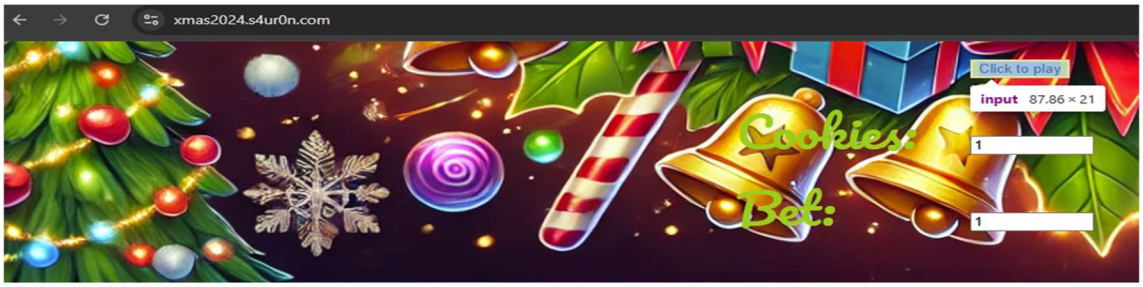
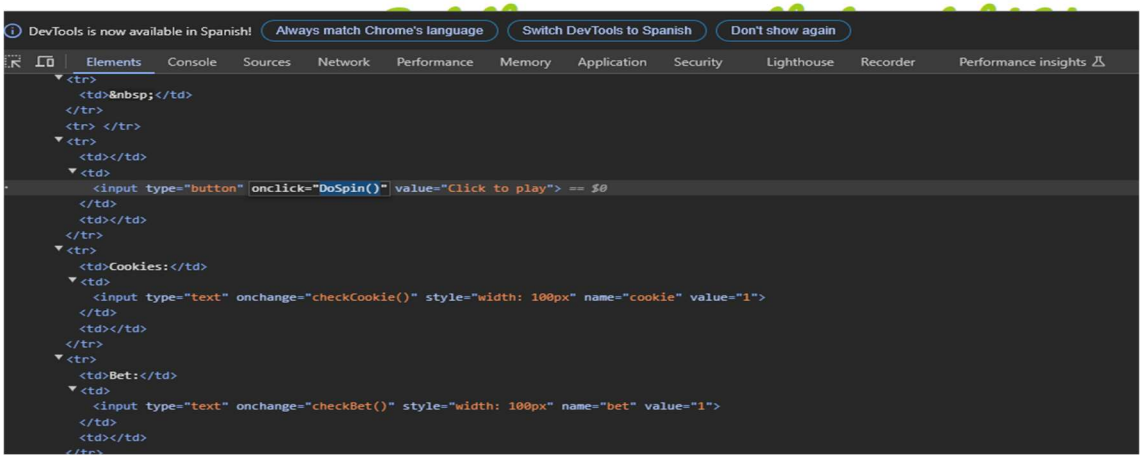
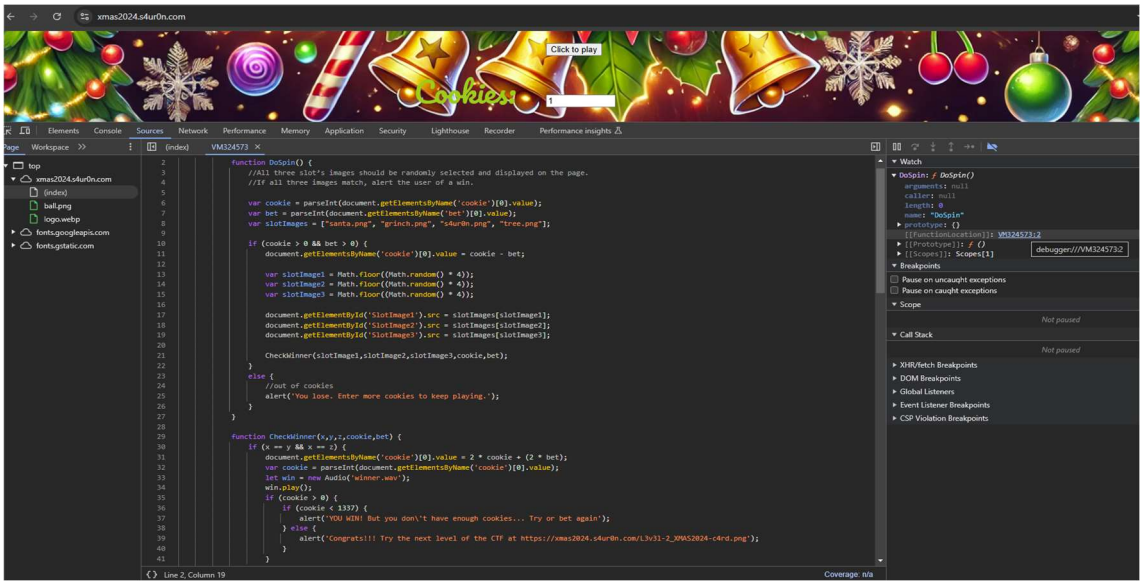
Nivel 1





Vamos a ver qué hace la función *DoSpin*(). Para eso la metemos como expresión a vigilar y nos dice dónde está definida (antes hemos tenido que deshabilitar los breakpoints en el depurador para que la carga de la página no se pare). Dentro del mismo fichero algo más abajo está la función *CheckWinner*():



Congrats!!! Try the next level of the CTF at https://xmas2024.s4ur0n.com/L3v3l-2_XMAS2024-c4rd.png

Nivel 2

Nos dan una clave privada parcialmente oculta.

El reto es parecido a este otro (https://blog.cryptohack.org/twitter-secrets). Por el tipo de estructura en la que está codificada la clave primaria (un fichero PEM), sabemos que los datos para la creación de la clave se almacenan siguiendo un orden y con un tipo determinado:

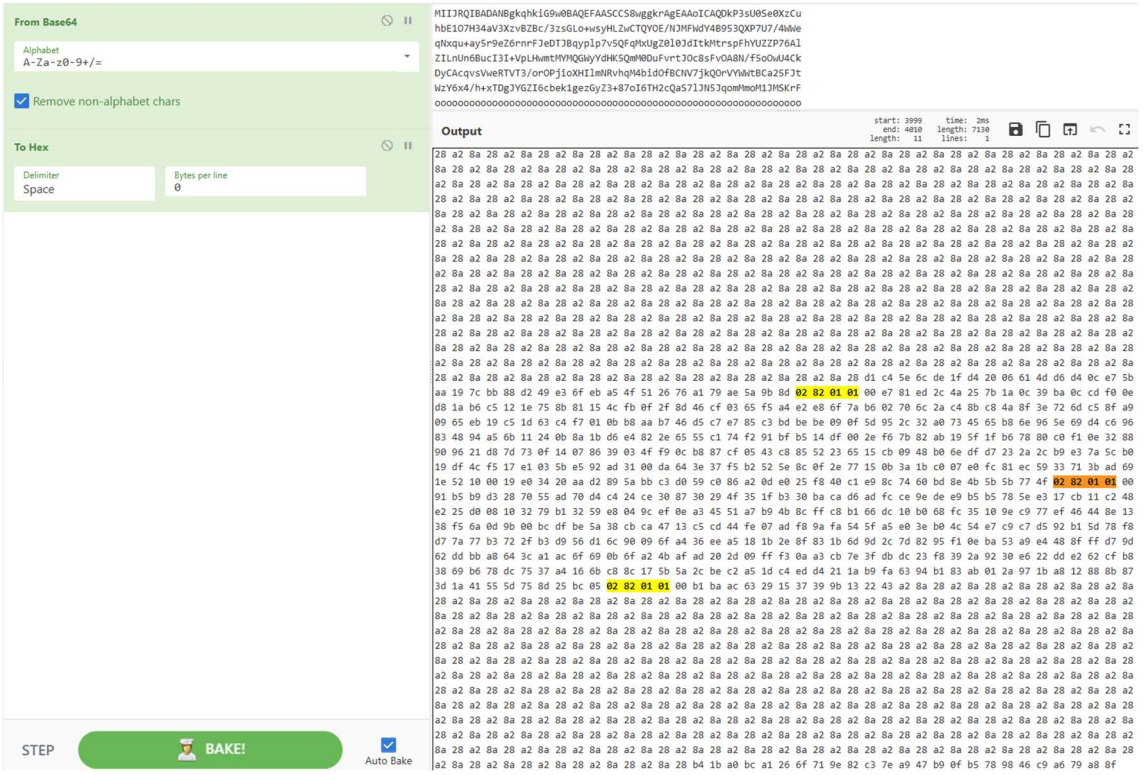
```
RSAPrivateKey ::= SEQUENCE {
  version           Version,
  modulus           INTEGER,  -- n
  publicExponent    INTEGER,  -- e
  privateExponent   INTEGER,  -- d
  prime1            INTEGER,  -- p
  prime2            INTEGER,  -- q
  exponent1         INTEGER,  -- d mod (p-1)
  exponent2         INTEGER,  -- d mod (q-1)
  coefficient       INTEGER,  -- (inverso de q) mod p
  otherPrimeInfos   OtherPrimeInfos OPTIONAL
}
```

Después de pasar la clave por un OCR y repasarla varias veces, tenemos el siguiente fichero:


```
MIIJRQIBADANBgkqhkiG9w0BAQEFAASCCS8wggkrAgEAAoICAQDkP3sU0Se0XzCu
hbE1O7H34aV3XzvBZBc/3zsGLo+wsyHLZwCTQYOE/NJMFWdY4B953QXP7U7/4WWe
qNxqu+ay5r9eZ6rnrFJeDTJBqyplp7v5QFqMxUgZ0l0JdItkMtrspFhYUZZP76Al
ZILnUn6BucI3I+VpLHwmtMYMQGWyYdHK5QmM0DuFvrtJOc8sFvOA8N/f5oOwU4Ck
DyCAcqvsVweRTVT3/orOPjioXHIlmNRvhqM4bidOfBCNV7jkQOrVYWWtBCa25FJt
WzY6x4/h+xTDgJYGZI6cbek1gezGyZ3+87oI6TH2cQaS7lJN5JqomMmoM1JMSKrF
oooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooo
oooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooo
oooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooo
oooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooo
oooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooo
oooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooo
oooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooo
oooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooo
oooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooo
oooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooo
oooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooo
oooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooo
oooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooo
oooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooo
oooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooo
oooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooo
oooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooo
oooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooo
oooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooo
oooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooo
oooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooo
0cRebN4f1CAGYU3W1AznW6oZfLuI0knjb+ulT1EmdqF5rlqbjQKCAQEA54HtLEol
exoMOboMzfAO2Bq2xRIedYuBFUz7Dy+NRs8DZfWk4uhverYCcGwqxIvISo8+cm3F
j6kJZesZxR1jxPcBC7iqt0bVx+eFw72+vgkPXZUsMqBzRWW4bpZeadTGloNIlKVr
ESQLihvW5IIuZVXBdPKRv7UU3wAu9nuCqxlfH7Z4gMDxDjKIkJYh2H1zDxQHhjkD
T/kMuIfPBUPIhVIjZRXLCUiwbt/XIyosueN6XLAZ30z1F+EDW+WSrTEA2mQ+N/Wy
Ul6MDy53FQs6G8AH4PyB7FkzcTutaR5SEAAZ4DQgqtKJWrvD0FnAhqIN4CX4QMHp
jHRgvY5LW1t3TwKCAQEAkbW50yhwVa1w1MQkzjCHMClPNR+zMLrK1q38zp7e6bW1
eF7jF8sRwkjiJdAIEDJ5sTJZ6ASc7w6jRVGnuUuM/8ixZtwQsGj8NRCeyXfvRkSO
Ezj1ag2bALzfvlo4y8pHE8XNRP4Hrfia+lRfpeA+sExU58nH1ZKxXXj413p3s3Iv
s9lW0WyQCW+kNu6lGBsuj4MbbZ0sfYKV8Q66U6nkSI//151i3buoZDyhrG9pC2+i
S6+tIC0J//MKo8t+P9vcI/g5KpIw5iLd4mLPuDhptnjcdTekFmvIjBdbWiy+wqUd
xO3UIRq5+mOUsYOrASqXG6gSiIuHPRpBVV11jSW8BQKCAQEAsbqsYykVNzmbEyJD
oooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooo
oooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooo
oooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooo
oooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooo
oooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooo
oooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooo
```

```
oooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooo
oooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooo
oooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooo
oooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooo
tBugvKEmb3GegsN+qUe5D7V4mEbJpnmojw==
```


Si decodificamos el base64 y lo pasamos a hex, teniendo en cuenta que cada campo va a venir delimitado por los bytes “02 82 01 01” que definen un tipo de dato INTEGER de 2 bytes:



Podemos ver que tenemos:

- Los últimos bytes de “*p*”: d1c45e6cde1fd42006614dd6d40ce75baa197cbb88d249e36feba54f512676a179ae5a9b8d
- “*q*”: 00e781ed2c4a257b1a0c39ba0ccdf00ed81ab6c5121e758b81154cfb0f2f8d46cf0365f5a4e2e86f7ab602706c2ac48bc84a8f3e726dc58fa90965eb19c51d63c4f7010bb8aab746d5c7e785c3bdbebe090f5d952c32a0734565b86e965e69d4c696834894a56b11240b8a1bd6e4822e6555c174f291bfb514df002ef67b82ab195f1fb67880c0f10e3288909621d87d730f14078639034ff90cb887cf0543c88552236515cb0948b06edfd7232a2cb9e37a5cb019df4cf517e1035be592ad3100da643e37f5b2525e8c0f2e77150b3a1bc007e0fc81ec5933713bad691e52100019e03420aad2895abbc3d059c086a20de025f840c1e98c7460bd8e4b5b5b774f
- Exponente 1 (“*d % (p-1)*”): 0091b5b9d3287055ad70d4c424ce308730294f351fb330bacad6adfcce9edee9b5b5785ee317cb11c248e225d008103279b13259e8049cef0ea34551a7b94b8cffc8b166dc10b068fc35109ec977ef46448e1338f56a0d9b00bcdfbe5a38cbca4713c5cd44fe07adf89afa5

45fa5e03eb04c54e7c9c7d592b15d78f8d77a77b3722fb3d956d16c90096fa436eea5181
b2e8f831b6d9d2c7d8295f10eba53a9e4488fffd79d62ddbba8643ca1ac6f690b6fa24baf
ad202d09fff30aa3cb7e3fdbdc23f8392a9230e622dde262cfb83869b678dc7537a4166b
c88c175b5a2cbec2a51dc4edd4211ab9fa6394b183ab012a971ba812888b873d1a41555
d758d25bc05

Con este programa calculamos “p” a partir del exponente 1 y usamos sus últimos bytes (que ya conocemos) para comprobar que el p calculado sea correcto:

```
from Cryptodome.Util.number import isPrime
from Cryptodome.PublicKey import RSA

# Exponente. A priori no lo sabemos, pero es el común
e = 65537

# Los últimos bytes de p (para comprobar cuando deduzcamos el p verdadero)
low_p = int("0xd1c45e6cde1fd42006614dd6d40ce75baa197cbb88d249e36feba54f51267" +
            "6a179ae5a9b8d", 16)

# q lo tenemos entero
q = int("0x00e781ed2c4a257b1a0c39ba0ccdf00ed81ab6c5121e758b81154cfb0f2f8d46c" +
        "f0365f5a4e2e86f7ab602706c2ac48bc84a8f3e726dc58fa90965eb19c51d63c4f7" +
        "010bb8aab746d5c7e785c3bdbebe090f5d952c32a0734565b86e965e69d4c696834" +
        "894a56b11240b8a1bd6e4822e6555c174f291bfb514df002ef67b82ab195f1fb678" +
        "80c0f10e3288909621d87d730f14078639034ff90cb887cf0543c88552236515cb0" +
        "948b06edfd7232a2cb9e37a5cb019df4cf517e1035be592ad3100da643e37f5b252" +
        "5e8c0f2e77150b3a1bc007e0fc81ec5933713bad691e52100019e03420aad2895ab" +
        "bc3d059c086a20de025f840c1e98c7460bd8e4b5b5b774f", 16)

# El primer exponente (dp = d % (p-1)) lo tenemos entero
dp = int("0x0091b5b9d3287055ad70d4c424ce308730294f351fb330bacad6adfcce9edee9" +
         "b5b5785ee317cb11c248e225d008103279b13259e8049cef0ea34551a7b94b8cff" +
         "c8b166dc10b068fc35109ec977ef46448e1338f56a0d9b00bcdfbe5a38cbca4713" +
         "c5cd44fe07adf89afa545fa5e03eb04c54e7c9c7d592b15d78f8d77a77b3722fb3" +
         "d956d16c90096fa436eea5181b2e8f831b6d9d2c7d8295f10eba53a9e4488fffd7" +
         "9d62ddbba8643ca1ac6f690b6fa24bafad202d09fff30aa3cb7e3fdbdc23f8392a" +
         "9230e622dde262cfb83869b678dc7537a4166bc88c175b5a2cbec2a51dc4edd421" +
         "1ab9fa6394b183ab012a971ba812888b873d1a41555d758d25bc05", 16)

# Tenemos que dp = d % (p-1), para sacar p hay que resolver p=((e*dp-1)/kp)+1
for kp in range(3, e):
    p_mul = dp * e - 1
    if p_mul % kp == 0:
        p = (p_mul // kp) + 1
        # Si el p que hemos deducido es primo y sus últimos bytes coinciden con
        # con los del PEM corrupto lo hemos encontrado
        if isPrime(p) and hex(p).endswith(hex(low_p)[2:]):
            break

# Construimos la clave
N = p*q
phi = (p-1)*(q-1)
d = pow(e, -1, phi)
key = RSA.construct((N, e, d))

# Exportamos el PEM
pem = key.export_key('PEM')
print(pem.decode())
```

Lo que nos da:

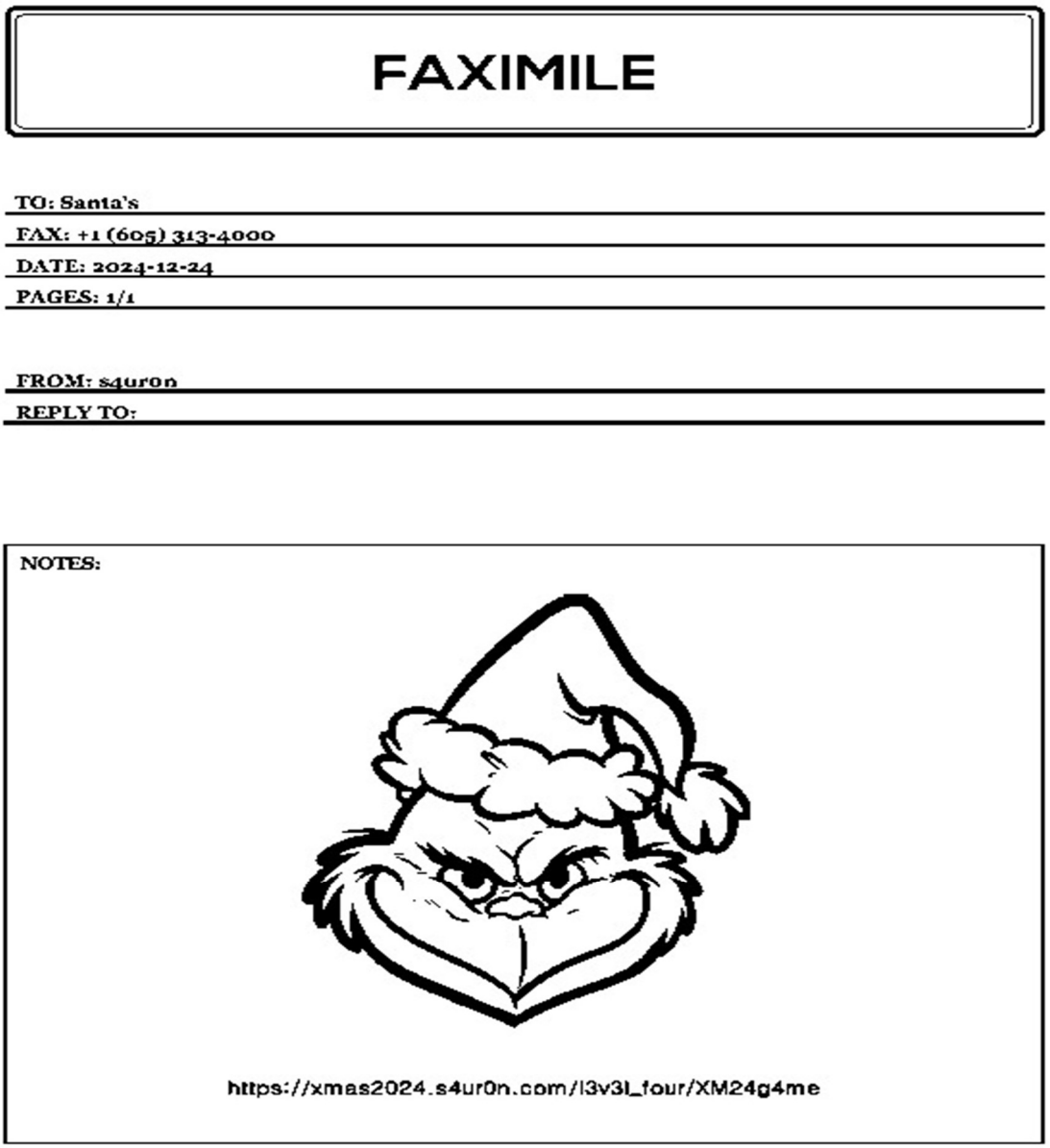
```
fuska@s0rr0w:~/Wargames/s4ur0n2024$ python3 ./reconstruir_pem.py > a.pem
fuska@s0rr0w:~/Wargames/s4ur0n2024$ cat a.pem
-----BEGIN RSA PRIVATE KEY-----
MIIJKwIBAAKCAgEA5D97FNEntF8wroWxNTux9+Gld187wWQXP987Bi6PsLMhy2cA
k0GDhPzSTBVnWOAfed0Fz+1O/+Flnqjcarvmsua/Xmeq56xSXg0yQasqZae7+UBa
jMVIGdJdCXSLZDLa7KRYWFGWT++gJWSC51J+gbnCNyPlaSx8JrTGDEBlsmHRyuUJ
jNA7hb67STnPLBbzgPDf3+aDsFOApA8ggHKr7FcHkU1U9/6Kzj44qFxyJZjUb4aj
OG4nTnwQjVe45EDq1WMFrQQmtuRSbVs2OseP4fsUw4CWBmSOnG3pNYHsxsmd/vO6
COkx9nEGku5STeSaqJjJqDNSTEiqxW9moiDVWNDUMS+9A+99EoZAzTkjcIh0YMF9
AxqkI6RnrDJqPeC9i7CM6mshzqVPKAzc8MzuEkIYmFy2M5FJynTommCamP+JwVLn
eFj+Yhrc5+O1tSrtuoeWQK3f1NzlvWb3AiXeK6RjSVyZQDHLWxqecW39J5eOrmtL
DaL4L3vcl9Cu+G1m1uBnu5F1uEWI2+z8LwjsTzSTqnLRdvYoBBg45mta4mxM3O3v
l11BlZnRIDjxldJ5xpR5seikfDdrCHv64yvRRIZN5jRf1kvz7DgKxVG1WDjf+INv
I4Q+v+6I+nwqByFqqQeNKbIWAcId2cha4hDIk9LQEMDzZiL3uNNcaFZji4MCAwEA
AQKCAgEA1Ar1eT1luXfFbhzdqCqxBywl4GQky2EFCF2GJBQVgX6pIqGqMyNl36JQ
bEZmIHb2RuxCfgxkm+r10RPm0XGGvSUJG9cLOvcn/iAcVE2DsbTGOKTEeoq8lOCN
dj9DT+6+26FCQapqDhD7okFiKyzEQhgkib1bXv3oyLygULlywOmHUQq+eIbrBTFQ
JJMEGF2qElu0X/ly1dh9Zex3sVzWw1WGvkItccaThU7gq+hWUv9MO9/EuqP6+Drh
1a1tIv/8Kgk4OKfmn3o16UoXczv6O2Jaw9UtivrYUhL52K+/HF4p3bTnW2fo9p3C
EbY92AdMdtyaWxxylFPd8lWv72a5S2wIL1XiUdFgJVbojVnj1v7VmG+R68AnrMqc
03kXCkHmxaWf76XI4e/CpxA8Ms46Dw8yvgfqHD1xBeK8j3jwgSWYu6G11zDKaxYb
FG0UQB4G5uC89wC4GHb3Yl5AU0jKpdkPpQ+7VjMnuf4YM0dYuuW5ljOma+vkvLJo
KrhDT9zJvixYqKjJV8FEwNRLjLPvMqUQt0OzkMvpJsI/lEgfXbZtXNdOqCNsfUS9
OqlNkIzWipDRdMu+H7eoPMMuT3OBx2TYnYAkTLNY5FOb+H4zL3hJ2gOntrDRAoYX
v4ONlrx/ityx1ff/3f9V31t2rxBoGIh/WEyp1Xla9CZ//aU6cyECggEBAPxlRg6C
MTRqmTc5yZMj+MdBS+nFt8e+2hXVWxWP0BldL4UL8FIMXZl7iKl+vL0QvuLtA8H7
/L/+0yxpuPYKvysXjDagemruYM+6hf+1CQjGTob6xiYdQDGHJR77d7b/7nLiAywF
55nb6ldELt1SiAuarV1tBYYMl23djGwliI/jw/vduhHKfTPArejIvY3UIeBb4MyP
fDZQHzQSEz7rqsoOCS+YW7eJjSZBoboH9BxXbmnS0hcWY/93O0jW2tsXS80DIH1w
/8PxAZXC8Tx/aFIh3U2oFqrRtI5HitHEXmzeH9QgBmFN1tQM51uqGXy7iNJJ42/r
pU9RJnahea5am40CggEBAOeB7SxKJXsaDDm6DM3wDtgatsUSHnWLgRVM+w8vjUbP
A2X1pOLob3q2AnBsKsSLyEqPPnJtxY+pCWXrGcUdY8T3AQu4qrdG1cfnhcO9vr4J
D12VLDKgc0VluG6WXmnUxpaDSJSlaxEkC4ob1uSCLmVVwXTykb+1FN8ALvZ7gqsZ
Xx+2eIDA8Q4yiJCWIdh9cw8UB4Y5A0/5DLiHzwVDyIVSI2UVywlIsG7f1yMqLLnj
elywGd9M9RfhA1vlkq0xANpkPjf1slJejA8udxULOhvAB+D8gexZM3E7rWkeUhAA
GeA0IKrSiVq7w9BZwIaiDeAl+EDB6Yx0YL2OS1tbd08CggEBAJG1udMocFWtcNTE
JM4whzApTzUfszC6ytat/M6e3um1tXhe4xfLEcJI4iXQCBAyebEyWegEnO8Oo0VR
p7lLjP/IsWbcELBo/DUQnsl370ZEjhM49WoNmwC8375aOMvKRxPFzUT+B634mvpU
X6XgPrBMVOfJx9WSsV14+Nd6d7NyL7PZVtFskAlvpDbupRgbLo+DG22dLH2ClfEO
ulOp5EiP/9edYt27qGQ8oaxvaQtvokuvrSAtCf/zCqPLfj/b3CP4OSqSMOYi3eJi
z7g4abZ43HU3pBZryIwXW1osvsKlHcTt1CEaufpjlLGDqwEqlxuoEoiLhz0aQVVd
dY0lvAUCggEBALG6rGMpFTc5mxMiQzxCxJKhh5kpvNqO2+2HaOKSpgorWTeIayqM
OTFi0+KNGBRGH+ElsVJV9arBoeZtpB4Q3wxSeKoP/nev2OWcV7QbUnlAKVy17fV7
+qLXYcz8gcULxd29MhZ0HAtPudAwaTyKuKWxPVDT/JLJqRk+Yc92qK1EUCPfiQmH
lkhJAVDHAXrbbF6yCMjBskpOL7bnBEbNb/7yPRwYrAQXmuOz0s07TpTzD3hi9anZ
wfuwEk0VpRJzIW2IMb/yTxEvZqUtDdzI/rZZKXNPR0s0e+q9XvbpgSSpfzQBsOaT
tUFEDyNAFC8H8FEZtUm51NuwaKh9ulqLkL8CggEBAINxIsHvYOPaMchRz2vwBhvj
SWFWxrWUYmUoV7No5EbJhpQl4CEN5FX6oS6yj4X+5aKT3P/W5WdQ0dph/K4vhrD2
ChbmXZ/y63xghCUjAU8rEfTIsu/6u36qxf7D4UAHwHPvB5Jx0to7HhkRFfXw1f16
NEZe+vhMIFRFvaipj95UNHuMnbmuaTBFxmgfRsbsg9AQorr5Ky/SAKROrXUfbyx4
wLerPPfMvrv3EmP8UJtSmXNkncStfBrzgOY0yZUMmh5ML8wK5YfrlQ0nc769okCl
VtqnfOK0g+1YGqdX1m5dxlqTVYGAQLQboLyhJm9xnoLDfqlHuQ+1eJhGyaZ5qI8=
-----END RSA PRIVATE KEY-----

fuska@s0rr0w:~/Wargames/s4ur0n2024$ sftp -i a.pem santa@xmas2024.s4ur0n.com
Connected to xmas2024.s4ur0n.com.
sftp> ls
README  grinch
```

Nivel 3

Recuperamos el fichero “grinch” mediante SFTP.

```
fuska@s0rr0w:~/Wargames/s4ur0n2024$ file grinch
grinch: raw G3 (Group 3) FAX
```

Se trata de una imagen G3 Fax. En Debian (seguro que en más distros) está el paquete "mgetty-viewfax", que provee el comando "viewfax". Si lo usamos para ver la imagen, vemos la URL del nivel 4:



Nivel 4

Descargamos el fichero XM24g4me y tenemos que es una ROM de GameBoy:

```
fuska@s0rr0w:~/Wargames/s4ur0n2024$ file XM24g4me
XM24g4me: Game Boy ROM image (Rev.01) [ROM ONLY], ROM: 256Kbit
```

Le paso el scalpel por si hubiese más archivos que no aparezcan a simple vista y detecta un GIF:

```
fuska@s0rr0w:~/Wargames/s4ur0n2024$ scalpel XM24g4me
Scalpel version 1.60
Written by Golden G. Richard III, based on Foremost 0.69.

Opening target "/home/fuska/Wargames/s4ur0n2024/XM24g4me"

Image file pass 1/2.
XM24g4me: 100.0%
|*************************************************************************************
*************************************************************************|    1.1
MB    00:00
...
XM24g4me: 100.0%
|*************************************************************************************
*************************************************************************|    1.1
MB    00:00 ETAProcessing of image file complete. Cleaning up...
Done.
Scalpel is done, files carved = 1, elapsed = 0 seconds.
fuska@s0rr0w:~/Wargames/s4ur0n2024$ cat scalpel-output/
audit.txt  gif-3-0/
fuska@s0rr0w:~/Wargames/s4ur0n2024$ cat scalpel-output/audit.txt

Scalpel version 1.60 audit file
Started at Fri Dec 27 12:33:54 2024
Command line:
scalpel XM24g4me

Output directory: /home/fuska/Wargames/s4ur0n2024/scalpel-output
Configuration file: /etc/scalpel/scalpel.conf

Opening target "/home/fuska/Wargames/s4ur0n2024/XM24g4me"

The following files were carved:
File              Start                Chop            Length          Extracted
From
00000000.gif        32768              NO            72561             XM24g4me
```

El GIF extraído parece estar corrupto, pero sacando una captura de pantalla mientras lo visualizamos en un editor, podemos sacar el QR con herramientas como zbaring:

El QR dice “Connect by ssh (user: level6, private key from #2)

```
fuska@s0rr0w:~/Wargames/s4ur0n2024/scalpel-output/gif-3-0$ zbarimg a.png
QR-Code:Connect by ssh (user: level6, private key from #2)
scanned 1 barcode symbols from 1 images in 0,04 seconds
```

Y siguiendo esas instrucciones:

```
fuska@s0rr0w:~/Wargames/s4ur0n2024$ ssh -i ./a.pem level6@xmas2024.s4ur0n.com
Linux xmas2024 6.1.0-28-amd64 #1 SMP PREEMPT_DYNAMIC Debian 6.1.119-1 (2024-11-22)
x86_64
```

```
  You are in my house... Merry XMAS!!!
Last login: Fri Dec 27 11:58:09 2024 from X.X.X.X
-bash-5.2$ ls -ltr
total 12484
-r--r----- 1 level6 level6 1267091 Dec 20 19:23 13d5c3ed-1337-4cde-a061-
9a429344a793.png
-r--r----- 1 level6 level6 1269123 Dec 20 19:23 296dd470-1337-485d-8481-
afcff05720bb.png
-r--r----- 1 level6 level6 1275276 Dec 20 19:23 a1277b71-1337-434f-abe2-
6cfa8dda6d40.png
-r--r----- 1 level6 level6 1274665 Dec 20 19:23 ae38bba9-1337-4546-a288-
1efaec2c9add.png
-r--r----- 1 level6 level6 1266573 Dec 20 19:23 bd6ab787-1337-4586-8745-
7d06d7d58f79.png
-r--r----- 1 level6 level6 1275926 Dec 20 19:23 ce18234d-1337-435d-a60a-
e82c1f5f65a6.png
-r--r----- 1 level6 level6 1258944 Dec 20 19:23 cec8906e-1337-4d9b-b6c9-
9563b9b963de.png
-r--r----- 1 level6 level6 1269528 Dec 20 19:23 f73ce5e6-1337-4091-81c6-
be4ab4f456d2.png
-rw-r--r-- 1 level6 level6 2576419 Dec 23 22:32 archivo_hex.txt
-rw-r--r-- 1 level6 level6       1 Dec 24 18:37 archivo.bin
-rw-r--r-- 1 level6 level6       1 Dec 24 19:41 13
-rw-r--r-- 1 level6 level6       1 Dec 24 19:41 29
-rw-r--r-- 1 level6 level6       1 Dec 24 19:41 a1
-rw-r--r-- 1 level6 level6       1 Dec 24 19:41 ae
-rw-r--r-- 1 level6 level6       1 Dec 24 19:41 bd
-rw-r--r-- 1 level6 level6       1 Dec 24 19:41 ce1
-bash-5.2$
```

Nivel 6

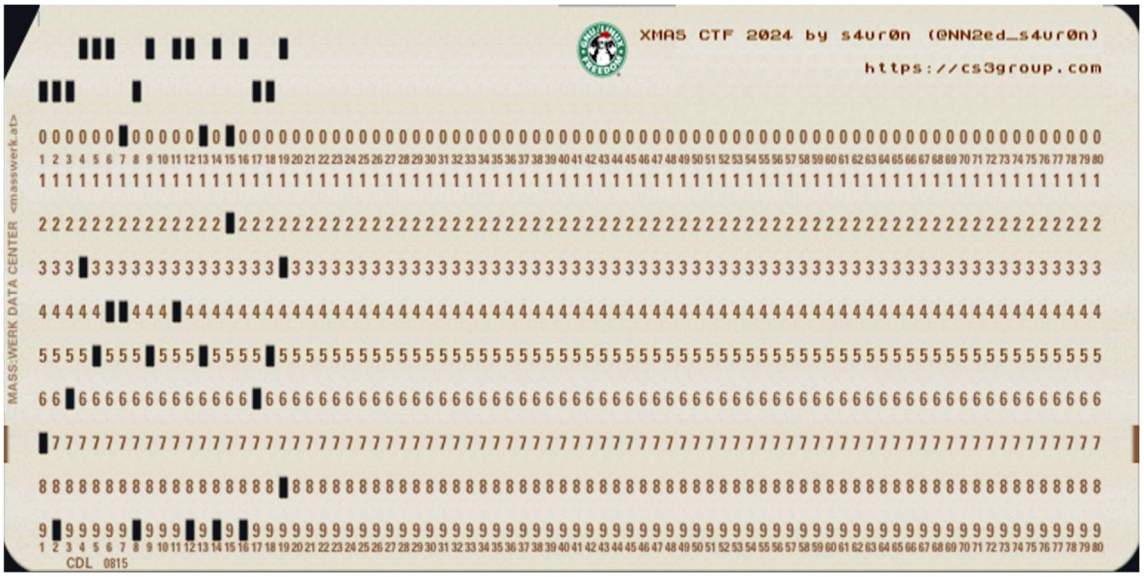
Probamos a recuperar los ficheros del servidor, pero:

```
fuska@s0rr0w:~/Wargames/s4ur0n2024/level6$ scp -i ./a.pem
level6@xmas2024.s4ur0n.com:./* ./
scp: Connection closed
```

Parece que el servidor no soporta el subsistema SFTP y necesitamos usar el protocolo SCP:

```
fuska@s0rr0w:~/Wargames/s4ur0n2024/level6$ ls -ltr
total 4
-rw------- 1 fuska fuska 3247 dic 27 13:07 a.pem
fuska@s0rr0w:~/Wargames/s4ur0n2024/level6$ scp -O -i ./a.pem
level6@xmas2024.s4ur0n.com:./*.png ./
13d5c3ed-1337-4cde-a061-9a429344a793.png                100% 1237KB 634.1KB/s
00:01
296dd470-1337-485d-8481-afcff05720bb.png                100% 1239KB 635.1KB/s
00:01
a1277b71-1337-434f-abe2-6cfa8dda6d40.png                100% 1245KB 628.3KB/s
00:01
ae38bba9-1337-4546-a288-1efaec2c9add.png                100% 1245KB 628.0KB/s
00:01
bd6ab787-1337-4586-8745-7d06d7d58f79.png                100% 1237KB 623.9KB/s
00:01
ce18234d-1337-435d-a60a-e82c1f5f65a6.png                100% 1246KB 628.8KB/s
00:01
cec8906e-1337-4d9b-b6c9-9563b9b963de.png                100% 1229KB 629.9KB/s
00:01
f73ce5e6-1337-4091-81c6-be4ab4f456d2.png                100% 1240KB 625.5KB/s
00:01
fuska@s0rr0w:~/Wargames/s4ur0n2024/level6$ ls -ltr
total 9940
-rw------- 1 fuska fuska    3247 dic 27 13:07 a.pem
-r--r----- 1 fuska fuska 1267091 dic 27 13:17 13d5c3ed-1337-4cde-a061-
9a429344a793.png
-r--r----- 1 fuska fuska 1269123 dic 27 13:17 296dd470-1337-485d-8481-
afcff05720bb.png
-r--r----- 1 fuska fuska 1275276 dic 27 13:17 a1277b71-1337-434f-abe2-
6cfa8dda6d40.png
-r--r----- 1 fuska fuska 1274665 dic 27 13:17 ae38bba9-1337-4546-a288-
1efaec2c9add.png
-r--r----- 1 fuska fuska 1266573 dic 27 13:17 bd6ab787-1337-4586-8745-
7d06d7d58f79.png
-r--r----- 1 fuska fuska 1275926 dic 27 13:17 ce18234d-1337-435d-a60a-
e82c1f5f65a6.png
-r--r----- 1 fuska fuska 1258944 dic 27 13:17 cec8906e-1337-4d9b-b6c9-
9563b9b963de.png
-r--r----- 1 fuska fuska 1269528 dic 27 13:18 f73ce5e6-1337-4091-81c6-
be4ab4f456d2.png
fuska@s0rr0w:~/Wargames/s4ur0n2024/level6$
```

Las imágenes son tarjetas perforadas:



Aquí estuve un buen rato intentando modificar las imágenes y pasarlas por varios sitios online que supuestamente las decodificaban automáticamente… pero no hubo suerte. Así que lo tuve que hacer de forma manual usando https://www.masswerk.at/keypunch/ . El resultado fue:

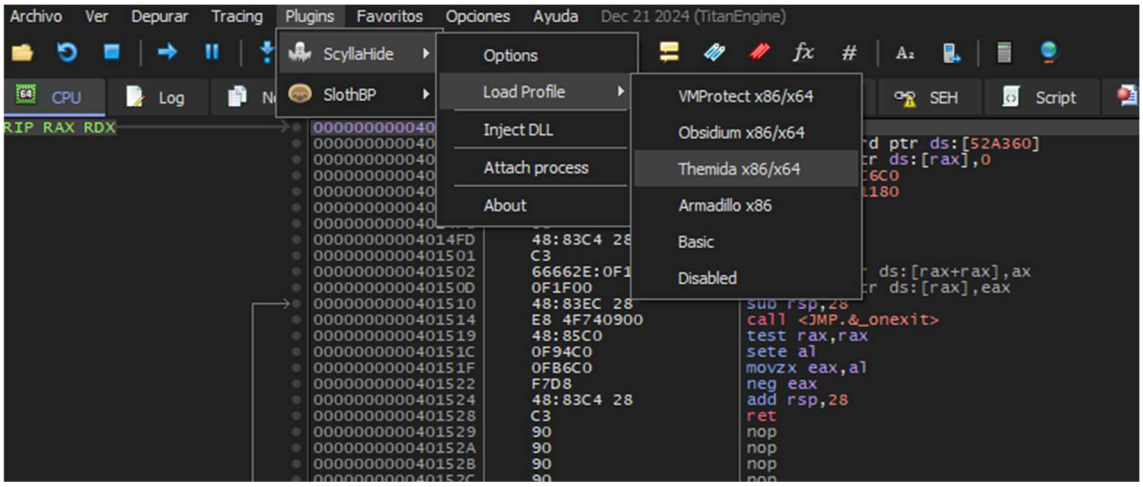
```
Procedure division.
identification.
display "xmas2024.s4ur0n.com/l3v".
display "nextic ic is reversing".
program=id. level6c
display "download it https:".
stop run.
display "3l7xmas24ctfr3v".
```

Lo que nos da la url:

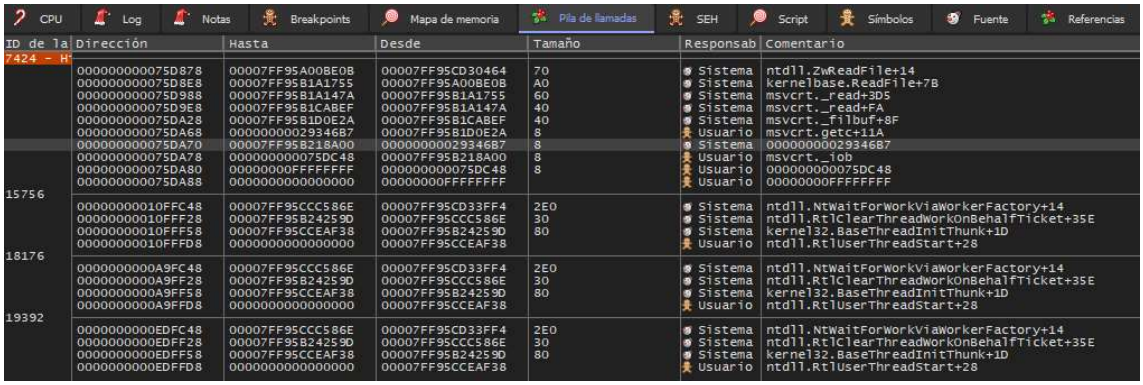
https://xmas2024.s4ur0n.com/l3v3l7xmas24ctfr3v

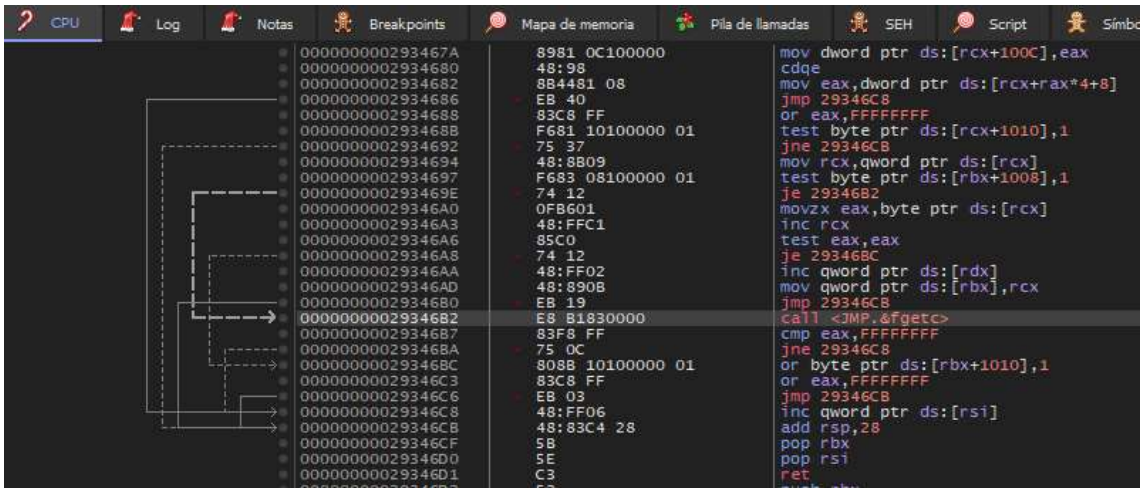
Nivel 7

Nos descargamos el binario y le echamos un vistazo con el xdbg. Tiene antidebugging, así que cargamos el ScyllaHide y lanzamos el programa:

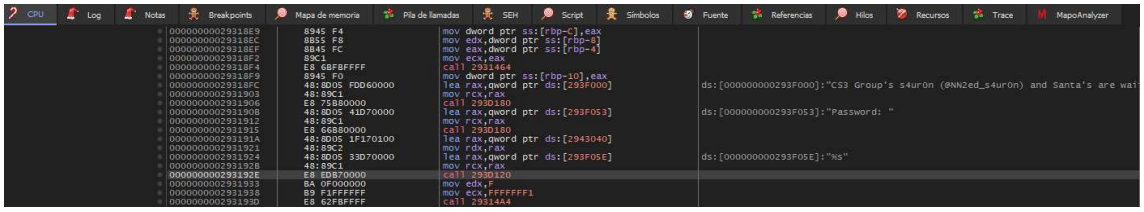


Al pararse el programa para recibir una entrada de usuario, no nos hace falta meter breakpoints de momento. Vamos a la pila de llamadas para buscar alguna función de entrada de caracteres y vemos el getc llamado desde 0x29346B7

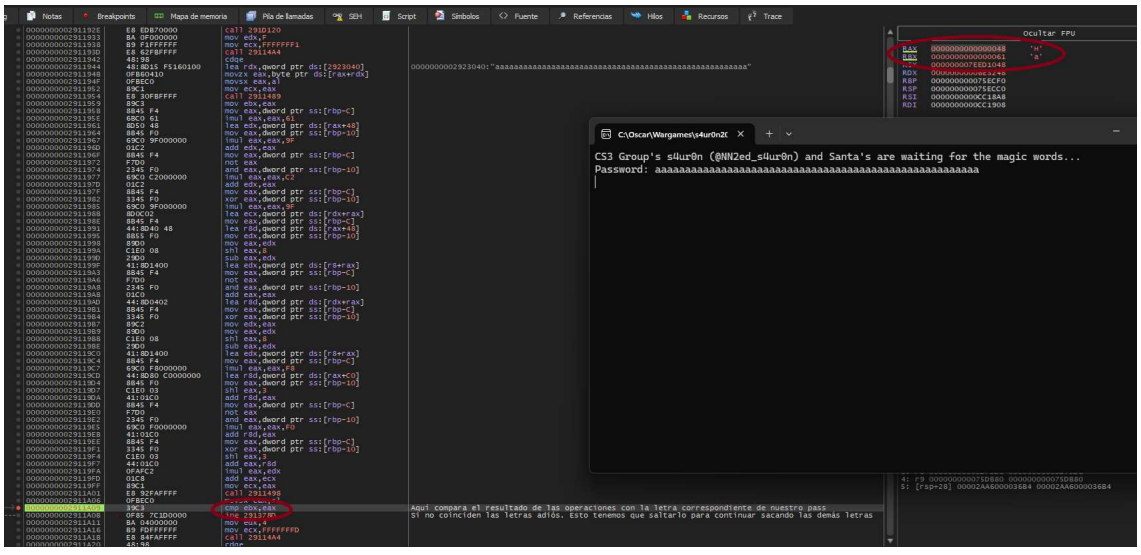




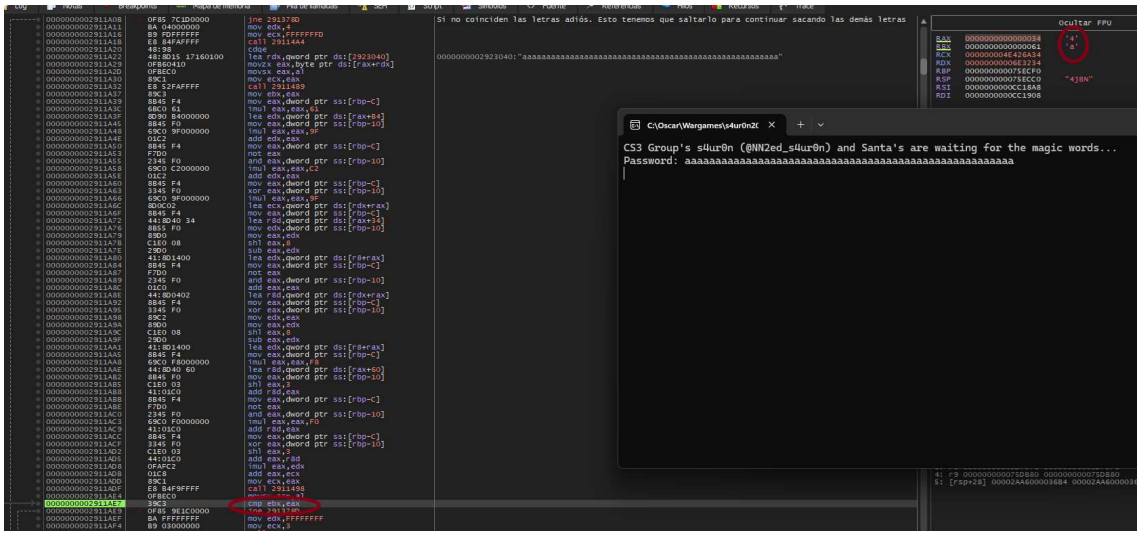
Ponemos un punto de interrupción justo en el retorno de la llamada y metemos una password de prueba (“aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa”, por ejemplo) en la ventana de la aplicación. Si vamos depurando llegamos a este punto del programa donde se recupera la password, se hace una serie de operaciones con una cadena para prepararla y luego, por cada una de las letras de la password que hemos metido, se hacen otra serie de operaciones y se compara con el la letra correspondiente de la clave correcta:



Si vamos pasando por todas las comprobaciones podemos ver la letra correspondiente a la password que hemos metido (en este caso una “a”), y la letra correcta (en este caso la “H”)

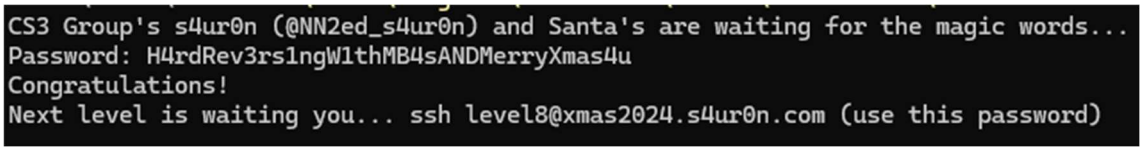


Nos saltamos el jne y avanzamos a la siguiente comparación



Y así con todas las letras de la password correcta:

H4rdRev3rs1ngW1thMB4sANDMerryXmas4u



Nivel 8

```
Linux xmas2024 6.1.0-28-amd64 #1 SMP PREEMPT_DYNAMIC Debian 6.1.119-1 (2024-11-22)
x86_64
```

```
  You are in my house... Merry XMAS!!!
Last login: Sat Jan  4 00:15:56 2025 from X.X.X.X
-bash-5.2$ ls -las
total 16
4 drwx------ 2 level8 level8 4096 Dec 24 01:13 .
4 drwxr-xr-x 3 root   root   4096 Dec 21 21:29 ..
4 -rw------- 1 level8 level8  507 Jan  4 00:19 .bash_history
4 -rw-r--r-- 1 level8 level8  281 Dec 21 21:39 README
-bash-5.2$ ls -las
total 16
4 drwx------ 2 level8 level8 4096 Dec 24 01:13 .
4 drwxr-xr-x 3 root   root   4096 Dec 21 21:29 ..
4 -rw------- 1 level8 level8  507 Jan  4 00:19 .bash_history
4 -rw-r--r-- 1 level8 level8  281 Dec 21 21:39 README
-bash-5.2$ cat README
Welcome home again... but Santa hasn't left any gifts yet.

Are you waiting for the exploiting levels?

It's time to fuck around a bit more before we get to them...

Level 9 is located at
https://xmas2024.s4ur0n.com/ea6a958571c0450486dd1c842326fefe1a4fc03c43c60efcbbae72966
c91e8cf
-bash-5.2$
```

Nivel 9

La URL que nos dan muestra un video que, según el código fuente de la página está en el directorio “assets/”. Pruebo por si hay suerte y si, tiene abierto el listado de archivos.

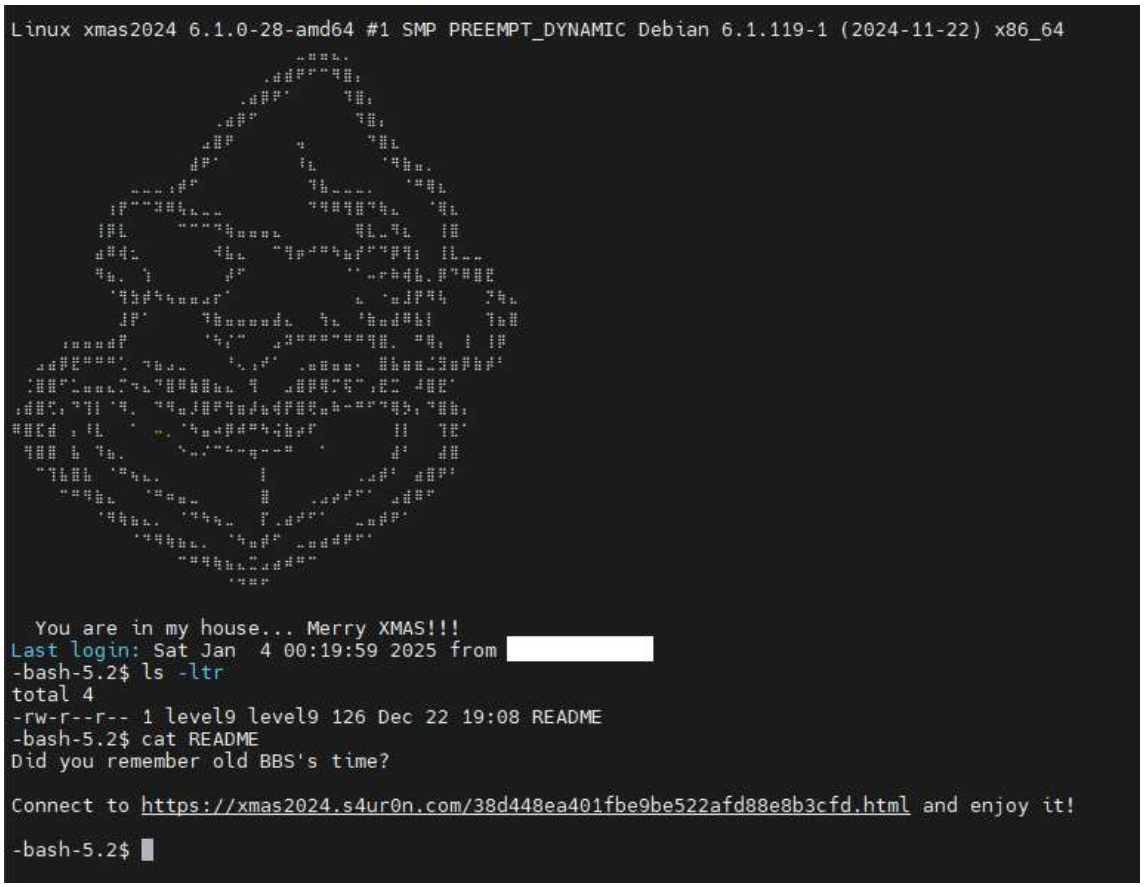
Index of /ea6a958571c0450486dd1c842326fefe1a4fc03c43c60efcbbae72966c91e8cf/assets

Name	Last modified	Size	Description
Parent Directory		-	
frame-01.png	2024-12-21 21:41	1.5M	
frame-02.png	2024-12-21 21:41	1.5M	
frame-03.png	2024-12-21 21:41	1.5M	
xmas2024.mp4	2024-12-21 21:41	33M	

Apache/2.4.62 (Debian) Server at xmas2024.s4ur0n.com Port 443

El frame2 y el frame3 nos dan la solución para continuar:

Conectamos a xmas2024.s4ur0n.com con el usuario level9 y la clave privada que ya teníamos.



Y vamos a la URL:

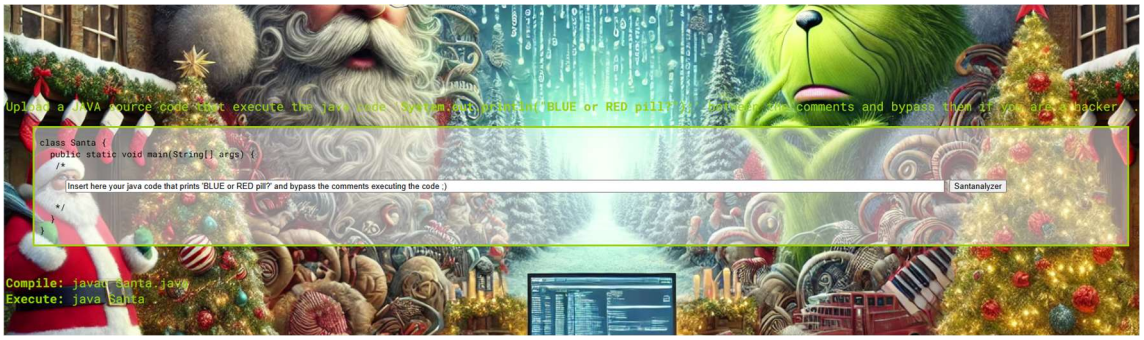
https://xmas2024.s4ur0n.com/38d448ea401fbe9be522afd88e8b3cfd.html

Tenemos un archivo de audio con el sonido de un modem. Uso minimodem para decodificarlo. Como en la imagen de la web aparece un 300, pruebo primero por si estuviese a 300bps y bingo:

```
fuska@s0rr0w:~/Wargames/s4ur0n2024$ minimodem -r -f santa.wav 300
### CARRIER 300 @ 1250.0 Hz ###
SG8hIEhvISBIbyEgQ29uZ3JhdHMgbXkgZnJpZW5kISBJIHdpbGwgaWdub3JlIHlvdXIgR29vZ2xlIGFuZCBQb
3JuaHViIGhpc3Rvcnkgd2hlbiByZWFkaW5nIHlvdXIgbGV0dGVyLiBJJ20gdmVyeSBidXN5IHJpZ2h0IG5vdy
BiZWNhdXNlIEkgaGF2ZSB0byBmZWVkIFJ1ZG9scGgsIERvbm5lciwgQmxpdHplbiwgVml4ZW4sIEN1cGlkLCB
Db21ldCwgRGFzaGVyLCBEYW5jZXIgYW5kIFByYW5jZXIuIEl0IG9ubHkgcmVtYWlucyBmb3IgbWUgdG8gY29u
dmV5IG15IGNvbmdyYXR1bGF0aW9ucyBhbmQgd2lzaCB5b3UgZXZlcnkgc3VjY2VzcyBmb3IgeW91ciBuZXh0I
GxldmVsIHRoYXQgaXMgYXQgaHR0cHM6Ly94bWFzMjAyNC5zNHVyMG4uY29tL2xldmVsMTAvZjc1MTcwYzM0MT
JmMGRkMjYxMDNjMGNjNGY0MjhlMDY3MTc1MjgzNi5odG1s

### NOCARRIER ndata=557 confidence=2.355 ampl=0.993 bps=300.00 (rate perfect) ###
fuska@s0rr0w:~/Wargames/s4ur0n2024$ minimodem -r -f santa.wav 300 | base64 -d
### CARRIER 300 @ 1250.0 Hz ###

### NOCARRIER ndata=557 confidence=2.355 ampl=0.993 bps=300.00 (rate perfect) ###
Ho! Ho! Ho! Congrats my friend! I will ignore your Google and Pornhub history when
reading your letter. I'm very busy right now because I have to feed Rudolph, Donner,
Blitzen, Vixen, Cupid, Comet, Dasher, Dancer and Prancer. It only remains for me to
convey my congratulations and wish you every success for your next level that is at
https://xmas2024.s4ur0n.com/level10/f75170c3412f0dd26103c0cc4f428e0671752836.html
fuska@s0rr0w:~/Wargames/s4ur0n2024$
```

Nivel 10

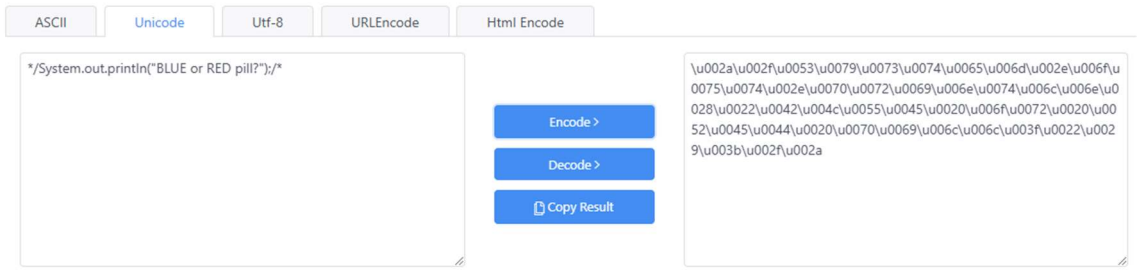
URL:
https://xmas2024.s4ur0n.com/level10/f75170c3412f0dd26103c0cc4f428e0671752836.html



Al entrar en la página tenemos la definición de una clase Java en la que todo el main está comentado (entre /* y */). El objetivo es inyectar el código dentro de ese comentario que, al ejecutarse, muestre por pantalla el texto "BLUE or RED pill?". Típicamente lo que está comentado no se ejecuta, pero Java tiene un comportamiento “especial” a la hora de parsear los ficheros antes de pasarlos al compilador, y es que en caso de existir caracteres Unicode, los procesa antes de pasar a la compilación. No tenía ni idea de este comportamiento hasta ahora, la verdad…. Podéis ver más sobre ello en https://programming.guide/java/executing-code-in-comments.html y en la sección 3.3 del Java Language Specification (https://docs.oracle.com/javase/specs/jls/se8/html/jls-3.html#jls-3.3)

Lo obvio es intentar inyectar el código ‘*/System.out.println("BLUE or RED pill?");/*’ para cerrar el comentario previo, sacar el mensaje por pantalla, y abrir comentario para no romper la parte de la clase que nos viene ya dada.

Para hacer la codificación a Unicode usé la web https://checkserp.com/encode/unicode/ pero vale cualquiera:



“\u002a\u002f\u0053\u0079\u0073\u0074\u0065\u006d\u002e\u006f\u0075\u0074\u002e\u0070\u0072\u0069\u006e\u0074\u006c\u006e\u0028\u0022\u0042\u004c\u0055\u0045\u0020\u006f\u0072\u0020\u0052\u0045\u0044\u0020\u0070\u0069\u006c\u006c\u003f\u0022\u0029\u003b\u002f\u002a”

Y nos vamos al siguiente nivel:

Nivel 5

https://xmas2024.s4ur0n.com/level5/xmas2024-vm.zip

Se trata de un binario junto con un fichero ".hex" con algunos bytes que no nos dicen mucho y dos cadenas de texto: "Well done!!!" y "Ops!".

El binario es un elf que implementa una máquina virtual. Recibe como parámetros el fichero con los opcodes a ejecutar en la VM (en el fichero también van cadenas) y una password que más adelante veremos para qué sirve.

Primeramente, lee el parámetro password que hemos pasado por línea de comandos. Su longitud debe estar entre 0x7h y 0x20h. Si no muestra el error “2024 XMAS CTF VM result: Your password length is incorrect”

```
.text:0804A10B loc_804A10B:                            ; CODE XREF: la_chicha+28↑j
.text:0804A10B                 mov     eax, [esi+4]
.text:0804A10E                 add     eax, 8
.text:0804A111                 mov     eax, [eax]
.text:0804A113                 sub     esp, 0Ch
.text:0804A116                 push    eax
.text:0804A117                 call    leer_puntero
.text:0804A11C                 add     esp, 10h
.text:0804A11F                 mov     [ebp+password], eax
.text:0804A122                 sub     esp, 0Ch
.text:0804A125                 push    [ebp+password]
.text:0804A128                 call    strlen
.text:0804A12D                 add     esp, 10h
.text:0804A130                 cmp     eax, 7
.text:0804A133                 jbe     short loc_804A148
.text:0804A135                 sub     esp, 0Ch
.text:0804A138                 push    [ebp+password]
.text:0804A13B                 call    strlen
.text:0804A140                 add     esp, 10h
.text:0804A143                 cmp     eax, 20h ; ' '
.text:0804A146                 jbe     short loc_804A164
.text:0804A148
.text:0804A148 loc_804A148:                            ; CODE XREF: la_chicha+7F↑j
.text:0804A148                 sub     esp, 0Ch
.text:0804A14B                 lea     eax, (a2024XmasCtfVmR - 80EDFF4h)[ebx] ; "2024 XMAS CTF VM result: Your password "...
.text:0804A151                 push    eax
.text:0804A152                 call    print
.text:0804A157                 add     esp, 10h
.text:0804A15A                 sub     esp, 0Ch
.text:0804A15D                 push    0FFFFFFFFh
.text:0804A15F                 call    return
```

A continuación, llama a la función en 0x08049E20 que hace una serie de XORs con los bytes de la clave que hemos metido por parámetros:

```
.text:08049E20 manipula_la_password proc near          ; CODE XREF: la_chicha+CE↓p
.text:08049E20
.text:08049E20 var_4           = dword ptr -4
.text:08049E20 arg_0           = dword ptr  8
.text:08049E20
.text:08049E20 ; __unwind {
.text:08049E20                 push    ebp
.text:08049E21                 mov     ebp, esp
.text:08049E23                 sub     esp, 10h
.text:08049E26                 call    sub_804A2F4
.text:08049E2B                 add     eax, (offset tbyte_80EDFF4 - $)
.text:08049E30                 mov     eax, [ebp+arg_0]
.text:08049E33                 movzx   edx, byte ptr ds:(tbyte_80EDFF4 - 80EDFF4h)[eax]
.text:08049E36                 mov     eax, [ebp+arg_0]
.text:08049E39                 add     eax, 4
.text:08049E3C                 movzx   eax, byte ptr ds:(tbyte_80EDFF4 - 80EDFF4h)[eax]
.text:08049E3F                 xor     eax, edx
.text:08049E41                 movsx   eax, al
.text:08049E44                 shl     eax, 18h
.text:08049E47                 mov     [ebp+var_4], eax
.text:08049E4A                 mov     eax, [ebp+arg_0]
.text:08049E4D                 add     eax, 1
.text:08049E50                 movzx   edx, byte ptr ds:(tbyte_80EDFF4 - 80EDFF4h)[eax]
.text:08049E53                 mov     eax, [ebp+arg_0]
.text:08049E56                 add     eax, 5
.text:08049E59                 movzx   eax, byte ptr ds:(tbyte_80EDFF4 - 80EDFF4h)[eax]
.text:08049E5C                 xor     eax, edx
.text:08049E5E                 movsx   eax, al
.text:08049E61                 shl     eax, 10h
.text:08049E64                 add     [ebp+var_4], eax
.text:08049E67                 mov     eax, [ebp+arg_0]
.text:08049E6A                 add     eax, 2
.text:08049E6D                 movzx   edx, byte ptr ds:(tbyte_80EDFF4 - 80EDFF4h)[eax]
.text:08049E70                 mov     eax, [ebp+arg_0]
.text:08049E73                 add     eax, 6
.text:08049E76                 movzx   eax, byte ptr ds:(tbyte_80EDFF4 - 80EDFF4h)[eax]
.text:08049E79                 xor     eax, edx
.text:08049E7B                 movsx   eax, al
.text:08049E7E                 shl     eax, 8
.text:08049E81                 add     [ebp+var_4], eax
.text:08049E84                 mov     eax, [ebp+arg_0]
.text:08049E87                 add     eax, 3
.text:08049E8A                 movzx   edx, byte ptr ds:(tbyte_80EDFF4 - 80EDFF4h)[eax]
.text:08049E8D                 mov     eax, [ebp+arg_0]
.text:08049E90                 add     eax, 7
.text:08049E93                 movzx   eax, byte ptr ds:(tbyte_80EDFF4 - 80EDFF4h)[eax]
.text:08049E96                 xor     eax, edx
.text:08049E98                 movsx   eax, al
.text:08049E9B                 add     [ebp+var_4], eax
.text:08049E9E                 mov     eax, [ebp+var_4]
.text:08049EA1                 leave
.text:08049EA2                 retn
```

En pseudocódigo vendría a ser algo así:

```
bytes = (clave[3]^clave[7])+((clave[2]^clave[6]) << 8)+((clave[1]^clave[5]) << 16)+
((clave[0]^clave[4]) << 24)
```

Seguimos:

```
.text:0804A187                  add     esp, 10h
.text:0804A18A                  mov     [ebp+clave_manipulada], eax
.text:0804A18D                  sub     esp, 0Ch
.text:0804A190                  push    1200h
.text:0804A195                  call    malloc
.text:0804A19A                  add     esp, 10h
.text:0804A19D                  mov     [ebp+contenido_fichero], eax
.text:0804A1A0                  sub     esp, 0Ch
.text:0804A1A3                  push    0Eh
.text:0804A1A5                  call    malloc
.text:0804A1AA                  add     esp, 10h
.text:0804A1AD                  mov     [ebp+pila_de_la_vm], eax
.text:0804A1B0                  mov     eax, [ebp+pila_de_la_vm]
.text:0804A1B3                  mov     word ptr [eax+0Ah], 0
.text:0804A1B9                  mov     eax, [ebp+pila_de_la_vm]
.text:0804A1BC                  mov     word ptr [eax+0Ch], 100h
.text:0804A1C2                  mov     eax, [ebp+pila_de_la_vm]
.text:0804A1C5                  mov     byte ptr [eax+8], 0
.text:0804A1C9                  mov     eax, [esi+4]
.text:0804A1CC                  add     eax, 4
.text:0804A1CF                  mov     eax, [eax]
.text:0804A1D1                  sub     esp, 0Ch
.text:0804A1D4                  push    eax
.text:0804A1D5                  call    leer_puntero
.text:0804A1DA                  add     esp, 10h
.text:0804A1DD                  mov     edx, eax
.text:0804A1DF                  sub     esp, 8
.text:0804A1E2                  lea     eax, (aRb - 80EDFF4h)[ebx] ; "rb"
.text:0804A1E8                  push    eax
.text:0804A1E9                  push    edx
.text:0804A1EA                  call    fopen
```

Aquí prepara las variables que le servirán para guardar el contenido del fichero .hex y la pila de la máquina virtual (inicializándola antes). Luego abre el fichero .hex.

```
.text:0804A217 loc_804A217:                             ; CODE XREF: sub_804A0B4+145↑j
.text:0804A217                  sub     esp, 4
.text:0804A21A                  push    2
.text:0804A21C                  push    0
.text:0804A21E                  push    [ebp+file_handle]
.text:0804A221                  call    sub_805C740
.text:0804A226                  add     esp, 10h
.text:0804A229                  sub     esp, 0Ch
.text:0804A22C                  push    [ebp+file_handle]
.text:0804A22F                  call    fsize
.text:0804A234                  add     esp, 10h
.text:0804A237                  mov     [ebp+file_size], eax
.text:0804A23A                  mov     eax, [ebp+file_size]
.text:0804A23D                  cmp     eax, 1000h
.text:0804A242                  jbe     short loc_804A260
.text:0804A244                  sub     esp, 0Ch
.text:0804A247                  lea     eax, (aErrorCodeAndDa - 80EDFF4h)[ebx] ; "*ERROR*\nCode and data is larger than t"...
.text:0804A24D                  push    eax
.text:0804A24E                  call    print
.text:0804A253                  add     esp, 10h
.text:0804A256                  mov     eax, 0FFFFFFFDh
.text:0804A25B                  jmp     loc_804A2E9
```

Si el fichero es mayor de 1000h bytes sale del programa con error.

```
.text:0804A260 loc_804A260:                            ; CODE XREF: sub_804A0B4+18E↑j
.text:0804A260                  sub     esp, 0Ch
.text:0804A263                  push    [ebp+file_handle]
.text:0804A266                  call    sub_805C820
.text:0804A26B                  add     esp, 10h
.text:0804A26E                  mov     edx, [ebp+file_size]
.text:0804A271                  mov     eax, [ebp+contenido_fichero]
.text:0804A274                  push    [ebp+file_handle]
.text:0804A277                  push    edx
.text:0804A278                  push    1
.text:0804A27A                  push    eax
.text:0804A27B                  call    fread
.text:0804A280                  add     esp, 10h
.text:0804A283                  sub     esp, 0Ch
.text:0804A286                  push    [ebp+file_handle]
.text:0804A289                  call    sub_8058AB0
.text:0804A28E                  add     esp, 10h
.text:0804A291                  mov     eax, [ebp+clave_manipulada]
.text:0804A294                  shr     eax, 10h
.text:0804A297                  mov     edx, eax
.text:0804A299                  mov     eax, [ebp+contenido_fichero]
.text:0804A29C                  mov     [eax+4], dl
.text:0804A29F                  mov     eax, [ebp+clave_manipulada]
.text:0804A2A2                  shr     eax, 18h
.text:0804A2A5                  mov     edx, eax
.text:0804A2A7                  mov     eax, [ebp+contenido_fichero]
.text:0804A2AA                  mov     [eax+5], dl
.text:0804A2AD                  mov     eax, [ebp+clave_manipulada]
.text:0804A2B0                  mov     edx, eax
.text:0804A2B2                  mov     eax, [ebp+contenido_fichero]
.text:0804A2B5                  mov     [eax+10h], dl
.text:0804A2B8                  mov     eax, [ebp+clave_manipulada]
.text:0804A2BB                  shr     eax, 8
.text:0804A2BE                  mov     edx, eax
.text:0804A2C0                  mov     eax, [ebp+contenido_fichero]
.text:0804A2C3                  mov     [eax+11h], dl
.text:0804A2C6                  sub     esp, 8
.text:0804A2C9                  push    [ebp+pila_de_la_vm]
.text:0804A2CC                  push    [ebp+contenido_fichero]
.text:0804A2CF                  call    maquina_virtual
.text:0804A2D4                  add     esp, 10h
.text:0804A2D7                  sub     esp, 0Ch
.text:0804A2DA                  push    0Ah
.text:0804A2DC                  call    sub_8059710
.text:0804A2E1                  add     esp, 10h
.text:0804A2E4                  mov     eax, 0
```

En el siguiente bloque lee el fichero hex y mete los valores resultantes de los XORs que ha hecho antes sobre la clave en las posiciones 4, 5, 10h y 11h del fichero, machacando lo que había anteriormente (que siempre son los bytes “de ad”). Una vez parcheadas las instrucciones que le vamos a pasar a la máquina virtual, se llama a su rutina principal.

Por recapitular: Tenemos un binario que lee un fichero con unos bytes que son parcheados en función de una serie de operaciones con los bytes de una clave que le pasamos por parámetros. Posteriormente esos bytes se pasan a la máquina virtual que los interpreta. Así que el objetivo es introducir la clave que haga que los bytes parcheados sean válidos para ser ejecutados en la máquina virtual y nos de la salida que esperamos.

La máquina principal consiste en un bucle que va leyendo bytes y comprueba si se corresponden con los opcodes que implementa. Tiene varios opcodes implementados, los más interesantes para lo que tenemos son:

```
ABh -> HALT.
90h -> NOP. Ocupa un byte.
13h -> MOV dst, src. Ocupa 4 bytes.
43h -> SUB a, b. Ocupa 4 bytes.
33h -> JNZ addr. Salta a addr si el flag Z no está activo. Ocupa 3 bytes
```

El contenido del fichero ctf.hex es:

```
90 90 13 00 de ad 43 00 2e 2e 90 33 25 00 13 01
de ad 90 43 01 26 31 90 13 00 2d 00 53 00 63 23
2c 00 13 00 3c 00 53 00 63 ab 57 65 6c 6c 20 64
6f 6e 65 21 21 21 0d 0a 00 4f 70 73 21 0d 0a 00
```

Con esto podemos “desensamblar” el fichero ctf.hex para ver qué opcodes deberíamos meter para que la VM haga lo que queremos (sacar el mensaje “Well done!!!”):

```
00h: 90              NOP
01h: 90              NOP
02h: 13 00 de ad     MOV registro, 0xdead
06h: 43 00 2e 2e     SUB registro, 0x2e2e
0ah: 90              NOP
0bh: 33 25 00        JNZ 0x25
0eh: 13 01 de ad     MOV registro, 0xdead
12h: 90              NOP
13h: 43 01 26 31     SUB registro, 0x2631
17h: 90              NOP
18h: 33 25 00        JNZ 0x25
1bh: 13 00 2d 00     MOV registro, 0x2d
1fh: 53 00
21h: 63 23 2c 00
25h: 13 00 3c 00
29h: 53 00
2bh: 63 ab
2dh: Well done!!!
3ch: Ops!
```

Para que no se ejecute el salto a 0x25, que lleva a sacar el mensje “Ops!”, debemos hacer que las restas “SUB registro, 0x2e2e“ y “SUB registro, 0x2631” den 0, con lo que necesitamos que los valores de las posiciones 4, 5, 10h y 11h sean 2eh, 2eh, 31h, y 26h respectivamente (está en Little Endian).

Para que esto sea así, debemos hacer que la serie de XORs entre los distintos bytes de la clave den 0x2E2E3126. Para calcular todos los posibles valores me hice un programa que sacaba todas las posibles combinaciones de 8 letras mayúsculas y minúsculas (gracias ahí a s4ur0n porque si no todavía estaba ejecutándose, al principio lo hice también con números y símbolos), las pasaba los XORs del programa, y comprobaba si el resultado era 0x2E2E3126. Dejé el programa corriendo y vi que los candidatos iban a rondar por los cientos de miles, así que descarté el enfoque y me puse a pensar en dejar fijas las 3 primeras letras a palabras que se repetían en los retos. Entre las palabras que probé estaba Xmas, lo que me dio como salida la palabra “XmasvCPU”:

```
#include <stdio.h>
#include <string.h>

int main() {
    int target = 0x2e2e3126, result = 0;
    char alfabeto[]="abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ";
    char ahora='X';
    char ahora2='m';
    char ahora3='a';
    char ahora4='s';
    int longitud=strlen(alfabeto);
    unsigned char a, b, c, d, e, f, g, h;

    //Fijamos las 3 primeras letras a “Xmas”
    //for (b = 0; b < longitud; b++) {
        //for (c = 0; c < longitud; c++) {
            //for (d = 0; d < longitud; d++) {
                for (e = 0; e < longitud; e++) {
                    for (f = 0; f < longitud; f++) {
                        for (g = 0; g < longitud; g++) {
                            for (h = 0; h < longitud; h++) {
                                    result = (char)(ahora4 ^ alfabeto[h]) +
                                            ((char)(ahora3 ^ alfabeto[g]) << 8) +
                                            ((char)(ahora2 ^ alfabeto[f]) << 16) +
                                            ((char)(ahora ^ alfabeto[e]) << 24);

                                    if (result == target) {
                                        printf("%c%c%c%c%c%c%c%c\n",
                                                    ahora, ahora2, ahora3,
ahora4, alfabeto[e], alfabeto[f], alfabeto[g], alfabeto[h]);
                                    }
                            }
                        }
                    }
                }
            //}
        //}
    //}

    return 0;
}


fuska@s0rr0w:~/Wargames/s4ur0n2024$ ./solve
XmasvCPU
```

```
fuska@s0rr0w:~/Wargames/s4ur0n2024$ ./vm.original ctf.hex XmasvCPU
         |
        -+-
         A
        /=\              /\  /\    ___  _ __  _ __ __    __
      i/ O \i           /  \/  \  / _ \| '__|| '__|\ \  / /
      /=====\          / /\  /\ \|  __/| |   | |    \ \/ /
      /  i  \          \ \ \/ / / \___/|_|   |_|     \  /
    i/ O * O \i                                      / /
    /=========\       __  __                        /_/    _
    /  *   *  \       \ \/ /        /\  /\    __ _  ____  | |
  i/ O   i   O \i      \  /   __   /  \/  \  / _` |/ ___\ |_|
  /=============\      /  \  |__| / /\  /\ \| (_| |\___ \  _
  /  O   i   O  \     /_/\_\      \ \ \/ / / \__,_|\____/ |_|
i/ *   O   O   * \i
/=================\
       |___|
```

```
XMAS CTF by @NN2ed_s4ur0n (https://cs3group.com)
Trying password... XmasvCPU.
If it is ok, then try https://xmas2024.s4ur0n.com/level5/verify-
xmas.php?pwd=sha256('XmasvCPU')

2024 XMAS CTF VM result... Well done!!!

fuska@s0rr0w:~/Wargames/s4ur0n2024$
```

Para mi tenía el suficiente sentido, no eran letras al azar, así que probé y… agua. Después de darle muchas vueltas se me encendió la bombilla y modifiqué las letras fijas en el programa por “vCPU”, teniendo en cuenta cómo se hacía el XOR con la clave, y di con la palabra correcta: “vCPUXmas”:

Muchas gracias por los retos s4ur0n, ha sido muy entretenido.