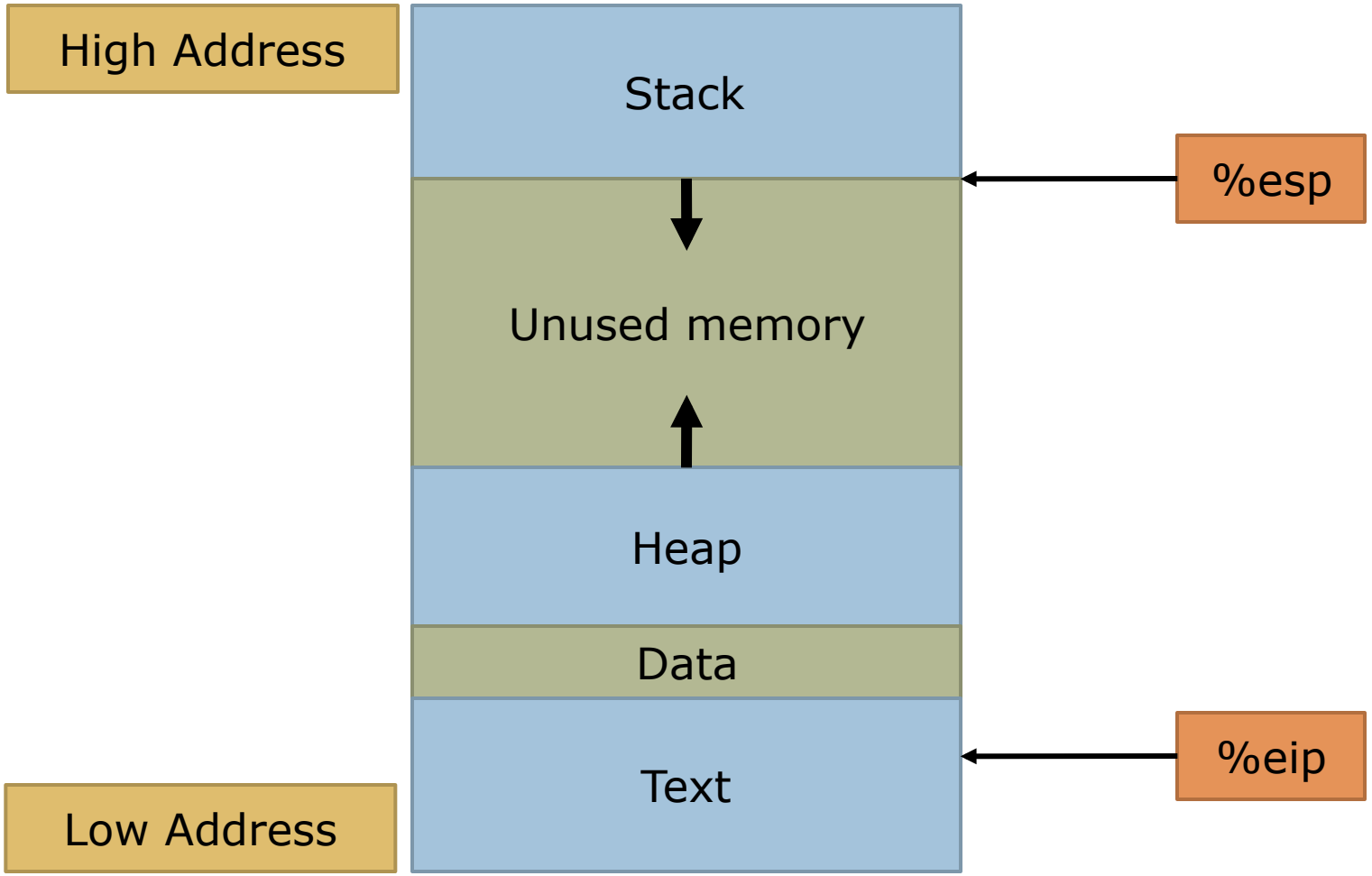
PROJECT 2 – SOFTWARE SECURITY

CS 6324

What is the project about?

- Understanding buffer overflow
- Exploiting some bugs
- Environment
  - Linux
  - Targets: C language
  - Exploits: in C or script

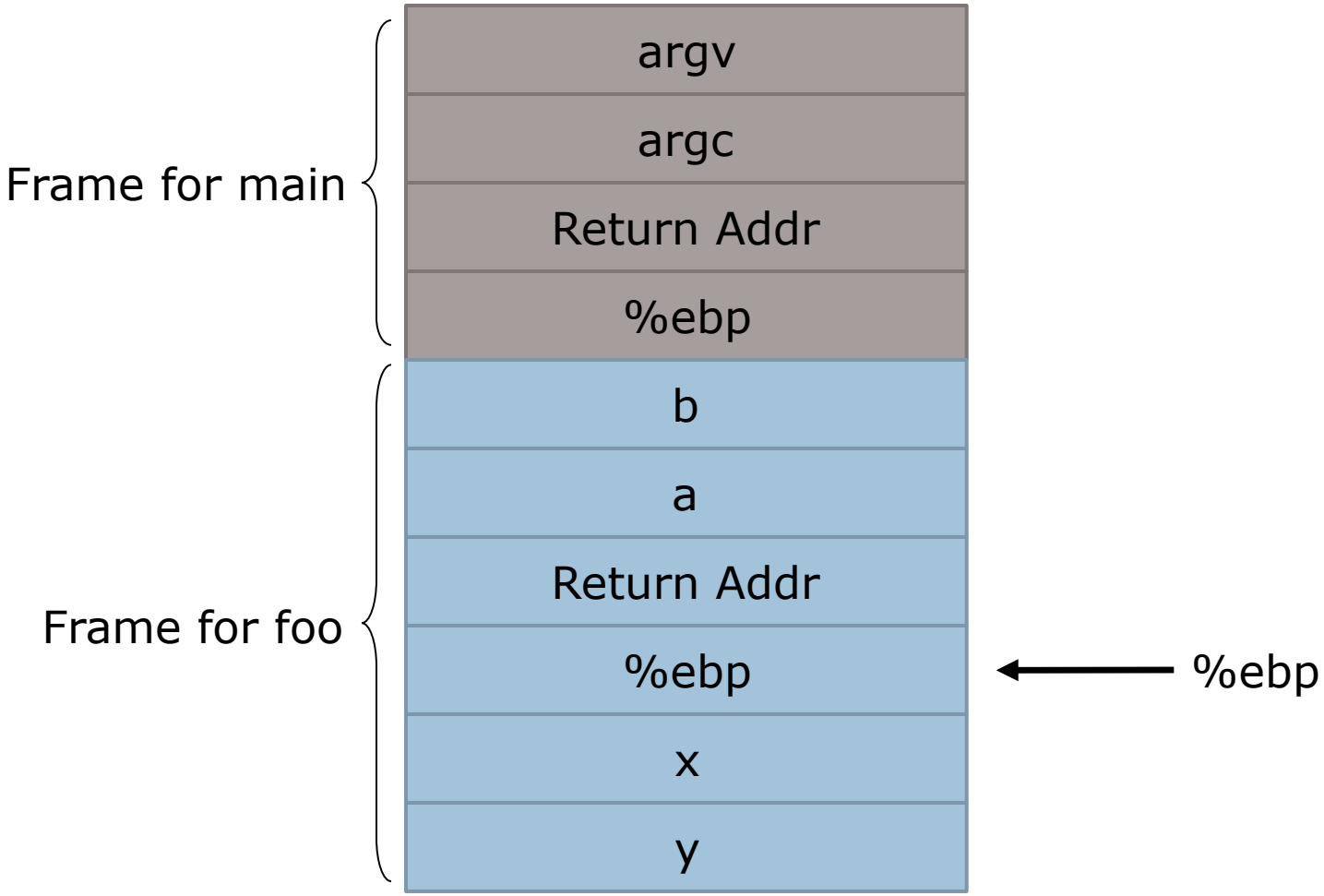
Memory Layout Overview



Function Call(1) example1.c

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

void foo(char * a, char * b)
{
    char x[8];
    char y[8];

    strcpy(x, a);
    strcpy(y, b);

    printf("x=%s y=%s\n", x, y);
}

int main(int argc, char ** argv)
{
    foo("Good", "Luck");
    return 0;
}
```

Function Call(2) function frame



Function Call(3) what happens?

- Caller
  - Push parameter(s) on stack
  - Push return addr
  - Jump to start addr of callee
- Callee
  - Push %ebp, %ebp ← %esp
  - Allocate space for local variables
  - …
  - %esp ← %ebp, Pop %ebp
- Return
  - Pop return addr, jump to the addr
  - Restore %esp

Function Call(4) assembly

```
0x8048400 <foo>:        push   %ebp
0x8048401 <foo+1>:      mov    %esp,%ebp
0x8048403 <foo+3>:      sub    $0x10,%esp
0x8048406 <foo+6>:      mov    0x8(%ebp),%eax
0x8048409 <foo+9>:      push   %eax
0x804840a <foo+10>:     lea    0xfffffff8(%ebp),%eax
0x804840d <foo+13>:     push   %eax
0x804840e <foo+14>:     call   0x8048340 <strcpy>
0x8048413 <foo+19>:     add    $0x8,%esp
0x8048416 <foo+22>:     mov    0xc(%ebp),%eax
0x8048419 <foo+25>:     push   %eax
0x804841a <foo+26>:     lea    0xfffffff0(%ebp),%eax
0x804841d <foo+29>:     push   %eax
0x804841e <foo+30>:     call   0x8048340 <strcpy>
0x8048423 <foo+35>:     add    $0x8,%esp
0x8048426 <foo+38>:     lea    0xfffffff0(%ebp),%eax
0x8048429 <foo+41>:     push   %eax
0x804842a <foo+42>:     lea    0xfffffff8(%ebp),%eax
0x804842d <foo+45>:     push   %eax
0x804842e <foo+46>:     push   $0x80484c0
0x8048433 <foo+51>:     call   0x8048330 <printf>
0x8048438 <foo+56>:     add    $0xc,%esp
0x804843b <foo+59>:     leave
0x804843c <foo+60>:     ret
```

Buffer Overflow example1b.c

- C doesn't check boundaries!

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

void foo(char * a, char * b)
{
    char x[8];
    char y[8];

    strcpy(x, a);
    strcpy(y, b);

    printf("x=%s y=%s\n", x, y);
}

int main(int argc, char ** argv)
{
    foo("Good", "Luck____Bad");
    return 0;
}
```

Project Instructions

- ~/
  - targets/
    - target1.c
    - target2.c
    - target3.c
    - target4.c
  - exploits/
    - [SOLUTIONS GO HERE]

Tasks #1: Simple Command Line Buffer Overflow

- target1:
  - A program that lists files in a directory
- Goal: to start a shell
- exploit1.sh: a shell script
- Do modern OS's prevent this attack?
  - Run outside VM and find out!

Task #2: Buffer Overflow To Rewrite a Return

- A program that takes a username and prints a coupon
- Goal: to print a lot coupons!
  - you can only launch the target once, of course
  - Think about how to manipulate address to cause this to go into a loop, without adding a loop.
- exploit2.c: C program
- Half the credit will be given if you can print two coupons
- A finite number of more than 20 coupons will get most credit
- Full credit will be given if you can print an infinite number of coupons!

Task #3: Return to libc

- target3: A program that matches virus signatures in network packets
- Goal: to start a shell
  - Assumption: stack is NOT executable
- exploit3.c: c program

Task #4: Format String Attacks Format String Attacks

- target4.c: program with a format string vulnerability. It asks the user a set of questions, and prints some secret.
- Use the format string vulnerabilities to
  - Print the addresses of local variables
  - Print the values of local variables
  - Change the values of local variables
- Explore possible defenses.

Useful tools

- GDB
  - Start: gdb ./example1
  - Source: list <line number>
  - Assembly: disassemble<function name>
  - Step: step/stepi
  - Memory: x <address>
  - Variables/registers: print <var/reg>

Warming up

- Understand what is going on
  - The assembly code
  - The memory(stack)
  - The registers
  - The variables
  - …

Environment Setup

- The OS is running in a virtual machine
- Login
  - Connect to the host machine
    - ssh edgar.utdallas.edu
  - Connect to the VM
    - ssh attackme
  - Tools available
    - gcc, make, gdb, vi

Submission

- Deadline is 11:59pm CST Apr 16th
- Leave your solution files in ./exploits of your home directory
  - .c files should be compiled and ready to run without any arguments
- Submit your project report to eLearning

Logistics

- Exploits codes are short
  - Several ways to exploit
- Start early
- Backup files often (outside the virtual machine)
- Make your exploits stable
- Most of the points are for correct exploits – if you answer a question without correctly exploiting, no credit will be given

- When writing an exploit in C, you should use a function like execve to launch the target, not a function like system. Passing in null for the environmental variables so that it will be consistent and repeatable from run to run.
  - int execve (const char *filename, char *const argv [], char *const envp[]);

Today

- I'll post
  - These slides (course website)
  - The full project description (eLearning)
  - Latex for assignment if you want to use that for writeup (eLearning)
- I'll email you
  - username/passwords for the edgar/attackme machine
  - Please feel free to modify your password

Questions?