Андрей Карпов

Лекция 11/12

Неопределённое поведение, или как выстрелить себе в ногу

Докладчик

Карпов
Андрей Николаевич

- Сооснователь компании PVS-Studio, технический директор
- Разработал первые реализации ядра C++ анализатора
- Автор статей о качестве кода и статическом анализе

Зачем это нужно

- C++ компиляторы создают всё более эффективный код
- Плата за это: плохой код всё чаще приводит к неопределённому поведению
- Узнаем побольше про неопределённое поведение
- И почему важно писать качественный код, не закладываясь на "уж я то знаю, что делаю"

Неуточнённое поведение

- Неуточнённое поведение (unspecified behavior) или поведение, определяемое реализацией — поведение компьютерной программы, которое может различаться на разных платформах и компиляторах поскольку спецификация языка программирования предлагает несколько допустимых вариантов реализации некой языковой конструкции.
- В отличие от неопределённого поведения, программа с неуточняемым поведением с точки зрения соответствия спецификации языка не считается ошибочной.
- Однако, нельзя писать код в котором есть неуточнённое поведение.

Неуточнённое поведение

```
int A = 0;
Foo(A = 2, A);
```

Неуточнённое поведение

- The value of E1 >> E2 is E1 right-shifted E2 bit positions. If E1 has an unsigned type or if E1 has a signed type and a non-negative value, the value of the result is the integral part of the quotient of E1/2^E2. If E1 has a signed type and a negative value, the resulting value is **implementation-defined**.

```
int B = (-1) >> 5;
```

Неопределённое поведение

- Неопределённое поведение (undefined behaviour) — свойство некоторых языков программирования в определённых маргинальных ситуациях выдавать результат, зависящий от реализации компилятора и случайных факторов наподобие состояния памяти или сработавшего прерывания.
- Зачем?
  - Ускоряется работа программ (так как не нужно проверять всевозможные «маргинальные» случаи)

Пример: выход за границу массива

```
void foo()
{
   ....
   int A[10];
   ...
   A[22] = 10;
   ...
}
```

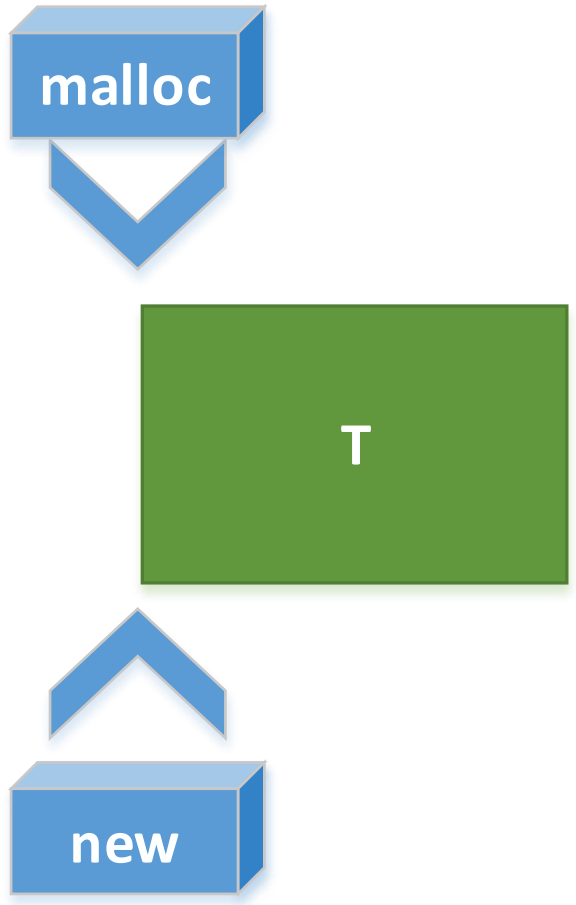
Пример: нет виртуального деструктора

```
struct Base {
  virtual ~Base() = default;
  virtual void f();
};

struct Derived : Base {};

void f() {
  Base *b = new Derived();
  // ...
  delete b;
}
```

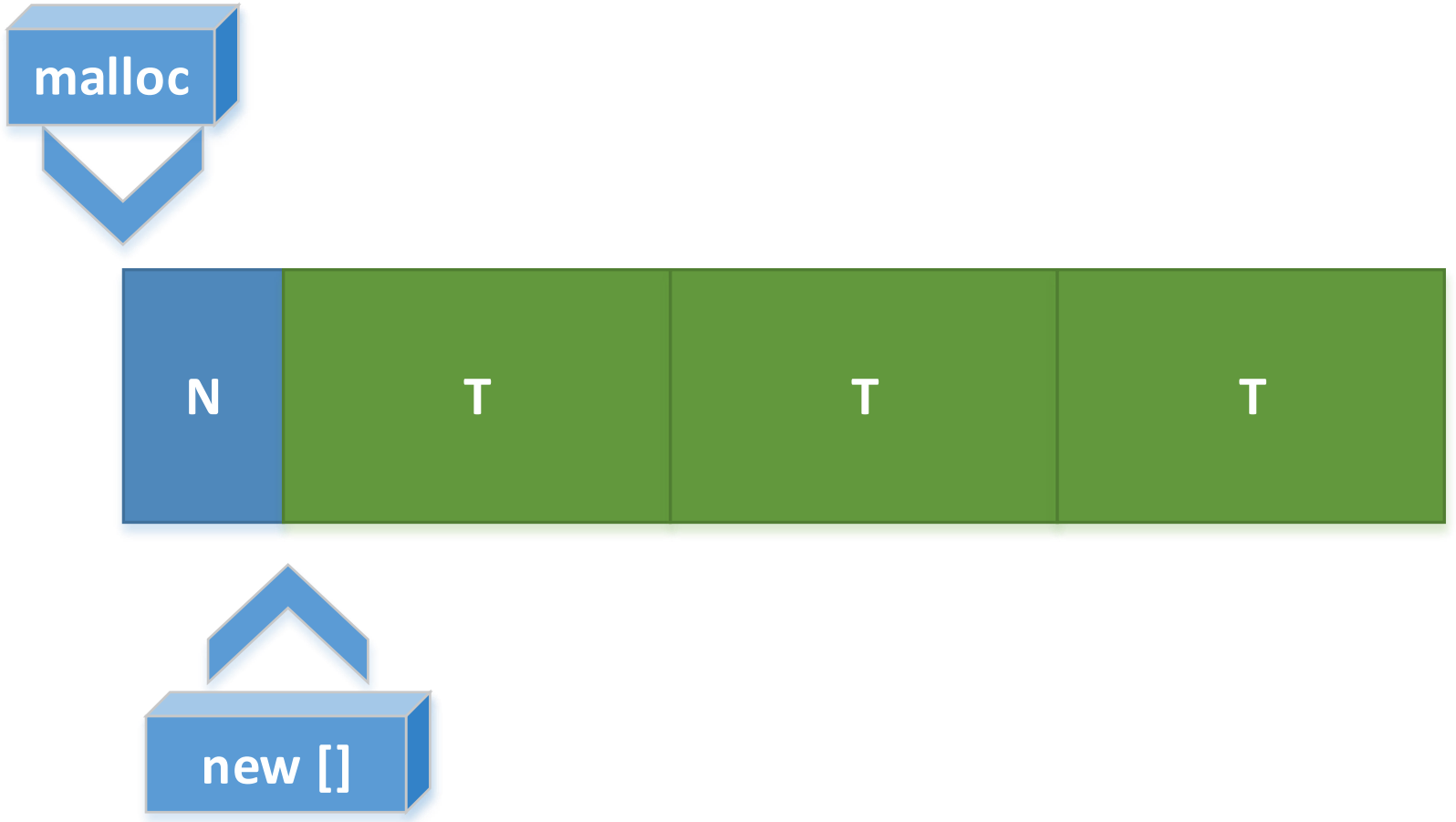
new[] / delete

- new[] -> delete
- new -> delete []
- new[] -> free
- new -> free

new[] / delete



new[] / delete



Если бы всё так было просто…

- Основную беду представляют не предыдущие очевидные случаи
- Программисты соглашаются, что писать за границу массива - это плохая идея

- Проблема в том, что иногда они думают, что знают, как и что будет происходить при нарушении правил языка C и C++

Нулевые указатели

```
static int podhd_try_init(struct usb_interface *interface,
          struct usb_line6_podhd *podhd)
{
  int err;
  struct usb_line6 *line6 = &podhd->line6;

  if ((interface == NULL) || (podhd == NULL))
    return -ENODEV;
  ....
}
```

Нулевые указатели

```
#define offsetof(st, m) ((size_t)(&((st *)0)->m))
```

Нулевые указатели

```
static int podhd_try_init(struct usb_interface *interface,
          struct usb_line6_podhd *podhd)
{
  int err;
  struct usb_line6 *line6 = &podhd->line6;

  if ((interface == NULL) || (podhd == NULL))
    return -ENODEV;
  ....
}
```

Нулевые указатели

Разыменовывание нулевого указателя приводит к неопределённому поведению
https://www.viva64.com/ru/b/0306/

Нулевые указатели (malloc)

```
static void
st_collections_group_parts_part_description_filter_data(void)
{
   ....
    filter->data_count++;
    array = realloc(filter->data,
      sizeof(Edje_Part_Description_Spec_Filter_Data) *
      filter->data_count);
    array[filter->data_count - 1].name = name;
    array[filter->data_count - 1].value = value;
    filter->data = array;
}
```

Сдвиги

- Библиотека JPEG
- Нужны значения:
  - `....`
  - `11...1111111111101b`
  - `11...1111111111001b`
  - `11...1111111110001b`
  - `11...1111111100001b`
  - `....`

Сдвиги

```
/* entry n is (-1 << n) + 1 */
static const int extend_offset[16] = { 0,
  ((-1)<<1) + 1, ((-1)<<2) + 1, ((-1)<<3) + 1,
  ((-1)<<4) + 1, ((-1)<<5) + 1, ((-1)<<6) + 1,
  ((-1)<<7) + 1, ((-1)<<8) + 1, ((-1)<<9) + 1,
  ((-1)<<10) + 1, ((-1)<<11) + 1, ((-1)<<12) + 1,
  ((-1)<<13) + 1, ((-1)<<14) + 1, ((-1)<<15) + 1
};
```

Сдвиги

- The value of E1 << E2 is E1 left-shifted E2 bit positions; vacated bits are zero-filled. If E1 has an unsigned type, the value of the result is E1 * 2^E2, reduced modulo one more than the maximum value representable in the result type. **Otherwise, if E1 has a signed type and non-negative value, and E1*2^E2 is representable in the result type, then that is the resulting value; otherwise, the behavior is undefined.**

Сдвиги

```
static const unsigned extend_offset[16] =
{ 0,
  ((~0u)<<1) | 1, ((~0u)<<2) | 1, ((~0u)<<3) | 1,
  ((~0u)<<4) | 1, ((~0u)<<5) | 1, ((~0u)<<6) | 1,
  ((~0u)<<7) | 1, ((~0u)<<8) | 1, ((~0u)<<9) | 1,
  ((~0u)<<10) | 1, ((~0u)<<11) | 1, ((~0u)<<12) | 1,
  ((~0u)<<13) | 1, ((~0u)<<14) | 1, ((~0u)<<15) | 1
};
```

Целочисленное переполнение (64-bit bug)

```
size_t Count = (size_t)(5) * 1024 * 1024 * 1024; // 5 Gb

char *array = (char *)malloc(Count);
memset(array, 0, Count);

int index = 0;
for (size_t i = 0; i != Count; i++)
  array[index++] = (char)(i) | 1;

if (array[Count - 1] == 0)
  printf("The last array element contains 0.\n");
```

Целочисленное переполнение (64-bit bug)

- *V575 [CWE-628] The potential null pointer is passed into 'memset' function. Inspect the first argument. Check lines: 30, 29. c_file.c 30*
- V127 An overflow of the 32-bit 'index' variable is possible inside a long cycle which utilizes a memsize-type loop counter. c_file.c 34
- V108 Incorrect index type: array[not a memsize-type]. Use memsize type instead. c_file.c 34

- Undefined behavior ближе, чем вы думаете
  https://www.viva64.com/ru/b/0374/

Целочисленное переполнение (64-bit bug)

```
  int index = 0;
  for (size_t i = 0; i != Count; i++)
000000013F6D102D  xor         ecx,ecx
000000013F6D102F  nop
    array[index++] = char(i) | 1;
000000013F6D1030  movzx       edx,cl
000000013F6D1033  or          dl,1
000000013F6D1036  mov         byte ptr [rcx+rbx],dl
000000013F6D1039  inc         rcx
000000013F6D103C  cmp         rcx,rdi
000000013F6D103F  jne         main+30h (013F6D1030h)
```

Целочисленное переполнение (64-bit bug)

```
  unsigned index = 0;
000000013F07102D  xor         r9d,r9d
  for (size_t i = 0; i != Count; i++)
000000013F071030  mov         ecx,r9d
000000013F071033  nop         dword ptr [rax]
000000013F071037  nop         word ptr [rax+rax]
    array[index++] = char(i) | 1;
000000013F071040  movzx       r8d,cl
000000013F071044  mov         edx,r9d
000000013F071047  or          r8b,1
000000013F07104B  inc         r9d
000000013F07104E  inc         rcx
000000013F071051  mov         byte ptr [rdx+rbx],r8b
000000013F071055  cmp         rcx,rdi
000000013F071058  jne         main+40h (013F071040h)
```

unsigned

Целочисленное переполнение (CRC)

```
int foo(const unsigned char *s)
{
  int r = 0;
  while(*s) {
    r += ((r * 20891 + *s *200) | *s ^ 4 | *s ^ 3) ^ (r >> 1);
    s++;
  }
  return r & 0x7fffffff;
}
```

Форум — Development

gcc-8 совсем поломанный

gcc

Я так понимаю в последних версиях gcc хипстеры вообще всё сишку решили сломать. Хотя пишут статейки что C никогда не был портабельным ассемблером, но ведь использовали его так и куча кода накопилась. Теперь этот весь код сломан. Например сброс знака по (a & 0x7fffffff) не работает и много всего вообще теперь не работает и писать теперь надо как на плюсах. И страшно бояться любого UB так как оно будет страшно падать на ровном месте и 2+2 будет 5. Особенно это опасно, если лет 30 устоявшиеся методы для решения задач были, и эти чувырлы с UB головного мозга это всё радостно ломают. Я так понимаю это делается специально, чтобы кому-то не было скучно на работе, а реального толку 0. Какой сейчас более-менее адекватный gcc, 5.4? Шланг не предлагать, он никогда не был компилятором, вообще. Это они весь сыр-бор и начали. Я так понимаю реально там полезное сейчас только в C++ делают, в C только ломают.

3

7

https://www.linux.org.ru/forum/development/14422428

Целочисленное переполнение (CRC)

```
int foo(const unsigned char *s)
{
  int r = 0;
  while(*s) {
    r += ((r * 20891 + *s *200) | *s ^ 4 | *s ^ 3) ^ (r >> 1);
    s++;
  }
  return r & 0x7fffffff;
}
```

PVS-Studio: V1026. The variable is incremented in the loop.
Undefined behavior will occur in case of signed integer overflow.

Как писать надёжный код?

- Хорошее теоретическое знание языка C++
- Максимально простые и понятные конструкции (привет языку Ада)
- Обзоры кода
- Придерживаться стандарта кодирования
- Статический анализ кода
- Динамический анализ кода

- P.S. Юнит тесты и ручное тестирование для борьбы с UB подходят плохо

Выводы

- Никто не знает, как работает код, содержащий неопределённое поведение
- Пишите простой понятный код
- Профессионализм не в том, чтобы ходить по лезвию бритвы, а чтобы написать безопасный код, который легко читать и поддерживать

Дополнительные ссылки

1. Не зная брода, не лезь в воду. Часть третья
   https://www.viva64.com/ru/b/0142/
2. Разыменовывание нулевого указателя приводит к неопределённому поведению
   https://www.viva64.com/ru/b/0306/
3. Undefined behavior ближе, чем вы думаете
   https://www.viva64.com/ru/b/0374/
4. A Guide to Undefined Behavior in C and C++
   https://blog.regehr.org/archives/213
   https://blog.regehr.org/archives/226
   https://blog.regehr.org/archives/232

Дополнительные ссылки

5. Скользкие места C++

6. Fun with NULL pointers
   https://lwn.net/Articles/342330/
   https://lwn.net/Articles/342420/

7. What Every C Programmer Should Know About Undefined Behavior
   http://blog.llvm.org/2011/05/what-every-c-programmer-should-know.html
   http://blog.llvm.org/2011/05/what-every-c-programmer-should-know_14.html
   http://blog.llvm.org/2011/05/what-every-c-programmer-should-know_21.html

END

Q&A