

Francisco Melendez Laureano 801-19-3512 francisco.melendez4@upr.edu

Prof. Rafael A. Arce

CCOM4702

Introduction: Basic Exploration Tools 

### 1) even-password:

The password is a hard coded string literal, can easily be found with

- strings even-password

We'll find along other texts:

**thecakeisalie**

If we utilize **GDB**, we will find a comparison between `<rdi>` and `<rsi>`, they are arguments for `strcmp`, inspecting `<rsi>` will hint us the correct password as well.

[Output]

**Congratulations, the test is now over.**

## 2) odd-password:

To find the solution, we inspect the binary with **GDB**, we'll encounter the function **Is\_good**, in here we'll trace a series of comparisons between the register `<al>` and some hexadecimal numbers:

`< cmp $0x88, %al >`

..

..

`< cmp $0xff, %al >`

Since the comparison aren't ASCII values that we can input with a simple keyboard, we'll use a *Python* script to output the required input:

- ```
python3 -c 'import sys;
sys.stdout.buffer.write(b"\x88\x99\xaa\xbb\xcc\xdd\xee\xff") |
./odd-password
```

[Output]

Welcome to the dark side!

### 3) halting-problem:

We utilize `objdump -d halting-problem`, and find **0x186a0** (100000 decimal) as argument for **sleep** (counted in seconds). This is what the instructions

< `mov edi, 0x186a0` >

are accomplishing.

We utilize ***bless***, a hex editor to edit the value for sleep.

```
)00010ff| 00 80 3D 31 2F 00 00 00 75 2F 55 48 83 3D E6 2E 00 00 00
)0001112| 48 89 E5 74 0C 48 8B 3D 12 2F 00 00 E8 2D FF FF FF E8 68
)0001125| FF FF FF C6 05 09 2F 00 00 01 5D C3 0F 1F 80 00 00 00
)0001138| C3 0F 1F 80 00 00 00 00 E9 7B FF FF FF 55 48 89 E5 48 81
)000114b| 3D B4 0E 00 00 E8 DB FE FF FF 48 8D 3D C5 0E 00 00 E8 CF
)000115e| FE FF FF BF A0 86 01 00 E8 D5 FE FF FF 48 8D 3D CC 0E 00
)0001171| 00 E8 B9 FE FF FF B8 00 00 00 00 5D C3 66 90 41 57 49 89
)0001184| D7 41 56 49 89 F6 41 55 41 89 FD 41 54 4C 8D 25 50 2C 00
)0001197| 00 55 48 8D 2D 50 2C 00 00 53 4C 29 E5 48 83 EC 08 E8 53
)00011aa| FE FF FF 48 C1 FD 03 74 1B 31 DB 0F 1F 00 4C 89 FA 4C 89
)00011bd| F6 44 89 EF 41 FF 14 DC 48 83 C3 01 48 39 DD 75 EA 48 83
)00011d0| C4 08 5B 5D 41 5C 41 5D 41 5E 41 5F C3 0F 1F 00 C3 00 00
```

Edit the values **A0 86 01** -> **00 00 00**.

#### [Output]

Brb, I'm out to get cookies.  
Going to halt anytime now..  
Done!

#### 4) straceme:

Running **straceme** in **GDB**, we observe that in the *stack register*, `< rbp - 0x44 >` holds the value 1, the comparison needs it to be 2 for it to proceed.

This is because it requires a *command line argument*, the password.  
It is 1 because the first value is the program name.

After following the program along we'll find the comparison and extract the correct password:

- sixoclockofachristmasmorning

[Output]:

You guessed it!

## 5) guesser:

We examine the control flow with **objdump -d guesser**

and notice that the program performs the following on the `/dev/urandom` file:

**opens**  
**reads**  
**close**

We are instructed to “Inspect the memory location of the variable where the random value was placed by read” which after reading it is placed to the frame pointer register `< rbp >`.

We'll then be prompted to input:

A number between (between 0 and 4\_294\_967\_295)

However my solution was a number much bigger than that:

**14\_981\_043\_473**

```
GuesSED arguments:
arg[0]: 0x55555555604d --> 0x6720756f59007525 ('%u')
arg[1]: 0x7fffffffdf14 --> 0x7cf0951100000000
[-----stack-----]
0000| 0x7fffffffdf10 --> 0x0
0008| 0x7fffffffdf18 --> 0x37cf09511
0016| 0x7fffffffdf20 --> 0x1
0024| 0x7fffffffdf28 --> 0x7ffff7df7510 (<__libc_start_call_main+128>: mov     edi, edi)
0032| 0x7fffffffdf30 --> 0x7fffffe020 --> 0x7fffffe028 --> 0x38 ('8')
0040| 0x7fffffffdf38 --> 0x55555555185 (<main>:      push    rbp)
0048| 0x7fffffffdf40 --> 0x155554040
0056| 0x7fffffffdf48 --> 0x7fffffe038 --> 0x7fffffe32a ("/home/kryozek/Shrine")
[-----]
Legend: code, data, rodata, value
0x0000555555551f9 in main ()
gdb-peda$ p/d $rbp-0x8
$2 = 140737488346904
gdb-peda$ x/d $rbp-0x8
0x7fffffffdf18: 14981043473
gdb-peda$
```

As shown above, examining `< x >` the register `$rbp` at an offset of `-0x8` we can see the stored random number, which will be compared shortly after with our input `< rdi >` ...

```
mov     -0xc(%rbp),%edx
mov     -0x8(%rbp),%eax
cmp     %eax,%edx
```

```

0x55555555214 <main+143>: jmp     0x55555555222 <main+157>
0x55555555216 <main+145>: lea     rdi,[rip+0xe46]      # 0x555555556063
0x5555555521d <main+152>: call    0x55555555030 <puts@plt>
0x55555555222 <main+157>: mov     eax,0x0
GuesSED arguments:
arg[0]: 0x555555556050 ("You guessed right!")
[-----stack-----]
0000| 0x7fffffffdf10 --> 0x7cf0951100000000
0008| 0x7fffffffdf18 --> 0x37cf09511
0016| 0x7fffffffdf20 --> 0x1
0024| 0x7fffffffdf28 --> 0x7fff7df7510 (<__libc_start_call_main+128>: mov     edi,edi)
0032| 0x7fffffffdf30 --> 0x7fffffe020 --> 0x7fffffe028 --> 0x38 ('8')
0040| 0x7fffffffdf38 --> 0x55555555185 (<main>:      push    rbp)
0048| 0x7fffffffdf40 --> 0x155554040
0056| 0x7fffffffdf48 --> 0x7fffffe038 --> 0x7fffffe32a ("/home/kryozek/Shrine")
[-----]
Legend: code, data, rodata, value
0x00005555555520f in main ()
gdb-peda$ 

```

Here in **arg[0]** we see the program output our success 😊

"You guessed right!"

