PURDUE UNIVERSITY®

CS 240: Programming in C

Lecture 23: Graphics Wrap-up
Buffer Overflows
System Calls

Prof. Jeff Turkstra

GTK+

- GIMP Toolkit
- Multi-platform toolkit for creating graphical user interfaces (GUIs)
- Some examples…

  https://book.huihoo.com/gtk+-gnome-application-development/cha-gtk.html

Points about examples

- Some variables referred to opaque data (e.g. window) whose contents were not manipulated directly but had access functions to make changes
- No new data structures here. Making large changes to the program would not involve a lot of data structure additions
- If you did want to add data structures, you know enough to do so and you could do great things…
- The data structures are not complicated
  - You can do this RIGHT NOW.

Love it / Hate it

- I love to write software, but I wasn't always very good at it.
- I made my own projects and learned a great deal in doing so.
- If you have project ideas, but don't know how to start them, come talk to me
- If you want to find out about crazy ideas that I don't have time to work on, come talk to me
- If you have no idea what to work on, I don't know that I'll be much help…

Disbelief is your greatest enemy

- The only thing standing between your ability to write useful and professional-looking software is your disbelief that you can actually do so.
- Later courses will refine your knowledge and discipline, but you may never be more practiced and capable than at this point
- If you enjoy programming, do what you can to make sure you get some exercise so that when you graduate, you have something interesting to show the world (and your potential employers)

Security

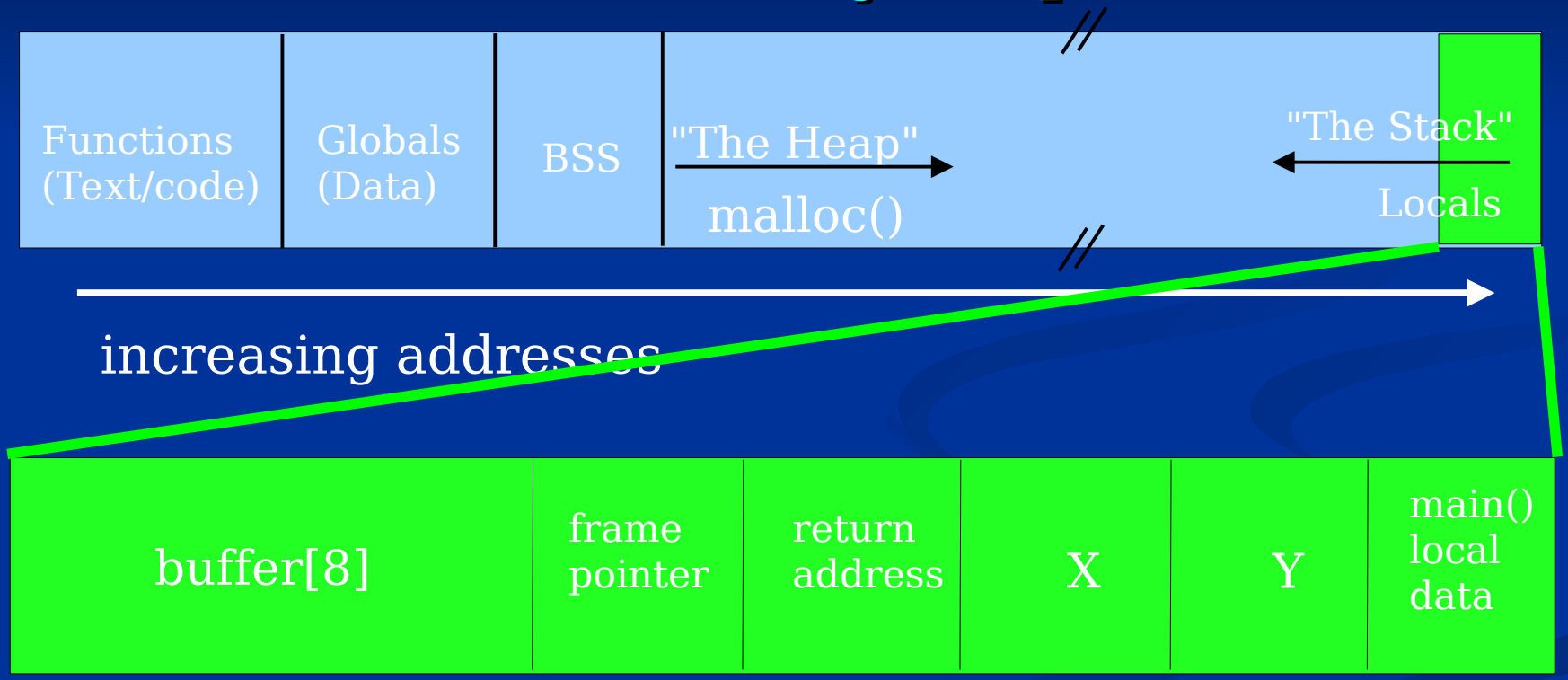
- The way that you write your software can make a difference in how secure it is
- At several points during the semester we told you to make sure to use strncpy() when copying data into a buffer array of limited size
- Not doing so could cause your program to be vulnerable to buffer overflows
- If your software is trusted by the system, others can use buffer overflows to break in

Consider the following function

```
void read_login(int x, int y) {
  char buffer[8];
  printf("Enter login: ");
  fscanf(stdin, "%s", buffer);
  printf("Hello, %s. Code %d.%d\n",
         buffer, x, y);
}

int main() {
  read_login(5, 6);
  return 0;
}
```

What does the stack look like? 32-bit

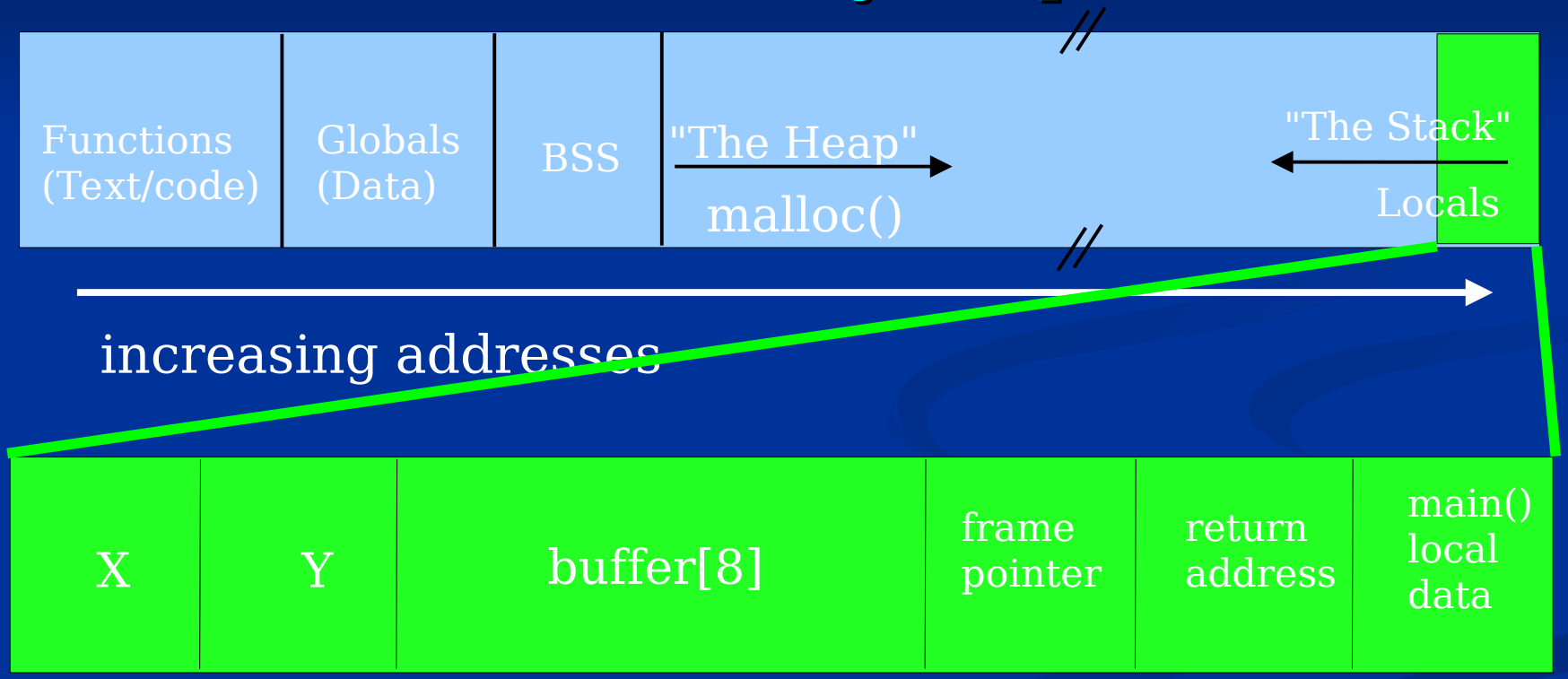
- Consider the memory map…



- If buffer is overrun, we can't get back to main()

What does the stack look like? 64-bit

- Consider the memory map…



- If buffer is overrun, we can't get back to main()

Caveats

- Padding for alignment and performance
- Arguments may be passed via registers
  - And stored on the stack after space has been allocated for local variables

walk

Using walk()...

```
#include "walk.c"

void hello(int value) {
  int local = 0xdecafbad;

  walk(&local+16, 112);
}

int main(int argc, char *argv[], char *envp[]) {
  int local = 0xbeefbeef;

  printf("Address of main is %p\n", main);
  printf("Address of local is %p\n", &local);
  hello(0xaaaaaaaa);

  return 0;
}
```

Address Space Layout Randomization

- ASLR changes the addresses of many things every time a program executes
  - Harder to exploit certain vulnerabilities

```
$ setarch x86_64 -R my_binary
```

```
$ gcc -fno-stack-protector -z
execstack -o doit doit.c
```

- When run, we will get a dump of 112 bytes of memory starting at the address of local
- We know basically what the output should look like, right?

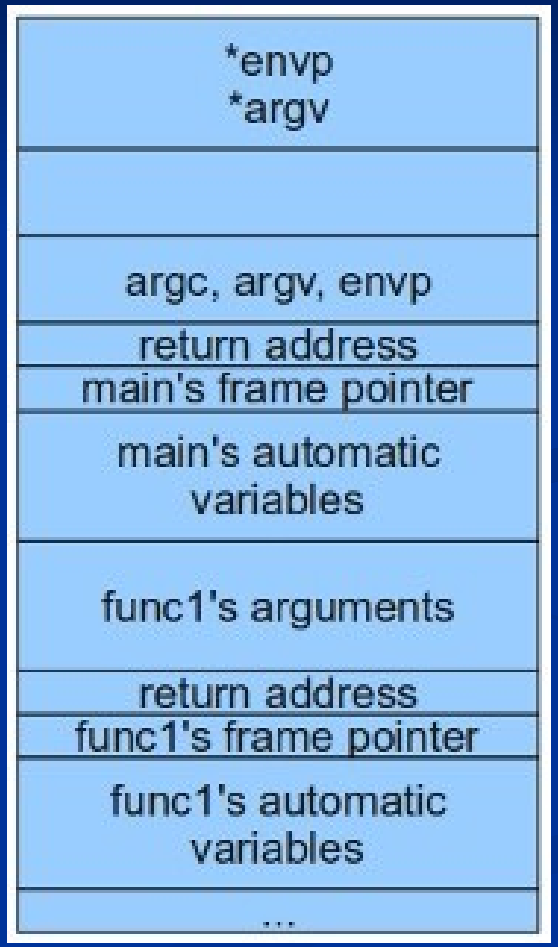
output...

```
Address of main is 0x401285
Address of local is 0x7ffd4c42d1dc
0x7ffd4c42d1e0: e0 12 40 00 00 00 00 00 ??@?????
0x7ffd4c42d1d8: 00 00 00 00 ef be ef be ????????
0x7ffd4c42d1d0: c0 d2 42 4c fd 7f 00 00 ??BL????
0x7ffd4c42d1c8: 60 10 40 00 01 00 00 00 `?@?????
0x7ffd4c42d1c0: c8 d2 42 4c fd 7f 00 00 ??BL????
0x7ffd4c42d1b8: d8 d2 42 4c fd 7f 00 00 ??BL????
0x7ffd4c42d1b0: c0 ae f7 ac a6 7f 00 00 ????????
0x7ffd4c42d1a8: d3 12 40 00 00 00 00 00 ??@?????
0x7ffd4c42d1a0: e0 d1 42 4c fd 7f 00 00 ??BL????
0x7ffd4c42d198: 50 71 f9 ac ad fb ca de Pq??????
0x7ffd4c42d190: 00 00 00 00 00 00 00 00 ????????
0x7ffd4c42d188: b6 d1 42 4c aa aa aa aa ??BL????
0x7ffd4c42d180: c2 00 00 00 00 00 00 00 ????????
0x7ffd4c42d178: 82 12 40 00 00 00 00 00 ??@?????
```

Purdue Trivia

- Purdue's official colors are Old Gold and Black
  - Determined in 1887, 13 years after the university started offering classes
  - Attributed to captain of first football team – John Breckenridge Burris
- According to the 1914 Debris yearbook, the decision to adopt Old Gold and Black was made in three minutes by a self-appointed committee of students and faculty
  - Met in University Hall prior to first football game
  - Burris admired Princeton – orange and black

Stack layout

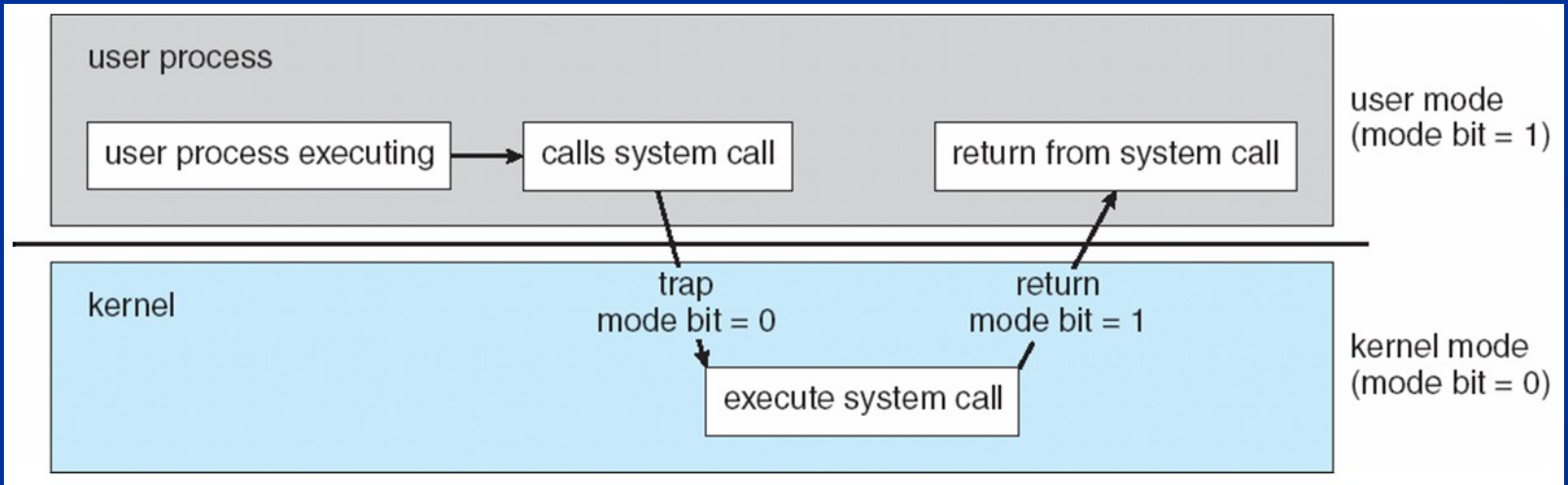


Kernel invocation

- What causes the switch to kernel mode?
  - System calls
  - Page faults
  - Signals
  - Hardware

System calls

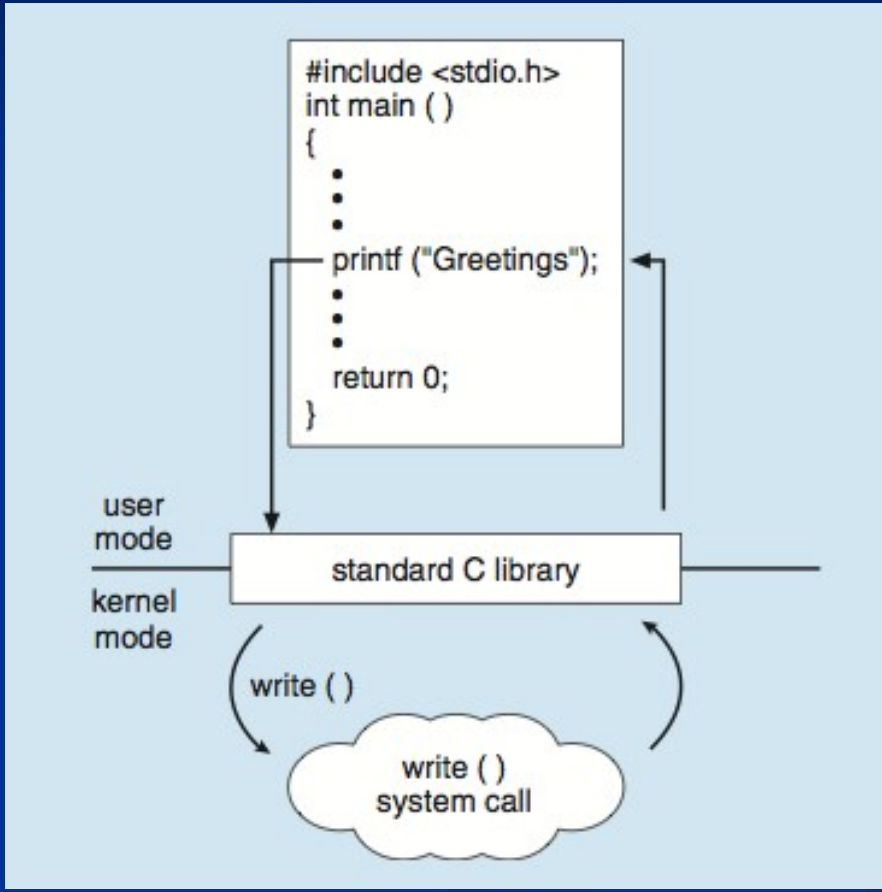
- System calls are the interface between processes and the OS kernel



System call types

- Process management
  - Create, terminate, execute, wait, etc
- File management
  - Create file, delete, open, close, read write, getattr, setattr, etc
- Device management
  - ioctl, read, write, etc
- Information management
  - getpid, alarm, sleep, etc
- Communication – between processes
  - pipe, shmget, mmap, etc

Standard C library



open() system call

```
int open(const char *pathname, int flags[, mode_t mode]);
```

- Flags includes:
  - Access mode (O_RDONLY, O_WRONLY, O_RDRWR) – required
  - File creation flags (O_CLOEXEC, O_CREAT, O_TRUNC, etc) – optional
  - File status flags (O_APPEND, O_SYNC, O_NONBLOCK, etc)
- Mode is your usual file creation mode

close() system call

```
int close(int fd);
```

- Decrements the reference count for the appropriate open file object
- Object is reclaimed if reference count == 0
- Returns -1 on error and sets errno
- Failing to close() fds results in a file descriptor leak
  - Arguably worse than a memory leak

...and

- `ssize_t read(int fd, void *buf, size_t count);`
- `ssize_t write(int fd, void *buf, size_t count);`

Using assembly language in C

- C gets compiled into a lower-level language called assembly language where each instruction represents one CPU instruction
- We generally do not have control over which assembly language instructions are chosen, but most compilers support a way of embedding specific instructions

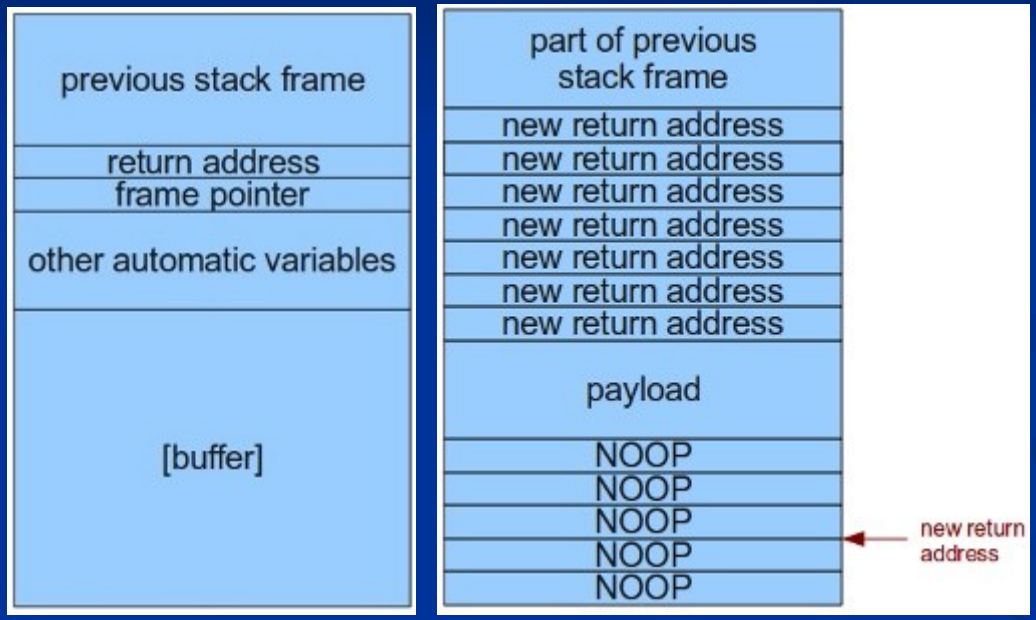
Example of asm in C

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

int main(int argc, char *argv[]) {
  unsigned char x;
  x = atoi(argv[1]);

  __asm__ __volatile__("add $1,%0" : "=r"(x) : "0"(x));

  printf("%d\n", x);
  return x;
}
```

More examples

- Shell
- chmod

A small server


previous stack frame
return address
frame pointer
other automatic variables
[buffer]
part of previous stack frame
new return address
new return address
new return address
new return address
new return address
new return address
new return address
payload
NOOP
NOOP
NOOP
NOOP
NOOP
new return address

Security

- You have a very, very small taste of what kind of problems can arise in terms of program security
- We've only touched the "tip of the iceberg" in terms of buffer overflows
- If you want to know more, check out:
  - https://turkeyland.net/projects/overflow/
  - ...or find "Smashing the Stack for Fun and Profit" using a search engine
- There are many, many other types of vulnerabilities
- If you enjoy this stuff, take a security course!

Boiler Up!