Ошибки в коде: ожидания и реальность

Инструменты - Статический анализ исходного кода (SAST)

Андрей Карпов

PVS-Studio, DevRel

Андрей Карпов

- Один из основателей PVS-Studio
- Более 15 лет в сфере анализа кода
- Автор статей про C++ и качество кода
- Хабр: @Andrey2008

Основа моих наблюдений и выводов

- 15 лет команда PVS-Studio пишет статьи про ошибки, найденные в открытых проектах
- Сейчас коллекция насчитывает **15 600 багов**

- Не 15К+ предупреждений, именно 15К+ ошибок!
- Нам приятен наш вклад в открытые проекты
- https://pvs-studio.ru/ru/blog/examples/

Ожидания

Неинициализированные переменные

- Везде про них
  - Книги
  - Статьи типа “Проклятие неинициализированных переменных”

Неинициализированные переменные

- Про них так много говорят, что с этим программисты внимательны
- Ошибки хорошо ловятся компиляторами

```
int i;
int A[10] = { 0 };
return A[i];
```

- Clang: variable 'i' is uninitialized when used here
- MSVC: C4700: uninitialized local variable 'i' used
- GCC: 'i' is used uninitialized

Неинициализированные переменные

- Загадайте число: сколько ошибок этого типа мы нашли за 10+ лет в открытых проектах?
- Пример реальной ошибки из Godot Engine (C++):

```
const char* CPPlayer::get_voice_sample_name(int p_voice)
{
  const char *name;
  if (!voice[p_voice].sample_ptr)
    name=voice[p_voice].sample_ptr->get_name();
  return name;
}
```

Неинициализированные переменные

- Загадайте число: сколько ошибок этого типа мы нашли за 10+ лет в открытых проектах (C, C++, C#, Java)?
- **120** ошибок (если добавить неинициализированные члены классов)
- Кажется, немало. Запомним это число.

Деление на 0

- Классика ведь… кажется...
- Везде про деление на 0
- Даже мемы есть

Деление на 0

- На самом деле, редкие краевые случаи, как в проекте Inkscape (C++)

```
} else if (type >= 3000 && type < 4000) {
  unsigned int chamferSubs = type-3000;
  ....
  // При type == 3000 получим NaN
  double chamfer_stepsTime = 1.0/chamferSubs;
```

Деление на 0

- 10+ лет анализа открытых проектов (C, C++, C#, Java)
- **20** ошибок

Выход за границу массива

- Примечание: не учитывая выход за границу буфера, например, при *strcat*
- Классический выход за границу массива в проекте Umbraco (C#)

```
case "SqlText":
  ....
  if (m.Arguments.Count == 2)
  {
    var n1 = Visit(m.Arguments[0]);
    var f = m.Arguments[2];
    ....
```

Выход за границу массива

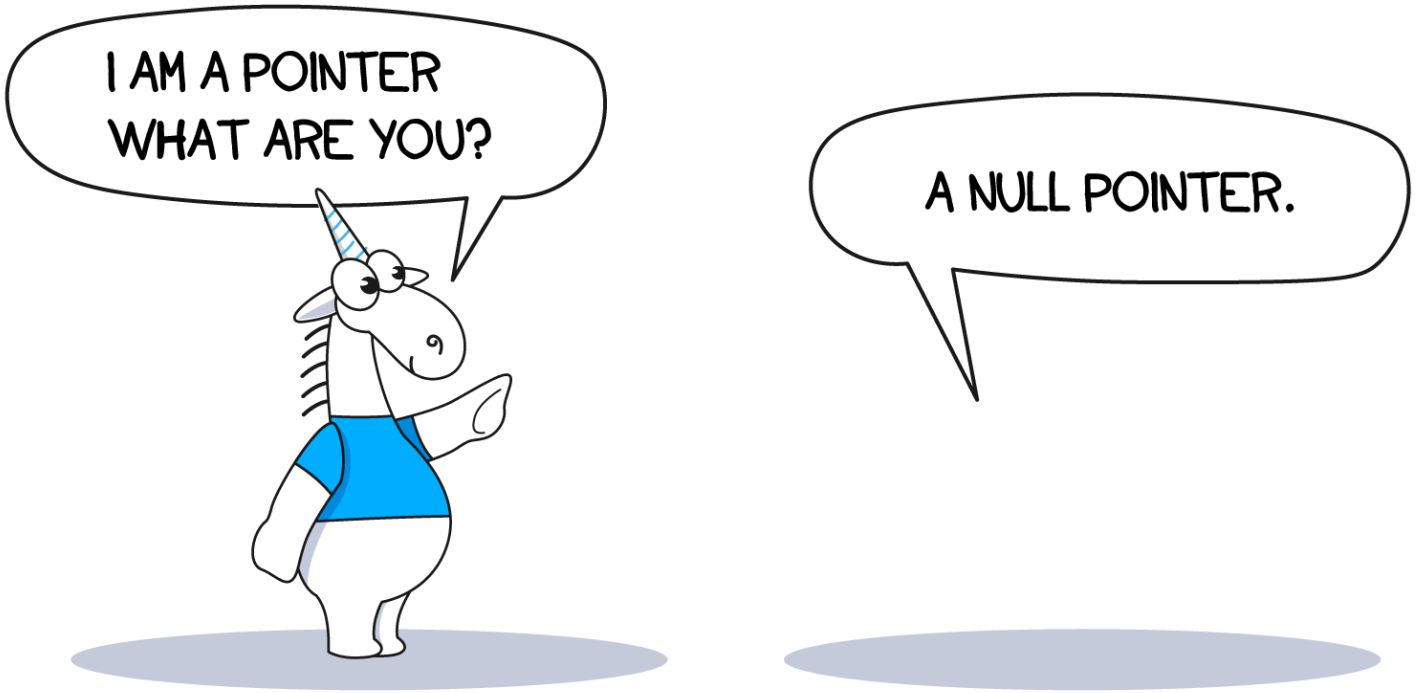
- Есть, но меньше ожидаемого
- Быстро выявляется предупреждениями
- Или при запуске (исключения)
- Или при тестировании

- 10+ лет анализа открытых проектов (C, C++, C#, Java)

- **215** ошибок

Где-то ощущения подтверждаются большим количеством реальных багов?

- Да, это разыменование нулевых указателей / ссылок



Классический пример: разыменование, потом проверка

- Проект Microsoft PowerToys (C#)

```
public List<Result> Query(Query query)
{
  bool isGlobalQuery = string.IsNullOrEmpty(query.ActionKeyword);
  ....
  if (query == null)
  {
    throw new ArgumentNullException(paramName: nameof(query));
  }
```

Нулевые ссылки и указатели

- 10+ лет анализа открытых проектов (C, C++, C#, Java)
- **3900** ошибок

Ожидания сбивают разработчиков анализаторов и тестовых наборов

Что объединяет рассмотренные ошибки?

- Их мало, но кажется, что много
  - Неинициализированные переменные
  - Деление на 0
  - Выход за границу массива
- Нулевые ссылки и указатели (их действительно много)

- Для их выявления нужен анализ потока данных!

Обманки: фокусировка на анализе потока данных

- Особенно все зацикливаются на разыменовании нулевых указателей / ссылок

- Но:
  - Часто они не критичны (на практике не проявляют себя)
  - В таких языках, как C#, это исключение (а не UB как в C++)
  - Критические моменты быстро отлавливаются на ранних этапах любого типа тестирования

Обманки: фокусировка на анализе потока данных

- **Можно сильно заморачиваться, выискивая более сложные ситуации, но это практически не улучшает качество кода в целом**
  - Некоторых ошибок просто мало (деление на 0)
  - Нулевые указатели и ссылки не так страшны, как кажется

Отталкиваясь от анализа потока данных, придумывается анализ несуществующих ошибок

- Из Toyota ITC

```
void null_pointer_006()
{
  int* p;
  p = (int*)(intptr_t)rand();
  *p = 1; /*Tool should detect this line as error*/
  /*ERROR:NULL pointer dereference*/
}
```

- Придумали. Какие-то анализаторы будут находить. Но к реальности это отношения не имеет.

Тест на выявление выдуманной ошибки

- Проблема не в NULL. Указатель в принципе нельзя использовать.
- rand() редко используемая функция. На практике таких ситуаций нет!

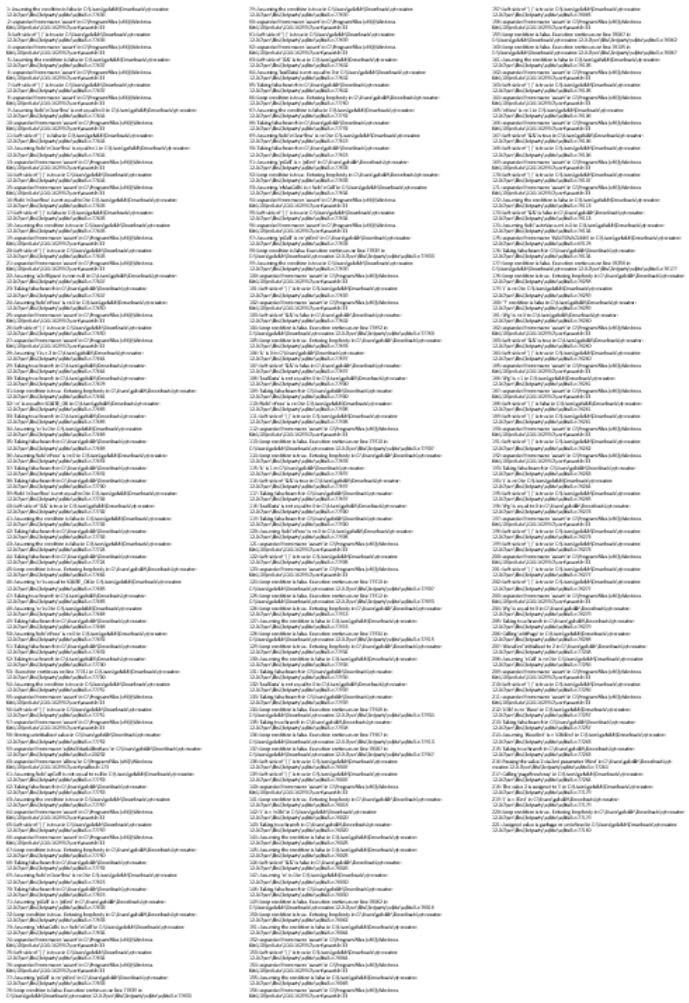
```
void null_pointer_006()
{
  int* p;
  p = (int*)(intptr_t)rand();
  *p = 1; /*Tool should detect this line as error*/
  /*ERROR:NULL pointer dereference*/
}
```

Обманки: фокусировка на трассе выполнения

- Это следствие фокусировки на анализе потока данных
- Кажется, что если важно отследить все шаги, то и полезно подробно показать трассу выполнения и данных
- Да, но нет
- Чем больше показываешь, и чем длиннее путь, тем менее понятно
- Или смотрится «душно» (не специально, но тебя держат за дурачка)

Assigned value is garbage or undefined [clang-analyzer-core.uninitialized.Assign]

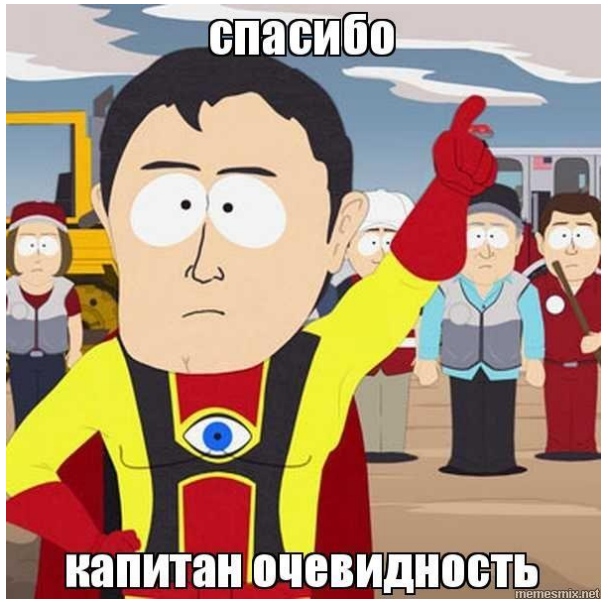
- В трассе 221 пункт
- Ну «спасибо», сейчас налью кофе и начну разбираться
- И это только одно предупреждение



Откуда ощущение, что тебя держат за дурачка

- 22: Assuming 'aOvflSpace' is non-null in ….\sqlite\sqlite3.c:77647

```
77647    if( !aOvflSpace ){
77648      return SQLITE_NOMEM_BKPT;
77649    }
```



- Мы доберёмся до ошибки, если *aOvflSpace* не нулевой
- Как будто я этого из кода не вижу

Пояснение

- Я не говорю, что анализ потока данных малополезен
- Очень полезен
- Просто им увлекаются в ущерб другим направлениям
- Анализ получается однобоким

Реальность

```
void sha1_hmac_finish(sha1_context* ctx,
                      unsigned char output[20])
{
    unsigned char tmpbuf[20];

    sha1_finish(ctx, tmpbuf);
    sha1_starts(ctx);
    sha1_update(ctx, ctx->opad, 64);
    sha1_update(ctx, tmpbuf, 20);
    sha1_finish(ctx, output);

    memset(tmpbuf, 0, sizeof(tmpbuf));
}
```

RPCS3, C++

Классика – CWE-14: V597 The compiler could delete the 'memset' function call, which is used to flush 'tmpbuf' buffer. sha1.cpp 371

Известный паттерн потенциальной уязвимости

- Что код содержит CWE-14, удивительного нет
- Многие программисты не знают этот нюанс работы оптимизирующего компилятора
- Удивительно, как часто можно встретить такой код
- **330** ошибок

Мини игра: загадайте, что чаще встречается

Неинициализированные переменные

vs

Повторная запись в переменную

Сейчас будет пара примеров, что имеется в виду

```
Context.eflags = 0x88898a8b;
Context.cs = 0x8c8d;
Context.fs = 0x8e8f;
Context.gs = 0x9091;
Context.ss = 0x9293;
Context.ds = 0x9495;
Context.ss = 0x9697;
llvm::ArrayRef<uint8_t>
  ContextRef(reinterpret_cast<uint8_t*>(&Context),
             sizeof(Context));
```

LLVM/Clang, C++

V519 [CWE-563, CERT-MSC13-C] The 'Context.ss' variable is assigned values twice successively. Perhaps this is a mistake. Check lines: 110, 112. RegisterContextMinidumpTest.cpp 112

```
public void SetString(string key, string value)
{
  if (m_syncDictionary.TryGetValue(key,
                                   out string existingValue))
  {
    if (value != existingValue)
    {
      m_syncDictionary[key] = value;
    }
  }
  m_syncDictionary[key] = value;
}
```

SanAndreasUnity, C#

V3008 The 'm_syncDictionary[key]' variable is assigned values twice successively. Perhaps this is a mistake. Check lines: 112, 108. SyncedBag.cs 112

Вернёмся к игре

Неинициализированные переменные – **120** ошибок

vs

Повторная запись в переменную – **410** ошибок

А вот обычно нестрашный, но всё-таки ляп

- Проверка указателя после new
- В этот момент обязательно кто-то вспомнит про new(std::nothrow)
- Не отвлекайтесь, PVS-Studio это учитывает
- Пример из проекта Minetest (C++)

```
clouds = new Clouds(smgr, -1, time(0));
if (!clouds) {
    *error_message = "Memory allocation error (clouds)";
    errorstream << *error_message << std::endl;
    return false;
}
```

Проверка указателя после new

- В голову приходит:
  - Это только у студентов такое в коде
  - Это редкий вид программистов, до конца не перешедших с C на C++
- В общем – экзотика

Проверка указателя после new

- В голову приходит:
  - Это только у студентов такое в коде
  - Это редкий вид программистов, до конца не перешедших с C на C++
- ~~В общем – экзотика~~
- Ха-ха!

- В нашей коллекции **1630** таких вот «редкостей»

Напоследок моё любимое: (A == A)

- Никто всерьёз не воспринимает опечатки вида “**if (A == A)**”
- Я не видел такие паттерны в наборах тестов

- И действительно… Вот нулевые указатели, это да!
- 100500 тестов на указатели сделаем
- А таких простых ошибок не бывает

```
private boolean checkDimensions(CLIQUEUnit other, int e) {
    for (int i = 0, j = 0; i < e; i++, j += 2) {
        if (dims[i] != other.dims[i]
            || bounds[j] != other.bounds[j]
            || bounds[j + 1] != bounds[j + 1]) {
            return false;
        }
    }
    return true;
}
```

other.

ELKI, Java

V6001 There are identical sub-expressions 'bounds[j + 1]' to the left and to the right of the '!=' operator. CLIQUEUnit.java(252)

```
public static bool IsTypeOf<T>(this TypeReference tr)
{
    var type = typeof(T);
    return tr.Name == type.Name && tr.Namespace == tr.Namespace;
}
```

.NET 8, C#

V3001 There are identical sub-expressions 'tr.Namespace' to the left and to the right of the '==' operator. TypeReferenceExtensions.cs 365

GTK, C++

```
Value* op_color_number(enum Sass_OP op, ....)
{
  double rval = rhs.value();

  if ((op == Sass_OP::DIV || op == Sass_OP::DIV) && rval == 0) {
    // comparison of Fixnum with Float failed?
    throw Exception::ZeroDivisionError(lhs, rhs);
  }
  ....
}
```

V501 [CWE-570] There are identical sub-expressions to the left and to the right of the '||' operator: op == Sass_OP::DIV || op == Sass_OP::DIV operators.cpp 250

Они везде!

- **600** полновесных ошибок

Подытожим рассмотренное

Ощущениям доверять плохо

- Ожидания:
  - Неинициализированные переменные – 120
  - Деление на ноль – 20
  - Выход за границу массива – 215
  - Нулевые ссылки и указатели – 3900
- Реальность:
  - CWE-14 (исчезновение memset) – 330
  - Повторная запись в переменную – 410
  - Проверка указателя после new – 1630
  - Опечатки вида (A == A) – 600

Ощущениям доверять плохо

- Например, ожидания по указателям совпали
- А в других местах сильно мимо можно усилия приложить
- Что делать? Наблюдать!
- Практика направляет разработку анализатора в нужную сторону

Практика

- Начинаешь понимать, на каких паттернах следует сосредоточиться

- Иначе это как оптимизация без профилировщика
  - Объявить «анализ потока данных» важнейшим механизмом и делать всё на его основе
  - Искать **1ul** вместо **1UL**. Ни разу не видел ошибку…
  - Это отсылка к стандарту MISRA, где многие правила придуманы, на мой взгляд, исключительно умозрительно

А что ещё интересного?

Эффект последней строки

- Пример из проекта eShopOnContainers (C#)

```
private bool CheckSameOrigin(string urlHook, string url)
{
  var firstUrl = new Uri(urlHook, UriKind.Absolute);
  var secondUrl = new Uri(url, UriKind.Absolute);

  return firstUrl.Scheme == secondUrl.Scheme &&
         firstUrl.Port == secondUrl.Port &&
         firstUrl.Host == firstUrl.Host;
}
```

0,1,2,3 (в именах переменных; индексы)

- Пример из проекта OpenCOLLADA (C++)

```
struct short2
{
  short values[2];
  short2(short s1, short s2)
  {
    values[0] = s1;
    values[2] = s2;
  }
  ....
};
```

Функции сравнения

- Пример из проекта IronPython (C#)

```
public static int Compare(SourceLocation left,
                          SourceLocation right) {
  if (left < right) return -1;
  if (right > left) return 1;
  return 0;
}
```

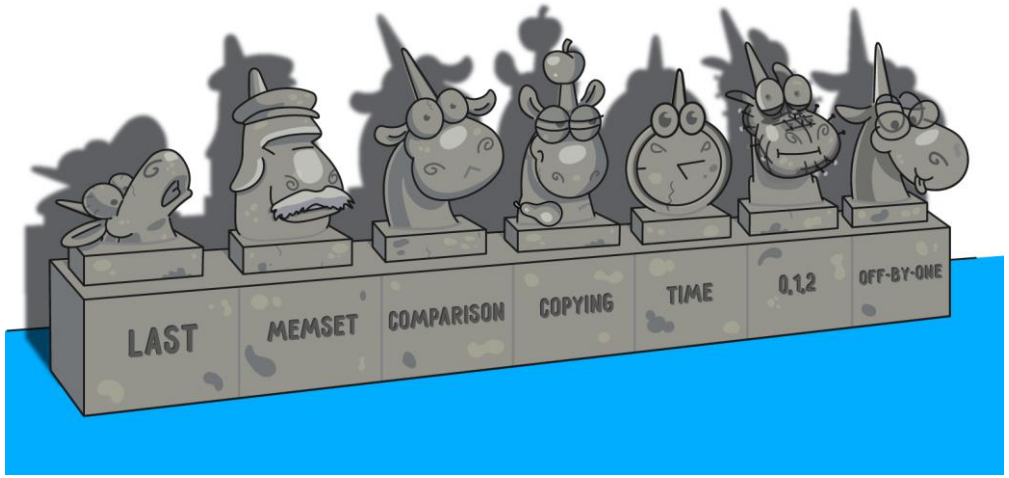
Функции времени

- Пример из проекта Tizen (C)

```
static void preview_down_cb(....)
{
  ....
  int delay = 0.5;
  double fdelay;
  fdelay = ((double)delay / 1000.0f);
  DbgPrint("Long press: %lf\n", fdelay);
  ....
}
```

Подробнее

- Распространённые паттерны опечаток при программировании



Выводы

Ошибки из книг – большой вопрос

- Хуже не будет
- Но увлекаться не стоит

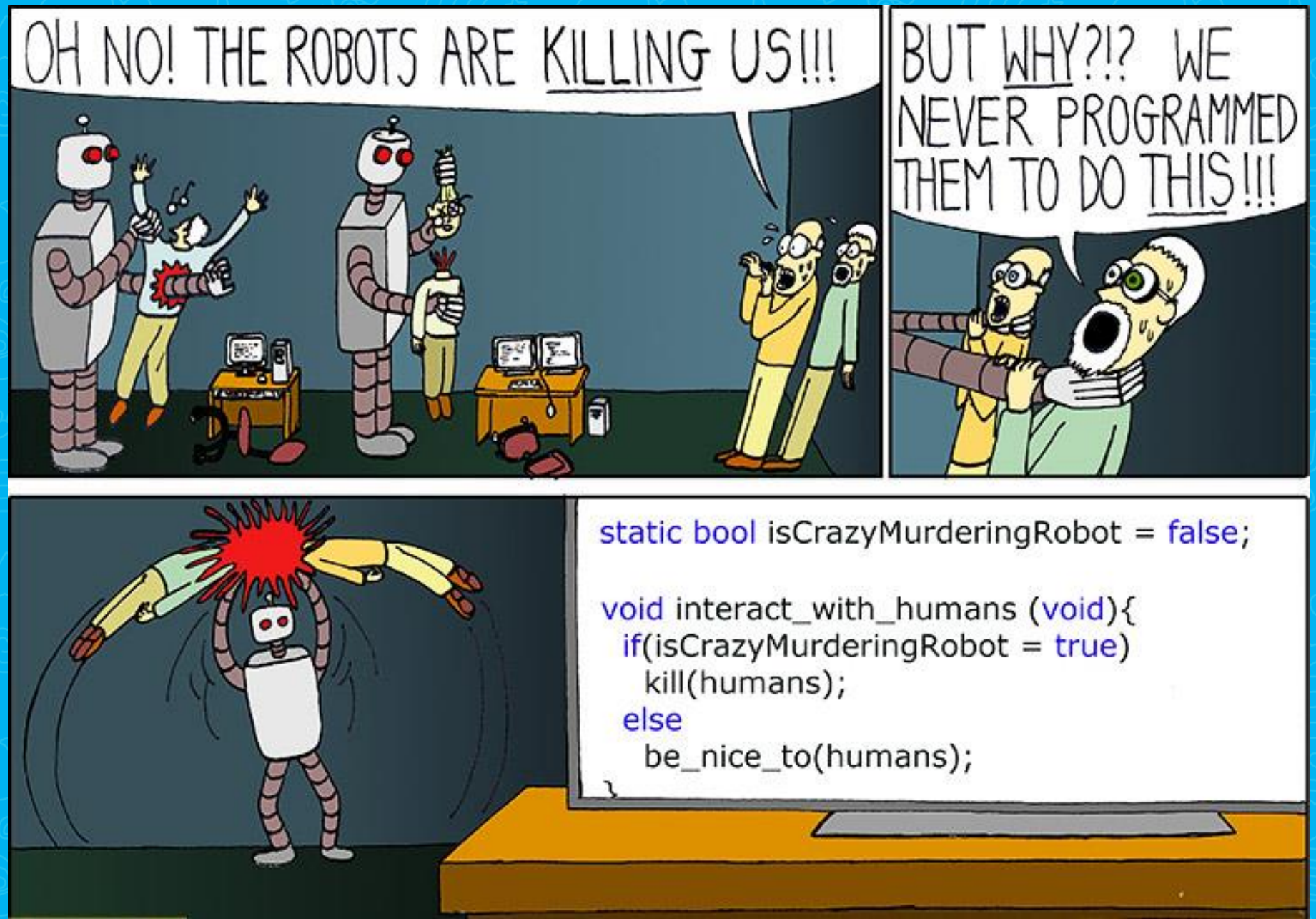
Следите за реальными ошибками и ищите их

- Свой код
- Сторонний код
- Код с ошибками, присланный пользователями анализатора

- Как показала практика, диагностики, созданные по реальным случаям, более полезны, чем придуманные теоретически

Бонус

Известный баг из мира C и C++: `if (isCrazyMurderingRobot=true)`



Такие ошибки мы действительно встречали

- Проект Intel AMT SDK (C++)

```
if(status = true)
{
  PrintSuccess();
}
```

Исчезающий вид! Внести в Красную книгу!

- В коллекции у нас собрано 14 подобных багов
- **Но последний раз мы находили такую ошибку в 2016 году!**

- Все компиляторы и анализаторы предупреждают

Q&A

Андрей Карпов
PVS-Studio, DevRel