Integer Overflows

Software Security

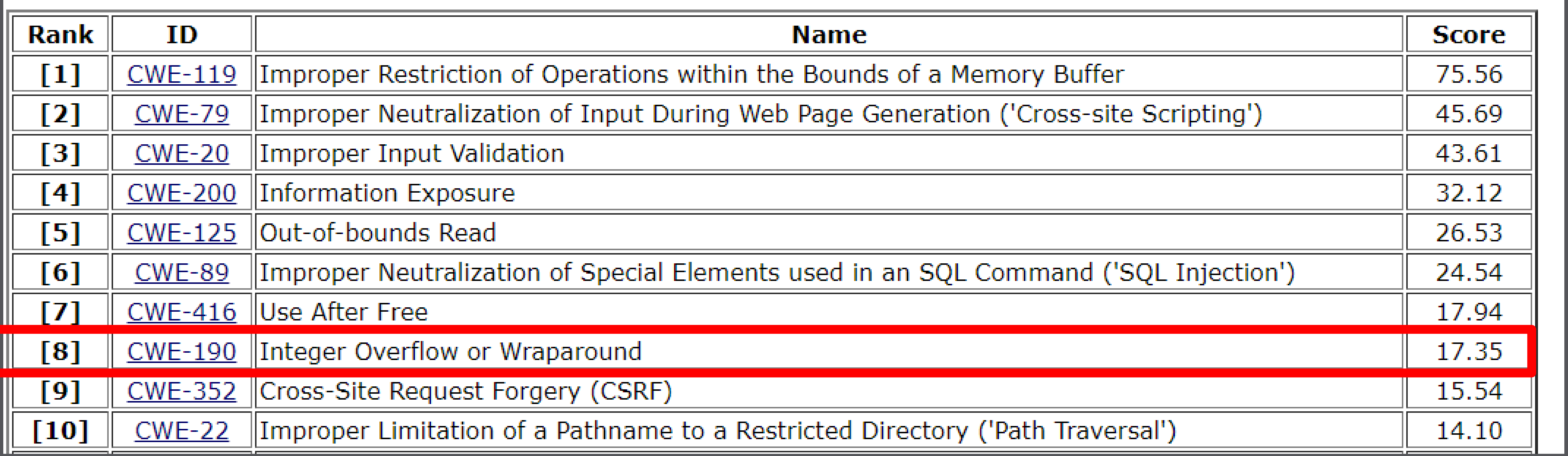
Goals

- Identify integer overflows and understand the associated risks
- Triage and remediate integer overflows in software

Integer Overflow

- an integer overflow occurs when an arithmetic operation attempts to create a numeric value that is outside of the range that can be represented with a given number of bits – either larger than the maximum or lower than the minimum representable value

Integer Overflow in CWE Top 25



Rank	ID	Name	Score
[1]	CWE-119	Improper Restriction of Operations within the Bounds of a Memory Buffer	75.56
[2]	CWE-79	Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')	45.69
[3]	CWE-20	Improper Input Validation	43.61
[4]	CWE-200	Information Exposure	32.12
[5]	CWE-125	Out-of-bounds Read	26.53
[6]	CWE-89	Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')	24.54
[7]	CWE-416	Use After Free	17.94
[8]	CWE-190	Integer Overflow or Wraparound	17.35
[9]	CWE-352	Cross-Site Request Forgery (CSRF)	15.54
[10]	CWE-22	Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')	14.10

Explained

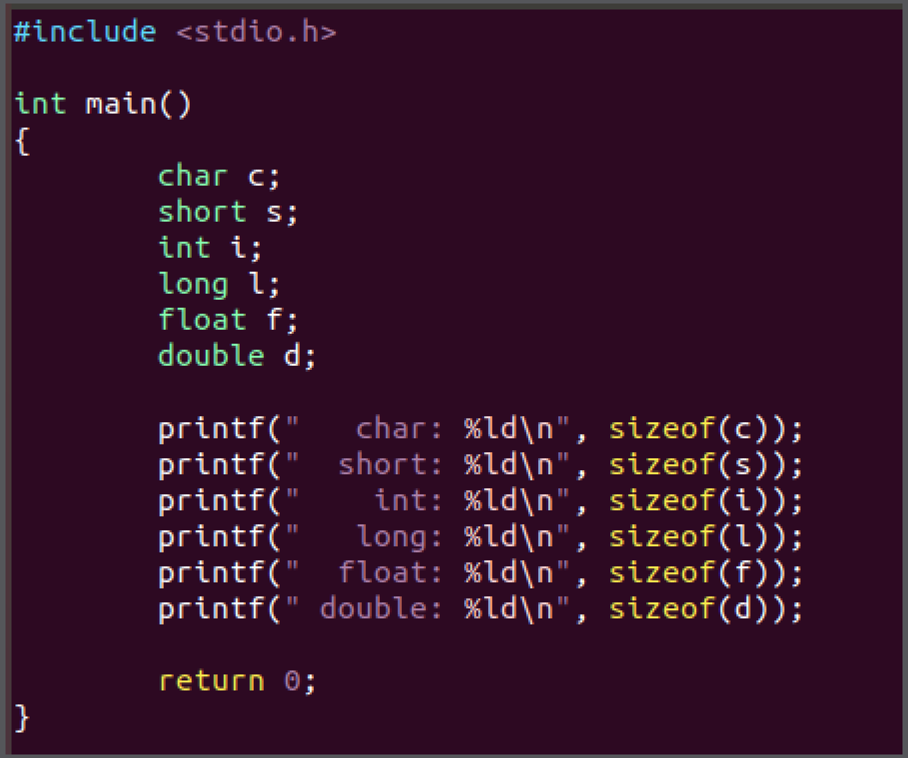
First – Type Sizes

- C has a few basic data types

Type	Notes
char	Single byte; holds a single character of the local set
int	integer of natural size on the host
float	single-precision floating point
double	double-precision floating point
short	at least 16 bits
long	at least 32 bits

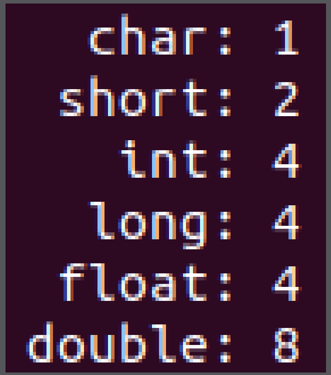
- long >= 32 bits > int > short

Type Sizes in an Environment

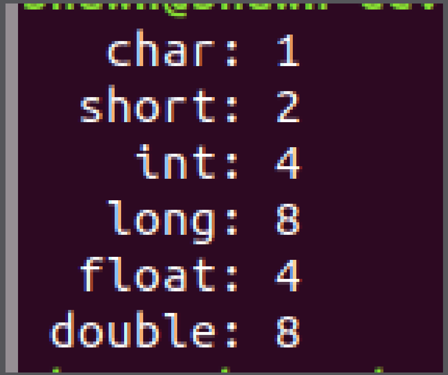


```c
#include <stdio.h>

int main()
{
        char c;
        short s;
        int i;
        long l;
        float f;
        double d;

        printf("   char: %ld\n", sizeof(c));
        printf("  short: %ld\n", sizeof(s));
        printf("    int: %ld\n", sizeof(i));
        printf("   long: %ld\n", sizeof(l));
        printf("  float: %ld\n", sizeof(f));
        printf(" double: %ld\n", sizeof(d));

        return 0;
}
```

32 bit vs 64 bit



```
  char: 1
 short: 2
   int: 4
  long: 4
 float: 4
double: 8
```



```
  char: 1
 short: 2
   int: 4
  long: 8
 float: 4
double: 8
```

Binary (base-2)

significance	128	64	32	16	8	4	2	1
bool	1	1	1	0	0	1	1	0

Unsigned int or char

- Value is limited to 2^n where n is bits in type
- Example
  - char – 8 bits
  - $2^8 = 256$
  - 0 – 255 can be represented with an unsigned char
- Overflows occur when an operation results in value > 255

Examples 1 & 2

unsigned and signed char

Signed int or char

- Value is limited to range from -(2^{n-1}) to 2^{n-1} - 1 where n is bits in type
- Example
  - char – 8 bits
  - 2^{8-1} = 128
  - -128 – 127 can be represented with a signed char
- Overflows occur when an operation results in value < -128 or > 127

How negative chars/ints work typically

1. Take positive value in binary
2. Invert all positions 0 -> 1 and 1-> 0
3. Add 1

Two’s Complement

Integer	Binary	Invert	Add 1	Result
0	0000	1111	0000	0
1	0001	1110	1111	-1
2	0010	1101	1110	-2
3	0011	1100	1101	-3
4	0100	1011	1100	-4
5	0101	1010	1011	-5
6	0110	1001	1010	-6
7	0111	1000	1001	-7

https://en.wikipedia.org/wiki/Two%27s_complement

Ex: 2 + -5

Integer	Binary	Invert	Add 1	Result
0	0000	1111	0000	0
1	0001	1110	1111	-1
2	0010	1101	1110	-2
3	0011	1100	1101	-3
4	0100	1011	1100	-4
5	0101	1010	1011	-5
6	0110	1001	1010	-6
7	0111	1000	1001	-7

Example 3

together now

What’s Happening

int i	unsigned char a	binary	int i	signed char b	binary
250	250	11111010	125	125	01111101
251	251	11111011	126	126	01111110
252	252	11111100	127	127	01111111
253	253	11111101	128	-128	10000000
254	254	11111110	129	-127	10000001
255	255	11111111	130	-126	10000010
256	0	00000000	131	-125	10000011
257	1	00000001	132	-124	10000100
258	2	00000010	133	-123	10000101
259	3	00000011	134	-122	10000110
260	4	00000100	135	-121	10000111

Problem 1: Wraparound (+ or -)

- Manifests with addition and subtraction operations resulting in
  - less than the minimum or
  - greater than maximum

- Generally, malicious input would lead to another flaw allowing code execution
  - Ex: User controlled output buffer

Examples 4, 5, 6

asterisks, percent, forward slashes oh my

Multiplication

significance	256	128	64	32	16	8	4	2	1
100	0	0	1	1	0	0	1	0	0
300	1	0	0	1	0	1	1	0	0

Division

significance	128	64	32	16	8	4	2	1
-128	1	0	0	0	0	0	0	0
one's compliment	0	1	1	1	1	1	1	1
two's compliment								

Division

- a = -128;
- a /= -1;
- -(-128)
  - Negation is just two’s compliment

significance	128	64	32	16	8	4	2	1
-128	1	0	0	0	0	0	0	0
one's compliment	0	1	1	1	1	1	1	1
two's compliment								

Modulus

Example 7

Casting tomfoolery

Problem 2: Wraparound (*, /, or %)

- Manifests with multiplication or division operations with results implicitly cast larger and outside the bounds of the type

- Generally, malicious input would lead to another flaw allowing code execution
  - Ex: User controlled output buffer

Discovery

- Code review
  - Any arithmetic operations, especially
    - With user input
    - Manipulating memory
  - Review of both your functions and called functions
    - Be sure you’re not implicitly casting a type incorrectly
- Compiler Output
  - Warnings for certain comparisons
- Testing
  - Fuzz with inputs of sizes around edges of types

Remediation / Defense

- Use unsigned numbers where possible
- Verify with math
  - Assert your own min and max within the constraints of your application and the types you're using
  - Validate against limit for the type in your case
- Use explicit casts for ease of review and to work through problems
- Use compiler flags to warn on implicit casts and trap on signed integer overflows

Remediation / Defense

- Use compiler assistance
  - gcc builtins (ex9_builtin.c)

Summary (From Book)

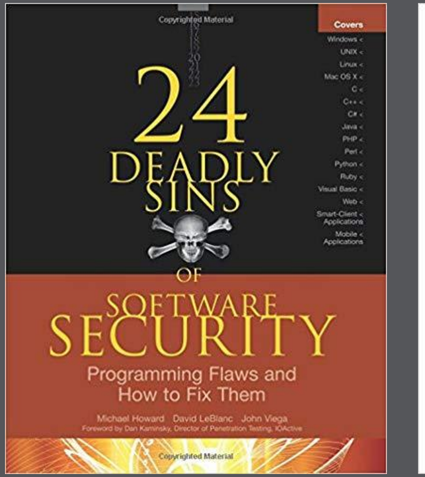
- Check all calculations used for memory allocations or array access
- Use unsigned integers (size_t) for memory and array access
- Check for truncation issues with unsigned types when subtracting
- Don’t think that this only happens in C/C++

Example

- Subtraction overflow

References

- https://Wikipedia.org
- 24 Deadly Sins of Software Security
  - ISBN-13: 978-0071626750
- The C Programming Language
  - ISBN-13: 978-0131103627


Copyrighted Material
Covers
Windows <
UNIX <
Linux <
Mac OS X <
C <
C++ <
C# <
Java <
PHP <
Perl <
Python <
Ruby <
Visual Basic <
Web <
Smart-Client < Applications
Mobile < Applications
24 DEADLY SINS OF SOFTWARE SECURITY
Programming Flaws and How to Fix Them
Michael Howard David LeBlanc John Viega
Foreword by Dan Kaminsky, Director of Penetration Testing, IOActive
Copyrighted Material