CMSC 426
Principles of Computer Security

Overflow Attack Basics

Last Class We Covered

- Security Standards
  - Standards Bodies
- Security Principles
- Security Strategy

Any Questions from Last Time?

Today's Topics

- Buffer overflow basics
- How the stack works
  - Overflowing the stack buffer
  - Example in action
- Vulnerable code
  - Finding vulnerable code
  - Avoiding vulnerable code
- Exploiting stack overflows
  - Shellcode

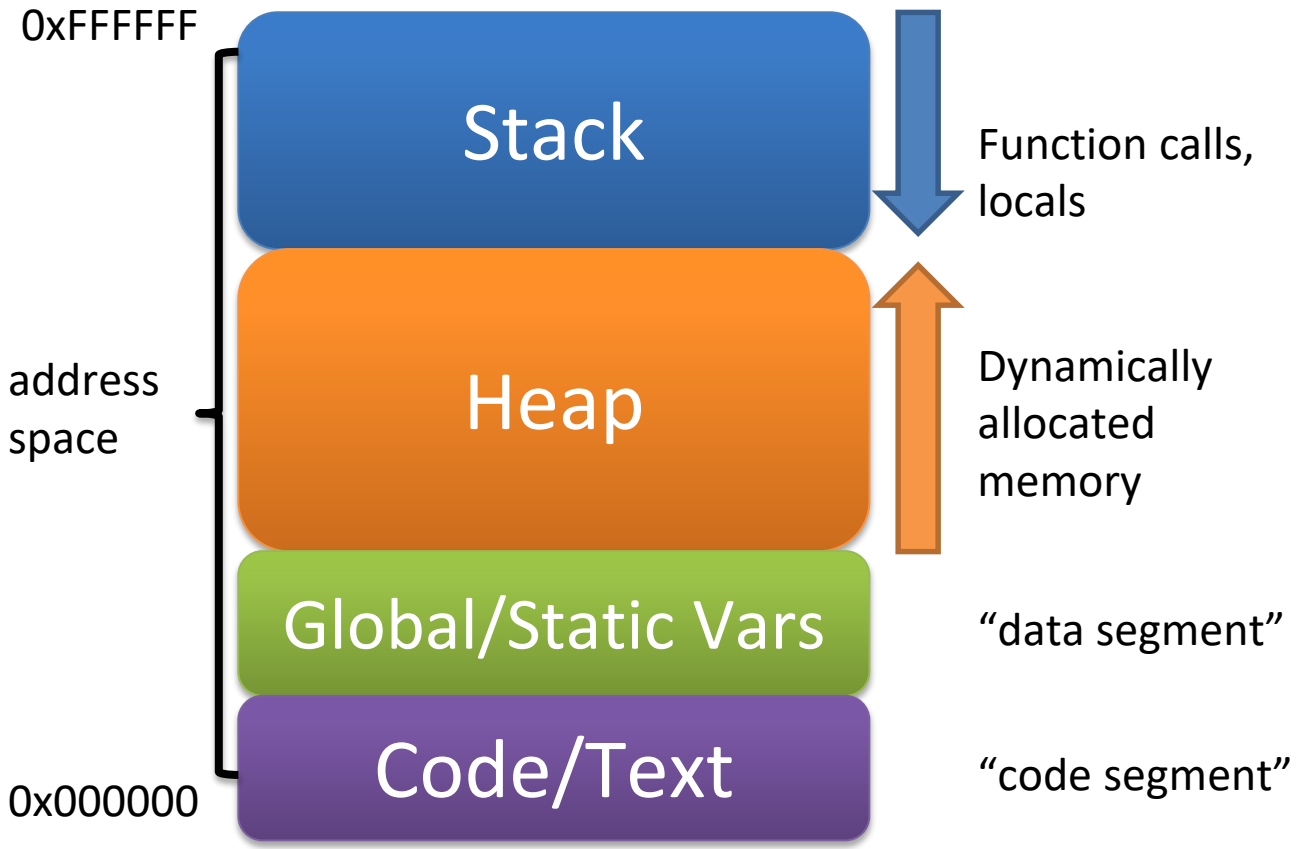
Buffer Overflows

- Programs constantly write data to areas of memory (buffers)
  - Higher level languages (Java, Python, etc.) do a lot of user hand-holding and won't allow unsafe use of the language
  - Lower level languages (C, etc.) do a minimal amount of checking, and assume that the programmer knows what they're doing

- When a buffer has data written to it that exceeds the size of the buffer, a buffer overflow occurs
  - The excess data continues to write, overflowing into nearby variables and other areas of memory

Stacks, Heaps, and More

- Processes get their own address space when run
- Address space is divided into smaller pieces, each with a specific purpose
- Stack grows “down” to lower addresses



Stack Allocation

- Memory allocated by the program as it runs
  - Local variables
  - Function calls
    - Parameters passed
    - Function-local variables
    - Return addresses

- (Somewhat) fixed at compile time

Heap Allocation

- Dynamically allocated memory
  - Memory explicitly allocated by the user
    - Using **`malloc()`**, **`calloc()`**, **`new`**, etc.
  - Creation and deletion (freeing) is controlled by the user
- Not determined at compile time

Stack Overflow Example

Stack Overflow Example Code

- Relevant code snippet:

```
int main()
{
  char first[5];
  char name[15];

  printf("Please enter a name: ");
  gets(name);
  printf("\nfirst: %s\n", first);
  printf("You entered the name %s\n", name);
  return 0;
}
```

Stack Overflow Example Run

```
linuxserver1[7]% ./a.out

Please enter a name: Gibson

first:
You entered the name Gibson

linuxserver1[8]% ./a.out

Please enter a name: Dr. Katherine L. Gibson

first: . Gibson
You entered the name Dr. Katherine L. Gibson
```

Stack Overflow Example Compile

```
linuxserver1[13]% gcc overflow.c
overflow.c: In function ‘main’:
overflow.c:16:3: warning: implicit declaration of function ‘gets’;
did you mean ‘fgets’? [-Wimplicit-function-declaration]
   gets(name);
   ^~~~
   fgets
/tmp/ccncipQo.o: In function `main':
overflow.c:(.text+0x3e): warning: the `gets' function is dangerous
and should not be used.
```

Overflowing the Stack Buffer

- Requires the use of a lower-level language (like C) that will allow the use of unsafe functions and methods
  - Like **`strcpy()`** or **`gets()`**

- End goal is to use the overflow to overwrite important things
  - Return addresses
  - Function parameters
  - “Normal” memory with code supplied by the attacker

Another Stack Overflow Example Run

```
linuxserver1[15]% ./a.out

Please enter a name: Dr. Katherine Gibson is teaching this course with a very long
title - CMSC 426: Principles of Computer Security

first: ibson is teaching this course with a very long title - CMSC 426: Principles of
Computer Security
You entered the name Dr. Katherine Gibson is teaching this course with a very long
title - CMSC 426: Principles of Computer Security
Segmentation fault (core dumped)
```

Segmentation Faults

- Happens when memory is written to that should not be
- Or when memory that is accessed should not be

- Not 100% consistent – sometimes C/C++ will let you "get away" with accessing or writing to memory that doesn't "belong" to you/the program
  - The more you mess up, the more likely it will be caught

Vulnerable Code

Finding Vulnerable Code

- Easiest way: inspect source code of programs
- Trace the execution of programs as they process oversized input
- Brute forcing or “fuzzing” a program with large inputs to see if errors arise

Information taken from Computer Security (Stallings & Brown)

Avoiding Vulnerable Code

- Ensure that buffers only take in the amount of data they can actually hold
- Enforce size limits on inputs from users and files
- Use a higher-level language when needed
- Don’t use bad, outdated functions!

Unsafe Functions and Alternatives

- Set a maximum size/number of characters to handle at once

Unsafe	Safe	Description
`gets()`	`fgets()`	Read characters from a stream
`strcpy()`	`strncpy()`	Copy from one string to another
`strcat()`	`strncat()`	Concatenate one string to another
`sprintf()`	`snprintf()`	Write data to a string

Safe Programming and Safe Libraries

- Early language designers hoped/assumed that programmers would exercise care and foresight when writing code
  - C allows for higher performance and space efficiency than Java
  - But, programmers are ***not*** generally careful or thoughtful

- Standard libraries allow for unsafe actions (like previous slide)
  - Create alternatives to unsafe functions/entire libraries
    - Requires rewriting/updating the source code
  - Create safe versions of type libraries (like strings)

Making a "Safe" C (or any language)

- Many have tried, many have failed
- There are literally dozens of "safe" C attempts out there
- Protip: don't pick this for a dissertation topic

Exploiting Stack Overflows

Overwriting Return Addresses

- Want to control where the program “returns” to after a function is completed
- If we can force it to return to somewhere in memory where malicious code, then it will execute that code instead
- Accomplish this by overwriting the actual return address with one of our own making

Shellcode

- The malicious code that we want to be run

- In our example, will be causing a shell to open
  - Ideally, with root privileges
  - Will let us be a “super user”
    - Remove and edit files, view all files and directories, make changes to permissions of other files

NOP Sleds

- Difficult to jump exactly to the start of the shellcode

- "NOP" means "no operation"
- When the program sees a NOP, it moves on to the next instruction

- Create a sequence of NOPs
  - Jumping anywhere inside it will allow you to "sled" to your actual shellcode

Daily Security Tidbit

- June 2007, Lifelock used CEO Todd Davis's social security number prominently in many of its advertisements
  - Meant to show how good the company was at preventing identity theft
- His identity was stolen 13 times within the year
  - Most of it was small charges ($100 - $500), probably done by people showing off that it could be done
- Lifelock was fined by the FTC for deceptive advertising

Information taken from https://www.wired.com/2010/05/lifelock-identity-theft/

Announcements

- Sign up for Piazza if you haven't already, as assignments will be starting soon