CSE 374: Programming Concepts and Tools

Eric Mullen
Spring 2017
Lecture 25: Undefined Behavior
It’s my Birthday!

• I’m super excited to tell you about one of my favorite topics today

• I brought you brownies, eat them

• Contains: flour, sugar, eggs, vanilla, chocolate, butter, pecans (in some)
Administrivia

• HW6 turned in last night, or using late days

• HW7 out today (demo at end of class)

• HW5 grading is almost done, we’ll grade HW6 as fast as we can

• Final review session 2-4pm on Tues, June 6 in CSE 403
Compilation

- In the beginning, the compiler was just a simple, straightforward translator
- As the age old story goes, we wanted our code to run faster
if (0)
    { do_something(); }  
int x=0;  
printf("%d\n", x);  

int x=0;  
printf("%d\n", x);
Compiler Optimization

- Your code doesn’t take a straight trip down
- All sorts of manipulation on the way down
- Compiler must maintain meaning of the program
Compiler’s Promise

I solemnly swear that the meaning of the output program will match the meaning of the input program*

*As long as the input program has meaning
What if program is weird?

```c
int x=0;
int y=0;
while (true) {
    y = x;
    x += 1;
    if (x <= y) {
        printf("weird");
    }
}
```
Overflow

\[\begin{array}{cccccccccc}
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
\hline
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{array}\]
What if program is weird?

```c
int x=0;
int y=0;
while (true) {
    y = x;
    x += 1;
    if (x <= y) {
        printf("weird");
    }
}
```
What if program is weird?

while (true) {
}

using clang on macOS, optimized with -O3
Undefined Behavior

• Compiler doesn’t have to maintain meaning if code doesn’t have meaning

• How to get faster code: declare all sorts of things to not have meaning, then allowed to do anything to them

• C may have taken this too far
Undefined Behavior

int y = x/0;

A. int y = 0;
B. int y = 5;
C. int y = -1;
D. format_drive();
E. launch_missiles();
Compiler’s Promise

I solemnly swear that the meaning of the output program will match the meaning of the input program*

If the input has no meaning, the compiler can do anything!

*As long as the input program has meaning
Does this happen in practice?

YES

Undefined Behavior: What Happened to My Code?
Wang et. al, APSys ‘12
Signed Overflow

• Signed overflow is undefined in C

• When you have the largest signed number, and you add more to it, the result is undefined according to the C language specification

• This allows for a lot of cool loop optimizations, but also puts us in awkward situations
How to test for overflow?

• Suppose $x \geq 0$ and $y > 0$

• If $x+y$ is negative, then overflow occurred

• This is problematic…
Example

int do_fallocate(..., loff_t offset, loff_t len)
{
    struct inode *inode = ...;
    if (offset < 0 || len <= 0)
        return -EINVAL;
    /* Check for wrap through zero too */
    if ((offset + len > inode->i_sb-s_maxbytes) || (offset + len < 0))
        return -EFBIG;
    ...
}

fs/open.c
Linux Kernel
int do_fallocate(...,loff_t offset, loff_t len)
{
    struct inode *inode = ...;
    if (offset < 0 || len <= 0)
        return -EINVAL;
    /* Check for wrap through zero too */
    if ((offset + len > inode->i_sb->s_maxbytes) ||
        (offset + len < 0))
        return -EFBIG;
    ...
}

fs/open.c
Linux Kernel
Division by 0

• division by 0 in C is undefined behavior

• If division by 0 ever occurs, entire program has no meaning, and can be transformed into anything

• In practice compilers generate nothing whenever they can
Example

if (msize == 0)
    msize = 1 / msize; /* provoke a signal */

Result: Entire check removed

from the Linux Kernel:
lib/mpi/mpi-pow.c
Uninitialized Read

• In C, you can make up a variable without putting something in it

• This variable is called “uninitialized”

• If you read from it, that is undefined behavior
Uninitialized Read

• Thought: if there’s nothing well defined in there, maybe it’s just “kinda random”

• We could use that, with some other stuff, to seed our random number generator
Example

struct timeval tv;
unsigned long junk; /* XXX left uninitialized on purpose */

gmtimeofday(&tv, NULL);
srandom((getpid() << 16) ^ tv.tv_sec ^
        tv.tv_usec ^ junk);

lib/libc/stdlib/rand.c in FreeBSD libc
Example

```c
struct timeval tv;
unsigned long junk; /* XXX left uninitialized on purpose */

gmtime(&tv, NULL);
srandom((getpid() << 16) ^ tv.tv_sec ^ tv.tv_usec ^ junk);
```

lib/libc/stdlib/rand.c in FreeBSD libc
How to solve?

• we have bandaids not cures

• many different compiler flags to disable optimizations (\(-fwrapv\) gives meaning to signed overflow)

• flags not complete, even if they were wouldn’t be satisfying

• we need something better