Модельные варианты ошибок

Или как статические анализаторы находят ошибки,
которые не могут искать

Андрей Карпов
PVS-Studio, DevRel

Андрей Карпов

- Один из основателей PVS-Studio
- 16 лет в сфере анализа кода
- Пишу про качество C++ кода
- Хабр: @Andrey2008

Как появилась идея доклада

Сижу, читаю ГОСТ Р 71207–2024 — Статический анализ программного обеспечения

- Выписываю задачи по доработке PVS-Studio.
- Читаю про термин «модельный вариант ошибки».

Модельный вариант ошибки

На всякий случай определение из ГОСТ.
Не читайте!

- Подтип некоторого типа ошибки в программе, отнесение к которому осуществляется исходя из особенностей реализации ошибки данного типа и особенностей языка программирования.
- Разбиение типа ошибки на модельные варианты применяется из-за технологических ограничений статического анализа в целях эффективной работы статических анализаторов, сводя задачу поиска ошибки более общего типа к задаче поиска ошибок множества подтипов.

Оказывается, у «нельзя, но очень надо» есть название!

- Осознаю, что всю историю PVS-Studio мы занимаемся поиском **модельных вариантов ошибок**, только не знали, что у них название есть!

Так и появилась идея сделать доклад

- Я узнал, что у неких сущностей есть название.
- Хороший повод поговорить про эту интересную тему.

Статический анализ:
от простого к сложному

Где-то хватает сигнатурного анализа (Squidex, C#)

```
public Task EnhanceAsync(UploadAssetCommand command)
{
    var pw = file.Properties.PhotoWidth;
    var ph = file.Properties.PhotoHeight;

    if (pw > 0 && pw > 0)
    {
        command.Metadata.SetPixelWidth(pw);
        command.Metadata.SetPixelHeight(ph);
```

PVS-Studio: V3001 There are identical sub-expressions 'pw > 0' to the left and to the right of the '&&' operator. FileTagAssetMetadataSource.cs 80

Можно использовать статистику (Linux Kernel, C)

```
static const
struct XGI330_LCDDataDesStruct2  XGI_LVDSNoScalingDesData[] = {
  {0,  648,  448,  405,  96, 2}, /* 00 (320x200,320x400,
                                    640x200,640x400) */
  {0,  648,  448,  355,  96, 2}, /* 01 (320x350,640x350) */
  {0,  648,  448,  405,  96, 2}, /* 02 (360x400,720x400) */
  {0,  648,  448,  355,  96, 2}, /* 03 (720x350) */
  {0,  648,    1,  483,  96, 2}, /* 04 (640x480x60Hz) */
  {0,  840,  627,  600, 128, 4}, /* 05 (800x600x60Hz) */
  {0, 1048,  805,  770, 136, 6}, /* 06 (1024x768x60Hz) */
  {0, 1328,    0, 1025, 112, 3}, /* 07 (1280x1024x60Hz) */
  {0, 1438,    0, 1051, 112, 3}, /* 08 (1400x1050x60Hz)*/
  {0, 1664,    0, 1201, 192, 3}, /* 09 (1600x1200x60Hz) */
  {0, 1328,    0, 0771, 112, 6}  /* 0A (1280x768x60Hz) */
};
```

PVS-Studio: V536 Be advised that the utilized constant value is represented by an octal form. Oct: 0771, Dec: 505. vb_table.h 1379

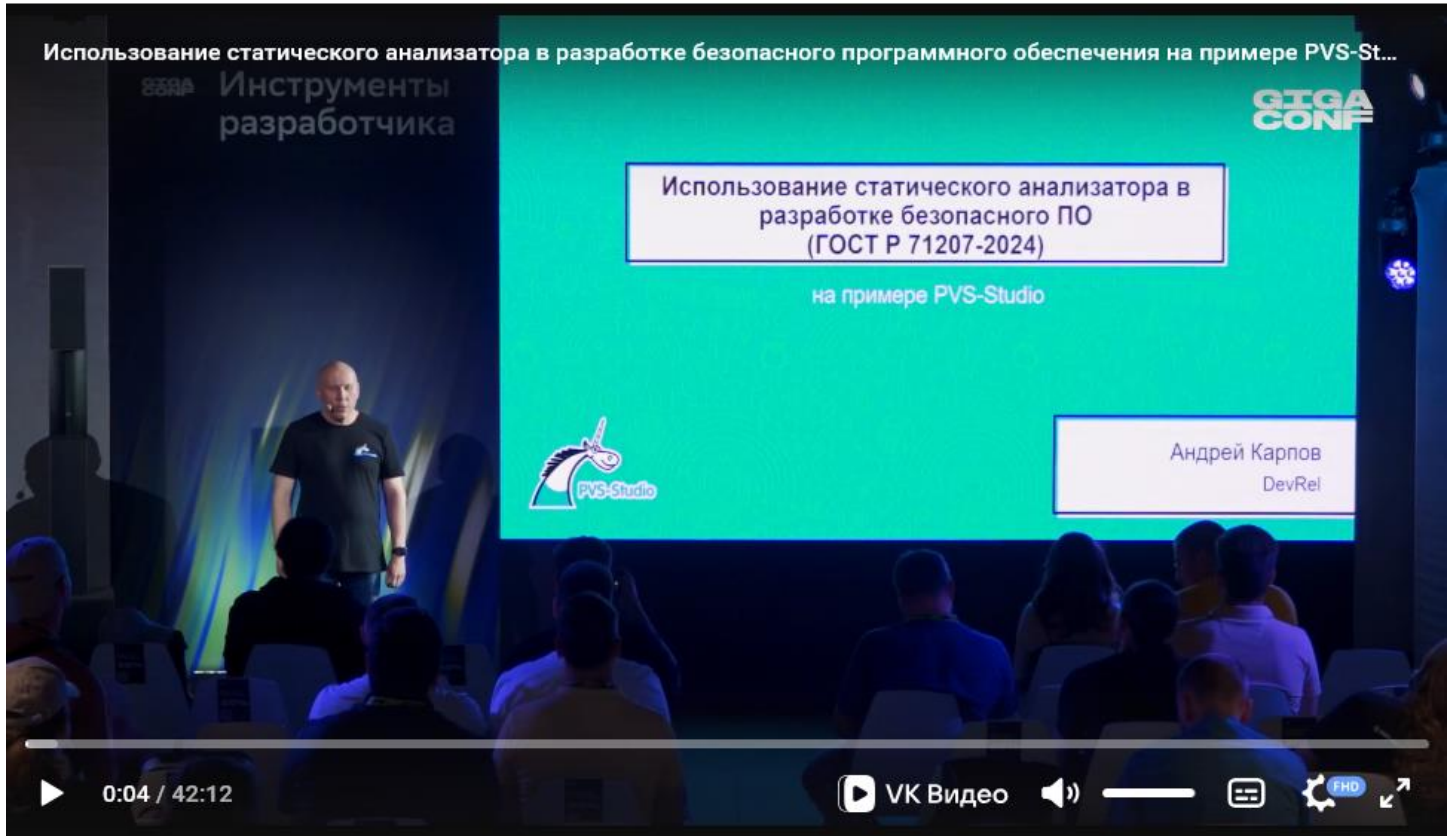
Но краеугольный камень — анализ потока данных

- Переполнение буфера.
- Ошибки управления динамической памятью.
- Разыменование нулевых указателей.
- Деление на 0.
- Ошибки при работе с многопоточными примитивами.
- Всегда истинные/ложные условия.
- И т.д.

Перечисленное почти совпадает со списком критических ошибок (ГОСТ Р 71207)

- Критические ошибки, поиск которых является первоочередной задачей статических анализаторов кода:
  - ошибки управления динамической памятью;
  - переполнение буфера;
  - разыменование нулевых указателей;
  - деление на 0;
  - ошибки при работе с многопоточными примитивами;
  - и т.д.

Подробнее про ГОСТ Р 71207-2024 и критические ошибки



Использование статического анализатора в разработке безопасного программного обеспечения (ГОСТ Р 71207-2024) на примере PVS-Studio

Ошибки управления динамической памятью (MuseScore, C++)

```
void GuitarPro6::readGpif(QByteArray* data)
{
  ....
  } else {
    delete slur;
    legatos[slur->track()] = 0;
  }
```

PVS-Studio: V774 The 'slur' pointer was used after the memory was released. importgtp-gp6.cpp 2592

Чуть посложнее (Augeas, C)

```
static void xfm_error(struct tree *xfm, const char *msg) {
  char *v = msg ? strdup(msg) : NULL;
  char *l = strdup("error");              // Выделение памяти

  if (l == NULL || v == NULL)
    return;                              // Утечка памяти
  tree_append(xfm, l, v);
}
```

PVS-Studio: V773 The function was exited without releasing the 'l' pointer. A memory leak is possible. transform.c 709

Переполнение буфера (EFL Core Libraries, C)

```
static Eina_Bool _convert_etc2_rgb8_to_argb8888(....)
{
  ....
  int out_step, x, y, k;
  unsigned int bgra[16];
  ....
  for (k = 0; k < 4; k++)
    memcpy(out + x + k * out_step, bgra + k * 16, 16);
```

PVS-Studio: V512 A call of the 'memcpy' function will lead to overflow of the buffer 'bgra + k * 16'. draw_convert.c 318

Красота, да и только

- Вычисляй и находи ошибки!

Дьявол кроется в росте сложности

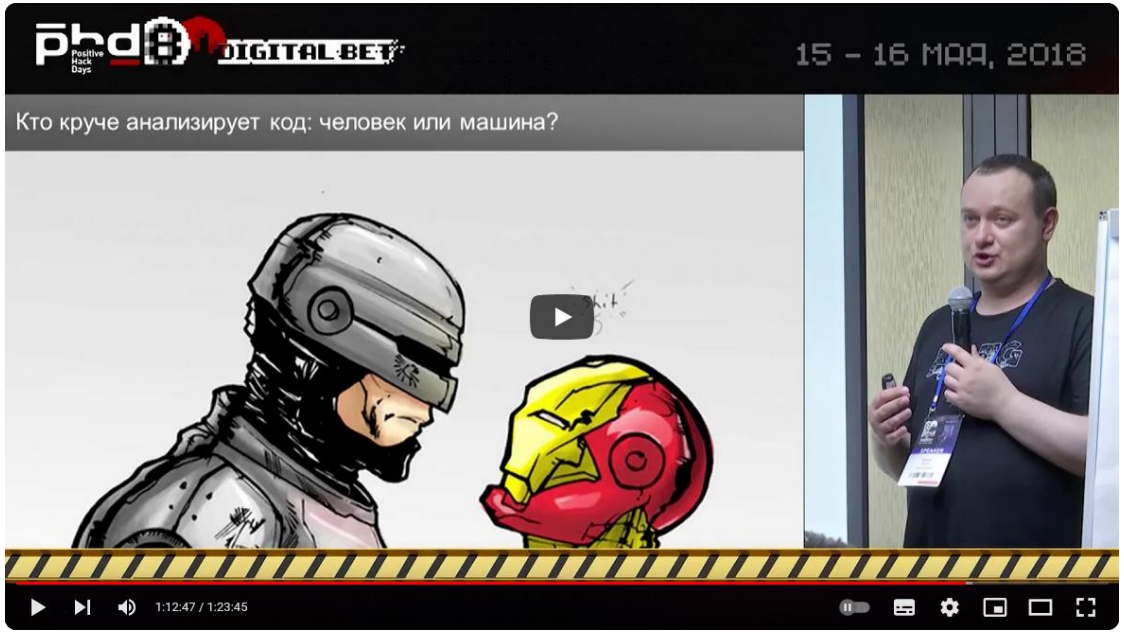
Здравствуйте, теоретические ограничения

- **Теорема Райса** — утверждение теории алгоритмов, согласно которому для любого нетривиального свойства вычислимых функций определение того, вычисляет ли произвольный алгоритм функцию с таким свойством, является алгоритмически неразрешимой задачей.

- **Проблема остановки** — одна из проблем в теории алгоритмов, которая может неформально быть поставлена в виде:
  - даны описание процедуры и её начальные входные данные;
  - требуется определить: завершится ли когда-либо выполнение процедуры с этими данными либо процедура будет работать без остановки.

Больше теории

- Владимир Кочетков. Идеальный статический анализ. https://youtu.be/RdeElwDJ3tg?si=lcO9jjiCA0grGXS2



Немного отвлечёмся: минутка компиляторного юмора

- Великая теорема Ферма́

Теорема утверждает[1], что для любого натурального числа $n > 2$ уравнение

$$a^n + b^n = c^n$$

не имеет решений в целых ненулевых числах a, b, c.

Проблема останова в контексте C++

- Стандарт C++ зачем-то считает, что валидная программа должна:
  - либо гарантированно завершаться;
  - либо гарантированно производить обозреваемые эффекты: запрашивать ввод-вывод, взаимодействовать с volatile-переменными и т.п.
- А иначе поведение программы — неопределённое.

- Рассуждения на эту тему:
  P2809R0
  Trivial infinite loops are not Undefined Behavior
  Published Proposal, 2023-03-14

Опровергнем теорему Ферма́

- «Правильные» компиляторы C++ настолько суровы, что способны решать алгоритмически неразрешимые задачи!

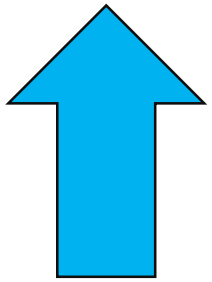
- Если в программе есть бесконечный цикл

  и компилятор решил, что этот цикл не имеет обозреваемых эффектов,

  то цикл не имеет смысла и может быть схлопнут.

```
int fermat () {
  const int MAX = 1000;
  int a=1,b=1,c=1;
  while (1) { - Бесконечный цикл
    if ( (a*a*a) == (b*b*b) + (c*c*c) ) return 1; - Первая точка выхода
    a++;
    if (a>MAX) {
      a=1;
      b++;
    }
    if (b>MAX) {
      b=1;
      c++;
    }
    if (c>MAX) {
      c=1;
    }
  }
  return 0; - Вторая точка выхода НЕДОСТИЖИМА!
}
```

```
int fermat () {
  const int MAX = 1000;
  int a=1,b=1,c=1;
  while (1) {
    if ( (a*a*a) == (b*b*b) + (c*c*c) ) return 1;
    a++;
    if (a>MAX) {
      a=1;
      b++;
    }
    if (b>MAX) {
      b=1;
      c++;
    }
    if (c>MAX) {
      c=1;
    }
  }
  return 0;
}
```



- Единственный выход из бесконечного цикла.
- У цикла нет никаких видимых эффектов.
- Компилятор просто заменил тело функции на return 1.

Подробнее

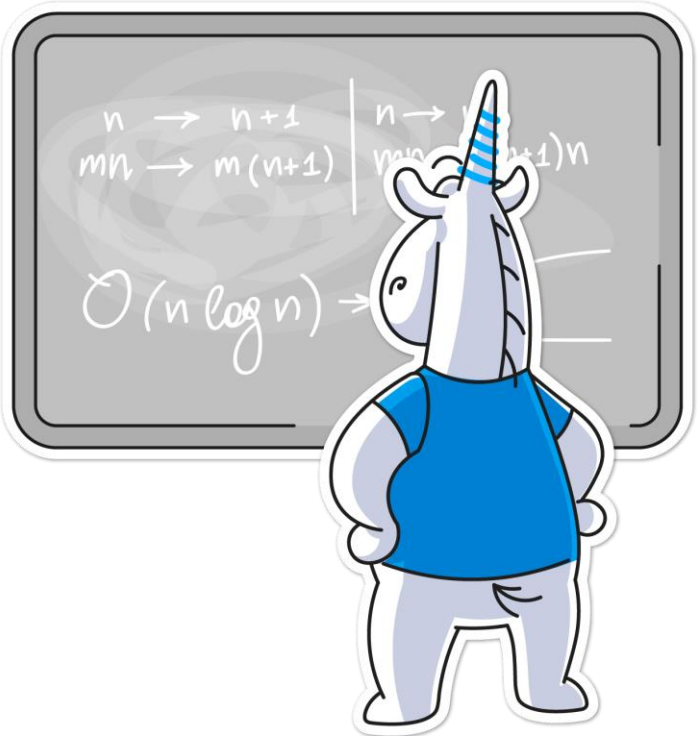
- Дмитрий Свиридкин.

  Путеводитель C++ программиста по неопределённому поведению.
- Глава «Бесконечные циклы и проблема останова».

На практике до теоретических ограничений даже не дойдёт

- Нас остановит рост вычислительной сложности:
  - требуется слишком много памяти;
  - требуется слишком много времени.

- Рассмотрим ещё не сложный, но уже сложный пример :)



Анализ потока данных (mc, C)

```c
// Файл: widget-common.c
// Освобождение буфера памяти

void widget_destroy (Widget * w)
{
  send_message (w, NULL, MSG_DESTROY, 0, NULL);
  g_free (w);
}
```

Другой файл

```
// Файл: editcmd.c
// Использование указателя после освобождения памяти
gboolean edit_close_cmd (WEdit * edit)
{
  Widget *w = WIDGET (edit);
  ....
  widget_destroy (w);      // <= Здесь освободили память
  if (....) .... else
  {
    edit = find_editor (DIALOG (g));
    if (edit != NULL)
      widget_select (w);  // <= Сейчас заглянем внутрь
  }
}
```

Идём дальше по цепочке вызова

```
void
widget_select (Widget * w)
{
  WGroup *g;
  if (!widget_get_options (w, WOP_SELECTABLE))
    return;
  ....
}
```

Молодцы, добрались до ошибки

```
// Добрались до использования уже разрушенной структуры

static inline gboolean
widget_get_options (const Widget * w, widget_options_t options)
{
  return ((w->options & options) == options);
}
```

PVS-Studio: V774 The 'w' pointer was used after the memory was released. editcmd.c 2258

Итог

- Ряд задач неразрешим даже теоретически.
- Где задачи разрешимы, есть ресурсные ограничения.
- Сложно реализовать нереализуемое.

- Ругаться, где мы что-то не поняли – плохой вариант:
  - будет слишком много ложных срабатываний;
  - в них утонет полезное.

Какие ошибки нельзя найти?

В общем виде — все критические ошибки

- Ошибки управления динамической памятью – **ГОСТ Р 71207: критические ошибки**
- Переполнение буфера – **ГОСТ Р 71207: критические ошибки**
- Разыменование нулевых указателей – **ГОСТ Р 71207: критические ошибки**
- Деление на 0 – **ГОСТ Р 71207: критические ошибки**
- Ошибки при работе с многопоточными примитивами – **ГОСТ Р 71207: критические ошибки**
- И т.д.

А ещё невозможно искать

- Неопределённое поведение.
- Опечатки.

Время магии!

Вернее, магией мы раньше занимались

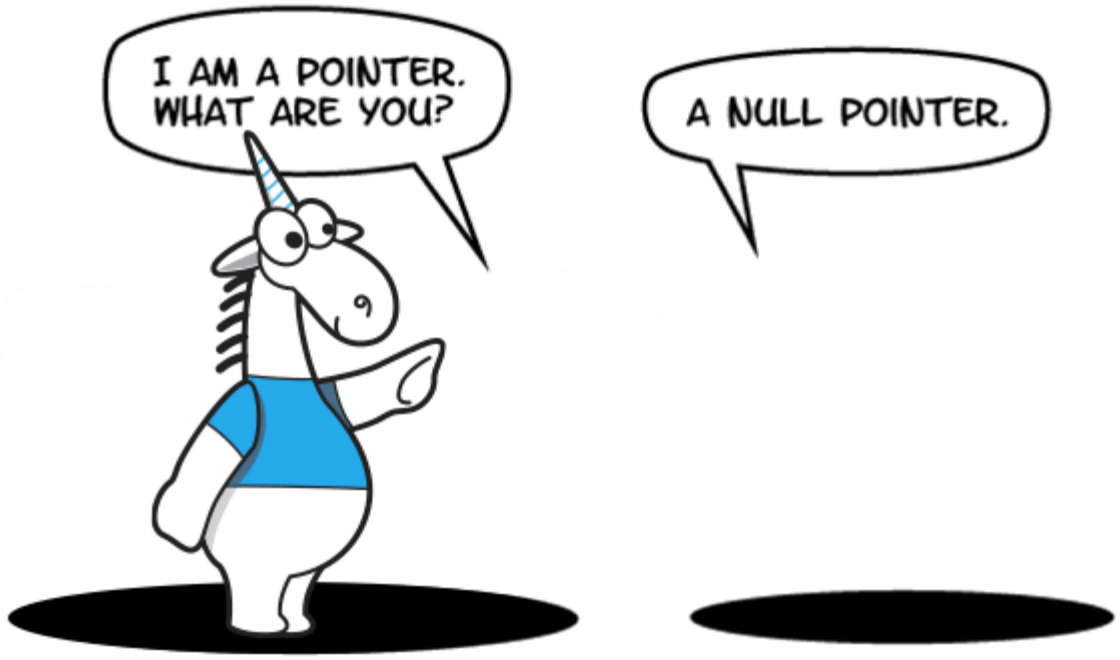
- **Теперь время модельных ошибок!**

- Давайте посмотрим, как разными эвристиками можно искать невозможное.

Разыменование нулевых указателей/ссылок

Разыменование нулевых указателей/ссылок

- В общем виде отследить значение невозможно.
- Что же можно придумать?



Использование указателя до проверки (GTK, C)

```
static gboolean
on_flash_timeout (GtkInspectorWindow *iw)
{
  iw->flash_count++;
  gtk_highlight_overlay_set_color(
    GTK_HIGHLIGHT_OVERLAY (iw->flash_overlay),
    &(GdkRGBA) { 0.0, 0.0, 1.0,
                 (iw && iw->flash_count % 2 == 0) ? 0.0 : 0.2
               });
  ....
```

PVS-Studio: V595 The 'iw' pointer was utilized before it was verified against nullptr. Check lines: 194, 199. inspect-button.c 194

Разновидность (FreeCAD, C++)

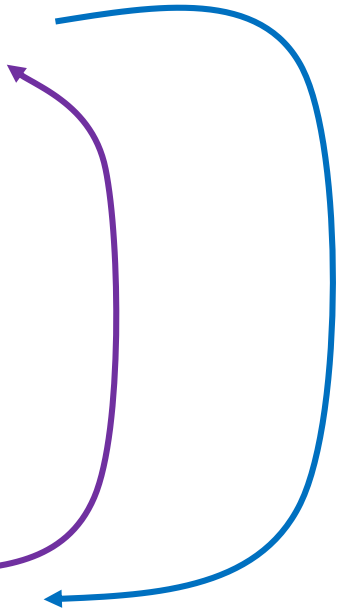
```
TaskTransformedParameters::TaskTransformedParameters(
  ViewProviderTransformed *TransformedView, QWidget *parent)
    : TaskBox(...., TransformedView->menuName, true, parent)
    , proxy(nullptr)
    , ....
{
  selectionMode = none;
  if (TransformedView) {
     Gui::Document* doc = TransformedView->getDocument();
     this->attachDocument(doc);
  }
  ....
}
```

PVS-Studio: V664 The 'TransformedView' pointer is being dereferenced on the initialization list before it is verified against null inside the body of the constructor function. Check lines: 57, 66. TaskTransformedParameters.cpp 57

Такой подход не отменяет анализ потока данных (синтетический пример, C++)

```
void UsePtr(int *p)
{
    *p = 1;
}

void foo(int *ptr)
{
    UsePtr(ptr);
    if (!ptr)
        std::cout << "error";
}
```



Информация, что указатель использовался без проверки.

PVS-Studio: V595 The 'ptr' pointer was utilized before it was verified against nullptr. Check lines: 8, 13, 14.

Использование ссылки до проверки Discord.NET (C#)

```
internal void Update(ClientState state, Model model)
{
  var roles =
        new ConcurrentDictionary<ulong, SocketRole>
            (ConcurrentHashSet.DefaultConcurrencyLevel,
            (int)(model.Roles.Length * 1.05));
  if (model.Roles != null)
  {
    for (int i = 0; i < model.Roles.Length; i++)
  ....
```

PVS-Studio: V3095 The 'model.Roles' object was used before it was verified against null. Check lines: 534, 535. SocketGuild.cs 534

Подход кажется неубедительным костылём?

- Ха!
- При написании статей нашей командой найдено ошибок в открытых проектах:
  - C, C++: 1840
  - C#: 166
  - Java: 24
- **Эвристика рулит!**

А если наоборот: сначала проверка, потом использование?

```
void Queries::popIfTop(QueryPtr query)
{
  if(!query)
    logGlobal->error("The query is nullptr! Ignoring.");
  popIfTop(*query);
}
```

PVS-Studio: V1004 The 'query' pointer was used unsafely after it was verified against nullptr. Check lines: 246, 249. CQuery.cpp 249

Это опять анализ потока данных.
Запоминаем, что указатель может быть нулевым, и затем используем эту информацию.

Хорошо, а ещё?

- Анализ потока данных – это сложно.
- Вдвойне сложно, когда меняются типы объектов с помощью dynamic_cast.

- Зато мы знаем, что люди делают опечатки, проверяя не тот указатель, когда используют dynamic_cast.

Проверили не тот указатель (MuditaOS, C++)

```
void CallLogDetailsWindow::onBeforeShow(...., SwitchData *data)
{
  ....
  if (auto switchData =
        dynamic_cast<calllog::CallLogSwitchData *>(data);
      data != nullptr)
  {
    record = switchData->getRecord();
    ....
```

PVS-Studio: V757 It is possible that an incorrect variable is compared with nullptr after type conversion using 'dynamic_cast'. Check lines: 214, 214. CallLogDetailsWindow.cpp 214

Примечание

- На самом деле мы ругаемся и на другие разыменования указателей, которые вернул оператор dynamic_cast, если нет проверки.
- Но паттерн, рассмотренный ранее, более чёткий и достоверный.

Что интересно, в C# это ещё более частый паттерн (Media Portal 2, C#)

```
public override bool Equals(object obj)
{
  EpisodeInfo other = obj as EpisodeInfo;
  if (obj == null) return false;
  if (TvdbId > 0 && other.TvdbId > 0)
    return TvdbId == other.TvdbId;
  ....
}
```

PVS-Studio: V3019 Possibly an incorrect variable is compared to null after type conversion using 'as' keyword. Check variables 'obj', 'other'. EpisodeInfo.cs 560

Хорошо, а ещё?

- Часто исходный код внешних библиотек недоступен, и анализатор не может сам узнать, корректно туда передать null или нет.

- Аннотирование внешних функций в известных библиотеках!
- Большая невидимая работа.

Шок, но это реальная опечатка (LLVM, C++)

```
OwningMemRef &operator=(const OwningMemRef &&other) {
  freeFunc = other.freeFunc;
  descriptor = other.descriptor;
  other.freeFunc = nullptr;
  memset(0, &other.descriptor, sizeof(other.descriptor));
}
```

PVS-Studio: V575 The null pointer is passed into 'memset' function. Inspect the first argument. MemRefUtils.h 194

На предварительном прогоне не поверили и попросили PROOF

[MLIR][NFC] Fix a memset in MemRefUtils.h

[MLIR][NFC] Fix a memset in MemRefUtils.h

Browse files

```
found by PVS-Studio - https://pvs-studio.com/en/blog/posts/cpp/1003/, N10.
memset function expects to take int as the second actual argument,
but receives a pointer. Here, the first and the second argument of
the function are mixed up.

Reviewed By: ftynse

Differential Revision: https://reviews.llvm.org/D142310
```

main

llvmorg-20-init ... llvmorg-16.0.0-rc1

xgupta committed on Jan 23, 2023 — 1 parent bee8860 commit 586ce6a

Showing **1 changed file** with **1 addition** and **1 deletion**. Whitespace | Ignore whitespace | Split | Unified

2 mlir/include/mlir/ExecutionEngine/MemRefUtils.h

```
@@ -191,7 +191,7 @@ class OwningMemRef {
191     freeFunc = other.freeFunc;                                  191     freeFunc = other.freeFunc;
192     descriptor = other.descriptor;                              192     descriptor = other.descriptor;
193     other.freeFunc = nullptr;                                   193     other.freeFunc = nullptr;
194 -   memset(0, &other.descriptor, sizeof(other.descriptor));     194 +   memset(&other.descriptor, 0, sizeof(other.descriptor));
195   }                                                             195   }
196   OwningMemRef(OwningMemRef &&other) { *this = std::move(other); }  196   OwningMemRef(OwningMemRef &&other) { *this = std::move(other); }
197                                                               197
```

Переполнение знаковых типов в C и C++

Переполнение знаковых типов в C и C++

- Это частный случай неопределённого поведения, как и разыменование нулевого указателя.
- Но если так подходить, то почти всё к UB можно свести! ☺

- Если предупреждать про все арифметические операции, когда мы не знаем значения переменных, то пользоваться анализатором будет нельзя.
- Путь в никуда.
- Пойдя по нему, можно просто сразу печатать:
  “У вас программа на C++, это подозрительно”.

Значит, на этом всё?

Эвристика: не там приведение типа (libtorrent, C++)

```
void torrent::get_download_queue(....) const
{
  ....
  const int blocks_per_piece = m_picker->blocks_in_piece(....);
  int counter = 0;
  for (auto i = q.begin(); i != q.end(); ++i, ++counter)
  {
    ....
    pi.blocks = &blk[std::size_t(counter * blocks_per_piece)];
```

PVS-Studio: V1028 Possible overflow. Consider casting operands of the 'counter * blocks_per_piece' operator to the 'size_t' type, not the result. torrent.cpp 7092

Эвристика: возможный подсчёт с переполнением

```
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.

Опечатка в константах

Невозможно

- Откуда мы знаем, какую константу хотел записать человек?
- Не знаем…

- Но раз очень хочется, давайте что-то попробуем!

Ошибка в Пи (Bullet Physics SDK, C++)

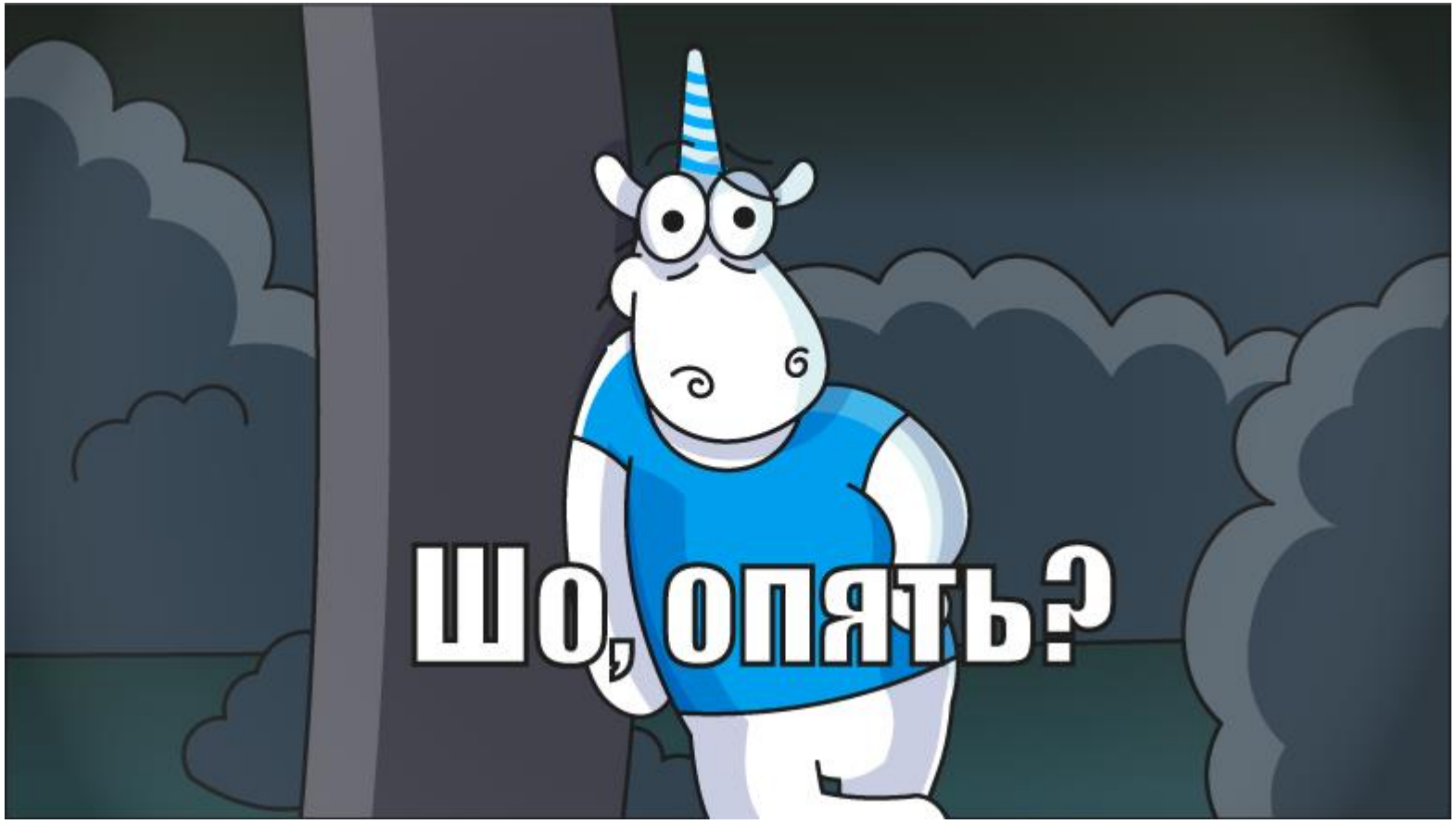
```
void b3ComputeProjectionMatrixFOV(float fov, ....)
{
  float yScale = 1.0 / tan((3.141592538 / 180.0) * fov / 2);
```

V624 There is probably a misprint in '3.141592538' constant. Consider using the M_PI constant from <math.h>. PhysicsClientC_API.cpp 4109

```
Число в коде: 3.141592 53 8
Настоящее Пи: 3.14159265358
```

Опечатки в целом

Невозможно



Но можно рассмотреть множество частных случаев

- Получается всегда истинное/ложное условие.
- Аномалии в вызовах функций.
- Странное сравнение с NULL.
- И т.д.

- Поиск опечаток вовсе не означает, что эти диагностики только для проверки лабораторных работ студентов :)
- **Они выявляют и проблемы безопасности!**

Всегда истинное условие (RavenDB, C#)

```
public override void VerifyCanExecuteCommand(....)
{
  ....
  var definition = JsonDeserializationServer.CertificateDefinition(read);
  if (
    definition.SecurityClearance != SecurityClearance.ClusterAdmin ||
    definition.SecurityClearance != SecurityClearance.ClusterNode
  )
    return;

  AssertClusterAdmin(isClusterAdmin);
}
```

PVS-Studio: V3022 Expression is always true. Probably the '&&' operator should be used here. DeleteCertificateFromClusterCommand.cs(21) Raven.Server

Всегда ложное условие (ONLYOFFICE, C#)

```
public void SetCredentials(string userName,
                           string password, string domain)
{
  if (string.IsNullOrEmpty(userName))
  {
     throw new ArgumentException("Empty user name.", "userName");
  }
  if (string.IsNullOrEmpty("password"))
  {
    throw new ArgumentException("Empty password.", "password");
  }
  CredentialsUserName = userName;
  CredentialsUserPassword = password;
```

PVS-Studio: V3022 Expression 'string.IsNullOrEmpty("password")' is always false. SmtpSettings.cs 104

Аномалии в вызове функции (NSS, C)

```
static SECStatus ssl3_SendEncryptedExtensions(sslSocket *ss)
{
  static const unsigned char P256_SPKI_PREFIX[] = {
    0x30, 0x59, 0x30, 0x13, 0x06, 0x07, 0x2a, 0x86,
    0x48, 0xce, 0x3d, 0x02, 0x01, 0x06, 0x08, 0x2a,
    0x86, 0x48, 0xce, 0x3d, 0x03, 0x01, 0x07, 0x03,
    0x42, 0x00, 0x04
  };
  ....
  if (.... || memcmp(spki->data, P256_SPKI_PREFIX,
                     sizeof(P256_SPKI_PREFIX) != 0))
```

PVS-Studio: V526 The 'memcmp' function returns 0 if corresponding buffers are equal. Consider examining the condition for mistakes. ssl3con.c 10533

Странное сравнение с NULL (Ultimate TCP/IP, C++)

```
char *CUT_CramMd5::GetClientResponse(LPCSTR ServerChallenge)
{
  ....
  if (m_szPassword != NULL)
  {
    ....
    if (m_szPassword != '\0')
    {
  ....
```

PVS-Studio: V528 It is odd that pointer to 'char' type is compared with the '\0' value. Probably meant: *m_szPassword != '\0'. UTMail ut_crammd5.cpp 333

Ошибки при работе с многопоточными примитивами

Даже ГОСТ Р 71207-2024 соглашается, что это сложно

- Методы анализа должны обеспечивать поиск критических ошибок …
- Требуемые типы критических ошибок …
- Кроме приведённых в перечислении **д)** …
  - **д)** ошибки при работе с многопоточными примитивами

- У Intel в своё время анализатор параллельных программ не взлетел.

- Остаётся искать ошибки «под фонарём».

Например, искать небезопасные double-checked locking (WildFly, Java)

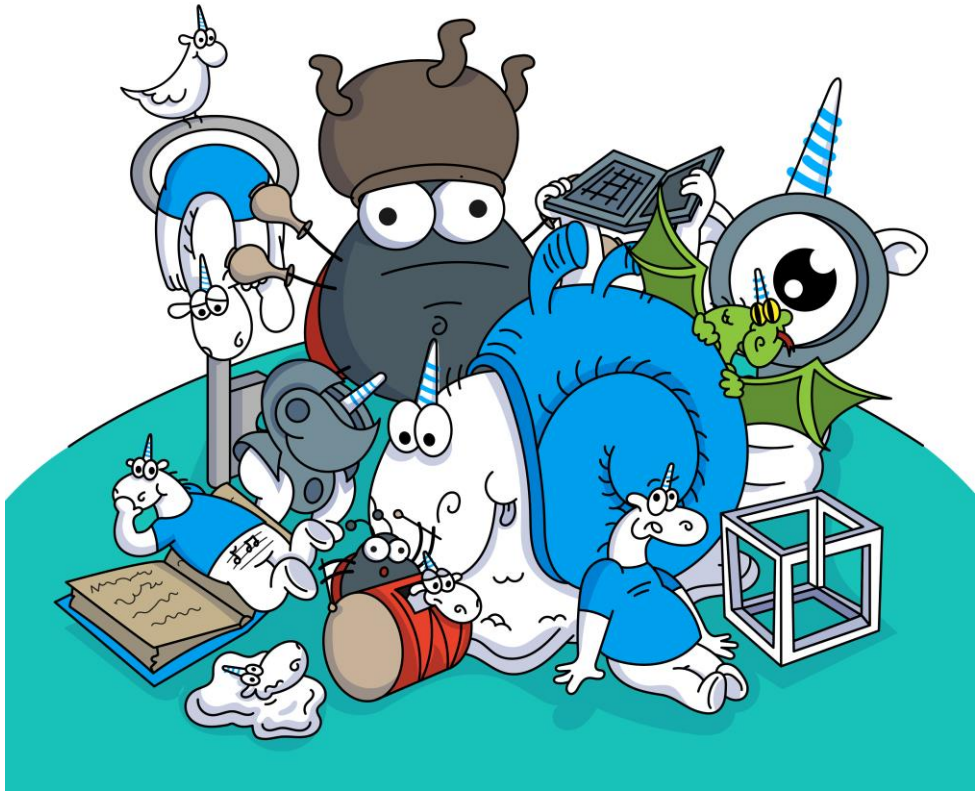
```
private volatile ExpressionFactory factory;
....
@Override
public ExpressionFactory getExpressionFactory() {
  if (factory == null) {
    synchronized (this) {
      if (factory == null) {
        factory = delegate.getExpressionFactory();
        for (ExpressionFactoryWrapper wrapper : wrapperList) {
          factory = wrapper.wrap(factory, servletContext);
        }
      }
    }
  }
  return factory;
}
```

PVS-Studio: V6082 Unsafe double-checked locking. A previously assigned object may be replaced by another object.

Неопределённое поведение

Невозможно

- Слишком разное.
- Никто и не спорит.



#Cpp #Knowledge 23 Авг 2024

Путеводитель C++ программиста по неопределённому поведению: часть 4 из 11

Андрей Карпов, Дмитрий Свиридкин

Вашему вниманию предлагается четвёртая часть электронной книги, которая посвящена неопределённому поведению. Книга не является учебным пособием и рассчитана на тех, кто уже хорошо знаком...

...

#Cpp #Knowledge 05 Авг 2024

Путеводитель C++ программиста по неопределённому поведению: часть 3 из 11

Андрей Карпов, Дмитрий Свиридкин

Вашему вниманию предлагается третья часть электронной книги, которая посвящена неопределённому поведению. Книга не является учебным пособием и рассчитана на тех, кто уже хорошо знаком...

...

#Cpp #Knowledge 24 Июн 2024

Путеводитель C++ программиста по неопределённому поведению: часть 2 из 11

Андрей Карпов, Дмитрий Свиридкин

Вашему вниманию предлагается вторая часть электронной книги, которая посвящена неопределённому поведению. Книга не является учебным пособием и рассчитана на тех, кто уже хорошо знаком...

...

#Cpp #Knowledge 07 Июн 2024

Путеводитель C++ программиста по неопределённому поведению: часть 1 из 11

Андрей Карпов, Дмитрий Свиридкин

Вашему вниманию предлагается первая часть электронной книги, которая посвящена неопределённому поведению. Книга не является учебным пособием и рассчитана на тех, кто уже хорошо знаком...

...

Что про LLM?

Однажды мы сделали неудачную диагностику V525

- Идея была красивая: эмпирическое/статистическое выявление опечаток в однотипных блоках.
- Проблема: очень много ложных срабатываний.
- Это самая большая диагностика по количеству строк кода.
- В ней больше всего исключений.

- Но всё равно много шума.

```
static int rr_cmp(uchar *a,uchar *b)
{
  if (a[0] != b[0])
    return (int) a[0] - (int) b[0];
  if (a[1] != b[1])
    return (int) a[1] - (int) b[1];
  if (a[2] != b[2])
    return (int) a[2] - (int) b[2];
  if (a[3] != b[3])
    return (int) a[3] - (int) b[3];
  if (a[4] != b[4])
    return (int) a[4] - (int) b[4];
  if (a[5] != b[5])
    return (int) a[1] - (int) b[5];
  if (a[6] != b[6])
    return (int) a[6] - (int) b[6];
  return (int) a[7] - (int) b[7];
}
```

PVS-Studio V525

MySQL, C++

Кажется, можно попробовать здесь AI


НО ЭТО СОВСЕМ
НТВ
ДРУГАЯ ИСТОРИЯ

Заключение

Как-то это всё неубедительно

- Множество отдельных эвристик не решают проблему поиска ошибок в целом.
- Но в процессе написания статей мы нашли 16000 реальных ошибок.
- Думаю, половина из них благодаря эвристикам!

- Аналогия: системы безопасности в автомобиле.

Q&A

Андрей Карпов
PVS-Studio, DevRel