Undefined Behavior and Compiler Optimization

Why Your Program Stopped Working With A Newer Compiler

Presented by

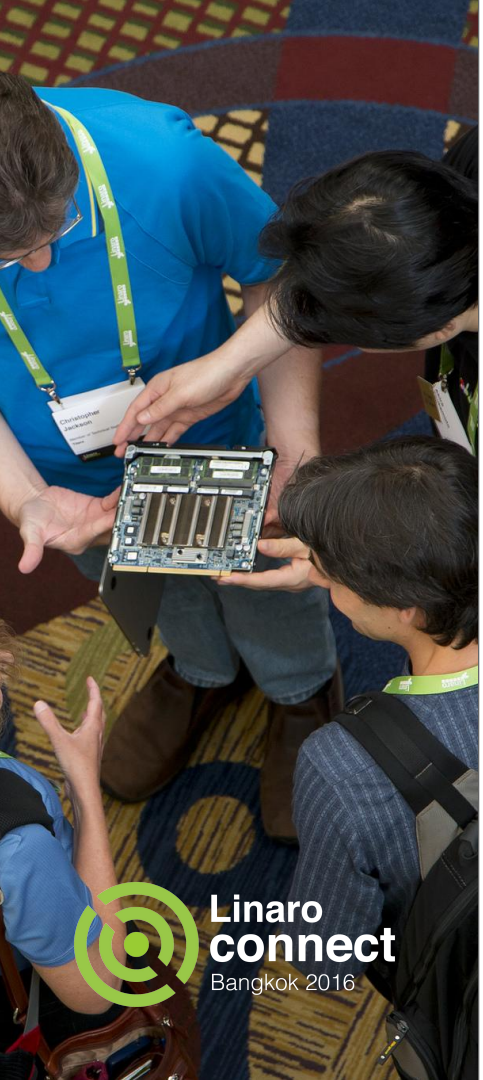
Kugan Vivekanandarajah and Yvan Roux

Date

BKK16-503 March 11, 2016

Event

Linaro Connect BKK16



Outline

- What is undefined behavior
- Examples for undefined behavior
- Detecting undefined behavior
- Conclusion

How many times does this loop iterate ?

```
int d[16];
int SATD (void)
{
  int satd = 0, dd, k;
  for (dd = d[k = 0]; k<16; dd = d[++k])
  {
    satd += (dd < 0 ? -dd : dd);
  }
  return satd;
}
```

How many times does this loop iterate ?

```
int d[16];
int SATD (void)
{
  int satd = 0, dd, k;
  for (dd = d[k = 0]; k<16; dd = d[++k])
  {
    satd += (dd < 0 ? -dd : dd);
  }
  return satd;
}
```

Infinite loop generated on non-infinite code (?)

- 464.h264ref goes into infinite loop for gcc 4.8
- https://gcc.gnu.org/PR53073

How many times does this loop iterate ?

```
int d[16];
int SATD (void)
{
  int satd = 0, dd, k;
  for (dd = d[k = 0]; k<16; dd = d[++k])
  {
    satd += (dd < 0 ? -dd : dd);
  }
  return satd;
}
```

- C standard says:
  - It is legal for a pointer to point to one element past the end
  - Accessing that location is undefined
- Compiler can therefore assume that the “k” can never be 16 at the point of k < 16

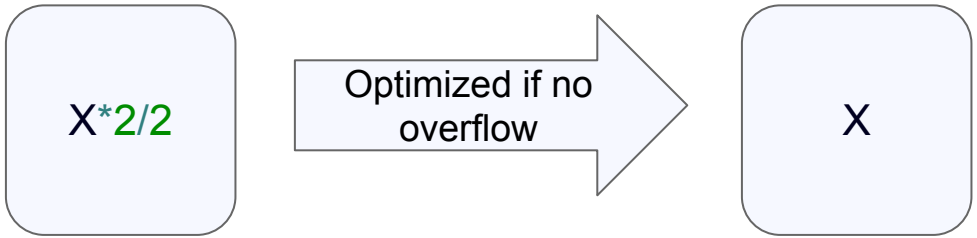
What is undefined behavior

- ISO Standard definition:
  “behavior, upon use of a nonportable or erroneous program construct or of erroneous data, for which this International Standard imposes no requirements”
- C FAQ definition:
  “Anything at all can happens; the Standard imposes no requirements. The program may fail to compile, or it may execute incorrectly (either crashing or silently generating incorrect results), or it may fortuitously do exactly what the programmer intended”
- They may lead to very subtle bugs or have critical impact on security
  - Must be avoided by programmers

Why undefined behavior exists ?

- C/C++ is designed to be an efficient low-level programming language
- Offers compiler writers freedom to optimize
- Safe languages like Java have few undefined behavior
  - Safe and reproducible behavior across implementations
  - At the expense of performance

Example:



Other kinds of behavior

- Implementation-defined behavior:
  unspecified behavior where each implementation documents how the choice is made

Other kinds of behavior

- Implementation-defined behavior:
  unspecified behavior where each implementation documents how the choice is made
- Unspecified behavior:
  use of an unspecified value, or other behavior where this International Standard provides two or more possibilities and imposes no further requirements on which is chosen in any instance

Other kinds of behavior

- Implementation-defined behavior:
  unspecified behavior where each implementation documents how the choice is made
- Unspecified behavior:
  use of an unspecified value, or other behavior where this International Standard provides two or more possibilities and imposes no further requirements on which is chosen in any instance
- Locale-specific behavior:
  behavior that depends on local conventions of nationality, culture, and language that each implementation documents

Examples for undefined behavior

- Signed integer overflow
- Shifting an n-bit integer by n or more bits
- Divide by zero
- Dereferencing a NULL pointer
- Pointer arithmetic that wraps
- Two pointers of different types that alias
- Reading an uninitialized variable

How to detect undefined behavior

- Compiler warnings
  - The compiler can issue a warning at compile time when it can statically detect some kind of wrongdoing
- Undefined Behavior Sanitizer - ubsan
  - GCC gained ubsan support from version 4.9
    - Run-time checker for the C and C++ languages
  - Some undefined uses will not be obvious
    - Input might come from user / other modules and statically not detectable
    - Ubsan can help here with right input

What will be the output when int is 32 bit ?

```
#include <stdio.h>
int foo (int a) {
if (a + 100 > a)
        printf ("%d GT %d\n", a + 100, a);
    else
        printf ("%d LT %d\n", a + 100, a);
    return 0;
}
int main () {
    foo (100);
    foo (0x7fffffff);
    return 0;
}
```

Signed integer overflow - PR30475

```
#include <stdio.h>
int foo (int a) {
if (a + 100 > a)
        printf ("%d GT %d\n", a + 100, a);
    else
        printf ("%d LT %d\n", a + 100, a);
    return 0;
}
int main () {
    foo (100);
    foo (0x7fffffff);
    return 0;
}
```

- Output At -O0

```
200 GT 100
-2147483549 LT 2147483647
```

- Output At -O2

```
200 GT 100
-2147483549 GT 2147483647
```

Signed integer overflow - PR30475

- According to C and C++ language standards overflow of a signed value is undefined behavior
  - A correct (standard conforming) C/C++ program must never generate signed overflow
- (int + 100 > int) in example is always true
- -fno-strict-overflow /-fwrapv disables it

gcc -O2 t.c -Wstrict-overflow
t.c: In function 'foo':
t.c:4:5: warning: assuming signed overflow does not occur when assuming that (X + c) >= X is always true [-Wstrict-overflow]

gcc -O2 t.c -fsanitize=undefined ; ./a.out
200 GT 100
t.c:5:7: runtime error: signed integer overflow: 2147483647 + 100 cannot be represented in type 'int'
-2147483549 GT 2147483647

Signed integer overflow and security implications

- Defence against many software security problems is validating input
- If the validating code is undefined, code can be vulnerable
- An example from https://lwn.net/Articles/278137/
  - If the “len” comes from user, then it can be used to overflow the buffer (and exploit)
- Some of these overflow checks are used as common idioms to prevent buffer overflows in security sensitive code

```
if (buffer + len >= buffer_end
      || buffer + len < buffer)
    die_a_gory_death ("len is out of range\n");
```

Will be optimized into:

```
if (buffer + len >= buffer_end)
      die_a_gory_death ("len is out of range\n");
```

What will be the output for this ?

```
#include <stdio.h>
int foo (int x, int y)
{
  x >>= (sizeof (int) << y);
  return x;
}
int main ()
{
  printf ("%d\n", foo (1000, 3));
  return 0;
}
```

Shifting n-bit integer by n or more bits undefined - PR48418

```
#include <stdio.h>
int foo (int x, int y)
{
  x >>= (sizeof (int) << y);
  return x;
}
int main ()
{
  printf ("%d\n", foo (1000, 3));
  return 0;
}
```

gcc t.c -O0; ./a.out

1000

gcc t.c -O2; ./a.out

0

gcc -O2 t.c -fsanitize=undefined ; ./a.out

t.c:5:4: runtime error: shift exponent 32 is too large for 32-bit type 'int'

1000

What will be the output for this ?

```
#include <stdio.h>
int testdiv (int i, int k) {
    if (k == 0) printf ("found divide by
zero\n");
    return (i / k);
}
int main() {
    int i = testdiv (1, 0);
    return (i);
}
```

Divide by zero - PR29968

- Divide by zero is undefined
- Based on error found with PostgreSQL 8.1.5 on Solaris 9 sparc with gcc-4.1
- Since k is divisor, compiler assumed “k” cannot be zero
  - print statement is optimized away

```
#include <stdio.h>
int testdiv (int i, int k) {
    if (k == 0) printf ("found divide by
zero\n");
    return (i / k);
}
int main() {
    int i = testdiv (1, 0);
    return (i);
}
```

Divide by zero - PR29968

- Divide by zero is undefined
- Based on error found with PostgreSQL 8.1.5 on Solaris 9 sparc with gcc-4.1
- Since k is divisor, compiler assumed “k” cannot be zero
  - print statement is optimized away

```
#include <stdio.h>
int testdiv (int i, int k) {
    if (k == 0) printf ("found divide by
zero\n");
    return (i / k);
}
int main() {
    int i = testdiv (1, 0);
    return (i);
}
```

An example from Linux : (http://www.spinics.net/linux/fedora/linux-security-module/msg12814.html)

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

What will be the output for this ?

```
unsigned char* addr = \
     (unsigned char*)0xfffffffe;
   unsigned len = 4;

   if (addr + len < addr)
   {
      printf( "wraps\n");
   }
   else
   {
      printf( "no wrap\n");
   }
```

Pointer arithmetic that wraps - PR54365 (for ARM)

```
unsigned char* addr = \
     (unsigned char*)0xfffffffe;
   unsigned len = 4;

   if (addr + len < addr)
   {
      printf( "wraps\n");
   }
   else
   {
      printf( "no wrap\n");
   }
```

gcc -O2 t.c; ./a.out

no wrap

gcc t.c; ./a.out

wraps

- Current version of ubsan does not model this
  - No errors flagged
  - Not all the undefined behaviours are modelled in ubsan / as compiler warnings

Dereferencing a NULL pointer

- Example from Linux Kernel (https://lwn.net/Articles/342330/)

```
static unsigned int tun_chr_poll (struct file
*file, poll_table * wait)
    {
    struct tun_file *tfile = file->private_data;
    struct tun_struct *tun = __tun_get(tfile);
    struct sock *sk = tun->sk;
    unsigned int mask = 0;

    if (!tun)
        return POLLERR;
   ....
   }
```

Calling a NULL Object - PR68853

- gcc-6 exposes undefined behavior in Chromium v8 garbage collector
  - Calling a NULL object is undefined
  - -fno-delete-null-pointer-checks gets this to work

Reading an uninitialized variable

- Reading an uninitialized variable is undefined behavior
  - Compiler can assign any value to the variable and expressions derived from the variable

```
struct timeval tv;
unsigned long junk;

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

- http://kqueue.org/blog/2012/06/25/more-randomness-or-less/
  - When compiled with a version of LLVM, entire seed computation is optimized away
  - Results of gettimeofday () and getpid () are not used at all
  - srandom () is called with some garbage value.

Conclusion

- Undefined behavior means standard non conforming code
  - Compilers will assume that undefined behavior is not present in the code while optimizing
- Undefined behavior can be subtle and not always obvious
- Use compiler warnings and ubsan to detect them
  - Unfortunately, not all the undefined behavior are modelled in ubsan
  - Know the compiler flags (if any) to disable optimization that might be relying on undefined behavior

Useful Links:

- http://blog.regehr.org/archives/213
- http://blog.llvm.org/2011/05/what-every-c-programmer-should-know.html
- http://developerblog.redhat.com/2014/10/16/gcc-undefined-behavior-sanitizer-ubsan
- http://www.open-std.org/
- https://gcc.gnu.org/onlinedocs/gcc/Standards.html
- http://www.open-std.org/jtc1/sc22/wg14/www/docs/n1124.pdf