Network Security Project
Software Vulnerability

Elin Ho, JuiChien Jao

Outline

- Introduction
- Project 3 Goal
- Buffer Overflow
- Shellcode
- Project 1 ~ Project 2
- Reference

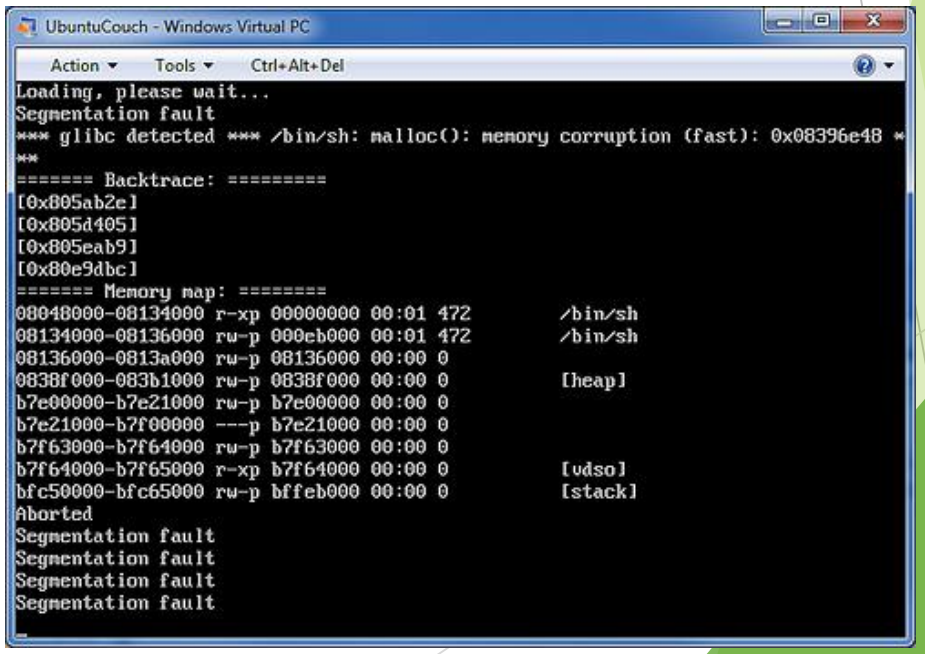
Project 3

Binary Program Security

Project 3 - Binary Program Security

- Many software systems are implemented in low level programming language(usually C and C++)
  - It's too complicated to manipulate memory access correctly, memory corruption bugs are widely happened

- *Types of* memory corruption bug
  - *Buffer Overflow*
  - *format string*
  - *Use-After-Free*



Project 3 - Goal

- Essential
  - Understand why software vulnerability effect network security threat
  - Understand some secure programming concepts
  - Practice to attack BOF vulnerability
  - Scanning Lan network using command `nmap`
  - ARP spoofing
- Optional
  - Practice tools for reversing program
  - Write some network program

Project 3 - Steps

- Student should follow these steps
  1. Connect the server’s vulnerable program
  2. Exploit the program – override some data
  3. Exploit the program – get the control right of the system (shell)
  4. Find some secret in server
  5. Scanning Lan network using command `nmap`
  6. ARP spoofing

Buffer Overflow

Bug → Vulnerability

Buffer Overflow – Steps 1 Find Entry Point

- Find every where you can input anything
- Read code
  - Source code
    - Almost cannot get it
  - Assembly code
    - Binary file
- Fuzz testing
  - Crash
  - Illegal output / Unexpected output

Buffer Overflow – Steps 2 Control-flow Hijack

- Try to control the control-flow
  - Return address
  - Function pointer
  - Variable value

Buffer Overflow – Steps 3 Execute Payload

- Run shell
- Capture data
- Put backdoor in system

Buffer Overflow

Find Entry Point

Buffer Overflow - Vulnerable Function

- gets
- scanf (%s)
- sprintf
- strcpy
- strcat

Buffer Overflow - Vulnerable Function

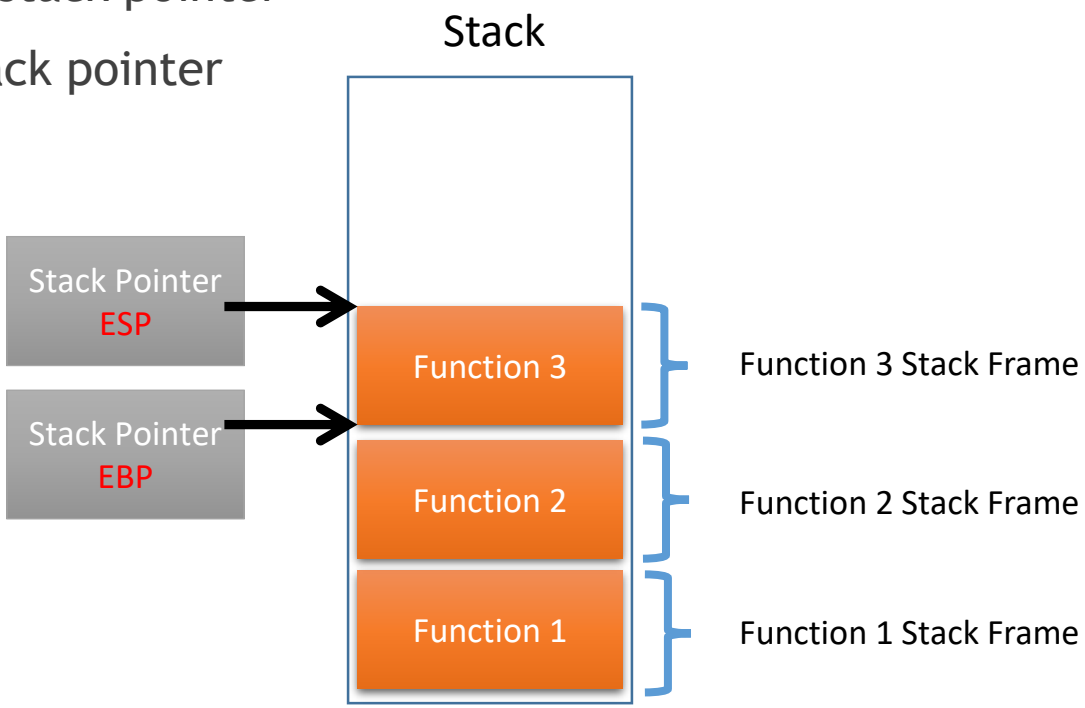
- ~~gets~~ → fgets
- ~~scanf (%s)~~ → scanf(%99s)
- ~~sprintf~~ → snprintf
- ~~strcpy~~ → strncpy
- ~~strcat~~ → strncat

Buffer Overflow - Stack Buffer Overflow

- Buffer overflow usually happened on stack
- Overwrite the return address
- ‘stack overflow’ alias ‘stack smashing’

Buffer Overflow - Stack

- ESP : current stack pointer
- EBP : Base stack pointer



Buffer Overflow - Stack

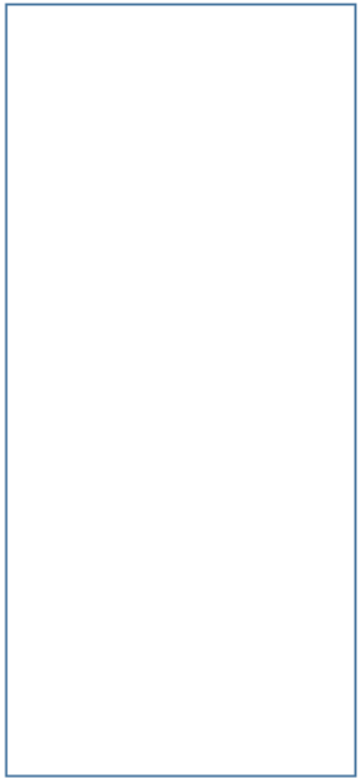
Caller Function

```
Int main(){
        add(1,2);
        return 0;
}
```

Callee Function

```
void add(intn1, intn2){
        intsum;
        sum = n1 + n2;
}
```

Stack



Buffer Overflow - Stack

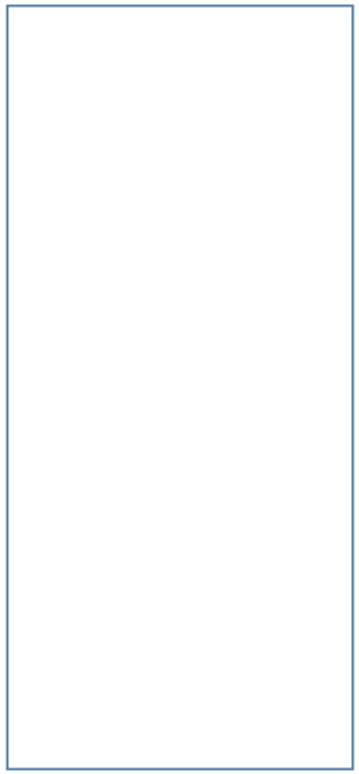
Caller Function

```
Int main(){
        add(1,2);
        return 0;
}
```

Callee Function

```
void add(intn1, intn2){
        intsum;
        sum = n1 + n2;
}
```

Stack



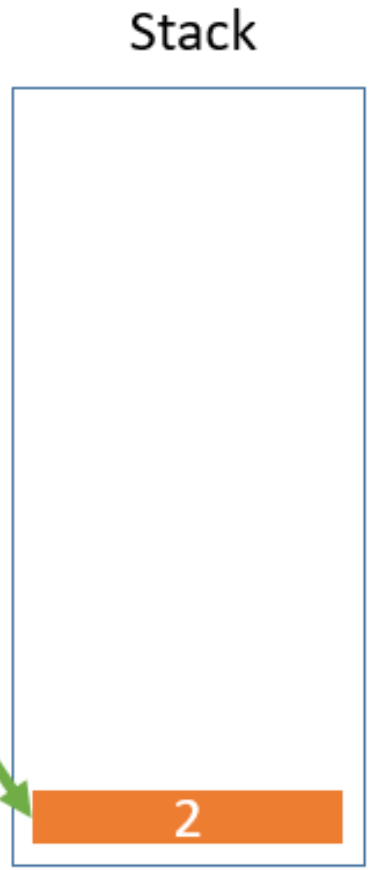
Buffer Overflow - Stack

Caller Function

```
Int main(){
        add(1,2);
        return 0;
}
```

Callee Function

```
void add(intn1, intn2){
        intsum;
        sum = n1 + n2;
}
```



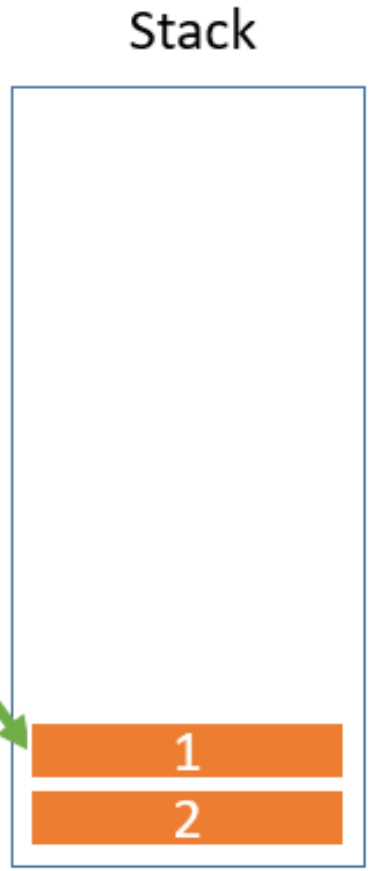
Buffer Overflow - Stack

Caller Function

```
Int main(){
        add(1,2);
        return 0;
}
```

Callee Function

```
void add(intn1, intn2){
        intsum;
        sum = n1 + n2;
}
```



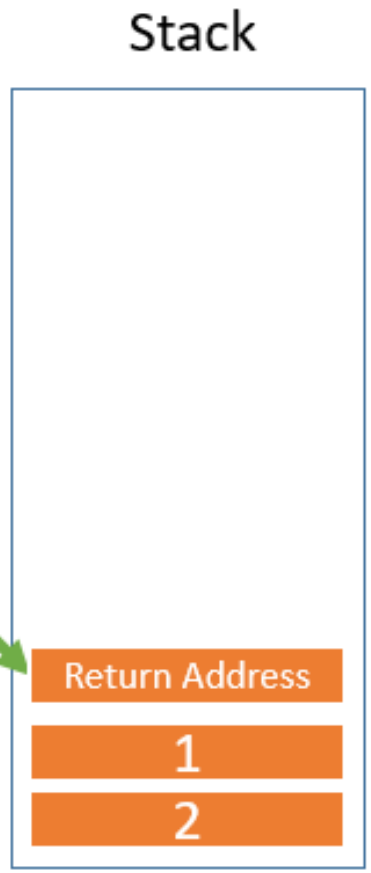
Buffer Overflow - Stack

Caller Function

```
Int main(){
        add(1,2);
        return 0;
}
```

Callee Function

```
void add(intn1, intn2){
        intsum;
        sum = n1 + n2;
}
```



Buffer Overflow - Stack

Caller Function

```
Int main(){
        add(1,2);
        return 0;
}
```

Callee Function

```
void add(intn1, intn2){
        intsum;
        sum = n1 + n2;
}
```

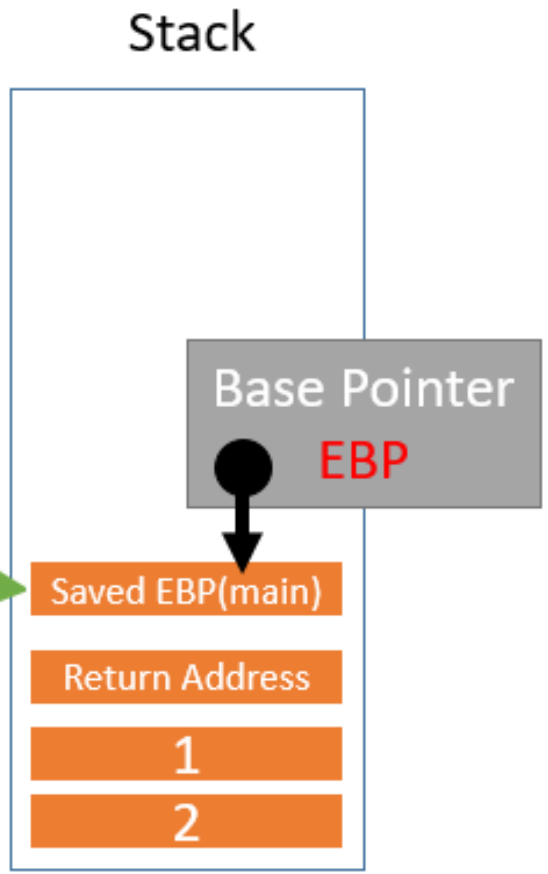
Buffer Overflow - Stack

Caller Function

```
Int main(){
        add(1,2);
        return 0;
}
```

Callee Function

```
void add(intn1, intn2){
        intsum;
        sum = n1 + n2;
}
```



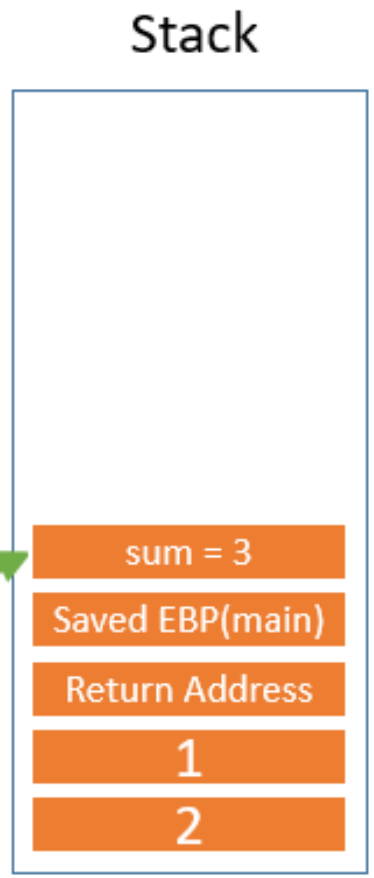
Buffer Overflow - Stack

Caller Function

```
Int main(){
        add(1,2);
        return 0;
}
```

Callee Function

```
void add(intn1, intn2){
        int sum;
        sum = n1 + n2;
}
```



Buffer Overflow - Stack

Caller Function

```
Int main(){
        add(1,2);
        return 0;
}
```

Callee Function

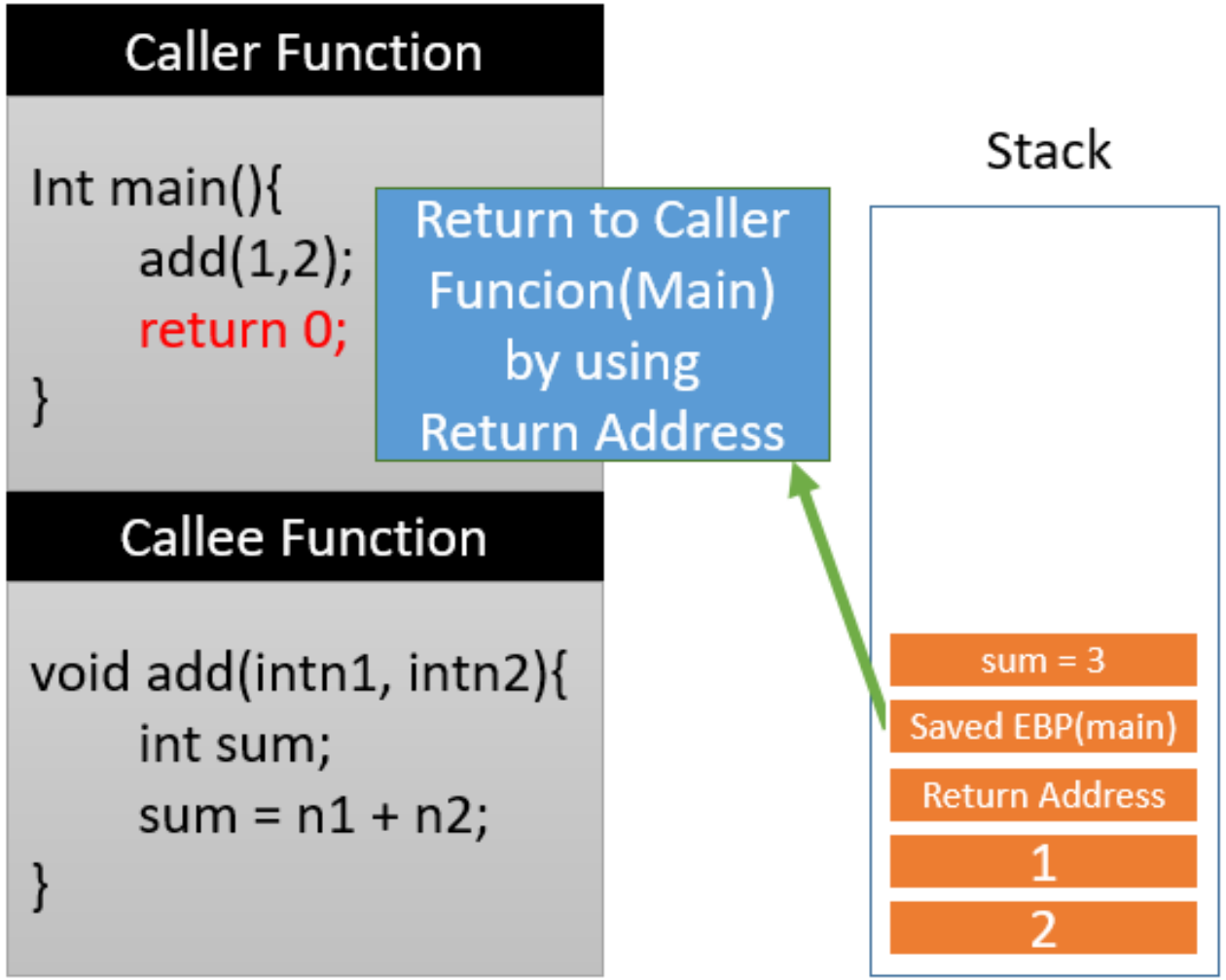
```
void add(intn1, intn2){
        int sum;
        sum = n1 + n2;
}
```

Restore the EBP (& Stack Pointer)

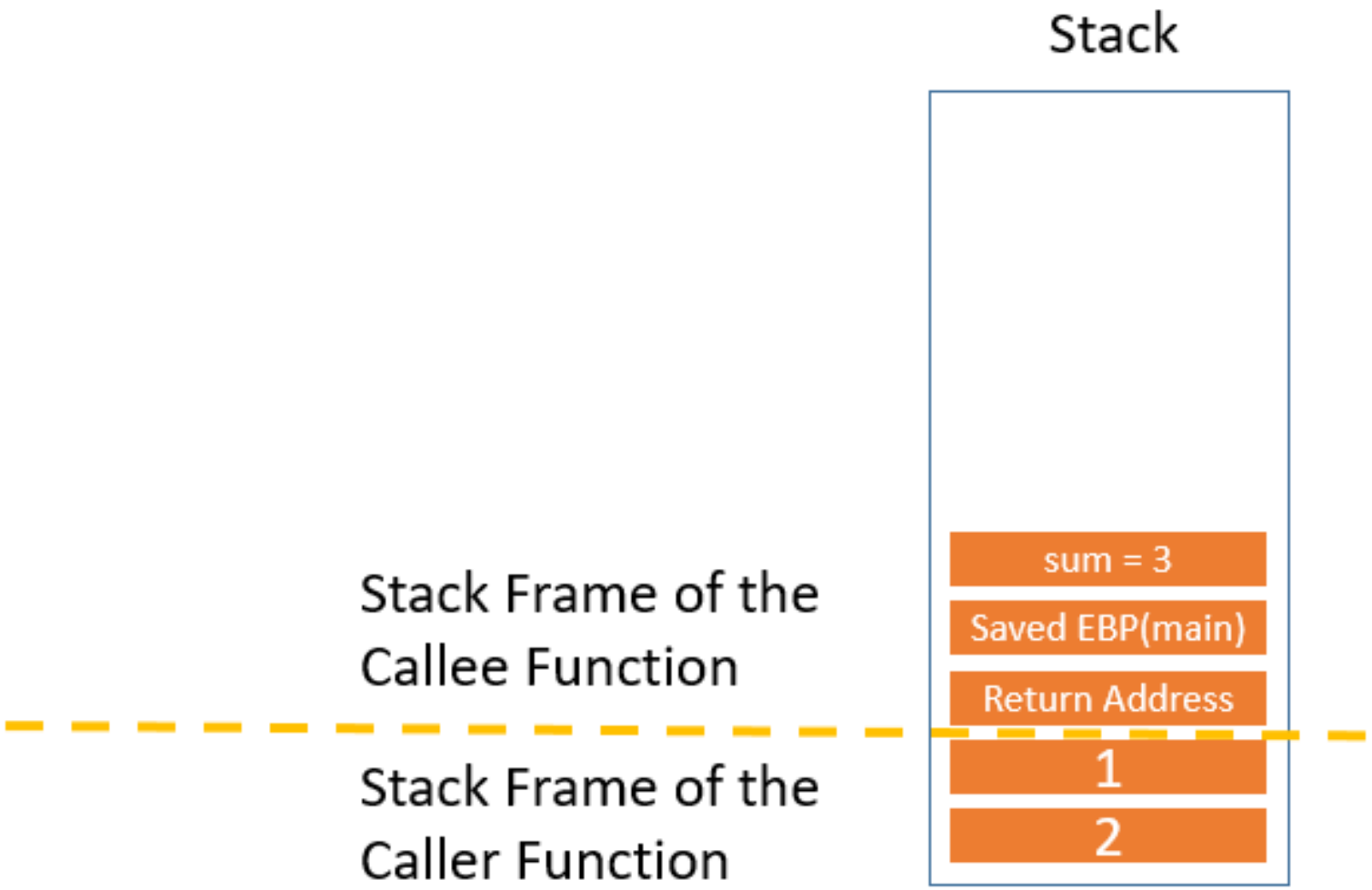
Stack

sum = 3
Saved EBP(main)
Return Address
1
2

Buffer Overflow - Stack



Buffer Overflow - Stack



mov	Copy the value of src operand to dst operand
push	Stores a value onto the top of the stack
pop	Loads a value from the top of the stack
call	Call the function(ex. call MessageBoxA)
cmp	Compare the value of src operand and dst operand
jmp(je, jg, jle)	Go to designated address(jmp is unconditional jump) je:jump if equal, jg:jump if greater, jle:jump if less or equal
ret	Return to called function
leave	Same as “mov esp, ebp”→”pop ebp”
add	Addition
sub	Subtraction

Buffer Overflow

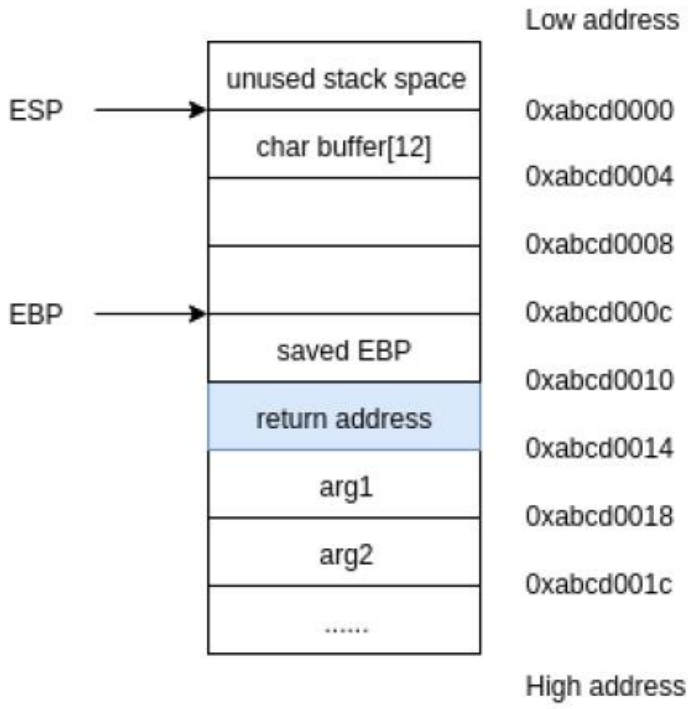
- **Return Hijack !!!**

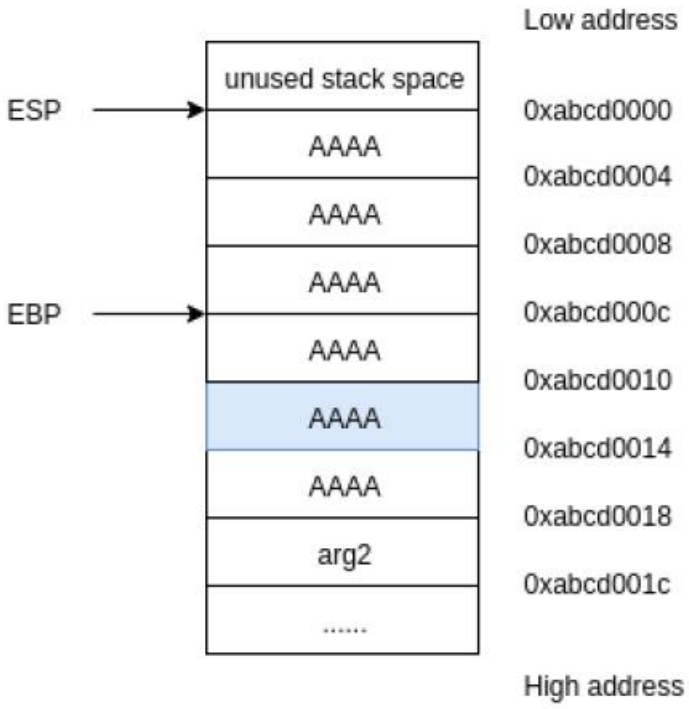
gets.c

```c
#include <stdio.h>

void hacked() {
    puts("Hacked!!");
}

int main() {
    char str[10];
    gets(str);
}
```

Buffer Overflow

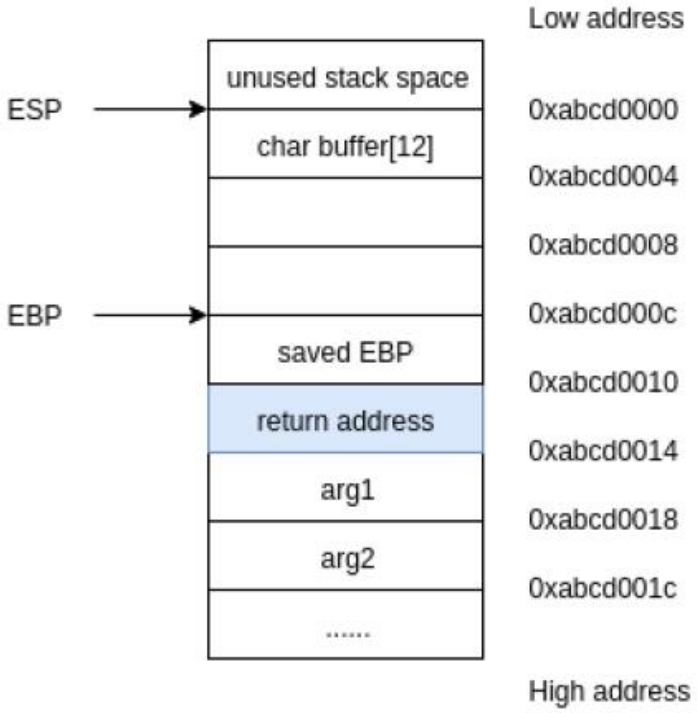
- **Let it crash !!!**

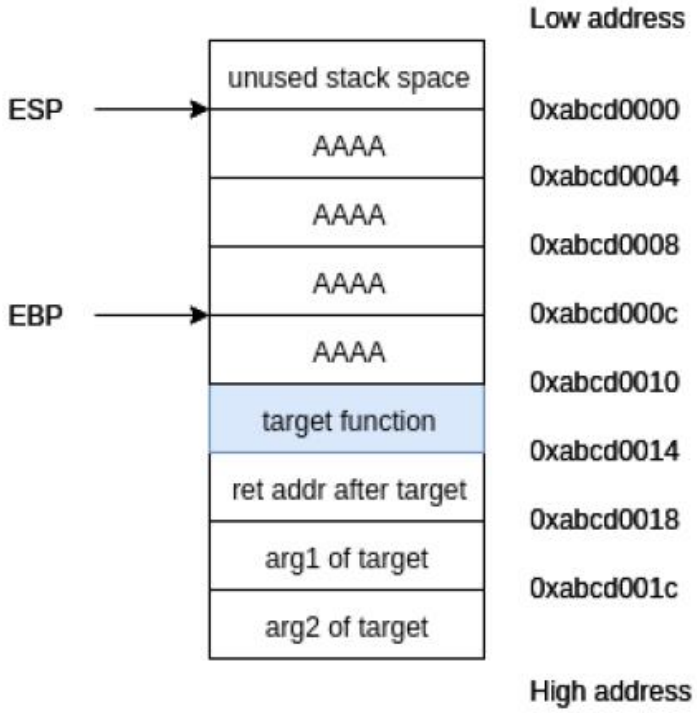




Buffer Overflow

- **From Crash to Hack !!!**





Shellcode

Shellcode

- An executable machine code put on the return address
- execve(“/bin/sh”)

```
#include <stdio.h>

int main(){
        char shellcode[] = \
                "\x31\xc0\x48\xbb\xd1\x9d\x96\x91\xd0\x8c\x97\xff\x48\xf7\xdb\x53\x54\x5f\x99\x52\x57\x54\x5e\xb0\x3b\x0f\x05";
    (*(void(*)()) shellcode)();
    return 0;
}
```

Shellcode Database

- Pay attention
  - operating system
  - instruction architecture
- http://shell-storm.org/shellcode/
  - Go the find which you can use

FreeBSD

Intel x86-64

- FreeBSD/x86-64 - execve - 28 bytes *by Gitsnik*
- FreeBSD/x86-64 - bind_tcp with passcode - 127 bytes *by Gitsnik*
- FreeBSD/x86-64 - exec(/bin/sh) Shellcode - 31 bytes *by Hack'n Roll*
- FreeBSD/x86-64 - execve /bin/sh shellcode 34 bytes *by Hack'n Roll*
- FreeBSD/x86-64 - Execve /bin/sh - Anti-Debugging *by c0d3_z3r0*

Intel x86

- FreeBSD/x86 - execve /tmp/sh - 34 bytes *by Claes M. Nyberg*
- FreeBSD/x86 - execve /bin/sh 23 bytes *by IZ*
- FreeBSD/x86 - reboot(RB_AUTOBOOT) - 7 bytes *by IZ*

Linux

ARM

- Linux/ARM - Add map in /etc/hosts file - 79 bytes *Osanda Malith Jayathissa*
- Linux/ARM - chmod("/etc/passwd", 0777) - 39 bytes *gunslinger_*
- Linux/ARM - creat("/root/pwned", 0777) - 39 bytes *gunslinger_*
- Linux/ARM - execve("/bin/sh", [], [0 vars]) - 35 bytes *gunslinger_*
- Linux/ARM - Bind Connect UDP Port 68 *by Daniel Godas-Lopez*
- Linux/ARM - Bindshell port 0x1337 *by Daniel Godas-Lopez*
- Linux/ARM - Loader Port 0x1337 *by Daniel Godas-Lopez*
- Linux/ARM - ifconfig eth0 and Assign Address *by Daniel Godas-Lopez*
- Linux/ARM - chmod("/etc/shadow", 0777) Shellcode - 35 Bytes *by Florian Gaultier*

Shellcode

- Test Shellcode
- gcc-z execstack shellcode.c
  - -z execstack : enable “execute code on stack”

Project 1 ~ Project 2

Project 1.2 and 1.3

- Generate RSA public and private key
  - Libraries
  - OpenSSL commands
- AES with CBC mode
  - Libraries
    - AES key
    - Initial Vector

Project 2.1

- Man in the middle(with assumptions)
  - ARP spoofing has been done
  - There is no PKI
  - Alice will want to check her bank money if you ask her to do it (…)
- Points deduction
  - Response message with wrong Account_ID
  - No response message back to Alice
  - “Use SHA256 to ‘encrypt’ my student id”
    - How do you decrypt???

Project 2.2

- Key functions have be mentioned in description
- “objdump”
  - “objdump --help” or “man objdump”
- “|” is useful
- “/” for searching in “less”
  - Guide you in the assembly codes

- Random numbers cannot be easily predicted, otherwise they are not random
- Patching is needless

Reference

Google is your good friend

Tools and commands

- Tools
  - Idapro
- Commands
  - gdb
  - objdump [–d] [-s] [-j]
- Shellstorm
  - http://shell-storm.org/shellcode/

Attention

- In order to prevent buffer overflow
  - Canaries
    - Protect return address
  - Data Execution Prevention (DEP)
  - Address space layout randomization (ASLR)

  - After gcc-4.9 (include gcc-4.9)
    - function epilogue
      - It will save the

Disable all protect of gcc

- Enable gdb
  - -g
- In order to prevent buffer overflow
  - Canaries
    - -fno-stack-protector
  - Data Execution Prevention (DEP)
    - -z execstack
  - Address space layout randomization (ASLR)
    - sudo bash -c 'echo 0 > /proc/sys/kernel/randomize_va_space‘
      - Usually have not to close this
  - If you want to write your own code to practice, do not use gcc after gcc-4.9

Demo – gets.c

- You can run this program “Gets_BSD” on NCTU CS BSD workwtation

  ssh bsd[1-5].cs.nctu.edu.tw
- Maybe you need these
  - objdump –d
  - gdb
    - b
    - r
    - n / s
    - x /20x $rsp
- SAMPLE SOLUTION is ‘attack_BSD’

```c
#include <stdio.h>

void hacked() {
  puts("Hacked!!");
}

int main() {
  char str[10];
  gets(str);
}
```

Demo - shellcode

- You can run this program “shellcode” on NCTU CS LINUX workwtation

  ssh linux[1-5].cs.nctu.edu.tw

```c
#include <stdio.h>

int main(){
    char shellcode[] = "\x31\xc0\x48\xbb\xd1\x9d\x96\x91\xd0\x8c\x97\xff\x48\xf7\xdb\x53\x54\x5f\x99\x52\x57\x54\x5e\xb0\x3b\x0f\x05";
    (*(void(*)()) shellcode)();
    return 0;
}
```

Project 3

Coming soon…