Level#01 → Level#02

flag: Santa / XMAS2026

Nos dan una URL donde en el source solo aparece una referencia a un jar

```
<!doctype html>
<html lang=es>
 <head>
  <meta charset=utf-8>
  <title>2026 XMAS CTF by s4ur0n (CS3 Group)</title>
  <style>
   body {
    background-image: url('level1.png');
    background-repeat: no-repeat;
    background-attachment: fixed;
    background-size: 100% 100%;
  }
  </style>
 </head>
 <body>
  <!-- Ciphertext messages: java -jar xmas2026.jar -->
 </body>
</html>
```

Enumeramos y no vemos nada

```
❯ gobuster dir -u https://xmas2026.s4ur0n.com/ -x jar -w /home/malekith/
Dropbox/CTFs/SecLists/Discovery/Web-Content/big.txt
===============================================================
Gobuster v3.8.2
by OJ Reeves (@TheColonial) & Christian Mehlmauer (@firefart)
===============================================================
[+] Url:                     https://xmas2026.s4ur0n.com/
[+] Method:                  GET
[+] Threads:                 10
[+] Wordlist:                /home/malekith/Dropbox/CTFs/SecLists/Discovery/W
eb-Content/big.txt
[+] Negative Status codes:   404
[+] User Agent:              gobuster/3.8.2
[+] Extensions:              jar
[+] Timeout:                 10s
===============================================================
Starting gobuster in directory enumeration mode
===============================================================
.htaccess.jar       (Status: 403) [Size: 199]
.htaccess           (Status: 403) [Size: 199]
.htpasswd           (Status: 403) [Size: 199]
.htpasswd.jar       (Status: 403) [Size: 199]
about               (Status: 301) [Size: 242] [--> https://xmas2026.s4ur0n.com/
about/]
favicon.ico         (Status: 200) [Size: 15406]
level2              (Status: 401) [Size: 325]
server-status       (Status: 403) [Size: 199]
Progress: 40962 / 40962 (100.00%)
===============================================================
Finished
===============================================================
```

```
~/Documents/CTFs via 🐍 v3.12.3 (ctf-venv) took 4m20s
```

Pero nos bajamos el png del fondo y ocupa demasiado

```
❯ wget https://xmas2026.s4ur0n.com/level1.png
--2026-01-03 00:09:47--  https://xmas2026.s4ur0n.com/level1.png
Resolving xmas2026.s4ur0n.com (xmas2026.s4ur0n.com)... 135.181.196.24
7
Connecting to xmas2026.s4ur0n.com (xmas2026.s4ur0n.com)|135.181.196.
247|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 3274213 (3,1M) [image/png]
Saving to: ‘level1.png’

level1.png                  100%[=============================
====================⇒]   3,12M  2,63MB/s    in 1,2s

2026-01-03 00:09:49 (2,63 MB/s) - ‘level1.png’ saved [3274213/3274213]
```

Lanzamos un zsteg -a y vemos un jar

```
❯ zsteg -a level1.png | more
[?] 2672 bytes of extra data after image end (IEND), offset = 0x31eb75

   00000000: 50 4b 03 04 0a 00 00 08  00 00 20 95 94 5b 00 00  |P
K........ ..[..|
   00000010: 00 00 00 00 00 00 00 00  00 00 09 00 04 00 4d 45
|..............ME|
   00000020: 54 41 2d 49 4e 46 2f fe  ca 00 00 50 4b 03 04 14  |TA-IN
F/....PK...|
   00000030: 00 08 08 08 00 20 95 94  5b 00 00 00 00 00 00 00  |..... ..
[.......|
   00000040: 00 00 00 00 00 14 00 00  00 4d 45 54 41 2d 49 4e  |.........M
ETA-IN|
   00000050: 46 2f 4d 41 4e 49 46 45  53 54 2e 4d 46 f3 4d cc  |F/MANIF
EST.MF.M.|
   00000060: cb 4c 4b 2d 2e d1 0d 4b  2d 2a ce cc cf b3 52 30  |.LK-...K-
```

```
*....R0|
   00000070: d4 33 e0 e5 72 ce 49 2c  2e d6 0d 48 2c c9 b0 52  |.3..r.I,...
H,..R|
   00000080: d0 e3 e5 f2 4d cc cc d3  05 8b 59 29 04 27 e6 95  |....M.....
Y).'..|
   00000090: 24 02 55 14 a5 26 96 a4  a6 e8 26 55 5a 29 14 9b  |$.U..&....
&UZ)..|
   000000a0: 94 16 19 e4 f1 72 f1 72  01 00 50 4b 07 08 11 81  |.....r.r..PK....|
   000000b0: 6b 21 4d 00 00 00 4f 00  00 00 50 4b 03 04 14 00  |k!M...
O...PK....|
   000000c0: 08 08 08 00 f1 94 94 5b  00 00 00 00 00 00 00 00  |.......
[........|
   000000d0: 00 00 00 00 0b 00 00 00  53 61 6e 74 61 2e 63 6c  |........San
ta.cl|
   000000e0: 61 73 73 8d 56 5b 70 13  d7 19 fe d6 5a 69 65 b1  |ass.V[p.....
Zie.|
   000000f0: f8 22 23 83 b1 ec c8 84  10 db b2 65 63 c0 01 9b  |."#........e
c...|
```

Lo extraemos con binwalk

```
❯ binwalk -c level1.png

                /home/malekith/Documents/CTFs/xmas2026 s4ur0n/Level0
1/extractions/level1.png
----------------------------------------------------------------------------
-------------------------------------------------
DECIMAL                   HEXADECIMAL               DESCRIPTION
----------------------------------------------------------------------------
-------------------------------------------------
0                         0x0                       PNG image, total size: 3271541
bytes
3271541                   0x31EB75                  ZIP archive, version: 1.0,
file count: 3, total size:
                                                    2672 bytes
----------------------------------------------------------------------------
-------------------------------------------------
```

```
Analyzed 1 file for 111 file signatures (251 magic patterns) in 8.0 millisecond
s
```

Nos movemos al directorio donde lo ha descargado binwalk y lanzamos un file y vemos nuestro JAR

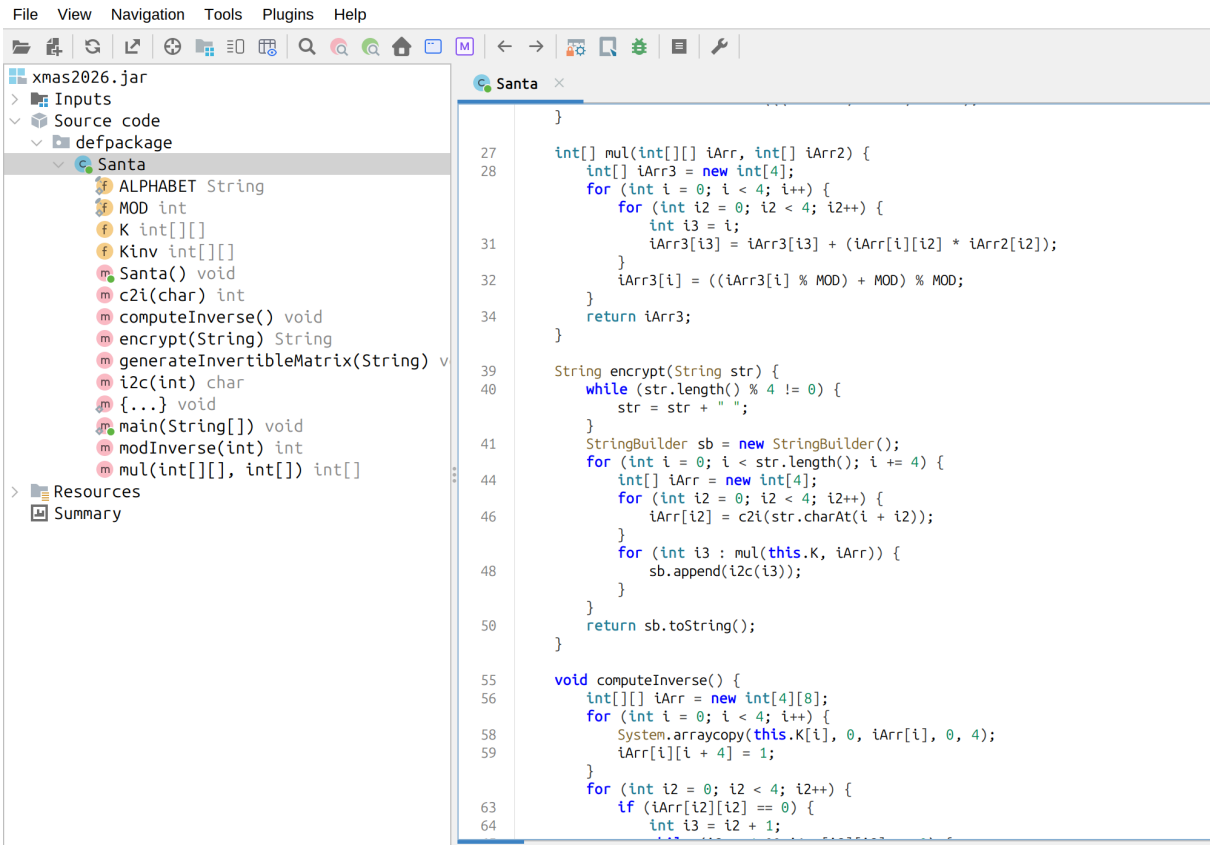
```
❯ file *
level1.png:                symbolic link to /home/malekith/Documents/CTFs/xma
s2026 s4ur0n/Level01/level1.png
level1.png_0_png.raw:      PNG image data, 1792 × 1024, 8-bit/color RGBA,
non-interlaced
level1.png_3271541_zip.raw: Java archive data (JAR)
```

Lo renombramos como xmas2026.jar

y al ejecutarlo nos da una semilla y un texto cifrado

```
❯ ../../jdk-24.0.2/bin/java -jar xmas2026.jar

Message:
Welcome to the 2026 XMAS CTF by s4ur0n!

Seed:
XMASCTF2026

Generating an invertible matrix using the seed

Encrypted text:
WsNjkGTKy7DJgQ8qAZt35jIatmyj7RvegrO2BvZ!
```

Usamos jadx para ver el fuente



Es básicamente un **cifrado tipo Hill** (álgebra lineal modular) con bloques de 4 caracteres, usando un **alfabeto fijo** y una **matriz clave 4×4** generada de forma pseudoaleatoria a partir de una semilla.

Este cifrado se puede romper dado que conocemos la semilla, un texto conocido, el texto cifrado y el cifrado desconocido

Nos podemos montar el siguiente script

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

ALPHABET = "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstu
vwxyz0123456789 !:/."
MOD = len(ALPHABET)  # 67

def c2i(c: str) -> int:
    i = ALPHABET.find(c)
    if i == -1:
        raise ValueError(f"Invalid char: [{c}]")
    return i
```

```
def i2c(i: int) → str:
    return ALPHABET[((i % MOD) + MOD) % MOD]

def java_string_hashcode(s: str) → int:
    h = 0
    for ch in s:
        h = (31 * h + ord(ch)) & 0xFFFFFFFF
    # to signed 32-bit
    if h & 0x80000000:
        h = -((~h + 1) & 0xFFFFFFFF)
    return h

class JavaRandom:
    multiplier = 0x5DEECE66D
    addend = 0xB
    mask = (1 << 48) - 1

    def __init__(self, seed: int):
        # Java: (seed ^ multiplier) & mask
        self.seed = (seed ^ self.multiplier) & self.mask

    def next(self, bits: int) → int:
        self.seed = (self.seed * self.multiplier + self.addend) & self.mask
        return self.seed >> (48 - bits)

    def nextInt(self, bound: int) → int:
        if bound <= 0:
            raise ValueError("bound must be positive")
        # power of two fast-path
        if (bound & (bound - 1)) == 0:
            return (bound * self.next(31)) >> 31
        while True:
            bits = self.next(31)
            val = bits % bound
            if bits - val + (bound - 1) >= 0:
                return val
```

```
def mul_mat_vec(K, v):
    out = []
    for i in range(4):
        s = 0
        for j in range(4):
            s += K[i][j] * v[j]
        out.append(s % MOD)
    return out

def invert_matrix_mod_prime(K, p):
    # Gauss-Jordan mod prime p
    n = 4
    A = [[K[i][j] % p for j in range(n)] + [1 if i == j else 0 for j in range(n)] for i
in range(n)]
    for i in range(n):
        # find pivot
        pivot = i
        while pivot < n and A[pivot][i] % p == 0:
            pivot += 1
        if pivot == n:
            raise ValueError("Non-invertible")
        if pivot != i:
            A[i], A[pivot] = A[pivot], A[i]

        inv = pow(A[i][i], -1, p)
        A[i] = [(x * inv) % p for x in A[i]]

        for r in range(n):
            if r == i:
                continue
            factor = A[r][i] % p
            if factor:
                A[r] = [(A[r][c] - factor * A[i][c]) % p for c in range(2 * n)]

    return [row[n:] for row in A]

def det_mod_prime(K, p):
    # elimination determinant
```

```
    import copy
    A = [row[:] for row in K]
    det = 1
    n = 4
    for i in range(n):
        pivot = None
        for r in range(i, n):
            if A[r][i] % p != 0:
                pivot = r
                break
        if pivot is None:
            return 0
        if pivot != i:
            A[i], A[pivot] = A[pivot], A[i]
            det = (-det) % p
        det = (det * (A[i][i] % p)) % p
        inv = pow(A[i][i] % p, -1, p)
        for c in range(i, n):
            A[i][c] = (A[i][c] * inv) % p
        for r in range(i + 1, n):
            factor = A[r][i] % p
            if factor:
                for c in range(i, n):
                    A[r][c] = (A[r][c] - factor * A[i][c]) % p
    return det % p

def generate_K_from_seed(seed_str: str):
    jr = JavaRandom(java_string_hashcode(seed_str))
    while True:
        K = [[jr.nextInt(MOD - 1) + 1 for _ in range(4)] for _ in range(4)]
        if det_mod_prime(K, MOD) != 0:
            return K

def decrypt(ct: str, seed_str="XMASCTF2026") -> str:
    # if user pasted an extra trailing space, drop only one to recover multiple
-of-4
    if len(ct) % 4 != 0 and ct.endswith(" "):
        ct = ct[:-1]
```

```
    if len(ct) % 4 != 0:
        raise ValueError("Ciphertext length must be multiple of 4 (check copy/
paste).")

    K = generate_K_from_seed(seed_str)
    Kinv = invert_matrix_mod_prime(K, MOD)

    out = []
    for i in range(0, len(ct), 4):
        v = [c2i(ct[i + j]) for j in range(4)]
        p = mul_mat_vec(Kinv, v)
        out.extend(i2c(x) for x in p)
    return "".join(out)

if __name__ == "__main__":
    ct = "Ko0n7zuiRF5KcReqKQUYe8XvjsUmWbJ3nzOumU8ghG0E3qywHW
jiBXxvGnAY1!KMLZ9qIYc aOV4MX3IAZt3 "
    print(decrypt(ct))
```

Lo ejecutamos y nos da:

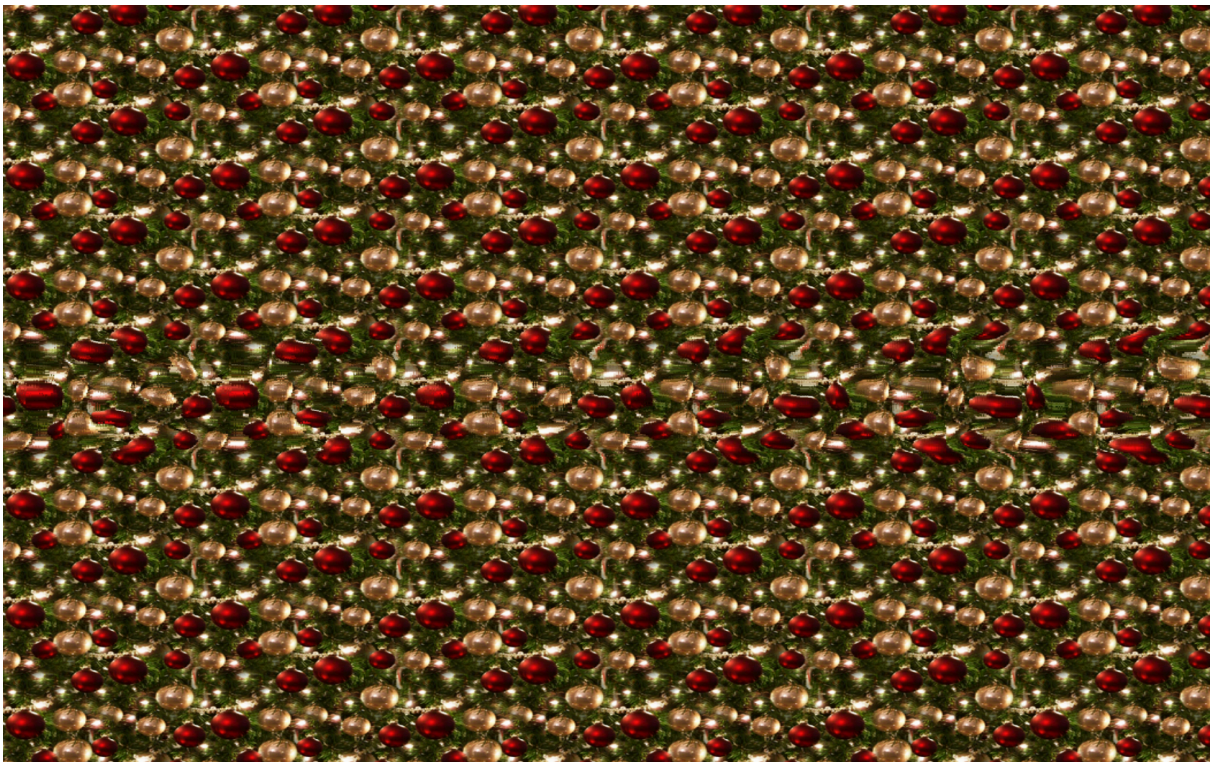
```
❯ python3 decoder.py
Well done! Try next levenons https://xmas2026.s4ur0n.com/level2 u4gi Sa
nta/XMAS2026
```

Con eso nos autenticamos en https://xmas2026.s4ur0n.com/level2 para acceder como Santa y pwd XMAS2026

Level#02 → Level#03

flag: Grinch / XmAs-2026

#magic #eye



Usando las credenciales del Level#01 → Level#02 nos logamos y solo vemos un png, asi que lo descargamos,

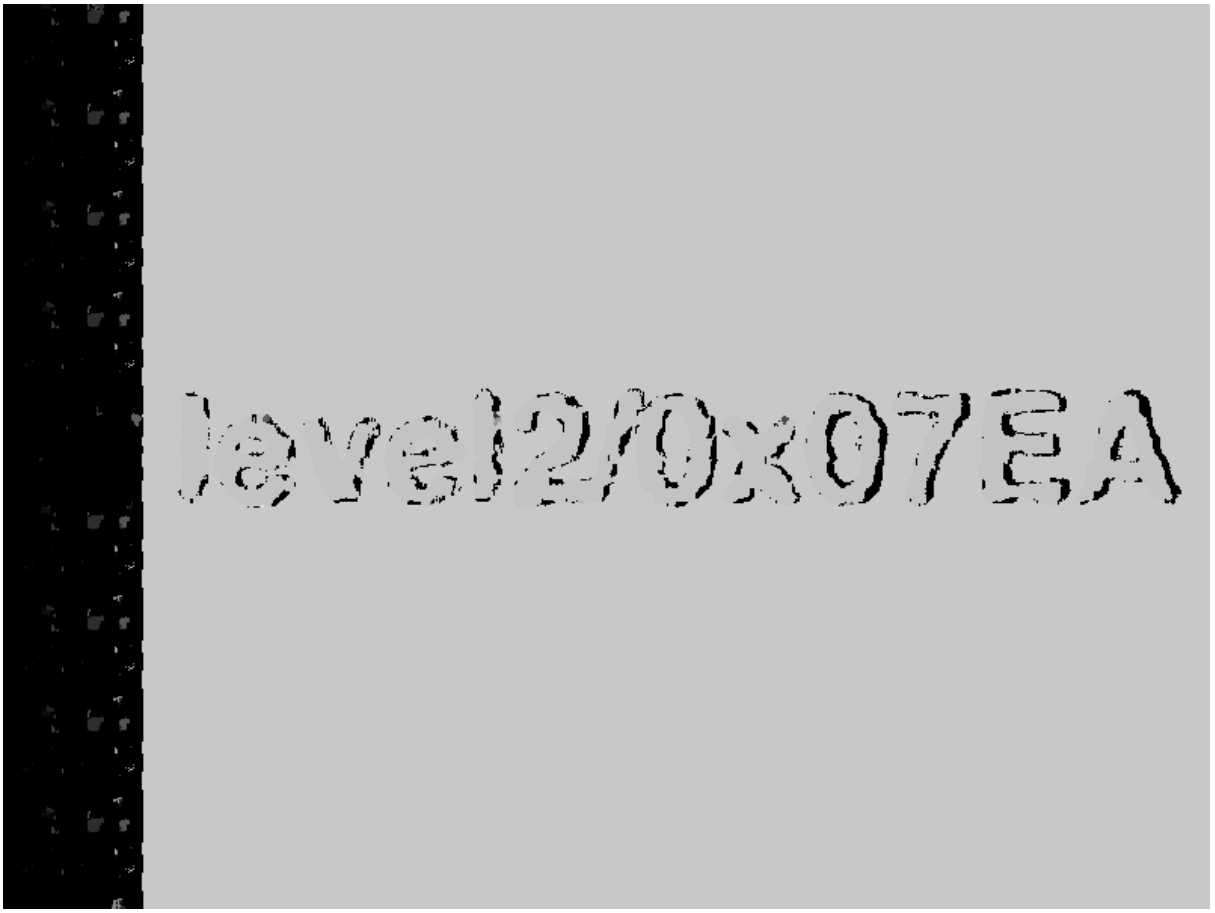
Lanzamos lo típico, strings, zsteg y exiftool, pero no saca nada de nada, en éste caso fue mi hijo el que me dijo, “papa veo unas letras”, asi que investigando parece que se trata de **un estereograma (Magic-Eye)**. No hay “LSB” ni píxeles raros: el texto aparece **en profundidad** cuando desenfocas la vista.

Para extraer la imagen podemos usar éste script

```
import cv2
import numpy as np

img = cv2.imread("level2.png", cv2.IMREAD_GRAYSCALE)
img = cv2.resize(img, (800, 600), interpolation=cv2.INTER_AREA)
```

```
h, w = img.shape
maxd = 120
ksize = 9

best_cost = np.full((h, w), 1e9, dtype=np.float32)
best_d = np.zeros((h, w), dtype=np.uint8)

for d in range(1, maxd + 1):
    diff = cv2.absdiff(img[:, d:], img[:, :-d]).astype(np.float32)
    diff = np.pad(diff, ((0, 0), (d, 0)), mode="constant", constant_values=25
5)
    cost = cv2.boxFilter(diff, ddepth=-1, ksize=(ksize, ksize), normalize=Fals
e)

    mask = cost < best_cost
    best_cost[mask] = cost[mask]
    best_d[mask] = d

disp = cv2.normalize(best_d, None, 0, 255, cv2.NORM_MINMAX)
cv2.imwrite("depth.png", disp)
print("[+] saved depth.png")
```

y nos da otro png con un texto



que pone level2/0x07EA ponemos esa ruta y nos manda al level3

Level#03 → Level#04

flag: SANTACLAUS / Hohoho-2026

Usando las credenciales encontradas en Level#03 → Level#04 entramos y nos da un emulador en JS del popular juego outrun

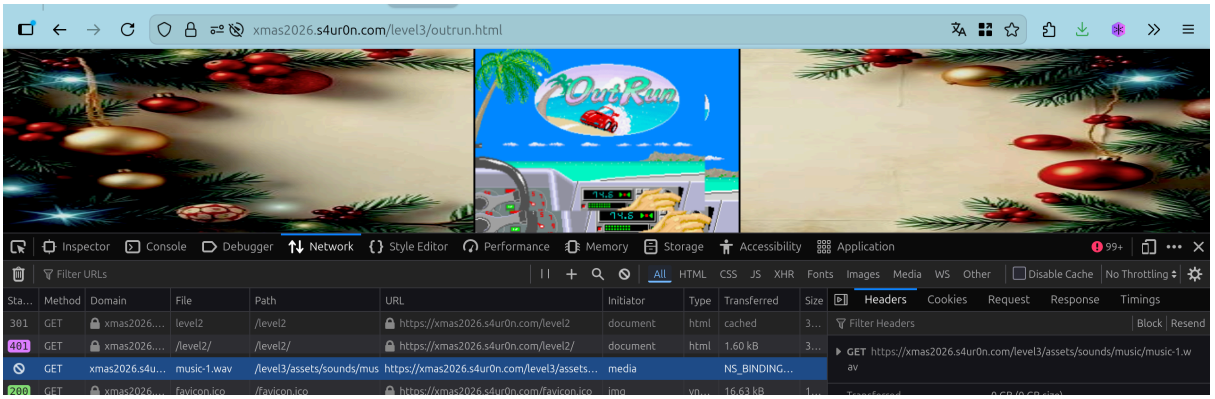
Asi que toca mirar el fuente

```
<!doctype html>
<html lang=es>

<head>
  <title>2026 XMAS CTF by s4ur0n (CS3 Group)</title>
  <meta charset=utf-8>
</head>

<body>
  <link rel="stylesheet" href="style.css">
```

```
    <canvas id="screen"></canvas>
    <script src="js/EventListener.js"></script>
    <script src="js/Vector3.js"></script>
    <script src="js/Canvas.js"></script>
    <script src="js/Assets.js"></script>
    <script src="js/Segment.js"></script>
    <script src="js/Camera.js"></script>
    <script src="js/WorldObject.js"></script>
    <script src="js/Player.js"></script>
    <script src="js/Game.js"></script>
    <script src="js/GameWorld.js"></script>
    <script src="js/Road.js"></script>
    <script src="js/Tile.js"></script>
    <script src="js/Junction.js"></script>
    <script src="js/AudioManager.js"></script>
    <script src="js/Vehicle.js"></script>
    <script>
        Outrun.start();
    </script>
</body>

</html>
```

Vemos que todos los JS están ofuscados, asi que primero pasamos por https://beautifier.io/ para dejarlos un poco mas leibles, dado que están bastante ofuscados se lo pasé a distintos motores de IA, pero la verdad que fue bastante desastroso, asi que al volver, si prestamos atencion vemos que aparece una pantalla en la que pide usar un audio, pero desaparece y aparece la parte de carga

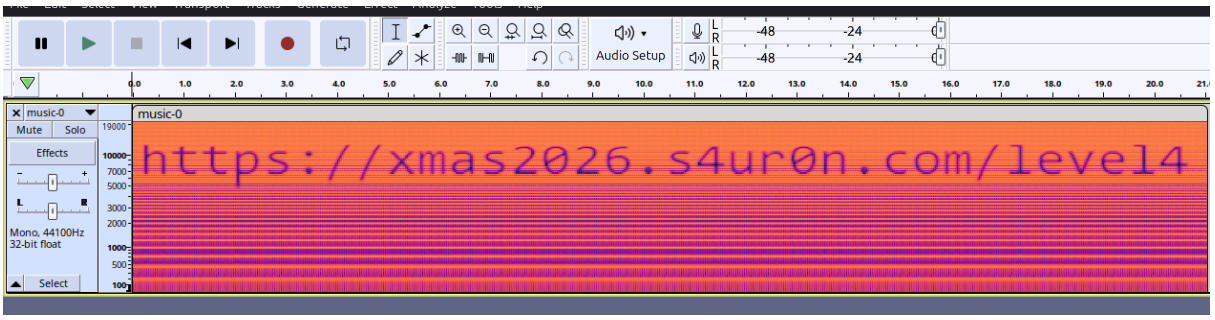
Asi que tiramos de las dev tools y vemos el link de la musica



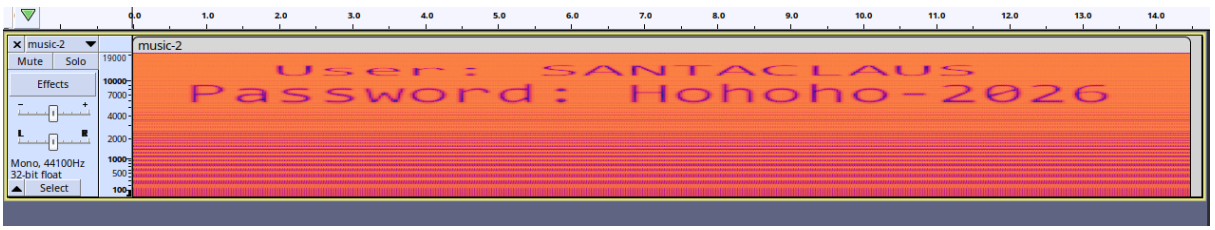
Nos bajamos los audios

```
❯ file *
music-0.wav: RIFF (little-endian) data, WAVE audio, Microsoft PCM, 16 bit, mono 44100 Hz
music-1.wav: RIFF (little-endian) data, WAVE audio, Microsoft PCM, 16 bit, stereo 44100 Hz
music-2.wav: RIFF (little-endian) data, WAVE audio, Microsoft PCM, 16 bit, mono 44100 Hz
```

Pensando que puede ser stego de audio, los cargamos en audacity y al poner el espectrograma del primero nos saca un texto



Hacemos lo mismo con los otros dos y en el último archivo nos encontramos las credenciales



Level#04 → Level#05

flag: ELFOS / EasyCTF2026

Usando las credenciales encontradas Level#03 → Level#04 vemos que nos podemos bajar un wav y por la imagen tiene que ser algo de un modem

Si lo reproducimos, efectivamente, se trata del sonido típico de un modem

Tiramos de minimodem

```
❯ minimodem --rx -f xms2026.wav 300
### CARRIER 300 @ 1250.0 Hz ###
begin 644 xmas2026.rom
M04(00````````````````"$A0,T80!C^?J?(S:(`(QCW1&P@<6P@:6)S8FDU
L('AK82!R<&(@0DE#3%`@=&9Q92!">'!V6E%#,C`R-B!X<"!M>'!P=&QO80``
`
end

### NOCARRIER ndata=153 confidence=2.296 ampl=0.993 bps=300.00 (r
ate perfect) ###
```

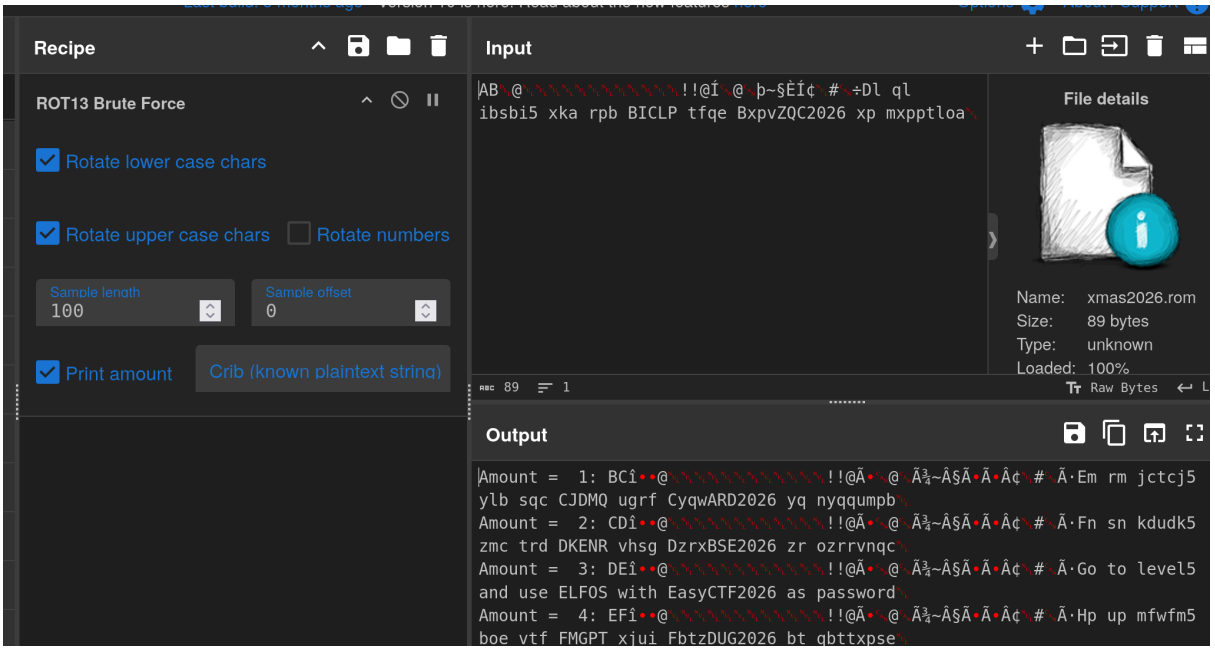
Vemos que se genera una rom uuencodeada (begin 644 xmas2026.rom), asi que lo ejecutamos con minimodem otra vez pero pasando a uuencode

```
❯ minimodem --rx -f xms2026.wav 300 | sed -n '/^begin /,/^end$/p' | uude
code

### CARRIER 300 @ 1250.0 Hz ###

### NOCARRIER ndata=153 confidence=2.296 ampl=0.993 bps=300.00 (r
ate perfect) ###
```

Eso nos saca la rom

```
❯ file xmas2026.rom
xmas2026.rom: MSX ROM, init=0x4010

CTFs/xmas2026 s4ur0n/level4 via 🐍 v3.12.3 (ctf-venv)
❯ cat xmas2026.rom
AB@!!@@~ #Dl ql ibsbi5 xka rpb BICLP tfqe BxpvZQC2026 xp
mxpptloa%
```

y vemos un texto parece ROT, lo cargamos en ciberchef



y nos da las credenciales para entrar en el siguiente nivel

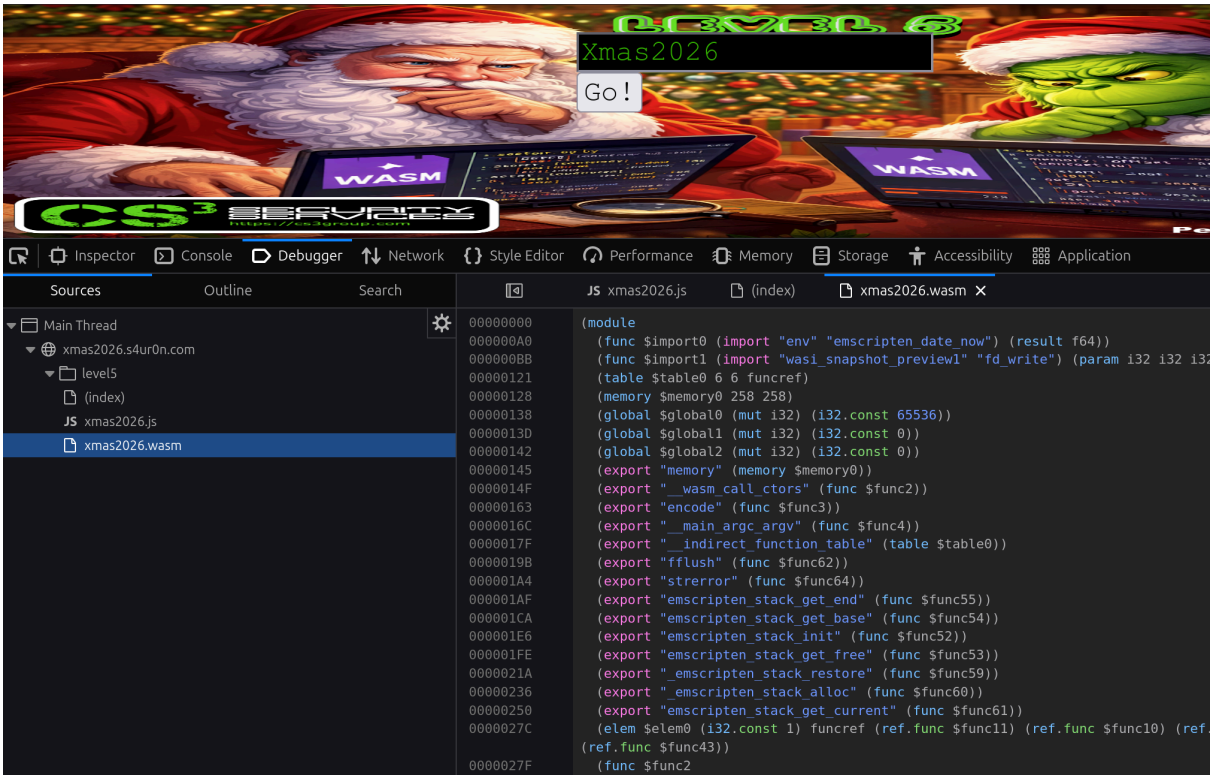
Level#05 → Level#06

flag: Rudolph / R3n0B3rnard0

#wasm

Usando las credenciales que hemos obtenido en Level#04 → Level#05

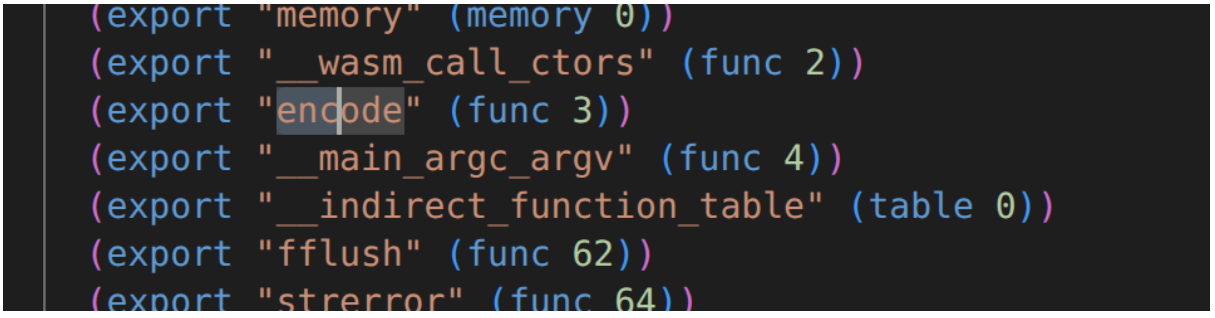
Por la pista es WASM, entramos en las developer tools y lo validamos



nos bajamos el wasm y lo convertimos a algo mas legible

```
❯ wasm2wat xmas2026.wasm -o xmas2026.wat
```

Cargamos el wat en VScode y vemos una función de encode



En ésta parte del código valida que el primer caracter debe de ser una letra

```
local.get 1
i32.load offset=72
local.get 1
i32.load offset=68
i32.add
```

```
i32.load8_u
local.set 2
i32.const 24
local.set 3
local.get 2
local.get 3
i32.shl
local.get 3
i32.shr_s
call 6                  ;; Verifica si es letra
br_if 2 (;@3;)
```

Luego se valida que la longitud es <15

```
local.get 1
i32.load offset=72
call 16                 ;; strlen
i32.const 0
i32.gt_u                ;; > 0
i32.const 1
i32.and
i32.eqz
br_if 0 (;@2;)
local.get 1
i32.load offset=72
call 16
i32.const 15
i32.lt_u                ;; < 15
```

Luego vemos 2 validaciones de posiciones pares e impares

```
;; SUMA POSICIONES IMPARES (debe ser 698)
local.get 1
i32.load8_u offset=49    ;; pos 0
...
local.get 1
i32.load8_u offset=61    ;; pos 12
i32.add
```

```
i32.const 698
i32.eq              ;; Compara con 698

;; SUMA POSICIONES PARES (debe ser 612)
local.get 1
i32.load8_u offset=50    ;; pos 1
...
local.get 1
i32.load8_u offset=62    ;; pos 13
i32.add
i32.const 612
i32.eq              ;; Compara con 612
```

En este punto, al intentar crear el script me encontré con un problemón, que se generaban millones de valores posibles.. asi que después de pedirle una pista a S4ur0n publicó esto en X.com, que contenía la parte que me faltaba

Nos sacamos todas las palabras de fail nos hacemos el siguiente script

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

import string
import itertools
import time
import random
import requests

# ==========================
# Target + Auth
# ==========================
BASE = "https://xmas2026.s4ur0n.com/level5/?id="
AUTH_USER = "ELFOS"
AUTH_PASS = "EasyCTF2026"

# ==========================
# Constraints
# ==========================
N = 14
SUM_EVEN = 698
SUM_ODD  = 612

ALNUM = string.ascii_letters + string.digits
LETTERS = string.ascii_letters

# ==========================
# Rate-limit controls
# ==========================
JITTER_MIN = 0.05
JITTER_MAX = 0.20

MAX_RETRIES = 6
BACKOFF_BASE = 0.6
BACKOFF_CAP  = 12.0

COOLDOWN_AFTER = 3
COOLDOWN_SECS  = 20.0
```

```
PRINT_EVERY = 1

# ==========================
# WASM transform (id)
# ==========================
def wasm_id_hex(pw: str) -> str:
    b = pw.encode("latin-1", errors="strict")
    out = []
    for i, ch in enumerate(b):
        out.append(ch ^ (0xAB if (i % 2 == 0) else 0xBA))
    return "".join(f"{x:02X}" for x in out)

# ==========================
# Rate-limit safe GET
# ==========================
def safe_get(sess: requests.Session, url: str, timeout=20):
    consecutive_bad = getattr(safe_get, "_consecutive_bad", 0)
    last_exc = None

    for attempt in range(1, MAX_RETRIES + 1):
        time.sleep(random.uniform(JITTER_MIN, JITTER_MAX))
        try:
            r = sess.get(url, allow_redirects=True, timeout=timeout)

            if r.status_code not in (429, 503, 502, 504):
                safe_get._consecutive_bad = 0
                return r, attempt, None

            consecutive_bad += 1
            safe_get._consecutive_bad = consecutive_bad

            if consecutive_bad >= COOLDOWN_AFTER:
                time.sleep(COOLDOWN_SECS)
                consecutive_bad = 0
                safe_get._consecutive_bad = 0

            backoff = min(BACKOFF_CAP, BACKOFF_BASE * (2 ** (attempt - 1)))
```

```
            time.sleep(backoff)

        except (requests.Timeout, requests.ConnectionError) as e:
            last_exc = e
            consecutive_bad += 1
            safe_get._consecutive_bad = consecutive_bad
            backoff = min(BACKOFF_CAP, BACKOFF_BASE * (2 ** (attempt - 1)))
            time.sleep(backoff)

    return None, MAX_RETRIES, last_exc

# ==========================
# Candidate generation (STRICT)
# ==========================
LEET = {
    "a": ["a", "4"],
    "e": ["e", "3"],
    "i": ["i", "1"],
    "o": ["o", "0"],
    "s": ["s", "5"],
}

def leet_variants(word):
    pools = []
    for ch in word:
        pools.append(LEET.get(ch.lower(), [ch]))
    for combo in itertools.product(*pools):
        yield "".join(combo)

def generate_candidates():
    WORD1 = ["reversing", "Reversing"]
    MID   = ["is", "Is"]
    WORD2 = ["fun", "Fun", "xmas", "Xmas"]

    seen = set()

    for w1, mid, w2 in itertools.product(WORD1, MID, WORD2):
        base = w1 + mid + w2
```

```
        if len(base) != 14:
            continue

        for v1 in leet_variants(w1):
            for v2 in leet_variants(mid):
                for v3 in leet_variants(w2):
                    pw = v1 + v2 + v3
                    if len(pw) != 14:
                        continue
                    if not pw.isalnum():
                        continue
                    if pw[0] not in LETTERS:
                        continue
                    if pw in seen:
                        continue
                    seen.add(pw)
                    yield pw

# ==========================
# Main
# ==========================
def main():
    requests_tried = 0

    with requests.Session() as sess:
        sess.auth = (AUTH_USER, AUTH_PASS)
        sess.headers["User-Agent"] = "Mozilla/5.0 (X11; Linux x86_64)"

        for pw in generate_candidates():
            # sumas
            se = sum(ord(pw[i]) for i in range(0, N, 2))
            so = sum(ord(pw[i]) for i in range(1, N, 2))
            if se != SUM_EVEN or so != SUM_ODD:
                continue

            hid = wasm_id_hex(pw)
            url = BASE + hid
```

```
            r, attempts, exc = safe_get(sess, url, timeout=20)
            requests_tried += 1

            if r is None:
                print(f"[{requests_tried}] PW={pw} ID={hid} → FAILED {exc}")
                continue

            final = r.url
            print(f"[{requests_tried}] PW={pw} ID={hid} → {r.status_code} {fina
l}")

            if not final.rstrip("/").endswith("/fail") and r.status_code != 503:
                print("\n[!] HIT DEFINITIVO\n")
                print("PASSWORD :", pw)
                print("ID_HEX   :", hid)
                print("FINAL_URL:", final)
                print("STATUS   :", r.status_code)
                print("\nBODY:\n")
                print(r.text)
                return

    print("[!] No hit.")

if __name__ == "__main__":
    main()
```

Lo lanzamos y bingo!

```
❯ python3 brute_xmas.py
[1] PW=R3v3rs1ngisFun ID=F989DD89D9C99AD4CCD3D8FCDED4 → 200
https://xmas2026.s4ur0n.com/level5/F989DD89D9C99AD4CCD3D8FCDE
D4/

[!] HIT DEFINITIVO

PASSWORD : R3v3rs1ngisFun
ID_HEX   : F989DD89D9C99AD4CCD3D8FCDED4
FINAL_URL: https://xmas2026.s4ur0n.com/level5/F989DD89D9C99AD4CC
```

```
D3D8FCDED4/
STATUS  : 200

BODY:

<!doctype html>
<html lang=es>
 <head>
  <meta charset=utf-8>
  <title>2026 XMAS CTF by s4ur0n (CS3 Group)</title>
  <style>
   body {
    background-image: url('level5.png');
    background-repeat: no-repeat;
    background-attachment: fixed;
    background-size: 100% 100%;
  }
 </style>
 </head>
 <body>
 </body>
```

Level#06 → Level#07

flag: NOEL / XmasCTF0x7EA

#arduino

Usamos las credenciales del reto previo Level#05 → Level#06, el texto ya nos indica cláramente que tenemos que reversear un firmware, nos bajamos el zip y lo primero es listar que tienes

```
❯ 7z l 2026xmas.zip | more

7-Zip (z) 25.01 (x64) : Copyright (c) 1999-2025 Igor Pavlov : 2025-08-03
 64-bit locale=en_US.UTF-8 Threads:16 OPEN_MAX:1024, ASM

Scanning the drive for archives:
1 file, 9893525 bytes (9662 KiB)

Listing archive: 2026xmas.zip

--
Path = 2026xmas.zip
Type = zip
Physical Size = 9893525

   Date      Time    Attr         Size   Compressed  Name
------------------- ----- ------------ ------------  ------------------------
```

```
1982-10-08 13:37:00 .....   4294967240     6666128  0
1982-10-08 13:37:00 .....   4294967209     6666092  1
1982-10-08 13:37:00 .....   4294967178     6666056  2
1982-10-08 13:37:00 .....   4294967147     6666020  3
1982-10-08 13:37:00 .....   4294967116     6665984  4
1982-10-08 13:37:00 .....   4294967085     6665948  5
1982-10-08 13:37:00 .....   4294967054     6665912  6
1982-10-08 13:37:00 .....   4294967023     6665876  7
1982-10-08 13:37:00 .....   4294966992     6665840  8
1982-10-08 13:37:00 .....   4294966961     6665804  9
```

y sinceramente nos quedamos a cuadro.. la verdad que estuve bastante rato dando vueltas, incluso le pregunté a un amigo, si eso era el firmware, porque era muy raro porque parecía un zip bomba, se puede leer más en internet (https://es.wikipedia.org/wiki/Bomba_zip). Asi que como siempre hago, miré el source de la página y vi que tenia el sha256 de validación

```
35   <a href="xmas2026.zip" download="xmas2026.zip"><img src="download.png" height="54px"/></a>
36   <!--
37    Always check the shasum for files. If doesn't match, open the file at your own risk. Refresh the page until matches.
38    sha512: 18846dbedfb84a2c755bc4759318e4ab498c385f1e532ba9c5ba21dbeb05c57c4a2e9694b7553071eafd635dc67777bc5e4d00d3efedd3f5f3e6dbf959cbdbc4
39    sha256: 1998e736ec2d92275b65b28c202a5ed641ed00e60ae459ac1529443243221964
40  -->
41  <h1> </h1><h1> </h1><h1>Your mission:</h1>
```

Asi que lo comparamos y efectivamente no es el mismo archivo, si que nos lo volvemos a bajar, pero usando otro navegador y vemos que se llama distinto y el hash ya coincide

```
❱ sha256sum 2026xmas.zip
f1dc920869794df3e258f42f9b99157104cd3f8c14394c1b9d043d6fcda14c0a  2026xmas.zip
❱ sha256sum *.zip
1998e736ec2d92275b65b28c202a5ed641ed00e60ae459ac1529443243221964  xmas2026.zip
```

Miramos el contenido y sí, ésta vez es el bueno

```
❱ 7z l xmas2026.zip

7-Zip (z) 25.01 (x64) : Copyright (c) 1999-2025 Igor Pavlov : 2025-08-03
 64-bit locale=en_US.UTF-8 Threads:16 OPEN_MAX:1024, ASM
```

```
Scanning the drive for archives:
1 file, 49424 bytes (49 KiB)

Listing archive: xmas2026.zip

--
Path = xmas2026.zip
Type = zip
Physical Size = 49424

   Date      Time    Attr         Size   Compressed  Name
------------------- ----- ------------ ------------  ------------------------
2025-12-24 12:54:48 D....            0            0  build
2025-12-24 12:54:38 D....            0            0  build/arduino.avr.uno
2025-12-24 12:53:00 .....        63952        25738  build/arduino.avr.uno/xma
sbomb.ino.elf
2025-12-24 12:53:00 .....           13           10  build/arduino.avr.uno/xmasbom
b.ino.eep
2025-12-24 12:53:00 .....        32768         5362  build/arduino.avr.uno/xmas
bomb.ino.with_bootloader.bin
2025-12-24 12:53:00 .....        22476         8781  build/arduino.avr.uno/xmas
bomb.ino.with_bootloader.hex
2025-12-24 12:53:00 .....        21540         8115  build/arduino.avr.uno/xmasb
omb.ino.hex
------------------- ----- ------------ ------------  ------------------------
2025-12-24 12:54:48           140749        48006  5 files, 2 folders
```

Asi que lo extraemos todo y con un file vemos que es

```
level06/build/arduino.avr.uno via 🐍 v3.12.3 (ctf-venv)
❯ file *
xmasbomb.ino.eep:               ASCII text, with CRLF line terminators
xmasbomb.ino.elf:               ELF 32-bit LSB executable, Atmel AVR 8-bit, v
ersion 1 (SYSV), statically linked, with debug_info, not stripped
xmasbomb.ino.hex:               ASCII text, with CRLF line terminators
xmasbomb.ino.with_bootloader.bin: data
xmasbomb.ino.with_bootloader.hex: ASCII text
```

Concretamente

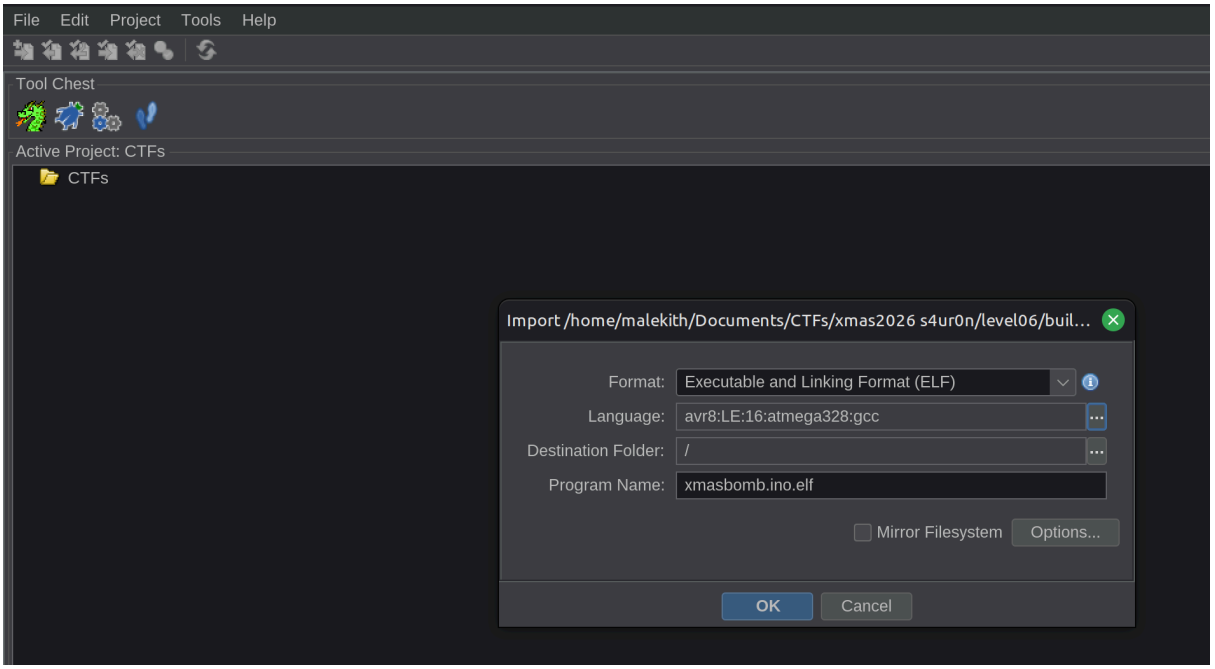
- xmasbomb.ino.elf: **ELF file** with debug symbols (not stripped!)
- xmasbomb.ino.eep: **EEPROM data** file
- el export del firmware en distintos formatos (HEX, BIN)

Dado que tenemos un ELF con símbolos nos centramos en ese, lanzamos un strings

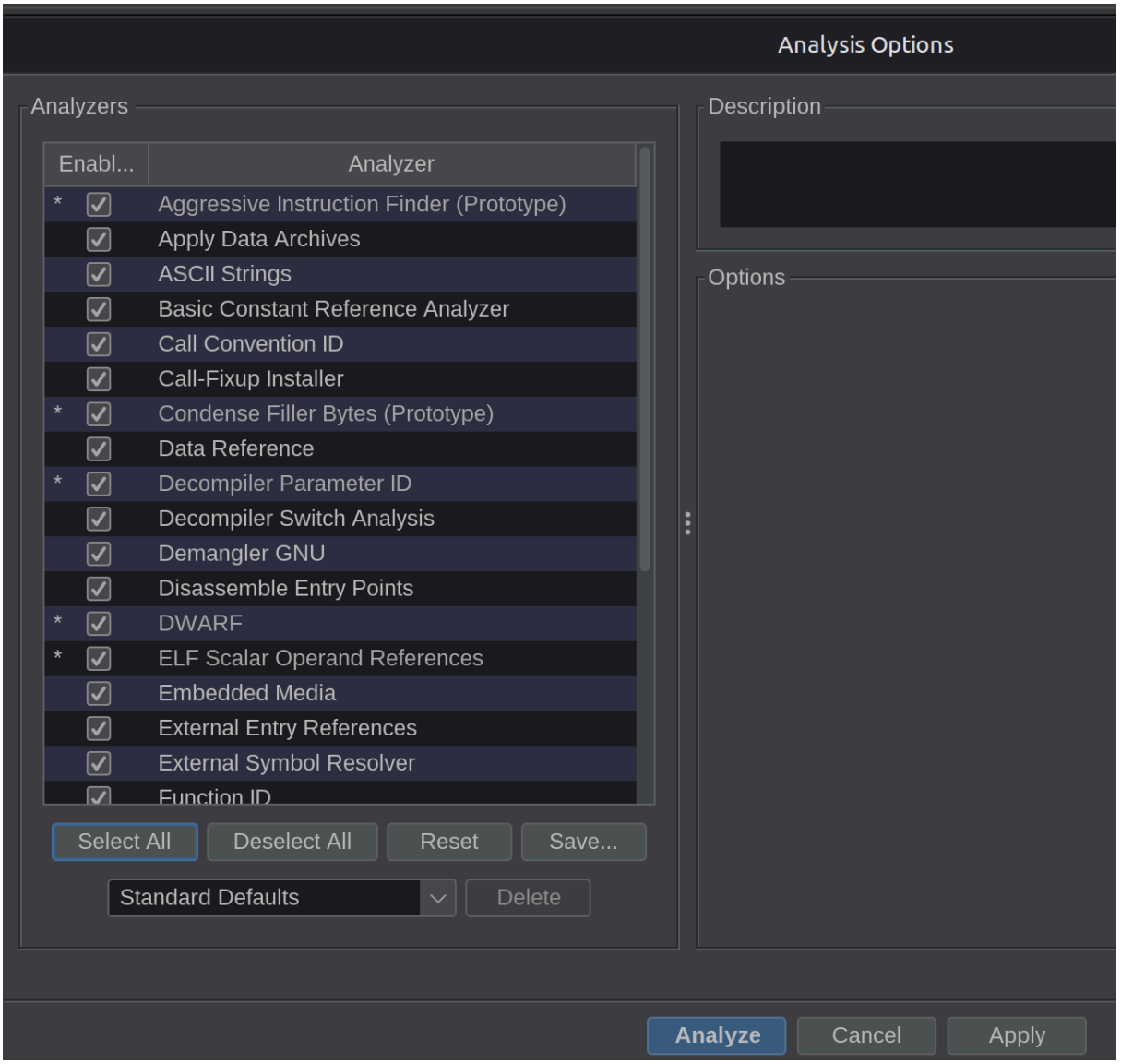
```
❯ strings xmasbomb.ino.elf | more
q,a,`
   h>s@
$Y3@
#+$+%+y
D'U'
]  n
X.a,q,
".3$:
?OOO_O
123A456B789C*0#D
2026 XMAS CTF by
s4ur0n (Pedro C)
--cs3group.com--
 Initializing *
ACTIVATED BOMB!
Remain
 s.
 * GAME OVER! *
Try again!!! ;-)
Deactivate?
Pin:
** CONGRATS!! **
Goto level6/PIN
Eg.level6/AB2026
 Disarming bomb
   POWER OFF
```

```
--More--
```

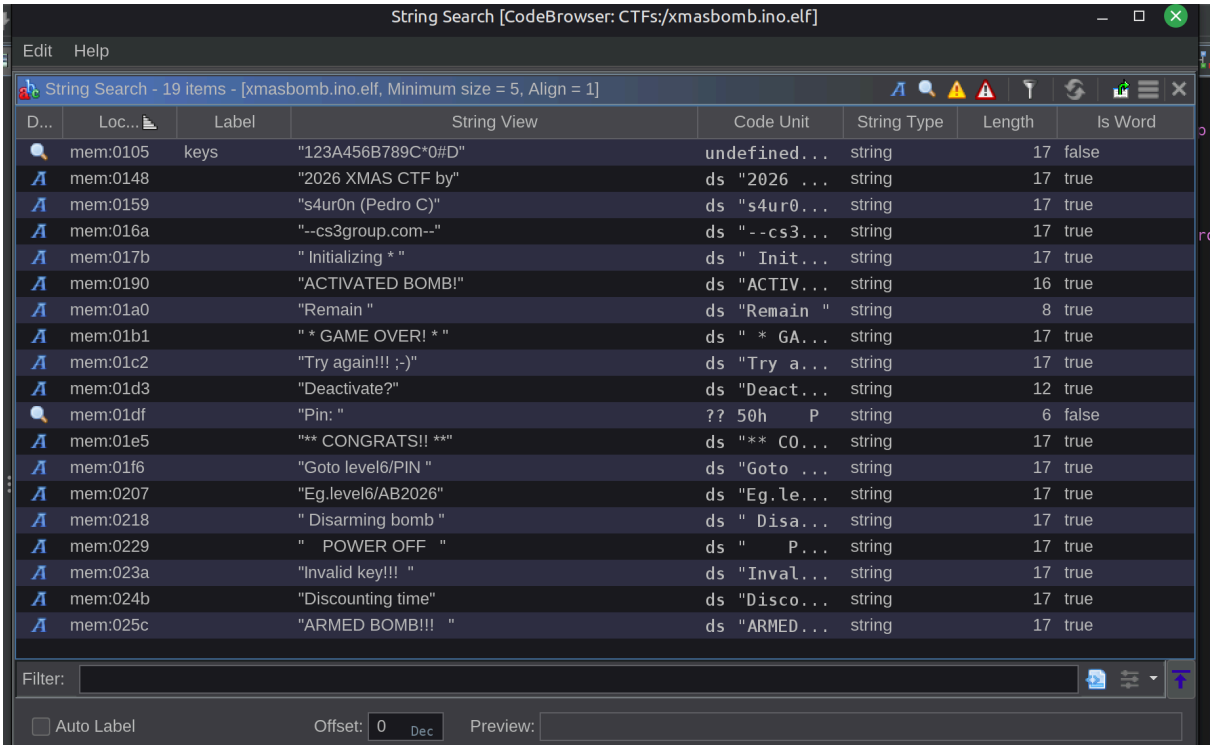
Asi que vamos a Ghidra y lo cargamos, casualmente, hace poco hice otro CTF donde se tenia que reversear un ARM y tengo el Ghidra preparado, pero si no aparece el lenguaje , se puede seguir ésta guia: https://www.jonaslieb.de/blog/arduino-ghidra-intro/



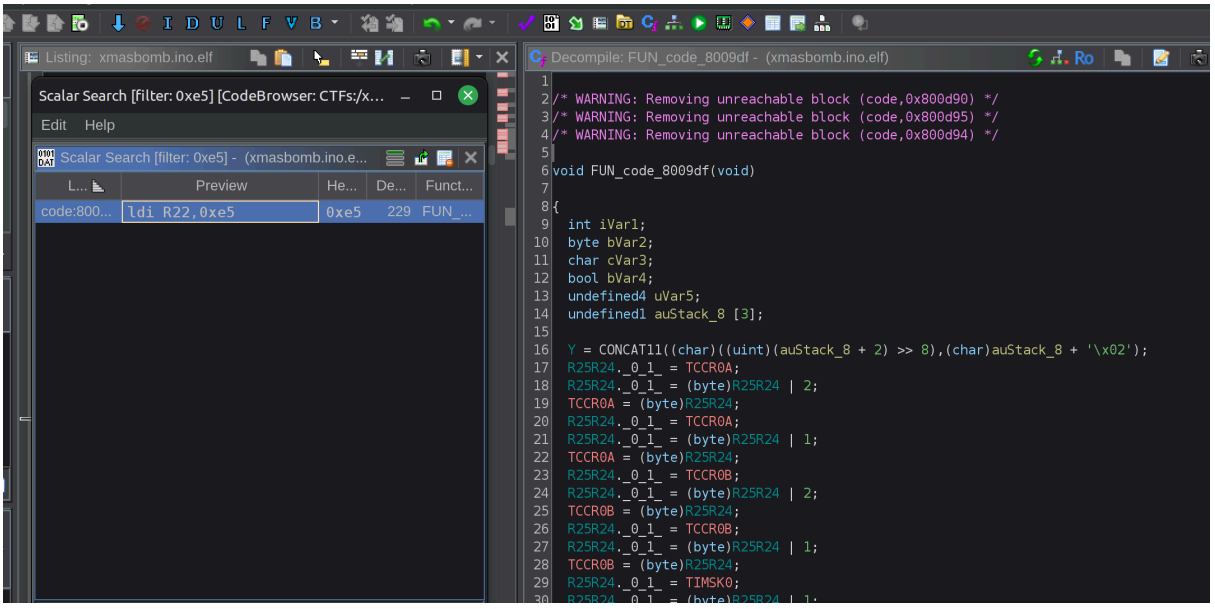
Activamos todas las opciones de Analice y buscamos strings



Buscamos las strings



y nos centramos en CONGRATS, la direccion de congrats es 0x8001e5, porque siempre empiezan en 0x80000, asi ahora buscamos el scalar 0xe5



En esa función encontramos la siguiente rutina

```
if ((byte)R25R24 != '\0') {
  auStack_8 = (undefined1  [3])0xc7a;
  func_0x000800();
```

```
      R25R24._1_1_ = *(char *)(Y + 1);
      if (R25R24._1_1_ == '5') {
        R25R24 = *(int *)(Y + 2) + 0x10;
        if (R16 == 1 && R17 == (byte)(R1 + (R16 == 0))) goto LAB_code_800c9
8;
      }
      else if (R25R24._1_1_ < '5') {
        if ((R25R24._1_1_ == '2') &&
           (R25R24 = *(int *)(Y + 2) + 0x20, R16 == R1 && R17 == (byte)(R1 +
(R16 < R1))))
        goto LAB_code_800c98;
      }
      else {
        R25R24._0_1_ = *(char *)(Y + 1);
        if ((byte)R25R24 == 'A') {
          if ((R16 == 3 && R17 == (byte)(R1 + (R16 < 3))) ||
             (R16 == 5 && R17 == (byte)(R1 + (R16 < 5)))) {
            R25R24 = *(int *)(Y + 2) + 4;
            goto LAB_code_800c98;
          }
        }
        else if (((byte)R25R24 == 'C') &&
              ((R16 == 2 && R17 == (byte)(R1 + (R16 < 2)) ||
               (R16 == 4 && R17 == (byte)(R1 + (R16 < 4)))))) {
          R25R24 = *(int *)(Y + 2) + 8;
```

que básicamente hace lo siguiente

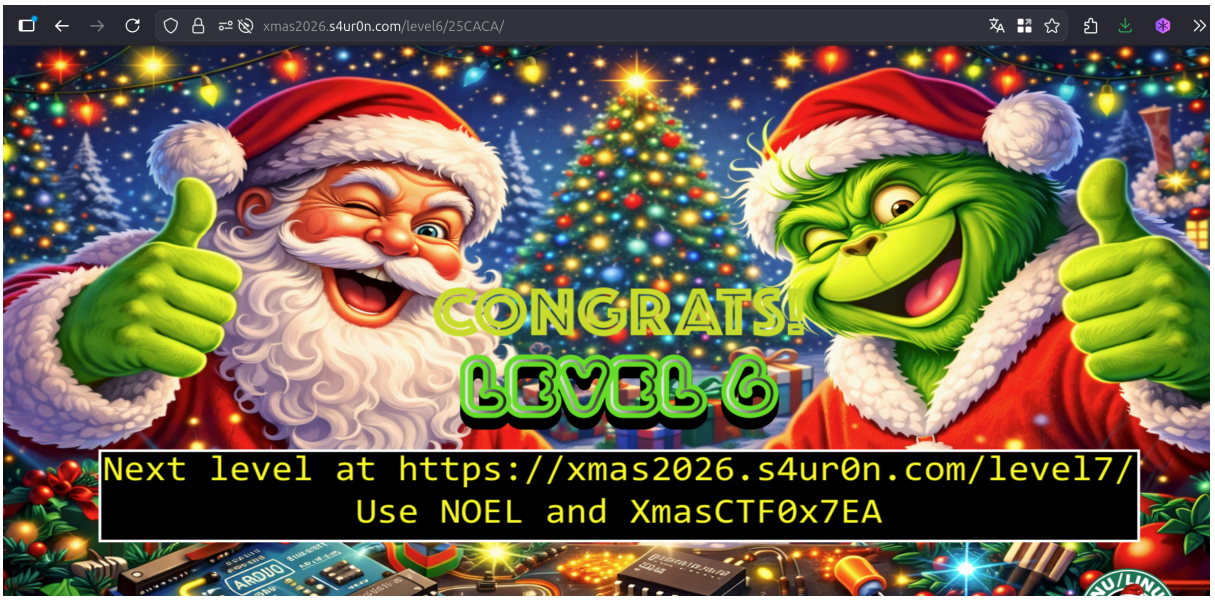
```
accumulator = 0
for position in range(6):
    key = getKey()

    if key == '2' and position == 0: accumulator += 32
    if key == '5' and position == 1: accumulator += 16
    if key == 'C' and position in [2,4]: accumulator += 8
    if key == 'A' and position in [3,5]: accumulator += 4
```

```
if accumulator == 0x48:  # 72
    print("CONGRATS!")
```

es decir

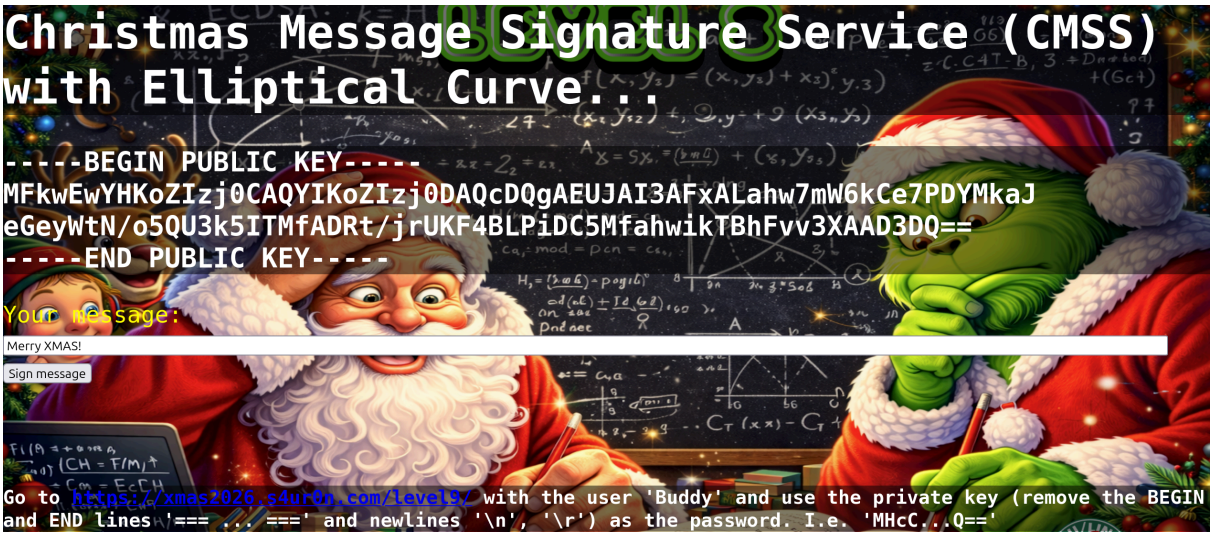
- **Posición 0**: Tecla '2' suma 0x20 (32)
- **Posición 1**: Tecla '5' suma 0x10 (16)
- **Posiciones 2,4**: Tecla 'C' suma 8
- **Posiciones 3,5**: Tecla 'A' suma 4

Con lo que nuestro código es 25CACA, probamos en el navegador



Level#08 → Level#09

flag:Buddy/MHcCAQEEIBao9HZP9JbOG/oPDdI2jGoYKgeDBIDRu7SKuMrCsLmHoAoGCCqGSM49AwEHoUQDQgAEUJAI3



Usando las credenciales del reto previo Level#07 → Level#08 nos logamos y nos encontramos con un reto de criptografía

Dado que nos permiten firmar varios mensajes, el error se trata de reutilización del nonce (k) en ECDSA lo que implica una colisión de `r`

En ECDSA:

- La firma es `(r, s)`
- `r = (k·G).x mod n`
- Si **se reutiliza el mismo** `k`, entonces:
  - `r` será **idéntico** en firmas distintas
  - y eso permite romper el cifrado

El objetivo:

1. Enviar varios mensajes
2. Recibir las firmas
3. Buscar colisiones
4. Recuperar la clave privada
5. **Reconstruye la clave privada EC**
6. La exporta en DER y la usa como password

Con eso podemos hacer el siguiente script basándonos en éste https://github.com/Marsh61/ECDSA-Nonce-Reuse-Exploit-Example?utm_source=chatgpt.com

```
#!/usr/bin/env python3
import requests
import hashlib
import base64
import re
from collections import defaultdict
from math import gcd

from cryptography.hazmat.primitives.asymmetric.utils import decode_dss_signature
```

```
from cryptography.hazmat.primitives.asymmetric import ec
from cryptography.hazmat.primitives import serialization

# URLs de los niveles
URL_L8 = "https://xmas2026.s4ur0n.com/level8/"
URL_L9 = "https://xmas2026.s4ur0n.com/level9/"

# credenciales level8
USER_L8 = "Vixen"
PASS_L8 = "8thXMASChallenge"

# usuario level9 (password lo calculamos)
USER_L9 = "Buddy"

# orden de la curva secp256r1 (P-256)
N = int(
    "FFFFFFFF00000000FFFFFFFFFFFFFFFFBCE6FAADA7179E84F3B9CAC2FC632551",
    16,
)

# sesión para level8
sess8 = requests.Session()
sess8.auth = (USER_L8, PASS_L8)

# regex para pescar base64 largos en la respuesta
B64_RE = re.compile(r"[A-Za-z0-9+/]{40,}={0,2}")

def inv_mod(x, m):
    # inverso modular (Python 3.8+ soporta pow(x, -1, m))
    return pow(x, -1, m)

def extract_sig_der(html: str) -> bytes:
    """
    Busca en la respuesta algo que parezca base64,
    lo decodifica y se queda con el primer DER que sea
    una firma ECDSA válida (dos INTEGERs).
    Si el body es solo la firma base64 (como en tu ejemplo),
    también funciona.
    """
    candidates = B64_RE.findall(html)
    if not candidates:
        raise ValueError("No se encontró ninguna cadena base64 en la respuesta")

    for cand in candidates:
        try:
            raw = base64.b64decode(cand, validate=True)
        except Exception:
            continue
        try:
            decode_dss_signature(raw)
            return raw
        except Exception:
            continue

    raise ValueError("No se encontró ninguna firma DER válida en la respuesta")

def get_sig_for_msg(msg: bytes):
    """
```

```
    Envía msg al servicio de level8, devuelve (r, s, z).
    """
    resp = sess8.post(URL_L8, data={"msg": msg.decode()})
    resp.raise_for_status()

    body = resp.text
    sig_der = extract_sig_der(body)

    R, S = decode_dss_signature(sig_der)
    z = int.from_bytes(hashlib.sha256(msg).digest(), "big")
    return R, S, z

def recover_private_and_password():
    """
    Pide firmas a level8 hasta encontrar colisión de r,
    recupera d y devuelve (d, password_b64_para_Buddy).
    """
    sigs_by_r = defaultdict(list)

    print("[+] Empezando a pedir firmas a level8...")

    for i in range(1, 5000):  # 5000 suele sobrar si hay reuse de nonce
        msg = f"message-{i}".encode()
        try:
            R, S, Z = get_sig_for_msg(msg)
        except Exception as e:
            print(f"[!] Error obteniendo firma {i}: {e}")
            continue

        print(f"[+] Firma {i}: r = {hex(R)}")
        sigs_by_r[R].append((S, Z, msg))

        if len(sigs_by_r[R]) >= 2:
            print("[+] Encontrada colisión de r, calculando clave privada...")
            (s1, z1, m1), (s2, z2, m2) = sigs_by_r[R][:2]

            num = (s1 * z2 - s2 * z1) % N
            den = (R * (s2 - s1)) % N

            if gcd(den, N) != 1:
                print("[!] Denominador no invertible, probando otra colisión...")
                continue

            d = (num * inv_mod(den, N)) % N
            print(f"[+] Clave privada d = {hex(d)}")

            # construir la EC PRIVATE KEY a partir de d
            priv = ec.derive_private_key(d, ec.SECP256R1())

            # exportar en formato SEC1 / TraditionalOpenSSL (EC PRIVATE KEY)
            der_sec1 = priv.private_bytes(
                encoding=serialization.Encoding.DER,
                format=serialization.PrivateFormat.TraditionalOpenSSL,
                encryption_algorithm=serialization.NoEncryption(),
            )

            password_b64 = base64.b64encode(der_sec1).decode()
            print("\n[+] PASSWORD PARA level9 (user Buddy):")
```

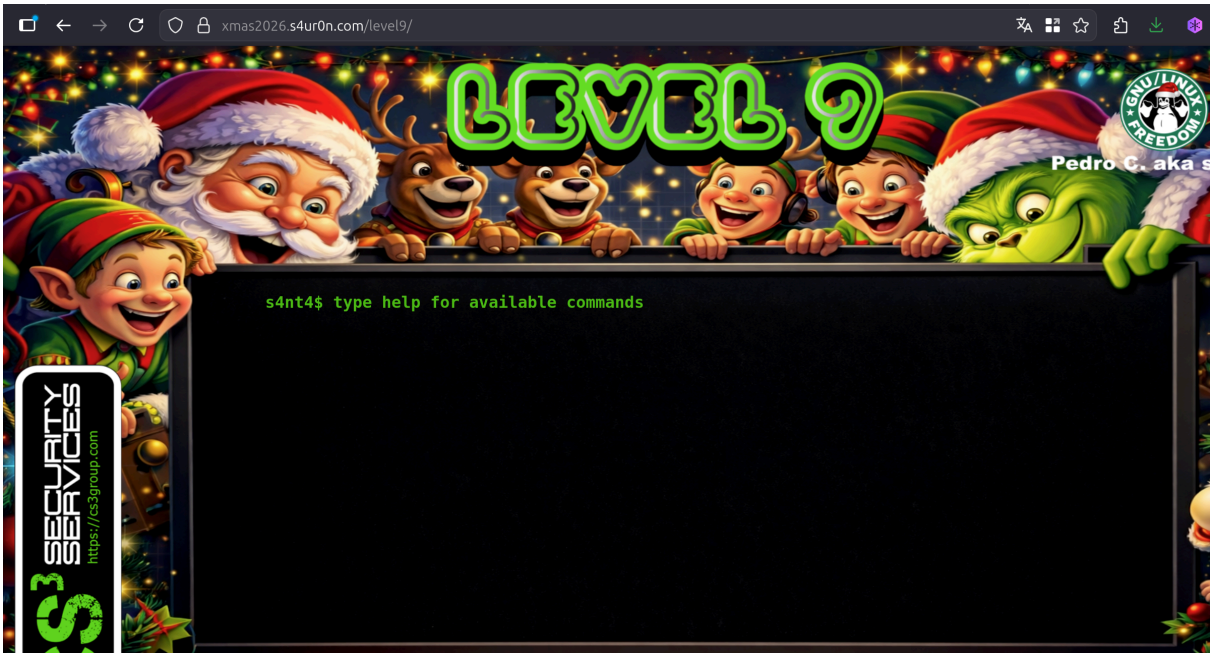
```
            print(password_b64)
            print("[*] Úsalo tal cual, sin BEGIN/END ni saltos de línea.\n")

            return d, password_b64

    raise RuntimeError(
        "No se encontró colisión de r en el rango dado; "
        "quizá no hay reuse de nonce o hay que ampliar el rango."
    )

def login_level9(password_b64: str):
    """
    Hace GET a level9 con Basic Auth Buddy:password_b64
    y muestra status + body.
    """
    print("[+] Intentando login en level9 con Buddy...")
    resp = requests.get(URL_L9, auth=(USER_L9, password_b64))
    print(f"[+] Respuesta level9: HTTP {resp.status_code}")
    print("----- BODY level9 -----")
    print(resp.text)
    print("-----------------------")

def main():
    d, pwd_b64 = recover_private_and_password()
    login_level9(pwd_b64)

if __name__ == "__main__":
    main()
```

y bingo

```
❯ python3 solver.py
[+] Empezando a pedir firmas a level8...
[+] Firma 1: r = 0x794e3e968325a26fb433d20e305d24b6a5032e87fb0abc035f2b8a448c9abf2
[+] Firma 2: r = 0x794e3e968325a26fb433d20e305d24b6a5032e87fb0abc035f2b8a448c9abf2
[+] Encontrada colisión de r, calculando clave privada...
[+] Clave privada d = 0x16a8f4764ff496ce1bfa0f0dd2368c6a182a07830480d1bbb48ab8cac2b0b987

[+] PASSWORD PARA level9 (user Buddy):
MHcCAQEEIBao9HZP9JbOG/oPDdI2jGoYKgeDBIDRu7SKuMrCsLmHoAoGCCqGSM49AwEHoUQDQgAEUJAI3AFxA
Lahw7mW6kCe7PDYMkaJeGeyWtN/o5QU3k5ITMfADRt/jrUKF4BLPiDC5MfahwikTBhFvv3XAAD3DQ==
[*] Úsalo tal cual, sin BEGIN/END ni saltos de línea.

[+] Intentando login en level9 con Buddy...
[+] Respuesta level9: HTTP 200
----- BODY level9 -----
<!doctype html>
<html lang=es>
 <head>
  <meta charset=utf-8>
  <title>2026 XMAS CTF by s4ur0n (CS3 Group)</title>
  <style>
   body {
    background-image: url('console.png');
    background-repeat: no-repeat;
    background-attachment: fixed;
    background-size: 100% 100%;
    color: white;
```

```
    font-family: Consolas,monaco,monospace;
   }
   #console {
    position: absolute;
    top: 40%;
    left: 310px;
    margin-top: -50px;
    margin-left: -50px;
    width: 1000px;
    height: 600px;
    color: #4CBB17;

   }
   #keyboard {
    position: fixed; bottom: 50px; left:220px; width:100%;
   }
  </style>
  </head>
  <body>
    <div id="console"><h4>s4nt4$ type help for available commands</h4></div>
    <form action="" enctype="multipart/form-data" method="POST" id="kbd">
     <input type="hidden" name="csrf_token" value="79ff3aff8b9ffb75f400515fd52f5d26f89096563976eddf431f4c
fc479277ab" />
     <div id="keyboard">Command: <input type="text" name="cmd" size="140" maxlength="140" /><input type="s
ubmit" value="Send" /></div>
    </form>
  </body>
</html>
```

y parece que funciona dado que en la respuesta aparece un “console.png” asi que vamos al nivel 9 y lo probamos y perfecto!

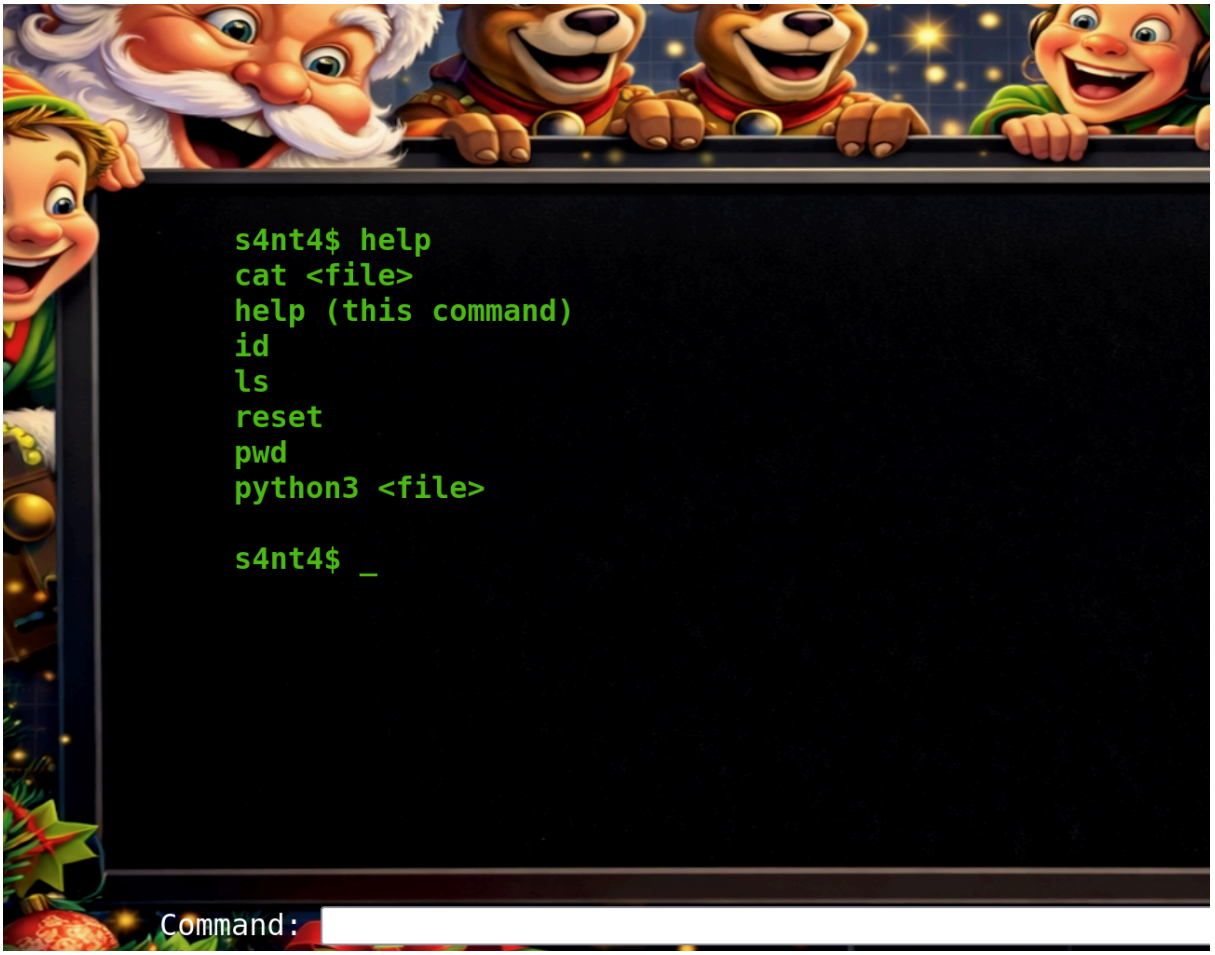


Level#09 → Level#10

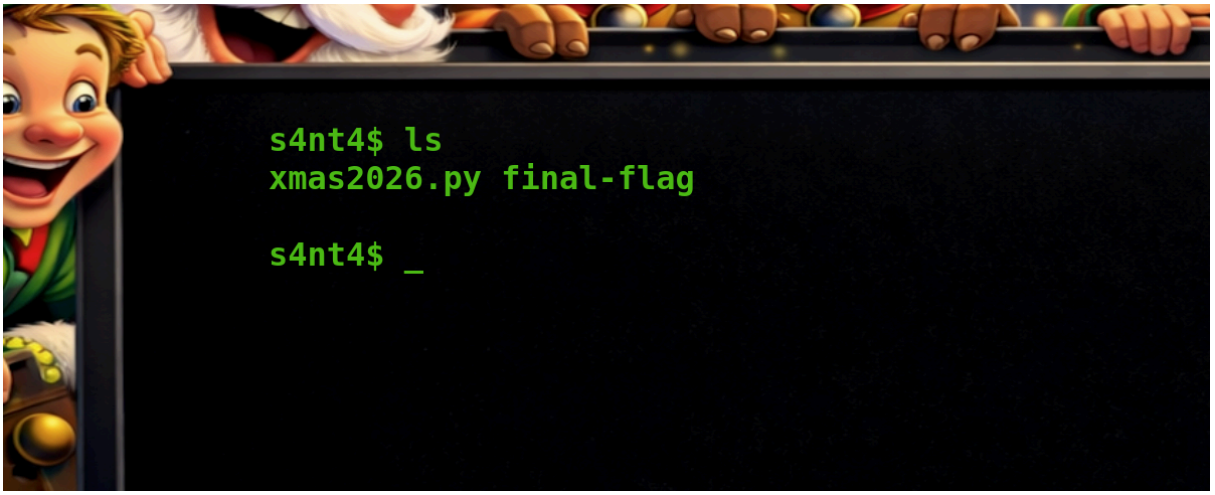
flag: Santa&Grinch / F1n4lXMASCtf

#pwn

Reto de pwn, nos dan una consola que puede hacer



Si hacemos un ls



Obviamente no podemos hacer un cat de la flag

Asi que editamos el py

```
#!/usr/bin/env python3
import tempfile
import subprocess
import os

comment = input("XMAS> ").replace(" ", "").replace(" ", "")

code = f"""print("2026 XMAS CTF")
# This is a python comment.
# Here's another from you:
# {comment}
```

```
print("Thanks for playing the XMAS CTF!")"""

with tempfile.NamedTemporaryFile(mode="w", suffix=".py", delete=False)
as f:
    f.write(code)
    temp_filename = f.name

try:
    result = subprocess.run(
        ["python3", temp_filename], capture_output=True, text=True, timeout=
5
    )

    if result.stdout:
        print(result.stdout, end="")
    if result.stderr:
        print(result.stderr, end="")

except subprocess.TimeoutExpired:
    print("Timeout")
finally:
    os.unlink(temp_filename)
```

La clave está en que el **input se inserta literalmente dentro del fichero Python**, solo precedido por `#`. Eso quiere decir que tenemos una inyección de código: **input del usuario dentro de un archivo Python que luego se ejecuta**.
Para probarlo podemos directamete usar éste código

```
print(open("/home/s4nt4/final-flag").read())
```

Ejecutamos el python del server

Cargamos nuestro py malicioso y nos da la flag!

Level#10

Nos bajamos los ficheros y nos dan credenciales y la key

```
❯ cat final creds.txt
-----BEGIN OPENSSH PRIVATE KEY-----
b3BlbnNzaC1rZXktdjEAAAAABG5vbmUAAAAEbm9uZQAAAAAAAAABAAA
AMwAAAAtzc2gtZW
QyNTUxOQAAACADAycPQ5IcP/2FKcZYQ+FotRvIeRuTpeVoOjr8YP+IpwAAA
KDmXiM55I4j
OQAAAAtzc2gtZWQyNTUxOQAAACADAycPQ5IcP/2FKcZYQ+FotRvIeRuTp
eVoOjr8YP+Ipw
AAAEC7B7QK+4nQep3FD3NOKHzyxPUQcshTtVybr7+gHoB0bQMDJw9Dk
hw//YUpxIhD4Wi1
G8h5G5OI5Wg6Ovxg/6WnAAAAGWZpbmFsQHhtYXMyMDI2LnM0dXIwbi5j
b20BAgME
-----END OPENSSH PRIVATE KEY-----
final@xmas2026.s4ur0n.com
16eDfv7hlWnrjHwl
```

Cambiamos los permisos de la key (final) para que funcione y nos conectamos

```
❯ chmod 600 final

CTFs/xmas2026 s4ur0n/Level10 via 🐍 v3.12.3 (ctf-venv)
```

```
❯ ssh -i ./final final@xmas2026.s4ur0n.com
ssh: connect to host xmas2026.s4ur0n.com port 22: Connection refused
```

Nos da un refused.. asi que toca buscar el puerto, no se porqué el nmap tarda horrores en mi casa

```
❯ nmap -p-  -vvv xmas2026.s4ur0n.com
Starting Nmap 7.94SVN ( https://nmap.org ) at 2026-01-03 08:59 CET
Initiating Ping Scan at 08:59
Scanning xmas2026.s4ur0n.com (135.181.196.247) [2 ports]
Completed Ping Scan at 08:59, 0.06s elapsed (1 total hosts)
Initiating Parallel DNS resolution of 1 host. at 08:59
Completed Parallel DNS resolution of 1 host. at 08:59, 0.07s elapsed
DNS resolution of 1 IPs took 0.07s. Mode: Async [#: 1, OK: 1, NX: 0, DR: 0, S
F: 0, TR: 1, CN: 0]
Initiating Connect Scan at 08:59
Scanning xmas2026.s4ur0n.com (135.181.196.247) [65535 ports]
Discovered open port 443/tcp on 135.181.196.247
Discovered open port 80/tcp on 135.181.196.247
Increasing send delay for 135.181.196.247 from 0 to 5 due to 30 out of 99 dr
opped probes since last increase.
Increasing send delay for 135.181.196.247 from 5 to 10 due to 11 out of 12 dr
opped probes since last increase.
Increasing send delay for 135.181.196.247 from 10 to 20 due to 11 out of 13 d
ropped probes since last increase.
Increasing send delay for 135.181.196.247 from 20 to 40 due to 11 out of 14
dropped probes since last increase.
Connect Scan Timing: About 0.67% done
Connect Scan Timing: About 1.81% done; ETC: 09:55 (0:55:03 remaining)
Connect Scan Timing: About 2.96% done; ETC: 09:51 (0:49:47 remaining)
Connect Scan Timing: About 4.10% done; ETC: 09:49 (0:47:10 remaining)
```

Pero al final @oreos nos pasa el puerto 3466

```
❯ ssh -i ./final final@xmas2026.s4ur0n.com -p34666
Linux xmas2026 6.12.57+deb13-cloud-amd64 #1 SMP PREEMPT_DYNAMIC
Debian 6.12.57-1 (2025-11-05) x86_64
```

```
 ####################""N########NN#########################
######### s4ur0n ##
 ################N@" _, #######F  0################ https://cs3
group.com ##
 ###############@" _g# j#@FN#@ _  0#########################
################
 #############NF _g##F 0" _ " j# j############################
############
 #############F g#### - _g   g@ q####@" "N##F""9N##N"P"N###
@"3#@"0#########
 #############NJ####   g#"  d@  0###F g@ g#P _  0##' , j##@ j0# j##
########
 ##################"  d#F  0#  J###" +"_d#@ J" g##F 0  0##  ##  0##
########
 ################NF _0#@ _0## j##NF  _g##F gF j#N"_A  0##  0#  J##
#########
 ###############N" _0#@  d##  0N@"  d##@"_d# qNF _#F J@"   @  jP
T##########
 ###############F _0##_ d###p ""_g#_`""_g###p  jg###_ _g0k _p F_A#
#########
 ################g#####N#####pgg####Ngg######NN#####NNN
###N#  j############
 ##################################################
###  j#############
 ######N@" "N#####NN#################################
##F j##############
 ####NF  ___ ##### j#######################################L 0
##############
 ###@   gN# j###@ j#############@"N#########################
N###############
 ##F  a0"jF d##N" ###############g0#########" B##############
##############
 #F  j##   j###F N######NNNN####N#####F"2NNF dN###N#####N@0#
####@@N#####@""
 #  j###pgg###@ g@""###@    ###F J###" j#E, ,_0#@ j@""0@  j##N@"_, j
###@  d
 F  #########@     d##@ jF J##F j### j d##' 0##@   _f ", j###  g#  ###@
p #
```

```
 F j########P   d j##@ jF j##@ j##@ gP 0#F j##P   d@  g' ###  0#  d##F
d  #
 # 3#####@F   _#F 0N@ g@ j##"  0N@  #  N"  N#"_ _0# _0# jN@  g#" j#@"
t@ j@
 #L_""""" _g" 0#_ P"_d#& 9" ap F"_p " ` arJ@ a' 0# _0## ?"__ " j_`F g_ ' "_
 ##Np___gg##pg###_jg####L_g0##__g##p_ag0##_jg#Ld##m0###p_g##
p_g##_a0##__gg#

The XMAS CTF is for everyone. Please don't touch anything on the server s
o
everyone else can keep playing. Watch all you want, but be respectful.

Enjoy it!

Last login: Fri Jan  2 22:36:21 2026 from 81.36.152.60
-bash-5.2$ id
uid=1002(final) gid=1002(final) groups=1002(final),100(users)
-bash-5.2$ sudo -l
Matching Defaults entries for final on xmas2026:
    env_reset, mail_badpass, secure_path=/usr/local/sbin\:/usr/local/bin\:/us
r/sbin\:/usr/bin\:/sbin\:/bin, use_pty

User final may run the following commands on xmas2026:
    (root) /usr/bin/zip
    (ALL) NOPASSWD: ALL
-bash-5.2$
```

vemos que la escalada es directa

```
-bash-5.2$ sudo -l
Matching Defaults entries for final on xmas2026:
    env_reset, mail_badpass, secure_path=/usr/local/sbin\:/usr/local/bin\:/us
r/sbin\:/usr/bin\:/sbin\:/bin, use_pty

User final may run the following commands on xmas2026:
    (root) /usr/bin/zip
    (ALL) NOPASSWD: ALL
```

```
-bash-5.2$ sudo -s
root@xmas2026:/home/final# id
uid=0(root) gid=0(root) groups=0(root)
root@xmas2026:/home/final#
```

Con eso nos bajamos el xmas2026 que tenemos en /home/final

Lo extramoe con

```
ffmpeg -i xmas2026 -c copy salida.mp4
```

y sacamos la flag