(Advanced) Computer Security!

Software Security

Buffer Overflow Vulnerabilities, Attacks, and Defenses

(part I)

Prof. Travis Peters

Montana State University

CS 476/594 – Computer Security

Spring 2021

https://www.traviswpeters.com/cs476

Today

Reminder!
Please update your Slack, GitHub, Zoom
(first/last name, professional photo/background)

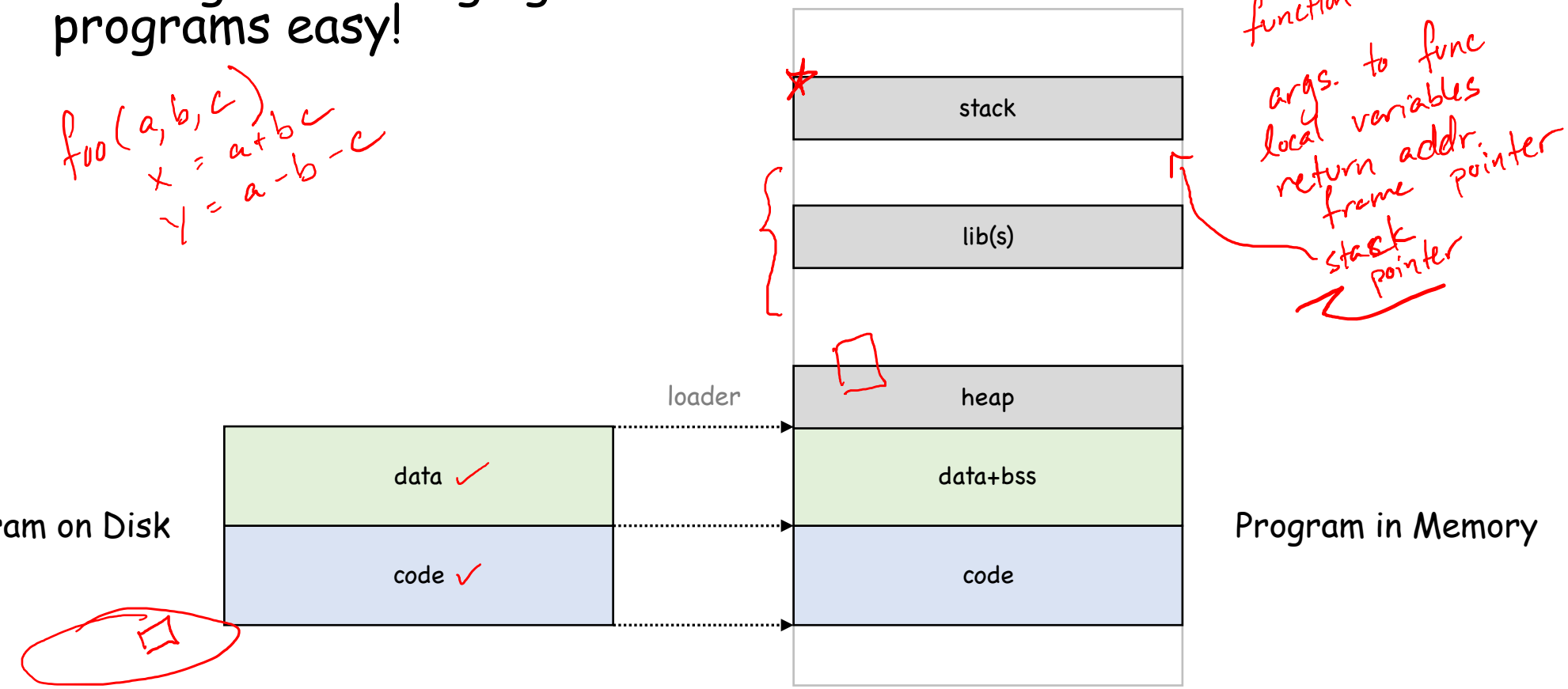
- Announcements
  - Lab 02 due today
  - Lab 03 released

- Learning Objectives
  - Review the layout of a program in memory & stack layout
  - Understand buffer overflows & vulnerable code
  - Challenges in exploiting buffer overflow vulnerabilities
  - Grasp major challenges and solutions of "shellcode"
  - Countermeasures (e.g., ASLR, StackGuard, non-executable stack)

Back to Memory Layout, Stacks, and Function Invocation

Loading a Program Into Memory

A standard layout for programs makes loading and managing programs easy!



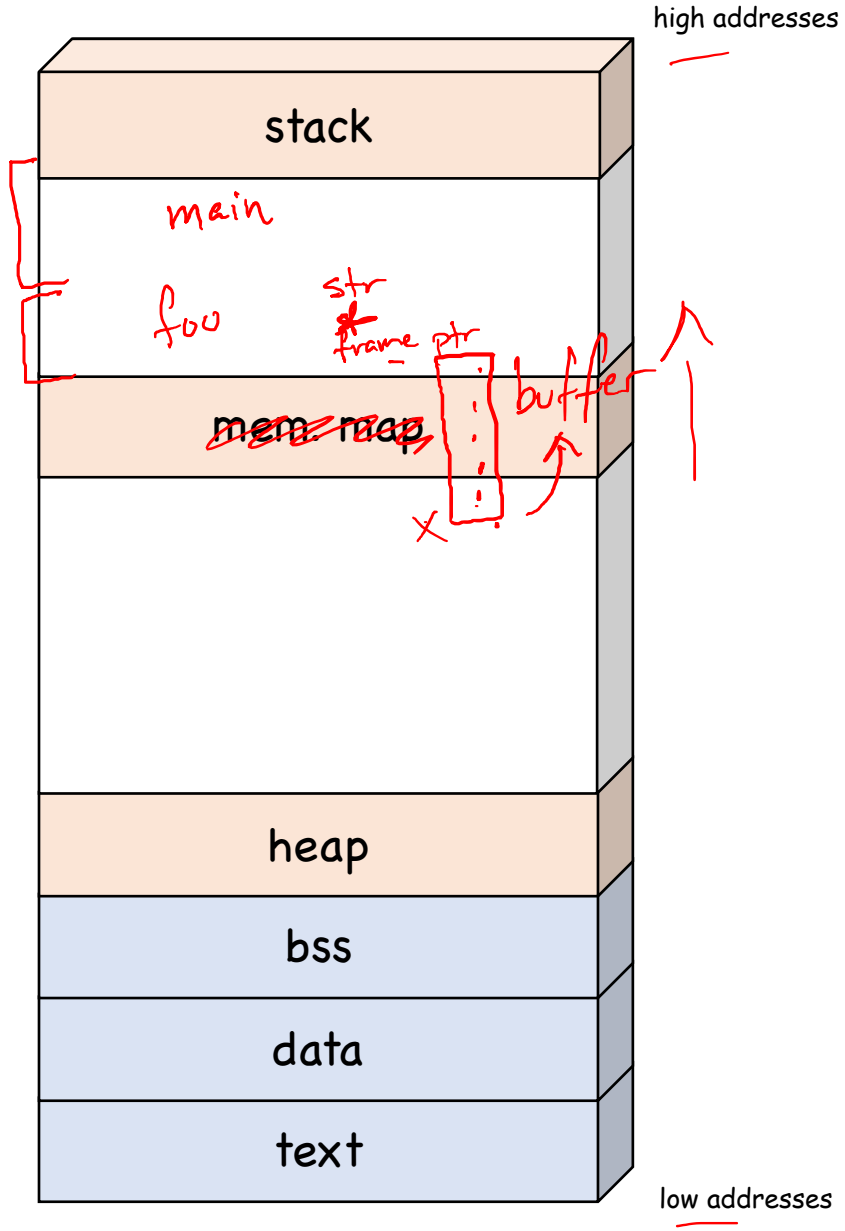
Read more about Linkers & Loaders: https://www.linuxjournal.com/article/6463

A Program in Memory

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

void foo(char *str)
{
    char buffer[10];
    strcpy(buffer, str);
}

int main(int argc, char *argv[])
{
    foo(argv[1]);
    printf("Returned Properly\n");
    return 0;
}
```



Example: Program Execution

A look at the stack and stack frames, and how they help to manage execution context

Live Demo in GDB

Stack Layout

Problem:

The compiler doesn't know where exactly the stack frame will "live" in memory...

How does the program know where to find function args. and local vars.?

-> They are stored on the stack, relative to the "frame pointer" aka "base pointer" (ebp/rbp)

32-bit addr. space

```
void main()
{
    foo(2,3);
    return 0;
}
```

```
void foo(int a, int b)
{
    int x, y;
    x = a + b;
    y = a - b;
}
```

```
push   $0x3          ; push b
push   $0x2          ; push a
call   .... <foo>    ; push RA
...
```

```
push   %ebp                ; save ebp
mov    %esp,%ebp           ; set ebp
...
mov    0x8(%ebp),%edx      ; a
mov    0xc(%ebp),%eax      ; b
add    %edx,%eax.          ; +
mov    %eax,-0x8(%ebp)     ; x=
mov    0x8(%ebp),%eax      ; etc.
sub    0xc(%ebp),%eax
mov    %eax,-0x4(%ebp)
...
leave  ; set esp = ebp
       ; pop ebp
ret    ; pop RA
```