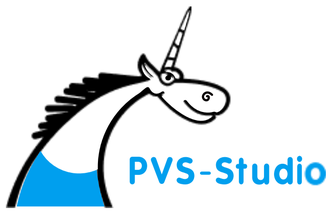
MISRA

Альтернативный взгляд на разработку безопасного ПО



Андрей Карпов

CBDO, ООО «ПВС»

Андрей Карпов

Директор по развитию бизнеса (CBDO)

- Один из основателей проекта PVS-Studio https://pvs-studio.ru
- 18 лет в сфере качества и анализа кода
- Хабр: @Andrey2008

- Telegram канал «Бестиарий программирования» t.me/programming_tales

Email: karpov@viva64.com

PVS-Studio

Стандарты/регламенты написания кода

- Набор правил и соглашений, используемых при написании исходного кода
- Наличие общего стиля облегчает понимание и поддержание исходного кода, написанного несколькими программистами
- Запрет использования некорректных и опасных конструкций

- Примеры: isocpp.org/wiki/faq/coding-standards

Статические анализаторы кода

- Контроль многих аспектов стандартов кодирования можно автоматизировать
- Это делают статические анализаторы кода

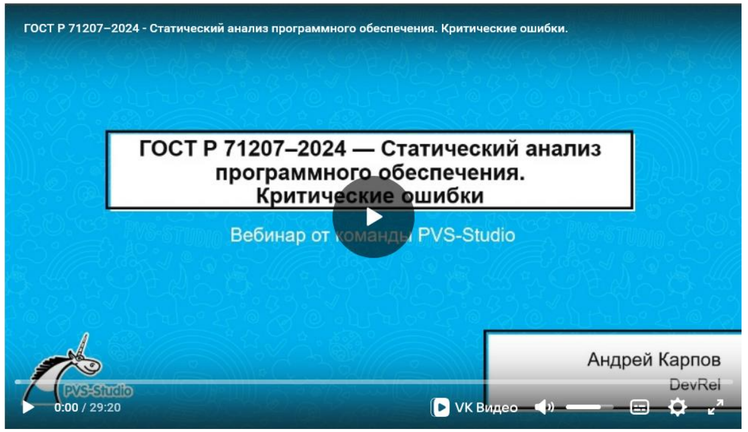
- Они ищут как «код с запахом», так и полновесные ошибки:
  - выход за границу массива
  - разыменовывание нулевых указателей/ссылок
  - неопределённое поведение
  - опечатки, меняющие логику работы программы
  - и т.д.

С точки зрения РБПО приоритетнее искать критические ошибки

- **ГОСТ Р 71207-2024 — Статический анализ программного обеспечения** вводит термин: **критические ошибки**
- Это ошибки, которые с наибольшей вероятностью приводят к уязвимостям ПО
  - Если ошибка критическая, это не значит, что она приводит к чему-то плохому
  - Если ошибка не критическая, это не значит, что она не может оказаться уязвимостью
  - Надо править все ошибки. Суть — куда обратить больше внимания и с чего начинать

ГОСТ Р 71207-2024: критические ошибки

- Ошибки управления динамической памятью
- Ошибки использования форматной строки
- Ошибки использования неинициализированных переменных
- Ошибки непроверенного использования чувствительных данных
- И т.д.



Вебинар про критические ошибки

pvs-studio.ru/ru/blog/video/11084/

Но есть и альтернативный подход

MISRA C
MISRA C++

MISRA C и MISRA C++

- Это больше про ограничение сложности
- В основном за счёт ограничений на использование конструкций языка, библиотек, подходов программирования
- Последние версии:
  - MISRA-C – 2025
  - MISRA-C++ – 2023

Познакомившись с MISRA в 2016 году, мы решили, что это не наша тема

- Из статьи «Краткий ответ про MISRA» (2016)

- Мы ориентированы на поиск уже присутствующих в коде ошибок, а не на предотвращение потенциальных проблем ценой ограничения программистов.

- Например, мы не заставляем разработчика обязательно делать ветку default в switch.

Старое видение

- Мы считаем малополезной рекомендацию не писать комментарии вида /* */, а использовать //.

- Подобные диагностики очень быстро "замусоривают" вывод анализатора, и вместо поиска настоящих ошибок человек начинает бороться с тысячами хоть и теоретически хороших, но ни на что не влияющих предложений по улучшению кода.

Но главное: про MISRA детекторы сложно писать статьи для продвижения

- Нам виделось,
  что **статьи буду выглядеть так** →

- Можно писать статьи про неинициализированные переменные, найденные в открытом проекте
- Но нельзя написать статью про «**ужас, они используют goto!**»

Поэтому долгое время в PVS-Studio

- Мы искали критические ошибки
  (только называли их потенциальными уязвимостями)
- Искали опечатки
- Искали частные случаи неопределённого поведения
  (в обобщенном виде это задача не решается)
- И писали про них статьи
  pvs-studio.ru/ru/blog/inspections/

Но MISRA – это другое

Это не только поиск проблем,
но и их предупреждение!

MISRA C/C++ тоже служит целям РБПО

- Про аналогию с языком ADA

- Код будет легче читаться и поддерживаться
- Код не будет содержать редких и нестандартных конструктов
- Снижается цикломатическая сложность
- Код более переносимый

Устраняет нестыковку сущностей

- Если функция не возвращает управление, то странно, что она возвращает значение
- Неправильно:

```
_Noreturn int sum(int a, int b)
{
  ....
  return a + b;
}
```

Меньше вероятность неправильного понимания кода

- Последовательности восьмеричных и шестнадцатеричных чисел внутри строковых и символьных литералов должны быть завершёнными

```
const char* str = "\x0exit";       // Неправильно

const char* str1 = "\x0" "exit";   // Правильно
```

Ограничения для безопасности

- Нельзя использовать variable-length array (VLA)
- Опасность переполнения стека и потенциально более уязвимый код
- Неправильно:

```
void foo(size_t n)
{
  int arr[n];
  // ....
}
```

Уменьшение сложности

- Использование оператора goto, осуществляющего переход к метке, находящейся выше по коду, ухудшает читаемость кода и, как следствие, усложняет его поддержку
- Не получится создать цикл с помощью goto
- Неправильно:

```
void init(....) {
again:
   ....
   if (....)
     goto again;
   ....
}
```

Что важно понимать и помнить про MISRA

MISRA – для микроконтроллеров!

- Попытка использовать MISRA правила в мире прикладного ПО вызовет недоумение и неудобства
- Многие рекомендации будут бесполезны или вредны

- Пример: предупреждение про использование функций и операторов выделения памяти (malloc, realloc, new, …)

Будет крайне трудоёмко адаптировать проект под MISRA

- MISRA предполагает, что вы сразу пишите код с учётом требований стандарта

Не все правила MISRA могу выявляться анализаторами кода

- Некоторые правила высокоуровневые или обобщённые
- Нет анализаторов , которые на 100% покрывают MISRA
  - Поэтому пишут:
    X MISRA checker covers 100% of all **automatically testable** MISRA rules.

- Поэтому часть работы по соблюдению правил MISRA будет всё равно выполняться людьми на обзорах кода

Многие MISRA правила не абсолютны

- Их можно нарушать
- Иначе многие программы просто не написать

- Но каждое отхождение от стандарта должны быть объяснено и задокументировано (например комментарием)

PVS-Studio и MISRA

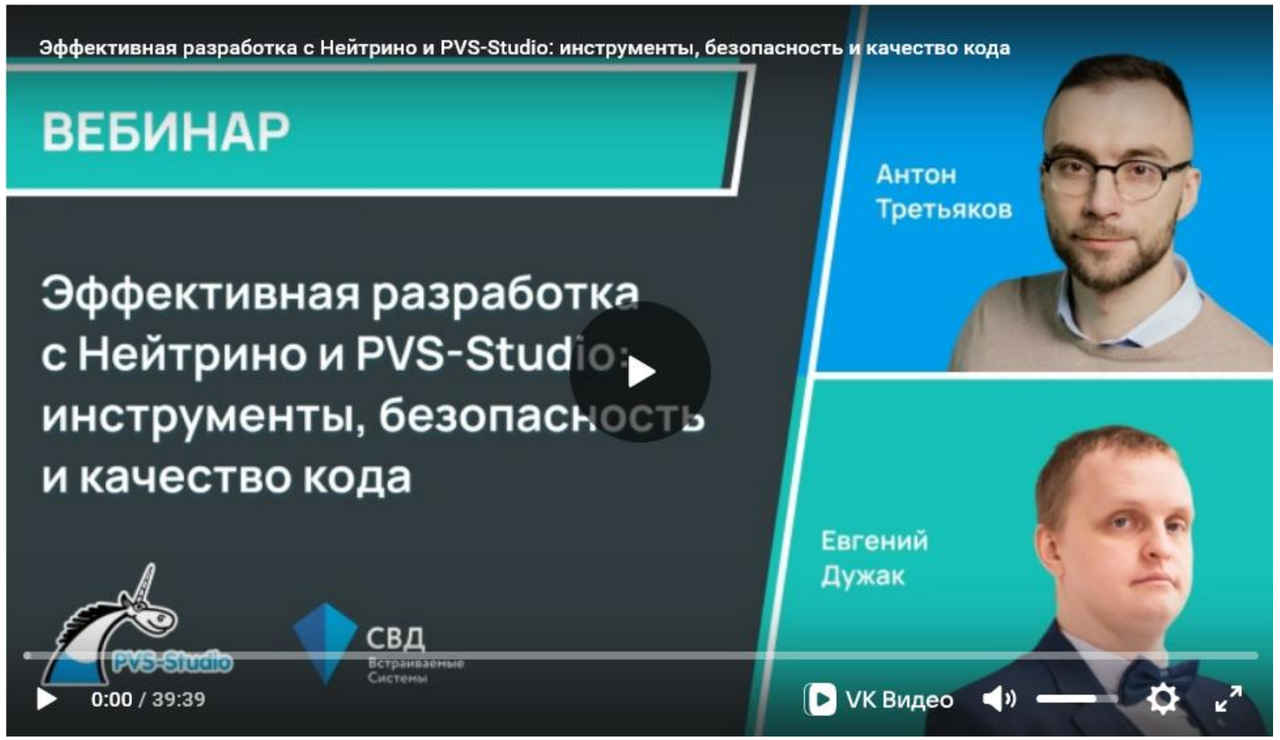
PVS-Studio

- Статический анализатор кода для поиска ошибок и потенциальных уязвимостей
- Поддерживает: **C, C++, C#, Java**
- Поддержка MISRA C:2012 – 81%
- Поддержка MISRA C:2023 – 71%
  - К концу года планируем уровень более 80%
- Поддержка MISRA C:2025 и MISRA C++ – в планах
- Подробнее: pvs-studio.ru/ru/pvs-studio/sast/misra/

PVS-Studio

- Включён в Реестр российского ПО: запись № 9837
- Совместим с **ГОСТ Р 71207-2024** (Статический анализ кода)
- Соответствует требованиям «Методики выявления уязвимостей и недекларированных возможностей в программном обеспечении» от 25 декабря 2020 г.
- Может применяться для РБПО согласно **ГОСТ Р 56939-2024**
- Сотрудничество с вузами: бесплатные лицензии для студентов и преподавателей
- Участвует в инициативе ФСТЭК по испытанию статических анализаторов

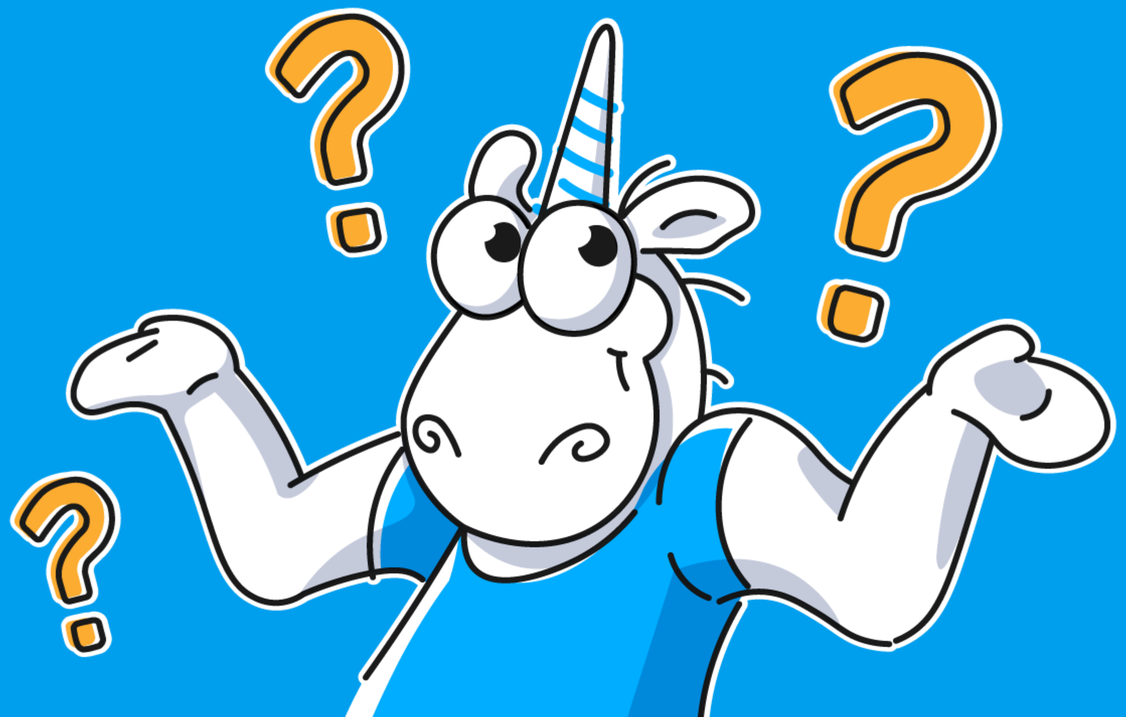
Интеграция PVS-Studio с комплектом разработчика Нейтрино



pvs-studio.ru/ru/blog/video/11355/

Остались вопросы

Бестиарий программирования



Дополнительные материалы

Дополнительные ссылки: доклады

- Цикл – Вокруг РБПО за 25 вебинаров: ГОСТ Р 56939-2024
  https://pvs-studio.ru/ru/webinar/rbpo/

- Статический анализ C++ кода по ГОСТ Р 71207-2024 на примере PVS-Studio
  https://pvs-studio.ru/ru/blog/video/11236/

Дополнительные ссылки: статьи

- PVS-Studio соответствует требованиям ГОСТ Р 71207-2024 (статический анализ программного обеспечения)
  https://pvs-studio.ru/ru/blog/posts/1204/

- Необходимость статического анализа для РБПО на примере 190 багов в TDengine
  https://pvs-studio.ru/ru/blog/posts/cpp/1249/

Дополнительные ссылки: статьи

- Что такое MISRA и как её готовить
  https://pvs-studio.ru/ru/blog/posts/cpp/0702/

- Зачем нужен отчёт MISRA Compliance и как его получить в PVS-Studio?
  https://pvs-studio.ru/ru/blog/posts/cpp/0863/

Дополнительные ссылки: статьи

- Интеграция PVS-Studio в PlatformIO
  https://pvs-studio.ru/ru/blog/posts/cpp/0714/
- Исследуем качество кода операционной системы Zephyr
  https://pvs-studio.ru/ru/blog/posts/0721/
- Espressif IoT Development Framework: 71 выстрел в ногу
  https://pvs-studio.ru/ru/blog/posts/cpp/0790/

Карпов Андрей Николаевич

- Карпов Андрей Николаевич, 1981
- ООО «ПВС», директор по развитию бизнеса
- Более 18 лет занимаюсь темой статического анализа кода и качества программного обеспечения. Автор большого количества статей, посвящённых написанию качественного кода на языке C++. Один из основателей проекта PVS-Studio. Долгое время являлся CTO компании и занимался разработкой C++ ядра анализатора. Основная деятельность на данный момент — развитие компании, обучение сотрудников и DevRel деятельность
- Другая информация и контакты