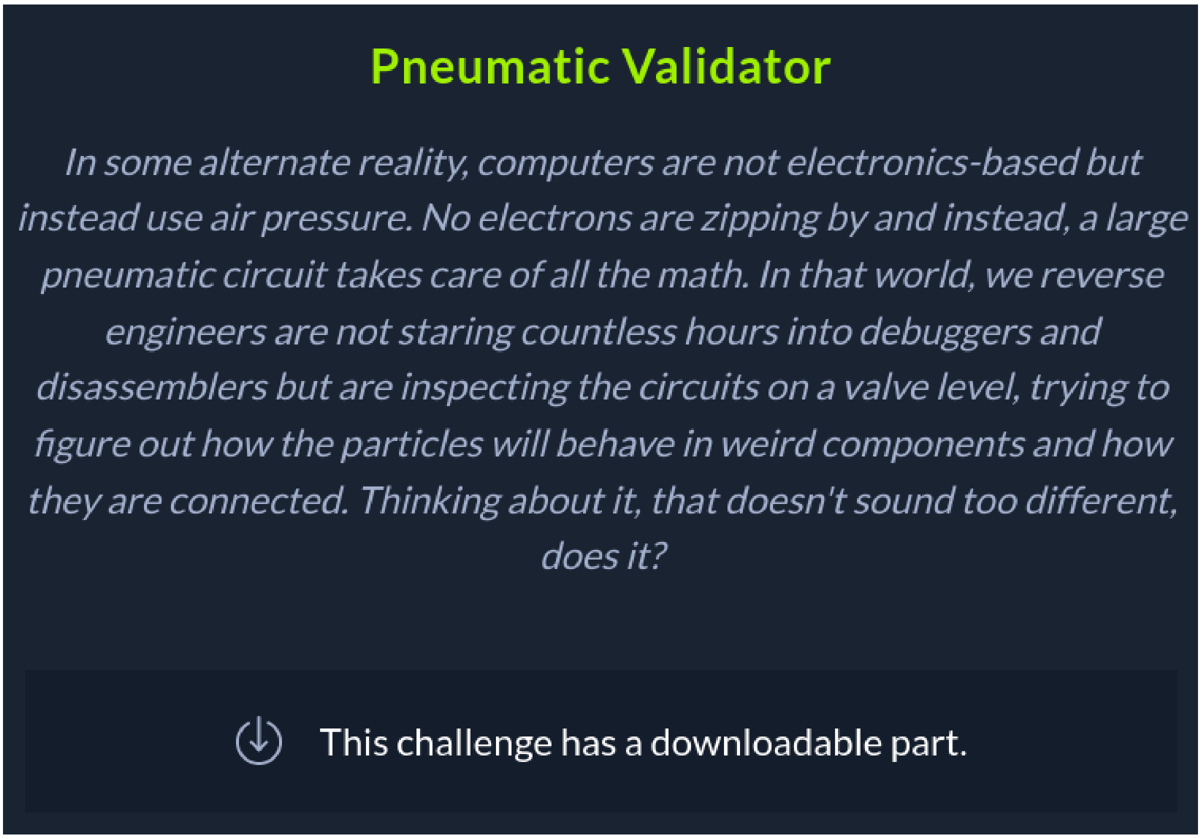
Pneumatic Validator

Uni CTF 2021 Quals - Reverse
Author: Skyf0l

The challenge

The challenge we are going to solve is a hard reversing challenge. We can see its details just below.



The downloadable part is a binary file.

Recon

As we can see below, we have here a stripped ELF 64-bit binary file, and it can be run on a linux OS.

```
$ file pneumaticvalidator
pneumaticvalidator: ELF 64-bit LSB pie executable, x86-64, version 1
(SYSV), dynamically linked, interpreter /lib64/ld-linux-x86-64.so.2,
BuildID[sha1]=b6e2c1c46822cfd6a752797e7d263dd6458cc3af, for GNU/Linux
3.2.0, stripped
```

First of all, we are going to see how it seems to work by simply executing it.

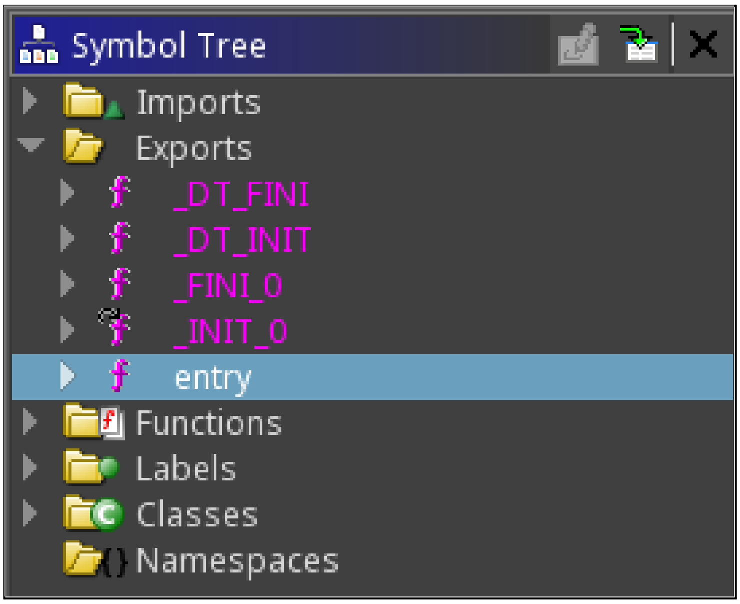
```
$ chmod +x ./pneumaticvalidator
$ ./pneumaticvalidator
Starting the Pneumatic Flag Validation Machine...
Please provide the flag to verify
$ ./pneumaticvalidator HTB{Fake_flag}
Starting the Pneumatic Flag Validation Machine...
Wrong length
```

We can deduce the binary file seems to be expecting a flag as argument with a specific size.

Reversing

Now, we are going to analyse the binary file statically. To do this, we open it in ghidra, a brilliant reversing tool produced by the NSA (https://ghidra-sre.org/).

As mentioned above, the binary file is stripped, this means we can't see any functions except the entry point.



Retrieve the main function

To begin the analysis, we need to find the main function in which the core of the program is located. To do this, we are going to start by analysing the entry function displayed below.

```
void entry(undefined8 param_1,undefined8 param_2,undefined8 param_3)

{
  undefined8 in_stack_00000000;
  undefined auStack8 [8];

  __libc_start_main(&LAB_0010554f,in_stack_00000000,&stack0x00000008,&LAB_00105680,&DAT_001056f0,
                    param_3,auStack8);
  do {
                    /* WARNING: Do nothing block with infinite loop */
  } while( true );
}
```

As we know, the entry function calls another function named **__libc_start_main** with the main function as the argument. So we can deduce the **LAB_0010554f** function is the main function.

```
0010553f 8b 00          MOV      EAX,dword ptr [RAX]
00105541 0f 28 c8       MOVAPS   XMM1,XMM0
00105544 66 0f 6e c0    MOVD     XMM0,EAX
00105548 e8 a3 bb       CALL     fmaxf                                     float fmaxf(float __x, float __y)
         ff ff
0010554d 5d             POP      RBP
0010554e c3             RET

                     LAB_0010554f                          XREF[2]:     entry:00101121(*), 0010617c
0010554f f3 0f 1e fa    ENDBR64
00105553 55             PUSH     RBP
00105554 48 89 e5       MOV      RBP,RSP
00105557 48 83 ec 20    SUB      RSP,0x20
0010555b 89 7d ec       MOV      dword ptr [RBP + -0x14],EDI
0010555e 48 89 75 e0    MOV      qword ptr [RBP + -0x20],RSI
00105562 48 8d 3d       LEA      RDI,[s_Starting_the_Pneumatic_Flag_Vali_001060... = "Starting the Pneumatic Flag V...
         a7 0a 00 00
00105569 e8 42 bb       CALL     puts                                      int puts(char * __s)
         ff ff
0010556e 83 7d ec 02    CMP      dword ptr [RBP + -0x14],0x2
00105572 74 16          JZ       LAB_0010558a
00105574 48 8d 3d       LEA      RDI,[s_Please_provide_the_flag_to_verif_001060... = "Please provide the flag to ve...
         cd 0a 00 00
0010557b e8 30 bb       CALL     puts                                      int puts(char * __s)
         ff ff
```

Ghidra couldn't create a function here, so we will force it by pressing F key, then edit the signature of this one by `int main(int argc, char **argv)`, the signature of the main function in C code.

```
                     *****************************************************************
                     *                          FUNCTION                             *
                     *****************************************************************
                     int __stdcall main(int argc, char * * argv)
      int              EAX:4          <RETURN>
      int              EDI:4          argc
      char * *         RSI:8          argv
      undefined4       Stack[-0xc]:4  local_c                  XREF[2]:     00105635(W),
                                                                            00105640(R)
      undefined4       Stack[-0x10]:4 local_10                 XREF[3]:     00105607(W),
                                                                            0010561a(RW),
                                                                            0010561e(R)
      undefined4       Stack[-0x1c]:4 local_1c                 XREF[2]:     0010555b(W),
                                                                            0010556e(R)
      undefined8       Stack[-0x28]:8 local_28                 XREF[3]:     0010555e(W),
                                                                            0010558a(R),
                                                                            001055b9(R)
                     main                                  XREF[2]:     entry:00101121(*), 0010617c
0010554f f3 0f 1e fa    ENDBR64
00105553 55             PUSH     RBP
00105554 48 89 e5       MOV      RBP,RSP
00105557 48 83 ec 20    SUB      RSP,0x20
0010555b 89 7d ec       MOV      dword ptr [RBP + local_1c],argc
0010555e 48 89 75 e0    MOV      qword ptr [RBP + local_28],argv
00105562 48 8d 3d       LEA      argc,[s_Starting_the_Pneumatic_Flag_Vali_00106... = "Starting the Pneumatic Flag V...
         a7 0a 00 00
00105569 e8 42 bb       CALL     puts                                      int puts(char * __s)
         ff ff
0010556e 83 7d ec 02    CMP      dword ptr [RBP + local_1c],0x2
00105572 74 16          JZ       LAB_0010558a
00105574 48 8d 3d       LEA      argc,[s_Please_provide_the_flag_to_verif_00106... = "Please provide the flag to ve...
         cd 0a 00 00
0010557b e8 30 bb       CALL     puts                                      int puts(char * __s)
         ff ff
```

Now that it's done, we can see the code of the main function in C code thanks to ghidra, which is easier to read and interpret than in assembler.

```
2  int main(int argc,char **argv)
3
4  {
5    int iVar1;
6    size_t sVar2;
7    float fVar3;
8    int local_10;
9
10   puts("Starting the Pneumatic Flag Validation Machine...");
11   if (argc == 2) {
12     sVar2 = strlen(argv[1]);
13     if (sVar2 == 0x14) {
14       FUN_00105498(argv[1],0x14);
15       puts("Initializing Simulation...");
16       FUN_001011e9();
17       FUN_001012bf();
18       FUN_0010149a();
19       puts("Simulating...");
20       local_10 = 0;
21       while (local_10 < 0x400) {
22         FUN_00101d67();
23         local_10 = local_10 + 1;
24       }
25       fVar3 = (float)FUN_001054e3();
26       if (15.00000000 <= fVar3) {
27         puts("Wrong \\o\\");
28       }
29       else {
30         puts("Correct /o/");
31       }
32       FUN_0010125a();
33       iVar1 = 0;
34     }
35     else {
36       puts("Wrong length");
37       iVar1 = 1;
38     }
39   }
40   else {
41     puts("Please provide the flag to verify");
42     iVar1 = 1;
43   }
44   return iVar1;
45 }
```

Understanding the code

Now that we can see the code, let's try to understand how it works.

1. Error handling

As we see in the code above, the program works as predicted.
The program starts by printing "Starting the Pneumatic Flag Validation Machine…" at line 10.
At line 11, it checks the count of arguments (stored in **argc**), if it is 2, it continues, otherwise it prints "Please provide the flag to verify" and exits with 1.

2. Check length

Then, at line 12, it calls the function **strlen** with the first argument of the program (stored in **argv[1]**). And at line 13, it compares this value with **0x14** which is 20 in decimal notation. If the values are equal, it continues, otherwise it prints "Wrong length" and exits with 1. We can deduce that the first argument is the expected flag.

3. Initialisation

Afterwards, at line 14, the program calls the function **FUN_00105498** with the flag and its length. This one calls another function **FUN_0010543c** in a loop, which seems to do some bytewise on the flag and store it in the global variable **DAT_0010a040**.

```c
void FUN_00105498(long param_1,int param_2)

{
  uint local_c;

  local_c = 0;
  while ((int)local_c < param_2) {
    FUN_0010543c((ulong)*(byte *)(param_1 + (int)local_c),(ulong)local_c,(ulong)local_c);
    local_c = local_c + 1;
  }
  return;
}
```

```c
void FUN_0010543c(byte param_1,int param_2)

{
  byte local_1c;
  int local_c;

  local_c = 0;
  local_1c = param_1;
  while (local_c < 7) {
    local_1c = local_1c << 1;
    (&DAT_0010a040)[param_2 * 7 + local_c] = (uint)(local_1c >> 7);
    local_c = local_c + 1;
  }
  return;
}
```

Then, it prints "Initializing Simulation…", and, from line 16 to 18, calls three other functions without arguments.

The first one is the function **FUN_001011e9** which seems to allocate memory, maybe for processing things.

```c
void FUN_001011e9(void)

{
  DAT_0010a2a0 = malloc(0x400);
  DAT_0010a290 = malloc(0x1000);
  DAT_0010a280 = malloc(0x1000);
  DAT_0010a278 = malloc(0x8000);
  DAT_0010a288 = malloc(0x2000);
  DAT_0010a298 = malloc(0x8000);
  return;
}
```

The following two make dark things with bitwises on these new variables, no matter.
This part seems to be the initialisation one.

4. Simulation

Following that, the program prints "Simulating…", and calls the function **FUN_00101d67** without arguments 0x400 times (1024 in decimal notation). This one also does dark things on variables. This part seems to be the simulation one.

5. Result display

After simulation, at line 25, the program calls one last function, **FUN_001054e3**, which calculates the maximum value between 3 variables, and stores the return value in var.

```c
void FUN_001054e3(void)

{
  float __y;

  __y = fmaxf(*(float *)(DAT_0010a278 + 0x6fe8),*(float *)(DAT_0010a278 + 0x6be8));
  __y = fmaxf(*(float *)(DAT_0010a278 + 0x6de4),__y);
  fmaxf(*(float *)(DAT_0010a278 + 0x6dec),__y);
  return;
}
```

Then, at line 26, it compares this var with **15.0** as float. If it is lower than or equal to **15.0**, it prints "Wrong \o\", otherwise it prints "Correct /o/".

6. Termination

Finally, at line 32, it calls the function **FUN_0010125** which seems to free all used variables and exists with 0.

```
void FUN_0010125a(void)

{
  free(DAT_0010a2a0);
  free(DAT_0010a290);
  free(DAT_0010a280);
  free(DAT_0010a278);
  free(DAT_0010a288);
  free(DAT_0010a298);
  return;
}
```

Interpretation of the programme's operation

What we can remember is that the program expects a flag in the first argument. This flag must be 20 characters long. And after some computation, if the result of the function **FUN_001054e3** is lower than or equal to **15.0**, the flag is valid.
We can interpret that the function **FUN_001054e3** seems to do fitness calculations, and the better the flag, the lower this fitness is. And when the fitness is lower than or equal to **15.0**, this is the minimum fitness and it only works for the good flag.
From here on, we will call this function the **fitness** function.

Exploitation

Now, we will see if the interpretation we have made is correct. To do this we will test several flags and compare the return value of the **fitness** function.
Then, with gdb (https://www.gnu.org/software/gdb/) and peda extension (https://github.com/longld/peda), we will find the offset of the **fitness** function in order to analyse its return value.

Gdb environment

To initialize all symbol offsets, we need to run at least one time the binary in gdb:

```
[skyf0l@skyf0l PneumaticValidator]$ gdb ./pneumaticvalidator 2> /dev/null
GNU gdb (GDB) Fedora 11.1-2.fc34
Copyright (C) 2021 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-redhat-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<https://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"...
Reading symbols from ./pneumaticvalidator...
(No debugging symbols found in ./pneumaticvalidator)
gdb-peda$ run
Starting program: /home/skyf0l/work/HTB/UniCTF/Reversing/PneumaticValidator/pneumaticvalidator
ERROR: Could not find ELF base!
Starting the Pneumatic Flag Validation Machine...
Please provide the flag to verify
[Inferior 1 (process 575797) exited with code 01]
Warning: not running
Missing separate debuginfos, use: dnf debuginfo-install glibc-2.33-20.fc34.x86_64
```

Enter in the main function

As the binary is stripped, we can’t directly set a breakpoint in the main, so we will put breakpoint on the **puts** function which is called several times in the main, then run the program with “aaaaaaaaaaaaaaaaaaaa” as argument to valid the flag length (20 chars).

```
gdb-peda$ b *puts
Breakpoint 1 at 0x7ffff7d087b0
gdb-peda$ r aaaaaaaaaaaaaaaaaaaa
Starting program: /home/skyf0l/work/HTB/UniCTF/Reversing/PneumaticValidator/pneumaticvalidator aaaaaaaaaaaaaaaaaaaa

[----------------------------------registers-----------------------------------]
RAX: 0x55555555954f (endbr64)
RBX: 0x555555559680 (endbr64)
RCX: 0x7ffff7e54598 --> 0x7ffff7e56960 --> 0x0
RDX: 0x7fffffffdb60 --> 0x7fffffffdf60 ("SHELL=/bin/bash")
RSI: 0x7fffffffdb48 --> 0x7fffffffdefe ("/home/skyf0l/work/HTB/UniCTF/Reversing/PneumaticValidator/pneumaticvalidator")
RDI: 0x55555555a010 ("Starting the Pneumatic Flag Validation Machine...")
RBP: 0x7fffffffda50 --> 0x0
RSP: 0x7fffffffda28 --> 0x55555555956e (cmp    DWORD PTR [rbp-0x14],0x2)
RIP: 0x7ffff7d087b0 (<puts>:    endbr64)
R8 : 0x0
R9 : 0x7ffff7fdb7b0 (<_dl_fini>:        endbr64)
R10: 0x7ffff7ca0790 --> 0xf001200002ab7
R11: 0x206
R12: 0x555555555100 (endbr64)
R13: 0x0
R14: 0x0
R15: 0x0
EFLAGS: 0x206 (carry PARITY adjust zero sign trap INTERRUPT direction overflow)
[-------------------------------------code-------------------------------------]
   0x7ffff7d0879f <popen@@GLIBC_2.2.5+143>:     jmp    0x7ffff7d08774 <popen@@GLIBC_2.2.5+100>
   0x7ffff7d087a1:      cs nop WORD PTR [rax+rax*1+0x0]
   0x7ffff7d087ab:      nop    DWORD PTR [rax+rax*1+0x0]
=> 0x7ffff7d087b0 <puts>:       endbr64
   0x7ffff7d087b4 <puts+4>:     push   r14
   0x7ffff7d087b6 <puts+6>:     push   r13
   0x7ffff7d087b8 <puts+8>:     push   r12
   0x7ffff7d087ba <puts+10>:    mov    r12,rdi
[------------------------------------stack-------------------------------------]
0000| 0x7fffffffda28 --> 0x55555555956e (cmp    DWORD PTR [rbp-0x14],0x2)
0008| 0x7fffffffda30 --> 0x7fffffffdb48 --> 0x7fffffffdefe ("/home/skyf0l/work/HTB/UniCTF/Reversing/PneumaticValidator/pneumaticvalidator")
0016| 0x7fffffffda38 --> 0x255555100
0024| 0x7fffffffda40 --> 0x7fffffffdb40 --> 0x2
0032| 0x7fffffffda48 --> 0x0
0040| 0x7fffffffda50 --> 0x0
0048| 0x7fffffffda58 --> 0x7ffff7cb8b75 (<__libc_start_main+213>:       mov    edi,eax)
0056| 0x7fffffffda60 --> 0x7fffffffdb48 --> 0x7fffffffdefe ("/home/skyf0l/work/HTB/UniCTF/Reversing/PneumaticValidator/pneumaticvalidator")
[------------------------------------------------------------------------------]
Legend: code, data, rodata, value

Breakpoint 1, 0x00007ffff7d087b0 in puts () from /lib64/libc.so.6
```

Here, the program has stopped on the puts function, we must then return to the main function. To avoid doing a hundred times the next command in gdb, we can put a breakpoint on the first variable of the stack which is the offset of the line just after the call of the first puts function in main. And then, continue.

```
gdb-peda$ b *0x55555555956e
Breakpoint 2 at 0x55555555956e
gdb-peda$ c
Continuing.
Starting the Pneumatic Flag Validation Machine...
[----------------------------------registers-----------------------------------]
RAX: 0x32 ('2')
RBX: 0x555555559680 (endbr64)
RCX: 0x7ffff7d82387 (<write+23>:        cmp    rax,0xfffffffffffff000)
RDX: 0x0
RSI: 0x55555555f2a0 ("Starting the Pneumatic Flag Validation Machine...\n")
RDI: 0x7ffff7e574d0 --> 0x0
RBP: 0x7fffffffda50 --> 0x0
RSP: 0x7fffffffda30 --> 0x7fffffffdb48 --> 0x7fffffffdefe ("/home/skyf0l/work/HTB/UniCTF/Reversing/PneumaticValidator/pneumaticvalidator")
RIP: 0x55555555956e (cmp    DWORD PTR [rbp-0x14],0x2)
R8 : 0x32 ('2')
R9 : 0x7ffff7e54a60 --> 0x55555555f6a0 --> 0x0
R10: 0x77 ('w')
R11: 0x246
R12: 0x555555555100 (endbr64)
R13: 0x0
R14: 0x0
R15: 0x0
EFLAGS: 0x246 (carry PARITY adjust ZERO sign trap INTERRUPT direction overflow)
[-------------------------------------code-------------------------------------]
   0x55555555955e:      mov    QWORD PTR [rbp-0x20],rsi
   0x555555559562:      lea    rdi,[rip+0xaa7]        # 0x55555555a010
   0x555555559569:      call   0x5555555550b0 <puts@plt>
=> 0x55555555956e:      cmp    DWORD PTR [rbp-0x14],0x2
   0x555555559572:      je     0x55555555958a
   0x555555559574:      lea    rdi,[rip+0xacd]        # 0x55555555a048
   0x55555555957b:      call   0x5555555550b0 <puts@plt>
   0x555555559580:      mov    eax,0x1
[------------------------------------stack-------------------------------------]
0000| 0x7fffffffda30 --> 0x7fffffffdb48 --> 0x7fffffffdefe ("/home/skyf0l/work/HTB/UniCTF/Reversing/PneumaticValidator/pneumaticvalidator")
0008| 0x7fffffffda38 --> 0x255555100
0016| 0x7fffffffda40 --> 0x7fffffffdb40 --> 0x2
0024| 0x7fffffffda48 --> 0x0
0032| 0x7fffffffda50 --> 0x0
0040| 0x7fffffffda58 --> 0x7ffff7cb8b75 (<__libc_start_main+213>:       mov    edi,eax)
0048| 0x7fffffffda60 --> 0x7fffffffdb48 --> 0x7fffffffdefe ("/home/skyf0l/work/HTB/UniCTF/Reversing/PneumaticValidator/pneumaticvalidator")
0056| 0x7fffffffda68 --> 0x200000064
[------------------------------------------------------------------------------]
Legend: code, data, rodata, value

Breakpoint 2, 0x000055555555956e in ?? ()
```

Find offset of the fitness function

Remove the puts breakpoint with “del 1” and move to the **fitness** function with the next command.
As in gdb the function does not have the same name, we will look for a pattern that looks like the code around the **fitness** function call.

```
00105625 7e e9          JLE        LAB_00105610
00105627 b8 00 00       MOV        EAX,0x0
         00 00
0010562c e8 b2 fe       CALL       FUN_001054e3
         ff ff
00105631 66 0f 7e c0    MOVD       EAX,XMM0
00105635 89 45 fc       MOV        dword ptr [RBP + local_c],EAX
00105638 f3 0f 10       MOVSS      XMM0,dword ptr [DAT_001060ec]
         05 ac 0a
         00 00
00105640 0f 2f 45 fc    COMISS     XMM0,dword ptr [RBP + local_c]
00105644 76 0e          JBE        LAB_00105654
```

```
[----------------------------------registers-----------------------------------]
RAX: 0x45344167 ('gA4E')
RBX: 0x555555559680 (endbr64)
RCX: 0x1f80
RDX: 0x7ffc
RSI: 0x38 ('8')
RDI: 0x79 ('y')
RBP: 0x7fffffffda50 --> 0x0
RSP: 0x7fffffffda30 --> 0x7fffffffdb48 --> 0x7fffffffdefe ("/home/skyf0l/work/HTB/UniCTF/Reversing/PneumaticValidator/pneumaticvalidator")
RIP: 0x555555559631 (movd   eax,xmm0)
R8 : 0xe
R9 : 0x7ffff7e54a60 --> 0x5555555573b00 --> 0x0
R10: 0x78 ('x')
R11: 0x246
R12: 0x555555555100 (endbr64)
R13: 0x0
R14: 0x0
R15: 0x0
EFLAGS: 0x242 (carry parity adjust ZERO sign trap INTERRUPT direction overflow)
[-------------------------------------code-------------------------------------]
   0x555555559625:      jle    0x555555559610
   0x555555559627:      mov    eax,0x0
   0x55555555962c:      call   0x5555555594e3
=> 0x555555559631:      movd   eax,xmm0
   0x555555559635:      mov    DWORD PTR [rbp-0x4],eax
   0x555555559638:      movss  xmm0,DWORD PTR [rip+0xaac]        # 0x55555555a0ec
   0x555555559640:      comiss xmm0,DWORD PTR [rbp-0x4]
   0x555555559644:      jbe    0x555555559654
[------------------------------------stack-------------------------------------]
0000| 0x7fffffffda30 --> 0x7fffffffdb48 --> 0x7fffffffdefe ("/home/skyf0l/work/HTB/UniCTF/Reversing/PneumaticValidator/pneumaticvalidator")
0008| 0x7fffffffda38 --> 0x255555100
0016| 0x7fffffffda40 --> 0x7fffffffdb40 --> 0x2
0024| 0x7fffffffda48 --> 0x400
0032| 0x7fffffffda50 --> 0x0
0040| 0x7fffffffda58 --> 0x7ffff7cb8b75 (<__libc_start_main+213>:       mov    edi,eax)
0048| 0x7fffffffda60 --> 0x7fffffffdb48 --> 0x7fffffffdefe ("/home/skyf0l/work/HTB/UniCTF/Reversing/PneumaticValidator/pneumaticvalidator")
0056| 0x7fffffffda68 --> 0x200000064
[------------------------------------------------------------------------------]
Legend: code, data, rodata, value
0x0000555555559631 in ?? ()
```

We found it, the **fitness** function is called at offset **0x55555555962c**, but we need to keep the offset one step further to get its return value stored in the RAX register: **0x555555559631**.
Here, the fitness of the flag “aaaaaaaaaaaaaaaaaaa” is **0x45344167** (**1161052519** in decimal).

Verification of interpretation

We have interpreted that the better the flag, the lower this fitness is. All we know about the flag is its format which is "HTB{...}".
We will therefore test with the chars we know.

Flag tested	RAX value
aaaaaaaaaaaaaaaaaaa	0x45344167

Haaaaaaaaaaaaaaaaaa	0x452f1d37
eaaaaaaaaaaaaaaaaaa	0x4536daa3
Laaaaaaaaaaaaaaaaaa	0x4530c1bd
laaaaaaaaaaaaaaaaaa	0x45328aec

For the first char, the H value seems to give a lower result than the other random chars.
So that it doesn't look like a fluke we will test with the 3rd char.

Flag tested	RAX value
aaaaaaaaaaaaaaaaaaa	0x45344167
aaBaaaaaaaaaaaaaaaa	0x452deb80
aaOaaaaaaaaaaaaaaaa	0x453499ce
aasaaaaaaaaaaaaaaaa	0x453499ce
aaSaaaaaaaaaaaaaaaa	0x45328aec

The same thing happens, when the char is good, the return value of the **fitness** function is much lower than the others.

Scripting time!

With what we have just seen above, to find the flag we need to test all chars for each char in the flag and select the one with the lowest score.
To get the return value of the **fitness** function, the script will have to load the program with gdb, set a breakpoint at **0x555555559631**, run the program with the flag and read the RAX value.

You will find the script in the resources linked with the WriteUp or here.

Finally, we just have to execute the script and we get the flag: **HTB{pN3Um4t1C_l0g1C}**.

```
[skyf0l@skyf0l PneumaticValidator]$ ./pneumaticvalidator HTB{pN3Um4t1C_l0g1C}
Starting the Pneumatic Flag Validation Machine...
Initializing Simulation...
Simulating...
Correct /o/
```

The flag validates the challenge, we did it!