CYBR 437: Secure Coding

C Review

Instructor: Abinash Borah

Outline

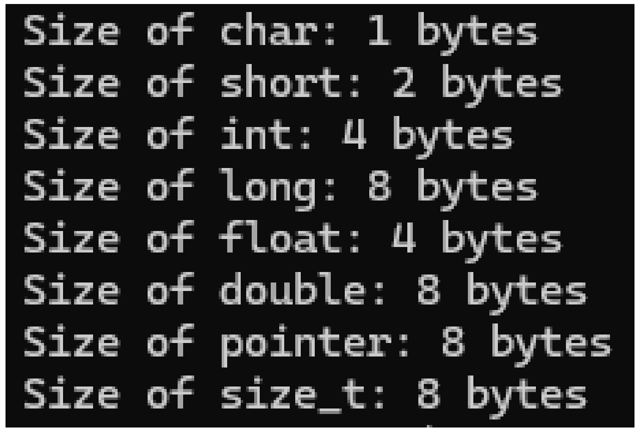
- C Data Types and Sizes
- Pointers and Pointer Arithmetic

C Data Types and Sizes

- There are many data types in C
- Type sizes are dependent on the machine and its processor
- The types we currently care about are:
    - char
    - short
    - int
    - long
    - float
    - double
    - void *
    - size_t

Obtaining Type Sizes with sizeof

```
#include <stdio.h>
int main()
{
    printf("Size of char: %zu bytes\n", sizeof(char));
    printf("Size of short: %zu bytes\n", sizeof(short));
    printf("Size of int: %zu bytes\n", sizeof(int));
    printf("Size of long: %zu bytes\n", sizeof(long));
    printf("Size of float: %zu bytes\n", sizeof(float));
    printf("Size of double: %zu bytes\n", sizeof(double));
    printf("Size of pointer: %zu bytes\n", sizeof(void *));
    printf("Size of size_t: %zu bytes\n", sizeof(size_t));
    return 0;
}
```



```
Size of char: 1 bytes
Size of short: 2 bytes
Size of int: 4 bytes
Size of long: 8 bytes
Size of float: 4 bytes
Size of double: 8 bytes
Size of pointer: 8 bytes
Size of size_t: 8 bytes
```

What is sizeof?

- sizeof is an operator - not a function
- The sizeof operator yields the size (in bytes) of its operand
- The result is an unsigned integer of type size_t
- The operand for sizeof may be an expression or the parenthesized name of a data type
- If the type of the operand is a variable-length array type, the operand is evaluated; otherwise, the operand is not evaluated, and the result is an integer constant
- Parentheses are unnecessary
    - Parentheses are needed because sizeof has high precedence
    - sizeof a + b is different than sizeof (a+b)

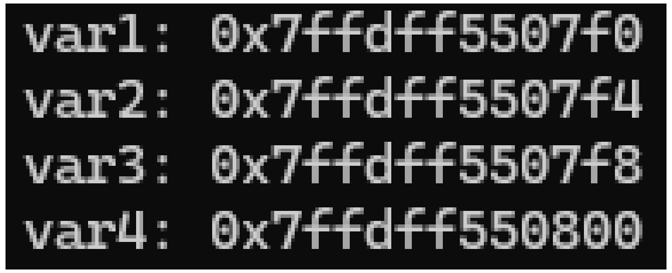
Pointers and Pointer Arithmetic

Why do we care about pointers?

- Everything we do in C relates to a memory location
- Being able to manipulate values at the memory locations allows us to find programming flaws
- We want to understand the power of memory and pointers to secure our code

Example

```
#include <stdio.h>
int main()
{
    int var1 = 20;
    int var2 = 150;
    int *var3 = &var1;
    int *var4 = &var1;
    printf("var1: %p\n", &var1);
    printf("var2: %p\n", &var2);
    printf("var3: %p\n", &var3);
    printf("var4: %p\n", &var4);
    return 0;
}
```



```
var1: 0x7ffdff5507f0
var2: 0x7ffdff5507f4
var3: 0x7ffdff5507f8
var4: 0x7ffdff550800
```