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]

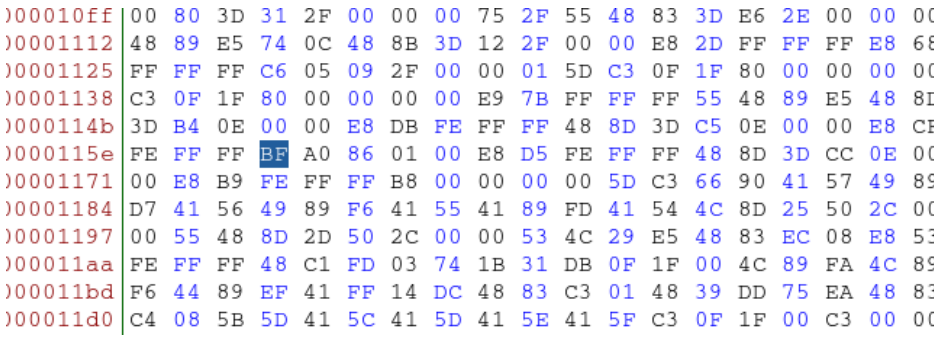
Welcome to the dark side!

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.



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:

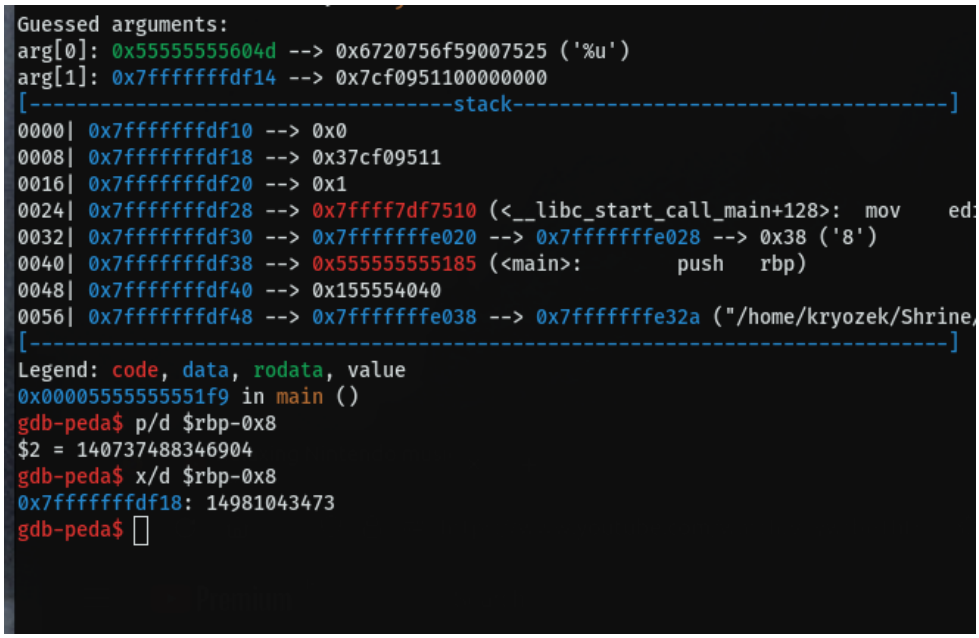
We examine the control flow with **objdump -d guesser**
and notice that the program performs the following on the */dev/urandom* file:

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:

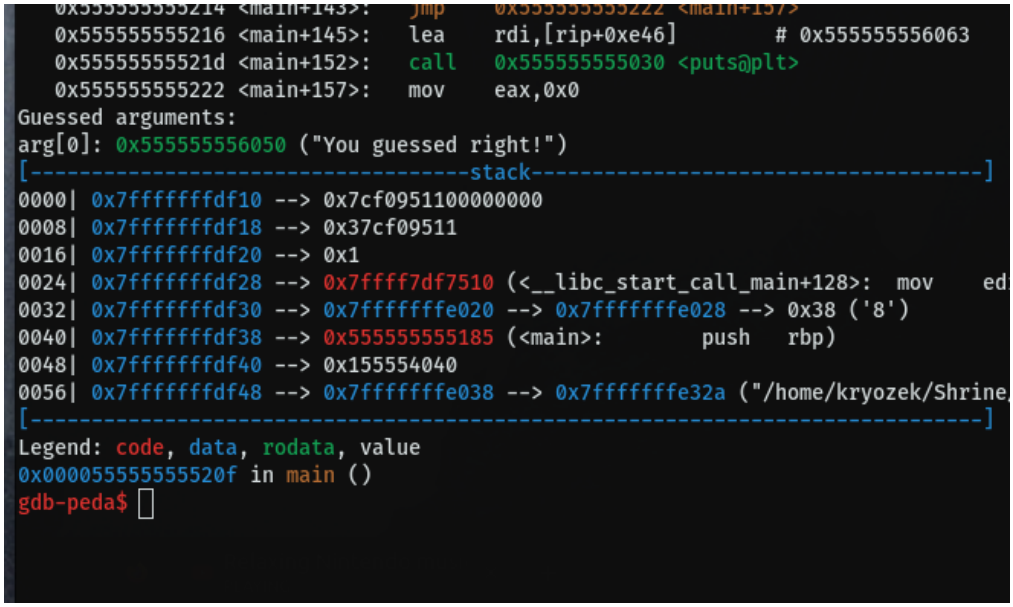
A number between (between 0 and 4_294_967_295)

However my solution was a number much bigger than that:
14_981_043_473



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
```



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

"You guessed right!"

🐧