Picker I

Medium | Reverse Engineering | picoGym Exclusive | Python

AUTHOR: LT 'SYREAL' JONES

Description

This service can provide you with a random number, but can it do anything else?

Connect to the program with netcat:

```
$ nc saturn.picoctf.net 56027
```

The program's source code can be downloaded here.

This challenge launches an instance on demand.

Its current status is: RUNNING

Instance Time Remaining: 14:45

Restart Instance

Hints

1

Can you point the program to a function that does something useful for you?

```
Picker1 % ls
picker-I.py
Picker1 % nc saturn.picoctf.net 63370
Try entering "getRandomNumber" without the double quotes...
==> 4
'int' object is not callable
Picker1 % nc saturn.picoctf.net 63370
Try entering "getRandomNumber" without the double quotes...
==> getRandomNumber
4
Try entering "getRandomNumber" without the double quotes...
==> getRandomNumber
4
Try entering "getRandomNumber" without the double quotes...
==> "getRandomNumber"
'str' object is not callable
Picker1 % nc saturn.picoctf.net 63370
Try entering "getRandomNumber" without the double quotes...
==> esoteric1

  int query_apm_bios(void)
{
        struct biosregs ireg, oreg;

        /* APM BIOS installation check */
        initregs(&ireg);
        ireg.ah = 0x53;
        intcall(0x15, &ireg, &oreg);

        if (oreg.flags & X86_EFLAGS_CF)
                return -1;              /* No APM BIOS */

        if (oreg.bx != 0x504d)          /* "PM" signature */
                return -1;
```

```
Try entering "getRandomNumber" without the double quotes...
==> win
0x70 0x69 0x63 0x6f 0x43 0x54 0x46 0x7b 0x34 0x5f 0x64 0x31 0x34 0x6d 0x30 0x6e 0x64 0x5f 0x31 0
x6e 0x5f 0x37 0x68 0x33 0x5f 0x72 0x30 0x75 0x67 0x68 0x5f 0x36 0x65 0x30 0x34 0x34 0x34 0x30 0x
64 0x7d
Try entering "getRandomNumber" without the double quotes...
==> %

Picker1 %
```

Kenapa gak harus enter getRandomNumber saja? bisa kita lihat di programnya.

```
import sys
```

```
def getRandomNumber():
 print(4)  # Chosen by fair die roll.
           # Guaranteed to be random.
           # (See XKCD)
```

```
def exit():
 sys.exit(0)
 def esoteric1():
 esoteric = \
 '''
 int query_apm_bios(void)
{
   struct biosregs ireg, oreg;
```

```
   /* APM BIOS installation check */
   initregs(&ireg);
   ireg.ah = 0x53;
   intcall(0x15, &ireg, &oreg);
```

```
   if (oreg.flags & X86_EFLAGS_CF)
       return -1;      /* No APM BIOS */
```

```
   if (oreg.bx != 0x504d)      /* "PM" signature */
       return -1;
```

```
   if (!(oreg.cx & 0x02))      /* 32 bits supported? */
       return -1;
```

```
   /* Disconnect first, just in case */
   ireg.al = 0x04;
   intcall(0x15, &ireg, NULL);
```

```
   /* 32-bit connect */
   ireg.al = 0x03;
   intcall(0x15, &ireg, &oreg);
```

```
   boot_params.apm_bios_info.cseg        = oreg.ax;
   boot_params.apm_bios_info.offset      = oreg.ebx;
   boot_params.apm_bios_info.cseg_16     = oreg.cx;
   boot_params.apm_bios_info.dseg        = oreg.dx;
   boot_params.apm_bios_info.cseg_len    = oreg.si;
   boot_params.apm_bios_info.cseg_16_len = oreg.hsi;
   boot_params.apm_bios_info.dseg_len    = oreg.di;
```

```
   if (oreg.flags & X86_EFLAGS_CF)
       return -1;
```

```
   /* Redo the installation check as the 32-bit connect;
      some BIOSes return different flags this way... */
```

```
   ireg.al = 0x00;
   intcall(0x15, &ireg, &oreg);
```

```
   if ((oreg.eflags & X86_EFLAGS_CF) || oreg.bx != 0x504d) {
       /* Failure with 32-bit connect, try to disconnect and ignore */
       ireg.al = 0x04;
       intcall(0x15, &ireg, NULL);
       return -1;
   }
```

```
   boot_params.apm_bios_info.version = oreg.ax;
   boot_params.apm_bios_info.flags   = oreg.cx;
   return 0;
}
 '''
 print(esoteric)
```

```
def win():
 # This line will not work locally unless you create your own
'flag.txt' in
 #   the same directory as this script
 flag = open('flag.txt', 'r').read()
 #flag = flag[:-1]
 flag = flag.strip()
 str_flag = ''
 for c in flag:
   str_flag += str(hex(ord(c))) + ' '
 print(str_flag)

def esoteric2():
 esoteric = \
 '''
#include "boot.h"
```

```
#define MAX_8042_LOOPS  100000
#define MAX_8042_FF 32
```

```
static int empty_8042(void)
{
   u8 status;
   int loops = MAX_8042_LOOPS;
   int ffs   = MAX_8042_FF;
```

```
   while (loops--) {
       io_delay();
```

```
       status = inb(0x64);
       if (status == 0xff) {
           /* FF is a plausible, but very unlikely status */
           if (!--ffs)
               return -1; /* Assume no KBC present */
       }
       if (status & 1) {
           /* Read and discard input data */
           io_delay();
           (void)inb(0x60);
       } else if (!(status & 2)) {
           /* Buffers empty, finished! */
           return 0;
       }
   }
```

```
   return -1;
}
```

```
/* Returns nonzero if the A20 line is enabled.  The memory address
  used as a test is the int $0x80 vector, which should be safe. */
```

```
#define A20_TEST_ADDR   (4*0x80)
#define A20_TEST_SHORT  32
#define A20_TEST_LONG   2097152 /* 2^21 */
```

```
static int a20_test(int loops)
{
   int ok = 0;
   int saved, ctr;
```

```
   set_fs(0x0000);
   set_gs(0xffff);
```

```
   saved = ctr = rdfs32(A20_TEST_ADDR);
```

```
   while (loops--) {
       wrfs32(++ctr, A20_TEST_ADDR);
       io_delay(); /* Serialize and make delay constant */
       ok = rdgs32(A20_TEST_ADDR+0x10) ^ ctr;
       if (ok)
           break;
   }
```

```
   wrfs32(saved, A20_TEST_ADDR);
   return ok;
```

```
}
```

```
/* Quick test to see if A20 is already enabled */
static int a20_test_short(void)
{
   return a20_test(A20_TEST_SHORT);
}
 '''
 print(esoteric)
```

```
while(True):
 try:
   print('Try entering "getRandomNumber" without the double quotes...')
   user_input = input('==> ')
   eval(user_input + '()')
 except Exception as e:
   print(e)
   break
```

ada fungsi win jadi kita bisa tulis win.