



CS 526 Lab1

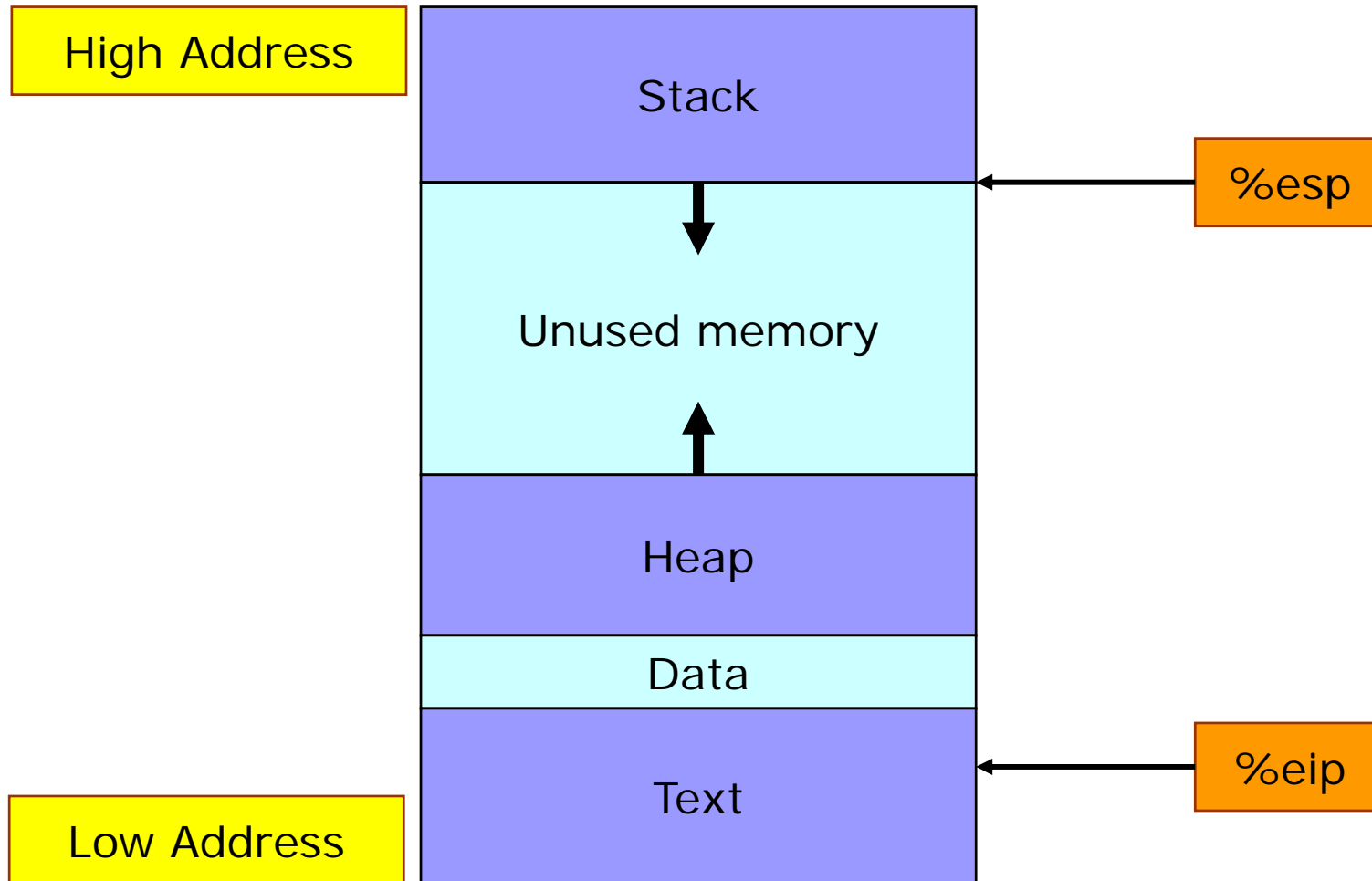
Ziqing Mao



Lab Overview

- You are asked to exploit software vulnerabilities
 - Stack overflow attacks
 - Return-to-libc attacks
 - Format string attacks
- 5 target programs
 - Write exploit programs
 - The frameworks for the exploits are provided
- The first step: clearly understand the function call mechanism and stack layout in C

Memory Layout Overview

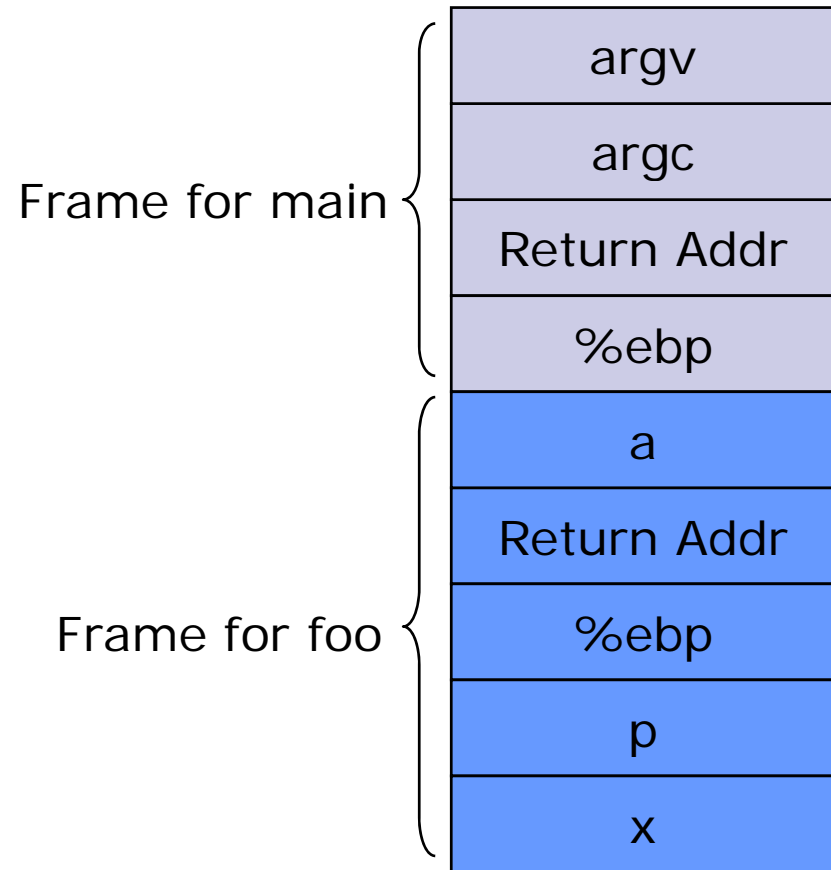


Stack Frame Layout

```
■ void foo(char* a)
{
    int p = 3;
    char x[4];

    strcpy(x, a);
}

int main(int argc, char** argv)
{
    foo("cat");
    foo("aaaabbbbccccdddd");
}
```

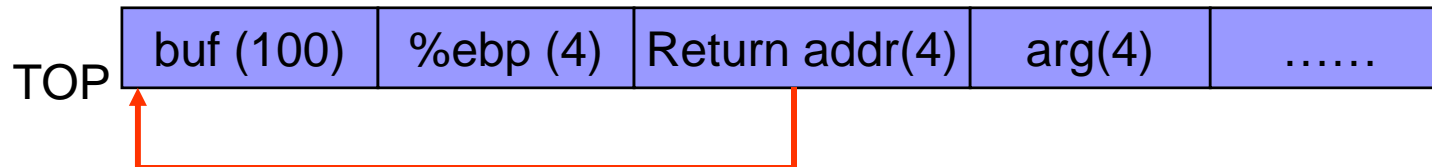


A Simple Example

- The vulnerable code

```
void foo(char* arg)
{
    char buf[100];
    strcpy(buf, arg);
}
```

- The stack layout before strcpy is invoked



- Exploit

- Put the shell code at the beginning of the buf
 - The shell code is provided in `exploits/shellcode.h`
- Overwrite the return address to point to buf
 - Need to find out the address of buf



Compiler Variants

- Extra space in the stack
 - alignment
 - compiler-dependent
- For the compiler used by the lab machine
 - 2 extra words (8 bytes) between local variable and %ebp
- Use gdb to figure out the exact layout of the stack



A quick tutorial for gdb

- Basic command

- start the debugger: `gdb [executable_name]`
- set a breakpoint: `b [function_name]`, `b [line_number]`
- run the program: `r`
- print the addresses: `print %ebp`, `print &buf`
- exit the debugger: `exit`

- <http://www.cs.princeton.edu/~benjasik/gdb/gdbtut.html>



Targets

- Target 1 (10 pts)
 - A program to check the correctness of the password
 - Goal: make the program accept your password
- Target 2 (20 pts)
 - A program to list files in a directory
 - Goal: to start a shell
- Target 3 (20 pts)
 - A program to check if a password is strong or weak
 - Goal: to start a shell



Return-to-libc Attack

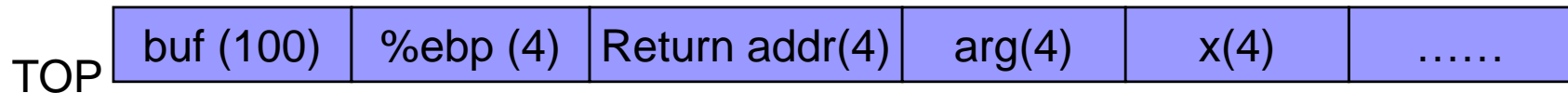
- Basic exploit for stack overflow
 - Inject shell code into the stack
 - Overwrite the return address to point to the injected code
- Non-executable stack
 - The injected code in the stack cannot be executed
- Use the existing libc function to launch attacks
 - libc functions are loaded by the operating system
 - E.g., to launch a shell using libc: `system("/bin/sh")`

A Simple Example

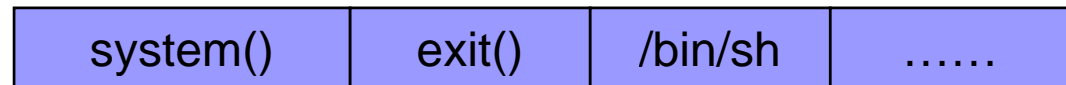
- The vulnerable code

```
void foo(char* arg, int x)
{
    char buf[100];
    strcpy(buf, arg);
}
```

- The stack layout before strcpy is invoked



- Exploit



- Overwrite the return address with the address of the function system()
- Overwrite "arg" with the address of the function exit() (optional)
- Overwrite "x" with the string "/bin/sh"



How to do that

- Need to find out (in addition to a normal stack overflow)
 - The address of the function system()
 - The address of the string “/bin/sh”
 - The address of the function exit() (optional)
- Read the reference
 - http://www.infosecwriters.com/text_resources/pdf/return-to-libc.pdf



Target 4 (25 pts)

- A program to copy the user input into a local buffer
- Goal: to start a shell using return-to-libc attack

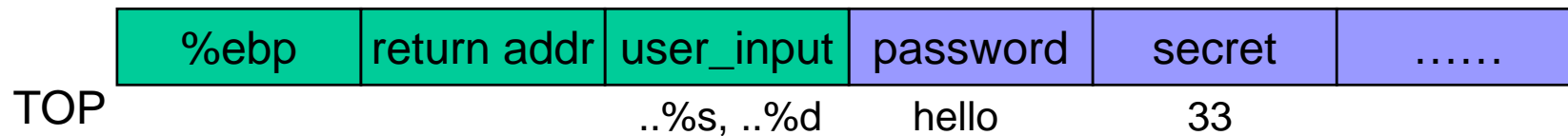
Format String Attacks

- An example

```
void func(char* user_input)
{
    int secret = 33;
    char password[] = "hello";
    printf(user_input);           //should be printf("%s", user_input)
}
```

- Exploit: func("password = %s, secret = %d")

- The stack layout when printf is invoked




- Output: password = hello, secret = 33



Format String Attacks (cont')

- Steal secret values
 - Print out a value stored in the stack
 - %d
 - Print out a value whose address is available in the stack
 - %s
 - Print out an a value stored in an arbitrary memory address
 - 1. Need to put the address in the stack, using normal stack overflow
 - 2. using %s

- Modify the value stored in an arbitrary memory address
 - %n



Target5 (25 pts)

- A program containing three secrets
- Goal I: Print out the secret values (15 pts)
- Goal II: Modify one of the secret values (10 pts)
- Follow the reference
 - <http://web.ics.purdue.edu/~zmao/formatstring-1.2.pdf>



Warming up

- C Programming
 - Function call mechanism
 - Stack layout
 - printf
- Use `printf("%x", &var)` to print out the address of a variable
- Use GDB
- Use Google



Environment Setup

- The OS is running in a virtual machine with RedHat 6.2

- Login
 - Connect to the host machine
 - `ssh lastname@forest.cs.purdue.edu`

 - Connect to the VM
 - `ssh cs526-lab`

 - Send email to TA as soon as possible to setup your initial password
 - try to make it strong

- Read the lab description carefully before you start



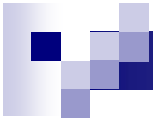
Grading

- The virtual machine will be shut down at the due time
- Leave your exploit programs in your home directory
 - ~/exploit/...
 - Other forms of code submission are **not** accepted
 - 80% credits
- Answer 8 questions in the lab descriptions
 - No lab report
 - Graded only when the corresponding exploit program works
 - 20% credits



More Hints

- Exploits codes are short
- Several ways to exploit
- Start early
- Codes from others may *not* work
- Make a copy before you make changes
- Prepare to demo: make your exploits stable
- Have fun 😊



Questions?