CS519/419
Cyber Attacks & Defense

Pwntools and other tips

10/09/18

Oregon State University

In Week1 Level5

- XOR-ing each character of "5385876381" with 0x1

In Week1 Level5

- XOR-ing each character of "5385876381" with 0x1
  - `''.join([chr(ord(c)^0x1) for c in "5385876381"])`

```
>>> ''.join([chr(ord(c)^0x1) for c in "5385876381"])
'4294967290'
```

Oregon State University

In Week1 Level5

- XOR-ing each character of "5385876381" with 0x1
  - `''.join([chr(ord(c)^0x1) for c in "5385876381"])`
- Description
  - `for c in “5385876381”` # for each character c in the string..
  - `ord(c)` # convert a character into an ascii value (int)
  - `chr(c)` # convert an ascii value (int) to a character
  - `''.join([‘a’,’b’,’c’])` # concatenate all elements in an array as a string, “abc”

In Week1 Level7

- Inverting memfrob()

In Week1 Level7

- Inverting memfrob()
- XOR-ing each character of a string with 42

In Week1 Level7

- Inverting memfrob()
- XOR-ing each character of a string with 42
  - `''.join([chr(ord(c)^42)) for c in string])`

In Week1 Level7

- Inverting add_a_space()

In Week1 Level7

- Inverting add_a_space()
- add_a_space adds 0x20 to each character…

In Week1 Level7

- Inverting add_a_space()
- add_a_space adds 0x20 to each character…
- Let's subtract 0x20 per each character…
  - `''.join([chr((ord(c)-0x20+0x100)&0xff) for c in string])`
  - `ord(c) – 0x20  # good,`
  - `+0x100  # why +0x100? For negative values..`
  - `&0xff  # why &0xff? To limit the value under 256..`
  - `(&0xff is to get the remainder when the number is divided by 256…)`

Hexadecimal Integers -> Decimal Integers

- Run python
- Type the number

```
$ python
Python 2.7.12 (d
[GCC 5.4.0 20160
Type "help", "co
>>> 0x41414141
1094795585
>>> 0xffffffff
4294967295
>>> 0xfffffffa
4294967290
```

Decimal Integers -> Hexadecimal Integers

- hex(value)
  - Returns hexadecimal string…

```
>>> hex(1094795585)
'0x41414141'
>>> hex(4294967295)
'0xffffffff'
>>> hex(4294967290)
'0xfffffffa'
```

To/from little endian string

- Int to little endian
  - from pwn import *
  - p32(0x41414141)
  - p64(0x41414141)
- Inverse?
  - u32(“\x41\x41\x41\x41”)
  - u64(“\x41\x41\x41\x41\x00\x00\x00\x00”)

```
>>> from pwn import *
>>> p32(0x41414141)
'AAAA'
>>> p64(0x41414141)
'AAAA\x00\x00\x00\x00'
>>> u32("\x41\x41\x41\x41")
1094795585
>>> u64("\x41\x41\x41\x41\x00\x00\x00\x00")
1094795585
>>> hex(1094795585)
'0x41414141'
```

Use pwntools to interact with the program

- p = process(“./program-name”)
- p.send(“asdf”) # send ‘asdf’ as an input
- p.sendline(“asdf”) # send ‘asdf\n’ as an input (newline!)
- p.recv(0x100) # recv upto 100 bytes
- p.recvline() # recv a line
- p.recvuntil(‘:’) # recv until output meets ‘:’
- p.interactive() # connect input/output to keyboard/console…

Use pwntools to interact with the program

- e = ELF(“./program-name”) # open the program as an ELF object
- e.symbols[‘get_a_shell’] # find the address of ‘get_a_shell()’

- p32(address) # get a little endian string from an integer (32-bit)
- p64(address) # .. (64-bit)

Tips for bof-level7 and bof-level8

- You can change the last byte of saved ebp
  - E.g., 0xffffd540 -> 0xffffd510 or 0xffffd500, etc.
- Your buffer is at below the saved ebp
  - So you would like to make that saved ebp as a lesser value!
  - Minimal value = 0x00!
- What if your input is not on 0xffffd500? (change the last byte as 00)
  - Run the program with some arguments
  - E.g., ./bof-level7 a b c d e f g ….
  - Eventually, your saved ebp will be 0xffffd4f0
    - -> 0xffffd400? 0xffffd480? Yes, you can control them.