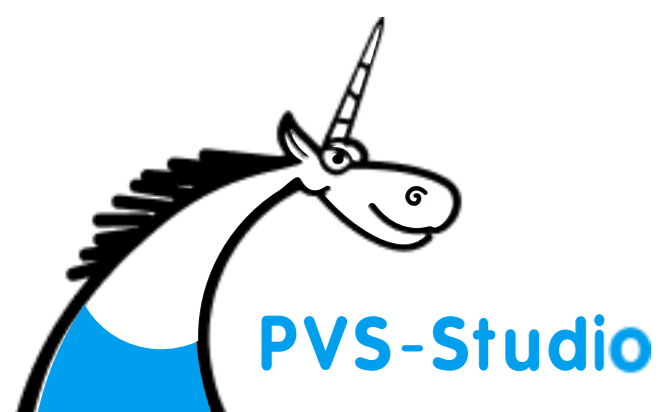


# SAST В РБПО

Что требует ГОСТ Р 71207-2024



**Валерий Филатов**  
Developer Advocate



# Валерий Филатов

Developer Advocate

- Занимаюсь поддержкой сборочной инфраструктуры и разработкой инструментов для разработчиков.
- Рассказываю про технологии статического анализа и не только в статьях и на различных мероприятиях.

Люблю Python и чистый код (да, это иногда сочетается).



# ПРЕДЫСТОРИЯ

I wanna tell a story...



## ГОСТ Р 56939-2016 «ЗАЩИТА ИНФОРМАЦИИ. РАЗРАБОТКА БЕЗОПАСНОГО ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ. ОБЩИЕ ТРЕБОВАНИЯ»

- Посвящён разработке безопасного программного обеспечения.



## ГОСТ Р 56939-2016 «ЗАЩИТА ИНФОРМАЦИИ. РАЗРАБОТКА БЕЗОПАСНОГО ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ. ОБЩИЕ ТРЕБОВАНИЯ»

- Посвящён разработке безопасного программного обеспечения.
- **Безопасное программное обеспечение** - программное обеспечение, разработанное с использованием совокупности мер, направленных на предотвращение появления и устранение уязвимостей программы.



# СТАТИЧЕСКИЙ АНАЛИЗ

- Пункт 3.10 этого стандарта посвящён статическому анализу.



# СТАТИЧЕСКИЙ АНАЛИЗ

- Пункт 3.10 этого стандарта посвящён статическому анализу.
- **Статический анализ исходного кода программы:** Вид работ по инструментальному исследованию программы, основанный на анализе исходного кода программы с использованием специализированных инструментальных средств (статических анализаторов) в режиме, не предусматривающем реального выполнения кода.



# ГОСТ Р 71207– 2024 «ЗАЩИТА ИНФОРМАЦИИ. РАЗРАБОТКА БЕЗОПАСНОГО ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ. СТАТИЧЕСКИЙ АНАЛИЗ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ. ОБЩИЕ ТРЕБОВАНИЯ»

- Вышел 1 апреля 2024 года



## ГОСТ Р 71207– 2024 «ЗАЩИТА ИНФОРМАЦИИ. РАЗРАБОТКА БЕЗОПАСНОГО ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ. СТАТИЧЕСКИЙ АНАЛИЗ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ. ОБЩИЕ ТРЕБОВАНИЯ»

- Вышел 1 апреля 2024 года
- Посвящён статическому анализу кода при РБПО.



## ГОСТ Р 71207– 2024 «ЗАЩИТА ИНФОРМАЦИИ. РАЗРАБОТКА БЕЗОПАСНОГО ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ. СТАТИЧЕСКИЙ АНАЛИЗ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ. ОБЩИЕ ТРЕБОВАНИЯ»

- Вышел 1 апреля 2024 года
- Посвящён статическому анализу кода при РБПО.
- Вводит требования как для пользователей статических анализаторов, так и непосредственно для инструментов.



## ГОСТ Р 71207– 2024 «ЗАЩИТА ИНФОРМАЦИИ. РАЗРАБОТКА БЕЗОПАСНОГО ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ. СТАТИЧЕСКИЙ АНАЛИЗ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ. ОБЩИЕ ТРЕБОВАНИЯ»

- Вышел 1 апреля 2024 года
- Посвящён статическому анализу кода при РБПО.
- Вводит требования как для пользователей статических анализаторов, так и непосредственно для инструментов.

Является дополнением ГОСТ Р 56939-2024.



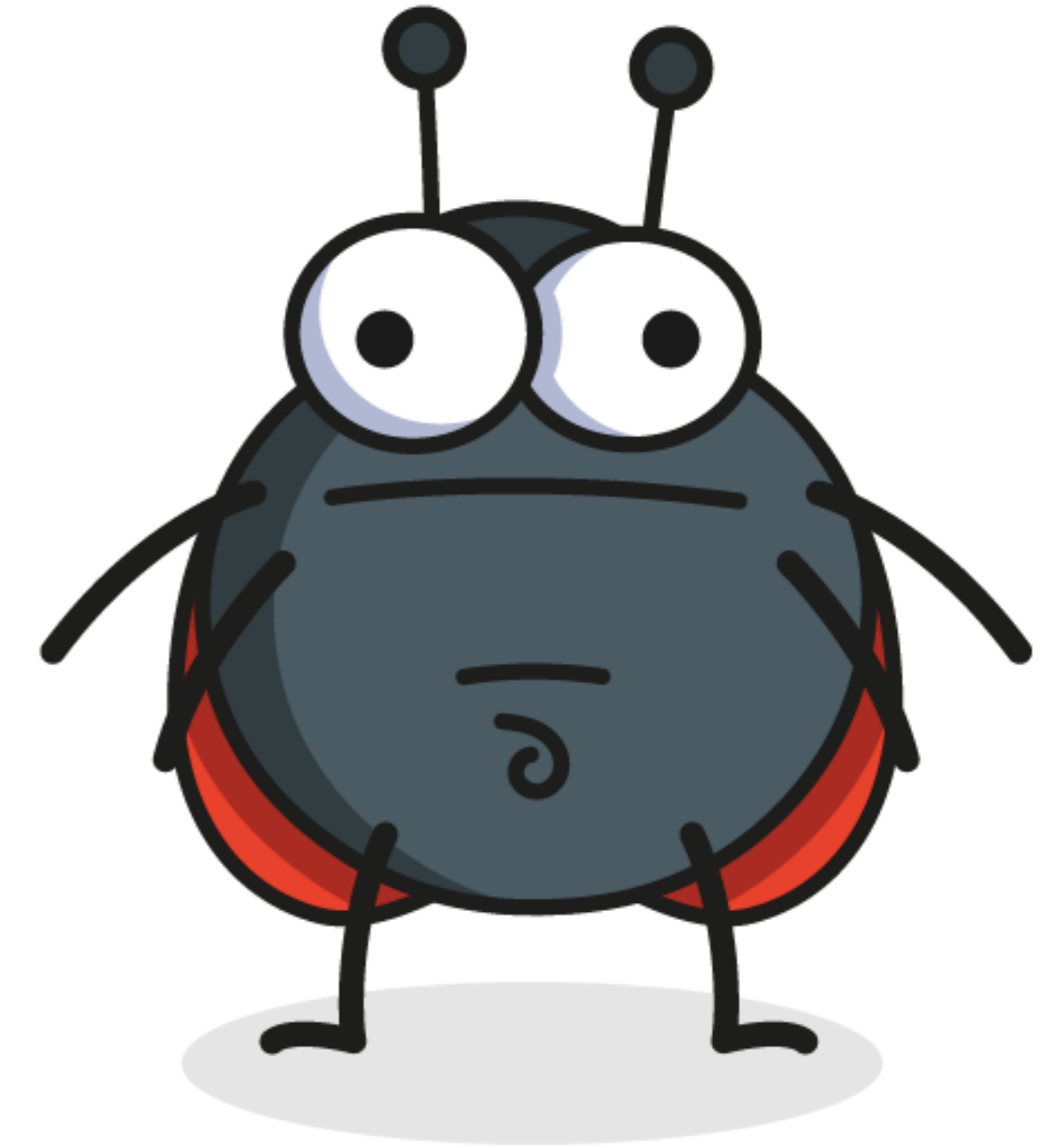
# КРИТИЧЕСКИЕ ОШИБКИ

И где они обитают



# ТЕРМИНОЛОГИЯ: КРИТИЧЕСКАЯ ОШИБКА

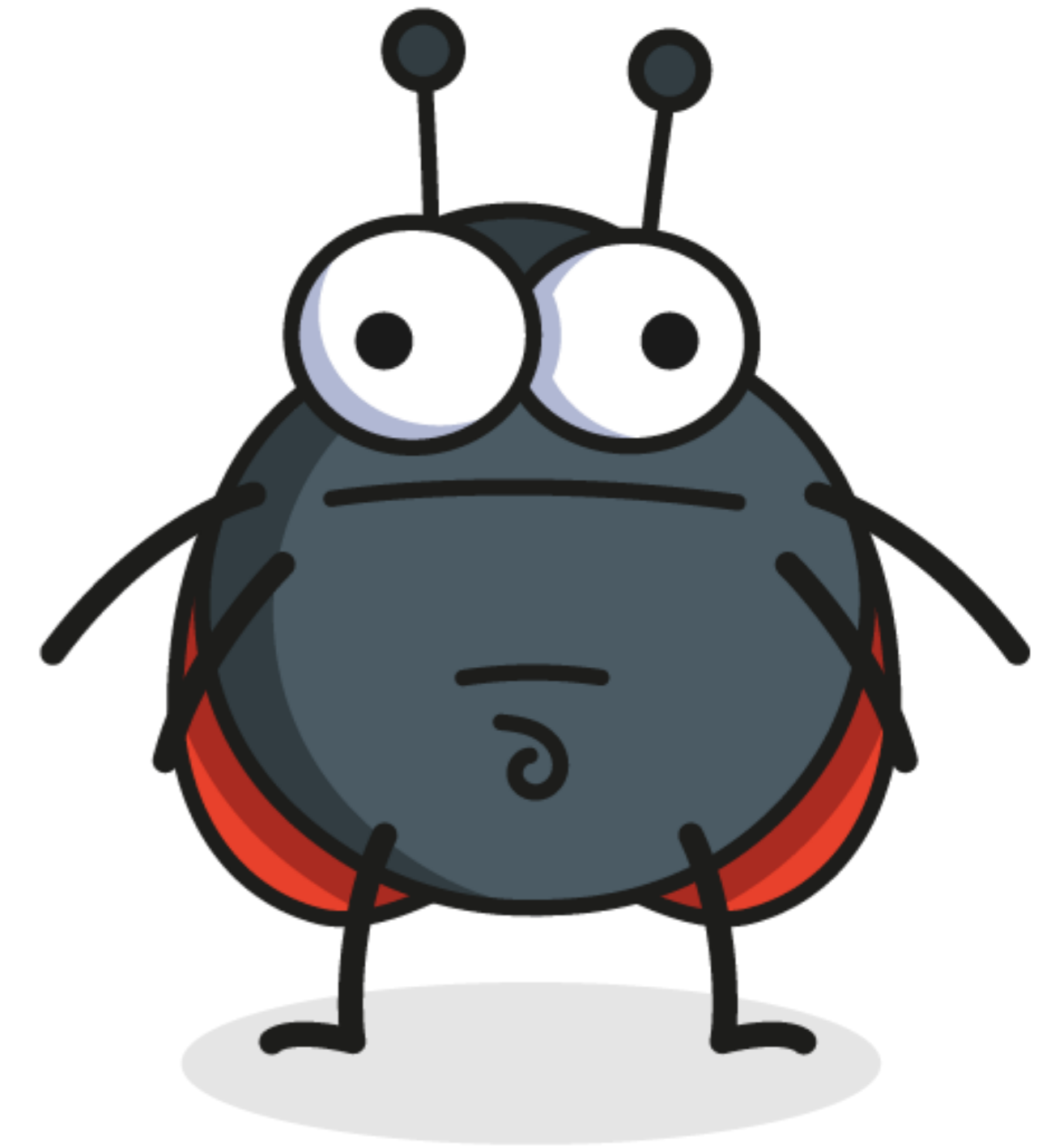
Критическая ошибка в программе: ошибка, которая может привести к нарушению безопасности обрабатываемой информации (п. 3.1.13).



# ТЕРМИНОЛОГИЯ: КРИТИЧЕСКАЯ ОШИБКА

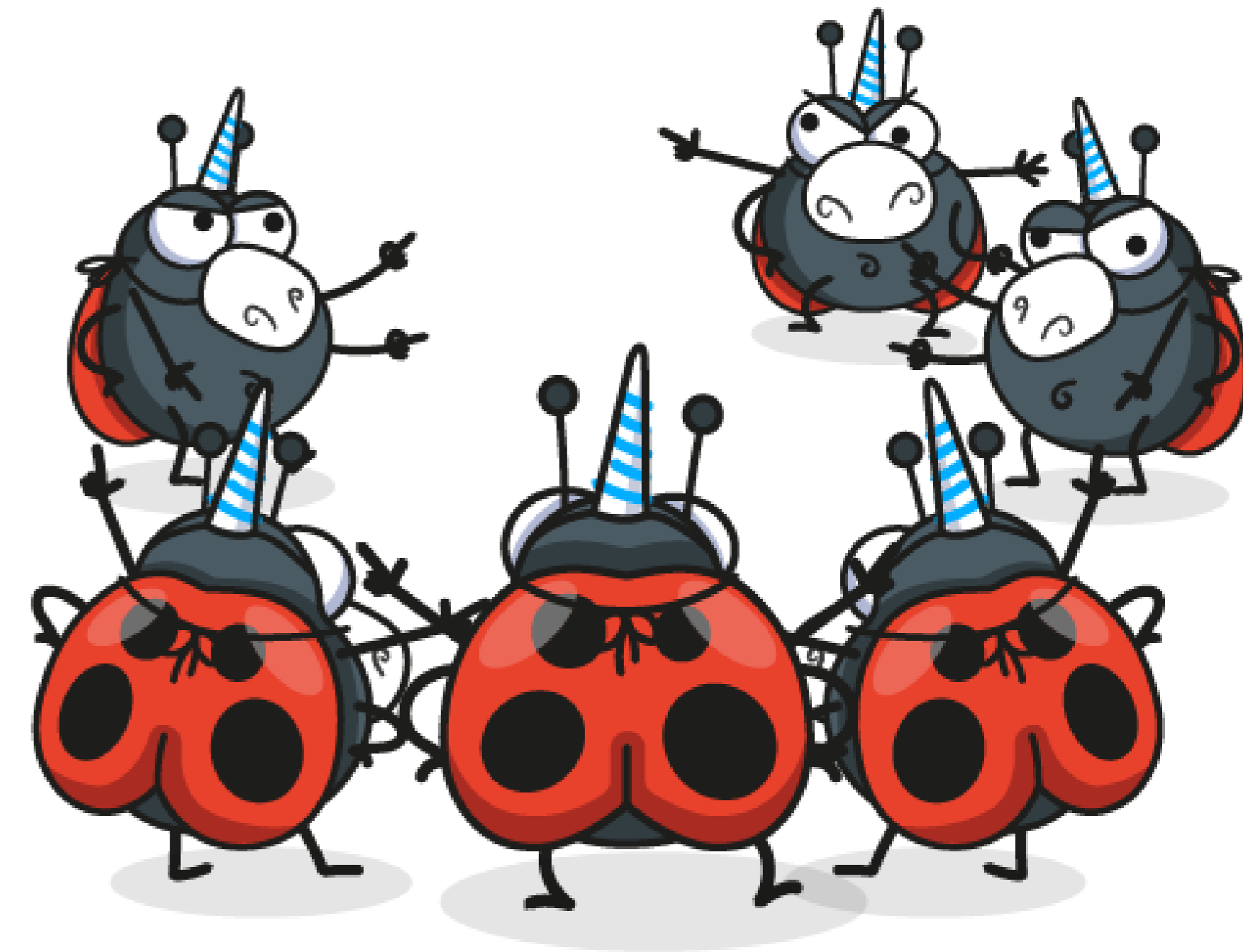
Критическая ошибка в программе: ошибка, которая может привести к нарушению безопасности обрабатываемой информации (п. 3.1.13).

Стандарт не разграничивает ошибки в области последствий - важен сам факт существования и необходимости исправления подобных ошибок.



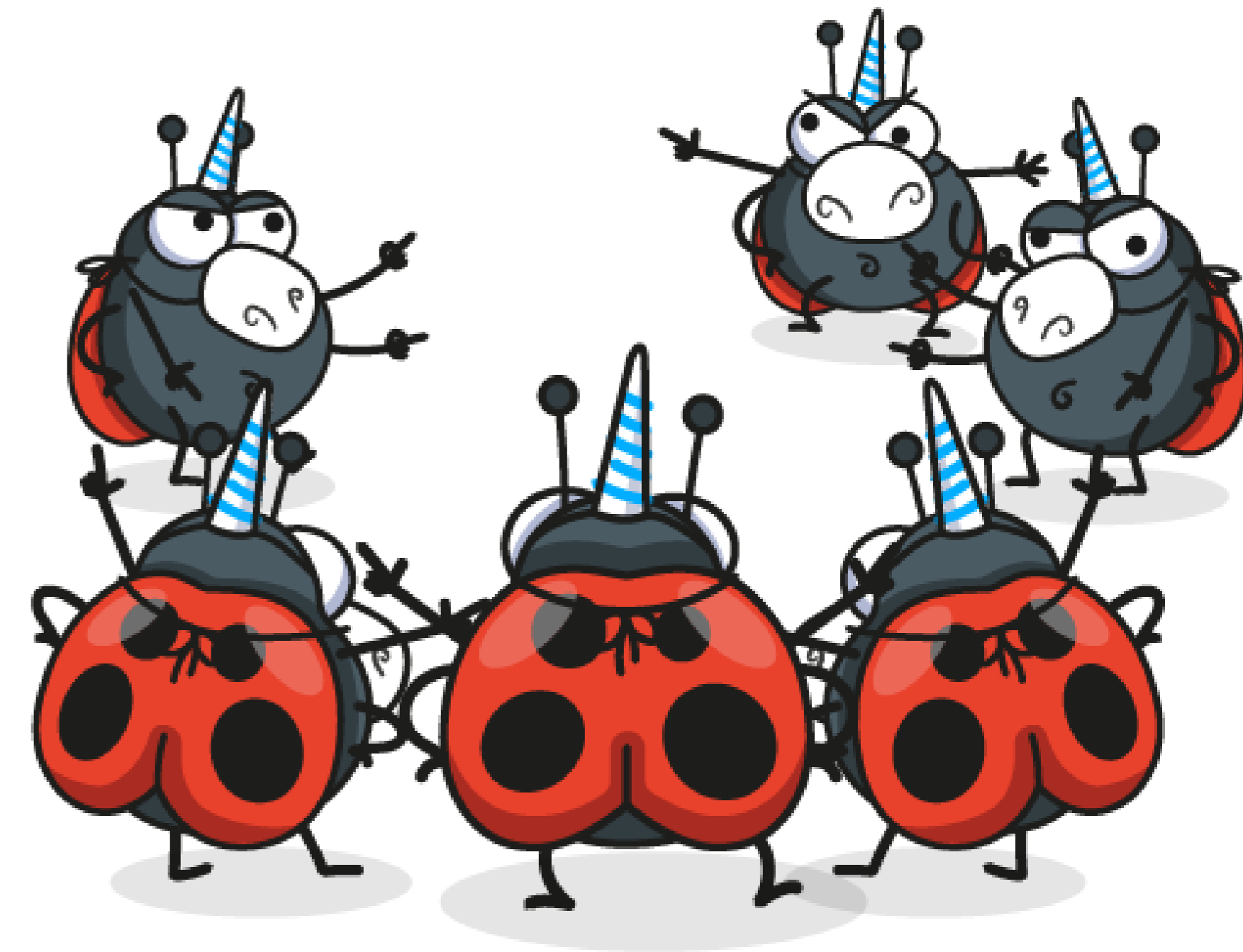
# КЛАССИФИКАЦИЯ КРИТИЧЕСКИХ ОШИБОК

- Стандарт разделяет критические ошибки на типы в зависимости от языка программирования (п. 6.3).



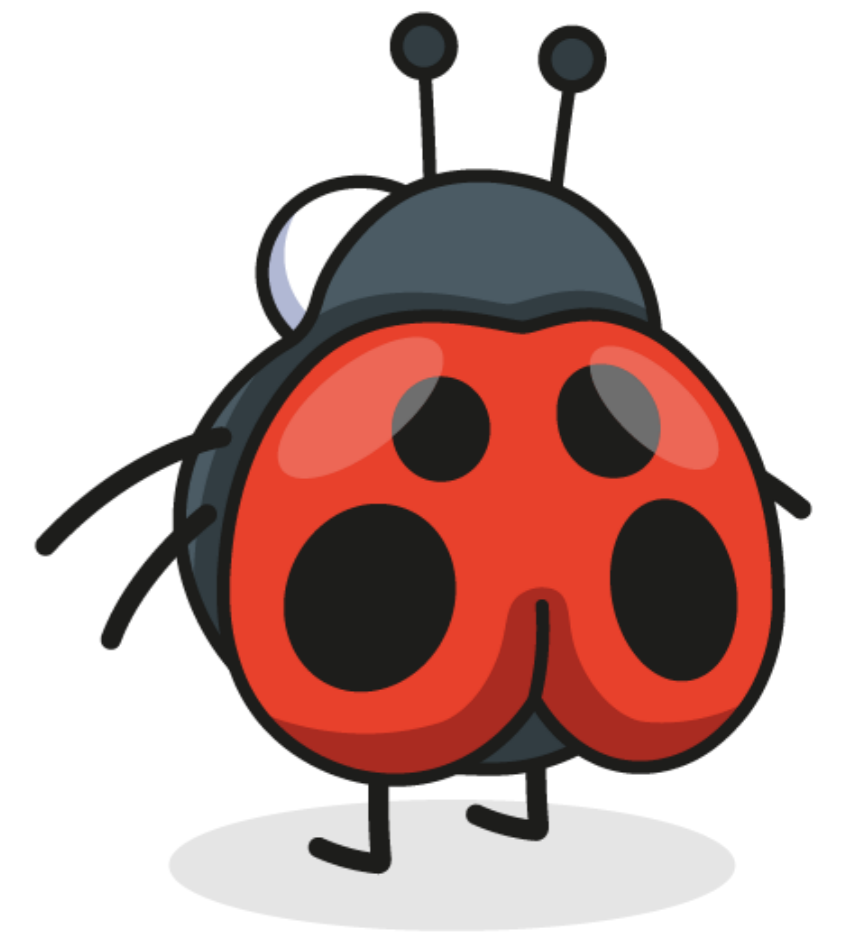
# КЛАССИФИКАЦИЯ КРИТИЧЕСКИХ ОШИБОК

- Стандарт разделяет критические ошибки на типы в зависимости от языка программирования (п. 6.3).
- Список критических ошибок для конкретного языка может дополняться (п. 6.5).



# КЛАССИФИКАЦИЯ КРИТИЧЕСКИХ ОШИБОК

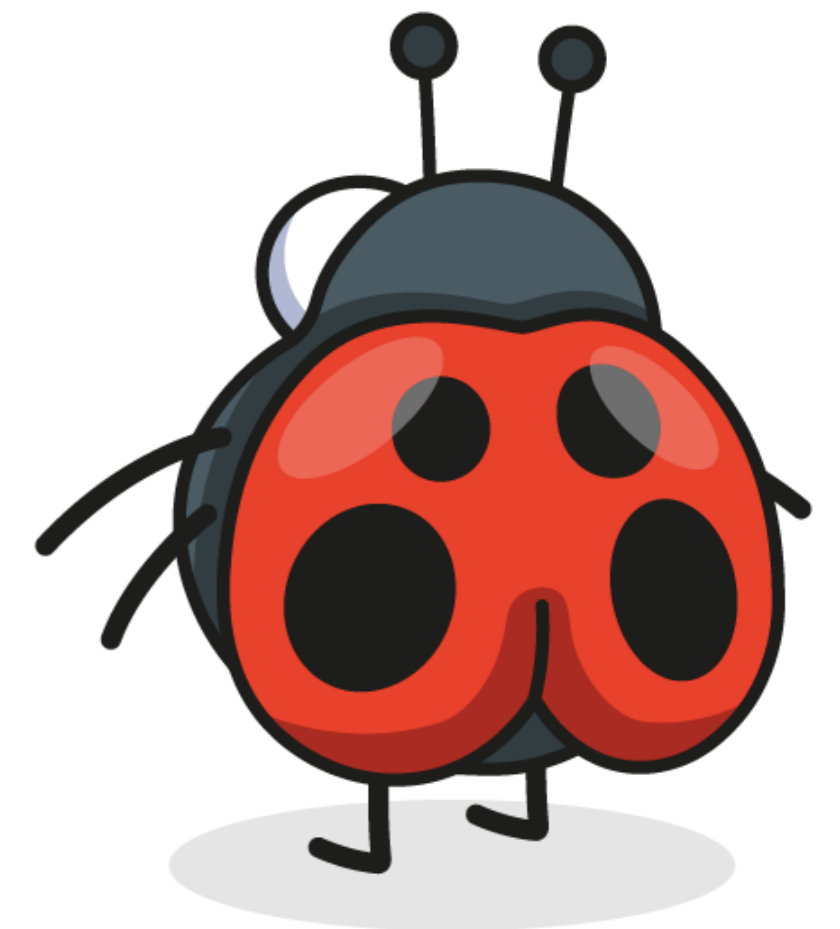
Для компилируемых языков:



# КЛАССИФИКАЦИЯ КРИТИЧЕСКИХ ОШИБОК

Для компилируемых языков:

- ошибки непроверенного использования чувствительных данных;



```
static const char *basic_gets(int *cnt)
{
    ....
    int c = getchar();
    if (c < 0) {
        if (fgets(command_buf, sizeof(command_buf) - 1, stdin)
            != command_buf) {
            break;
        }
        command_buf[strlen(command_buf)-1] = '\0';
        break;
    }
    ....
}
```



```
static const char *basic_gets(int *cnt)
{
    ....
    int c = getchar();
    if (c < 0) {
        if (fgets(command_buf, sizeof(command_buf) - 1, stdin)
            != command_buf) {
            break;
        }
        command_buf[strlen(command_buf)-1] = '\0';
        break;
    }
    ....
}
```



```
static const char *basic_gets(int *cnt)
{
    ....
    int c = getchar();
    if (c < 0) {
        if (fgets(command_buf, sizeof(command_buf) - 1, stdin)
            != command_buf) {
            break;
        }
        command_buf[strlen(command_buf)-1] = '\0';
        break;
    }
    ....
}
```



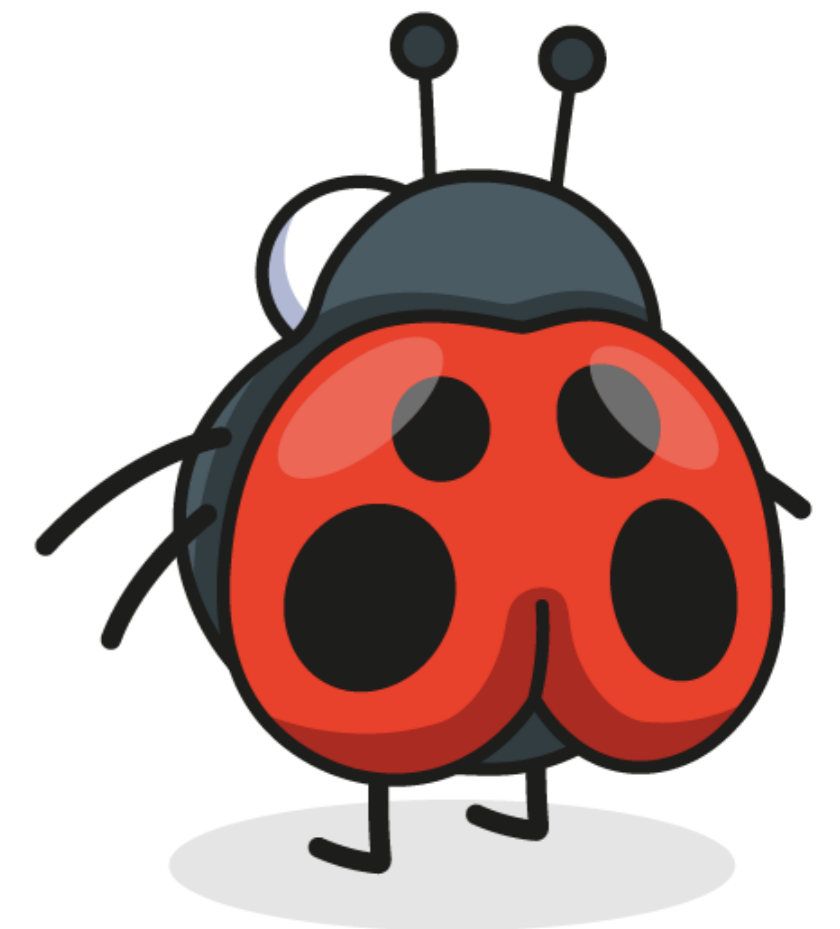
Предупреждение PVS-Studio:

**V1010** Unchecked tainted data is used in index: 'strlen(command\_buf)'.

# КЛАССИФИКАЦИЯ КРИТИЧЕСКИХ ОШИБОК

Для компилируемых языков:

- ошибки непроверенного использования чувствительных данных;
- ошибки целочисленного переполнения и некорректного совместного использования знаковых и беззнаковых чисел;



```
int NumElts = Mask.size();  
ScaledMask.assign(  
    static_cast<size_t>(NumElts * Scale),  
    -1  
);
```



```
int NumElts = Mask.size();  
ScaledMask.assign(  
    static_cast<size_t>(NumElts * Scale),  
    -1  
);
```



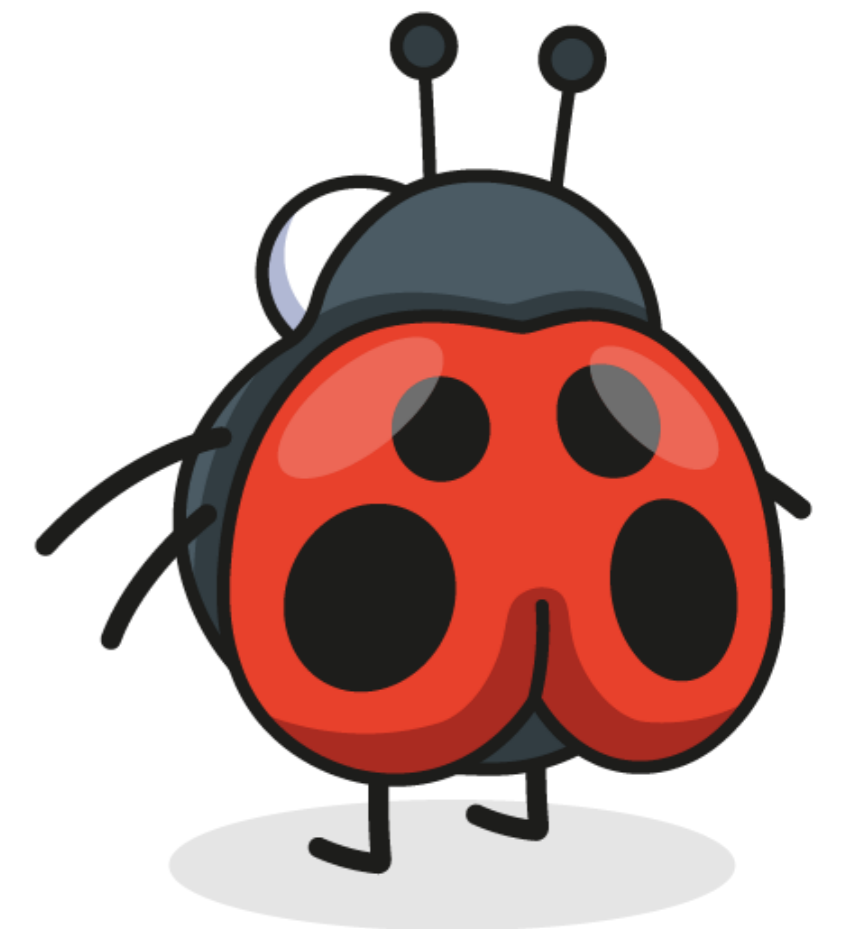
Предупреждение PVS-Studio:

**V1028** Possible overflow. Consider casting operands of the 'NumElts \* Scale' operator to the 'size\_t' type, not the result

# КЛАССИФИКАЦИЯ КРИТИЧЕСКИХ ОШИБОК

Для компилируемых языков:

- ошибки непроверенного использования чувствительных данных;
- ошибки целочисленного переполнения и некорректного совместного использования знаковых и беззнаковых чисел;
- ошибки переполнения буфера;



```
...  
wcsncpy(  
    psci->wszTitle,  
    ColumnInfoTable[dwIndex].wszTitle,  
    (sizeof(psci->wszTitle) - 1)  
);  
return S_OK;
```



```
••••  
wcsncpy(  
    psci->wszTitle,  
    ColumnInfoTable[dwIndex].wszTitle,  
    (sizeof(psci->wszTitle) - 1)  
);  
return S_OK;
```



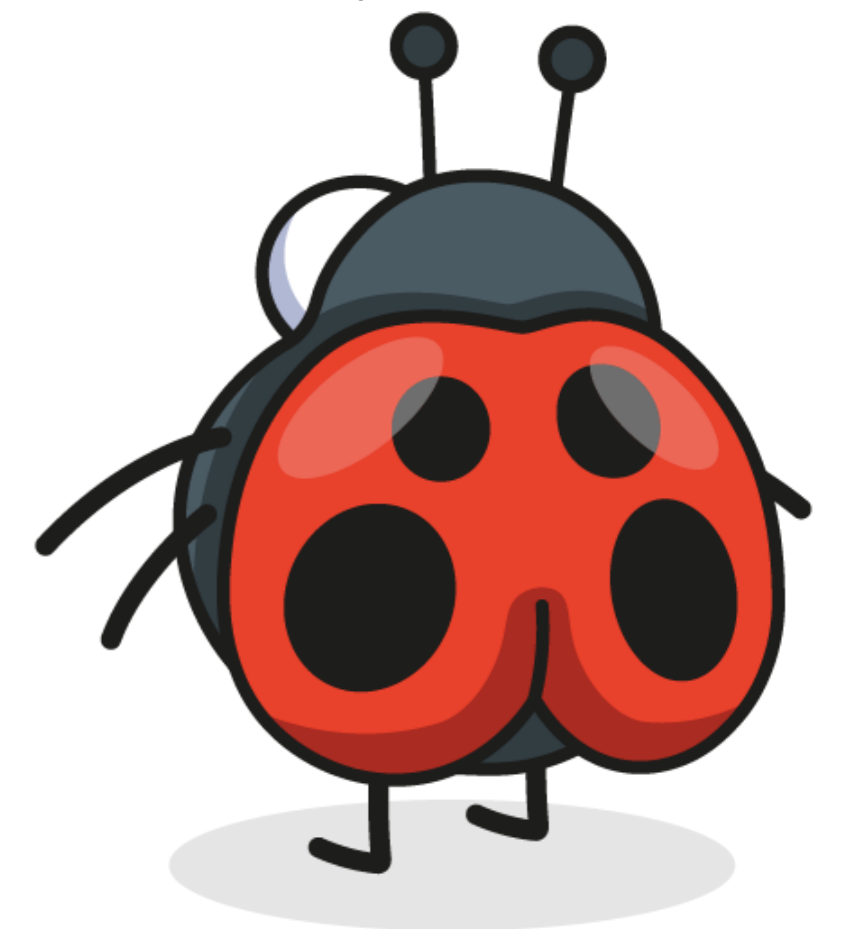
Предупреждение PVS-Studio:

**V512** A call of the 'wcsncpy' function will lead to overflow of the buffer 'psci->wszTitle'.

# КЛАССИФИКАЦИЯ КРИТИЧЕСКИХ ОШИБОК

Для компилируемых языков:

- ошибки непроверенного использования чувствительных данных;
- ошибки целочисленного переполнения и некорректного совместного использования знаковых и беззнаковых чисел;
- ошибки переполнения буфера;
- ошибки некорректного использования системных процедур и интерфейсов, связанных с обеспечением информационной безопасности;



....

```
if(!CryptAcquireContext(  
    &m_hProvider, 0, 0,  
    PROV_RSA_FULL,  
    CRYPT_VERIFYCONTEXT)  
)  
    throw OS_RNG_Err("CryptAcquireContext");
```

••••

```
if(!CryptAcquireContext(  
    &m_hProvider, 0, 0,  
    PROV_RSA_FULL,  
    CRYPT_VERIFYCONTEXT)  
)  
    throw OS_RNG_Err("CryptAcquireContext");
```

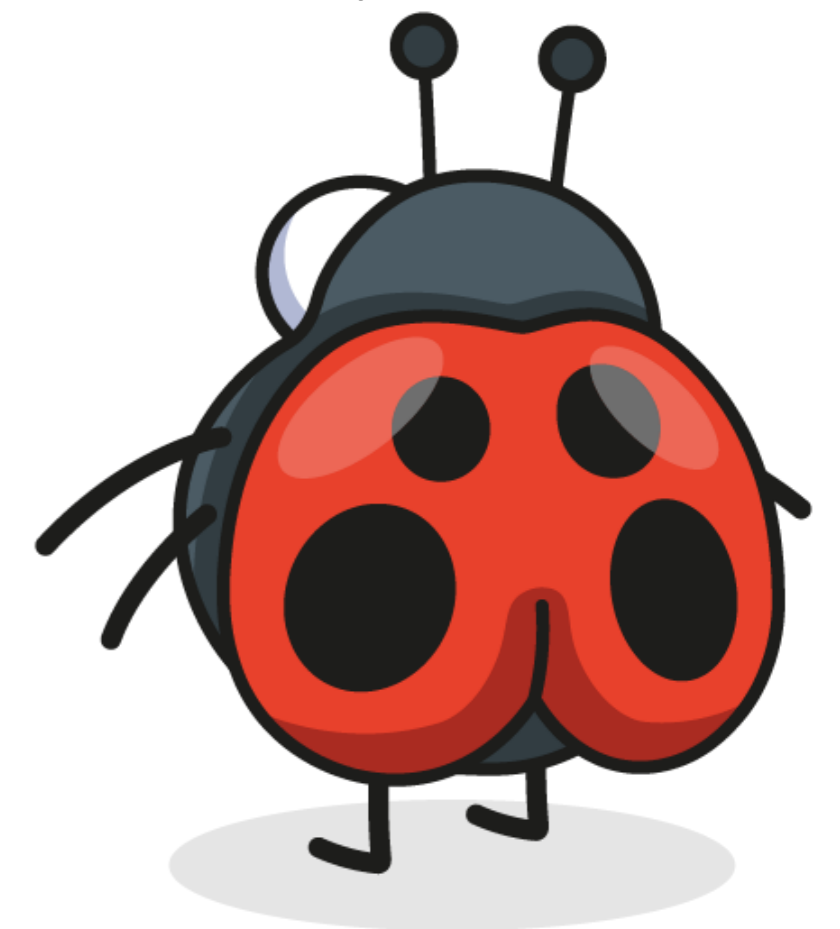
Предупреждение PVS-Studio:

**V1109** The 'CryptAcquireContextA' function is deprecated. Consider switching to an equivalent newer function.

# КЛАССИФИКАЦИЯ КРИТИЧЕСКИХ ОШИБОК

Для компилируемых языков:

- ошибки непроверенного использования чувствительных данных;
- ошибки целочисленного переполнения и некорректного совместного использования знаковых и беззнаковых чисел;
- ошибки переполнения буфера;
- ошибки некорректного использования системных процедур и интерфейсов, связанных с обеспечением информационной безопасности;
- ошибки при работе с многопоточными примитивами.



```
private List<DBTTaskRun> runs;
```

```
....
```

```
private void loadRunsIfNeeded() {
```

```
    if (runs == null) {
```

```
        synchronized (this) {
```

```
            if (runs == null) {
```

```
                runs = new ArrayList<>(loadRunStatistics());
```

```
            }
```

```
        }
```

```
    }
```

```
}
```



```
private List<DBTTaskRun> runs;
```

```
....
```

```
private void loadRunsIfNeeded() {  
    if (runs == null) {  
        synchronized (this) {  
            if (runs == null) {  
                runs = new ArrayList<>(loadRunStatistics());  
            }  
        }  
    }  
}
```



Предупреждение PVS-Studio:

[V6082](#) Unsafe double-checked locking. The field should be declared as volatile.

# КЛАССИФИКАЦИЯ КРИТИЧЕСКИХ ОШИБОК

Дополнительно для C и C++:



# КЛАССИФИКАЦИЯ КРИТИЧЕСКИХ ОШИБОК

Дополнительно для C и C++:

- ошибки разыменования нулевого указателя;



# КЛАССИФИКАЦИЯ КРИТИЧЕСКИХ ОШИБОК

Дополнительно для C и C++:

- ошибки разыменовывания нулевого указателя;
- ошибки деления на ноль;



# КЛАССИФИКАЦИЯ КРИТИЧЕСКИХ ОШИБОК

Дополнительно для C и C++:

- ошибки разыменованного нулевого указателя;
- ошибки деления на ноль;
- ошибки управления динамической памятью;



# КЛАССИФИКАЦИЯ КРИТИЧЕСКИХ ОШИБОК

Дополнительно для C и C++:

- ошибки разыменованя нулевого указателя;
- ошибки деления на ноль;
- ошибки управления динамической памятью;
- ошибки использования форматной строки;



# КЛАССИФИКАЦИЯ КРИТИЧЕСКИХ ОШИБОК

Дополнительно для C и C++:

- ошибки разыменованя нулевого указателя;
- ошибки деления на ноль;
- ошибки управления динамической памятью;
- ошибки использования форматной строки;
- ошибки использования неинициализированных переменных;



# КЛАССИФИКАЦИЯ КРИТИЧЕСКИХ ОШИБОК

Дополнительно для C и C++:

- ошибки разыменованя нулевого указателя;
- ошибки деления на ноль;
- ошибки управления динамической памятью;
- ошибки использования форматной строки;
- ошибки использования неинициализированных переменных;
- ошибки утечек памяти, незакрытых файловых дескрипторов и дескрипторов сетевых соединений.



# ВНЕДРЕНИЕ SAST

Достаём бубны?



## ПОДГОТОВИТЕЛЬНЫЙ ЭТАП (п. 5.2-5.3)

Необходимо выбрать анализатор.



## ПОДГОТОВИТЕЛЬНЫЙ ЭТАП (п. 5.2-5.3)

Необходимо выбрать анализатор.

Критерии:

- Поддержка современных стандартов языков программирования;
- Поддержка используемых в проекте систем сборки;
- Исчерпывающая документация.



## 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-MS15-C

Описание

Пример  
ошибочного кода

Примеры  
исправления

Сопоставление с SEI  
CERT, OWASP, CWE

## ПОДГОТОВИТЕЛЬНЫЙ ЭТАП (п. 5.2-5.3)

Необходимо выбрать анализатор.

Критерии:

- Поддержка современных стандартов языков программирования;
- Поддержка используемых в проекте систем сборки;
- Исчерпывающая документация.

Можно использовать несколько анализаторов.



## ПОДГОТОВИТЕЛЬНЫЙ ЭТАП (п. 5.2-5.3)

Необходимо настроить сборочную среду и среду анализа ПО.



## ПОДГОТОВИТЕЛЬНЫЙ ЭТАП (п. 5.2-5.3)

Необходимо настроить сборочную среду и среду анализа ПО.

Анализ всего кода в выбранной среде должен проходить быстрее чем за двое суток.



## ПОДГОТОВИТЕЛЬНЫЙ ЭТАП (п. 5.2-5.3)

Необходимо настроить сборочную среду и среду анализа ПО.

Анализ всего кода в выбранной среде должен проходить быстрее чем за двое суток.

Библиотеки – тоже ваш код.



## НАЧАЛЬНЫЙ ЭТАП (п. 5.4-5.5)

Необходимо настроить анализатор, выбрав предупреждения, соответствующие критическим ошибкам.



## НАЧАЛЬНЫЙ ЭТАП (п. 5.4-5.5)

Необходимо настроить анализатор, выбрав предупреждения, соответствующие критическим ошибкам.

После настройки проводится первичный анализ.



## НАЧАЛЬНЫЙ ЭТАП (п. 5.4-5.5)

Необходимо настроить анализатор, выбрав предупреждения, соответствующие критическим ошибкам.

После настройки проводится первичный анализ.

Полученные предупреждения размечаются и составляют начальную базу предупреждений.



## НАЧАЛЬНЫЙ ЭТАП (п. 5.4-5.5)

Необходимо настроить анализатор, выбрав предупреждения, соответствующие критическим ошибкам.

После настройки проводится первичный анализ.

Полученные предупреждения размечаются и составляют начальную базу предупреждений.

При разметке предупреждения делятся на:

- Истинные;
- Ложные;
- Истинные, но не требующие исправления.



# РЕГУЛЯРНЫЙ АНАЛИЗ (П. 5.6-5.13)

Накопление непроанализированных изменений приводит к:



## РЕГУЛЯРНЫЙ АНАЛИЗ (П. 5.6-5.13)

Накопление непроанализированных изменений приводит к:

- ухудшению качества проводимой экспертизы результатов анализа;



## РЕГУЛЯРНЫЙ АНАЛИЗ (П. 5.6-5.13)

Накопление непроанализированных изменений приводит к:

- ухудшению качества проводимой экспертизы результатов анализа;
- увеличению длительности проводимой экспертизы;



## РЕГУЛЯРНЫЙ АНАЛИЗ (П. 5.6-5.13)

Накопление непроанализированных изменений приводит к:

- ухудшению качества проводимой экспертизы результатов анализа;
- увеличению длительности проводимой экспертизы;
- усложнению исправления ошибок.



## РЕГУЛЯРНЫЙ АНАЛИЗ (П. 5.6-5.13)

Накопление непроанализированных изменений приводит к:

- ухудшению качества проводимой экспертизы результатов анализа;
- увеличению длительности проводимой экспертизы;
- усложнению исправления ошибок.

Для упрощения проведения регулярного анализа можно автоматизировать процесс.



## РЕГУЛЯРНЫЙ АНАЛИЗ (П. 5.6-5.13)

Для всего кода проводить анализ не реже, чем раз в 10 дней после внесения изменений.

Для нового кода проводить анализ сразу после внесения изменений.



## РЕГУЛЯРНЫЙ АНАЛИЗ (П. 5.6-5.13)

Для всего кода проводить анализ не реже, чем раз в 10 дней после внесения изменений.

Для нового кода проводить анализ сразу после внесения изменений.

Просмотр и разметка предупреждений:

- Для нового кода не позднее, чем через 3 дня
- Для всего кода не позднее, чем через 10 дней



# ИТОГИ

- ГОСТ устанавливает список **критических ошибок**, на которые **необходимо обращать внимание в первую очередь**



# ИТОГИ

- ГОСТ устанавливает список **критических ошибок**, на которые **необходимо обращать внимание в первую очередь**
- Используемый анализатор должен поддерживать последние стандарты языков программирования и используемые в проекте сборочные системы, обладать исчерпывающей документацией



# ИТОГИ

- ГОСТ устанавливает список **критических ошибок**, на которые **необходимо обращать внимание в первую очередь**
- Используемый анализатор должен поддерживать последние стандарты языков программирования и используемые в проекте сборочные системы, обладать исчерпывающей документацией
- Проводить статический анализ, обрабатывать результаты и исправлять ошибки нужно **регулярно**



# QIA



@feelindex



filatov@pvs-studio.ru

