# 0x41haz

> **Instructions**
>
> In this challenge, you are asked to solve a simple reversing solution. Download and analyze the binary to discover the password.
>
> *There may be anti-reversing measures in place!*

## View file info

```
┌──(kali㉿kali)-[~/TryHackMe/0x41haz]
└─$ strings 0x41hax.0x41haz
/lib64/ld-linux-x86-64.so.2
gets
exit
puts
strlen
__cxa_finalize
__libc_start_main
libc.so.6
GLIBC_2.2.5
_ITM_deregisterTMCloneTable
__gmon_start__
_ITM_registerTMCloneTable
u/UH
2@@25$gfH
sT&@f
[]A\A]A^A_
========================
Hey , Can You Crackme ?
========================
It's jus a simple binary
Tell Me the Password :
Is it correct , I don't think so.
Nope
Well Done !!
;*3$"
GCC: (Debian 10.3.0-9) 10.3.0
.shstrtab
.interp
.note.gnu.build-id
.note.ABI-tag
.gnu.hash
.dynsym
.dynstr
.gnu.version
.gnu.version_r
.rela.dyn
.rela.plt
.init
.plt.got
.text
.fini
.rodata
.eh_frame_hdr
.eh_frame
.init_array
.fini_array
.dynamic
.got.plt
.data
.bss
.comment
```

```
┌──(kali㉿kali)-[~/TryHackMe/0x41haz]
└─$ readelf 0x41hax.0x41haz -e
ELF Header:
  Magic:   7f 45 4c 46 02 02 01 00 00 00 00 00 00 00 00 00
  Class:                             ELF64
  Data:                              2's complement, big endian
  Version:                           1 (current)
  OS/ABI:                            UNIX - System V
  ABI Version:                       0
  Type:                              <unknown>: 300
  Machine:                           <unknown>: 0x3e00
  Version:                           0x1000000
  Entry point address:               0x8010000000000000
  Start of program headers:          4611686018427387904 (bytes into file)
  Start of section headers:          6931321301499904000 (bytes into file)
  Flags:                             0x0
  Size of this header:               16384 (bytes)
  Size of program headers:           14336 (bytes)
  Number of program headers:         2816
  Size of section headers:           16384 (bytes)
  Number of section headers:         7168
  Section header string table index: 6912
readelf: Warning: The e_shentsize field in the ELF header is larger than the size of an ELF section header
readelf: Error: Reading 117440512 bytes extends past end of file for section headers
readelf: Error: Too many program headers - 0xb00 - the file is not that big
```

```
┌──(kali㉿kali)-[~/TryHackMe/0x41haz]
└─$ file 0x41hax.0x41haz
0x41hax.0x41haz: ELF 64-bit MSB *unknown arch 0x3e00* (SYSV)
```

This source explains the display error message `*unknown arch 0x3e00*`

At the result of using `readelf` to analyze the file, focus on these points:

```
Data:                              2's complement, big endian

readelf: Warning: The e_shentsize field in the ELF header is larger than the size of an ELF section header
readelf: Error: Too many program headers - 0xb00 - the file is not that big
```

The current **data type** of the file is recognized as **MSB** (Most Significant Bit, **big-endian**) but the error is trying to say `the file is not that big`. Therefore, this file might be a **LSB** (Least Significant Bit, **little-endian**) one.

# Modify magic bytes

Read more about the **ELF Header**, **Class**, **Data** from here. Then, modify the header magic bytes:

From

```
7f 45 4c 46 02 02 01 00 00 00 00 00 00 00 00 00
```

To

```
7f 45 4c 46 02 01 01 00 00 00 00 00 00 00 00 00
```

After that, save the modified value and verify the file is correct format:

```
┌──(kali㉿kali)-[~/TryHackMe/0x41haz]
└─$ file 0x41hax.0x41haz
0x41hax.0x41haz: ELF 64-bit LSB pie executable, x86-64, version 1 (SYSV), dynamically linked, interpreter /lib64/ld-linux-x86-64.so.2, Bu
ildID[sha1]=6c9f2e85b64d4f12b91136ffb8e4c038f1dc6dcd, for GNU/Linux 3.2.0, stripped
```

# Analyze the content

## Radare2

```
┌──(kali㉿kali)-[~/TryHackMe/0x41haz]
└─$ r2 -d 0x41hax.0x41haz
[0x7fc7c03f19c0]> aaa
[x] Analyze all flags starting with sym. and entry0 (aa)
[x] Analyze function calls (aac)
[x] Analyze len bytes of instructions for references (aar)
[x] Finding and parsing C++ vtables (avrr)
[x] Skipping type matching analysis in debugger mode (aaft)
[x] Propagate noreturn information (aanr)
[x] Use -AA or aaaa to perform additional experimental analysis.
[0x7fc7c03f19c0]> afl
0x558179851080    1 43           entry0
0x558179853fe0    1 4124         reloc.__libc_start_main
0x558179851030    1 6            sym.imp.puts
0x558179851040    1 6            sym.imp.strlen
0x558179850000    2 40           loc.imp._ITM_deregisterTMCloneTable
0x558179851050    1 6            sym.imp.gets
0x558179851060    1 6            sym.imp.exit
0x558179851070    1 6            sym.imp.__cxa_finalize
0x558179851165    8 219          main
0x558179851160    5 133  -> 56   entry.init0
0x558179851120    5 57   -> 50   entry.fini0
0x5581798510b0    4 41   -> 34   fcn.5581798510b0
[0x7fc7c03f19c0]> s main
[0x558179851165]> pdf
           ; DATA XREF from entry0 @ 0x55817985109d
┌ 219: int main (int argc, char **argv, char **envp);
│           ; var int64_t var_40h @ rbp-0x40
│           ; var int64_t var_16h @ rbp-0x16
│           ; var int64_t var_eh @ rbp-0xe
│           ; var int64_t var_ah @ rbp-0xa
│           ; var int64_t var_8h @ rbp-0x8
│           ; var int64_t var_4h @ rbp-0x4
│           0x558179851165      55             push rbp
│           0x558179851166      4889e5         mov rbp, rsp
│           0x558179851169      4883ec40       sub rsp, 0x40
│           0x55817985116d      48b832404032.  movabs rax, 0x6667243532404032 ; '2@@25$gf'
│           0x558179851177      488945ea       mov qword [var_16h], rax
│           0x55817985117b      c745f2735426.  mov dword [var_eh], 0x40265473 ; 'sT&@'
│           0x558179851182      66c745f64c00   mov word [var_ah], 0x4c ; 'L' ; 76
│           0x558179851188      488d3d790e00.  lea rdi, str._nHey___Can_You_Crackme___n ; 0x558179852008 ; "======================\nHey
 , Can You Crackme ?\n======================"
│           0x55817985118f      e89cfeffff     call sym.imp.puts       ; int puts(const char *s)
│           0x558179851194      488d3db50e00.  lea rdi, str.Its_jus_a_simple_binary__n ; 0x558179852050 ; "It's jus a simple binary \n"
│           0x55817985119b      e890feffff     call sym.imp.puts       ; int puts(const char *s)
│           0x5581798511a0      488d3dc40e00.  lea rdi, str.Tell_Me_the_Password_: ; 0x55817985206b ; "Tell Me the Password :"
│           0x5581798511a7      e884feffff     call sym.imp.puts       ; int puts(const char *s)
│           0x5581798511ac      488d45c0       lea rax, [var_40h]
│           0x5581798511b0      4889c7         mov rdi, rax
│           0x5581798511b3      b800000000     mov eax, 0
│           0x5581798511b8      e893feffff     call sym.imp.gets       ; char *gets(char *s)
│           0x5581798511bd      488d45c0       lea rax, [var_40h]
│           0x5581798511c1      4889c7         mov rdi, rax
│           0x5581798511c4      e877feffff     call sym.imp.strlen     ; size_t strlen(const char *s)
│           0x5581798511c9      8945f8         mov dword [var_8h], eax
│           0x5581798511cc      837df80d       cmp dword [var_8h], 0xd
│       ┌─< 0x5581798511d0      7416           je 0x5581798511e8
│       │   0x5581798511d2      488d3daf0e00.  lea rdi, str.Is_it_correct___I_dont_think_so. ; 0x558179852088 ; "Is it correct , I don't
 think so."
│       │   0x5581798511d9      e852feffff     call sym.imp.puts       ; int puts(const char *s)
│       │   0x5581798511de      bf00000000     mov edi, 0
│       │   0x5581798511e3      e878feffff     call sym.imp.exit
│       └─> 0x5581798511e8      c745fc000000.  mov dword [var_4h], 0
│       ┌─< 0x5581798511ef      eb34           jmp 0x558179851225
│      ┌──> 0x5581798511f1      8b45fc         mov eax, dword [var_4h]
│      ¦¦   0x5581798511f4      4898           cdqe
│      ¦¦   0x5581798511f6      0fb65405ea     movzx edx, byte [rbp + rax - 0x16]
│      ¦¦   0x5581798511fb      8b45fc         mov eax, dword [var_4h]
│      ¦¦   0x5581798511fe      4898           cdqe
│      ¦¦   0x558179851200      0fb64405c0     movzx eax, byte [rbp + rax - 0x40]
│      ¦¦   0x558179851205      38c2           cmp dl, al
│     ┌───< 0x558179851207      7506           jne 0x55817985120f
│     ¦¦¦   0x558179851209      8345fc01       add dword [var_4h], 1
```

```
|    ┌─────< 0x55817985120d      eb16           jmp 0x558179851225
|    │└────> 0x55817985120f      488d3d940e00.  lea rdi, str.Nope       ; 0x5581798520aa ; "Nope"
|    │ ¦|   0x558179851216      e815feffff     call sym.imp.puts       ; int puts(const char *s)
|    │ ¦|   0x55817985121b      bf00000000     mov edi, 0
|    │ ¦|   0x558179851220      e83bfeffff     call sym.imp.exit
|    │ ¦|   ; CODE XREFS from main @ 0x5581798511ef, 0x55817985120d
|    └──└──> 0x558179851225      8b45fc         mov eax, dword [var_4h]
|      ¦     0x558179851228      3b45f8         cmp eax, dword [var_8h]
|      └────< 0x55817985122b      7cc4           jl 0x5581798511f1
|            0x55817985122d      488d3d7b0e00.  lea rdi, str.Well_Done___ ; 0x5581798520af ; "Well Done !!"
|            0x558179851234      e8f7fdffff     call sym.imp.puts       ; int puts(const char *s)
|            0x558179851239      b800000000     mov eax, 0
|            0x55817985123e      c9             leave
└            0x55817985123f      c3             ret
[0x558179851165]>
```

## Ghidra

Navigate to the function **FUN_00101165**:

```
undefined8 FUN_00101165(void)

{
  size_t sVar1;
  char local_48 [42];
  undefined8 local_1e;
  undefined4 local_16;
  undefined2 local_12;
  int local_10;
  int local_c;

  local_1e = 0x6667243532404032;
  local_16 = 0x40265473;
  local_12 = 0x4c;
  puts("======================\nHey , Can You Crackme ?\n======================");
  puts("It\'s jus a simple binary \n");
  puts("Tell Me the Password :");
  gets(local_48);
  sVar1 = strlen(local_48);
  local_10 = (int)sVar1;
  if ((int)sVar1 != 0xd) {
    puts("Is it correct , I don\'t think so.");
                    /* WARNING: Subroutine does not return */
    exit(0);
  }
  local_c = 0;
  while( true ) {
    if (0xc < local_c) {
      puts("Well Done !!");
      return 0;
    }
    if (*(char *)((long)&local_1e + (long)local_c) != local_48[local_c]) break;
    local_c = local_c + 1;
  }
  puts("Nope");
                    /* WARNING: Subroutine does not return */
  exit(0);
}
```

---

Notice on these hex value:

- 0x6667243532404032
- 0x40265473
- 0x4c

Decode these value with `xxd -r`

```
┌──(kali㉿kali)-[~/TryHackMe/0x41haz]
└─$ echo "0x6667243532404032" | xxd -r
fg$52@@2
┌──(kali㉿kali)-[~/TryHackMe/0x41haz]
└─$ echo "0x40265473" | xxd -r
```

```
@&Ts
┌──(kali㉿kali)-[~/TryHackMe/0x41haz]
└─$ echo "0x4c" | xxd -r
L
```

Because the data type of the file is **little-endian**, it stores data with the last byte at first (reverse). For example:

```
Orignal Value: 0x12345678

# LSB (Least Significant, little-endian)
Stored value: 78 56 34 12

# MSB (Most Significant, big-endian)
Stored value: 12 34 56 78
```

Read more at here.

For that, the above decoded strings must be reversed before input as the required password of the file:

- fg$52@@2 → 2@@25$gf
- @&Ts → sT&@
- L → L

Combine the above reversed strings and the final string would be: **2@@25$gfsT&@L**

```
┌──(kali㉿kali)-[~/TryHackMe/0x41haz]
└─$ ./0x41hax.0x41haz
=======================
Hey , Can You Crackme ?
=======================
It's jus a simple binary

Tell Me the Password :
2@@25$gfsT&@L
Well Done !!
```