

Secure Programming

Integer Overflows

1

Ahmet Burak Can
Hacettepe University

2

Learning objectives

- Know the internal representation of integers
- Be able to determine when an integer overflow can occur
- Understand the consequences of integer overflows

3

Integers

- Fixed number of bytes
- Signed and unsigned
- Types:
 - Char
 - "char" is different from "unsigned char" and "signed char"
 - Short
 - Int
 - Long
- Extended types
 - `uint_least16_t` (integer of at least 16 bits)
 - etc...

4

Internal Representation

- Signed Short:
 - -1 is FFFF
 - 32767 is 7FFF
 - -32768 is 8000
- If $a = -32768$, what is $-a$?
 - $-(-32768)$ is $-32768!$
- if $a = 32767$, what is $a+1$?
 - $32767 + 1$ is -32768

5

Internal Representation, Unsigned

- Unsigned Short:
 - 65535 is FFFF
 - 0 is 0000
- If $a = 0$, what is $a-1$?
 - $0-1$ is 65535
- if $a = 65535$, what is $a+1$?
 - $65535 + 1 = 0!$

6

Integer Overflow Example*

```
#include <stdio.h>

int main(void){
    int i;
    short s;
    char c;

    i = 0xdeadbeef;
    s = 1;
    c = 1;

    printf("i = 0x%x (%d bits)\n", i, sizeof(i) * 8);
    printf("s = 0x%x (%d bits)\n", s, sizeof(s) * 8);
    printf("c = 0x%x (%d bits)\n", c, sizeof(c) * 8);

    return 0;
}
```

The output:

```
i = 0xdeadbeef (32 bits)
s = 0xffffbeef (16 bits)
c = 0xffffffff (8 bits)
```

* <https://www.securecoding.com/blog/integer-overflow-attack-and-prevention/>

7

Silent Signed to Unsigned Conversions

- No warning, or compiler warning was ignored
- What happens when you pass a negative number to a function expecting an unsigned integer?
- `void *malloc(size_t size);`

8

Malloc(0) Attack Scenario

- Overflow in the size calculations can be engineered to allocate no memory
- Malloc(0) is legal, but returned value OS-dependent
 - Sun: returns pointer to the "arena"
 - Pointer to buffer of size 0, or a minimum size
- Program happily trashes the arena, or heap
 - "Fandango on core"