hw7 Writeup

School/Grade: 交大資科工所 碩一

Student ID: 309551004 (王冠中)

ID: aesophor

Going Crazy (200 pts)

- **File**

```
gogo: ELF 64-bit LSB executable, x86-64, version 1 (SYSV),
statically linked, Go BuildID=bWoDILdD603fUIBmlGJS/duyHh-
_NeTNpIDsXZ4ip/Ol8BUeq8DSNmXb7rqRYB/JLKVQcvRvFU2oFSAsqux, stripped
```

- **Checksec**

```
Arch:     amd64-64-little
RELRO:    No RELRO
Stack:    No canary found
NX:       NX enabled
PIE:      No PIE (0x400000)
```

- **Overview**
  - 從 `file` 的結果可以知道這題是 Golang 的 binary，且沒有留下 symbols
  - 這支程式需要使用者輸入一串 input，其中 input 的頭和尾必須是 `x` 這個字元
  - input 頭尾的兩個 `x` 中間需要包含 0x24 (36) 個數字，並且以 `,` 區隔
  - 範例 input：x0,1,2,3,...,34,35x
  - 接著程式會將 [0 1 2 3 ... 35] 這 36 個數字進行某種數學運算，在 bezu() 中會有一項檢查，我們需讓 rcx == rdx，否則函數會直接 return
  - 只要我們的每個數字都能讓 rcx == rdx，loop 就能順利掃描完 36 個數字
  - 掃描完畢會印出 `Crazy!Crazier!Craziest` ，而此時我們的數字就是 flag 了（轉ASCII）

- **Static Analysis**
  - 用 cutter 開啟可以還原 function 名。以下是比較重要的幾個：

```
main.main()
main.check_input()
main.bezu()
main.rchvf()
```

  - main.main() 的邏輯大概是從 stdin 吃一個字串，去頭去尾 (兩個 `x` 字元) 後將中間的 36 個數按照 `,` 切開，並且擺進一個 slice，然後把 slice 丟到 main.check_input()

```
func main() {
  var input string

  for {
    fmt.Println("+================+")
    fmt.Println("|    Go Crazy    |")
    fmt.Println("+================+")
    fmt.Println("Say something :")
    fmt.Scanln(&input)

    if input[0] == 'x' && input[len(input)-1] == 'x' {
      tokens := strings.Split(input[1:len(input)-1], ",")
      check_input(tokens)
    }

    fmt.Println("no ! tOo NORmAL ! yoU hAvE To Be crAzY eNoUgh...")
  }
}
```

  - main.check_input() 只有一個參數，是剛剛那個包含 36 個數字的 slice。吃進來之後，會按照一個奇怪的順序掃描這 36 個數字（而不是 linear search）

```
func check_input(tokens []string) {
  // ...
  i, _ := strconv.Atoi(tokens[i])
  bezu(i, 0xfbc56a93, arg3, arg4)
}
```

  - main.bezu() 裡面又會呼叫 main.rchvf() 來進行一連串的數學運算，這裡面看得不是很懂，感覺 rchvf() 好像是在做輾轉相除法，但不太確定。

- **Dynamic Analysis**
  - 經過 gdb 觀察，在 check_input() 中，只要任何一個數字被 bezu() 檢查完後 `rcx != rdx`，整個 function 就會返回 main() 並且要求使用者重新輸入。
  - 目前要檢查的數字的 index 可以用 gdb 在 `0x0048e6bd` 設 breakpoint，並且用 `printf "%d\n",$rcx` 印出 rcx 的值

```
0x0048e6b0      mov     rcx, qword [rsp + rax*8 + 0x30]
0x0048e6b5      mov     rdx, qword [arg_288h]
0x0048e6bd      cmp     rcx, rdx
```

  - 會發現這點是經過了無數次令人崩潰的嘗試
  - 接下來是 bezu() 後對 rcx 和 rdx 的檢查，只要 `rcx == rdx`， check_input() 就不會 return，也就是說他就會繼續對其餘數字進行處理。

```
0x0048e6f3      call    sym.go.main.bezu
0x0048e6f8      mov     rax, qword [var_28h]
0x0048e6fd      mov     rcx, qword [rsp + rax*8 + 0x150]
0x0048e705      mov     rdx, qword [var_8h]
0x0048e70a      cmp     rcx, rdx
0x0048e70d      je      0x48e6a7
0x0048e70f      mov     byte [arg_298h], 0
0x0048e717      mov     rbp, qword [var_270h]
0x0048e71f      add     rsp, 0x278
0x0048e726      ret
```

  - 第一個被處理的數字的 index 是 0xf (15)，用 gdb 從 1 爆到 114，終於發現 114 可以讓 bezu() 後的 `rcx == rdx`，然後我們就不會被彈回 main()

- **Exploitation**
  - 我們可以試著爆出全部的數字，讓 36 個數字都被處理完，看看會發生什麼事
  - 36 個數字能被處理完的條件是：我們要找出 36 個不同且正確的數字，讓 bezu() 後的 `rcx == rdx`
  - 第一個數字我手動爆出來是 index = 15 的地方，必須放 114

```
x0,1,2,3,4,5,6,7,8,9,0,11,12,13,14,114,16,17,...34,35x
```

- 手動搜了前幾個感覺都是 printable ASCII，所以感覺這 36 個數字就是 flag
- 但是如果要手動爆出其他所有數字，實在太耗時了，所以我就寫了一個 gdb extension 來自動化爆搜流程。其中幾個關鍵的點是：
  - `0x0048e6bd` 處抓出 rcx 的值，可以知道目前處理的數字的 index
  - `0x0048e70a` 處抓出 rcx, rdx 的值，如果相等代表我們目前嘗試的數字是對的
  - 剩下的就是用兩層 for loop 去爆搜

```
#!/usr/bin/env python3
# -*- encoding: utf-8 -*-
'''
[HW 0X07] Going Crazy - NCTU SQLAB 309551004 王冠中
usage: gdb -batch-silent -x ./exploit.gdb ./gogo
'''
import sys
import gdb

class Exploit(gdb.Command):
    def __init__ (self):
        super(Exploit, self).__init__("exploit", gdb.COMMAND_SUPPORT)
        self.payload = [str(i) for i in range(36)]

    def invoke(self, arg, from_tty):
        print('[*] init...')
        gdb.execute('set pagination off')
        gdb.execute('set confirm off')
        gdb.execute('break *0x0048e6bd')
        gdb.execute('break *0x0048e70a')

        target_idx = 0
        for i in range(0, len(self.payload)):
            for num in range(30, 128):
                self.payload[target_idx] = str(num)
                self.prepare_payload()
                self.maybe_print_flag()
                gdb.execute('r < payload.txt > /dev/null')

                for k in range(i):
                    gdb.execute('c')
```

```
                    gdb.execute('c')

                # breakpoint 1: 0x0048e6bd (right before `call bezu()`)
                target_idx = self.get_current_idx()
                gdb.execute('c')

                # breakpoint 2: 0x0048e70a (right before `cmp rcx, rdx`)
                if self.are_rcx_edx_equal():
                    gdb.execute('c')
                    target_idx = self.get_current_idx()
                    break

    def prepare_payload(self):
        with open('payload.txt', 'w') as f:
            f.write('x' + ','.join(self.payload) + 'x')

    def maybe_print_flag(self):
        flag = ''.join([chr(int(n)) for n in self.payload])
        if flag.startswith('FLAG{'):
            sys.stderr.write(flag + '\n')

    def get_current_idx(self) -> int:
        gdb.execute('set logging file idx.txt')
        gdb.execute('set logging overwrite on')
        gdb.execute('set logging on')
        gdb.execute('printf "%d\\n",$rcx')
        gdb.execute('set logging off')
        with open('idx.txt', 'r') as f:
            current_idx = int(f.read())
        return current_idx

    def are_rcx_edx_equal(self) -> bool:
        gdb.execute('set logging file reg.txt')
        gdb.execute('set logging overwrite on')
        gdb.execute('set logging on')
        gdb.execute('printf "%d\\n",$rcx')
        gdb.execute('printf "%d\\n",$rdx')
        gdb.execute('set logging off')
        with open('reg.txt', 'r') as f:
            rcx, rdx = f.read().split()
        return rcx == rdx

Exploit()
```

然後開啟 vanilla gdb（pwndbg 會有問題，peda 不確定）

```
> gdb -q ./gogo
```

```
Reading symbols from ./gogo...
(No debugging symbols found in ./gogo)
(gdb) source exploit.py
(gdb) exploit
```

source 後會執行 `Exploit::invoke()` 並開始爆搜。

- **Flag**

  我的 exploit 跑大概 1 分鐘就可以爆搜到 flag

```
total 1.3M
-rwxr--r-- 1 aesophor aesophor 1.3M Dec  1 12:26 gogo
-rw-r--r-- 1 aesophor aesophor   26 Dec 11 17:15 exploit.gdb
-rwxr-xr-x 1 aesophor aesophor 2.5K Dec 11 17:51 exploit.py

/home/aesophor/CTF/nctu-secure-programming-2020-fall/hw7-reverse [git::master]
17:52]
> gdb -batch-silent -x ./exploit.gdb ./gogo
FLAG{gogo_p0werr4ng3r!you_did_IT!!!}

/home/aesophor/CTF/nctu-secure-programming-2020-fall/hw7-reverse [git::master]
17:53]
>
```