

ВОКРУГ РБПО ЗА 25 ВЕБИНАРОВ

ГОСТ Р 56939-2024

Вебинар 10. Статический анализ исходного кода



ГОСТ Р 71207



ГОСТ Р 71207–2024

Защита информации

РАЗРАБОТКА БЕЗОПАСНОГО ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

Статический анализ программного обеспечения

Общие требования

Введён в действие 01.04.2024

Введён впервые

ГОСТ Р 71207–2024

Защита информации

РАЗРАБОТКА БЕЗОПАСНОГО ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

Статический анализ программного обеспечения

Общие требования

Введён в действие 01.04.2024

Введён впервые

To Be Continued

ГОСТ Р 56939



ГОСТ Р 56939

Защита информации

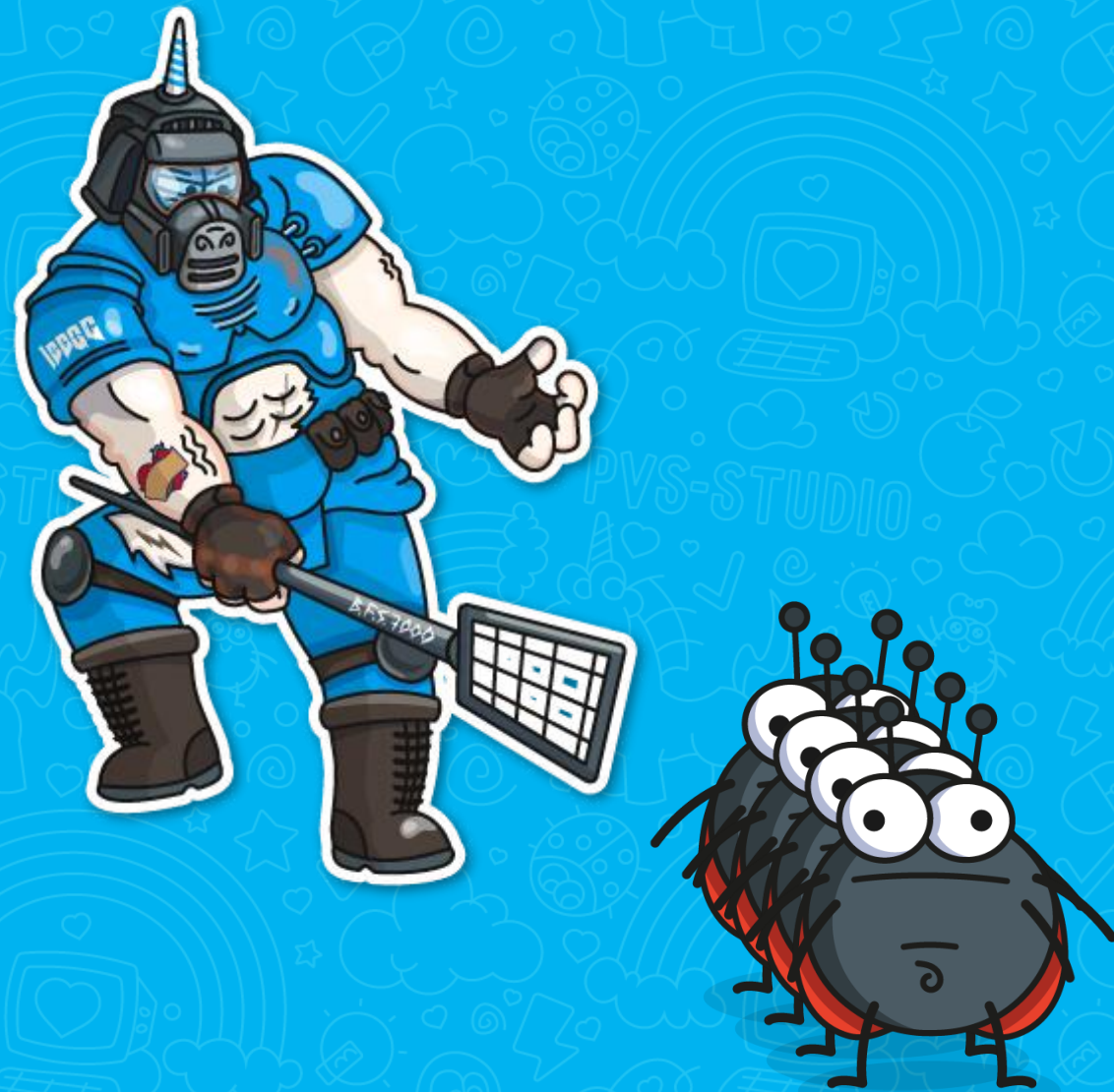
РАЗРАБОТКА БЕЗОПАСНОГО ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

Общие требования

Введён в действие в 2016 г.

ГОСТ Р 56939

- Направлен на предотвращение и устранение уязвимостей программ
- Меры по разработке безопасного ПО:
 - ...
 - Динамический анализ кода
 - **Статический анализ кода**
 - ...



В соответствии с требованиями ГОСТ Р 56939...

- Разработчик ПО должен проводить **регулярный поиск уязвимостей** на этапе эксплуатации жизненного цикла ПО.
- В состав применяемых инструментальных средств разработчик ПО **должен включать статический анализатор**.
- Рекомендуется настраивать конфигурацию статического анализатора и **применять** специализированные **методы статического анализа для** более глубокого **поиска ошибок**.



ГОСТ Р 71407-2024

ГОСТ Р 71207–2024

Описывает:

- Термины;

ГОСТ Р 71207–2024

Описывает:

- Термины;
- Порядок внедрения и выполнения статического анализа;

ГОСТ Р 71207–2024

Описывает:

- Термины;
- Порядок внедрения и выполнения статического анализа;
- Классификацию ошибок, обнаруживаемых статическими анализаторами;

ГОСТ Р 71207–2024

Описывает следующие требования:

ГОСТ Р 71207–2024

Описывает следующие требования:

- К выполнению анализа;

ГОСТ Р 71207–2024

Описывает следующие требования:

- К выполнению анализа;
- К методам анализа;

ГОСТ Р 71207–2024

Описывает следующие требования:

- К выполнению анализа;
- К методам анализа;
- К инструментам анализа;

ГОСТ Р 71207–2024

Описывает следующие требования:

- К выполнению анализа;
- К методам анализа;
- К инструментам анализа;
- К специалистам, участвующим в выполнении анализа;

ГОСТ Р 71207–2024

Описывает следующие требования:

- К выполнению анализа;
- К методам анализа;
- К инструментам анализа;
- К специалистам, участвующим в выполнении анализа;
- К методике проверки статических анализаторов на соответствие стандартам.

Причины возникновения



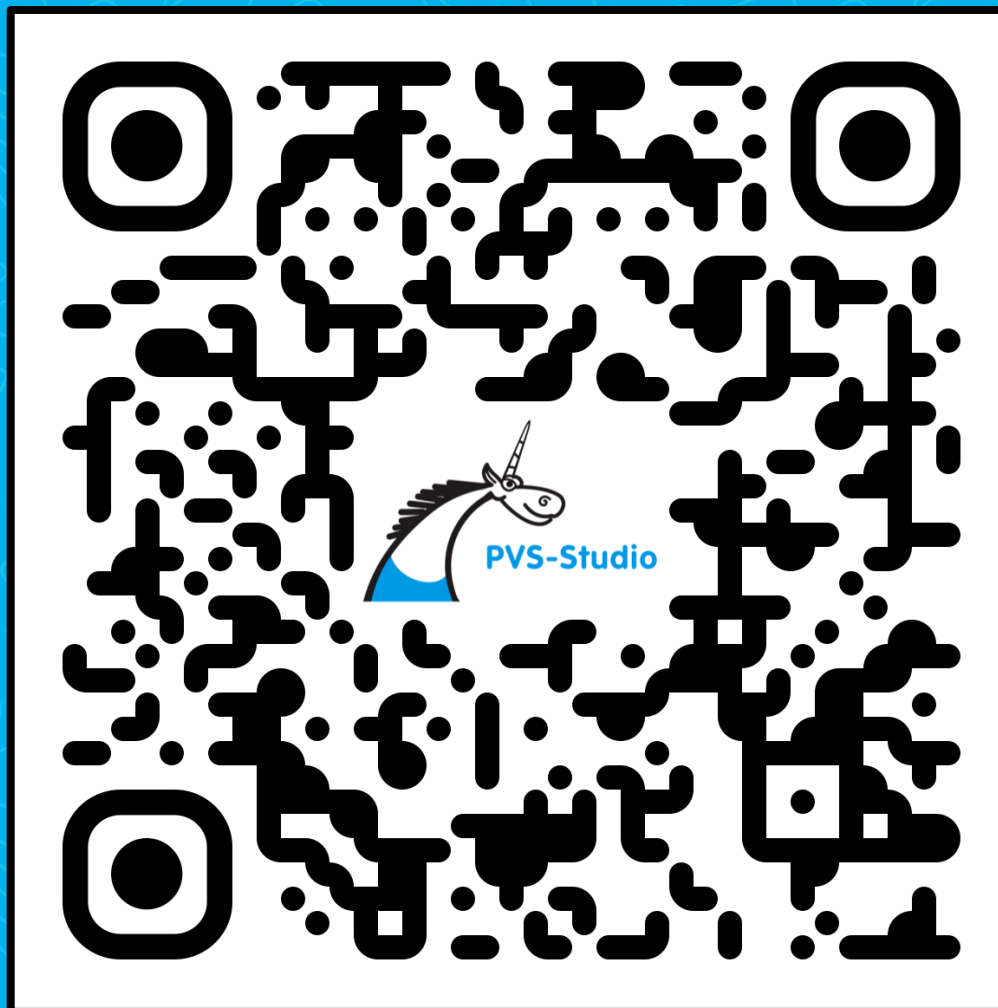
Проблематика: инструменты

- Путаница
- FindBugs для Java?
Последний релиз был 9 лет назад.
- Ну, хорошо, а SpotBugs?
Последний стабильный релиз был 2 года назад.
- Srrprcheck для C++?



Полная версия.

ГОСТ Р 71207–2024. Статический анализ программного обеспечения.
Общее описание и актуальность



pvs-studio.ru/ru/webinar/

Терминология



Термины

Термины

- Императивный язык программирования:

Термины

- Императивный язык программирования:
Язык программирования, в котором используется парадигма программирования, описывающая действия над данными в терминах последовательности команд.

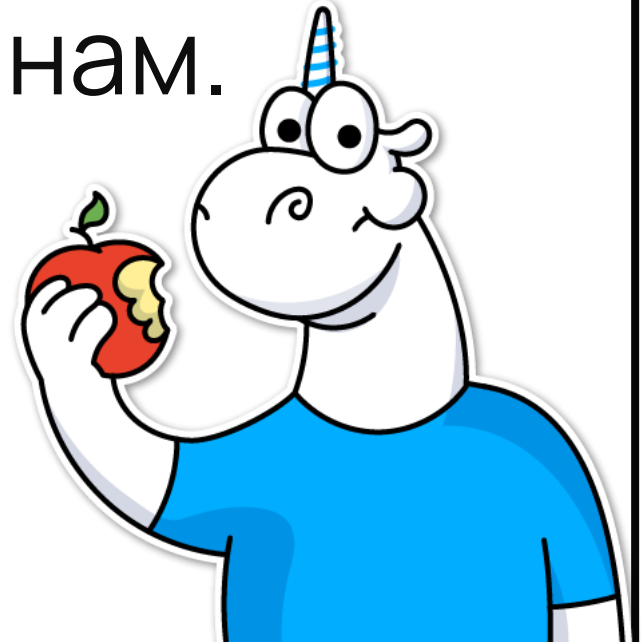
Термины

- Императивный язык программирования:
Язык программирования, в котором используется парадигма программирования, описывающая действия над данными в терминах последовательности команд.
- Пропустим.



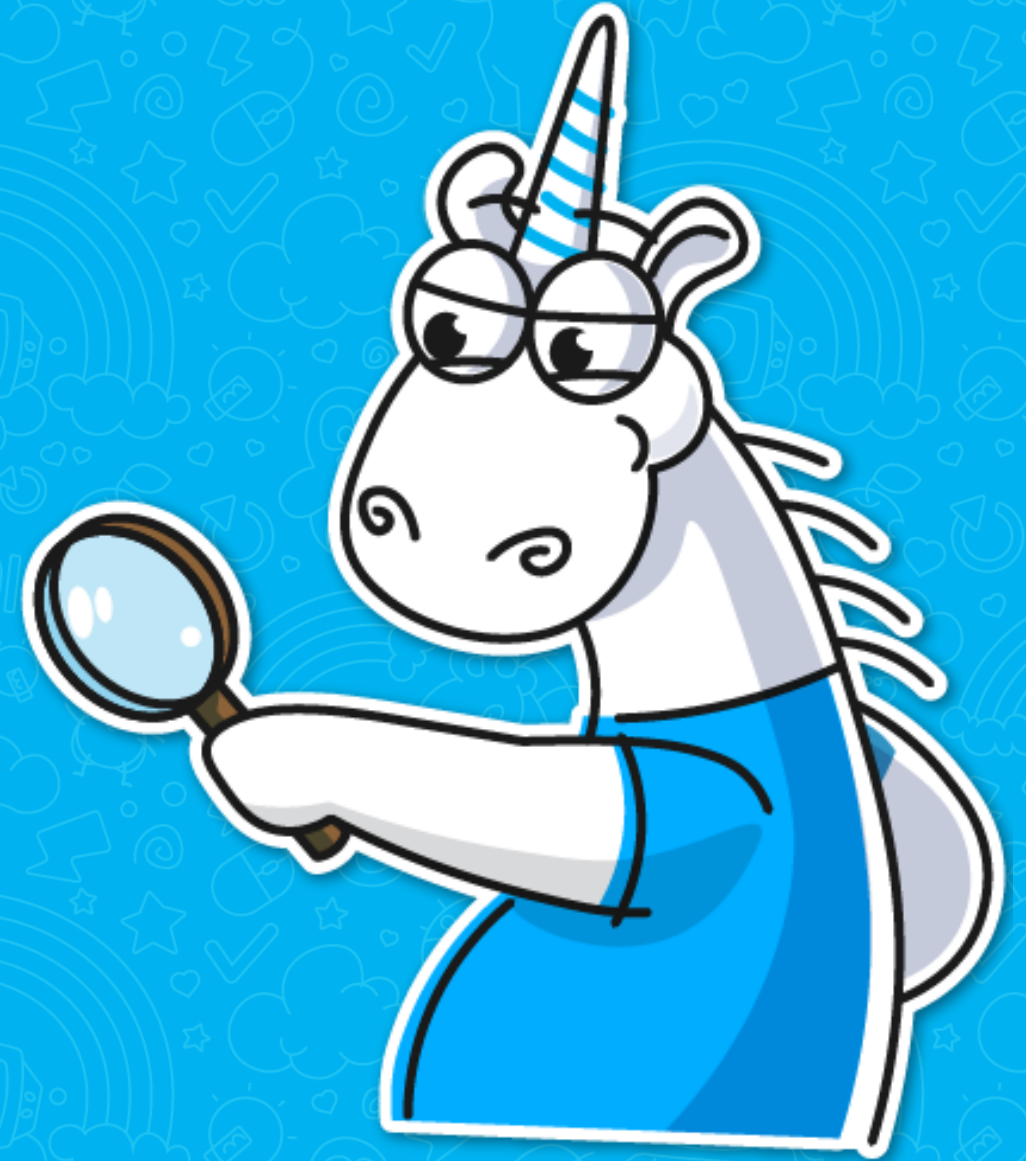
Термины

- Императивный язык программирования
- Пропустим.
- Пройдёмся по интересным / полезным с практической точки зрения терминам.



Анализ потока данных

- Определяет предположительное значение переменных и констант.
- Примеры артефактов:
 - диапазон значений
 - точное значение
 - множество значений



Анализ потока данных (RavenDB)

```
public override void VisitMethod(MethodExpression expr)
{
    if (    expr.Name.Value == "id"
        && expr.Arguments.Count == 0)
    {
        . . .
    }
}
```

Анализ потока данных (RavenDB)

```
public override void VisitMethod(MethodExpression expr)
{
    if (    expr.Name.Value == "id"
        && expr.Arguments.Count == 0)
    {
        if ( expr.Arguments.Count != 1)
        {
            throw new InvalidOperationException("....");
        }
        ....
    }
}
```


Анализ потока данных (RavenDB)

```
public override void VisitMethod(MethodExpression expr)
{
    if (    expr.Name.Value == "id"
        && expr.Arguments.Count == 0)
    {
        if ( expr.Arguments.Count != 1)
        {
            throw new InvalidOperationException("...");
        }
    }
}
```

Предупреждение PVS-Studio:

V3022 Expression 'expr.Arguments.Count != 1' is always true.

Межпроцедурный контекстно-чувствительный анализ

```
// Если в функцию передали нулевую ссылку, то она
// вернёт false.
private static bool IsBuiltInType(ClangType cursor)
{
    var result = false;
    if (cursor != null && ...)
    {
        return true;
    }
    return result;
}
```

Межпроцедурный контекстно-чувствительный анализ

```
if (      cursor.ResultType != null)
....
else if (cursor.CursorType != null)
{
    ....
    // Значит здесь cursor.ResultType == null
    result.Append(cursor.CursorType.Spelling + " ",
                  IsBuiltInType(cursor.ResultType)
                  ? theme.Keyword : theme.Type);
}
```

Как появилась эта ошибка

```
// Был вот такой код

if (cursor.ResultType != null)
{
    result.Append(    cursor.ResultType.Spelling + " ",
        IsBuiltInType(cursor.ResultType)
        ? theme.Keyword : theme.Type);
}
```

Как появилась эта ошибка

```
// Неудачный Copy-Paste:  
// забыли заменить ResultType на CursorType  
if (cursor.CursorType != null)  
{  
    result.Append(    cursor.CursorType.Spelling + " ",  
        IsBuiltInType(cursor.ResultType)  
        ? theme.Keyword : theme.Type);  
}
```

Межмодульный анализ (Barotrauma)

```
// Функция Remove() делает ссылку Sprite нулевой
partial class DecorativeSprite : ISerializableEntity
{
    public Sprite Sprite { get; private set; }
    ....
    public void Remove()
    {
        Sprite?.Remove();
        Sprite = null;           // <=
        ....
    }
}
```


Межмодульный анализ (Barotrauma)

```
public void RecreateSprites()  
{  
    for (int i = 0; i < DecorativeSprites.Count; i++)  
    {  
        var decorativeSprite = DecorativeSprites[i];  
        decorativeSprite.Remove();  
        var source =  
            decorativeSprite.Sprite.SourceElement;  
        . . .  
    }  
}
```

Межмодульный анализ (Barotrauma)

```
public void RecreateSprites()  
{  
    for (int i = 0; i < DecorativeSprites.Count; i++)  
    {  
        var decorativeSprite = DecorativeSprites[i];  
        decorativeSprite.Remove();  
        var source =  
            decorativeSprite.Sprite.SourceElement;  
    }  
}
```

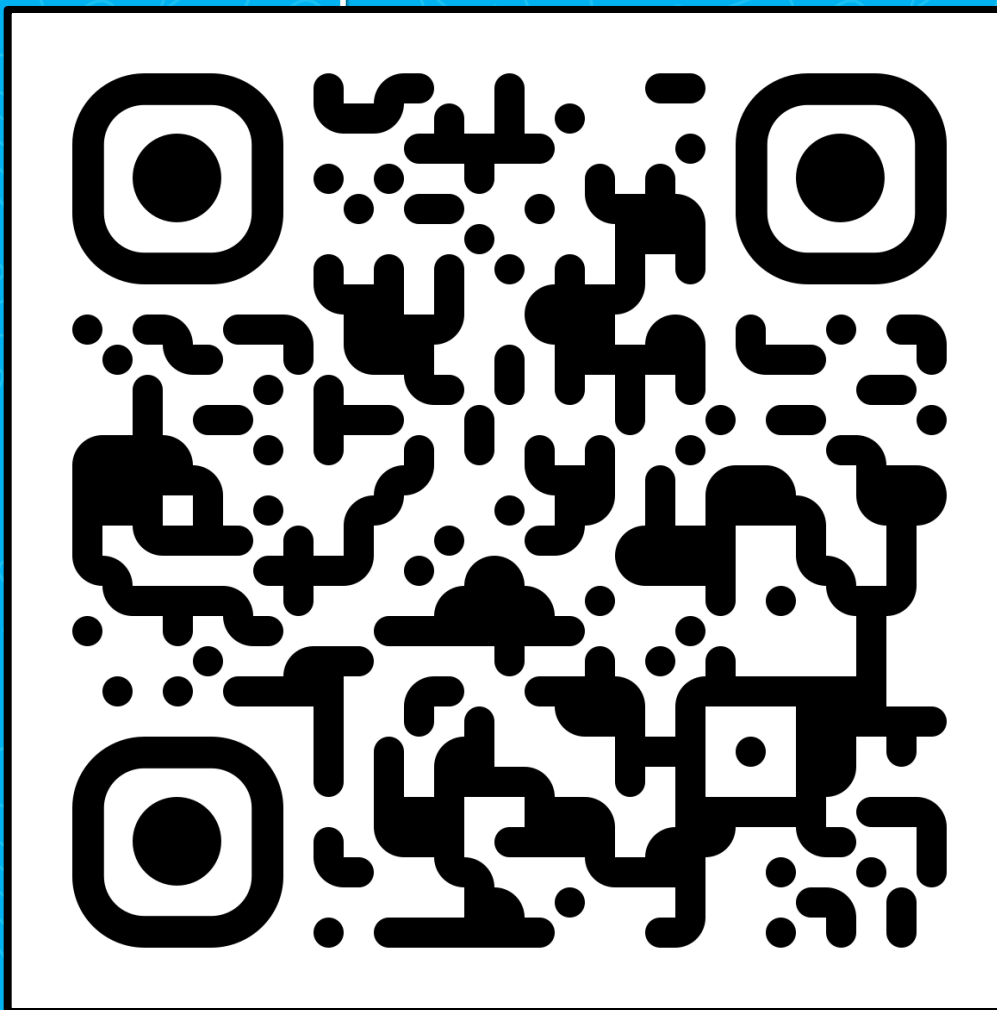
Предупреждение PVS-Studio:

V3080. Possible null dereference. Consider inspecting 'decorativeSprite.Sprite'.

Полная версия.

ГОСТ Р 71207–2024 - Статический анализ программного обеспечения.

Терминология



Критическая ошибка (!)

- Самое важное понятие
- Ошибка, которая может привести к нарушению безопасности.
- Мы называли такие ошибки потенциальными уязвимостями.



Пример критической ошибки



Ошибки непроверенного использования чувствительных данных (ONLYOFFICE Community Server, 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;
    CredentialsDomain = domain;
}
```


Ошибки непроверенного использования чувствительных данных (ONLYOFFICE Community Server, 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;
    CredentialsDomain = domain;
}
```

Ошибки непроверенного использования чувствительных данных (ONLYOFFICE Community Server, 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;
    CredentialsDomain = domain;
}
```

Ошибки непроверенного использования чувствительных данных (ONLYOFFICE Community Server, 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;
    CredentialsDomain = domain;
}
```

Ошибки непроверенного использования чувствительных данных (ONLYOFFICE Community Server, 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;
    CredentialsDomain = domain;
}
```

Ошибки непроверенного использования чувствительных данных (ONLYOFFICE Community Server, C#)

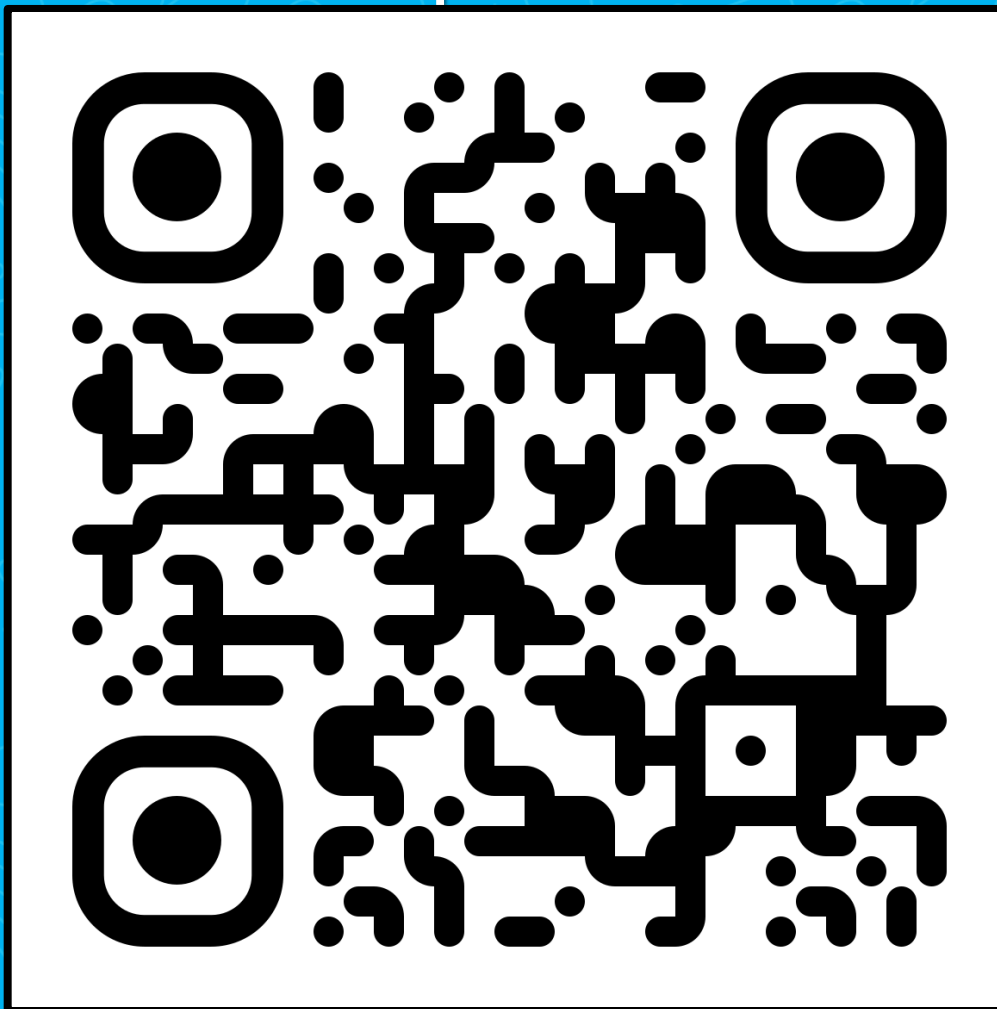
```
public void SetCredentials
(string userName, string password, string domain) {
    if (string.IsNullOrEmpty(userName)) {
        throw new ArgumentException
            ("Empty user name.", "userName");
    }
    if (string.IsNullOrEmpty("password")) {
        throw new ArgumentException
       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

Предупреждение PVS-Studio:

V3022 Expression 'string.IsNullOrEmpty("password")' is always false.

Полная версия.

ГОСТ Р 71207–2024 — Статический анализ программного обеспечения. Критические ошибки



**Какие методы анализа должны
использовать инструменты**



Внутрипроцедурный анализ потоков данных и управления (Unity3D, C#)

```
public NetworkConnection Get(int connId)
{
    if (connId < 0)
    {
        return m_LocalConnections[Mathf.Abs(connId) - 1];
    }

    if (connId < 0 || connId > m_Connections.Count)
    {
        ....
    }
}
```

Внутрипроцедурный анализ потоков данных и управления (Unity3D, C#)

```
public NetworkConnection Get(int connId)
{
    if (connId < 0)
    {
        return m_LocalConnections[Mathf.Abs(connId) - 1];
    }

    if (connId < 0 || connId > m_Connections.Count)
    {
        return null;
    }
}
```

Предупреждение PVS-Studio:

V3063 A part of conditional expression is always false: connId < 0.

UnityEngine.Networking ConnectionArray.cs 59

Анализ псевдонимов (C#)

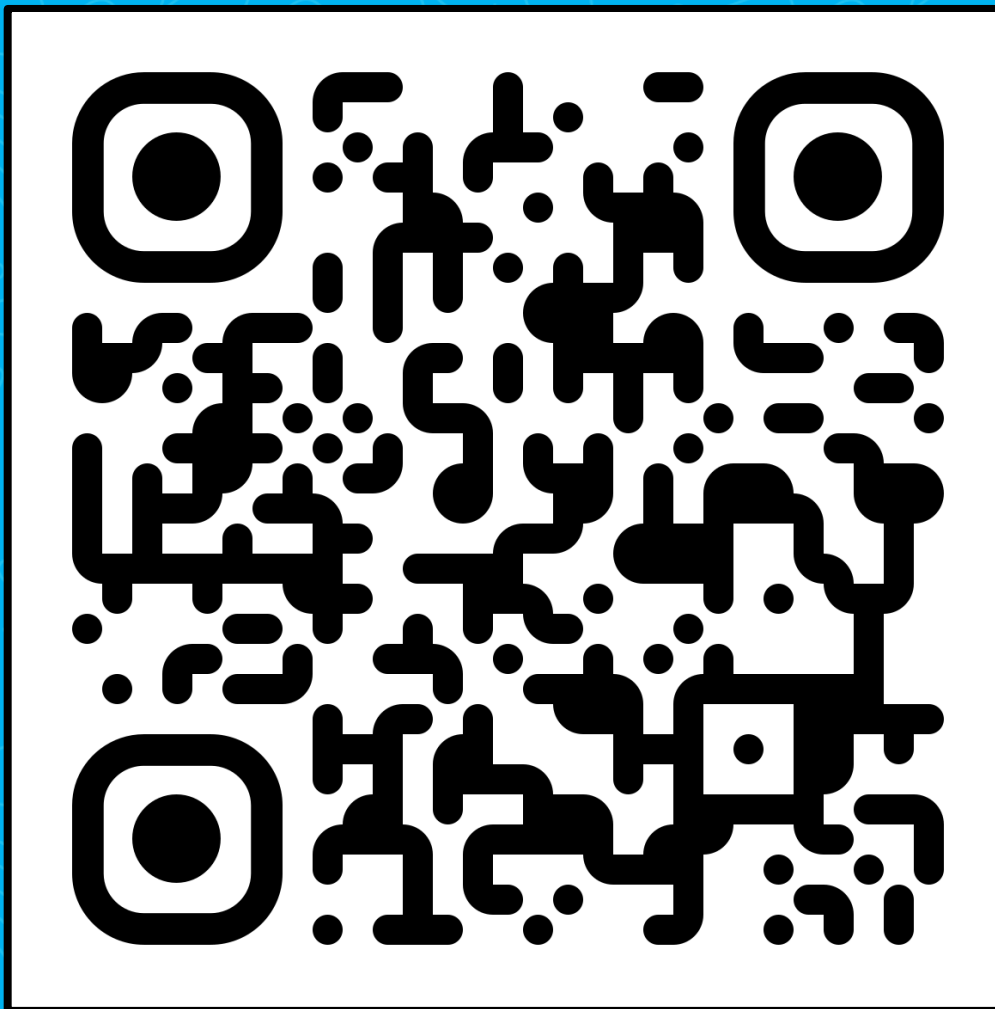
```
object GetPotentialNull() {  
    Random random = new();  
    return random.NextDouble() > 0.5 ? new object() : null;  
}  
  
void Example_1() {  
    object potentialNull = GetPotentialNull();  
    ref object alias = ref potentialNull;  
  
    if (alias != null) {  
        // Диагностика V3080 не срабатывает,  
        // т. к. alias и potentialNull - одно и тоже значение.  
        _ = potentialNull.ToString();  
    }  
}
```

Статистический анализ (Daggerfall Unity, C#)

```
case GuildServices.Identify:
    ....
    uiManager.PushWindow(UIWindowFactory.GetInstanceWithArgs(...));
    break;
case GuildServices.Repair:
    ....
    uiManager.PushWindow(UIWindowFactory.GetInstanceWithArgs(...));
    break;
case GuildServices.Training:
    ....
    UIWindowFactory.GetInstanceWithArgs(...);
    break;
case GuildServices.Donate:
    ....
    uiManager.PushWindow(UIWindowFactory.GetInstanceWithArgs(...));
    break;
```


Полная версия.

ГОСТ Р 71207–2024 — Статический анализ программного обеспечения. Технологии анализа кода

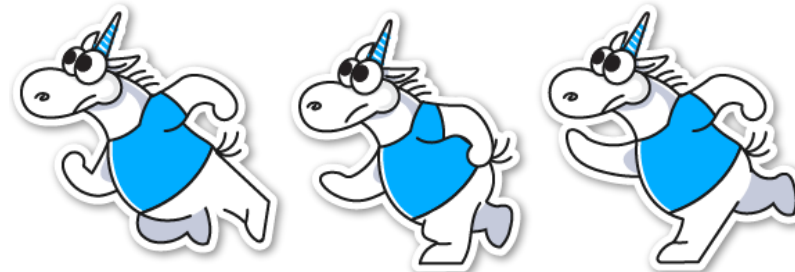


Новые требования



Про выбор анализаторов

- Приоритет следует отдавать инструментам, демонстрирующим лучшие показатели.
- Согласно ГОСТу, в ходе разработки необходимо использовать статический анализатор или **набор статических анализаторов** для поиска ошибок;
- Не требуется выбирать один универсальный. Можно использовать несколько, выбирая наиболее подходящие и мощные решения.



Требования к обновлениям

- Следует использовать **регулярно обновляющиеся статические анализаторы**
- Вспоминаем FindBugs, SpotBugs.



Подготовка сборочной среды

Следует использовать среду анализа, позволяющую выполнить проверку ПО выбранным инструментом (инструментами) с учётом временных ограничений:

- анализ проекта (**с учётом заимствованных компонентов**) должен выполняться **не более чем 2 суток**;
- простыми словами: всё должно проанализироваться за выходные 😊
- Допустимо разворачивать анализатор в виртуальной инфраструктуре или использовать в виде сервиса.

Внедрение: начальный этап



Начальный этап

- Настройка инструмента статического анализа для данного ПО.
 - Выбор и включение типов предупреждений, соответствующий списку **критических ошибок**.
 - Можно включить и другие типы предупреждений.
- **Выполнение первичного статического анализа** кода.
- **Разметка полученных результатов** и формирование начальной базы предупреждений о потенциальных ошибках.

Первичная разметка предупреждений

- Полученный набор предупреждений должен быть размечен.
- Результаты разметки должны быть сохранены для возможности сравнения результатов последующих запусков.
- После первичной разметки конфигурация статического анализатора может быть доработана.

Примеры доработок:

- подавление предупреждений в макросах (C, C++);
- сокращённый набор правил для тестов.

Ещё про первичную разметку предупреждений

- Все предупреждения о критических ошибках должны быть отнесены к одной из категорий:
 - истинные предупреждения;
 - истинные предупреждения, но не требующие исправления кода;
 - ложные предупреждения.
- Допускается расширять перечень категорий.



Проведение регулярного статического анализа ПО

Требования к регулярности

- Для своевременного выявления и исправления ошибок статический анализ должен выполняться регулярно;
- Накопление непроанализированных изменений ухудшает качество проводимой экспертизы;
 - Постоянно про это рассказываем;
 - Боремся с подходом «запустим анализатор перед релизом».



«Анализатор запускается» не равно «анализатор внедрён»

- Статический анализ следует проводить регулярно на этапе конструирования;
- Статический анализ **всего** разрабатываемого ПО следует выполнять **не реже одного раза в 10 рабочих дней**, если за данный период времени исходный код был изменён.

«Анализатор запускается» не равно «анализатор внедрён»

- Для своевременной и эффективной работы с отчётом **просмотр предупреждений** следует выполнять:
 - при анализе **измененных** частей ПО — не позже, **чем через 3 рабочих дня** после выполнения анализа;
 - при анализе ПО целиком — не позже, **чем через 10 рабочих дней** после выполнения анализа.

Требования ко времени работы инструментов

- Анализатор должен производить **полный анализ ПО** со всеми компонентами не более **двух суток**.
- PVS-Studio:
 - Xeon Gold 5220R, 24 ядра, 2.20 GHz, 128 Gb;
 - 122 C и C++ проекта (Visual C++);
 - 1 час 34 минуты.

Требования к документации

- Для каждого типа ошибки должны быть приведены:
 - описания;
 - возможная причина возникновения;
 - примеры ошибочного кода, для которого выдаётся предупреждение о данном типе ошибки;
 - примеры или рекомендации исправления данного типа ошибки.
- Очевидное требования, но на практике далеко не все анализаторы справляются с документацией.

Сppcheck (описания просто нет)

Auto Variables

A pointer to a variable is only valid as long as the variable is in scope.

Check:

- returning a pointer to auto or temporary variable
- assigning address of an variable to an effective parameter of a function
- returning reference to local/temporary variable
- returning address of function parameter
- suspicious assignment of pointer argument
- useless assignment of function argument

Boolean

Boolean type checks

- using increment on boolean
- comparison of a boolean expression with an integer other than 0 or 1
- comparison of a function returning boolean value using relational operator

SpotBugs (чуть лучше)

FI: Finalizer only nulls fields (FI_FINALIZER_ONLY_NULLS_FIELDS)

This finalizer does nothing except null out fields. This is completely pointless, and requires that the object be garbage collected, finalized, and then garbage collected again. You should just remove the finalize method.

FI: Finalizer nulls fields (FI_FINALIZER_NULLS_FIELDS)

This finalizer nulls out fields. This is usually an error, as it does not aid garbage collection, and the object is going to be garbage collected anyway.

UI: Usage of GetResource may be unsafe if class is extended (UI_INHERITANCE_UNSAFE_GETRESOURCE)

Calling `this.getClass().getResource(...)` could give results other than expected if this class is extended by a class in another package.

Ещё рекомендация по документированию диагностик

- Применяемая в статических анализаторах типизация разнится, что затрудняет соотнесение предупреждений, полученных от разных анализаторов;
- Для облегчения работы с предупреждениями в диагностическую информацию добавляют соотнесение ошибки с одной из популярных систем классификации дефектов безопасности, например, с **MITRE CWE**;
- В описании ошибок также следует указывать их соответствие идентификаторам в системе классификации дефектов безопасности MITRE CWE.

V772. Calling a 'delete' operator for a void pointer will cause undefined behavior.

Анализатор обнаружил потенциально возможную ошибку в коде, связанную с тем, что оператор 'delete' или 'delete[]' применяется для нетипизированного указателя (void*). Согласно стандарту C++20 (п. п. [§7.6.2.8/3](#)) такое применение ведет к неопределенному поведению.

Рассмотрим пример такого кода:

```
class Example
{
    int *buf;
public:
    Example(size_t n = 1024) { buf = new int[n]; }
    ~Example() { delete[] buf; }
};

....
void *ptr = new Example();
....
delete ptr;
....
```

Подобный пример опасен тем, что компилятор в реальности не знает, к какому типу относится указатель 'ptr'. Поэтому, при удалении такого нетипизированного указателя могут произойти различные неприятности, например, может возникнуть утечка памяти: оператор 'delete' не вызовет деструктор объекта типа 'Example', на который ссылается указатель 'ptr'.

Если подразумевалась именно работа с нетипизированным указателем, то перед применением оператора 'delete' ('delete[]') его необходимо привести к изначальному типу, например так

```
....
void *ptr = new Example();
....
delete (Example*)ptr;
....
```

Иначе, во избежание ошибок, рекомендуется использовать только типизированные указатели совместно с оператором 'delete' ('delete[]'):

```
....
Example *ptr = new Example();
....
delete ptr;
....
```

Данная диагностика классифицируется как:

- CWE-758
- CERT-MSC15-C

Документация здорового человека (PVS-Studio)

Описание

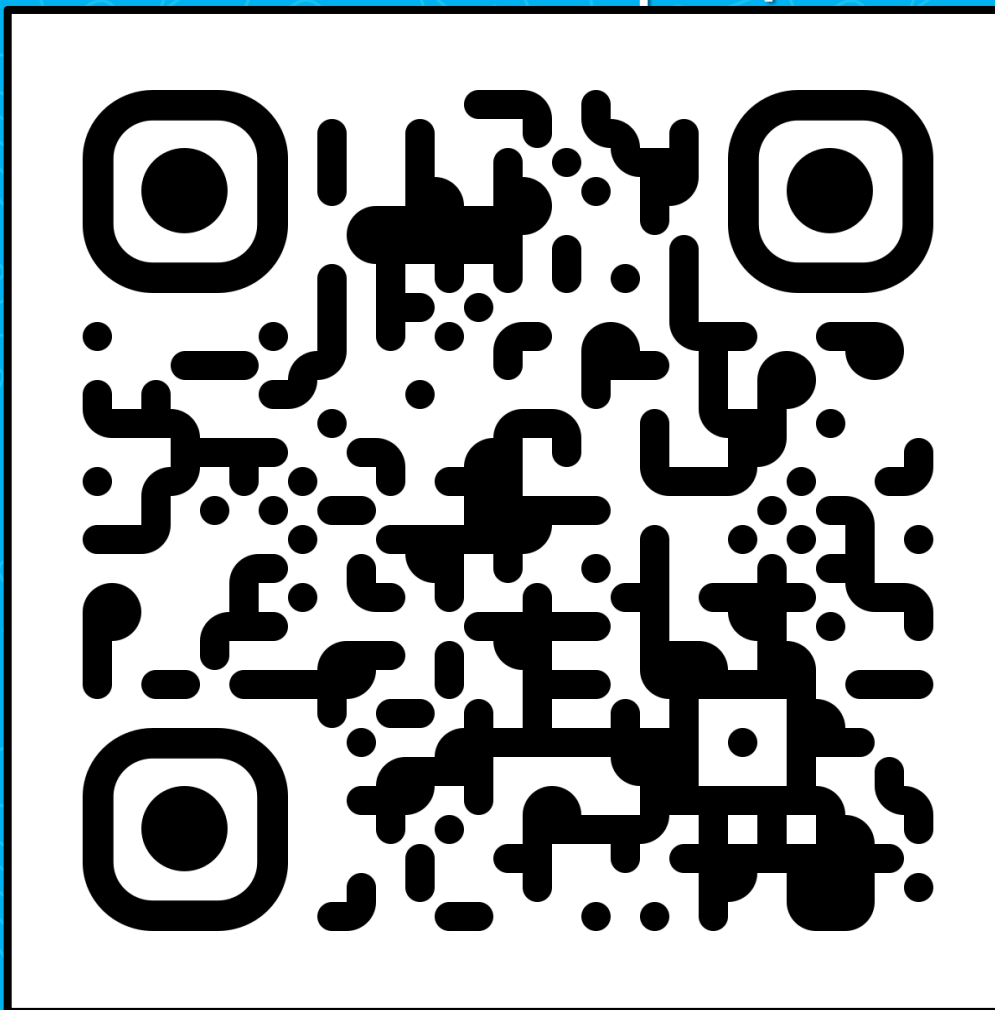
Пример ошибочного кода

Примеры исправления

Сопоставление с CWE, SEI CERT, OWASP

Заключительная часть

ГОСТ Р 71207–2024 — Статический анализ программного обеспечения. Процессы



pvs-studio.ru/ru/webinar/



ИТОГИ



Для пользователей анализаторов кода

- Можно понять, используются ли качественные инструменты анализа или нужно поискать что-то помощнее;
- Можно понять, правильно ли в компании построен процесс использования анализаторов;
- Хорошо описан процесс внедрения;
- Понятно, как проводить аттестацию инструментов.

Q&A



Задавайте
вопросы



Глеб Асламов