



HackerClass

CTF COMPFEST 14

Wave 1

Challenges

Title	Tags
<u>Binary Exploitation Is Ez</u>	Buffer Overflow
<u>Yours Truly</u>	RSA
<u>Naik Pangkat</u>	Steganography
<u>Binary Pin</u>	Java, JAR
<u>Hospital Donation</u>	JavaScript



Binary Exploitation

**Binary
Exploitation is Ez**

Diberikan zip file yang berisi 64 bit elf binary dan source code dari binary tersebut.

```
[*] '/home/reeyadono/CTF/compfest/binex_is_ez/ez'  
Arch:      amd64-64-little  
RELRO:     Full RELRO  
Stack:     Canary found  
NX:        NX enabled  
PIE:       No PIE (0x400000)  
FORTIFY:   Enabled
```

Terdapat fungsi "win" yang jika terpanggil akan memberikan shell. Fungsi tersebut merupakan target utama kita untuk menyelesaikan challenge ini

```
void EZ_WIN() {  
    puts("EAAAAAAAAAAAAASYYYYYYYYYYYY");  
    system("/bin/sh");  
    exit(0);  
}
```

Vulnerability

- Fokus utama selanjutnya adalah pada struct meme yang ada di source code.
- Terdapat menu untuk create, edit, dan print meme
- Menu create akan mengalokasikan memory (malloc) untuk struct meme dan content. Bisa kita lihat juga pada struct meme ada pointer menuju fungsi yang akan diisi oleh address dari meme printer.
- Menu edit bisa membuat user mengedit isi dari string content namun fungsi ini menggunakan `gets()` untuk mengambil input user (Vulnerability buffer overflow). Kita akan memanfaatkan fungsi ini untuk melakukan overwrite pada variabel func menjadi address fungsi win.
- Menu print akan mengeksekusi fungsi yang ada pada variabel func pada struct meme. Bisa kita gunakan untuk eksekusi fungsi win yang sudah kita tulis addressnya.

```
struct meme
{
    void (*func)(char*);
    char* content;
};
```

```
void edit_meme() {
    unsigned int idx;
    printf("Index: ");
    idx = read_int();
    if(memes[idx] == NULL) {
        puts("There's no meme there!");
        return;
    }
    printf("Enter meme content: ");
    gets(memes[idx]->content);
    puts("Done!");
}
```

```
void print_meme() {
    unsigned int idx;
    printf("Index: ");
    idx = read_int();
    if(memes[idx] == NULL) {
        puts("There's no meme there!");
        return;
    }
    (*(memes[idx]->func))(memes[idx]->content);
}
```

Solusi

Basic ret2win buffer overflow with some extra step.

- Jalankan new_meme 2 kali.
- Edit meme index 0, lakukan buffer overflow, overwrite func pada meme index 1 dengan address dari EZ_WIN(). (PIE mati sehingga bisa langsung saja kita cari addressnya menggunakan aplikasi yang kalian biasa gunakan).
- Jalankan print_meme untuk meme index 1.
- Dapat Shell :D.

```
reeyadono@DESKTOP-71GBV73:~/CTF/compfest/binex_is_ez$ python3 payload.py
[*] '/home/reeyadono/CTF/compfest/binex_is_ez/ez'
Arch:      amd64-64-little
RELRO:     Full RELRO
Stack:     Canary found
NX:        NX enabled
PIE:       No PIE (0x400000)
FORTIFY:   Enabled
[+] Opening connection to 103.185.38.238 on port 15733: Done
[*] Switching to interactive mode
$ cat flag.txt
COMPFEST1{[REDACTED]5}$
```

Tambahan

Kebingungan dengan solusinya? Pastikan lagi anda sudah paham beberapa basic problem binex (Untuk soal ini: buffer overflow, basic ret2win, basic heap). Berikut ini resource yang bisa menambah pengetahuan kalian:

- [LiveOverflow Binary Exploitation playlist](#)
- <https://ir0nstone.gitbook.io/notes/types/stack/introduction>
- <https://guyinatuxedo.github.io/index.html>

Bonus ← ini link btw (bukan rickroll)





Cryptography **Yours Truly**

Yours Truly

- Terdapat suatu attachment dengan dua file, yaitu **pubkey.pem** dan **message.enc**. Dari namanya, kita mengetahui bahwa file pubkey merupakan kunci publik RSA seseorang.
- Untuk melihat isi pesan yang terenkripsi, kita perlu untuk mendapatkan kunci privat yang berpasangan dengan kunci publik tersebut.
- Pertama, dapat kita ekstrak modulus dan eksponen dari kunci publik tersebut untuk mempelajarinya.

```
>>> from Crypto.PublicKey import RSA
>>> pubkey = RSA.importKey(open("pubkey.pem", "r").read())
>>> print(pubkey.n, pubkey.e)
580642391898843192929563856870897799650883152718761762932292482252152591279871421569162037190419036435041797739880389529
593674485555792234900969402019055601781662044515999210032698275981631376651117318677368742867687180140048715627160641771
118040372573575479330830092989800730105573700557717146251860588802509310534792310748898504394966263819959963273509119791
037525504422606634640173277598774814099540555569257179715908642917355365791447508751401889724095964924513196281345665480
688029639999472649549163147599540142367575413885729653166517595719991872223011969856259344396899748662101941230745601719
730556631637 65537
>>> _
```

Yours Truly

- Ternyata modulus tersebut sudah pernah ditemukan sebelumnya dan terdapat faktor-faktornya pada [FactorDB](#). Lalu, kita menemukan bahwa modulus tersebut terdiri atas lebih dari dua faktor prima.
- Kita dapat mencari $\phi(N)$ dari modulus tersebut sebagaimana pada RSA dengan dua faktor, tetapi kali ini lebih banyak faktornya.

```
>>> factors = [9282105380008121879, 9303850685953812323, 9389357739583927789, 10336650220878499841, 10638241655447339831
, 11282698189561966721, 11328768673634243077, 11403460639036243901, 11473665579512371723, 11492065299277279799, 11530534
813954192171, 11665347949879312361, 12132158321859677597, 12834461276877415051, 12955403765595949597, 129739723367779797
01, 13099895578757581201, 13572286589428162097, 14100640260554622013, 14178869592193599187, 14278240802299816541, 145230
70016044624039, 14963354250199553339, 15364597561881860737, 15669758663523555763, 15824122791679574573, 1599836546307426
8941, 16656402470578844539, 16898740504023346457, 17138336856793050757, 17174065872156629921, 17281246625998849649]
>>> phi = math.prod([i - 1 for i in factors])
>>> phi
580642391898843191487404652150193463439642600155214610402815446275117822457602964108279991178253399797937961990567828910
3189443760209418749959129424571837785408787232677949141800666918857974957805860863128934894322453334083108951829885566055
541321469492863749601696156719452204625091396670183612468234453545730714411260422415053794985900973204357184470104831581
957188055965458235216412636167147716884241110058234315146752181551500634472836779298794303330378218517375396697137335380
548442818167481491481087606998467945980408909917714107491743183639877866494931448463312524563384587536906823474872320000
000000000000
```

Yours Truly

- Lalu, kita mendapatkan private key dan bisa melakukan dekripsi.

```
>>> c = bytes_to_long(open("message.enc", "rb").read())
>>> d = pow(pubkey.e, -1, phi)
>>> flag = pow(c, d, pubkey.n)
```



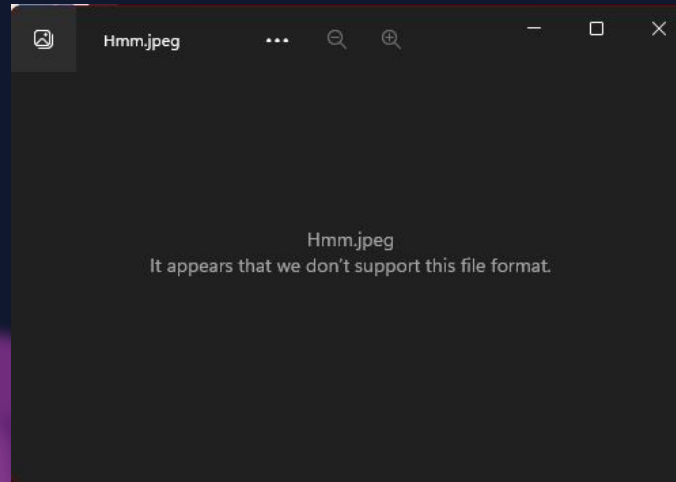
Forensic **Naik Pangkat**

tl;dr

1. Peserta diberi file Hmm.jpeg yang ketika dibuka menjadi gambar error.
2. Dengan pesan implisit dari deskripsi dan hint, file tersebut harus diubah menjadi sebuah audio file. Peserta bisa menebak, atau agar jelas, dapat memeriksa binary header dari file tersebut, untuk mengetahui bahwa file tersebut harus direname menjadi .wav.
3. Audio file tersebut berisi noise. File Hmm.wav bukan didengar, tetapi dilihat berdasarkan spectogramnya untuk mendapatkan flag.

Attachment

Pak Josefumi memberikan kamu sebuah file bernama **Hmm.jpeg**. Akan tetapi, ketika kamu mencoba membukanya, gambar tersebut tidak dapat dibuka dan terlihat rusak.



Approach Pertama

Karena pengalaman kamu dalam bidang CTF, kamu paham bahwa yang mendefinisikan file bukanlah filename extension seperti **.jpeg**, **.pdf**, atau **.txt**, melainkan struktur binary file di dalamnya lah yang mendefinisikan jenis file tersebut.

Dengan informasi tersebut, kamu mencoba analisis jenis dan isi binary file tersebut dengan bantuan command **file** dan **xxd** beberapa karakter pertama untuk melihat **file header** nya.

```
houseoforion@houseoforion:/mnt/c/Temporaries/compfest-14/Hackerclass/naik-pangkat/public$ file Hmm.jpeg
Hmm.jpeg: RIFF (little-endian) data, WAVE audio, Microsoft PCM, 16 bit, mono 44100 Hz
houseoforion@houseoforion:/mnt/c/Temporaries/compfest-14/Hackerclass/naik-pangkat/public$ xxd -l 40 Hmm.jpeg
00000000: 5249 4646 d4a8 0a00 5741 5645 666d 7420  RIFF....WAVEfmt
00000010: 1000 0000 0100 0100 44ac 0000 8858 0100  ....D....X..
00000020: 0200 1000 6461 7461                ....data
houseoforion@houseoforion:/mnt/c/Temporaries/compfest-14/Hackerclass/naik-pangkat/public$ _
```


lanjutan...

```
houseoforion@houseoforion:/mnt/c/Temporaries/compfest-14/Hackerclass/naik-pangkat/public$ file Hmm.jpeg
Hmm.jpeg: RIFF (little-endian) data, WAVE audio, Microsoft PCM, 16 bit, mono 44100 Hz
houseoforion@houseoforion:/mnt/c/Temporaries/compfest-14/Hackerclass/naik-pangkat/public$ xxd -l 40 Hmm.jpeg
00000000: 5249 4646 d4a8 0a00 5741 5645 666d 7420  RIFF....WAVEfmt
00000010: 1000 0000 0100 0100 44ac 0000 8858 0100  ....D....X..
00000020: 0200 1000 6461 7461                      ....data
houseoforion@houseoforion:/mnt/c/Temporaries/compfest-14/Hackerclass/naik-pangkat/public$ _
```

Hex signature ▲	ISO 8859-1 ◆	Offset ◆	Extension ◆	Description
52 49 46 46 ?? ?? ?? ?? 57 41 56 45	RIFF????WAVE	0	wav	Waveform Audio File Format ^[31]

[List of file signature - Wikipedia](#)

Dapat dilihat bahwa walaupun Pak Josefumi menamai file tersebut dengan .jpeg, struktur bagian dalam file tersebut ternyata merupakan file audio .wav

.jpeg, tapi kok wav?

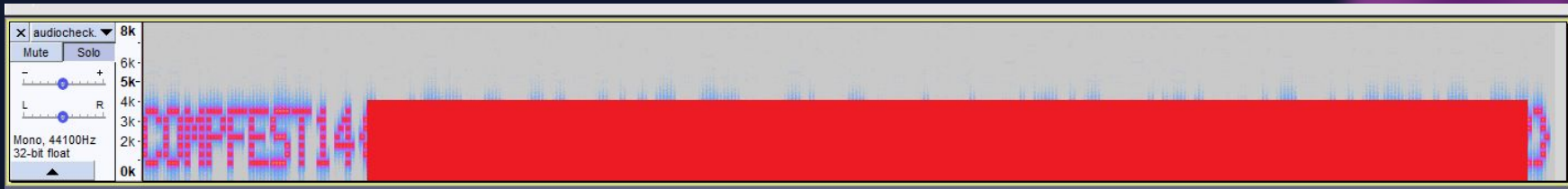
Seperti yang sudah diberitahukan sebelumnya, filename extension belum tentu mendefinisikan tipe file itu sendiri.

Struktur file memiliki “aturan” tertentu. Pada kasus di atas, aplikasi Microsoft Photo mencoba membaca file tersebut berdasarkan [aturan “.jpeg”](#). Aplikasi tersebut mencoba mencari file header .jpeg, markernya, dan data-data lainnya dari binary file tersebut berdasarkan struktur .jpeg. Karena data yang diberikan menyalahi aturan .jpeg, file tersebut tidak dapat dibaca oleh Microsoft Photo.

Aturan ini **berlaku bergantung aplikasi yang digunakan**. Ada aplikasi yang mem-parse (membaca) file berdasarkan filename extension nya, ada yang berdasarkan struktur di dalamnya. Sebagai contoh, [audacity](#) dapat membuka file tersebut tanpa harus me-rename file tersebut menjadi Hmm.wav terlebih dahulu.

Langkah terakhir

Setelah merename file tersebut menjadi Hmm.wav agar dapat didengar melalui Windows Media Player, kamu mendengar suara bising dari file tersebut. Dengan intuisi kamu, kamu coba beberapa hal dan berhasil menemukan flag dengan melihat spectrogram persebaran frekuensi suara tersebut menggunakan [audacity](#).





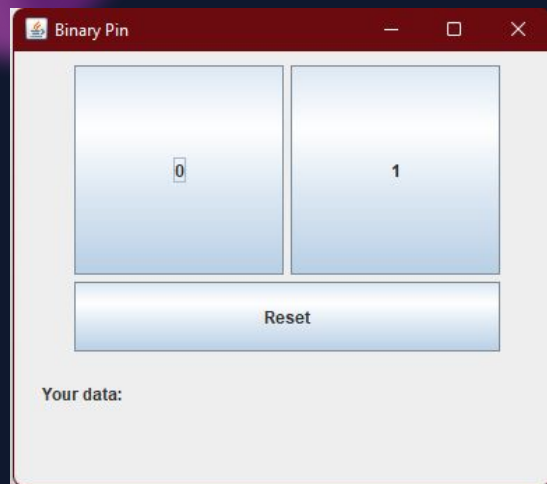
Reverse Engineering **Binary Pin**

tl;dr

1. Decompile jar menggunakan java decompiler online
2. Baca dan Implement solusi dari kode program tersebut
3. Solusi berupa kode bruteforce dengan mencoba seluruh kemungkinan input untuk mendapat flag

Attachment

Kamu diberikan file Binary-pin.jar yang ketika dibuka muncul tampilan seperti ini.







Kita dapat menekan 2 tombol dan muncul string pada Your data berdasarkan kombinasi dari hasil tekan kedua tombol tersebut.

Step pertama

Kamu dapat mencoba berulang kali hingga muncul flag, atau, sesuai dengan nama kategori soal ini, me-reverse soal ini menjadi kode pemrograman yang dapat kita baca.

Gunakanlah salah satu tool java decompiler untuk mengembalikan file jar tersebut kembali menjadi class-class java. Berikut hasil decompile file tersebut menggunakan [Decompilers Online](#).

</> Binary-Pin.jar		
 PinButton.java	.java	668 Bytes
 ResetButton.java	.java	645 Bytes
 Binarypin.java	.java	1.14 KB
 Secret.java	.java	4.05 KB

lanjutan...

Silakan dicoba decompile dan analisis fungsi dari masing-masing class tersebut. Dari keempat file tersebut, class yang paling menarik adalah secret.java. Berdasarkan hasil analisis class tersebut, setelah melalui proses enkripsi yang cukup rumit, input maksimal yang dapat kita berikan adalah 9 digit.

Oleh karena itu, kita dapat simpulkan bahwa kemungkinan input yang dapat kita berikan adalah $2^9 = 1024$ kombinasi input yang menghasilkan output berbeda-beda, yang mana satu di antaranya merupakan flag.

Kita bisa mendapatkan flag dengan membuat kode baru yang dapat mencoba masing-masing kombinasi ke dalam alur hasil dekompile file tersebut.



Web Exploitation

Hospital Donation

Hospital Donation

- Diberikan suatu **web app**. Kita perlu untuk melakukan donasi item “transport ventilator” sebanyak 10 kali, tetapi uang yang kita miliki terlalu sedikit.
- Kita dapat melakukan **inspeksi request** yang dikirim dengan DevTools pada browser atau meng-**intercept request** dengan proxy seperti yang disediakan Burpsuite.
- Kita mendapati struktur request seperti ini:

```
{"items":[{"id":x,"quantity":n},{ "id":y,"quantity":m}, ...]}
```

- Mengingat request tersebut dalam bentuk JSON, kita dapat mengasumsikan bahwa server akan mem-parse nilai-nilai yang ada pada request tersebut dengan fungsi seperti `parseInt()`.

```
> parseInt(1000000000000000000000)
1
> parseInt(1e21) // artinya 10 pangkat 21 atau 1 dengan 21 trailing zero
1
```

- Order to
ulkan pre

Hospital Donation

- Namun, kita tidak bisa langsung menggunakan integer 1e13. Kita perlu menjadikannya sebagai string, e.g. "1e13".
- Lantas, mengapa demikian?

```
> parseInt(1e13)
100000000000000
> parseInt("1e13")
1
```

- Kuantitas hanya akan bernilai 1 apabila bilangan direpresentasikan sebagai string. Kita tidak menginginkan kuantitas yang betul-betul 1e13, karena akan lebih daripada batasnya yaitu 50.
- Secara singkatnya, kita ingin precision error terjadi pada harga totalnya (kuantitas * harga) agar menjadi sangat kecil, sedangkan kita juga ingin agar kuantitasnya tetap kecil pula.



THANK YOU!

COMPFEST 14