CS 240: Programming in C

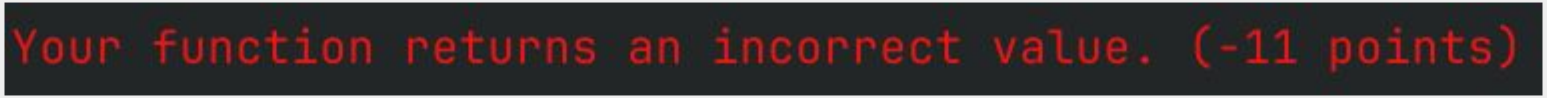
Lecture 24: Terminal Effects, Buffer Overflows, Core Files

Announcements

- Midterm 2 regrades close tomorrow
- Homework 12 is out!
    - This is the last required homework

Terminal effects

- How do we get colored text output?



```
Your function returns an incorrect value. (-11 points)
```

- Colors (and other effects) are implemented with special byte sequences called ANSI escape sequences

ANSI escape sequences

- A sequence of bytes that are interpreted by the terminal emulator to do various things
- Most of them have the form:

```
<ESC> [ <parameters> <end>
```

  - <ESC> is the ASCII code for ESC: 0x1B
  - <parameters>: a number of control bytes
  - <end>: a single byte indicating the command

PURDUE UNIVERSITY®

Text formatting

- The <end> byte for text formatting is the letter ‘m’.
- The <parameters> control the color and styling
- For example:

```
<ESC>[31m
```

  - Sets the text color to be red

- Empty parameters reset all formatting

```
<ESC>[m
```

Text formatting

- You can also combine parameters

```
<ESC>[41;93m
```

  - Bright yellow text on a red background

```
<ESC>[4;3;36m
```

  - Underlined italic cyan text

- See Wikipedia for a list of ANSI escape codes

ANSI formatting in C

- Use “\x1b” to send the ESC code
- Everything else is in ASCII

```
printf(“\x1b[31mSome text\x1b[m\n”);
```

- Don’t forget about string concatenation

```
#define RED “\x1b[31m”
#define RESET “\x1b[m”

printf(RED “Some text” RESET “\n”);
```

PURDUE UNIVERSITY

Other ANSI codes

- You can do more than just format text
    - Clear the screen / current line
    - Move the cursor
    - Set the cursor position
    - Read the current cursor position
- Many text-based programs use these
    - Including Vim
- Here is a nice tutorial on how to use them in C

Security

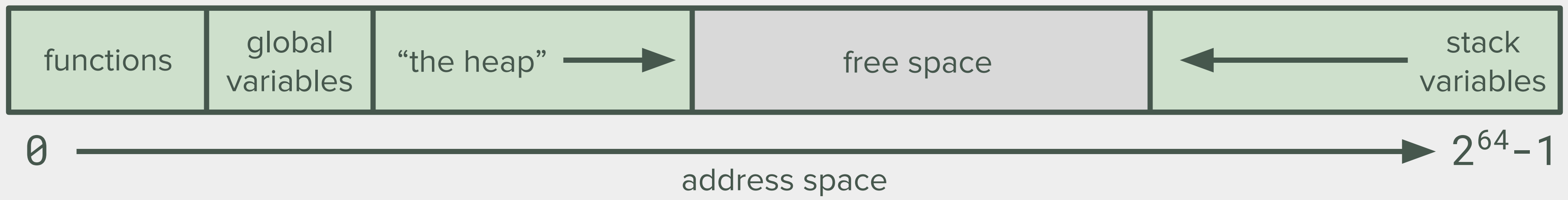
- The way you write your software can make a difference in how secure it is
- Make sure you use str**n**cpy() when copying data into a buffer array of limited size
- Not doing so could cause your program to be vulnerable to buffer overflows
- If your software is trusted by the system, others can use buffer overflows to break in

Consider the following function

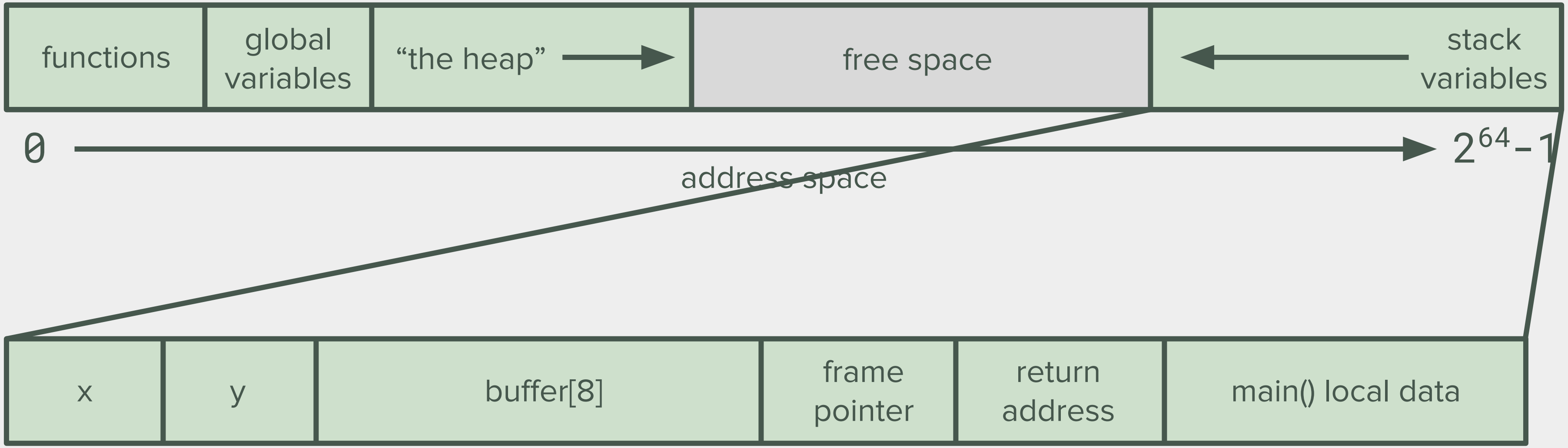
```
void read_login(int x, int y) {
  char buffer[8];
  printf(“Enter login: “);
  fscanf(stdin, “%s”, buffer);
  printf(“Hello, %s. Code %d.%d\n”,
    buffer, x, y);
}

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

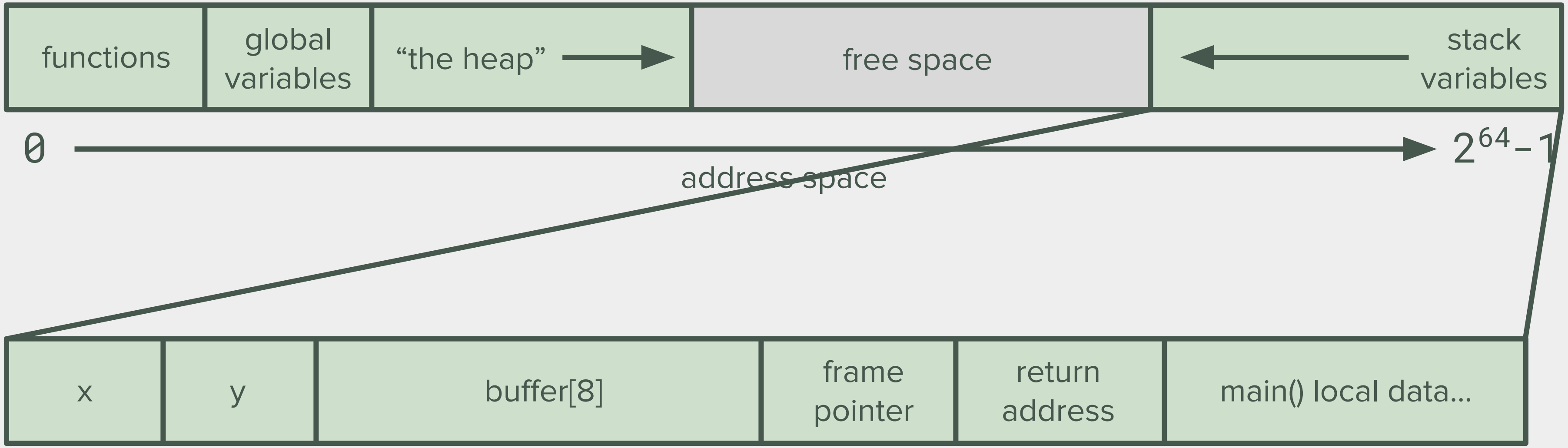
What does the stack look like?



What does the stack look like?



What does the stack look like?



- If the buffer overflows, it overwrites the return address
- We can’t get back to main()

Stack example

```
void walk(void *a, int bytes) {
  char *addr = a;
  /* Make sure address is 8-byte aligned */
  while (((long)addr) % 8) addr++;

  for (int i = 0; i < bytes / 8; i++) {
    printf(“%p: “, addr);
    /* Print hex values */
    for (int j = 0; j < 8; j++)
      printf(“%02hhx “, addr[j]);
    /* Print ASCII values */
    for (int j = 0; j < 8; j++)
      printf(“%c”, isprint(addr[j]) ? addr[j] : ‘?’);

    addr -= 8;
    printf(“\n”);
  }
  return;
}
```

Stack example

```
#include <ctype.h>
#include “walk.c”

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

  walk(&local+16, 112);
}

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

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

  return 0;
}
```

Stack example

```
Address of main is 0x5555555552a2
Address of local is 0x7fffffffdb5c
0x7fffffffdb70: b0 db ff ff ff 7f 00 00 ........
0x7fffffffdb68: 08 fe da f7 ff 7f 00 00 ........
0x7fffffffdb60: 00 dc ff ff ff 7f 00 00 ........
0x7fffffffdb58: 88 dc ff ff ef be ef be ........
0x7fffffffdb50: 00 00 00 00 00 00 00 00 ........
0x7fffffffdb48: e0 53 fe f7 01 00 00 00 .S......
0x7fffffffdb40: 88 dc ff ff ff 7f 00 00 ........
0x7fffffffdb38: fb 52 55 55 55 55 00 00 .RUUUU..
0x7fffffffdb30: 60 db ff ff ff 7f 00 00 `.......
0x7fffffffdb28: e0 e2 ff f7 ad fb ca de ........
0x7fffffffdb20: 88 dc ff ff ff 7f 00 00 ........
0x7fffffffdb18: 00 00 00 00 aa aa aa aa ........
0x7fffffffdb10: 00 00 00 00 00 00 00 00 ........
0x7fffffffdb08: 9f 52 55 55 55 55 00 00 .RUUUU..
```

Stack example

```
Address of main is 0x5555555552a2
Address of local is 0x7fffffffdb5c
0x7fffffffdb70: b0 db ff ff ff 7f 00 00 ........
0x7fffffffdb68: 08 fe da f7 ff 7f 00 00 ........
0x7fffffffdb60: 00 dc ff ff ff 7f 00 00 ........
0x7fffffffdb58: 88 dc ff ff ef be ef be ........
0x7fffffffdb50: 00 00 00 00 00 00 00 00 ........
0x7fffffffdb48: e0 53 fe f7 01 00 00 00 .S......
0x7fffffffdb40: 88 dc ff ff ff 7f 00 00 ........
0x7fffffffdb38: fb 52 55 55 55 55 00 00 .RUUUU..
0x7fffffffdb30: 60 db ff ff ff 7f 00 00 `.......
0x7fffffffdb28: e0 e2 ff f7 ad fb ca de ........
0x7fffffffdb20: 88 dc ff ff ff 7f 00 00 ........
0x7fffffffdb18: 00 00 00 00 aa aa aa aa ........
0x7fffffffdb10: 00 00 00 00 00 00 00 00 ........
0x7fffffffdb08: 9f 52 55 55 55 55 00 00 .RUUUU..
```

Return address

Frame pointer

Compile without stack protection

- GCC adds extra code to protect against buffer overflows in general
- Runtime looks for “canaries” before and after return addresses and checks to make sure they haven’t been overwritten
- Let’s disable it

```
gcc -fno-stack-protector -z execstack -o stack1 stack1.c
```

Address Space Layout Randomization

- Modern operating systems randomize the address space each time a program is run
- It makes it hard for an attacker to guess were certain things are, such as the exact address of the stack
- Let’s also disable it

```
echo 0 > /proc/sys/kernel/randomize_va_space
```

- Alternatively:

```
setarch x86_64 -R ./stack1
```

PURDUE UNIVERSITY

Buffer overflow

```
#include <ctype.h>
#include “walk.c”

void hello(int value) {
  int local = 0xdecafbad;
  char buf[16];

  walk(&local+4, 64);

  printf(“Please enter a string: “);
  scanf(“%s”, buf);
  printf(“You entered %s!\n”, buf);

  walk(&local+4, 64);
}
```

Buffer overflow

```
Address of main is 0x555555555311
Address of local is 0x7fffffffdb5c
0x7fffffffdb40: 88 dc ff ff ff 7f 00 00 ........
0x7fffffffdb38: 6a 53 55 55 55 55 00 00 jSUUUU..
0x7fffffffdb30: 60 db ff ff ff 7f 00 00 `.......
0x7fffffffdb28: e0 e2 ff f7 ad fb ca de ........
0x7fffffffdb20: 88 dc ff ff ff 7f 00 00 ........
0x7fffffffdb18: 00 00 00 00 00 00 00 00 ........
0x7fffffffdb10: 00 00 00 00 00 00 00 00 ........
0x7fffffffdb08: 00 00 00 00 aa aa aa aa ........
Please enter a string: Hello
You entered Hello!
0x7fffffffdb40: 88 dc ff ff ff 7f 00 00 ........
0x7fffffffdb38: 6a 53 55 55 55 55 00 00 jSUUUU..
0x7fffffffdb30: 60 db ff ff ff 7f 00 00 `.......
0x7fffffffdb28: e0 e2 ff f7 ad fb ca de ........
0x7fffffffdb20: 88 dc ff ff ff 7f 00 00 ........
0x7fffffffdb18: 00 00 00 00 00 00 00 00 ........
0x7fffffffdb10: 48 65 6c 6c 6f 00 00 00 Hello...
0x7fffffffdb08: 00 00 00 00 aa aa aa aa ........
```

Buffer overflow

```
Address of main is 0x555555555311
Address of local is 0x7fffffffdb5c
0x7fffffffdb40: 88 dc ff ff ff 7f 00 00 ........
0x7fffffffdb38: 6a 53 55 55 55 55 00 00 jSUUUU..
0x7fffffffdb30: 60 db ff ff ff 7f 00 00 `.......
0x7fffffffdb28: e0 e2 ff f7 ad fb ca de ........
0x7fffffffdb20: 88 dc ff ff ff 7f 00 00 ........
0x7fffffffdb18: 00 00 00 00 00 00 00 00 ........
0x7fffffffdb10: 00 00 00 00 00 00 00 00 ........
0x7fffffffdb08: 00 00 00 00 aa aa aa aa ........
Please enter a string: hellohereisalotoftextthatshouldcauseanoverflow
You entered hellohereisalotoftextthatshouldcauseanoverflow!
0x7fffffffdb40: 88 dc ff ff ff 7f 00 00 ........
0x7fffffffdb38: 65 72 66 6c 6f 77 00 00 erflow..
0x7fffffffdb30: 61 75 73 65 61 6e 6f 76 auseanov
0x7fffffffdb28: 74 73 68 6f 75 6c 64 63 tshouldc
0x7fffffffdb20: 66 74 65 78 74 74 68 61 ftexttha
0x7fffffffdb18: 65 69 73 61 6c 6f 74 6f eisaloto
0x7fffffffdb10: 68 65 6c 6c 6f 68 65 72 helloher
0x7fffffffdb08: 00 00 00 00 aa aa aa aa ........
Segmentation fault (core dumped)
```

Core dump files

- When your program has an unrecoverable error, the operating system saves the heap/stack memory at the exact time of the failure into a file named “core”

```
Segmentation fault (core dumped)
```

- You can use the core file with the debugger
- May need to enable it first

```
$ ulimit -c unlimited
```

PURDUE UNIVERSITY

Core dump files

- To debug a core file, make sure you compile with debug symbols enabled
- Then run gdb

```
$ gdb path/to/binary path/to/core
```

- Inside gdb you can bt to get a backtrace
- Read the man page for more info

```
$ man core
```

PURDUE UNIVERSITY

Takehome Quiz #5

- Given the function and stack dump:

```
void hello(int arg) {
  int local = 0x1ceb00da
}

(1) 0x7ffc4a8bee80: c0 ef 4b 5c 10 7f 00 00 ..K\....
(2) 0x7ffc4a8bee78: d3 12 40 00 00 00 00 00 ..@.....
(3) 0x7ffc4a8bee70: b0 ee 8b 4a fc 7f 00 00 ...J....
(4) 0x7ffc4a8bee68: 50 b1 4d 5c da 00 eb 1c P.M\....
(5) 0x7ffc4a8bee60: 00 00 00 00 00 00 00 00 ........
(6) 0x7ffc4a8bee58: 86 ee 8b 4a cc cc cc cc ...J....
```

1. Which line number contains the local variable?
2. Which line number contains the frame pointer?
3. Which line number contains the argument, assuming the function is passed the value 0xcccccccc?