.

3. Rellenar las letras del mensaje cifrado según el patrón.
4. Leer en orden para obtener el mensaje original.

### ¿Dónde se usa en CTF?

- Cuando ves un mensaje aparentemente aleatorio pero sin cifrado estándar (no base64, no hexadecimal).
- Cuando el mensaje tiene longitud similar a la original pero no tiene sentido claro.
- A veces está disfrazado de arte ASCII, onda de texto o niveles con saltos.

### Herramientas útiles

- [dCode - Rail Fence Cipher](#)
- CyberChef → “Rail Fence Decode”
- Script Python

### Python - Descifrado (3 rails)

```
def decrypt_rail_fence(cipher, rails):
    pattern = [''] * rails
    idx = 0
    direction = 1
    rail_pattern = []

    for _ in cipher:
        rail_pattern.append(idx)
        idx += direction
        if idx == 0 or idx == rails - 1:
            direction *= -1

    counts = [rail_pattern.count(r) for r in range(rails)]
    pos = 0
    rail_letters = []
    for count in counts:
        rail_letters.append(list(cipher[pos:pos+count]))
        pos += count

    result = ''
    for r in rail_pattern:
        result += rail_letters[r].pop(0)
```

```
return result
```

```
cipher = "WECRLTEERDSOEFEAOCAIVDEN"  
print(decrypt_rail_fence(cipher, 3)) # →  
WEAREDISCOVEREDFLEEATONCE
```

## ✓ Resumen rápido

Elemento	Valor
Tipo	Transposición
Clave	Número de rails (líneas)
Letras modificadas	No, solo cambia su posición
Reconocimiento	Texto de longitud normal pero sin sentido claro
Uso común en CTF	Medio (muy usado como cifrado oculto)
Herramientas útiles	dCode, CyberChef, Python
Dificultad	Baja-media

## Brainfuck

### ¿Qué es?

**Brainfuck** es un lenguaje de programación **esotérico**, creado en 1993 por Urban Müller, diseñado para ser minimalista y extremadamente difícil de leer. Se usa en CTFs y retos de programación para ocultar mensajes o lógica bajo un código casi ilegible.

### Características clave

- Solo tiene **8 comandos**.
- Trabaja con una **cinta de memoria** (array) y un puntero que se mueve a lo largo de ella.
- El código no usa palabras ni símbolos normales, sino símbolos específicos: `><+-.,[ ]`

### Los comandos de Brainfuck

Comando	Descripción
>	Mueve el puntero a la derecha
<	Mueve el puntero a la izquierda
+	Incrementa el byte en la posición actual
-	Decrementa el byte en la posición actual
.	Imprime el carácter ASCII del byte actual
,	Lee un carácter de la entrada y lo guarda en la posición actual
[	Si el byte actual es 0, salta al comando después del ] correspondiente
]	Si el byte actual no es 0, vuelve al comando después del [ correspondiente

### ¿Cómo funciona?

El programa opera sobre una cinta (array) de bytes, inicialmente cero, y un puntero que apunta a una celda. Mediante los comandos, modifica valores y mueve el puntero para manipular datos y mostrar resultados.

## Uso en CTFs

- Ocultar texto cifrado o mensajes en código Brainfuck.
- Programas ofuscados que al ejecutarse revelan banderas.
- Rompecabezas para entender el flujo y decodificar manualmente o con herramientas.

## Cómo resolver retos con Brainfuck

- Ejecutar el código en intérpretes online o locales para ver qué imprime.
- Analizar el código para entender qué hace, especialmente ciclos y modificaciones.
- Convertir código a texto si es muy simple.
- Usar herramientas como:
  - <https://copy.sh/brainfuck/>
  - <https://fatiherikli.github.io/brainfuck-visualizer/>
  - Scripts en Python que interpreten Brainfuck

## Ejemplo simple de Brainfuck

Código para imprimir “A” (ASCII 65):

```
+++++++[>++++>++++>++++>+<<<<-]>+.
```

Explicación rápida:

- ++++++ → incrementa la primera celda a 10
- [>++++>++++>++++>+<<<<-] → ciclo para llenar otras celdas con valores
- >+. → mueve el puntero, incrementa y muestra el carácter

## Resumen rápido

Elemento	Valor
Tipo	Lenguaje de programación esotérico
Comandos	8 (><+-.,[,])
Propósito	Obfuscación, retos, enseñanza de lógica
Uso en CTF	Alto para ocultar banderas y código ofuscado
Dificultad	Media-alta (para interpretar manualmente)
Herramientas útiles	Intérpretes online, visualizadores, scripts



## JSFuck

### ¿Qué es?

JSFuck es una técnica de ofuscación de JavaScript que permite escribir **cualquier código JavaScript válido usando solo 6 caracteres:**

[ ] ( ) ! +

Se aprovecha del comportamiento del intérprete de JavaScript y de coerciones de tipo para representar caracteres y expresiones complejas con estas pocas piezas.

### ¿Cómo funciona?

JSFuck construye todo el código a partir de combinaciones muy rebuscadas de estos símbolos. Por ejemplo:

- `![]` representa `false`
- `!![]` representa `true`
- `[]` representa un arreglo vacío
- Combinando estos y con operadores se pueden formar números y letras.

Por ejemplo, el carácter "a" se construye como:

```
(![]+[])[1]
```

Aquí:

- `![]` → `false`
- `![]+[]` → `"false"` (cadena)
- `[1]` → accede al segundo carácter: `"a"`

### Características clave

- Permite escribir **cualquier código JavaScript** (variables, funciones, expresiones).
- Muy usado para ofuscar código malicioso o para retos que buscan ocultar banderas en scripts.
- Aunque parece código inútil, el motor JS lo interpreta correctamente.

## Uso en CTF y seguridad

- Ocultar **payloads** o scripts maliciosos en un formato difícil de leer.
- Reto para **reconstruir** el código original.
- Ejercicios para entender coerciones y tipos en JS.

## Cómo resolver retos JSFuck

- Puedes usar **deofuscadore**s online que transforman JSFuck en código legible:
  - <https://enkhee-osiris.github.io/Decoder-JSFuck/>
- Ejecutar el código en un entorno controlado para ver qué hace.
- Analizar patrones comunes de JSFuck.

## Ejemplo mínimo

Código JSFuck para mostrar `alert('Hi')` sería una cadena enorme con solo los 6 símbolos. Por ejemplo, solo `alert` se escribe así:

```
[ ] [ ( ! [ ] + [ ] ) [ + [ ] ] + ( ! [ ] + [ ] ) [ ! + [ ] + ! + [ ] ] + ... ]
```

(¡Es muy largo y complicado!)

## Resumen rápido

Elemento	Valor
Tipo	Ofuscación de JavaScript usando solo 6 caracteres
Caracteres usados	[ ] ( ) ! +
Propósito	Ocultación, ofuscación, retos de seguridad
Uso en CTF	Alto en retos de web y scripting
Dificultad	Alta para leer manualmente, fácil con herramientas
Herramientas útiles	JSFuck Decoder online, entornos JS

## Esoteric Languages: Malbolge

### ¿Qué es?

**Malbolge** es un lenguaje de programación esotérico creado en 1998 por Ben Olmstead, diseñado para ser prácticamente imposible de entender o programar manualmente. Su nombre viene del octavo círculo del infierno en "La Divina Comedia" de Dante, simbolizando lo infernalmente difícil que es.

### Características clave

- Lenguaje auto-modificable: el código se **modifica a sí mismo** durante la ejecución.
- Su sintaxis y comportamiento fueron diseñados para ser extremadamente crípticos.
- Las instrucciones son bytes que cambian con cada ejecución.
- Muy difícil de escribir o leer sin herramientas automáticas.

### Comandos básicos

Malbolge opera sobre una memoria con valores de 10 bits y tiene un conjunto limitado de instrucciones codificadas en caracteres específicos. No es sencillo memorizar comandos como en Brainfuck.

### ¿Cómo funciona?

- El código fuente es una cadena de caracteres.
- Cada instrucción es transformada antes de ejecutarse.
- El programa modifica su propia memoria, complicando el seguimiento.
- El comportamiento depende del estado interno y del programa anterior.

### Uso en CTFs

- Retos de ingeniería inversa y análisis de código.
- Descifrar código Malbolge para obtener banderas ocultas.
- Rara vez se programa a mano; se usan generadores o decodificadores.

### Cómo resolver retos Malbolge

- Usar **intérpretes Malbolge** para ejecutar código.
- Herramientas para **desensamblar** o traducir a lenguaje más legible.
- Analizar salida y buscar banderas o pistas.

## Ejemplo clásico: “Hello, World!”

El programa "Hello, World!" en Malbolge tiene un código ofuscado imposible de entender a simple vista:

```
(=<`#9]~6ZY32Vx/4Rs+0Po-&JkD"GCA]z'!~}|{zyxwv-,POqponlH+fdb
```

Este código al ejecutarse imprime “Hello, World!” pero su estructura es indescifrable sin un intérprete.

## ✓ Resumen rápido

Elemento	Valor
Tipo	Lenguaje de programación esotérico, auto-modificable
Complejidad	Muy alta, casi imposible de escribir a mano
Uso en CTF	Muy específico, para retos avanzados de ingeniería inversa
Herramientas útiles	Intérpretes Malbolge, desensambladores
Propósito	Reto mental, ofuscación extrema

# WhiteSpace

## ¿Qué es?

Whitespace es un lenguaje de programación esotérico creado en 2003 por Edwin Brady y Chris Morris, que usa **solo espacios, tabulaciones y saltos de línea** para escribir código. **Ignora todos los caracteres visibles**, es decir, el código “visible” es ignorado, y solo cuenta la estructura invisible (espacios, tabs, saltos).

## ¿Cómo funciona?

- Usa tres caracteres significativos:
  - **Espacio** ( )
  - **Tabulación** (\t)
  - **Nueva línea** (\n)
- Cada combinación representa instrucciones, datos, o saltos de flujo.
- Se parece a una máquina de pila (stack machine).
- Ignora cualquier otro carácter (comentarios invisibles).

## Características clave

- Lenguaje de bajo nivel que maneja:
  - Pila (stack)
  - Heap (memoria)
  - Entrada/salida
- Instrucciones se codifican en combinaciones de espacios y tabulaciones.
- Ejemplo: push, pop, add, jump, call, etc.

## Uso en CTFs

- Ocultar código en medio de texto visible.
- Reto para identificar que el código está en Whitespace y no en texto normal.
- Analizar código difícil porque no ves nada en pantalla.
- Ejecutar código oculto en scripts.

## Herramientas para Whitespace

- Intérpretes y compiladores online:
  - <https://vii5ard.github.io/whitespace-online/>
  - <https://copy.sh/whitespace/>
- Desensambladores para convertir secuencias de espacios y tabs a código legible.

## Ejemplo (representación visual)

Por ejemplo, para hacer push 1 en Whitespace sería algo como:

[espacio][tab][espacio][tab][espacio]... (combinación de espacios y tabs)

Pero no verás caracteres “normales” al leer el código.

## Resumen rápido

Elemento	Valor
Tipo	Lenguaje de programación esotérico
Caracteres usados	Solo espacio ( ), tabulación (\t) y salto línea (\n)
Propósito	Ocultación de código, retos de análisis
Uso en CTF	Moderado, especialmente en esteganografía de código
Dificultad	Media-alta (por invisibilidad del código)
Herramientas útiles	Intérpretes online, desensambladores Whitespace

# PikaLang

## ¿Qué es?

**PikaLang** es un lenguaje esotérico inspirado en Pikachu (Pokémon) que usa repeticiones de la palabra "**pika**" y sus variantes para escribir programas.

Cada palabra o combinación representa una instrucción o dato. Es divertido y extraño a la vez, ideal para retos que mezclan cultura pop y programación.

## ¿Cómo funciona?

- Los comandos están formados por combinaciones de sílabas similares a:
  - pika
  - pika-pi
  - pikachu
  - pi-ka
  - etc.
- La sintaxis es muy libre, pero la interpretación depende de la posición y secuencia de estas palabras.
- Cada secuencia representa operaciones básicas como:
  - Manejar memoria
  - Condiciones
  - Loops
  - Entrada/salida

## Características clave

- Lenguaje con temática Pokémon / Pikachu.
- Usado más para diversión y retos ligeros que para programación seria.
- Ejercicios para entender lógica con una capa de ofuscación graciosa.

## Uso en CTF

- Apariciones en retos temáticos o “fun challenges”.
- Codificación o cifrado disfrazada con palabras tipo “pika”.
- Ejercicios de interpretación y traducción de instrucciones.

## Cómo resolver retos PikaLang

- Analizar las repeticiones y patrones de palabras.
- Buscar referencias oficiales o comunitarias para mapear instrucciones.
- Ejecutar intérpretes si están disponibles o crear scripts para automatizar la traducción.

## Ejemplo básico (pseudo)

pika pika-pi pikachu pi-ka pika

Podría representar un programa que imprime un carácter o suma números, según la interpretación del reto.

## Resumen rápido

Elemento	Valor
Tipo	Lenguaje esotérico temático (Pokémon)
Sintaxis	Palabras tipo “pika”, “pikachu”, “pi-ka”, etc.
Propósito	Diversión, retos temáticos, ofuscación ligera
Uso en CTF	Bajo-medio, según la temática
Dificultad	Baja-media (depende del reto y la documentación)
Herramientas útiles	Scripts personalizados, referencias comunitarias





## RSA Attacks and Variants

### 1 Classic RSA

#### Descripción

RSA es un sistema criptográfico asimétrico basado en la dificultad de factorizar un número grande (el módulo  $n = p \times q$ ).

- Clave pública:  $(n, e)$
- Clave privada:  $d$  tal que  $ed \equiv 1 \pmod{\phi(n)}$
- Cifrado:  $c = m^e \pmod n$
- Descifrado:  $m = c^d \pmod n$

### 2 Multi-Prime RSA

#### Descripción

Variante de RSA que usa más de dos primos para generar el módulo  $n = p \times q \times r \times \dots$

#### Ventajas

- Generación de claves y operaciones más rápidas.
- Más eficiente para hardware y entornos específicos.

#### Riesgos

- Dependiendo del número y tamaño de primos, puede introducir nuevas vulnerabilidades en ataques de factorización.

### 3 Cube Root Attack (Low Exponent Attack)

#### Descripción

Si se usa un exponente público pequeño, como  $e = 3$ , y el mensaje cifrado no está adecuadamente rellenado (sin padding), es posible recuperar el mensaje original tomando la raíz cúbica del ciphertext directamente si:

$$m^3 < n \Rightarrow m = \sqrt[3]{c} \implies m = \sqrt[3]{c}$$

### Cómo funciona

- El mensaje es pequeño y sin padding, por lo que el valor cifrado es menor que el módulo.
- Se calcula la raíz cúbica entera del ciphertext para recuperar el mensaje.

### Prevención

- Usar padding adecuado (PKCS#1, OAEP).
- Evitar exponentes muy bajos sin precauciones.

## 4 Wiener's Attack

### Descripción

Ataca claves privadas pequeñas  $dd$ , usando aproximaciones por fracciones continuas.

### Idea clave

Si  $d < \frac{1}{3} n^{1/4}$ , el ataque puede recuperar  $dd$  eficientemente.

### Cómo funciona

- Calcula convergentes de la fracción continua en  $\frac{e}{n}$ .
- Busca valores candidatos para  $dd$ .
- Verifica cuál descifra correctamente.

### Importancia

Demuestra que la clave privada debe ser suficientemente grande para evitar esta vulnerabilidad.

## 5 Common Modulus Attack

### Descripción

Cuando dos mensajes cifrados diferentes comparten el mismo módulo  $n$ , pero tienen diferentes exponentes públicos  $e_1, e_2$ ,  $e_1, e_2$  que son coprimos.

### Cómo funciona

- Se pueden combinar los dos ciphertexts  $c_1 = m^{e_1} \bmod n$ ,  $c_2 = m^{e_2} \bmod n$  para recuperar  $m$  usando el algoritmo extendido de Euclides para hallar  $a, b$  tales que:

$$ae_1 + be_2 = 1$$

- Se calcula:

$$m = c_1^a \times c_2^b \bmod n$$

## 6 Chinese Remainder Theorem Attack (CRT Attack)

### Descripción

Si un mensaje es enviado cifrado con el mismo pequeño exponente  $e$  a varios destinatarios con diferentes módulos, se pueden usar sus ciphertexts para recuperar el mensaje sin factorizar  $n$ .

### Cómo funciona

- Recoger varios ciphertexts  $c_i = m^e \bmod n_i$ .
- Usar el Teorema del Resto Chino para combinar y obtener  $m^e \bmod N$  donde  $N = \prod n_i$ .
- Si  $m^e < N$ , se recupera  $m$  tomando la raíz  $e$ -ésima.

### Prevención

- Usar padding.
- No enviar el mismo mensaje sin cambios a varios receptores con exponentes bajos.

## 7 Twin Prime RSA

### Descripción

Variante donde los primos  $pp$  y  $qq$  usados para el módulo  $nn$  son primos gemelos (diferencia de 2).

### Ventajas

- Puede facilitar ciertos cálculos internos.

### Riesgos

- Podría facilitar la factorización si alguien detecta el patrón.
- No recomendado para seguridad fuerte.



## Resumen rápido

Ataque/Variante	Condición Clave	Vulnerabilidad / Aplicación
Classic RSA	N/A	Base del cifrado, seguro si parámetros correctos
Multi-Prime RSA	Módulo con $> 2$ primos	Más rápido pero puede facilitar factorización
Cube Root Attack	$e=3$ y mensaje pequeño	Recuperar mensaje con raíz cúbica sin padding
Wiener's Attack	Clave privada $dd$ muy pequeña	Recuperar $dd$ con fracciones continuas
Common Modulus Attack	Mismo $nn$ , distintos exponentes $e_1, e_2$	Recuperar mensaje combinando ciphertexts
CRT Attack	Mismo mensaje cifrado con mismo $ee$ , diferentes $n_{i_i}$	Recuperar mensaje con varios ciphertexts
Twin Prime RSA	$pp$ y $qq$ primos gemelos	Puede facilitar factorización

## Herramientas y recursos útiles

- **RsaCtfTool**: <https://github.com/Ganapati/RsaCtfTool> – Automatiza la explotación de ataques RSA.
- **Factores y cracks**: SageMath, WolframAlpha, herramientas para factorización.
- Artículos y papers sobre ataques específicos (Wiener, Common Modulus, etc).

### *Herramientas utilizadas para resolver desafíos criptográficos*

1. [Base65536](#) - Respuesta de Unicode a Base64.
2. [Traductor de Braille](#) - Traduce de braille a texto.
3. [Ciphey](#) - Herramienta para descifrar automáticamente cifrados sin conocer la clave o el cifrado, decodificar codificaciones y descifrar hashes.
4. [CyberChef](#) : una aplicación web para cifrado, codificación, compresión y análisis de datos.
5. [Cryptii](#) - Conversión, codificación y cifrado modular en línea.
6. [dCode.fr](#) - Solucionadores de Criptomonedas, Matemáticas y Codificaciones en línea.
7. [Decodificación](#) : detecte y decodifique cadenas codificadas, de forma recursiva.
8. [Máquina Enigma](#) - Simulador de Máquina Enigma Universal.
9. [FeatherDuster](#) - Una herramienta de criptoanálisis modular y automatizada.
10. [Galois](#) - Una biblioteca/kit de herramientas aritmética de campo de Galois rápida.
11. [HashExtender](#) - Herramienta para realizar ataques de extensión de longitud de hash.
12. [Hash-identifier](#): identificador de algoritmo hash simple.
13. [padding-oracle-attacker](#) : herramienta y biblioteca CLI para ejecutar ataques de oráculo de relleno fácilmente.
14. [PadBuster](#) - Script automatizado para realizar ataques de Padding Oracle.
15. [PEMCrack](#) - Descifra archivos SSL PEM que contienen claves privadas cifradas. Fuerzas brutas o grietas en el diccionario.
16. [PKCrack](#) - Cracker de cifrado PkZip.
17. [Cifrado cuadrado Polybius](#) - Tabla que permite a alguien traducir letras en números.
18. [Quipqiup](#) - Solucionador de criptogramas automatizado.
19. [RsaCtfTool](#) - Herramienta de ataques múltiples RSA.

20. [RSATool](#): herramienta para calcular el parámetro RSA y RSA-CRT.
21. [Herramientas de cifrado Rumkin](#) - Colección de herramientas de cifrado/codificadores.
22. [Vigenere Solver](#) - Herramienta en línea que rompe los cifrados de Vigenère sin conocer la clave.
23. [XOR Cracker](#) - Herramienta de descifrado XOR en línea capaz de adivinar la longitud de la clave y la clave de cifrado para descifrar cualquier archivo.
24. [XORTool](#) : una herramienta para analizar el cifrado xor multibyte.
25. [yagu](#) - Factorización automatizada de enteros.
26. [Crackstation](#) - Cracker de hashes (base de datos).
27. [Enciclopedia Online de Secuencias Enteras](#) - OEIS: La Enciclopedia On-Line de Secuencias Enteras

## Exploiting / Pwn

*Herramientas utilizadas para resolver desafíos de Pwn*

1. [afl](#) - Fuzzer orientado a la seguridad.
2. [honggfuzz](#) - Fuzzer de software orientado a la seguridad. Admite fuzzing evolutivo basado en retroalimentación basado en la cobertura de código.
3. [libformatstr](#) - Simplifica la explotación de cadenas de formato.
4. [One gadget](#) - Herramienta para encontrar un gadget RCE.
5. [Pwntools](#) - Marco CTF para escribir exploits.
6. [ROPgadget](#) - Marco para la explotación de ROP.
7. [Ropper](#) : muestre información sobre archivos en diferentes formatos de archivo y encuentre gadgets para construir cadenas rop para diferentes arquitecturas.
8. [Base de datos de shellcodes](#) - Una base de datos masiva de shellcodes.

## Forense

*Herramientas utilizadas para resolver desafíos forenses*

1. [A-Packets](#) : análisis de archivos PCAP sin esfuerzo en su navegador.
2. [Autopsy](#) - Plataforma forense digital de código abierto de extremo a extremo.
3. [Binwalk](#) - Herramienta de análisis de firmware.
4. [Bulk-extractor](#) - Herramienta de explotación forense digital de alto rendimiento.
5. [Bkhive & samdump2](#) - Descarga archivos SYSTEM y SAM.

6. [ChromeCacheView](#) : pequeña utilidad que lee la carpeta de caché del navegador web Google Chrome y muestra la lista de todos los archivos almacenados actualmente en el caché.
7. [Creddump](#) : volcar credenciales de Windows.
8. [Exiftool](#) - Leer, escribir y editar metadatos de archivos.
9. [Extundelete](#) : utilidad que puede recuperar archivos eliminados de una partición ext3 o ext4.
10. [firmware-mod-kit](#) : modifique las imágenes de firmware sin volver a compilar.
11. [Foremost](#) : programa de consola para recuperar archivos en función de sus encabezados, pies de página y estructuras de datos internas.
12. [Forensic Toolkit](#) - Escanea un disco duro en busca de información diversa. Puede, potencialmente, localizar correos electrónicos eliminados y escanear un disco en busca de cadenas de texto para usarlas como diccionario de contraseñas para descifrar el cifrado.
13. [Forensemente](#): herramienta en línea gratuita para analizar imágenes Esta herramienta tiene muchas características.
14. [MZCacheView](#) - Pequeña utilidad que lee la carpeta de caché de los navegadores web Firefox / Mozilla / Netscape y muestra la lista de todos los archivos almacenados actualmente en el caché.
15. [Herramienta de análisis forense de red NetworkMiner \(NFAT\)](#).
16. [OfflineRegistryView](#) : herramienta simple para Windows que le permite leer archivos de registro sin conexión desde una unidad externa.
17. [photorec](#) - Software de recuperación de datos de archivos diseñado para recuperar archivos perdidos, incluidos videos, documentos y archivos de discos duros, CD-ROM e imágenes perdidas (de ahí el nombre de Photo Recovery) de la memoria de la cámara digital.
18. [Visor del Registro](#) - Herramienta para ver los registros de Windows.
19. [Scalpel](#) - Herramienta de tallado de datos de código abierto.
20. [The Sleuth Kit](#) : colección de herramientas de línea de comandos y una biblioteca C que le permite analizar imágenes de disco y recuperar archivos de ellas.
21. [USBrip](#) - Herramienta forense CLI simple para rastrear artefactos de dispositivos USB (historial de eventos USB) en GNU/Linux.
22. [Volatilidad](#) : un marco forense de memoria avanzado.
  1. Puede encontrar [la hoja de trucos de volatilidad aquí](#).
23. [Wireshark](#) - Herramienta para analizar archivos pcap o pcapng.

24. [X-Ways](#) - Entorno de trabajo avanzado para examinadores forenses informáticos.

## Misc

*Herramientas utilizadas para resolver desafíos misceláneos*

1. [boofuzz](#) - Fuzzing de protocolo de red para humanos.
2. [Veles](#) - Herramienta de análisis y visualización de datos binarios.

## Bruteforcers:

1. [changeme](#) : un escáner de credenciales predeterminado.
2. [Hashcat](#) - Recuperación avanzada de contraseña.
3. [Hydra](#) : cracker de inicio de sesión paralelizado que admite numerosos protocolos para atacar.
4. [John the Ripper](#) - Auditoría de seguridad de contraseñas de código abierto y recuperación de contraseñas.
5. [jwt\\_tool](#) - Un conjunto de herramientas para probar, ajustar y descifrar tokens web JSON.
6. [Ophcrack](#) - Descifrador de contraseñas gratuito de Windows basado en tablas arcoíris.
7. [Patator](#) - Fuerza bruta multipropósito, con un diseño modular y un uso flexible.
8. [Turbo Intruder](#) - Extensión de Burp Suite para enviar un gran número de solicitudes HTTP y analizar los resultados.

## Lenguajes esotéricos:

1. [Brainfuck](#) - IDE de lenguaje de programación esotérico Brainfuck.
2. [COW](#) - Es una variante de Brainfuck diseñada con humor pensando en Bovinae.
3. [Malbolge](#) - Solucionador de lenguajes de programación esotéricos de Malbolge.
4. [¡Además!](#) - Herramienta para decodificar / codificar en Ook!
5. [Piet](#) - Compilador del lenguaje de programación Piet.
6. [Rockstar](#) - Un lenguaje destinado a parecerse a las letras de las canciones.
7. [Pruébalo en línea](#) - Una herramienta en línea que tiene un montón de intérpretes de idiomas esotéricos.



## Sandboxes:

1. [Any.run](#) : servicio interactivo de búsqueda de malware.
2. [Intezer Analyze](#) - Plataforma de análisis de malware.
3. [Triage](#) : sandbox de análisis de malware de última generación diseñado para soporte multiplataforma.

## Reversing

### *Herramientas utilizadas para resolver Reversing challenges*

1. [Androguard](#) - Androguard es una herramienta completa de Python para jugar con archivos de Android.
2. [Angr](#) - Una plataforma de análisis binario potente y fácil de usar.
3. [Apk2gold](#) - Herramienta CLI para descompilar aplicaciones de Android a Java.
4. [ApkTool](#) - Una herramienta para la ingeniería inversa de aplicaciones de Android binarias, cerradas y de 3rd party.
5. [Binary Ninja](#) - Marco de análisis binario.
6. [BinUtils](#) - Colección de herramientas binarias.
7. [CTF import](#) - Ejecute funciones básicas desde binarios eliminados multiplataforma.
8. [Explorador del compilador](#) : herramienta de compilación en línea.
9. [CWE checker](#) - Encuentra patrones vulnerables en ejecutables binarios.
10. [Demovfuscator](#) - Un desofuscador de trabajo en curso para binarios movifuscados.
11. [Disassembler.io](#) - Desmontar a pedido. Un servicio en línea liviano para cuando no tiene el tiempo, los recursos o los requisitos para usar una alternativa más pesada.
12. [dnSpy](#) : depurador y editor de ensamblados de .NET.
13. [EasyPythonDecompiler](#) - Una pequeña aplicación GUI de .exe que "descompilará" el código de bytes de Python, que a menudo se ve en la extensión .pyc.
14. [Frida](#) - Kit de herramientas de instrumentación dinámica para desarrolladores, ingenieros inversos e investigadores de seguridad.
15. [GDB](#) - El depurador del Proyecto GNU.
16. [GEF](#) - Una experiencia moderna para GDB con características avanzadas de depuración para desarrolladores de exploits e ingenieros inversos.
17. [Ghidra](#) - Un conjunto de herramientas de ingeniería inversa de software (SRE) desarrolladas por la NSA.
18. [Hopper](#) - Herramienta de ingeniería inversa (desensamblador) para OSX y Linux.
19. [IDA Pro](#) - El software de marcha atrás más utilizado.

20. [Jadx](#) - Herramientas de línea de comandos y GUI para producir código fuente Java a partir de archivos Dex y Apk de Android.
21. [Descompiladores de Java](#): un descompilador en línea para APK de Java y Android.
22. [JSDetox](#) - Una herramienta de análisis de malware de JavaScript.
23. [miasm](#) - Marco de ingeniería inversa en Python.
24. [Objeción](#) - Exploración móvil en tiempo de ejecución.
25. [Ensamblador/desensamblador en línea](#) : envoltorios en línea alrededor de los proyectos Keystone y Capstone.
26. [PEDA](#) - Asistencia para el desarrollo de exploits de Python para GDB.
27. [PEfile](#) - Módulo de Python para leer y trabajar con archivos PE (Portable Executable).
28. [Pwndbg](#) - Desarrollo de exploits e ingeniería inversa con GDB Made Easy.
29. [radare2](#) : marco de ingeniería inversa similar a UNIX y conjunto de herramientas de línea de comandos.
30. [Rizin](#) - Rizin es una bifurcación del marco de ingeniería inversa radare2 con un enfoque en la usabilidad, las características de trabajo y la limpieza del código.
31. [Uncompyle](#) - Un descompilador de código de bytes de Python 2.7 (.pyc)
32. [WinDBG](#) : depurador de Windows distribuido por Microsoft.
33. [Z3](#) - Un demostrador de teoremas de Microsoft Research.

## Steganography

### *Herramientas utilizadas para resolver desafíos de Stego*

1. [AperiSolve](#) - Plataforma que realiza análisis de capas en imágenes.
2. [BPStegano](#) - Esteganografía LSB basada en Python3.
3. [DeepSound](#) : herramienta de esteganografía gratuita y convertidor de audio que oculta datos secretos en archivos de audio.
4. [Detección DTMF](#) - Frecuencias de audio comunes a un botón de teléfono.
5. [Tonos DTMF](#) - Frecuencias de audio comunes a un botón de teléfono.
6. [Exif](#) - Muestra información EXIF en archivos JPEG.
7. [Exiv2](#) - Herramienta de manipulación de metadatos de imágenes.
8. [FotoForensics](#) - Proporciona a los investigadores en ciernes y a los investigadores profesionales acceso a herramientas de vanguardia para el análisis forense de la fotografía digital.

9. [hipshot](#) - Herramienta para convertir un archivo de video o una serie de fotografías en una sola imagen que simula una fotografía de larga exposición.
10. [Image Error Level Analyzer](#) - Herramienta para analizar imágenes digitales. También es gratuito y está basado en la web. Cuenta con análisis de nivel de error, detección de clones y más.
11. [Esteganografía de imágenes](#) : herramienta Javascript del lado del cliente para ocultar/mostrar esteganográficamente imágenes dentro de los "bits" inferiores de otras imágenes.
12. [ImageMagick](#) - Herramienta para manipular imágenes.
13. [jsteg](#) - Herramienta de línea de comandos para usar contra imágenes JPEG.
14. [Magic Eye Solver](#) - Obtén información oculta de las imágenes.
15. [Outguess](#) - Herramienta esteganográfica universal.
16. [Pngcheck](#) : verifica la integridad de PNG y vuelca toda la información de nivel de fragmento en forma legible por humanos.
17. [Pngtools](#) - Para varios análisis relacionados con PNG.
18. [sigBits](#) - Decodificador de imágenes de bits significativos de esteganografía.
19. [SmartDeblur](#) - Restauración de fotos/imágenes desenfocadas y borrosas.
20. [Snow](#) - Herramienta de esteganografía de espacios en blanco
21. [Sonic Visualizer](#) - Visualización de archivos de audio.
22. [Esteganografía Online](#) - Codificador y decodificador de esteganografía en línea.
23. [Stegbreak](#) : lanza ataques de diccionario de fuerza bruta en la imagen JPG.
24. [StegCracker](#) - Utilidad de fuerza bruta para descubrir datos ocultos dentro de los archivos.
25. [stegextract](#) - Detecta archivos ocultos y texto en imágenes.
26. [Steghide](#) - Oculta datos en varios tipos de archivos de imagen y audio.
27. [StegOnline](#) : realice una amplia gama de operaciones de esteganografía de imágenes, como ocultar/revelar archivos ocultos dentro de bits.
28. [Stegosaurus](#) - Una herramienta de esteganografía para incrustar cargas útiles dentro del código de bytes de Python.
29. [StegoVeritas](#) - Otra herramienta de stego.
30. [Stegpy](#) - Programa de esteganografía simple basado en el método LSB.
31. [stegseek](#) - Cracker de steghide ultrarrápido que se puede utilizar para extraer datos ocultos de archivos.

32. [stegsnow](#) - Programa de esteganografía de espacios en blanco.
33. [Stegsolve](#) - Aplicar varias técnicas de esteganografía a las imágenes.
34. [Zsteg](#) - Análisis PNG/BMP.

## Web

### *Herramientas utilizadas para resolver desafíos web*

1. [Arachni](#) - Marco de análisis de seguridad de aplicaciones web.
2. [Beautifier.io](#) - Embellecedor de JavaScript en línea.
3. [BurpSuite](#) - Una herramienta gráfica para probar la seguridad del sitio web.
4. [Commix](#) : herramienta automatizada de explotación de inyección de comandos del sistema operativo todo en uno.
5. [debugHunter](#) : descubra parámetros de depuración ocultos y descubra secretos de aplicaciones web.
6. [Dirhunt](#) - Encuentra directorios web sin fuerza bruta.
7. [dirsearch](#) : escáner de ruta web.
8. [nomore403](#) - Herramienta para evitar errores 40x.
9. [ffuf](#) - Fuzzer web rápido escrito en Go.
10. [git-dumper](#) - Una herramienta para volcar un repositorio de git de un sitio web.
11. [Gopherus](#) - Herramienta que genera enlace gopher para explotar SSRF y obtener RCE en varios servidores.
12. [Hookbin](#) : servicio gratuito que le permite recopilar, analizar y ver solicitudes HTTP.
13. [JSFiddle](#) : pruebe su JavaScript, CSS, HTML o CoffeeScript en línea con el editor de código JSFiddle.
14. [ngrok](#) : túneles introspectivos seguros en localhost.
15. [OWASP Zap](#) : intercepta el proxy para reproducir, depurar y eliminar solicitudes y respuestas HTTP.
16. [PHPGGC](#) - Biblioteca de cargas útiles de PHP unserialize() junto con una herramienta para generarlas, desde la línea de comandos o mediante programación.
17. [Postman](#) - Complemento para Chrome para depurar solicitudes de red.
18. [REQBIN](#) - Herramienta de prueba de API REST & SOAP en línea.
19. [Contenedor de](#) solicitudes: un contenedor de solicitudes moderno para inspeccionar cualquier evento de Pipedream.
20. [Revelo](#) - Analiza el código Javascript ofuscado.
21. [Smuggler](#) - Una herramienta de prueba de contrabando / desincronización de solicitudes HTTP escrita en Python3.
22. [SQLMap](#) - Herramienta automática de inyección SQL y toma de control de bases de datos.
23. [W3af](#) - Marco de auditoría y ataque de aplicaciones web.

24. [XSSer](#) - Probador XSS automatizado.
25. [ysoserial](#) - Herramienta para generar cargas útiles que explotan la deserialización de objetos Java inseguros.

## OSINT

*Open Source Intelligence Herramientas para recopilar información de fuentes públicas*

1. [haveibeenpwned](#) : verifique si su correo electrónico o nombre de usuario ha estado en una violación de datos.
2. [snusbase](#) - Busca en bases de datos filtradas correos electrónicos, nombres de usuario y contraseñas.
3. [dehashed](#) - Motor de búsqueda de bases de datos hackeadas.
4. [leakcheck](#) : compruebe si hay credenciales filtradas.
5. [Shodan](#) : el primer motor de búsqueda del mundo para dispositivos conectados a Internet.
6. [BinaryEdge](#) - Plataforma para escanear, adquirir y clasificar datos públicos de Internet.
7. [whopostedwhat](#) : busque palabras clave y encuentre cuentas que publiquen sobre un tema o evento.
8. [Graph.tips](#) : busque datos públicos de Facebook de formas específicas.
9. [búsqueda instantánea de nombres de usuario](#) - Verifique la disponibilidad del nombre de usuario en 100+ sitios de redes sociales.
10. [ExifTool](#) - Herramienta de línea de comandos para leer/escribir metainformación en archivos.
11. [WiGLE](#) : recopila información sobre puntos de acceso inalámbricos en todo el mundo.
12. [Creepy](#) - Herramienta OSINT de geolocalización para recopilar información de las redes sociales.
13. [Social Mapper](#) - Utiliza el reconocimiento facial para correlacionar los perfiles de las redes sociales.
14. [theHarvester](#) : recopila correos electrónicos, nombres, subdominios, IP y URL de fuentes públicas.
15. [TinEye](#) - Motor de búsqueda inversa de imágenes.
16. [YandexImages](#) - Motor de búsqueda inversa de imágenes.

## Privilege Escalation

*Herramientas y recursos para escalar privilegios en Linux y Windows*

1. [Basic Linux Privilege Escalation](#) - Guía y script fundamentales.
2. [Fundamentos de la escalada de privilegios de Windows](#) : guía y herramienta para Windows.

3. [linux-smart-enumeration](#) : script para mostrar información de seguridad relevante en Linux.
4. [linux-exploit-suggester](#) : detecta deficiencias de seguridad para el kernel o las máquinas de Linux.
5. [LinEnum](#) - Comprobaciones locales de enumeración y escalada de privilegios de Linux con scripts.
6. [PEASS](#) - Escalada de privilegios Awesome Scripts SUITE.
7. [go awayBins](#) - Lista seleccionada de binarios de Unix que se pueden explotar para eludir las restricciones de seguridad locales.
8. [PowerUp](#) : vectores comunes de escalada de privilegios de Windows.
9. [JAWS](#) : script para identificar rápidamente posibles vectores de escalada de privilegios en Windows.

## Hash Cracking

*Herramientas para identificar y descifrar hashes de contraseñas*

1. [MD5Hashing.net](#) - Cracker y herramientas de hash en línea.
2. [CyberChef](#) : aplicación web para cifrado, codificación, compresión y análisis de datos.
3. [hash-identifier](#) : identifica diferentes tipos de hashes.
4. [hashID](#) : identifica los tipos de hash.
5. [HashTag](#) : analice e identifique hashes de contraseñas.
6. [hashcat](#) - Utilidad de recuperación de contraseña rápida y avanzada.
7. [John the Ripper](#) - Descifrador rápido de contraseñas.
8. [HashPump](#) : herramienta para explotar los ataques de extensión de longitud de hash.

## Utilidades de red

*Herramientas esenciales para el análisis y la explotación de redes*

1. [nmap](#) : descubra hosts y servicios en una red enviando paquetes y analizando respuestas.
2. [Netcat](#) - Utilidad de red para leer/escribir en conexiones de red usando TCP o UDP.
3. [wireshark](#) - Analizador de protocolos de red para inspeccionar el tráfico de red registrado.