Lec02: x86_64 / Shellcode / Tools

Taesoo Kim

Administrivia

- Survey: how many hours did you spend? (<3h, 6h, 10h, >20h)
- Please join Piazza
- An optional recitation at 5-7pm on every Wed (in **CoC 052**)
- Lab02 is already out! (8pm every Thursday)
- Due: Sept 6th at midnight

About Write-up

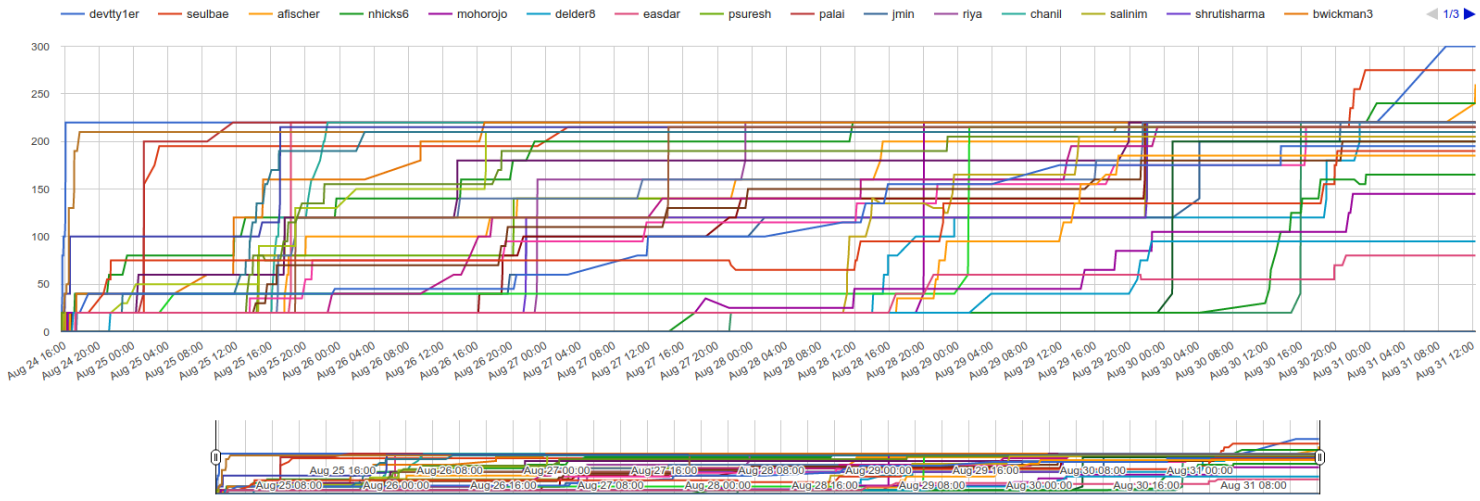
1) Write-up:

In this problem, ebp and ret value are protected by gsstack. while debugging, you can see all ebp and ret values are keep tracking and storing somewhere. However, when you make an input large enough, you will see that a function pointer will be overwritten. And the overwritten value will be store in EAX and make it jump at <main+96>. I put my shellcode as env, get the address, and put it. In my case, the function pointer(0x08048b0a at 0xbffff654) was overwritten. So we could learn, we could jump using the weakpoint even though the stackshiled is working on.

2) Exploit:

```
$(python -c 'print "\x90"*108+"\x90"*44+"\x87\xf8\xff\xbf"+"\x90"*50')
```

Scoreboard


devtty1er
seulbae
afischer
nhicks6
mohorojo
delder8
easdar
psuresh
palai
jmin
riya
chanil
salinim
shrutisharma
bwickman3
1/3
300
250
200
150
100
50
0
Aug 24 16:00
Aug 24 20:00
Aug 25 00:00
Aug 25 04:00
Aug 25 08:00
Aug 25 12:00
Aug 25 16:00
Aug 25 20:00
Aug 26 00:00
Aug 26 04:00
Aug 26 08:00
Aug 26 12:00
Aug 26 16:00
Aug 26 20:00
Aug 27 00:00
Aug 27 04:00
Aug 27 08:00
Aug 27 12:00
Aug 27 16:00
Aug 27 20:00
Aug 28 00:00
Aug 28 04:00
Aug 28 08:00
Aug 28 12:00
Aug 28 16:00
Aug 28 20:00
Aug 29 00:00
Aug 29 04:00
Aug 29 08:00
Aug 29 12:00
Aug 29 16:00
Aug 29 20:00
Aug 30 00:00
Aug 30 04:00
Aug 30 08:00
Aug 30 12:00
Aug 30 16:00
Aug 30 20:00
Aug 31 00:00
Aug 31 04:00
Aug 31 08:00
Aug 31 12:00

Best Write-ups for Lab01

- bomblab1_01: nhicks6, burak
- bomblab1_02: nhicks6, gkamuzora3
- bomblab1_03: seulbae, riya
- bomblab1_04: nhicks6, riya
- bomblab1_05: ibtehaj, easdar
- bomblab1_06: seulbae, riya
- bomblab1_07: mlanden, gkamuzora3
- bomblab1_08: nhicks6, gkamuzora3
- bomblab1_09: burak, seulbae
- bomblab1_10: nhicks6, fsang

Bomb Stats

- Bombs exploded ?? times in total?
- in ?? phases?
- ?? people exploded at least once?

Bomb Stats

- Bombs exploded 68 times in total (68 x -5 = -340 pts)
- in 9 phases!
- 15 people exploded at least once! (so how many alive?)
  - Each lab: 39/39/39/39/39/38/38/37/36/32/36 people
  - Each lab: 00/26/14/04/08/01/04/01/04/01/05 times

Discussion 0

1. How does the bomb notify the explosion to the server?

Discussion 1

1. How did you prevent bombs from explosion?

Discussion 2

1. What made your bombs exploded?

Discussion 2

1. What was the most difficult/annoying phase?

Discussion 3

1. How did you find ‘secret_phrase’?

Discussion 4

1. Any tricky assembly?

ASMs that you read in Lab1

- function calls (phase_funcall)
- switch: jump table (phase_jump)
- for/while loops (phase_quick)
- recursion (phase_binary)
- data structure: array/list/tree
- etc

ASM Show Case 1: funcall

```
push   0x804b96b              ; -> "scissors"
push   0x804b974              ; -> "paper"
push   0x804b97a              ; -> "rock"
push   DWORD PTR [ebp+0x8]    ; -> ????
call   8049d0b <func_game>
```

ASM Show Case 2: switch (jump table)

```
        cmp    eax,0x7                                  ; ???
        ja     0x8049e09 <phase_jump+147> ------+       ; ???
        mov    eax,DWORD PTR [eax*4+0x804b948]  |       ; ???
     ** jmp    eax                              |
                                                |
 <+75>: mov    DWORD PTR [ebp-0xc],0x25b        |
        jmp    0x8049e0e <phase_jump+152>       |
 <+84>: mov    DWORD PTR [ebp-0xc],0x232        |
        jmp    0x8049e0e <phase_jump+152>       |
        ...                                     |
<+147>: call   0x8049a3f <explode_bomb>   <-----+
        mov    eax,DWORD PTR [ebp-0x18]
```

ASM Show Case 2: switch (jump table)

```
> telescope 0x804b948
00| 0x804b948→ phase_jump+75  ←mov dword ptr [ebp - 0xc], 0x25b
04| 0x804b94c→ phase_jump+84  ←mov dword ptr [ebp - 0xc], 0x232
08| 0x804b950→ phase_jump+93  ←mov dword ptr [ebp - 0xc], 0x282
0c| 0x804b954→ phase_jump+102 ←mov dword ptr [ebp - 0xc], 0x16c
10| 0x804b958→ phase_jump+111 ←mov dword ptr [ebp - 0xc], 0x2af
...
```

ASM Show Case 2: switch (jump table)

```
switch(index) {
  case 0: ...
  case 1: ...
  case 7: ...
  default ...
}
```

ASM Show Case 3: for/while loops

```
        mov    DWORD PTR [ebp-0x18],eax            ; p
        mov    DWORD PTR [ebp-0xc],0x0             ; i = 0
        jmp    0x8049f7c <phase_array+92> ----+    ;
                                              |
<+69>:  add    DWORD PTR [ebp-0xc],0x1   <----|--+ ; i ++
        mov    eax,DWORD PTR [ebp-0x18]       |  | ;
        mov    eax,DWORD PTR [eax*4+0x804e5a0]|  | ; p = ((int *)0x804e5a0)[
        mov    DWORD PTR [ebp-0x18],eax       |  |
        mov    eax,DWORD PTR [ebp-0x18]       |  |
        ...                                   |  |
<+92>:  mov    eax,DWORD PTR [ebp-0x18]    <--+  | ; p
        cmp    eax,0xf                           | ;
        jne    0x8049f65 <phase_array+69> -------+ ; while (p != 15)
```

Lab02: Bomb Lab2 / Shellcode

- Another Bomblab (be extra careful this time)!
- Writing five different shellcodes
  - x86, x86_64, both!, ascii, minimal size (competition)
  - Bonus : the smallest shellcode gets extra **10 pts**!

Today's Tutorial

- x86 shellcode overview
- In-class tutorial
  - pwndbg (modernizing gdb for reverse engineering)
  - IDA (interactive disassembler)
  - Walk over x86 shellcode (+ excercise!) and various tools

DEMO: pwndbg commands

- vmmap
- procinfo/elfheader
- telescope/hexdump
- context/stack/regs
- nearpc
- pdisass
- search

shellcode (in C)

```c
#include <stdio.h>
#include <unistd.h>

int main() {
  char *sh = "/bin/sh";
  char *argv[] = {sh, NULL};
  char *envp[] = {NULL};
  execve(sh, argv, envp);
  return 0;
}
```

DEMO: shellcode.S

- explain: asm, structure
- `man syscall` (about convention)
- execve()
- debugging shellcode/target
- tutorial: /bin/sh to /bin/cat /proc/flag

In-class Tutorial

- Step 1: Install pwdbg/IDA
- Step 2: Play with shellcode!

```
$ ssh lab02@cyclonus.gtisc.gatech.edu -p 9002
or
$ ssh lab02@computron.gtisc.gatech.edu -p 9002
Password: lab02

$ cd tut02-shellcode
$ cat README
```

References

- Assembly
- x86
- x86_64
- pwndbg