АВТОМАТИЗИРОВАННАЯ СИСТЕМА
ЦЕНТР КИБЕРБЕЗОПАСНОСТИ
Руководство Пользователя

Методика выявления уязвимостей и недекларированных возможностей в программном обеспечении на языке Java с применением Svace

Москва 2023

Аннотация

Документ является Руководством пользователя по применению «Методики выявления уязвимостей и недекларированных возможностей в программном обеспечении на языке Java с применением Svace». В документе описаны общие понятия и определения, используемые при анализе, подготовка к работе с сервисом, описание функциональных возможностей статического анализа при помощи Svace и Svacer, рекомендации по освоению. В том числе описана методика выявления уязвимостей в прикладном ПО, которая включает в себя два основных этапа: обработка ветки с использованием Svace, Svacer и разметка результатов обработки ветки с целью соотнесения правильных дефектов к типам уязвимостей согласно федеральной службе по техническому и экспортному контролю. Данный документ нужен для обучению пользованию инструментами Svace и Svacer, поиску и устранению уязвимостей программ на языке Java.

Перед работой пользователя с сервисом рекомендуется внимательно ознакомиться с настоящим руководством. Документ разработан с учетом Национального стандарта Российской Федерации ГОСТ Р 59795—2021 «Информационные технологии. Комплекс стандартов на автоматизированные системы. Автоматизированные системы. Требования к содержанию документов».

Содержание

1. Термины и определения

	Сокращение/Термин	Описание
	Авторизация	Предоставление определённому лицу или группе лиц прав на выполнение определённых действий; а также процесс проверки (подтверждения) данных прав при попытке выполнения этих действий
	ГОСТ Р	Государственный стандарт Российской Федерации - стандарт, принятый Государственным комитетом Российской Федерации по стандартизации, метрологии и сертификации (Госстандарт России) или Государственным комитетом Российской Федерации по жилищной и строительной политике (Госстрой России).
	Операционная система (ОС)	Программное обеспечение, управляющее аппаратным обеспечением, предоставляющее абстрактный программный интерфейс для взаимодействия с ним и занимающееся распределением предоставляемых ресурсов, в том числе между прикладными программами.
	IP- адрес	Уникальный числовой идентификатор устройства в компьютерной сети, работающей по протоколу IP.
	Персональный компьютер (ПК)	Персональная электронно-вычислительная машина) – однопользовательская (предназначенная для использования одним пользователем) ЭВМ, имеющая эксплуатационные характеристики

		бытового прибора и универсальные функциональные возможности
	Пользователь	Авторизованный пользователь Сервиса
	Сервис	Сервис для выявления уязвимостей и недекларированных возможностей в программном обеспечении
	Учетная запись	Данные, однозначно идентифицирующие пользователя, и используемые для аутентификации пользователя на сайте.
	Разметка	Проставление статуса маркера с опциональным добавлением комментария
0	Маркер	Предупреждение от Svace с информацией о позиции в исходном файле
1	Svacer	Сервер хранения и обработки результатов работы статического анализатора Svace
2	Дефект/ ошибка программы	Конструкция или набор конструкций в исходном коде программы, наличие которой указывает на возможность реализации угроз безопасности информации и/или на ошибку реализованного в программе алгоритма
3	Ложноположительное срабатывание (ошибка первого рода)	Выданное статическим анализатором предупреждение, которое указывает на конструкцию или несколько конструкций в программе – возможно, в

		совокупности с некоторым множеством возможных значений переменных программы – не содержащих ошибку, которая может реализоваться в ходе выполнения программы
4	Ложноотрицательное срабатывание (ошибка второго рода)	Отсутствие предупреждения об ошибке со стороны статического анализатора в ситуации наличия заведомо известной ошибки в программе, которая способна реализоваться в ходе выполнения программы
5	Статический анализ	Вид работ по инструментальному исследованию программы, основанный на анализе исходных кодов в режиме, не предусматривающем реального выполнения кода, и выполняемый для определения свойств программы
6	Детектор	Предупреждение от Svace с информацией о типе ошибке, ее критичности и всеми сопутствующими данными

2. Общая часть

2.1. Область применения

Svace применяется для выполнения статического анализа исходного кода, направленного на выявление потенциальных уязвимостей кода и недекларированных возможностей в исходном коде согласно требованиям «Методики выявления уязвимостей и недекларированных возможностей в программном обеспечении» (ФСТЭК, 2020 год) и ГОСТ Р 56939-2016.

Выполнение статического анализа производится на исходном коде программ, написанных на языке программирования Java.

Svace обеспечивает проведение статического анализа исходных кодов методами автоматизированного анализа исходного кода на уровне синтаксического дерева, статического символьного выполнения чувствительного к контексту и путям выполнения. Обеспечивает полный контроль над сборкой ПО, в том числе учет параметров работы компилятора и статического компоновщика.

2.2. Перечень эксплуатационной документации, с которой необходимо ознакомиться пользователю

Для работы с сервисом Svace пользователю необходимо ознакомиться со следующими документами - Руководством администратора Svace и Руководством оператора Svace, а также, с настоящим документом - Руководством пользователя для выполнения статического анализа исходного кода в Svace, направленного на выявление потенциальных уязвимостей кода и недекларированных возможностей в исходном коде на языке Java.

3. Подготовка к работе

3.1. Порядок загрузки программ и данных

Подробный интерфейс Сервиса хранения и управления результатами анализа Svacer описан в разделе 3 руководства оператора Svace.

Запустить браузер и открыть веб-интерфейс Сервиса хранения и управления результатами анализа (далее – svacer). Ввести логин и пароль учётной записи в соответствующие поля и нажать кнопку «Войти» (Рисунок 9).

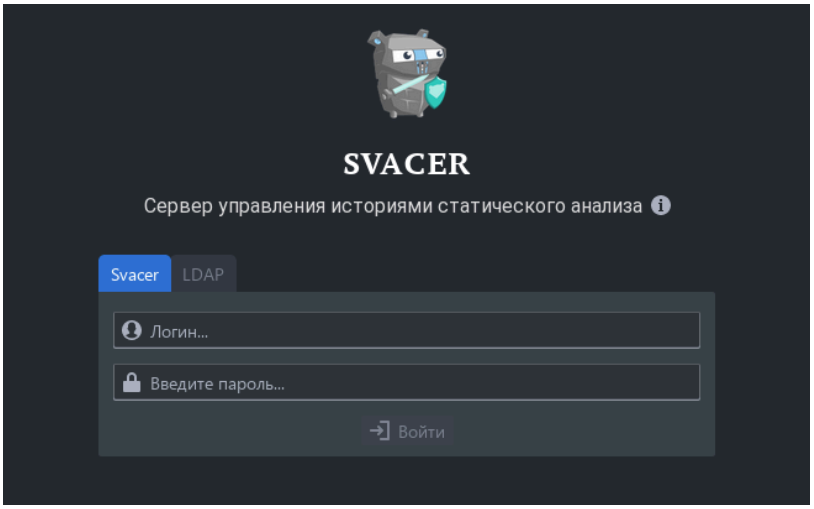


Рисунок 9 – Окно авторизации

После успешной авторизации загрузится основное окно Svacer (Рисунок 10).

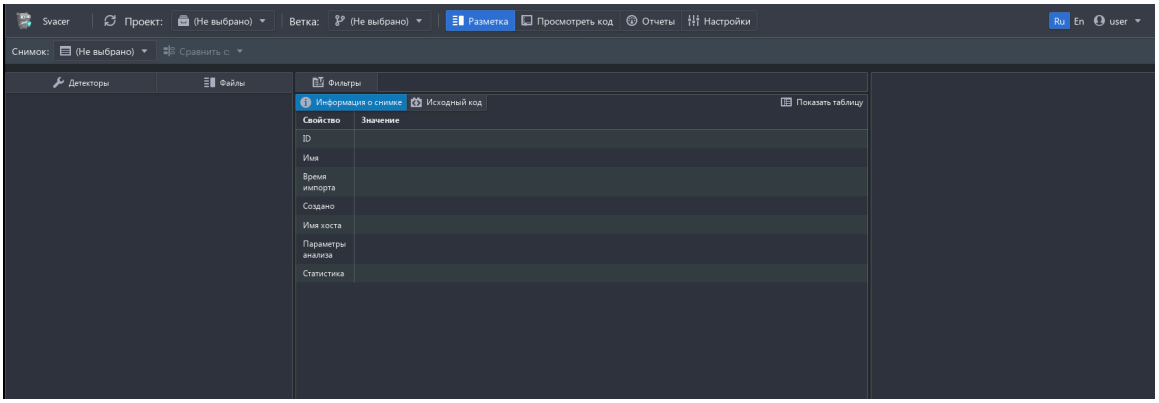


Рисунок 10 – Основное окно сервиса хранения и управления результатами анализа

Перейти в окно интерфейса Разметка нажав кнопку Разметка на панели навигации svacer (Рисунок 11).

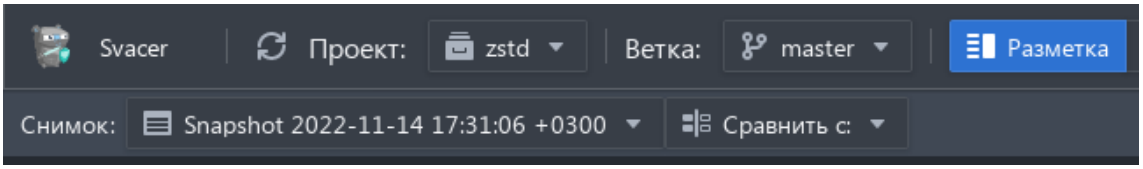


Рисунок 11 – Интерфейс кнопки «Разметка»

Нажать на Выпадающий список снимков на панели выбора снимков и выбрать снимок для анализа. По умолчанию после выбора проекта автоматически выбирается ветка master и самый свежий снимок (Рисунок 12).

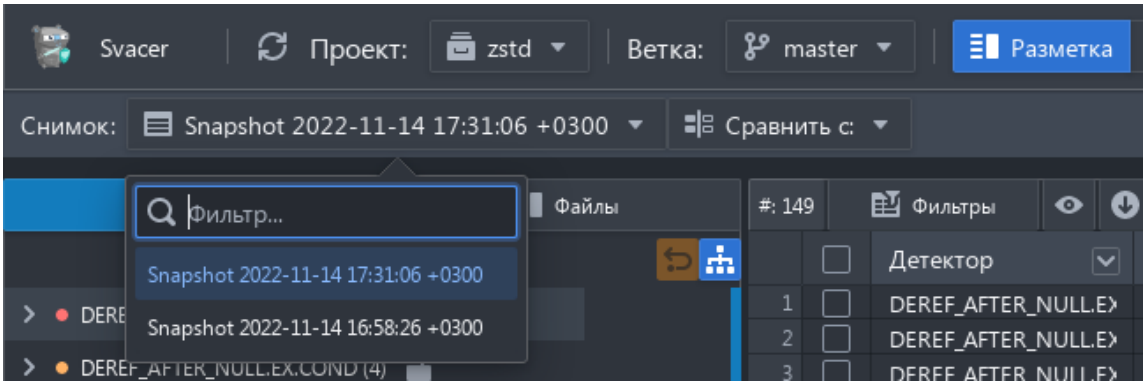


Рисунок 12 – Выбор снимка

В панели Детекторы переключить отображение детекторов в древовидный вид кнопкой Группировать (серьезность) (Рисунок 13).

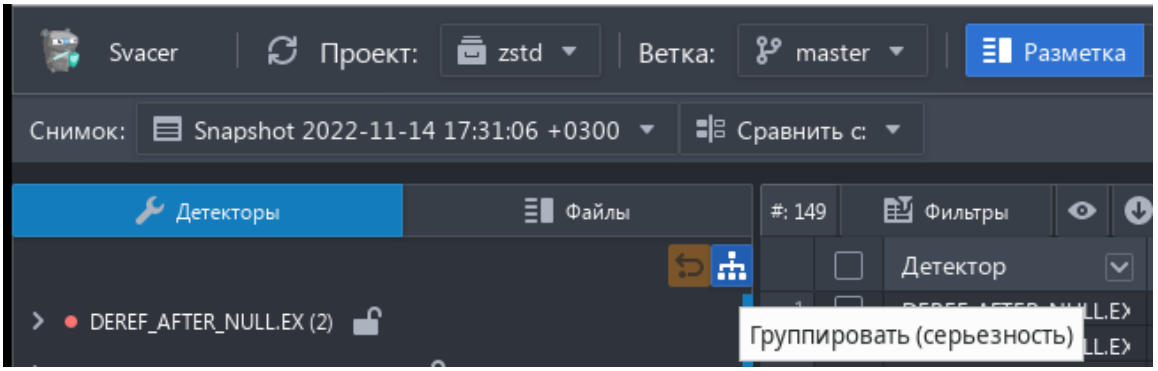


Рисунок 13 – Кнопка Группировать

На Рисунке 14 представлен интерфейс пользователя по просмотру и разметке результатов анализа, доступный через браузер при запущенном сервере просмотра результатов Svacer.

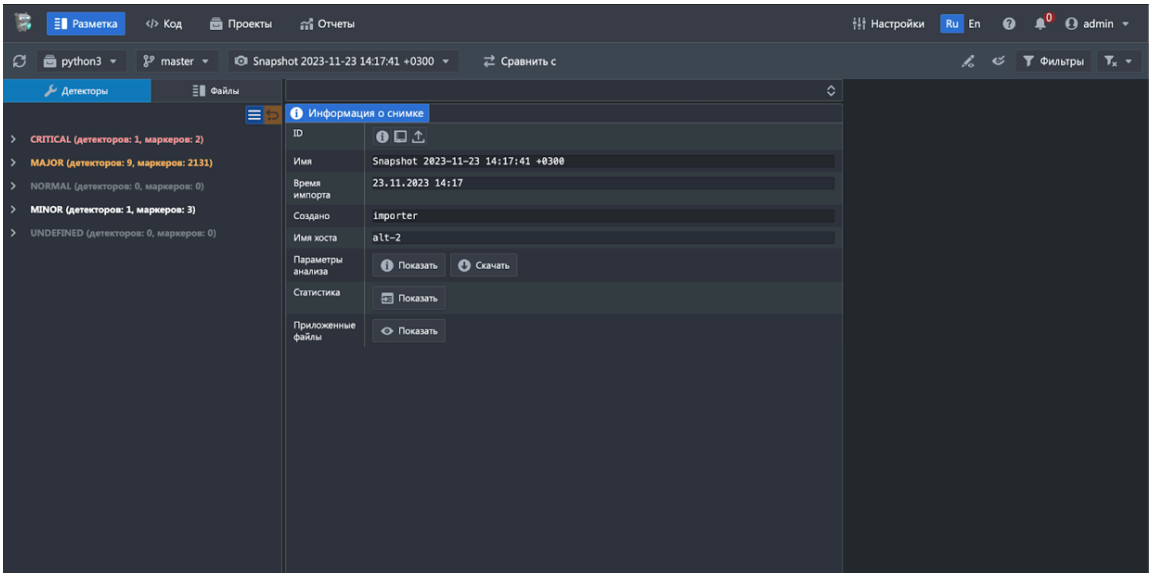


Рисунок 14 – Интерфейс по просмотру и разметке результатов анализа

При нажатии на верхней панели кнопки «Код» (Рисунок 15) пользователь может полностью ознакомиться с кодом и структурой проекта.

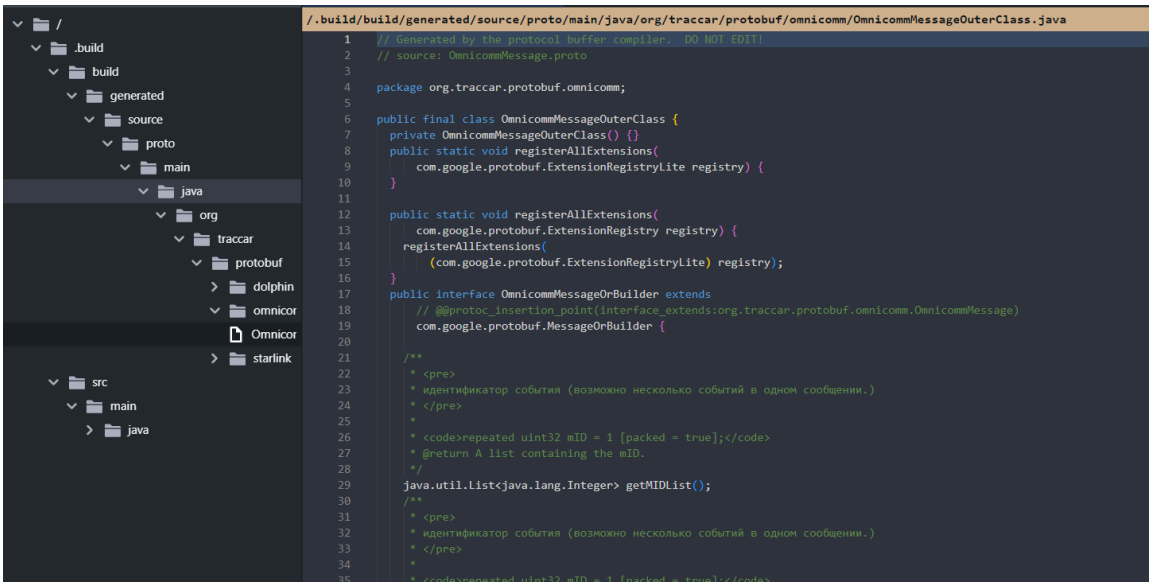
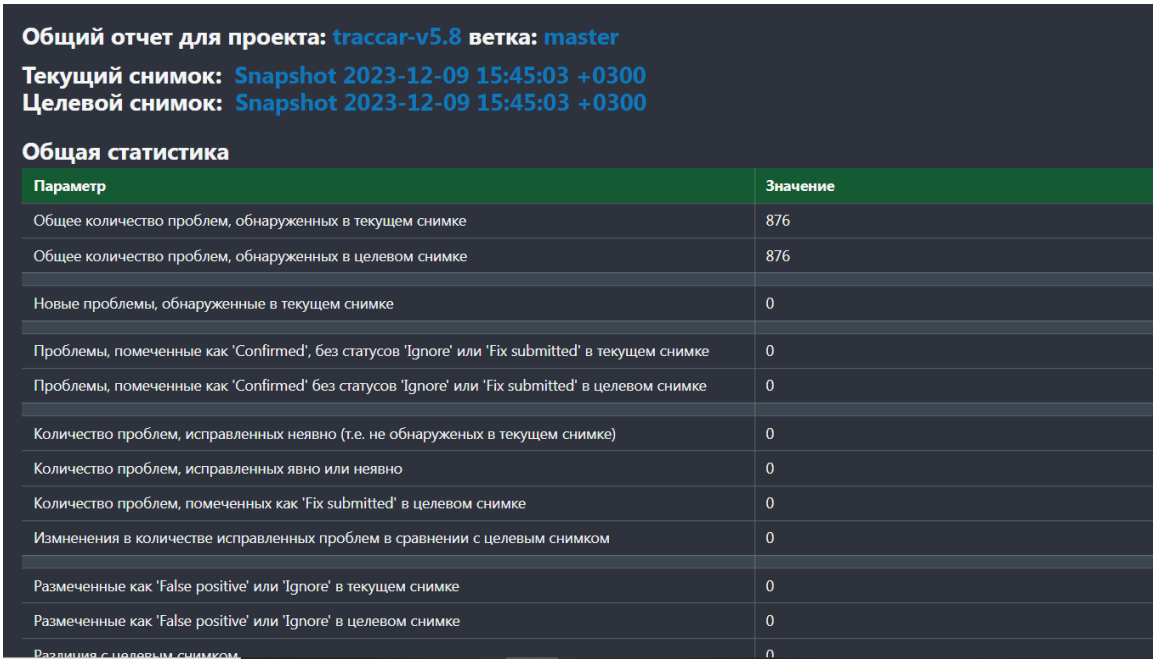


Рисунок 15 – Кнопка Код

При нажатии на верхней панели кнопки «Отчеты» (Рисунок 16) пользователь может составить отчет на основе полученных снимков после анализа.

После выбора снимков, перед пользователем появляется статистика выбранных проектов.



Общий отчет для проекта: traccar-v5.8 ветка: master

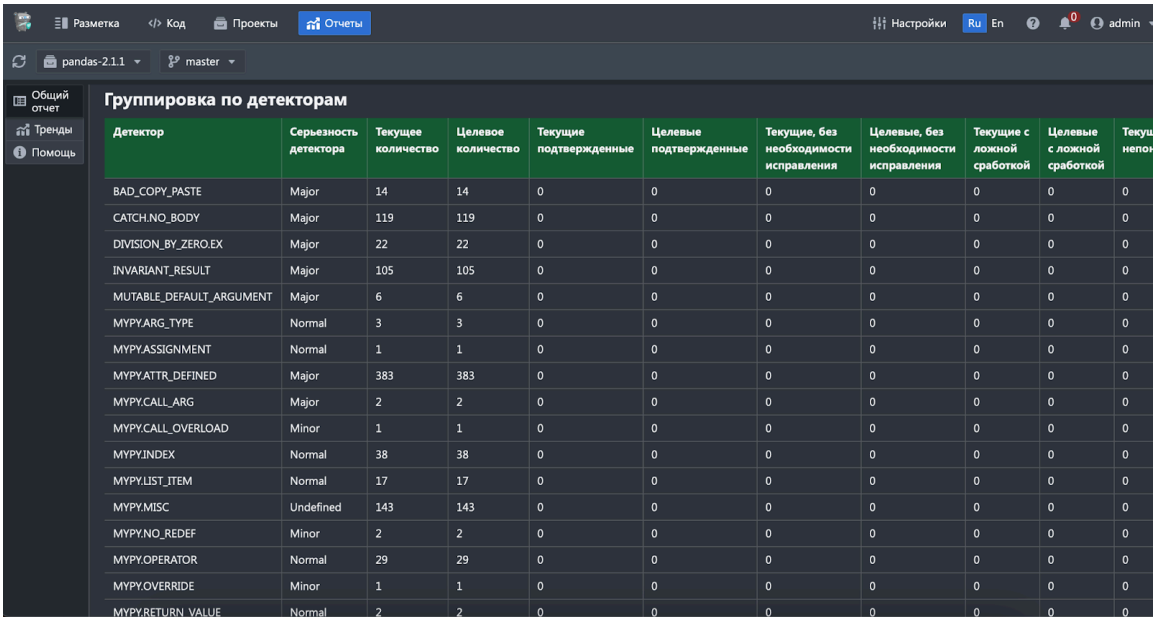
Текущий снимок: Snapshot 2023-12-09 15:45:03 +0300
Целевой снимок: Snapshot 2023-12-09 15:45:03 +0300

Общая статистика

Параметр	Значение
Общее количество проблем, обнаруженных в текущем снимке	876
Общее количество проблем, обнаруженных в целевом снимке	876
Новые проблемы, обнаруженные в текущем снимке	0
Проблемы, помеченные как 'Confirmed', без статусов 'Ignore' или 'Fix submitted' в текущем снимке	0
Проблемы, помеченные как 'Confirmed' без статусов 'Ignore' или 'Fix submitted' в целевом снимке	0
Количество проблем, исправленных неявно (т.е. не обнаруженых в текущем снимке)	0
Количество проблем, исправленных явно или неявно	0
Количество проблем, помеченных как 'Fix submitted' в целевом снимке	0
Измнения в количестве исправленных проблем в сравнении с целевым снимком	0
Размеченные как 'False positive' или 'Ignore' в текущем снимке	0
Размеченные как 'False positive' или 'Ignore' в целевом снимке	0
Различия с целевым снимком	0

Рисунок 16 – Интерфейс при нажатии кнопки «Отчеты»

Информация о типах и видах ошибок представлена на Рисунке 17.



Разметка | Код | Проекты | Отчеты | Настройки | Ru | En | admin

pandas-2.1.1 | master

Общий отчет | Тренды | Помощь

Группировка по детекторам

Детектор	Серьезность детектора	Текущее количество	Целевое количество	Текущие подтвержденные	Целевые подтвержденные	Текущие, без необходимости исправления	Целевые, без необходимости исправления	Текущие с ложной сработкой	Целевые с ложной сработкой	Текущ непо
BAD_COPY_PASTE	Major	14	14	0	0	0	0	0	0	0
CATCH.NO_BODY	Major	119	119	0	0	0	0	0	0	0
DIVISION_BY_ZERO.EX	Major	22	22	0	0	0	0	0	0	0
INVARIANT_RESULT	Major	105	105	0	0	0	0	0	0	0
MUTABLE_DEFAULT_ARGUMENT	Major	6	6	0	0	0	0	0	0	0
MYPY.ARG_TYPE	Normal	3	3	0	0	0	0	0	0	0
MYPY.ASSIGNMENT	Normal	1	1	0	0	0	0	0	0	0
MYPY.ATTR_DEFINED	Major	383	383	0	0	0	0	0	0	0
MYPY.CALL_ARG	Major	2	2	0	0	0	0	0	0	0
MYPY.CALL_OVERLOAD	Minor	1	1	0	0	0	0	0	0	0
MYPY.INDEX	Normal	38	38	0	0	0	0	0	0	0
MYPY.LIST_ITEM	Normal	17	17	0	0	0	0	0	0	0
MYPY.MISC	Undefined	143	143	0	0	0	0	0	0	0
MYPY.NO_REDEF	Minor	2	2	0	0	0	0	0	0	0
MYPY.OPERATOR	Normal	29	29	0	0	0	0	0	0	0
MYPY.OVERRIDE	Minor	1	1	0	0	0	0	0	0	0
MYPY.RETURN_VALUE	Normal	2	2	0	0	0	0	0	0	0

Рисунок 17 – Типы и виды ошибок в интерфейсе «Отчеты»

4. Функциональные возможности статистического анализа при помощи Svace и Svacer

Статический анализ программ посредством Svace позволяет обнаружить 852 детектора для языка программирования Java. Каждый детектор относится к определенной категории критичности, всего их пять: CRITICAL, MAJOR, NORMAL, MINOR, UNDEFINED.

Критические ошибки (CRITICAL) - ошибки, наиболее сильно вредящие программе. Такие следует устранять в первую очередь, так как они могут привести к неправильной работе программы или ее использованию. Такие ошибки могут привести к критическим проблемам при работе программы как на Go, так и на C/C++, начиная от банального неправильного запуска и заканчивая неограниченными возможностями злоумышленников.

Ошибкам класса MAJOR следует уделить повышенное внимания. Они, скорее всего, приведут к неправильной работе программы на Java.

Ошибки класса NORMAL - ошибки, которые могут не дать о себе знать при идеально верном использовании программы. Однако, при попытке ввести неверные данные или каким-либо другим способом воздействовать на программу, отличным от “эталонного”, может возникнуть неприятный результат, вплоть до остановки программы.

Ошибки класса MINOR - ошибки, которые вряд ли приведут к какому- либо негативному результату, однако при наличии ресурсов их все же стоит устранить.

Ошибки, класс которых не определен (UNDEFINED), а потому к ним нужно относиться с повышенным вниманием.

Таблица 1 – Детекторы Java

Количество обнаруживаемых детекторов во всех категориях критичности в Java	Для категории: “critical”	Для категории: “major”	Для категории: “normal”	Для категории: “minor”	Для категории: “undefined”

852	40	301	83	318	10

Используя статический анализ при помощи Svace и Svacer пользователю, предлагается выявить уязвимости и недекларированные возможности в программном обеспечении по следующей методике.

5. Методика статического анализа при помощи Svace и Svacer

Методика направлена на выявление дефектов, посредством статического анализа, и вследствие уязвимостей в программах написанных на языке программирования Java.

Основные этапы методики:

1) Статический анализ при помощи Svace и Svacer

Этап включает в себя тестирование программы в статическом анализаторе Svace и сохранение результатов (обнаруженных детекторов) в Svacer. После выполнения данного шага в Svacer должен появится сводный отчет с обнаруженными детекторами.

2) Наличие или отсутствие обнаруженных детекторов

При отсутствии обнаруженных детекторов следует окончание анализа, в ином случае переход к следующим этапам.

3) Анализ выявленных детекторов

Этап предполагает просмотр pdf-отчета, результатов на сервере хранения (Svacer). А именно предполагается узнать какие детекторы и из каких категорий критичности были выявлены. Необходимо понять общее состояние программы на данный момент. После выполнения данного шага пользователь обладает информацией детекторы каких категорий критичности были выявлены и в каких участках программы они были обнаружены.

4) Выявление и фиксирование ложных и истинных срабатываний (разметка (см. пункт 6. Разметка предупреждения))

На данном этапе подразумевается работа с pdf-отчетом Svacer, необходимо проверить возможно-ошибочные конструкции программы, изучить рекомендации Svacer по их исправлению и сменить статус обнаруженного детектора, если это необходимо (подробнее см. «Руководство оператора»). После выполнения данного этапа должен быть готов отчет о количестве ошибок первого и второго рода, в котором указано какие дефекты программы действительно обнаружены.

5) Интерпретация дефектов как уязвимостей согласно ФСТЭК

Этап включает в себя соотнесение выявленных дефектов с типами уязвимостей ФСТЭК (см. таб. 3 - Интерпретация обнаруженных детекторов как уязвимостей согласно ФСТЭК). Отметим, что в некоторых случаях детектор может относится к иному от приведённому в таблице типу уязвимости из-за особенностей программы и конструкции, в которой он обнаружен. После выполнения данного шага должен быть подготовлен отчет о наличии уязвимостей в программе.

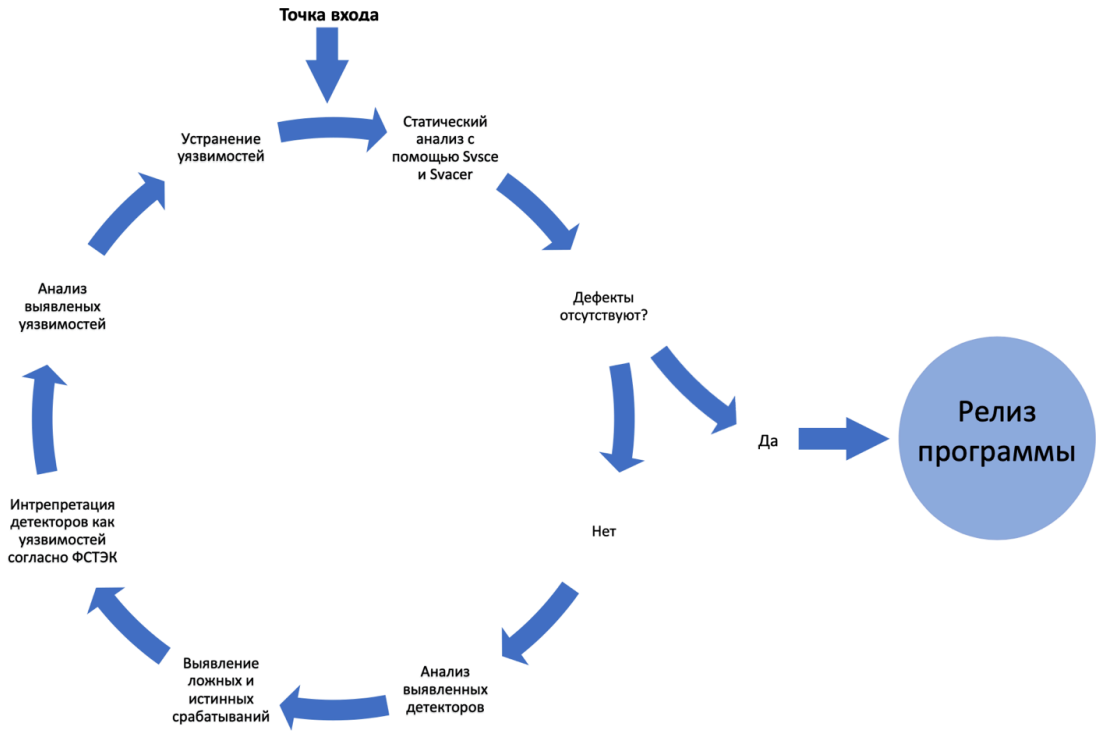
6) Анализ выявленных уязвимостей

Этап предполагает подведение промежуточного итога общего состояния программы или

ветки программы на данный момент, со стороны наличия уязвимостей. По итогу отчет ,созданный в пятом пункте должен быть дополнен описанием каждой обнаруженной уязвимости и рекомендациями по их устранению.

7) Устранение уязвимостей

На данном этапе предполагается доработка программы с последующей проверкой. По итогу в программе должны быть устранены уязвимости и недекларированные возможноости.



Блок-схема 1 – Методика выявления уязвимостей и недекларированных возможностей в программном обеспечении на языке Python с применением Svace

6. Разметка предупреждения

В данном разделе предложено более подробное описание четвертого этапа методики, а именно рекомендации по выявлению ложных и истинных срабатываний.
Чтобы выявить истинные и ложные срабатывания, необходимо:

1. Рассмотреть контекст:

Понять назначение кода и входные данные, которые он обрабатывает.

2. Проверить код вручную:

Внимательно изучить код и подтвердить наличие или отсутствие ошибки.

Проверить наличие признаков ложных срабатываний, таких как использование безопасных методов, правильная обработка ошибок или наличие комментариев, объясняющих отсутствующую уязвимость.

Использовать техники обхода анализа, чтобы попытаться вызвать уязвимость.

Рассмотреть возможность написания тестов, специально предназначенных для проверки отсутствия уязвимости.

3. Использовать дополнительные инструменты:

Применить другие инструменты SAST или динамического анализа программ (DAST) для подтверждения результатов.

Использовать инструменты для отслеживания потока данных, чтобы проверить, может ли ошибочная конструкция действительно привести к нарушению безопасности.

5. Оценить критичность детектора:

Определить потенциал дефекта для фактического компрометации системы.
Учесть влияние дефекта на безопасность пользователей.

6. Посоветоваться с экспертами:

Обсудить результаты анализа с коллегами или экспертами по безопасности, чтобы получить независимое мнение.

7. Документировать

Задокументировать все шаги и результаты процесса выявления истинных и ложных срабатываний(разметки). Это обеспечит прозрачность и позволит сократить будущие затрат анализа.

Выполняя эти шаги, можно повысить точность результатов SAST, уменьшить количество ложных срабатываний и более эффективно сосредоточиться на устранении подлинных дефектов.

7. Рекомендации для анализа результатов тестирования на Java и проведения разметки

Далее будут рассмотрены в качестве примера результаты тестирования. Отчет, получаемый с использованием Svacer характеризует состояние программы на данном этапе. Исходя из данных выдаваемых Svacer выстраивается последующий план работы, анализа и устранения обнаруженных дефектов.

В процессе анализа с помощью Svace и Svacer могут допускаться ошибки первого и второго родов, которые также необходимо устранить.

В левой части (Рисунок 18) приведен пример списка найденных ошибок и типов предупреждений. В скобках после названия типа предупреждения приведено количество найденных предупреждений данного типа.

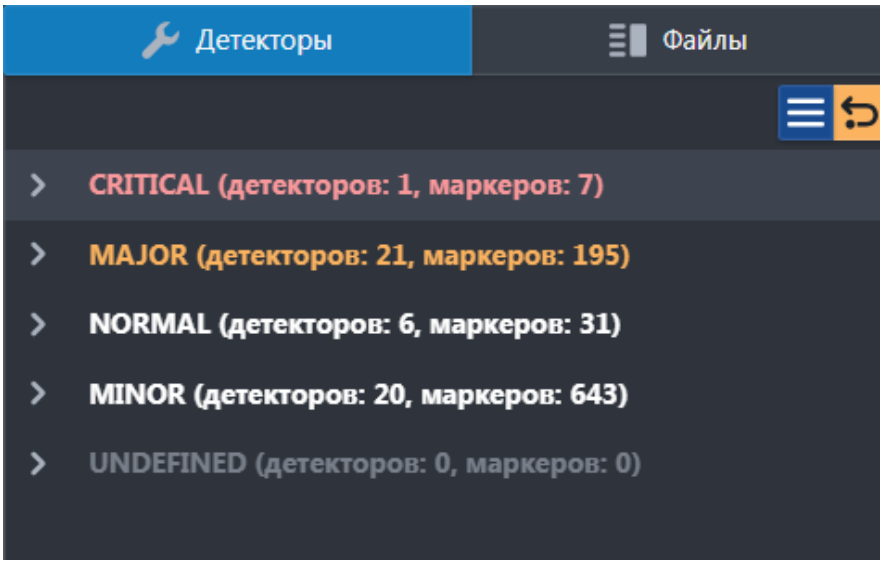


Рисунок 18 – Список всех ошибок в интерфейсе Svacer

При щелчке левой кнопкой мыши по элементу этого списка конкретные предупреждения данного типа показываются в центральной таблице.

7.1.1. CRITICAL ошибки

Критические ошибки (Рисунок 19) - ошибки, наиболее сильно вредящие программе. Такие следует устранять в первую очередь, так как они могут привести к неправильной работе программы или ее использованию. Такие ошибки могут привести к критическим проблемам при работе программы, начиная от банального неправильного запуска и заканчивая неограниченными возможностями злоумышленников.

Рисунок 19 – Обозначение найденного количества CRITICAL ошибок в интерфейсе Svacer

STATIC_OVERFLOW

Детектор STATIC_OVERFLOW предназначен для поиска простых переполнений буфера, которые происходят всегда с константным индексом и размером массива.

STATIC_OVERFLOW.PROC

Этот детектор указывает на переполнение буфера, связанное с использованием системных вызовов

BUFFER_OVERFLOW

Этот детектор указывает на переполнение буфера

BUFFER_OVERFLOW.DFA

Этот детектор указывает на переполнение буфера, связанное с использованием детерминированного конечного автомата

BUFFER_OVERFLOW.LOOP

Этот детектор указывает на переполнение буфера, связанное с использованием циклов

BUFFER_OVERFLOW.ARGV

Этот детектор указывает на переполнение буфера, связанное с обработкой аргументов командной строки

STATIC_OVERFLOW.LOCAL

Этот детектор указывает на переполнение буфера, вызванное неправильной работой с локальными переменными

BUFFER_OVERFLOW.EX

Этот детектор указывает на переполнение буфера, вызванное неправильной работой с исключениями

TAINTED_ARRAY_INDEX.EX

Этот детектор указывает на потенциальные проблемы при обработке исключений, связанные с использованием элементов массива с определенными индексами

TAINTED_ARRAY_INDEX

Этот детектор указывает на потенциальные проблемы работы кода, связанные с использованием элементов массива с определенными индексами

BUFFER_UNDERFLOW

Этот детектор указывает на то, что происходит опустошение буфера. Например, из-за скорости извлечения данных из него большей, чем скорость внесения новых данных

BUFFER_OVERFLOW.PROC

Этот детектор указывает на то, что происходит переполнение буфера, связанное с использованием системных вызовов

DYNAMIC_OVERFLOW

Dynamic_overflow отличается от static_overflow чувствительностью к путям. В остальном этот детектор так же указывает на переполнение буфера. Он делится на две части, первая из которых посвящена глобальным буферам и локально выделенным буферам с известным размером, а вторая – динамически выделенным буферам

DYNAMIC_OVERFLOW.PROC

Этот детектор указывает на переполнение буфера, связанное с использованием системных вызовов

DYNAMIC_OVERFLOW.EX

Этот детектор указывает на переполнение буфера, вызванное неправильной работой с исключениями

TAINTED_DYNAMIC_OVERFLOW

Этот детектор указывает на возможго ошибки при работе с данными, полученными из потенциально небезопасных источников

BUFFER_OVERFLOW.LIB.EX

Этот детектор указывает на то, что при обработке исключений происходит переполнение буфера, связанное с использованием библиотечных функций

OVERFLOW_AFTER_CHECK

Этот детектор указывает на то, что происходит переполнение буфера, происходящее даже после проверки данных

OVERFLOW_AFTER_CHECK.EX

Этот детектор указывает на то, что при обработке исключений происходит переполнение буфера, происходящее даже после проверки

OVERFLOW_UNDER_CHECK

Этот детектор указывает на то, что происходит переполнение буфера, происходящее до проверки данных

OVERFLOW_UNDER_CHECK.LIB

Этот детектор указывает на то, что происходит переполнение буфера, связанное с использованием библиотечных функций и происходящее до проверки данных

OVERFLOW_UNDER_CHECK.PROC

Этот детектор указывает на то, что происходит переполнение буфера, связанное с использованием системных вызовов

DEREF_OF_NULL

Этот детектор указывает на разыменование нулевого указателя

DEREF_OF_NULL.CONST

Этот детектор указывает на разыменование нулевого константного указателя

DEREF_AFTER_NULL

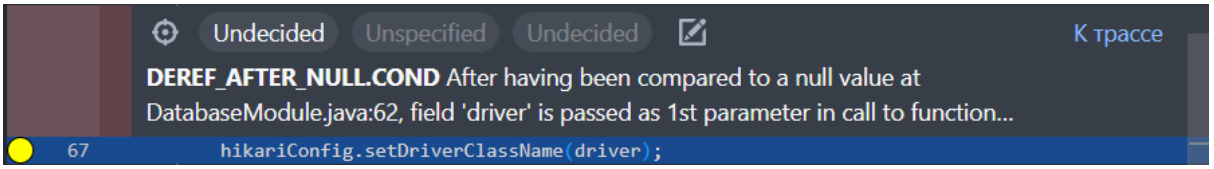


Рисунок 20 –Детектор DEREF_AFTER_NULL в интерфейсе Svacer

Эта ошибка связана с тем, что вы сравнили поле driver с null, и после этого передали его как первый параметр в функцию, где оно разыменовывается (dereferenced). Это может привести к исключению NullPointerException, так как вы пытаетесь обратиться к полю объекта, которое имеет значение null.

DEREF_AFTER_NULL.EX

Этот детектор указывает на то, что при обработке исключения после обозначения указателя как NULL происходит разыменование указателя.

DEREF_OF_NULL.EX

Этот детектор указывает на то, что после обозначения указателя как NULL происходит разыменование указателя

TAINTED_INT

Этот детектор указывает на наличие int-значений, заданных посредством ненадежного источника (например, пользовательский ввод с клавиатуры).

TAINTED_INT.LOOP

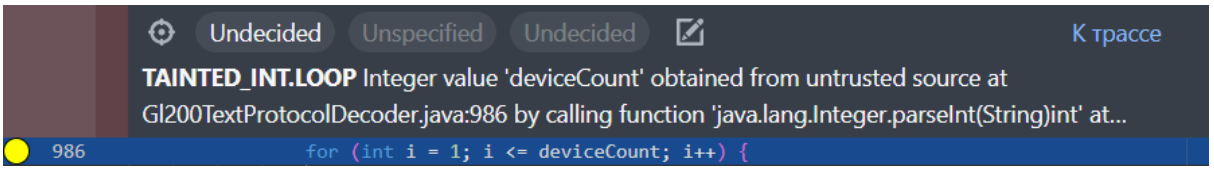


Рисунок 21 –Детектор TAINTED_INT.LOOP в интерфейсе Svacer

Этот детектор указывает на то, что вы используете значение, полученное из ненадежного источника (например, ввод от пользователя или данные из внешнего источника), как верхнюю границу цикла, не выполнив проверку этого значения на корректность.

TAINTED_INT.LOOP.MIGHT

Этот детектор указывает на то, что используемое в цикле int-значение, получено из ненадежного источника и потенциально (MIGHT) может привести к разного рода ошибкам

TAINTED_INT.PTR

Этот детектор указывает на использование указателей на int-значения, которые получены из ненадежного источника.

TAINTED_PTR

Этот детектор указывает на то, что в коде используется указатель, полученный или измененный ненадежным источником.

TAINTED_PTR.MIGHT

Этот детектор указывает на то, что в коде используется указатель, полученный или измененный ненадежным источником, что потенциально (MIGHT) может привести к разного рода ошибкам.

TAINTED_PTR.FORMAT_STRING

Этот детектор указывает на потенциально неправильный формат использования строки указателя

AUTHENTICATE_IN_LOOP

Этот детектор указывает на наличие аутентификации пользователя в цикле, что может стать угрозой безопасности

DEADLOCK

Этот детектор указывает на возможную взаимоблокировку процессов - параллельного ожидания несколькими процессами друг друга.

FB.IL_INFINITE_RECURSIVE_LOOP

Этот детектор указывает на бесконечную рекурсию или бесконечный цикл, обнаруженные при сканировании IL - кода (Intermediate Language)

FB.ML_SYNC_ON_FIELD_TO_GUARD_CHANGING_THAT_FIELD

Этот детектор обнаруживает синхронизацию поля, что выглядит как попытка защититься от одновременных обновлений этого поля. Но при защите поля блокируется объект, на который ссылается, а не поле.

FB.ML_SYNC_ON_UPDATED_FIELD

Этот детектор синхронизирует объект, на который ссылается изменяемое поле. Вряд ли это будет иметь полезный смысл, поскольку разные потоки могут синхронизироваться с разными объектами.

FB.UL_UNRELEASED_LOCK_EXCEPTION_PATH

Этот метод получает блокировку JSR-166 (java.util.concurrent), но не освобождает ее для всех путей исключений из метода.

7.1.2. MAJOR ошибки

Ошибкам класса “Major” следует уделить повышенное внимания. Они, скорее всего, приведут к неправильной работе программы или дадут возможности неправильного использования. (Рисунок 21)

Рисунок 22 – Обозначение MAJOR ошибок в интерфейсе Svacer

TAINTED_ARRAY_INDEX.MIGHT

Этот детектор указывает на обращение к элементу массива по индексу, полученному из ненадежного источника, что может привести к проблемам с безопасностью или остановке программы

TAINTED_ARRAY_INDEX.LOOP

Этот детектор указывает на обращение к элементу массива внутри цикла по индексу, полученному из ненадежного источника, что может привести к проблемам с безопасностью или остановке программы

BUFFER_OVERFLOW.PROC.STRICT

Этот детектор указывает на то, что есть небольшая вероятность (STRICT) переполнения буфера, связанного с использованием системных вызовов

DYNAMIC_SIZE_MISMATCH

Этот детектор указывает на несоответствие размеров данных с динамическими размерами ожидаемому размеру

CHECK_AFTER_OVERFLOW

Этот детектор указывает на то, что проверка буфера происходит уже после переполнения

OVERFLOW_AFTER_CHECK.MACRO

Детектор, разработанный на основе эвристического алгоритма, который, используя информацию об индуктивных переменных и граничных условиях цикла, строит значения для переменных цикла и ищет ошибочные ситуации на основе этих значений.

HANDLE_LEAK

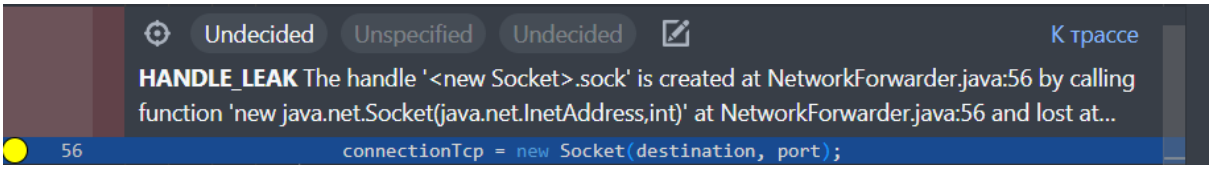


Рисунок 23 –Детектор HANDLE_LEAK в интерфейсе Svacer

Ошибка "The handle is created by calling function and lost" (или подобные предупреждения) указывает на потенциальную проблему в вашем коде, когда вы создаёте ресурс (handle) с использованием функции, но затем теряете ссылку на этот ресурс, что может привести к утечке ресурсов или другим проблемам.

HANDLE_LEAK.EX

Этот детектор указывает на утечку дескриптора объекта при обработке исключения

DEREF_OF_NULL.ASSIGN

Этот детектор указывает на то, что указатель при определенных условиях приходит на точку разыменования как NULL

NULL_AFTER_DEREF

Этот детектор указывает на приравнивание указателя к NULL после разыменования

NULL_AFTER_DEREF.MIGHT

Этот детектор указывает на возможное (MIGHT) приравнивание указателя к NULL после разыменования

DEREF_AFTER_NULL.COND

Этот детектор указывает на разыменование ранее обозначенного как NULL - указателя при выполнении определенных условий

DEREF_AFTER_NULL.EX.COND

Этот детектор указывает на то, что при обработке исключений происходит разыменование ранее обозначенного как NULL - указателя при выполнении определенных условий

DEREF_OF_NULL.EX.COND

Этот детектор указывает на то, что при обработке исключений происходит разыменование NULL - указателя при выполнении определенных условий

DEREF_OF_NULL.RET.LIB

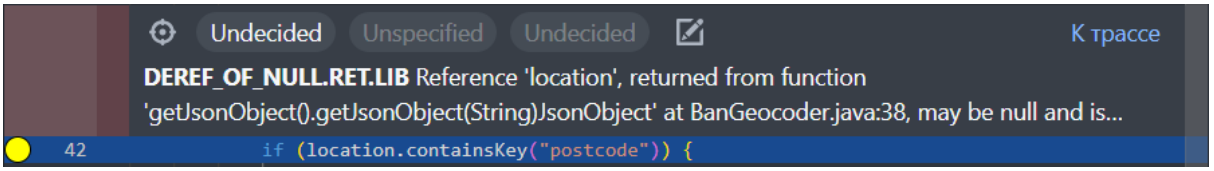


Рисунок 24 –Детектор DEREF_OF_NULL.RET.LIB в интерфейсе Svacer

Это предупреждение указывает на то, что переменная или ссылка location, которая возвращается из функции, может иметь значение null, и затем в вашем коде она разыменовывается (dereferenced). Разыменование null может привести к исключению NullPointerException.

DEREF_OF_NULL.RET.STAT

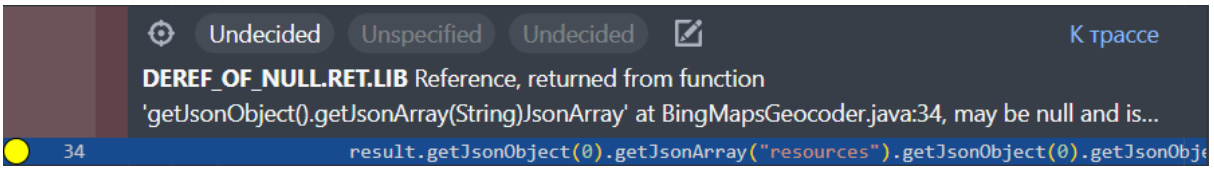


Рисунок 25 –Детектор DEREF_OF_NULL.RET.STAT в интерфейсе Svacer

Это предупреждение указывает на то, что возвращаемый объект может иметь значение null, и затем в вашем коде она разыменовывается (dereferenced). Разыменование null может привести к исключению NullPointerException.

DEREF_AFTER_NULL.MIGHT

Этот детектор указывает на разыменование ранее потенциально (MIGHT) обозначенного как NULL - указателя

DEREF_OF_NULL.ANNOT.CONST

Этот детектор указывает на разыменование константного NULL - указателя при использовании аннотации

DEREF_OF_NULL.ANNOT.ASSIGN

Этот детектор указывает на то, что используемый совместно с аннотацией указатель при определенных условиях приходит на точку разыменования как NULL

DEREF_OF_NULL.ANNOT.EX

Этот детектор указывает на разыменование NULL - указателя при обработке исключений и использовании аннотации

DEREF_OF_NULL.ANNOT.EX.COND

Этот детектор указывает на то, что при определенных условиях происходит разыменование NULL - указателя при обработке исключений и использовании аннотации

TAINTED_INT.MIGHT

Этот детектор указывает на возможное появление int-значений, заданных посредством ненадежного источника (например, пользовательский ввод с клавиатуры).

TAINTED_INT.COND

Этот детектор указывает на то, что при выполнении какой-то условной конструкции появляются int-значений, заданные посредством ненадежного источника (например, пользовательский ввод с клавиатуры).

TAINTED_INT.MIGHT.COND

Этот детектор указывает на то, что при выполнении какой-то условной конструкции могут появиться int-значений, заданные посредством ненадежного источника (например, пользовательский ввод с клавиатуры).

TAINTED_INT.INFINITE_LOOP

Этот маркер указывает возникновение int-значений из ненадежного источника при использовании ошибочно бесконечных циклов

TAINTED_INT.INFINITE_LOOP.MIGHT

Этот маркер указывает возможное возникновение int-значений из ненадежного источника при использовании ошибочно бесконечных циклов

TAINTED.INT_OVERFLOW

Этот маркер указывает возможное возникновение переполнение данных в int-формате, берущихся из ненадежного источника

TAINTED_INT.PTR.MIGHT

Этот детектор указывает на возможное использование указателей на int-значения, которые получены из ненадежного источника.

TAINTED_PTR.COND

Этот детектор указывает на то, что при выполнении определенной логической конструкции используются указатели на int-значения, которые получены из ненадежного источника.

TAINTED_PTR.MIGHT.COND

Этот детектор указывает на то, что при выполнении определенной логической конструкции могут использоваться указатели на int-значения, которые получены из ненадежного источника.

FORMAT_STRING.PARAM_LACK

Этот детектор указывает на недостаток аргументов в строке форматирования

UNINIT.BASE_CTOR

Этот детектор указывает об отсутствии вызова базового конструктора класса

UNCHECKED_FUNC_RES.LIB

Этот детектор указывает об отсутствии проверки значений библиотечной функции

CHECK_AFTER_PASS_TO_PROC

Этот детектор указывает на то, что необходимая проверка значений производится после передачи данных процессу

INVARIANT_RESULT

Этот детектор указывает на наличие конструкции, которые всегда принимают одно и то же значение (инвариант) независимо от других данных, хотя таковыми не задуманы

DOUBLE_LOCK

Этот детектор указывает на двукратную блокировку объекта

NO_UNLOCK

Этот детектор указывает на отсутствие необходимой разблокировки объекта

WRONG_LOCK.STATIC

Этот детектор указывает на наличие доступа к статическим полям класса по не статическим объектам при синхронизации

NO_LOCK.STAT

Этот детектор указывает на отсутствие необходимой блокировки ресурсов, что может привести к проблемам в многопоточной программе

NO_LOCK.STAT.EX

Детектор, разработанный на основе эвристического алгоритма, который, используя информацию об индуктивных переменных и граничных условиях цикла, строит значения для переменных цикла и ищет ошибочные ситуации на основе этих значений, для отсутствия необходимой блокировки ресурсов, что может привести к проблемам в многопоточной программе.

NO_LOCK.GUARD

DEADLOCK.CLASS_LOADING

Этот детектор указывает на возможность взаимной блокировки процессов при загрузке класса

INFINITE_LOOP.EX

Этот детектор указывает на связанное с ошибкой в написании кода появление бесконечных циклов при обработке исключений

INFINITE_LOOP.STRING

Этот детектор указывает на связанное с ошибкой в написании кода появление бесконечных циклов при обработке строковых данных

INFINITE_LOOP.NEW

Этот детектор указывает на связанное с ошибкой в написании кода появление выделяющих объектов в памяти бесконечных циклов

INFINITE_LOOP.INT_OVERFLOW

Этот детектор указывает на связанное с ошибкой в написании кода появление бесконечных циклов, переполняющих значения int-переменных

INFINITE_LOOP.INT_OVERFLOW.STRICT

INFINITE_LOOP.INT_OVERFLOW.ARRAY

Этот детектор указывает на связанное с ошибкой в написании кода появление бесконечных циклов, переполняющих значения int-элементов при обработке массивов

DIVISION_BY_ZERO

Этот детектор указывает на наличие деления на ноль

DIVISION_BY_ZERO.EX

Этот детектор указывает на наличие деления на ноль при обработке исключения

DIVISION_BY_ZERO.UNDER_CHECK

Этот детектор указывает на наличие потенциального деления на ноль, не подверженного проверке

SIMILAR_BRANCHES

Этот детектор указывает на то, что присутствуют разные блоки кода, но с одинаковой функциональностью (например, в конструкции if {…} else {…})

SIMILAR_BRANCHES.SWITCH

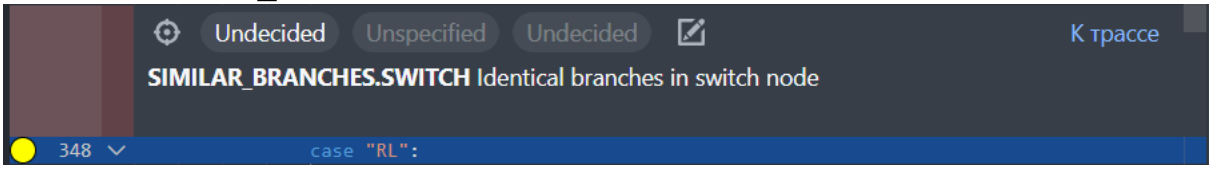


Рисунок 26 –Детектор SIMILAR_BRANCHES.SWITCH в интерфейсе Svacer

Этот детектор указывает на то, что в блоке switch вашего кода есть два (или более) case, которые выполняют одни и те же действия. Такие идентичные ветви могут быть ошибочными или могут сигнализировать о некорректной логике в вашем коде.

DEFAULT_CASE_MISSING

Этот детектор указывает на отсутствие оператора default в блоке switch, выполняющего код, если условия ни одного из case не были выполнены.

CATCH.NULL_POINTER_EXCEPTION

Этот детектор указывает на отсутствие необходимой обработки исключения NullPointerException

CATCH.GENERIC_EXCEPTION

Этот детектор указывает на отсутствие необходимой обработки неспецифического исключения

CATCH.NO_BODY

Этот детектор указывает на наличие блока обработки исключения, не содержащего кода.

RETURN_IN_FINALLY

Этот детектор указывает на присутствующие в блоках finally процедуры return, что чревато ошибками

THROW.GENERIC_EXCEPTION

Этот детектор указывает на использование неспецифических исключений, неудобных для отладки

WRONG_SEMICOLON

Этот детектор указывает на ошибки в проставлении точек с запятой

BAD_COPY_PASTE

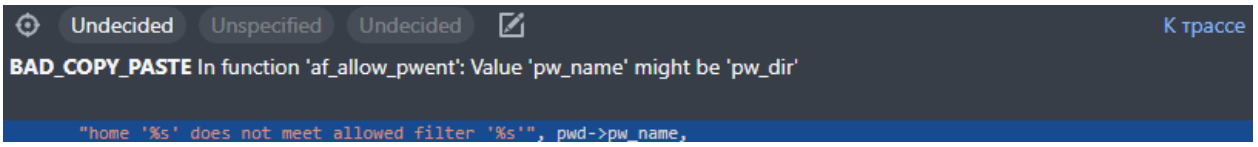


Рисунок 27 –Детектор BAD_COPY_PASTE в интерфейсе Svacer

Ошибка "value 'false' might be 'true'" может указывать на ситуацию, когда ваш код или условие возвращает логическое значение **false**, но существует потенциальная причина, по которой это значение может быть интерпретировано как **true**. Это может привести к нежелательному поведению в вашем программном коде.

WRONG_ARGUMENTS_ORDER

Этот детектор указывает на неправильный порядок аргументов (например, при вызове функции)

BAD_WAIT_OF_COND

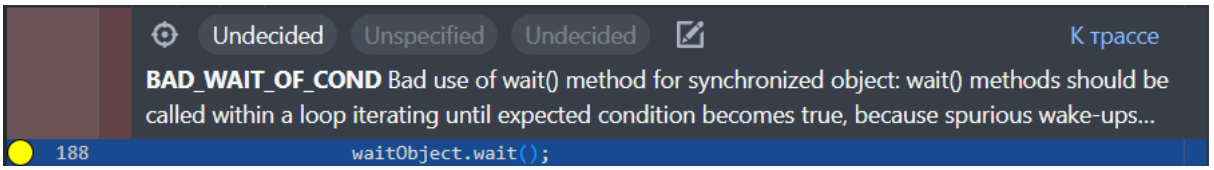


Рисунок 28 –Детектор BAD_WAIT_OF_COND в интерфейсе Svacer

Это сообщение предупреждает о том, что использование метода wait() в синхронизированном контексте не всегда безопасно, и рекомендует обернуть вызов wait() в цикл, который продолжает ожидание до тех пор, пока не выполняется определенное условие. Это связано с возможностью "ложных пробуждений" (spurious wake-ups), когда поток может быть пробужден даже без явного вызова notify() или notifyAll().

В случае программы на Java часто встречаются ошибки, связанные с разыменовыванием указателя, такие ошибки начинаются с **DEREF** (сокращение от "dereference").

WRONG_LOCK

Этот детектор указывает на наличие ошибочных блокировок объектов

FALL_THROUGH

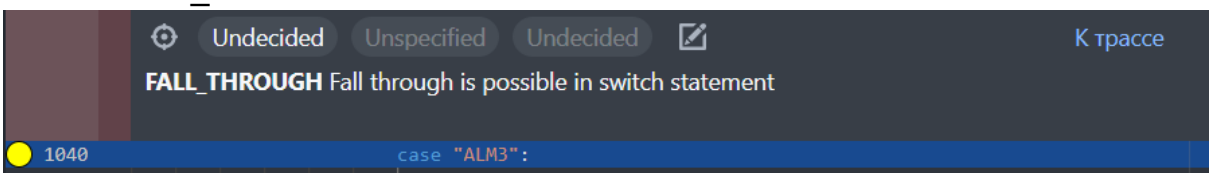


Рисунок 29 –Детектор FALL_THROUGH в интерфейсе Svacer

Это предупреждение указывает, что вы можете проскочить через switch-case сразу в brake, без каких-либо сопутствующих действий.

CONFUSING_INDENTATION

Этот детектор указывает на такое вертикальное выравнивание строк заставляет пользователя предположить структуру потока управления,

отличную от той, которая присутствует на самом деле. Более того, это может привести к ошибкам в ходе работы программы

FB.BC_EQUALS_METHOD_SHOULD_WORK_FOR_ALL_OBJECTS

Этот детектор указывает на потенциальную ошибку работы метода equals (Object o). Суть ошибки заключается в том, что метод не должен делать никаких предположений о типе o. Он должен просто возвращать false, если o не имеет того же типа, что и этот.

FB.BIT_SIGNED_CHECK

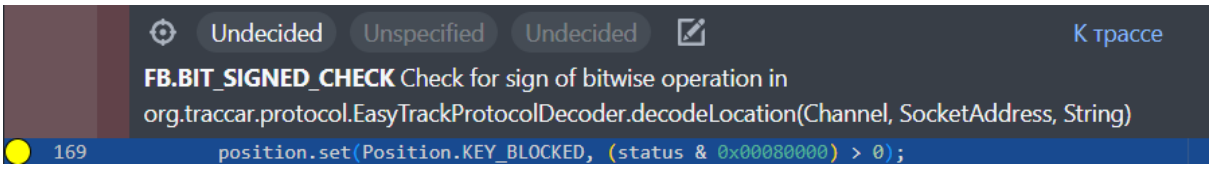


Рисунок 30 –Детектор FB.BIT_SIGNED_CHECK в интерфейсе Svacer

Предупреждение "check for sign of bitwise" означает, что в коде выполняется операция с битами, и вам, возможно, стоит проверить, как эта операция взаимодействует со знаком чисел. Оно может возникнуть, когда вы используете битовые операции (например, побитовое И, ИЛИ, сдвиги) с знаковыми целыми числами, и компилятор обращает внимание на потенциальные проблемы с интерпретацией знака.

FB.CN_IDIOM

Детектор определяет ошибку, в ходе которой класс реализует Cloneable, но не определяет и не использует метод clone.

FB.CN_IMPLEMENTS_CLONE_BUT_NOT_CLONEABLE

Детектор определяет ошибку, в ходе которой класс определяет метод clone(), но класс не реализует Cloneable. В некоторых ситуациях это нормально (например, вы хотите контролировать, как подклассы могут клонировать себя), но просто убедитесь, что это именно то, что вы задумали.

FB.CNT_ROUGH_CONSTANT_VALUE

Детектор определяет уязвимость, в которой рекомендуется использовать предопределенную библиотечную константу для ясности кода и большей точности.

FB.DE_MIGHT_IGNORE

Детектор определяет ошибку, в ходе которой метод может игнорировать исключение. В общем, исключения должны каким-либо образом обрабатываться или сообщаться о них, либо их следует выбрасывать из метода.

FB.DMI_RANDOM_USED_ONLY_ONCE

Детектор определяет ошибку, в ходе которой код создает объект java.util.Random, использует его для генерации одного случайного числа, а затем отбрасывает объект Random. Это дает случайные числа посредственного качества и неэффективно. Если возможно, перепишите код так, чтобы объект Random создавался один раз и сохранялся, и каждый раз, когда требуется новое случайное число, вызывайте метод существующего объекта Random для его получения.

Если важно, чтобы сгенерированные случайные числа нельзя было угадать, вы не должны создавать новые случайные числа для каждого случайного числа; значения слишком легко угадать. Вместо этого вам следует серьезно рассмотреть возможность использования java.security.SecureRandom (и избегать выделения нового SecureRandom для каждого необходимого случайного числа).

FB.DMI_USING_REMOVEALL_TO_CLEAR_COLLECTION

Детектор определяет ошибку, для исправления которой лучше использовать c.clear, а не c.removeAll(c) для удаления коллекции. Вызов c.removeAll(c) для очистки коллекции менее понятен, подвержен ошибкам из-за опечаток, менее эффективен и для некоторых коллекций может вызвать исключение ConcurrentModificationException.

FB.DM_BOXED_PRIMITIVE_FOR_PARSING

Детектор определяет ошибку, в ходе которой упакованный примитив создается из строки просто для извлечения значения распакованного примитива. Более эффективно просто вызвать статический метод parseXXX.

FB.DM_EXIT

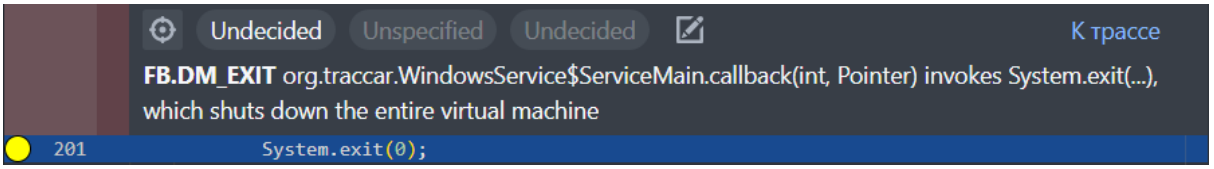


Рисунок 31 –Детектор FB.DM_EXIT в интерфейсе Svacer

Предупреждение "invokes System.exit, which shuts down the entire virtual machine" указывает на то, что в вашем коде есть вызов метода System.exit(). Этот метод является механизмом завершения работы виртуальной машины Java (JVM) и приводит к выходу из программы и завершению работы всей виртуальной машины.

FB.DM_RUN_FINALIZERS_ON_EXIT

Никогда ни по какой причине не вызывайте System.runFinalizersOnExit или Runtime.runFinalizersOnExit: они являются одними из самых опасных методов в библиотеках Java. -- Джошуа Блох

FB.ES_COMPARING_PARAMETER_STRING_WITH_EQ

Детектор определяет ошибку, в ходе которой код сравнивает параметр java.lang.String на предмет равенства ссылок с помощью операторов == или !=. Требование от вызывающих сторон передавать в метод только строковые константы или встроенные строки является излишне хрупким и редко приводит к измеримому увеличению производительности. Вместо этого рассмотрите возможность использования метода equals(Object).

FB.ES_COMPARING_STRINGS_WITH_EQ

Детектор определяет ошибку, в ходе которой код сравнивает объект java.lang.String на предмет равенства ссылок с помощью операторов == или !=. Требование от вызывающих сторон передавать в метод только строковые константы или встроенные строки является излишне хрупким и редко приводит к измеримому увеличению производительности. Вместо этого рассмотрите возможность использования метода equals(Object).

FB.EQ_CHECK_FOR_OPERAND_NOT_COMPATIBLE_WITH_THIS

Детектор определяет ошибку, в ходе которой метод равенства проверяет, является ли аргумент каким-либо несовместимым типом (т. е. классом, который не является ни супертипом, ни подтипом класса, определяющего метод равенства).

FB.EQ_COMPARETO_USE_OBJECT_EQUALS

Детектор определяет ошибку, в ходе которой класс определяет метод CompareTo(...), но наследует его метод Equals() от java.lang.Object. Как правило, значение CompareTo должно возвращать ноль тогда и только тогда, когда метод равенства возвращает true. Если это правило будет нарушено, в таких классах, как PriorityQueue, возникнут странные и непредсказуемые сбои. В Java 5 метод PriorityQueue.remove использует метод CompareTo, а в Java 6 — метод равенства.

FB.EQ_GETCLASS_AND_CLASS_CONSTANT

Детектор определяет ошибку, в ходе которой в данном классе есть метод равенства, который будет нарушен, если он унаследован подклассами. Он сравнивает литерал класса с классом аргумента. Лучше проверить, есть ли this.getClass() == o.getClass().

FB.FI_EMPTY

Детектор определяет ошибку, в ходе которой обнаружены пустые методы Finalize() бесполезны, поэтому их следует удалить.

FB.FI_EXPLICIT_INVOCATION

Детектор определяет ошибку, в ходе которой метод содержит явный вызов метода Finalize() для объекта. Поскольку методы финализатора должны выполняться один раз и только виртуальной машиной, это плохая идея.

FB.FI_FINALIZER_NULLS_FIELDS

Детектор определяет ошибку, в ходе которой финализатор обнуляет поля. Обычно это ошибка, поскольку она не помогает сборке мусора, и объект в любом случае будет подвергнут сборке мусора.

FB.FI_NULLIFY_SUPER

Детектор определяет ошибку, в ходе которой пустой метод Finalize() явно сводит на нет эффект любого финализатора, определенного его суперклассом. Любые действия финализатора, определенные для суперкласса, выполняться не будут. Если это не предусмотрено, удалите этот метод.

FB.FI_USELESS

Детектор определяет ошибку, в ходе которой единственное, что делает метод Finalize(), — это вызывает метод Finalize() суперкласса, что делает его избыточным. Его нужно удалить.

FB.VA_FORMAT_STRING_USES_NEWLINE

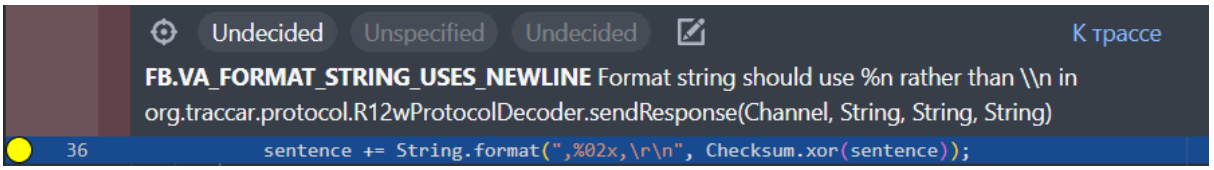


Рисунок 32 –Детектор FB.VA_FORMAT_STRING_USES_NEWLINE в интерфейсе Svacer

Ошибка "Format string should use %n rather than \n" указывает на то, что в строке форматирования используется символ новой строки (\n), и предлагается использовать спецификатор формата %n вместо него.

В Java, %n - это платформозависимая последовательность символов для перевода строки, которая будет заменена на соответствующую последовательность символов новой строки для текущей платформы (например, "\r\n" для Windows или "\n" для Unix).

FB.HE_EQUALS_NO_HASHCODE

Детектор определяет ошибку, в ходе которой класс переопределяет метод равенства(Object), но не переопределяет метод hashCode(). Следовательно, класс может нарушить инвариант, согласно которому равные объекты должны иметь равные хэш-коды.

FB.HE_EQUALS_USE_HASHCODE

Детектор определяет ошибку, в ходе которой класс переопределяет метод equals(Object), но не переопределяет hashCode() и наследует реализацию hashCode() из java.lang.Object (который возвращает идентификационный хэш-код — произвольное значение, назначенное объекту виртуальной машиной). Следовательно, класс с большой вероятностью нарушит инвариант, согласно которому равные объекты должны иметь равные хэш-коды.

FB.HE_INHERITS_EQUALS_USE_HASHCODE

Детектор определяет ошибку, в ходе которой класс наследует Equals(Object) от абстрактного суперкласса и hashCode() от java.lang.Object (который возвращает идентификационный хеш-код — произвольное значение, присвоенное объекту виртуальной машиной). Следовательно, класс с большой вероятностью нарушит инвариант, согласно которому равные объекты должны иметь равные хэш-коды.

FB.IC_SUPERCLASS_USES_SUBCLASS_DURING_INITIALIZATIO N

Детектор определяет ошибку, в ходе которой во время инициализации класса класс активно использует подкласс. Этот подкласс еще не будет инициализирован на момент использования.

FB.IMSE_DONT_CATCH_IMSE

Детектор определяет ошибку, в ходе которой IllegalMonitorStateException обычно генерируется только в случае конструктивного недостатка вашего кода (вызов ожидания или уведомления об объекте, который вы не блокируете).

FB.IT_NO_SUCH_ELEMENT

Детектор определяет ошибку, в ходе которой класс реализует интерфейс java.util.Iterator. Однако его метод next() не способен генерировать

исключение java.util.NoSuchElementException. Метод next() следует изменить, чтобы он выдавал NoSuchElementException, если вызывается, когда больше нет элементов для возврата.

FB.NP_BOOLEAN_RETURN_NULL

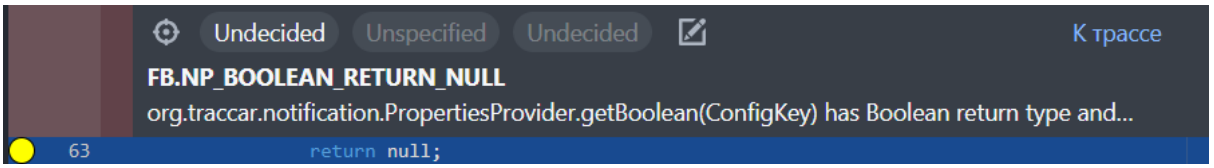


Рисунок 33 –Детектор FB.NP_BOOLEAN_RETURN_NULL в интерфейсе Svacer

Ошибка "has boolean type and returns explicit null" означает, что функция (метод) возвращает явный null, но при этом её тип данных объявлен как boolean. В языках программирования, где тип возвращаемого значения функции строго определен, это может считаться необычным и часто считается ошибкой.

FB.NP_CLONE_COULD_RETURN_NULL

Детектор определяет ошибку, в ходе которой в некоторых случаях метод клонирования возвращает значение null, но клону никогда не разрешается возвращать нулевое значение. Если вы убеждены, что этот путь недоступен, вместо этого выдайте AssertionError.

FB.NP_EQUALS_SHOULD_HANDLE_NULL_ARGUMENT

Детектор определяет ошибку, в ходе которой реализация метода equals(Object) нарушает контракт, определенный в java.lang.Object.equals(), поскольку он не проверяет передачу значения null в качестве аргумента. Все методы Equals() должны возвращать значение false, если им передано нулевое значение.

FB.NP_TOSTRING_COULD_RETURN_NULL

Детектор определяет ошибку, в ходе которой toString, похоже, в некоторых случаях возвращает значение null. Либеральное прочтение спецификации можно было бы интерпретировать как разрешение этого, но это, вероятно, плохая идея и может привести к поломке другого кода. Верните пустую строку или другую подходящую строку вместо нуля.

FB.NM_METHOD_NAMING_CONVENTION

Детектор определяет ошибку, в ходе которой методы, при которых следует произносить такие глаголы в смешанном регистре, первая буква должна быть строчной, а первая буква каждого внутреннего слова — заглавной.

FB.NP_METHOD_PARAMETER_TIGHTENS_ANNOTATION

Детектор определяет ошибку, в ходе которой метод всегда должен реализовывать контракт метода, который он переопределяет. Таким образом, если метод принимает параметр, помеченный как @Nullable, вам не следует переопределять этот метод в подклассе методом, где этот параметр имеет значение @Nonnull. Это нарушает контракт, согласно которому метод должен обрабатывать нулевой параметр.

FB.PZ_DONT_REUSE_ENTRY_OBJECTS_IN_ITERATORS

Детектор определяет ошибку, в ходе которой методу входа entrySet() разрешено возвращать представление базовой карты, в которой есть итератор и Map.Entry. Эта умная идея использовалась в нескольких реализациях Map, но допускает неприятные ошибки в кодировании. Если карта m возвращает такой итератор для набора записей, то c.addAll(m.entrySet()) пойдет не так. Все реализации Map в OpenJDK 7 были переписаны, чтобы избежать этого, вам тоже следует это сделать.

FB.RC_REF_COMPARISON_BAD_PRACTICE

Детектор определяет ошибку, в ходе которой метод сравнивает ссылочное значение с константой с помощью оператора == или !=, при этом правильный способ сравнения экземпляров этого типа обычно заключается в метод equals(). Можно создавать отдельные экземпляры, которые равны, но не сравниваются как ==, поскольку это разные объекты. Примерами классов, которые обычно не следует сравнивать по ссылке, являются java.lang.Integer, java.lang.Float и т. д.

FB.RC_REF_COMPARISON_BAD_PRACTICE_BOOLEAN

Детектор определяет ошибку, в ходе которой метод сравнивает два логических значения с помощью оператора == или !=. Обычно существует только два логических значения (Boolean.TRUE и Boolean.FALSE), но можно создавать другие логические объекты с помощью нового конструктора Boolean(b). Лучше всего избегать таких объектов, но если они существуют, то проверка логических объектов на равенство с помощью == или != даст результаты, отличные от тех, которые вы получили бы, используя .equals(...).

FB.RV_NEGATING_RESULT_OF_COMPARETO

Детектор определяет ошибку, в ходе которой код отменяет возвращаемое значение метода CompareTo или Compare. Это сомнительная или плохая практика программирования, поскольку, если возвращаемое значение равно Integer.MIN_VALUE, отрицание возвращаемого значения не приведет к отрицанию знака результата. Вы можете добиться того же желаемого результата, изменив порядок операндов на противоположный, а не отрицая результаты.

FB.RV_RETURN_VALUE_IGNORED_BAD_PRACTICE

Детектор определяет ошибку, в ходе которой метод возвращает значение, которое не проверено. Возвращаемое значение следует проверять, поскольку оно может указывать на необычное или неожиданное выполнение функции. Например, метод File.delete() возвращает false, если файл не удалось успешно удалить (вместо того, чтобы выдать исключение). Если вы не проверите результат, вы не заметите, что вызов метода сигнализирует о неожиданном поведении, возвращая нетипичное возвращаемое значение.

FB.SI_INSTANCE_BEFORE_FINALS_ASSIGNED

Детектор определяет ошибку, в ходе которой статический инициализатор класса создает экземпляр класса до того, как будут назначены все статические конечные поля.

FB.SE_BAD_FIELD

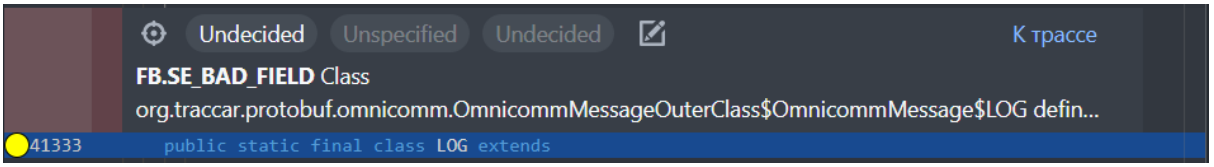


Рисунок 34 –Детектор FB.SE_BAD_FIELD в интерфейсе Svacer

Ошибка "defines non-transient non-serializable field" указывает на то, что в вашем коде есть объявление поля, которое не является transient и не реализует интерфейс Serializable. Это может быть проблемой, если вы планируете сериализовать объекты, например, для передачи через сеть, сохранения на диске или использования в других механизмах сериализации.

FB.SE_BAD_FIELD_INNER_CLASS

Детектор определяет ошибку, в ходе которой сериализуемый класс является внутренним классом несериализуемого класса. Таким образом, попытки сериализации также будут пытаться связать экземпляр внешнего класса, с которым он связан, что приведет к ошибке во время выполнения.

FB.SE_BAD_FIELD_STORE

Детектор определяет ошибку, в ходе которой несериализуемое значение сохраняется в непереходном поле сериализуемого класса.

FB.SE_COMPARATOR_SHOULD_BE_SERIALIZABLE

Детектор определяет ошибку, в ходе которой класс реализует интерфейс Comparator. Вам следует подумать, следует ли ему также реализовывать интерфейс Serializable. Если компаратор используется для создания

упорядоченной коллекции, такой как TreeMap, тогда TreeMap будет сериализуемым, только если компаратор также сериализуем. Поскольку большинство компараторов имеют небольшое состояние или вообще не имеют его, сделать их сериализуемыми, как правило, легко и является хорошим защитным программированием.

FB.SE_INNER_CLASS

Детектор определяет ошибку, в ходе которой сериализуемый класс является внутренним классом. Любая попытка сериализовать его также приведет к сериализации связанного внешнего экземпляра. Внешний экземпляр является сериализуемым, поэтому это не приведет к сбою, но может сериализовать гораздо больше данных, чем предполагалось. Если возможно, проблему следует решить, сделав внутренний класс статическим внутренним классом (также известным как вложенный класс).

FB.SE_NO_SUITABLE_CONSTRUCTOR

Детектор определяет ошибку, в ходе которой класс реализует интерфейс Serializable, а его суперкласс — нет. Когда такой объект десериализуется, поля суперкласса необходимо инициализировать, вызывая конструктор void суперкласса. Поскольку у суперкласса его нет, сериализация и десериализация завершится неудачей во время выполнения.

FB.SE_NO_SUITABLE_CONSTRUCTOR_FOR_EXTERNALIZATION

Детектор определяет ошибку, в ходе которой класс реализует интерфейс Externalizable, но не определяет общедоступный конструктор void. Когда объекты externalizable десериализуются, их сначала необходимо создать, вызвав общедоступный конструктор void. Поскольку у этого класса его нет, сериализация и десериализация завершатся ошибкой во время выполнения.

FB.SE_TRANSIENT_FIELD_NOT_RESTORED

Детектор определяет ошибку, в ходе которой класс содержит поле, которое обновляется в нескольких местах класса, поэтому оно кажется частью состояния класса. Однако, поскольку поле помечено как временное и не задано в readObject или readResolve, оно будет содержать значение по умолчанию в любом десериализованном экземпляре класса.

FB.SE_NO_SERIALVERSIONID

Детектор определяет ошибку, в ходе которой класс реализует интерфейс Serializable, но не определяет поле SerialVersionUID. Такое простое изменение, как добавление ссылки на объект .class, добавит в класс

синтетические поля, что, к сожалению, изменит неявный SerialVersionUID (например, добавление ссылки на String.class приведет к созданию статического поля class$java$lang$String). Кроме того, разные исходные коды для компиляторов байт-кода могут использовать разные соглашения об именах для синтетических переменных, созданных для ссылок на объекты классов или внутренние классы. Чтобы обеспечить совместимость Serializable между версиями, рассмотрите возможность добавления явного SerialVersionUID.

FB.UI_INHERITANCE_UNSAFE_GETRESOURCE

Детектор определяет ошибку, в ходе которой вызов this.getClass().getResource(...) может дать результаты, отличные от ожидаемых, если этот класс расширен классом в другом пакете.

FB.BC_IMPOSSIBLE_CAST

Детектор определяет ошибку, в ходе которой приведение всегда будет вызывать исключение ClassCastException. SpotBugs отслеживает информацию о типах из проверок экземпляров, а также использует более точную информацию о типах значений, возвращаемых методами и загружаемых из полей. Таким образом, он может иметь более точную информацию, чем просто объявленный тип переменной, и может использовать ее, чтобы определить, что приведение всегда будет вызывать исключение во время выполнения.

FB.BC_IMPOSSIBLE_DOWNCAST

Детектор определяет ошибку, в ходе которой приведение всегда будет вызывать исключение ClassCastException. Анализ предполагает, что он знает точный тип приводимого значения, и попытка приведения его к подтипу всегда будет завершаться неудачей из-за выдачи исключения ClassCastException.

FB.BC_IMPOSSIBLE_DOWNCAST_OF_TOARRAY

Детектор определяет ошибку, в ходе которой код приводит результат вызова toArray() в коллекции к типу, более конкретному, чем Object[]. Обычно это не удается из-за выдачи исключения ClassCastException. toArray() почти всех коллекций возвращает Object[]. На самом деле они не могут сделать ничего другого, поскольку объект Collection не имеет ссылки на объявленный универсальный тип коллекции.

FB.BC_IMPOSSIBLE_INSTANCEOF

Детектор определяет ошибку, в ходе которой тест экземпляра всегда будет возвращать false. Хотя это и безопасно, убедитесь, что это не является

признаком какого-то недопонимания или какой-либо другой логической ошибки.

FB.BIT_ADD_OF_SIGNED_BYTE

Детектор определяет ошибку, в ходе которой происходит добавление значений байта и значение, у которого известно, что 8 младших битов очищены. Значения, загружаемые из байтового массива, расширяются по знаку до 32 битов перед выполнением каких-либо побитовых операций над значением. Таким образом, если b[0] содержит значение 0xff, а x изначально равен 0, то код ((x << 8) + b[0]) подпишет расширение 0xff, чтобы получить 0xffffffff, и, таким образом, даст значение 0xffffffff как результат.

FB.BIT_AND_ZZ

Детектор определяет ошибку, в ходе которой метод сравнивает выражение формы (e & 0) с 0, которое всегда будет сравниваться равным. Это может указывать на логическую ошибку или опечатку.

FB.BIT_IOR

Детектор определяет ошибку, в ходе которой метод сравнивает выражение формы (e | C) с D. Это сравнение всегда будет неравным из-за конкретных значений констант C и D. Это может указывать на логическую ошибку или опечатку.

FB.BIT_IOR_OF_SIGNED_BYTE

Детектор определяет ошибку, в ходе которой метод загружает байтовое значение (например, значение, загруженное из массива байтов или возвращаемое методом с возвращаемым типом байт) и выполняет поразрядное ИЛИ с этим значением. Байтовые значения расширяются до 32 битов перед выполнением каких-либо побитовых операций над значением. Таким образом, если b[0] содержит значение 0xff, а x изначально равен 0, то код ((x << 8) | b[0]) подпишет расширение 0xff, чтобы получить 0xffffffff, и, таким образом, даст значение 0xffffffff как результат.

FB.ICAST_BAD_SHIFT_AMOUNT

Детектор определяет ошибку, в ходе которой код выполняет сдвиг 32-битного целого числа на постоянную величину за пределами диапазона -31..31. В результате используются младшие 5 бит целочисленного значения, чтобы решить, на сколько сместить (например, сдвиг на 40 бит аналогичен сдвигу на 8 бит, а сдвиг на 32 бита аналогичен сдвигу на ноль). биты). Вероятно, это не то, чего ожидали, и это, по крайней мере, сбивает с толку.

FB.DMI_BAD_MONTH

Детектор определяет ошибку, в ходе которой код передает методу постоянное значение месяца, выходящее за пределы ожидаемого диапазона 0–11.

FB.DMI_BIGDECIMAL_CONSTRUCTED_FROM_DOUBLE

Детектор определяет ошибку, в ходе которой код создает BigDecimal из двойного значения, которое плохо преобразуется в десятичное число. Например, можно предположить, что запись нового BigDecimal(0.1) в Java создает BigDecimal, который точно равен 0,1 (немасштабированное значение 1, со шкалой 1), но на самом деле он равен 0,1000000000000000055511151231257827021181583404541015625. Вероятно, вы захотите использовать метод BigDecimal.valueOf(double d), который использует строковое представление double для создания BigDecimal (например, BigDecimal.valueOf(0,1) дает 0,1).

FB.DMI_COLLECTIONS_SHOULD_NOT_CONTAIN_THEMSELVES

Детектор определяет ошибку, в ходе которой вызов метода универсальной коллекции имел бы смысл только в том случае, если коллекция содержала бы саму себя (например, если бы s.contains(s) было истинным). Это маловероятно, и если бы это было правдой, это вызвало бы проблемы (например, вычисление хеш-кода, приводящее к бесконечной рекурсии). Вероятно, в качестве параметра передается неправильное значение.

FB.DMI_INVOKING_HASHCODE_ON_ARRAY

Детектор определяет ошибку, в ходе которой код вызывает hashCode в массиве. Вызов hashCode для массива возвращает то же значение, что и System.identityHashCode, и игнорирует содержимое и длину массива. Если вам нужен хэш-код, который зависит от содержимого массива a, используйте java.util.Arrays.hashCode(a).

FB.DMI_VACUOUS_SELF_COLLECTION_CALL

Детектор определяет ошибку, в ходе которой call не имеет смысла. Для любой коллекции c вызов c.containsAll(c) всегда должен иметь значение true, а c.retainAll(c) не должен иметь никакого эффекта.

FB.DMI_ANNOTATION_IS_NOT_VISIBLE_TO_REFLECTION

Детектор определяет ошибку, в ходе которой если аннотация сама не была аннотирована с помощью @Retention(RetentionPolicy.RUNTIME), аннотацию нельзя наблюдать с помощью отражения (например, с помощью метода isAnnotationPresent).

FB.EC_ARRAY_AND_NONARRAY

Детектор определяет ошибку, в ходе которой метод вызывает .equals(Object o) для сравнения массива и ссылки, которая не является массивом. Если сравниваемые объекты относятся к разным типам, они гарантированно будут неравными, и сравнение почти наверняка будет ошибкой. Даже если они оба являются массивами, метод Equals() для массивов определяет только то, являются ли эти два массива одним и тем же объектом. Чтобы сравнить содержимое массивов, используйте java.util.Arrays.equals(Object[], Object[]).

FB.EC_BAD_ARRAY_COMPARE

Детектор определяет ошибку, в ходе которой метод вызывает метод .equals(Object o) для массива. Поскольку массивы не переопределяют метод равенства объекта Object, вызов метода равенства в массиве аналогичен сравнению их адресов. Чтобы сравнить содержимое массивов, используйте java.util.Arrays.equals(Object[], Object[]). Для сравнения адресов массивов было бы проще явно проверить равенство указателей с помощью ==.

FB.EC_NULL_ARG

Детектор определяет ошибку, в ходе которой метод вызывает метод equals(Object), передавая в качестве аргумента нулевое значение. Согласно контракту метода equals(), этот вызов всегда должен возвращать false.

FB.EC_UNRELATED_CLASS_AND_INTERFACE

Детектор определяет ошибку, в ходе которой метод вызывает метод equals(Object) по двум ссылкам, одна из которых является классом, а другая — интерфейсом, причем ни класс, ни какой-либо из его неабстрактных подклассов не реализуют интерфейс. Таким образом, сравниваемые объекты вряд ли будут членами одного и того же класса во время выполнения (если только некоторые классы приложения не были проанализированы или во время выполнения может произойти динамическая загрузка классов). Согласно контракту методаquals(), объекты разных классов всегда должны сравниваться как неравные; следовательно, согласно контракту, определенному java.lang.Object.equals(Object), результат этого сравнения всегда будет ложным во время выполнения.

FB.EC_UNRELATED_INTERFACES

Детектор определяет ошибку, в ходе которой метод вызывает методquals(Object) по двум ссылкам на несвязанные типы интерфейсов, где ни один из них не является подтипом другого, и нет известных неабстрактных классов, реализующих оба интерфейса. Таким образом, сравниваемые объекты вряд ли будут членами одного и того же класса во время выполнения (если только некоторые классы приложения не были

проанализированы или во время выполнения может произойти динамическая загрузка классов). Согласно контракту методаquals(), объекты разных классов всегда должны сравниваться как неравные; следовательно, согласно контракту, определенному java.lang.Object.equals(Object), результат этого сравнения всегда будет ложным во время выполнения.

FB.EC_UNRELATED_TYPES

Детектор определяет ошибку, в ходе которой метод вызывает метод Equals(Object) для двух ссылок на разные типы классов, и анализ показывает, что они будут относиться к объектам разных классов во время выполнения. Кроме того, изучение методов равенства, которые будут вызываться, позволяет предположить, что либо этот вызов всегда будет возвращать значение false, либо метод равенства не является симметричным (что является свойством, требуемым контрактом для равенства в классе Object).

FB.EC_UNRELATED_TYPES_USING_POINTER_EQUALITY

Детектор определяет ошибку, в ходе которой метод использует равенство указателей для сравнения двух ссылок, которые, по-видимому, относятся к разным типам. Результат этого сравнения всегда будет ложным во время выполнения.

FB.EQ_ALWAYS_FALSE

Детектор определяет ошибку, в ходе которой класс определяет метод равенства, который всегда возвращает false. Это означает, что объект не равен самому себе и невозможно создать полезные Карты или Наборы этого класса. На более фундаментальном уровне это означает, что равенство не является рефлексивным, что является одним из требований метода равенства.

FB.EQ_ALWAYS_TRUE

Детектор определяет ошибку, в ходе которой класс определяет метод равенства, который всегда возвращает true. Это изобретательно, но не очень умно. Кроме того, это означает, что метод равенства не симметричен.

FB.EQ_COMPARING_CLASS_NAMES

Детектор определяет ошибку, в ходе которой класс определяет метод равенства, который проверяет, относятся ли два объекта к одному и тому же классу, проверяя, равны ли имена их классов. У вас могут быть разные классы с одним и тем же именем, если они загружаются разными загрузчиками классов. Просто проверьте, совпадают ли объекты класса.

FB.EQ_OVERRIDING_EQUALS_NOT_SYMMETRIC

Детектор определяет ошибку, в ходе которой класс определяет метод равенства, который переопределяет метод равенства в суперклассе. Оба метода равенства используют instanceof для определения равенства двух объектов. Это таит в себе опасность, поскольку важно, чтобы метод равенства был симметричным (другими словами, a.equals(b) == b.equals(a)). Если B является подтипом A, и метод Equals A проверяет, что аргумент является экземпляром A, а метод Equals B проверяет, что аргумент является экземпляром B, вполне вероятно, что отношение эквивалентности, определенное этими методами, не является симметричным.

FB.EQ_SELF_USE_OBJECT

Детектор определяет ошибку, в ходе которой класс определяет ковариантную версию метода Equals(), но наследует обычный метод Equals(Object), определенный в базовом классе java.lang.Object. Класс, вероятно, должен определить логический метод равенства (Object).

FB.FE_TEST_IF_EQUAL_TO_NOT_A_NUMBER

Детектор определяет ошибку, в ходе которой код проверяет, равно ли значение с плавающей запятой специальному значению Not A Number (например, if (x == Double.NaN)). Однако из-за особой семантики NaN ни одно значение не равно Nan, включая NaN. Таким образом, x == Double.NaN всегда дает значение false. Чтобы проверить, является ли значение, содержащееся в x, специальным значением Not A Number, используйте Double.isNaN(x) (или Float.isNaN(x), если x имеет точность с плавающей запятой).

FB.GC_UNRELATED_TYPES

Детектор определяет ошибку, в ходе которой вызов универсального метода коллекции содержит аргумент с несовместимым классом из класса параметра коллекции (т. е. тип аргумента не является ни супертипом, ни подтипом соответствующего аргумента универсального типа). Поэтому маловероятно, что коллекция содержит какие-либо объекты, равные используемому здесь аргументу метода. Скорее всего, методу передается неправильное значение.

FB.HE_USE_OF_UNHASHABLE_CLASS

Детектор определяет ошибку, в ходе которой класс определяет метод Equals(Object), но не метод hashCode(), и, таким образом, не удовлетворяет требованию, чтобы равные объекты имели равные хэш-коды. Экземпляр этого класса используется в хэш-структуре данных, поэтому необходимость решения этой проблемы имеет первостепенное значение.

FB.ICAST_INT_2_LONG_AS_INSTANT

Детектор определяет ошибку, в ходе которой код преобразует 32-битное целое число в 64-битное длинное значение, а затем передает это значение в качестве параметра метода, которому требуется абсолютное значение времени. Абсолютное значение времени — это количество миллисекунд, прошедших с момента стандартного базового времени, известного как «эпоха», а именно 1 января 1970 года, 00:00:00 по Гринвичу.

FB.ICAST_INT_CAST_TO_DOUBLE_PASSED_TO_CEIL

Детектор определяет ошибку, в ходе которой код преобразует целочисленное значение (например, int или long) в число с плавающей запятой двойной точности, а затем передает результат в функцию Math.ceil(), которая округляет двойное число до следующего более высокого целочисленного значения. Эта операция всегда должна быть пустой, поскольку преобразование целого числа в двойное должно давать число без дробной части. Вполне вероятно, что операция, которая генерировала значение, передаваемое в Math.ceil, предназначалась для выполнения с использованием арифметики двойной точности с плавающей запятой.

FB.ICAST_INT_CAST_TO_FLOAT_PASSED_TO_ROUND

Детектор определяет ошибку, в ходе которой код преобразует значение int в число с плавающей запятой точности float, а затем передает результат в функцию Math.round(), которая возвращает значение int/long, ближайшее к аргументу. Эта операция всегда должна быть пустой, поскольку преобразование целого числа в число с плавающей запятой должно давать число без дробной части. Вполне вероятно, что операция, создавшая значение, передаваемое в Math.round, предназначалась для выполнения с использованием арифметики с плавающей запятой.

FB.IJU_ASSERT_METHOD_INVOKED_FROM_RUN_METHOD

Детектор определяет ошибку, в ходе которой утверждение JUnit выполняется в методе run. Неудачные утверждения JUnit просто приводят к возникновению исключений. Таким образом, если это исключение возникает в потоке, отличном от потока, вызывающего метод тестирования, исключение завершит поток, но не приведет к сбою теста.

FB.IL_CONTAINER_ADDED_TO_ITSELF

Детектор определяет ошибку, в ходе которой коллекция добавляется сама в себя. В результате вычисление хэш-кода этого набора приведет к возникновению исключения StackOverflowException.

FB.IL_INFINITE_LOOP

Детектор определяет ошибку, в ходе которой цикл не имеет способа завершиться (кроме, возможно, выдачи исключения).

FB.INT_BAD_COMPARISON_WITH_NONNEGATIVE_VALUE

Детектор определяет ошибку, в ходе которой код сравнивает значение, которое гарантированно неотрицательное, с отрицательной константой или нулем.

FB.INT_BAD_COMPARISON_WITH_SIGNED_BYTE

Детектор определяет ошибку, в ходе которой байты со знаком могут иметь значение только в диапазоне от -128 до 127. Сравнение байта со знаком со значением вне этого диапазона бессмысленно и, вероятно, будет неверным. Чтобы преобразовать знаковый байт b в беззнаковое значение в диапазоне 0..255, используйте 0xff & b.

FB.IP_PARAMETER_IS_DEAD_BUT_OVERWRITTEN

Детектор определяет ошибку, в ходе которой начальное значение этого параметра игнорируется, и здесь параметр перезаписывается. Это часто указывает на ошибочное мнение, что запись в параметр будет передана обратно вызывающей стороне.

FB.MF_CLASS_MASKS_FIELD

Детектор определяет ошибку, в ходе которой класс определяет поле с тем же именем, что и видимое поле экземпляра в суперклассе. Это сбивает с толку и может указывать на ошибку, если методы обновляют или получают доступ к одному из полей, когда им нужно другое.

FB.NP_NONNULL_PARAM_VIOLATION

Детектор определяет ошибку, в ходе которой метод передает нулевое значение в качестве параметра метода, который не должен быть нулевым. Либо этот параметр явно помечен как @Nonnull, либо анализ определил, что этот параметр всегда разыменовывается.

FB.NP_NONNULL_RETURN_VIOLATION

Детектор определяет ошибку, в ходе которой метод может возвращать нулевое значение, но объявлен метод (или метод суперкласса, который он переопределяет) возвращающий @Nonnull.

FB.NP_UNWRITTEN_FIELD

Детектор определяет ошибку, в ходе которой программа разыменовывает поле, в которое, похоже, никогда не было записано ненулевое значение. Если поле не инициализировано с помощью какого-либо механизма, не видимого при анализе, разыменование этого значения приведет к возникновению исключения нулевого указателя.

FB.NM_VERY_CONFUSING

Детектор определяет ошибку, в ходе которой у упомянутых методов есть имена, которые отличаются только заглавными буквами. Это очень сбивает с толку, поскольку если бы капитализация была идентична, один из методов переопределял бы другой.

FB.NM_WRONG_PACKAGE

Детектор определяет ошибку, в ходе которой метод в подклассе не переопределяет аналогичный метод в суперклассе, поскольку тип параметра не совсем соответствует типу соответствующего параметра в суперклассе.

FB.RC_REF_COMPARISON

Детектор определяет ошибку, в ходе которой метод сравнивает два ссылочных значения с помощью оператора == или !=, при этом правильный способ сравнения экземпляров этого типа обычно заключается в метод equals(). Можно создавать отдельные экземпляры, которые равны, но не сравниваются как ==, поскольку это разные объекты. Примерами классов, которые обычно не следует сравнивать по ссылке, являются java.lang.Integer, java.lang.Float и т. д. RC_REF_COMPARISON охватывает только типы-оболочки для примитивов.

FB.RE_BAD_SYNTAX_FOR_REGULAR_EXPRESSION

Детектор определяет ошибку, в ходе которой в приведенном здесь коде используется регулярное выражение, которое недопустимо в соответствии с синтаксисом регулярных выражений. Этот оператор при выполнении выдаст исключение PatternSyntaxException.

FB.RE_POSSIBLE_UNINTENDED_PATTERN

Детектор определяет ошибку, в ходе которой вызывается строковая функция и "." или "|" передается параметру, который принимает регулярное выражение в качестве аргумента.

FB.RV_ABSOLUTE_VALUE_OF_HASHCODE

Детектор определяет ошибку, в ходе которой код генерирует хэш-код, а затем вычисляет абсолютное значение этого хэш-кода. Если хеш-код — Integer.MIN_VALUE, то результат также будет отрицательным (поскольку Math.abs(Integer.MIN_VALUE) == Integer.MIN_VALUE).

FB.RV_ABSOLUTE_VALUE_OF_RANDOM_INT

Детектор определяет ошибку, в ходе которой код генерирует случайное целое число со знаком, а затем вычисляет абсолютное значение этого случайного целого числа. Если число, возвращаемое генератором случайных чисел, равно Integer.MIN_VALUE, то результат также будет отрицательным

(поскольку Math.abs(Integer.MIN_VALUE) == Integer.MIN_VALUE). (Та же проблема возникает и для длинных значений).

FB.RV_EXCEPTION_NOT_THROWN

Детектор определяет ошибку, в ходе которой код создает объект исключения (или ошибки), но ничего с ним не делает.

FB.RV_RETURN_VALUE_IGNORED

Детектор определяет ошибку, в ходе которой код вызывает метод и игнорирует возвращаемое значение. Возвращаемое значение имеет тот же тип, что и тип, для которого вызывается метод, и наш анализ показывает, что возвращаемое значение может быть важным (например, как игнорирование возвращаемого значения String.toLowerCase()).

FB.RPC_REPEATED_CONDITIONAL_TEST

Детектор определяет ошибку, в ходе которой код содержит условную проверку, выполняемую дважды, одну за другой (например, x == 0 || x == 0). Возможно, второе вхождение должно быть чем-то другим (например, x == 0 || y == 0).

FB.SA_FIELD_SELF_ASSIGNMENT

Детектор определяет ошибку, в ходе которой метод содержит самостоятельное присвоение поля

FB.SA_FIELD_SELF_COMPARISON

Детектор определяет ошибку, в ходе которой метод сравнивает поле само с собой и может указывать на опечатку или логическую ошибку. Убедитесь, что вы сравниваете правильные вещи.

FB.SA_LOCAL_SELF_ASSIGNMENT_INSTEAD_OF_FIELD

Детектор определяет ошибку, в ходе которой метод содержит самостоятельное присвоение локальной переменной и есть поле с таким же именем.

FB.SA_LOCAL_SELF_COMPARISON

Детектор определяет ошибку, в ходе которой метод сравнивает локальную переменную с самой собой и может указывать на опечатку или логическую ошибку. Убедитесь, что вы сравниваете правильные вещи.

FB.SA_LOCAL_SELF_COMPUTATION

Детектор определяет ошибку, в ходе которой метод выполняет бессмысленное вычисление локальной переменной с другой ссылкой на ту же переменную (например, x&x или x-x). Из-за характера вычислений эта

операция кажется бессмысленной и может указывать на опечатку или логическую ошибку. Дважды проверьте расчет.

FB.SF_DEAD_STORE_DUE_TO_SWITCH_FALLTHROUGH

Детектор определяет ошибку, в ходе которой значение, сохраненное в предыдущем случае переключения, здесь перезаписывается из-за сбоя переключения. Вполне вероятно, что вы забыли поставить перерыв или возврат в конце предыдущего дела.

FB.SF_DEAD_STORE_DUE_TO_SWITCH_FALLTHROUGH_TO_THROW

Детектор определяет ошибку, в ходе которой значение, сохраненное в предыдущем случае переключения, здесь игнорируется из-за перехода переключателя в место, где генерируется исключение. Вполне вероятно, что вы забыли поставить перерыв или возврат в конце предыдущего дела.

FB.SQL_BAD_PREPARED_STATEMENT_ACCESS

Детектор определяет ошибку, в ходе которой был выполнен вызов метода setXXX подготовленного оператора, где индекс параметра равен 0. Поскольку индексы параметров начинаются с индекса 1, это всегда является ошибкой.

FB.STI_INTERRUPTED_ON_UNKNOWNTHREAD

Детектор определяет ошибку, в ходе которой метод вызывает метод Thread.interrupted() для объекта Thread, который выглядит как объект Thread, который не является текущим потоком. Поскольку метод прерывания() является статическим, метод прерывания будет вызываться для другого объекта, отличного от того, который задумал автор.

FB.SE_METHOD_MUST_BE_PRIVATE

Детектор определяет ошибку, в ходе которой класс реализует интерфейс Serializable и определяет метод пользовательской сериализации/десериализации. Но поскольку этот метод не объявлен закрытым, он будет молча игнорироваться API сериализации/десериализации.

FB.UMAC_UNCALLABLE_METHOD_OF_ANONYMOUS_CLASS

Детектор определяет ошибку, в ходе которой анонимный класс определяет метод, который не вызывается напрямую и не переопределяет метод в суперклассе. Поскольку методы других классов не могут напрямую вызывать методы, объявленные в анонимном классе, кажется, что этот метод не вызывается. Метод может быть просто мертвым кодом, но также возможно, что метод предназначен для переопределения метода,

объявленного в суперклассе, и из-за опечатки или другой ошибки метод фактически не переопределяет метод, для которого он предназначен. .

FB.UR_UNINIT_READ

Детектор определяет ошибку, в ходе которой Этот конструктор считывает поле, которому еще не присвоено значение. Это часто происходит, когда программист по ошибке использует поле вместо одного из параметров конструктора.

FB.UR_UNINIT_READ_CALLED_FROM_SUPER_CONSTRUCTOR

Детектор определяет ошибку, в ходе которой метод вызывается в конструкторе суперкласса. На данный момент поля класса еще не инициализированы.

FB.DMI_INVOKING_TOSTRING_ON_ANONYMOUS_ARRAY

Детектор определяет ошибку, в ходе которой код вызывает toString в (анонимном) массиве. Вызов toString для массива приводит к довольно бесполезному результату.

FB.DMI_INVOKING_TOSTRING_ON_ARRAY

Детектор определяет ошибку, в ходе которой код вызывает toString для массива, что генерирует довольно бесполезный результат.

FB.UWF_NULL_FIELD

Детектор определяет ошибку, в ходе которой все записи в данное поле имеют постоянное значение null, поэтому все операции чтения поля возвращают значение null. Проверьте наличие ошибок или удалите его, если он бесполезен.

FB.UWF_UNWRITTEN_FIELD

Детектор определяет ошибку, в ходе которой поле никогда не записывается. Все его чтения будут возвращать значение по умолчанию. Проверьте наличие ошибок (нужно ли было его инициализировать?), или удалите, если он бесполезен.

FB.VA_PRIMITIVE_ARRAY_PASSED_TO_OBJECT_VARARG

Детектор определяет ошибку, в ходе которой код передает примитивный массив функции, которая принимает переменное количество аргументов объекта. Это создает массив длиной один для хранения примитивного массива и передает его функции.

FB.DM_DEFAULT_ENCODING

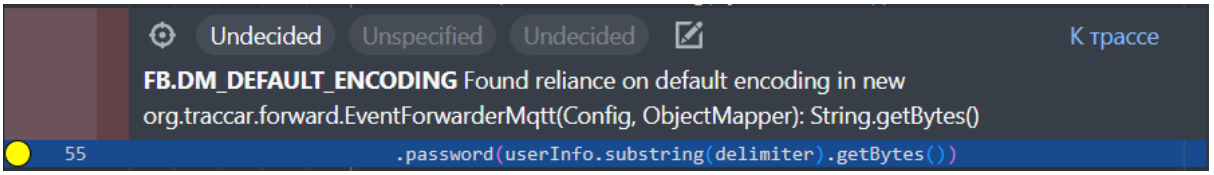


Рисунок 35 –Детектор FB.DM_DEFAULT_ENCODING в интерфейсе Svacer

Детектор определяет ошибку, в ходе которой вызов метода, который выполняет преобразование байта в строку (или строку в байт) и предполагает, что кодировка платформы по умолчанию подходит. Это приведет к тому, что поведение приложения будет различаться на разных платформах. Используйте альтернативный API и явно укажите имя набора символов или объект Charset.

FB.DP_CREATE_CLASSLOADER_INSIDE_DO_PRIVILEGED

Детектор определяет ошибку, в ходе которой код создает загрузчик классов, которому требуется разрешение, если установлено управление безопасностью. Если этот код может быть вызван кодом, не имеющим разрешений безопасности, то создание загрузчика классов должно происходить внутри блока doPrivileged.

FB.FI_PUBLIC_SHOULD_BE_PROTECTED

Детектор определяет ошибку, в ходе которой Метод Finalize() класса должен иметь защищенный, а не общедоступный доступ.

FB.AT_OPERATION_SEQUENCE_ON_CONCURRENT_ABSTRACTION

Детектор определяет ошибку, в ходе которой код содержит последовательность вызовов параллельной абстракции (например, параллельной хэш-карты). Эти вызовы не будут выполняться атомарно.

FB.DL_SYNCHRONIZATION_ON_BOOLEAN

Детектор определяет ошибку, в ходе которой код синхронизируется с помощью упакованной примитивной константы, например логического значения.

FB.DL_SYNCHRONIZATION_ON_BOXED_PRIMITIVE

Детектор определяет ошибку, в ходе которой код синхронизируется с помощью упакованной примитивной константы, например целого числа.

FB.DL_SYNCHRONIZATION_ON_SHARED_CONSTANT

Детектор определяет ошибку, в ходе которой код синхронизируется по строковому литералу.

FB.DL_SYNCHRONIZATION_ON_UNSHARED_BOXED_PRIMITIV E

Детектор определяет ошибку, в ходе которой Код синхронизируется с очевидно необщим упакованным примитивом, таким как Integer.

FB.ESYNC_EMPTY_SYNC

Детектор определяет ошибку, в ходе которой код содержит пустой синхронизированный блок

FB.JLM_JSR166_LOCK_MONITORENTER

Детектор определяет ошибку, в ходе которой метод выполняет синхронизацию объекта, реализующего java.util.concurrent.locks.Lock. Такой объект блокируется/разблокируется с помощью метода Acquire()/release(), а не с помощью конструкции Synchronized (...).

FB.JLM_JSR166_UTILCONCURRENT_MONITORENTER

Детектор определяет ошибку, в ходе которой метод выполняет синхронизацию объекта, который является экземпляром класса из пакета java.util.concurrent (или его подклассов). Экземпляры этих классов имеют свои собственные механизмы управления параллелизмом, которые ортогональны синхронизации, обеспечиваемой ключевым словом Java Synchronized. Например, синхронизация по AtomicBoolean не помешает другим потокам изменять AtomicBoolean.

FB.LI_LAZY_INIT_STATIC

Детектор определяет ошибку, в ходе которой метод содержит несинхронизированную ленивую инициализацию энергонезависимого статического поля. Поскольку компилятор или процессор может изменить порядок инструкций, потоки не гарантируют, что увидят полностью инициализированный объект, если метод может быть вызван несколькими потоками. Чтобы исправить проблему, вы можете сделать поле нестабильным.

FB.LI_LAZY_INIT_UPDATE_STATIC

Детектор определяет ошибку, в ходе которой метод содержит несинхронизированную ленивую инициализацию статического поля. После установки поля объект, хранящийся в этом месте, дополнительно обновляется или доступен. Настройка поля видна другим потокам, как только она установлена. Если дальнейшие обращения в методе, устанавливающем поле, служат для инициализации объекта, то у вас очень серьезная ошибка многопоточности, если только что-то еще не мешает любому другому потоку получить доступ к хранимому объекту до тех пор, пока он не будет полностью инициализирован.

FB.MWN_MISMATCHED_NOTIFY

Детектор определяет ошибку, в ходе которой метод вызывает Object.notify() или Object.notifyAll() без явной блокировки объекта. Вызов notify() или notifyAll() без удержания блокировки приведет к выдаче исключения IllegalMonitorStateException.

FB.MWN_MISMATCHED_WAIT

Детектор определяет ошибку, в ходе которой метод вызывает Object.wait() без явной блокировки объекта. Вызов wait() без удержания блокировки приведет к созданию исключения IllegalMonitorStateException.

FB.NN_NAKED_NOTIFY

Детектор определяет ошибку, в ходе которой вызов notify() или notifyAll() был выполнен без какого-либо (кажущегося) сопутствующего изменения состояния изменяемого объекта. Обычно вызов метода уведомления на мониторе выполняется потому, что какое-то условие, которого ожидает другой поток, стало истинным. Однако чтобы условие было значимым, оно должно включать объект кучи, видимый обоим потокам.

FB.NP_SYNC_AND_NULL_CHECK_FIELD

Детектор определяет ошибку, в ходе которой обнаруживается факт, что поскольку поле синхронизировано, оно вряд ли будет нулевым. Если он равен нулю, а затем синхронизируется с NullPointerException, будет выброшено исключение, и проверка будет бессмысленной. Лучше синхронизироваться на другом поле.

FB.RS_READOBJECT_SYNC

Детектор определяет ошибку, в ходе которой сериализуемый класс определяет readObject(), который синхронизируется. По определению, объект, созданный путем десериализации, доступен только одному потоку, поэтому нет необходимости синхронизировать readObject(). Если метод readObject() сам по себе приводит к тому, что объект становится видимым для другого потока, это пример очень сомнительного стиля кодирования.

FB.RV_RETURN_VALUE_OF_PUTIFABSENT_IGNORED

Детектор определяет уязвимость, для исправления которой требуется знать, что метод putIfAbsent обычно используется, чтобы гарантировать, что с данным ключом связано одно значение (первое значение, для которого put, если оно отсутствует, завершается успешно). Если вы проигнорируете возвращаемое значение и сохраните ссылку на переданное значение, вы рискуете сохранить значение, отличное от того, которое связано с ключом в карте. Если имеет значение, какой из них вы используете, и вы используете

тот, который не сохранен в карте, ваша программа будет вести себя неправильно.

FB.RU_INVOKE_RUN

Детектор определяет ошибку, в ходе которой метод явно вызывает run() для объекта. В общем, классы реализуют интерфейс Runnable, потому что их метод run() будет вызываться в новом потоке, и в этом случае Thread.start() является подходящим методом для вызова.

FB.SC_START_IN_CTOR

Детектор определяет ошибку, в ходе которой конструктор запускает поток. Скорее всего, это будет неправильно, если класс когда-либо будет расширен/подклассифицирован, поскольку поток будет запущен до запуска конструктора подкласса.

FB.SP_SPIN_ON_FIELD

Детектор определяет ошибку, в ходе которой метод вращается в цикле, который считывает поле. Компилятор может законно вывести чтение из цикла, превратив код в бесконечный цикл. Класс следует изменить, чтобы он использовал правильную синхронизацию (включая вызовы ожидания и уведомления).

FB.STCAL_INVOKE_ON_STATIC_CALENDAR_INSTANCE

Детектор определяет ошибку, в ходе которой несмотря на то, что JavaDoc не содержит никаких намеков на это, календари по своей сути небезопасны для многопоточного использования. Детектор обнаружил вызов экземпляра Calendar, полученный через статическое поле. Это выглядит подозрительно.

FB.STCAL_INVOKE_ON_STATIC_DATE_FORMAT_INSTANCE

Детектор определяет ошибку, в ходе которой DateFormats по своей сути небезопасны для многопоточного использования. Детектор обнаружил вызов экземпляра DateFormat, полученного через статическое поле. Это выглядит подозрительно.

FB.STCAL_STATIC_SIMPLE_DATE_FORMAT_INSTANCE

Детектор определяет ошибку, в ходе которой как указано в JavaDoc, DateFormats по своей сути небезопасны для многопоточного использования. Совместное использование одного экземпляра за границами потоков без надлежащей синхронизации приведет к нестабильному поведению приложения.

FB.SWL_SLEEP_WITH_LOCK_HELD

Детектор определяет ошибку, в ходе которой метод вызывает Thread.sleep() с удержанной блокировкой. Это может привести к очень низкой производительности и масштабируемости или к взаимоблокировке, поскольку другие потоки могут ожидать получения блокировки. Гораздо лучше вызвать функцию wait() для блокировки, которая снимает блокировку и позволяет запускаться другим потокам.

FB.TLW_TWO_LOCK_WAIT

Детектор определяет ошибку, в ходе которой ожидание на мониторе, пока удерживаются две блокировки, может привести к взаимоблокировке. Выполнение ожидания снимает только блокировку ожидающего объекта, но не какие-либо другие блокировки. Это не обязательно ошибка, но ее стоит внимательно изучить.

FB.UG_SYNC_SET_UNSYNC_GET

Детектор определяет ошибку, в ходе которой класс содержит методы get и set с одинаковыми именами, где метод set синхронизируется, а метод get — нет. Это может привести к некорректному поведению во время выполнения, поскольку вызывающие метод get не обязательно увидят согласованное состояние объекта. Метод get следует сделать синхронизированным.

FB.UL_UNRELEASED_LOCK

Детектор определяет ошибку, в ходе которой метод получает блокировку JSR-166 (java.util.concurrent), но не снимает ее на всех путях выхода из метода.

FB.UW_UNCOND_WAIT

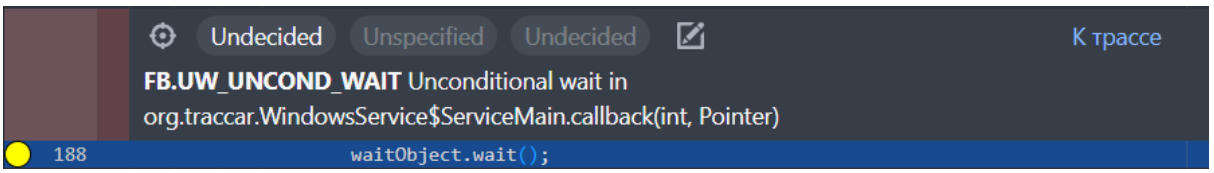


Рисунок 36 –Детектор FB.UW_UNCOND_WAIT в интерфейсе Svacer

Ошибка "unconditional wait" (безусловное ожидание) указывает на то, что в вашем коде присутствует операция ожидания (например, использование метода wait() в Java), которая выполняется без проверки какого-либо условия. Это может привести к блокировке (замораживанию) выполнения программы, если не выполнены определенные условия для продолжения работы.

FB.WL_USING_GETCLASS_RATHER_THAN_CLASS_LITERAL

Детектор определяет ошибку, в ходе которой метод экземпляра синхронизируется с this.getClass(). Если этот класс является подклассом, подклассы будут синхронизироваться с объектом класса для подкласса, что вряд ли было задумано.

FB.WA_AWAIT_NOT_IN_LOOP

Детектор определяет ошибку, в ходе которой метод содержит вызов java.util.concurrent.await() (или его вариантов), который не находится в цикле. Если объект используется для нескольких условий, то условие, которого вызывающий объект намеревался ожидать, может оказаться не тем, которое произошло на самом деле.

FB.WA_NOT_IN_LOOP

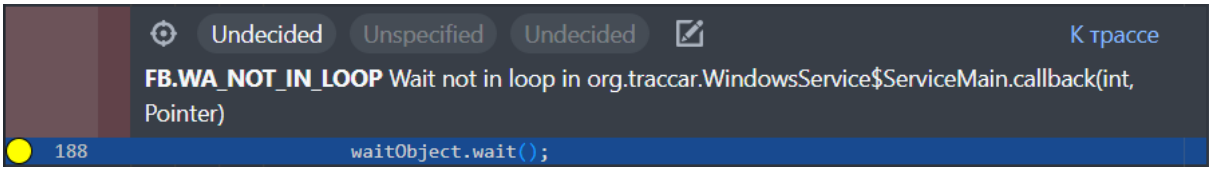


Рисунок 37 –Детектор FB.WA_NOT_IN_LOOP в интерфейсе Svacer

Ошибка "Wait not in loop" или "Wait not in loop in" указывает на то, что операция wait() используется внутри кода, но без обертывания её в цикл. Обычно, использование wait() должно быть в цикле, чтобы предотвратить "ложные пробуждения" (spurious wake-ups) и правильно обрабатывать условия, по которым поток должен продолжить выполнение.

FB.BX_BOXING_IMMEDIATELY_UNBOXED

Детектор определяет ошибку, в ходе которой примитив упаковывается, а затем сразу же распаковывается. Вероятно, это связано с ручным боксом в месте, где требуется неупакованное значение, что вынуждает компилятор немедленно отменить работу бокса.

FB.BX_BOXING_IMMEDIATELY_UNBOXED_TO_PERFORM_COERCION

Детектор определяет ошибку, в ходе которой примитивное упакованное значение, созданное и затем немедленно преобразованное в другой тип примитива (например, new Double(d).intValue()). Просто выполните прямое примитивное приведение (например, (int) d)

FB.BX_UNBOXING_IMMEDIATELY_REBOXED

Детектор определяет ошибку, в ходе которой упакованное значение распаковывается, а затем немедленно упаковывается заново.

FB.DM_NUMBER_CTOR

Детектор определяет ошибку, в ходе которой использование new Integer(int) гарантированно всегда приводит к созданию нового объекта, тогда как Integer.valueOf(int) позволяет кэшировать значения, выполняемые компилятором, библиотекой классов или JVM. Использование кэшированных значений позволяет избежать выделения объектов, и код будет работать быстрее.

FB.DMI_BLOCKING_METHODS_ON_URL

Детектор определяет ошибку, в ходе которой методы URL-адреса Equals и hashCode выполняют разрешение доменных имен, что может привести к значительному снижению производительности.

FB.DMI_COLLECTION_OF_URLS

Детектор определяет ошибку, в ходе которой метод или поле является или использует карту или набор URL-адресов. Поскольку оба метода URL-адреса, Equals и hashCode, выполняют разрешение доменных имен, это может привести к значительному снижению производительности.

FB.DM_BOOLEAN_CTOR

Детектор определяет ошибку, в ходе которой создание новых экземпляров java.lang.Boolean тратит впустую память, поскольку логические объекты неизменяемы и существует только два полезных значения этого типа. Вместо этого используйте метод Boolean.valueOf() (или автоупаковку Java 5) для создания логических объектов.

FB.DM_GC

Детектор определяет ошибку, в ходе которой код явно вызывает сбор мусора. За исключением конкретного использования в бенчмаркинге, это очень сомнительно.

FB.DM_NEW_FOR_GETCLASS

Детектор определяет ошибку, в ходе которой этот метод выделяет объект только для вызова метода getClass(), чтобы получить для него объект класса. Проще получить доступ к свойству .class класса.

FB.DM_NEXTINT_VIA_NEXTDOUBLE

Детектор определяет ошибку, в ходе которой если r является java.util.Random, вы можете сгенерировать случайное число от 0 до n-1, используя r.nextInt(n), а не используя (int)(r.nextDouble() * n).

FB.DM_STRING_CTOR

Детектор определяет ошибку, в ходе которой использование конструктора java.lang.String(String) тратит впустую память, поскольку созданный таким образом объект будет функционально неотличим от строки, переданной в качестве параметра. Просто используйте аргумент String напрямую.

FB.DM_STRING_TOSTRING

Детектор определяет ошибку, в ходе которой вызов String.toString() является избыточной операцией. Просто используйте строку.

FB.DM_STRING_VOID_CTOR

Детектор определяет ошибку, в ходе которой создание нового объекта java.lang.String с использованием конструктора без аргументов тратит впустую память, поскольку созданный таким образом объект будет функционально неотличим от пустой строковой константы "". Java гарантирует, что идентичные строковые константы будут представлены одним и тем же объектом String. Поэтому вам следует просто использовать константу пустой строки напрямую.

FB.HSC_HUGE_SHARED_STRING_CONSTANT

Детектор определяет ошибку, в ходе которой большая строковая константа дублируется в нескольких файлах классов. Вероятно, это связано с тем, что последнее поле инициализируется строковой константой, а язык Java требует, чтобы все ссылки на последнее поле из других классов были встроены в этот файл классов.

FB.SBSC_USE_STRINGBUFFER_CONCATENATION

Детектор определяет ошибку, в ходе которой кажется, что этот метод строит строку, используя конкатенацию в цикле. На каждой итерации строка преобразуется в StringBuffer/StringBuilder, добавляется и преобразуется обратно в строку. Это может привести к квадратичной стоимости числа итераций, поскольку растущая строка копируется заново на каждой итерации.

FB.SIC_INNER_SHOULD_BE_STATIC

Детектор определяет ошибку, в ходе которой класс является внутренним классом, но не использует встроенную ссылку на объект, который его создал. Эта ссылка увеличивает размер экземпляров класса и может сохранять ссылку на объект-создатель дольше, чем необходимо. Если возможно, класс следует сделать статическим.

FB.SS_SHOULD_BE_STATIC

Детектор определяет ошибку, в ходе которой класс содержит последнее поле экземпляра, которое инициализируется статическим значением времени компиляции. Рассмотрите возможность сделать поле статическим.

FB.URF_UNREAD_FIELD

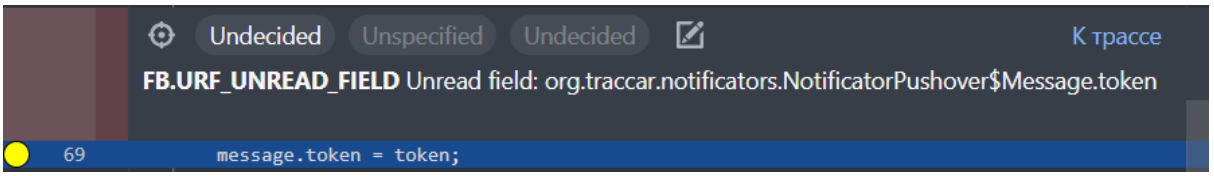


Рисунок 38 –Детектор FB.URF_UNREAD_FIELD в интерфейсе Svacer

Ошибка "unread field" указывает на то, что в вашем коде есть объявление поля (переменной класса), но это поле не используется (т.е., оно "не читается"). Это может быть признаком лишних или неиспользуемых данных в вашем коде, что может привести к избыточности и понижению читаемости.

FB.UUF_UNUSED_FIELD

Детектор определяет ошибку, в ходе которой частный метод никогда не вызывается. Хотя вполне возможно, что метод будет вызван посредством отражения, более вероятно, что метод никогда не будет использоваться и его следует удалить.

FB.WMI_WRONG_MAP_ITERATOR

Детектор определяет ошибку, в ходе которой метод получает доступ к значению записи Map, используя ключ, полученный из итератора keySet. Более эффективно использовать итератор в наборе записей карты, чтобы избежать поиска Map.get(key).

FB.BC_BAD_CAST_TO_ABSTRACT_COLLECTION

Детектор определяет ошибку, в ходе которой код преобразует коллекцию в абстрактную коллекцию (например, List, Set или Map). Убедитесь, что вам гарантировано, что объект имеет тип, к которому вы выполняете приведение. Если все, что вам нужно, — это иметь возможность перебирать коллекцию, вам не нужно приводить ее к набору или списку.

FB.BC_BAD_CAST_TO_CONCRETE_COLLECTION

Детектор определяет ошибку, в ходе которой код преобразует абстрактную коллекцию (например, Collection, List или Set) в конкретную конкретную реализацию (например, ArrayList или HashSet). Это может быть неправильно и может сделать ваш код хрупким, поскольку в будущем будет

сложнее переключиться на другие конкретные реализации. Если у вас нет для этого особой причины, просто используйте абстрактный класс коллекции.

FB.BC_UNCONFIRMED_CAST

Детектор определяет ошибку, в ходе которой приведение не отмечено, и не все экземпляры типа, из которого происходит приведение, могут быть приведены к типу, к которому оно приводится. Убедитесь, что логика вашей программы гарантирует, что это приведение не завершится неудачей.

FB.BC_VACUOUS_INSTANCEOF

Детектор определяет ошибку, в ходе которой тест экземпляра всегда будет возвращать значение true (если только проверяемое значение не равно нулю). Хотя это и безопасно, убедитесь, что это не является признаком какого-то недопонимания или какой-либо другой логической ошибки. Если вы действительно хотите проверить значение на нулевое значение, возможно, было бы яснее выполнить нулевую проверку, а не проверку экземпляра.

FB.ICAST_QUESTIONABLE_UNSIGNED_RIGHT_SHIFT

Детектор определяет ошибку, в ходе которой код выполняет беззнаковый сдвиг вправо, результат которого затем преобразуется в короткое число или байт, при этом старшие биты результата отбрасываются. Поскольку старшие биты отбрасываются, разницы между знаковым и беззнаковым сдвигом вправо может не быть (в зависимости от размера сдвига).

FB.DB_DUPLICATE_BRANCHES

Детектор определяет ошибку, в ходе которой метод использует один и тот же код для реализации двух ветвей условного перехода. Убедитесь, что это не ошибка кодирования.

FB.DLS_DEAD_LOCAL_STORE

Детектор определяет ошибку, в ходе которой инструкция присваивает значение локальной переменной, но это значение не считывается и не используется ни в одной последующей инструкции. Часто это указывает на ошибку, поскольку вычисленное значение никогда не используется.

FB.DLS_DEAD_LOCAL_STORE_IN_RETURN

Детектор определяет ошибку, в ходе которой оператор присваивает значение локальной переменной в операторе возврата. Это назначение не имеет никакого эффекта. Пожалуйста, убедитесь, что это утверждение соответствует действительности.

FB.DLS_DEAD_LOCAL_STORE_OF_NULL

Детектор определяет ошибку, в ходе которой код сохраняет значение null в локальной переменной, и сохраненное значение не считывается. Возможно, это хранилище было введено для помощи сборщику мусора, но начиная с Java SE 6.0 оно больше не нужно и не полезно.

FB.DLS_DEAD_LOCAL_STORE_SHADOWS_FIELD

Детектор определяет ошибку, в ходе которой инструкция присваивает значение локальной переменной, но это значение не считывается и не используется ни в одной последующей инструкции. Часто это указывает на ошибку, поскольку вычисленное значение никогда не используется. Существует поле с тем же именем, что и у локальной переменной.

FB.DMI_HARDCODED_ABSOLUTE_FILENAME

Детектор определяет ошибку, в ходе которой код создает объект File, используя жестко запрограммированный абсолютный путь.

FB.DMI_NONSERIALIZABLE_OBJECT_WRITTEN

Детектор определяет ошибку, в ходе которой код передает несериализуемый объект методу ObjectOutput.writeObject. Если объект действительно несериализуем, возникнет ошибка.

FB.DMI_USELESS_SUBSTRING

Детектор определяет ошибку, в ходе которой код вызывает substring(0) для строки, которая возвращает исходное значение.

FB.DMI_THREAD_PASSED_WHERE_RUNNABLE_EXPECTED

Детектор определяет ошибку, в ходе которой объект Thread передается в качестве параметра методу, в котором ожидается Runnable. Это довольно необычно и может указывать на логическую ошибку или вызывать неожиданное поведение.

FB.EQ_DOESNT_OVERRIDE_EQUALS

Детектор определяет ошибку, в ходе которой класс расширяет класс, который определяет метод равенства и добавляет поля, но не определяет сам метод равенства. Таким образом, равенство экземпляров этого класса будет игнорировать идентичность подкласса и добавленных полей. Убедитесь, что это именно то, что задумано, и что вам не нужно переопределять метод равенства. Даже если вам не нужно переопределять метод равенства, все равно рассмотрите возможность его переопределения, чтобы задокументировать тот факт, что метод равенства для подкласса просто возвращает результат вызова super.equals(o).

FB.EQ_UNUSUAL

Детектор определяет ошибку, в ходе которой класс не использует никаких шаблонов, которые мы распознаем для проверки совместимости типа аргумента с типом объекта this. Возможно, в этом коде нет ничего плохого, но его стоит просмотреть.

FB.FE_FLOATING_POINT_EQUALITY

Детектор определяет ошибку, в ходе которой операция сравнивает два значения с плавающей запятой на предмет равенства. Поскольку вычисления с плавающей запятой могут включать округление, вычисленные значения float и double могут быть неточными. Для значений, которые должны быть точными, например денежных значений, рассмотрите возможность использования типа с фиксированной точностью, такого как BigDecimal. Для значений, которые не должны быть точными, рассмотрите возможность сравнения на равенство в некотором диапазоне, например: if (Math.abs(x - y) < .0000001).

FB.IA_AMBIGUOUS_INVOCATION_OF_INHERITED_OR_OUTER_METHOD

Детектор определяет ошибку, в ходе которой внутренний класс вызывает метод, который может быть преобразован либо в унаследованный метод, либо в метод, определенный во внешнем классе. Например, вы вызываете foo(17), который определен как в суперклассе, так и во внешнем методе. Семантика Java разрешает вызов унаследованного метода, но это может быть не то, что вы хотите.

FB.IC_INIT_CIRCULARITY

Детектор определяет ошибку, в ходе которой в статических инициализаторах двух классов, на которые ссылается экземпляр ошибки, была обнаружена цикличность. Из-за такой цикличности может возникнуть множество видов неожиданного поведения.

FB.ICAST_IDIV_CAST_TO_DOUBLE

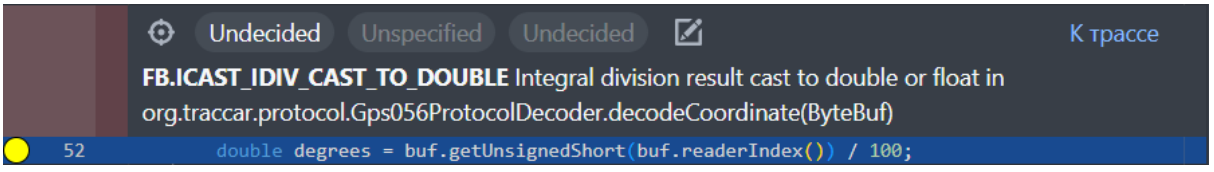


Рисунок 39 –Детектор FB.ICAST_IDIV_CAST_TO_DOUBLE в интерфейсе Svacer

Ошибка "integral division result cast to double or float" указывает на то, что результат целочисленного деления (например, int или long на int или long) затем явным образом преобразуется в тип с плавающей точкой (double или

float). Это может привести к потере точности или представления данных, поскольку целочисленное деление не сохраняет десятичные части.

FB.ICAST_INTEGER_MULTIPLY_CAST_TO_LONG

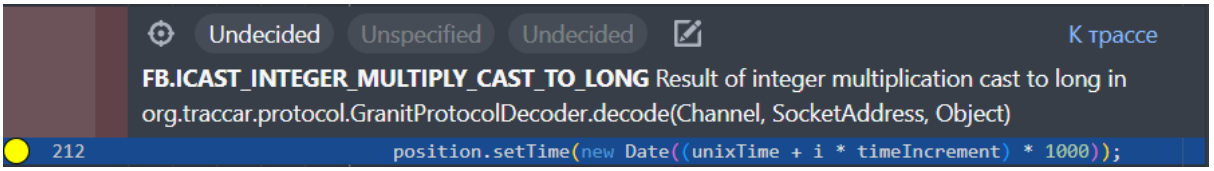


Рисунок 40 –Детектор FB.ICAST_INTEGER_MULTIPLY_CAST_TO_LONG в интерфейсе Svacer

Ошибка "result of integer multiplication cast to long" указывает на то, что результат умножения целых чисел затем приводится к типу long. Это предупреждение может появиться, когда результат умножения целочисленных значений, которые могут поместиться в типе int, приводится к типу long.

FB.IIO_INEFFICIENT_INDEX_OF

Детектор определяет ошибку, в ходе которой код передает константную строку длины 1 в String.indexOf(). Более эффективно использовать целочисленную реализацию String.indexOf(). ф. е. вызовите myString.indexOf('.') вместо myString.indexOf(".")

FB.IIO_INEFFICIENT_LAST_INDEX_OF

Детектор определяет ошибку, в ходе которой код передает константную строку длиной 1 в String.lastIndexOf(). Более эффективно использовать целочисленные реализации String.lastIndexOf(). ф. е. вызовите myString.lastIndexOf('.') вместо myString.lastIndexOf(".")

FB.IM_AVERAGE_COMPUTATION_COULD_OVERFLOW

Детектор определяет ошибку, в ходе которой код вычисляет среднее значение двух целых чисел, используя либо деление, либо сдвиг вправо со знаком, а затем использует результат в качестве индекса массива. Если усредняемые значения очень велики, это может привести к переполнению (что приведет к вычислению отрицательного среднего). Предполагая, что результат должен быть неотрицательным, вместо этого вы можете использовать беззнаковый сдвиг вправо. Другими словами, вместо использования (низкий+высокий)/2 используйте (низкий+высокий) >>> 1

FB.IM_BAD_CHECK_FOR_ODD

Детектор определяет ошибку, в ходе которой в коде используется x % 2 == 1, чтобы проверить, является ли значение нечетным, но это не работает для отрицательных чисел (например, (-5) % 2 == -1). Если этот код

предназначен для проверки на нечетность, рассмотрите возможность использования (x & 1) == 1 или x % 2 != 0.

FB.INT_BAD_REM_BY_1

Детектор определяет ошибку, в ходе которой любое выражение (exp % 1) гарантированно всегда возвращает ноль.

FB.INT_VACUOUS_BIT_OPERATION

Детектор определяет ошибку, в ходе которой целочисленная битовая операция (и, или, или исключающее или), которая не выполняет никакой полезной работы (например, v & 0xffffffff).

FB.INT_VACUOUS_COMPARISON

Детектор определяет ошибку, в ходе которой существует целочисленное сравнение, которое всегда возвращает одно и то же значение (например, x <= Integer.MAX_VALUE).

FB.NP_PARAMETER_MUST_BE_NONNULL_BUT_MARKED_AS_NULLABLE

Детектор определяет ошибку, в ходе которой параметр всегда используется таким образом, чтобы он был ненулевым, но параметр явно помечен как имеющий значение Nullable. Либо использование параметра, либо аннотация неверны.

FB.NP_UNWRITTEN_PUBLIC_OR_PROTECTED_FIELD

Детектор определяет ошибку, в ходе которой программа разыменовывает общедоступное или защищенное поле, в которое, похоже, никогда не было записано ненулевое значение. Если поле не инициализировано с помощью какого-либо механизма, не видимого при анализе, разыменование этого значения приведет к возникновению исключения нулевого указателя.

FB.NS_DANGEROUS_NON_SHORT_CIRCUIT

Детектор определяет ошибку, в ходе которой в коде используется некороткая логика (например, & или |), а не упрощенная логика (&& или ||). Кроме того, кажется возможным, что в зависимости от значения левой части вы можете не захотеть оценивать правую часть (поскольку она будет иметь побочные эффекты, может вызвать исключение или может быть дорогостоящей.

FB.QF_QUESTIONABLE_FOR_LOOP

Детектор определяет ошибку, в ходе которой другая переменная инициализируется и проверяется циклом for.

FB.RCN_REDUNDANT_COMPARISON_TWO_NULL_VALUES

Детектор определяет ошибку, в ходе которой метод содержит избыточное сравнение двух ссылок, обе из которых определенно равны нулю.

FB.RCN_REDUNDANT_NULLCHECK_OF_NONNULL_VALUE

Детектор определяет ошибку, в ходе которой метод содержит избыточную проверку известного ненулевого значения на соответствие постоянному нулю.

FB.RCN_REDUNDANT_NULLCHECK_OF_NULL_VALUE

Детектор определяет ошибку, в ходе которой метод содержит избыточную проверку известного нулевого значения на соответствие постоянному нулю.

FB.REC_CATCH_EXCEPTION

Детектор определяет ошибку, в ходе которой метод использует блок try-catch, который перехватывает объекты Exception, но Exception не генерируется внутри блока try, а RuntimeException не перехватывается явным образом. Распространенной ошибкой является выражение try { ... } catch (Exception e) { Something } как сокращение для перехвата нескольких типов исключений, каждый из блоков catch которых идентичен, но эта конструкция также случайно перехватывает RuntimeException. , маскируя потенциальные ошибки.

FB.RV_DONT_JUST_NULL_CHECK_READLINE

Детектор определяет ошибку, в ходе которой значение, возвращаемое readLine, отбрасывается после проверки, не является ли возвращаемое значение ненулевым. Почти во всех ситуациях, если результат не равен NULL, вы захотите использовать это ненулевое значение. Повторный вызов readLine даст вам другую строку.

FB.SA_FIELD_DOUBLE_ASSIGNMENT

Детектор определяет ошибку, в ходе которой метод содержит двойное присвоение поля

FB.SA_LOCAL_DOUBLE_ASSIGNMENT

Детектор определяет ошибку, в ходе которой метод содержит двойное присвоение локальной переменной

FB.SA_LOCAL_SELF_ASSIGNMENT

Детектор определяет ошибку, в ходе которой метод содержит самостоятельное присвоение локальной переменной локальной переменной.

FB.SF_SWITCH_FALLTHROUGH

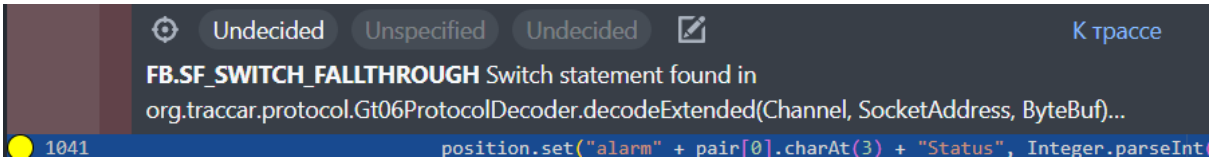


Рисунок 41 –Детектор FB.SF_SWITCH_FALLTHROUGH в интерфейсе Svacer

Ошибка "statement found where one case falls through to the next case" указывает на то, что в вашем коде внутри конструкции switch для одного из случаев (case) отсутствует оператор break или return, который предотвратил бы "проброс" управления к следующему случаю.

FB.ST_WRITE_TO_STATIC_FROM_INSTANCE_METHOD

Детектор определяет ошибку, в ходе которой метод экземпляра записывает в статическое поле. Это сложно исправить, если манипулируют несколькими экземплярами, и, как правило, это плохая практика.

FB.SE_PRIVATE_READ_RESOLVE_NOT_INHERITED

Детектор определяет ошибку, в ходе которой класс определяет частный метод readResolve. Поскольку он является частным, он не будет унаследован подклассами. Это может быть намеренно и нормально, но следует проверить, чтобы убедиться, что это именно то, что задумано.

FB.UCF_USELESS_CONTROL_FLOW

Детектор определяет ошибку, в ходе которой метод содержит бесполезный оператор потока управления, где поток управления продолжается в том же месте независимо от того, выбрана ветвь или нет. Например, это вызвано наличием пустого блока операторов для оператора if.

FB.UCF_USELESS_CONTROL_FLOW_NEXT_LINE

Детектор определяет ошибку, в ходе которой метод содержит бесполезный оператор потока управления, в котором поток управления следует к той же или следующей строке независимо от того, выбрана ли ветвь или нет. Часто это вызвано непреднамеренным использованием пустого оператора в качестве тела оператора if.

FB.URF_UNREAD_PUBLIC_OR_PROTECTED_FIELD

Детектор определяет ошибку, в ходе которой поле никогда не читается. Поле является общедоступным или защищенным, поэтому, возможно, оно

предназначено для использования с классами, не рассматриваемыми как часть анализа. Если нет, рассмотрите возможность удаления его из класса.

FB.UUF_UNUSED_PUBLIC_OR_PROTECTED_FIELD

Детектор определяет ошибку, в ходе которой поле никогда не используется. Поле является общедоступным или защищенным, поэтому, возможно, оно предназначено для использования с классами, не рассматриваемыми как часть анализа. Если нет, рассмотрите возможность удаления его из класса.

FB.UWF_UNWRITTEN_PUBLIC_OR_PROTECTED_FIELD

Детектор определяет ошибку, в ходе которой в общедоступное/защищенное поле не было обнаружено никаких записей. Все его чтения будут возвращать значение по умолчанию. Проверьте наличие ошибок (нужно ли было его инициализировать?), или удалите, если он бесполезен.

FB.DLS_DEAD_LOCK_STORE

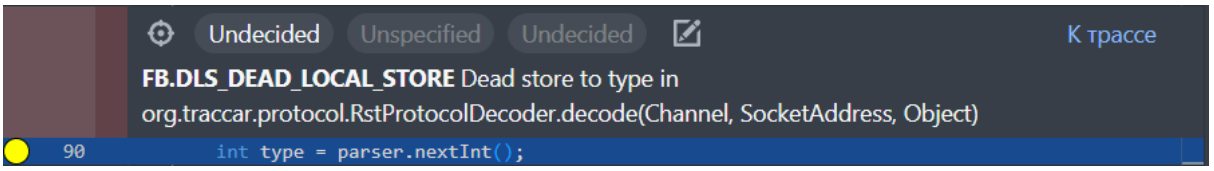


Рисунок 42 –Детектор FB.DLS_DEAD_LOCK_STORE в интерфейсе Svacer

Ошибка "found reliance on default encoding" означает, что в вашем коде используется операция, которая предполагает использование кодировки по умолчанию (default encoding). Это может быть проблемой, поскольку кодировка по умолчанию зависит от окружения выполнения программы и может привести к несовместимостям при переносе кода между разными системами.

7.1.3. NORMAL ошибки

Ошибки класса “Normal” (Рисунок 42) - ошибки, которые могут не дать о себе знать при идеально верном использовании программы. Однако, при попытке ввести неверные данные или каким-либо другим способом воздействовать на программу, отличным от “эталонного”, может возникнуть неприятный результат, вплоть до остановки программы.

Рисунок 43 – Обозначение NORMAL ошибок в интерфейсе Svacer

DYNAMIC_SIZE_MISMATCH.STRICT

Этот детектор указывает на несоответствие размеров данных с динамическими размерами ожидаемому размеру, но с относительно малой вероятностью возникновения.

DOUBLE_CLOSE

Происходит дублирующий вызов системного вызова close в одном и том же ресурсе.

DOUBLE_CLOSE.PROC

Это более конкретное предупреждение, которое генерируется, когда процесс закрывается дважды.

USE_AFTER_RELEASE

Указывает на ситуацию, когда ресурс (часто это память или системный ресурс) используется после того, как он был освобожден или удален.

HANDLE_LEAK.FRUGAL

Относится к ситуациям, когда дескриптор (например, файловый дескриптор или дескриптор сокета), полученный из системы, не освобождается в тех местах кода, где это возможно.

HANDLE_LEAK.CLOSEABLE

Указывает на утечку ресурса, в частности, на то что объект, который реализует интерфейс не был правильно закрыт.

HANDLE_LEAK.EXCEPTION

Генерируется, когда ресурс не освобождается в случае возникновения исключения. Если ресурс не освобождается при обработке исключений, это может привести к утечке этого ресурса при возникновении ошибки.

HANDLE_LEAK.FRUGAL.EXCEPTION

Обращает внимание на проблемы управления ресурсами, такие как утечки памяти, утечки файловых дескрипторов или другие типы ресурсов, которые не были корректно освобождены после использования. Такие утечки могут привести к неэффективному использованию ресурсов и повысить риск возникновения проблем с производительностью и надежностью программы.

Преффиск FRUGAL.EXCEPTION может указывать на усиленное внимание к обработке исключений с учетом экономии ресурсов. Это может подразумевать эффективное использование исключений для управления ошибками в коде, предотвращение излишних операций или повторной инициализации ресурсов при возникновении исключительных ситуаций.

HANDLE_LEAK.CLOSEABLE.EXCEPTION

Выявляет проблемы утечек ресурсов с интерфейсом Closeable и некорректной обработки исключений, связанных с закрытием ресурсов

DEREF_OF_NULL.UNCLEAR

Если вызываемая функция, в которой может происходить разыменование, неизвестна анализу, то потенциальные предупреждения выделяются в этот детектор

DEREF_OF_NULL.STRICT

Детектор указывает, что ошибка может возникать даже с относительно малой вероятностью.

DEREF_AFTER_NULL.RET

Детектор отслеживает ситуации проверки результата некоторых функций, возвращающих указатель, на нулевое значение.

DEREF_AFTER_NULL.UNCLEAR

Детектор соответствует передаче указателя в неизвестную функцию.

DEREF_AFTER_NULL.LOOP

Детектор указывает на ошибку, что связанную с попыткой разыменования (то есть обращения к значению, на которое указывает ссылка) нулевой ссылки внутри цикла

DEREF_OF_NULL.RET

Детектор предназначен для ситуаций проверки возвращаемого значения функций на ноль.

DEREF_OF_NULL.RET.ASSERT

Указывает на что ошибка возникает в специальной конструкции assert.

DEREF_OF_NULL.COND

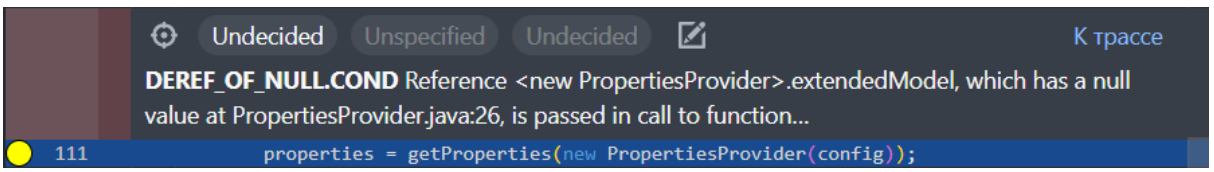


Рисунок 44 –Детектор DEREF_OF_NULL.COND в интерфейсе Svacer

Означает, что разыменуется указатель, который был сравнен с NULL, при некоторых условиях, которые в точности не могут быть оценены

анализатором (т.е. соответствующее значение атрибута отлично от истины и отражает только возможность).

DEREF_OF_NULL.ANNOT

Этот детектор указывает на разыменование нулевого указателя, который был аннотирован как @Nullable или аналогично. Это означает, что разработчик явно отметил, что этот указатель может быть null, но в коде есть место, где он разыменовывается без предварительной проверки.

DEREF_OF_NULL.ANNOT.STRICT

Этот детектор подобен DEREF_OF_NULL.ANNOT, но он используется в более строгом режиме. Он будет срабатывать на любое разыменование указателя, который был аннотирован как @Nullable или аналогично, даже если перед этим была проверка на null.

DEREF_OF_NULL.ANNOT.COND

Этот детектор срабатывает, когда указатель, который был аннотирован как @Nullable или аналогично, разыменовывается после условной проверки на null. Это может быть индикатором того, что условие проверки на null не полностью покрывает все возможные случаи.

DEREF_OF_NULL.RET.ANNOT

Детектор предназначен для ситуаций проверки возвращаемого значения функций на ноль. Он срабатывает, когда функция, которая может вернуть null (как указано в аннотации @Nullable или аналогичной), возвращает значение, которое затем разыменовывается без проверки на null.

TAINTED.INT_OVERFLOW.TRUNC

Детектор указывает на переполнение int из ненадежного источника. Trunk указывает, что некоторые данные усекаются или округляются, что приводит к потере точности или искажению информации.

INT_OVERFLOW

Обычно возникает, когда результат арифметической операции превышает максимальное или минимальное допустимое значение для типа int.

INT_OVERFLOW.AFTER_CHECK

В статическом анализаторе Svace, предупреждение OVERFLOW_AFTER_CHECK указывает на ситуацию, когда происходит переполнение (overflow) после проверки на переполнение. Это означает, что в коде есть проверка на переполнение (например, проверка, не превышает ли значение некоторого числа максимально допустимое значение), но после этой проверки все равно происходит операция, которая может вызвать переполнение.

INT_OVERFLOW.CONST

Детектор указывает на ситуацию, когда происходит переполнение целочисленного типа данных из-за использования константных значений.

INT_OVERFLOW.LOOP

Детектор указывает на ситуацию, когда происходит переполнение целочисленного типа данных в цикле.

INT_OVERFLOW.LOOP.STRICT

Детектор указывает на ситуацию, когда происходит переполнение целочисленного типа данных в цикле. "STRICT" в данном контексте указывает на то, что анализатор кода строго проверяет все возможные пути выполнения кода на предмет таких ошибок.

INT_OVERFLOW.LIB

Обычно возникает, когда результат арифметической операции превышает максимальное или минимальное допустимое значение для типа int, вызванное библиотечной функцией.

INT_OVERFLOW.SHIFT

Предупреждение указывает на ситуацию, когда происходит переполнение целого числа в результате операции сдвига.

INT_OVERFLOW.ZERO.WRAP

Указывает на потенциальную ошибку переполнения целых чисел, которая приводит к "заворачиванию" значения обратно к нулю или близкому к нему значению.

INT_OVERFLOW.COND

Указывает на ситуацию, когда переполнение целочисленного типа может произойти в условном выражении (например, в if-else структуре).

INT_OVERFLOW.TRUNC.COND

Указывает на ситуацию, когда происходит переполнение целочисленного типа данных из-за операции усечения, и это переполнение происходит в условном выражении.

INT_OVERFLOW.TRUNC.UNDER_BITMASK

Детектор указывает на потенциальную ошибку переполнения целых чисел, которая приводит к "обрезанию" значения под битовой маской.

INT_OVERFLOW.TRUNC.UNDER_BITMASK.LONG

Указывает на потенциальную ошибку, связанную с переполнением целочисленной переменной типа long.

UNCHECKED_FUNC_RES.USER

Предупреждение UNCHECKED_FUNC_RES.USER указывает на то, что результат функции не проверяется перед использованием. Это может привести к непредсказуемому поведению программы, если функция возвращает ошибку или неожиданное значение. Например, если функция должна возвращать указатель на объект, но происходит ошибка и возвращается NULL, а затем этот NULL используется без проверки - это приведет к ошибке.

UNCHECKED_FUNC_RES.USER.STRICT

Аналогично верхнему, но "STRICT" указывает на строгий уровень проверки.

UNCHECKED_FUNC_RES.STAT

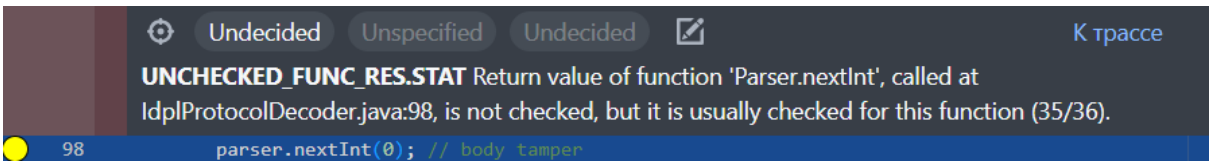


Рисунок 45 –Детектор UNCHECKED_FUNC_RES.STAT в интерфейсе Svacer

Детектор также предназначен для ситуаций, где функция возвращает значение, может указывать на ошибку, и это значение не проверяется и не присваивается локальной переменной. Однако он использует статистический подход. Если в заданном количестве случаев результат функции проверяется, то для остальных случаев детектором также будет предложено проверить результат.

NO_BASE_CALL.STAT

При переопределении метода не вызван метод базового класса, который должен быть вызван (либо потому, что остальные переопределения его вызывают, либо согласно документации). Но в этом детекторе используется статистический подход. Если при заранее указанной доле случаев процент указан.

NO_BASE_CALL.LIB

При переопределении метода не вызван метод базового класса, который должен быть вызван (либо потому, что остальные переопределения его вызывают, либо согласно документации). Но детектор с подкатегорией LIB проверяет библиотечные функции.

PASS_TO_PROC_AFTER_CHECK

Когда проверка значения происходит, но после этой проверки значение передается в процедуру или функцию без дополнительной проверки.

Это может быть потенциально опасно, если функция или процедура не предполагает получение некорректных или неожиданных значений и не выполняет свои собственные проверки. Это может привести к ошибкам выполнения, сбоям программы или уязвимостям безопасности.

UNREACHABLE_CODE

Детектор занимается поиском недостижимого кода, то есть такого кода, которого не достигнет ни один путь выполнения.

UNREACHABLE_CODE.ENUM

Детектор указывает на ситуацию, когда определенный код оказывается недостижимым из-за использования перечисления (enum).

Это может произойти, например, когда в switch-case конструкции обрабатываются все возможные значения перечисления, и есть дополнительный блок case или default, который никогда не будет выполнен.

UNREACHABLE_CODE.EXCEPTION

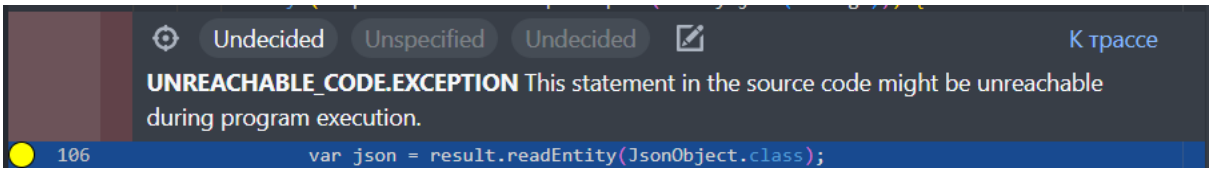


Рисунок 46 –Детектор UNREACHABLE_CODE.EXCEPTION в интерфейсе Svac

Детектор UNREACHABLE_CODE.EXCEPTION указывает на обнаружение кода, который никогда не будет выполнен из-за того, что перед этим кодом выбрасывается исключение.

Это может быть результатом ошибки программирования или неправильного понимания потока управления. Например, если после блока try-catch идет код, но в блоке try гарантированно выбрасывается исключение, то код после блока try-catch никогда не будет выполнен.

UNREACHABLE_CODE.RET

Детектор указывает на то, что существует некоторый код, который никогда не будет выполнен из-за того, что перед ним есть инструкция возврата (return).

UNREACHABLE_CODE.GLOBAL

Детектор указывает на то, что существует некоторый код, который никогда не будет выполнен из-за ошибки с глобальной переменной.

REDUNDANT_COMPARISON

Результат определенных сравнений иногда можно вывести из их контекста и результатов других сравнений. Это может указывать на ошибочную логику и может привести к неработающему коду или бесконечным циклам, если, например, условие цикла никогда не меняет своего значения.

REDUNDANT_COMPARISON.ALWAYS_FALSE

Означает то, что в коде присутствует сравнение, которое всегда будет ложным независимо от значений переменных.

UNREACHABLE_CODE.SWITCH

Означает то, что в блоке кода switch-case есть код, который никогда не будет выполнен или достигнут. Это может произойти по различным причинам. Например, если у вас есть case без соответствующего break, и следующий case никогда не будет выполнен. Или это может быть связано с логическими ошибками в условиях case.

NO_CHECK_IN_LOCK

Детектор указывает на ситуацию, когда в коде программы отсутствует проверка успешности операции блокировки (lock) перед её использованием.

LOCK_INCONSISTENT

Детектор LOCK_INCONSISTENT обычно указывает на потенциальное нарушение согласованности использования блокировок. Это может означать, что в коде присутствуют проблемы с управлением блокировками, что может привести к конкуренции за доступ к данным, взаимоблокировкам или другим проблемам с многопоточностью.

PROC_USE.HASH_FUNC

Детектор PROC_USE.HASH_FUNC предназначен для обнаружения проблем в использовании хеш-функций в коде. Этот детектор фокусируется на случаях, когда используются хеш-функции, которые могут быть неадекватными для конкретного применения, например, с точки зрения безопасности или производительности.

NO_CATCH

Предназначен для обнаружения ситуаций в коде, когда исключения, которые могут быть выброшены, не обрабатываются должным образом. Часто это указывает на потенциальные проблемы с надежностью выполнения программы, так как необработанные исключения могут привести к неожиданному прерыванию программы.

NO_CATCH.LIBRARY

Фокусируется на идентификации ситуаций, когда возможные исключения, выбрасываемые библиотечными функциями, не обрабатываются в пользовательском коде.

SHADOWED_NAME

Предназначен для выявления ситуаций, когда локальная переменная или параметр метода "затеняет" переменную в более широкой области

видимости. Это может привести к ошибкам в коде из-за того, что разработчик может случайно использовать одно и то же имя для разных переменных в различных контекстах.

Что подразумевает "затенение" переменной:

Затенение переменной происходит, когда две переменные в одной области действия имеют одинаковое имя, но разные области видимости. Наиболее частый случай - это когда локальная переменная в методе имеет то же имя, что и переменная класса (поле). Это может вызвать путаницу, так как в зависимости от контекста использования имя переменной будет ссылаться на разные объекты.

UNREACHABLE_CODE.CATCH

Предназначен для нахождения ситуаций, когда код внутри блока catch не может быть достигнут. Это часто указывает на логические ошибки в структуре управления исключениями или на неправильную организацию блоков try-catch.

FB.JUA_DONT_ASSERT_INSTANCEOF_IN_TESTS

Утверждать проверки типов в тестах не рекомендуется, поскольку сообщение об исключении приведения класса может лучше указать причину использования экземпляра неправильного класса, чем утверждение экземпляра.

При отладке тестов, которые завершаются неудачей из-за неправильного приведения, может быть более полезно наблюдать за выходными данными результирующего исключения ClassCastException, которое может предоставить информацию о фактическом обнаруженном типе. Вместо этого утверждение типа перед приведением приведет к менее информативному сообщению «false is not true».

Если JUnit используется с hamcrest, вместо него можно использовать класс IsInstanceOf из hamcrest.

FB.OVERRIDING_METHODS_MUST_INVOKE_SUPER

Детектор определяет ошибку, в которой суперметод помечен @OverridingMethodsMustInvokeSuper, но переопределяющий метод не вызывает суперметод.

FB.MS_EXPOSE_BUF

Детектор определяет ошибку, в которой публичный статический метод либо возвращает буфер (java.nio.*Buffer), который обертывает массив, являющийся частью статического состояния класса, сохраняя ссылку только на этот же массив, либо возвращает неглубокую копию буфера, который является частью статической статистики класса, которая разделяет свою ссылку с исходным буфером. Любой код, вызывающий этот метод, может свободно изменять базовый массив. Одним из исправлений является возврат буфера только для чтения или нового буфера с копией массива.

FB.EI_EXPOSE_BUF

Возврат ссылки на буфер (java.nio.*Buffer), который оборачивает массив, хранящийся в одном из полей объекта, раскрывает внутреннее представление элементов массива, поскольку буфер хранит только ссылку на массив, а не копирует его содержимое. Аналогичным образом, возврат поверхностной копии такого буфера (с использованием его метода Dupe()), хранящейся в одном из полей объекта, также раскрывает внутреннее представление буфера. Если доступ к экземплярам осуществляется с помощью ненадежного кода, а непроверенные изменения в массиве могут поставить под угрозу безопасность или другие важные свойства, вам придется сделать что-то другое. Во многих ситуациях лучшим подходом является возврат буфера, доступного только для чтения (с использованием метода asReadOnly()) или копирование массива в новый буфер (с использованием метода put()).

FB.EI_EXPOSE_BUF2

Детектор определяет ошибку, в которой код создает буфер, который сохраняет ссылку на внешний массив или массив внешнего буфера во внутреннем представлении объекта. Если доступ к экземплярам осуществляется с помощью ненадежного кода, а непроверенные изменения в массиве могут поставить под угрозу безопасность или другие важные свойства, вам придется сделать что-то другое. Во многих ситуациях лучше хранить копию массива.

FB.EI_EXPOSE_STATIC_BUF2

Детектор определяет уязвимость, в которой код создает буфер, который сохраняет ссылку на внешний массив или массив внешнего буфера в статическое поле. Если непроверенные изменения в массиве могут поставить под угрозу безопасность или другие важные свойства, вам придется сделать что-то другое. Во многих ситуациях лучше хранить копию массива.

FB.TLW_TWO_LOCK_NOTIFY

Детектор определяет ошибку, в которой код вызывает notify() или notifyAll(), пока две блокировки проводятся. Если это уведомление предназначено для пробуждения метода wait(), который удерживает те же блокировки, он может заблокироваться, поскольку ожидание откажется только от одной блокировки, и уведомление не сможет получить обе блокировки, и, таким образом, уведомление не будет успешным.

FB.LI_LAZY_INIT_INSTANCE

Детектор определяет ошибку, в которой метод содержит несинхронизированную ленивую инициализацию энергонезависимого поля. Поскольку компилятор или процессор могут изменить порядок инструкций, потоки не гарантированно увидят полностью инициализированный объект, если метод может быть вызван несколькими потоками. Чтобы исправить проблему, вы можете сделать поле изменяемым.

FB.BRSA_BAD_RESULTSET_ACCESS

Был выполнен вызов методов getXXX или updateXXX набора результатов, где индекс поля равен 0. Поскольку поля ResultSet начинаются с индекса 1, это всегда ошибка. Это правило устарело

FB.DCN_NULLPOINTER_EXCEPTION

Согласно правилу SEI Cert, ERR08-J NullPointerException не должен перехватываться. Обработка NullPointerException считается худшей альтернативой проверке нуля.

Этот несоответствующий код перехватывает исключение NullPointerException, чтобы проверить, имеет ли входящий параметр значение NULL:

```
boolean hasSpace(String m) {
  try {
    String ms[] = m.split(" ");
    return names.length != 1;
  } catch (NullPointerException e) {
    return false;
  }
}
```

Соответствующее решение будет использовать нулевую проверку, как показано в следующем примере:

```
boolean hasSpace(String m) {
```

```
    if (m == null) return false;

    String ms[] = m.split(" ");

    return names.length != 1;

}
```

FB.BC_NULL_INSTANCEOF

Детектор указывает на возможную ошибку в работе кода. Этот тест экземпляра всегда будет возвращать значение false, поскольку проверяемое значение гарантированно будет нулевым. Хотя это безопасно, убедитесь, что это не является признаком какого-либо недоразумения или какой-либо другой логической ошибки.

FB.FB_UNEXPECTED_WARNING

Детектор выдал предупреждение, которое, согласно аннотации @NoWarning, является неожиданным или нежелательным.

FB.FB_MISSING_EXPECTED_WARNING

Детектор выдал уведомление, так как код не сгенерировал предупреждение, которое, согласно аннотации @ExpectedWarning, является ожидаемым или желательным.

FB.EOS_BAD_END_OF_STREAM_CHECK

Метод java.io.FileInputStream.read() возвращает целое число. Если это целое число преобразуется в байт, то -1 (что указывает на EOF) и байт 0xFF становятся неразличимыми, это сравнение (преобразованного) результата с -1 приводит к преждевременному завершению чтения (вероятно, в цикле), если символ 0xFF встречается. Аналогично, метод java.io.FileReader.read() также

возвращает целое число. Если он преобразуется в символ, то -1 становится 0xFFFF, что соответствует символу MAX_VALUE. Сравнивать результат с -1 бессмысленно, поскольку в Java символы не имеют знака. Если проверка на EOF является условием цикла, то этот цикл бесконечен.

FB.REFLC_REFLECTION_MAY_INCREASE_ACCESSIBILITY_OF_CLASS

Детектор уведомляет об обхождении правила, которое запрещает использование отражения для повышения доступности классов, методов или полей. Если класс в пакете предоставляет общедоступный метод, который принимает экземпляр java.lang.Class в качестве параметра и вызывает его метод newInstance(), то это увеличивает доступность классов в том же пакете без общедоступных конструкторов. Код злоумышленника может вызвать этот метод и передать такой класс, чтобы создать его экземпляр. Этого следует избегать, либо сделав метод закрытым, либо проверив разрешение на доступ к пакету. Третья возможность — использовать метод java.beans.Beans.instantiate() вместо метода java.lang.Class.newInstance(), который проверяет, имеет ли получаемый объект класса какие-либо общедоступные конструкторы.

FB.REFLF_REFLECTION_MAY_INCREASE_ACCESSIBILITY_OF_FIELD

Детектор уведомляет об обхождении правила, которое запрещает использование отражения для повышения доступности классов, методов или полей. Если класс в пакете предоставляет общедоступный метод, который принимает экземпляр java.lang.reflect.Field в качестве параметра и вызывает метод установки (или setAccessible()), то это увеличивает доступность полей

в том же пакете, которые являются частными. защищенный или пакет частный. Код злоумышленника может вызвать этот метод и передать такое поле, чтобы изменить его. Этого следует избегать, либо сделав метод закрытым, либо проверив разрешение на доступ к пакету.

FB.MC_OVERRIDABLE_METHOD_CALL_IN_CONSTRUCTOR

Детектор уведомляет о вызове переопределяемого метода в конструкторе может привести к использованию неинициализированных данных. Также может произойти утечка ссылки this частично созданного объекта. Из конструктора следует вызывать только статические, финальные или частные методы.

FB.MC_OVERRIDABLE_METHOD_CALL_IN_CLONE

Детектор уведомляет о вызове переопределяемых методов из метода clone() небезопасен, поскольку подкласс может переопределить метод, что повлияет на поведение clone(). Он также может наблюдать или изменять объект-клон в частично инициализированном состоянии. Из метода clone() следует вызывать только статические, финальные или частные методы.

FB.SSD_DO_NOT_USE_INSTANCE_LOCK_ON_SHARED_STATIC_DATA

Если блокировка или синхронизированный метод не являются статическими, это изменяет статическое поле и может оставить общие статические данные незащищенными от одновременного доступа. Это может произойти двумя способами: если метод синхронизации использует нестатический объект блокировки или синхронизированный метод объявлен как нестатический. Оба пути неэффективны. Лучшее решение — использовать частный статический объект окончательной блокировки для защиты общих статических данных.

FB.FL_FLOATS_AS_LOOP_COUNTERS

Переменные с плавающей запятой не следует использовать в качестве счетчиков циклов, поскольку они неточны, что может привести к неправильным циклам. Счетчик цикла — это переменная, которая изменяется на каждой итерации и определяет, когда цикл должен завершиться. Оно уменьшается или увеличивается на фиксированную величину на каждой итерации.

FB.THROWS_METHOD_THROWS_RUNTIMEEXCEPTION

Метод намеренно генерирует RuntimeException. Согласно правилу SEI CERT ERR07-J, выдача RuntimeException может вызвать ошибки, например, вызывающая сторона не сможет проверить исключение и, следовательно, не сможет должным образом восстановиться после него. Более того, выдача RuntimeException заставит вызывающую сторону перехватить RuntimeException и, следовательно, нарушит правило SEI CERT ERR08-J. Обратите внимание, что вы можете получить производное от Exception или RuntimeException и можете создать новый экземпляр этого исключения.

FB.THROWS_METHOD_THROWS_CLAUSE_BASIC_EXCEPTION

Метод перечисляет Exception в своем предложении throws.

При объявлении метода типы исключений в предложении throws должны быть максимально конкретными. Таким образом, использование Exception в предложении throws заставит вызывающую сторону либо использовать его в своем собственном предложении throws, либо использовать его в блоке try-catch (когда он не обязательно содержит какую-либо значимую информацию о выброшенном исключении).

FB.THROWS_METHOD_THROWS_CLAUSE_THROWABLE

Метод перечисляет Throwable в своем предложении throws.

При объявлении метода типы исключений в предложении throws должны быть максимально конкретными. Таким образом, использование Throwable в предложении throws заставит вызывающую сторону либо использовать его в своем собственном предложении throws, либо использовать его в блоке try-catch (когда он не обязательно содержит какую-либо значимую информацию о выброшенном исключении).

Более того, такое использование Throwable является семантически плохой практикой, учитывая, что Throwables также включает в себя ошибки, но по определению они возникают в неисправимых сценариях.

FB.PERM_SUPER_NOT_CALLED_IN_GETPERMISSIONS

Детектор уведомляет об обхождении правила, которое требует, чтобы пользовательские загрузчики классов всегда вызывали метод getPermissions() своего суперкласса в своем собственном методе getPermissions() для инициализации объекта, который они возвращают в конце. Отсутствие этого параметра означает, что класс, определенный с помощью этого специального загрузчика классов, имеет разрешения, полностью независимые от разрешений, указанных в общесистемном файле политики. По сути, разрешения класса переопределяют их.

FB.USC_POTENTIAL_SECURITY_CHECK_BASED_ON_UNTRUSTED_SOURCE

Открытый метод открытого класса может быть вызван извне пакета, что означает, что ему могут быть переданы ненадежные данные. Вызов метода до doPrivileged для проверки его возвращаемого значения, а затем вызов того же метода внутри класса опасен, если метод или включающий его класс не является окончательным. Злоумышленник может передать экземпляр

злонамеренного потомка класса вместо экземпляра ожидаемого, где этот метод переопределен таким образом, что он возвращает разные значения при разных вызовах. Например, метод, возвращающий путь к файлу, может возвращать безопасный путь для проверки перед входом в блок doPrivileged, а затем конфиденциальный файл при вызове внутри блока doPrivileged. Чтобы избежать такого сценария, защитно скопируйте объект, полученный в параметре, например. с помощью конструктора копирования класса, используемого в качестве типа формального параметра. Это гарантирует, что метод ведет себя точно так, как ожидалось.

С ключевым словом **DEREF** (разыменовыванием указателя)

DEREF_OF_NULL.COND

Ошибка "Reference ..., which has a null value, is passed in call to function at ..., where it may be dereferenced" указывает на то, что в вашем коде имеется передача нулевого (null) значения в функцию, где это значение может быть использовано (разыменовано), что может привести к ошибке выполнения, такой как NullPointerException в Java.

DEREF_OF_NULL.RET

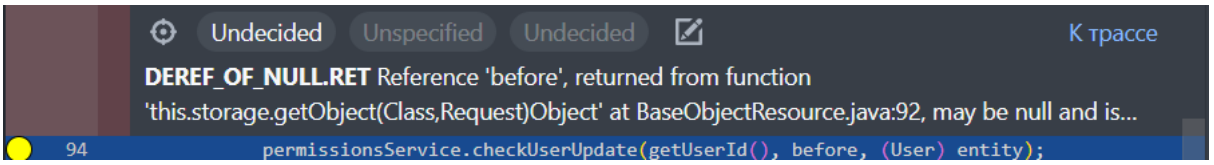


Рисунок 47 –Детектор DEREF_OF_NULL.RET в интерфейсе Svacer

Ошибка "Reference 'before', returned from function, may be null and is passed as 2nd parameter in call to function, where it is dereferenced" указывает на то, что в вашем коде есть ситуация, где возвращаемое значение из функции может быть равно null, и это значение передается в качестве второго параметра в другую функцию, где оно разыменовывается (используется) без предварительной проверки на null. Это может привести к ошибке выполнения, такой как NullPointerException в Java.

HANDLE_LEAK.EX.EXCEPTION

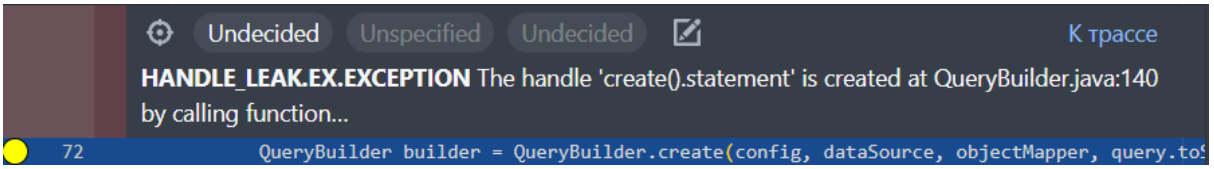


Рисунок 48 –Детектор HANDLE_LEAK.EX.EXCEPTION в интерфейсе Svacer

Ошибка "The handle 'create().statement' is created by calling function and lost" указывает на то, что в вашем коде создается ресурс (handle), например, вызывается функция create(), которая возвращает какой-то ресурс, и этот ресурс затем теряется, то есть на него нет ссылки, и нет возможности освободить ресурс или продолжить использование его в коде.

TERNARY_OPERATOR_PRECEDENCE

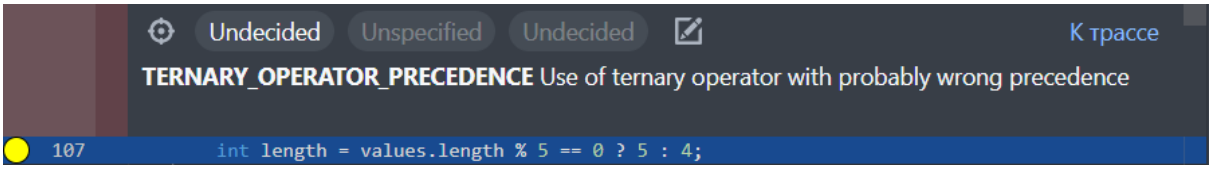


Рисунок 49 –Детектор TERNARY_OPERATOR_PRECEDENCE в интерфейсе Svacer

Ошибка "Use of ternary operator with probably wrong precedence" указывает на то, что в вашем коде использован тернарный оператор (? :), и возможно, порядок выполнения операторов не соответствует вашим намерениям из-за неправильного приоритета операторов.

7.1.4. MINOR ошибки

Ошибки класса “Minor” (Рисунок 49) - ошибки, которые вряд ли приведут к какому-либо негативному результату, однако при наличии ресурсов их все же стоит устранить.

Рисунок 50 – Обозначение MINOR ошибок в интерфейсе Svacer

OVERFLOW_AFTER_CHECK.VAR

Указывает на ситуации, когда происходит проверка на переполнение (overflow), и затем значение переменной изменяется без дополнительной проверки, находясь в потенциально опасном состоянии.

PASSED_TO_PROC_AFTER_RELEASE

Указывает на ситуации, когда указатель на объект или ресурс передается в функцию (процедуру) после его освобождения или удаления.

Это может привести к серьезным ошибкам в программах, так как использование освобожденных ресурсов или объектов может вызвать неопределенное поведение программы, включая сбои в работе и уязвимости в безопасности.

HANDLE_LEAK.STRICT

Указывает на ситуации, когда произошел утечка ресурсов из-за неправильного управления ими в программе. Утечка ресурсов может возникать, когда программа не освобождает или не закрывает ресурсы после использования, что может привести к нежелательным последствиям, таким как истощение памяти или других ресурсов системы.

Детектор HANDLE_LEAK.STRICT выявляет такие утечки ресурсов, когда ресурсы были выделены, но ими не было корректно управлено, например, не было вызвано закрытие или освобождение ресурсов после завершения работы с ними. Данный детектор работает в строгом режиме, что подразумевает более тщательное обнаружение утечек ресурсов в коде

HANDLE_LEAK.EXCEPTION.STRICT

Указывает на ситуации, когда происходит утечка ресурсов из-за неправильного управления ими в блоках кода, содержащих обработку исключений. Утечка ресурсов в этом контексте может быть особенно опасной, так как возможность утечки ресурсов может усугубиться в случае генерации исключения в процессе работы с ресурсами.

Детектор HANDLE_LEAK.EXCEPTION.STRICT работает в строгом режиме, что означает, что он более детально анализирует блоки кода, которые обрабатывают исключения, и ищет ситуации, когда ресурсы не освобождаются или не закрываются правильно в случае возникновения исключительной ситуации. Такие ситуации могут привести к утечке ресурсов, что может отрицательно сказаться на производительности и стабильности программы.

NULL_AFTER_DEREF.RET

Следит за ситуациями, когда указатель разыменовывается (dereferencing), а затем проверяется на равенство нулю. Это может указывать на неправильное использование указателей в коде, когда программа сначала разыменовывает указатель, а затем сравнивает это значение с нулевым указателем.

UNCHECKED_FUNC_RES.LIB.STRICT

Обозначает ситуации, когда результат вызова функции не проверяется на предмет ошибок или исключений, возможно, в рамках работы с внешней библиотекой (lib - сокр. library) или сторонними функциями.

В строгом (STRICT) режиме детектор анализирует такие случаи безопасности и надёжности программы, где возможна ошибка возвращаемого результата или исключения, но при этом не производится соответствующая обработка и проверка этого результата. Необработанный результат функции или исключения может привести к непредсказуемому поведению программы, нестабильности работы, ошибкам или уязвимостям.

UNCHECKED_FUNC_RES.LIB.MINOR

Указывает на ситуации, когда результат функции, вызываемой из внешней библиотеки (lib), не проверяется на наличие ошибок или исключений, но это не является критической проблемой.

В режиме MINOR (незначительная проблема) детектор обращает внимание на такие случаи, которые, хотя и могут потенциально привести к нежелательным последствиям, не являются критическими или сильно влияющими на работоспособность программы. Тем не менее, необработанный результат функции или исключения может создать потенциальные проблемы в коде, такие как недостаточная обратная связь по выполнению операции или уязвимости.

UNUSED_FUNC_RES

Берет на себя задачу выявления ситуаций, когда результат вызова функции не используется в коде. Это может быть признаком неэффективного или ошибочного использования функций в программе, когда результат операции или функции получается, но не используется нигде дальше в коде.

UNUSED_FUNC_RES.MINOR

Обнаруживает ситуации, когда результат вызова функции не используется в коде, но это не является критической проблемой. В режиме MINOR (незначительная проблема) детектор указывает на места, где можно улучшить чистоту и эффективность кода, хотя обнаруженные проблемы могут не оказывать серьезного влияния на работу программы.

UNUSED_FUNC_RES.SAME_LINE

Указывает на ситуации, когда результат вызова функции возвращается в той же строке кода, где он был вызван, но не используется дальше в этой же строке или в ближайшем контексте. Неиспользование результата вызова функции в той же строке, где он был вызван, может сигнализировать о лишнем коде, который не влияет на дальнейшее выполнение программы и может быть оптимизирован или удален.

UNUSED_FUNC_RES.REWRITE.MINOR

Обозначает незначительные случаи, когда результат вызова функции не используется в коде, но предлагает возможность оптимизации за счет переопределения кода. Этот тип детектора фокусируется на предложениях по улучшению кода в местах, где результат функции не используется, но его использование может улучшить читаемость или эффективность кода.

UNREACHABLE_CODE.TERMINATION

Указывает на ситуации, когда код содержит участки, которые никогда не будут достигнуты из-за операторов завершения, таких как операторы "return", "throw" или "break". В таких случаях код после операторов завершения становится недостижимым, поскольку выполнение программы завершается в этих местах.

INVARIANT_RESULT.EX

Этот детектор указывает на наличие выражения, независимо от входных параметров принимающего одно и то же значение (инварианта)

REDUNDANT_NULL_CHECK

Этот детектор указывает на наличие избыточной проверки значения на NULL

REDUNDANT_COMPARISON.RET

Этот детектор указывает на наличие избыточного сравнения возвращаемого функцией значения

REDUNDANT_COMPARISON.OVERLAP

Этот детектор указывает на наличие избыточного сравнения значений, которое не нужно из-за совпадения области памяти у этих значений

REDUNDANT_COMPARISON.GLOBAL

Этот детектор указывает на наличие избыточного сравнения значений глобальных переменных

TEST.UNUSED_FUNC_RES

Этот детектор указывает на неиспользуемые результаты выполнения функций

NO_UNLOCK.STRICT

Этот детектор указывает на то, что отсутствует разблокировка мьютексов

UNUSED_VALUE

Этот детектор указывает на возникновение значений, которые в дальнейшем никак в работе программы не участвуют

UNUSED_VALUE.STRICT

Этот детектор указывает на возникновение значений, которые в дальнейшем никак в работе программы потенциально могут не участвовать

UNUSED_PARAM

Этот детектор указывает на наличие в функции входных параметров, которые никак в работе функции не участвуют

UNUSED_VALUE.PARAM_ASSIGN

Этот детектор указывает на наличие у функции параметров, изначальные значения которых еще до использования инициализируются другими значениями внутри этой функции

UNUSED_VALUE.PARAM_ASSIGN.NULL

Этот детектор указывает на наличие у функции параметров, изначальные значения которых зануляются внутри этой функции

BAD_KEYWORD

Этот детектор указывает на некорректное использование зарезервированных языком слов (например, назначение переменной имени if)

DIVISION_BY_ZERO.EX.FLOAT

Этот детектор указывает на наличие деления на 0 float - значения

INFO.PARSED_BUFFER

Предоставляет информацию о разборе или обработке буфера данных в рамках анализа кода. Этот вид детектора может быть полезен для отслеживания процесса работы с буферами данных в программе, анализа их содержимого и обработки в соответствии с требованиями безопасности и эффективности.

FB.PREDICTABLE_RANDOM

Данный детектор указывает на предсказуемость случайного выбора в коде.

FB.SERVLET_PARAMETER

Сервлет может читать параметры GET и POST различными методами. Полученное значение следует считать небезопасным. Возможно, вам придется проверить или очистить эти значения, прежде чем передавать их конфиденциальным API.

FB.SERVLET_CONTENT_TYPE

Тип содержимого HTTP-заголовка может контролироваться клиентом. Таким образом, его значение не следует использовать в каких-либо критических для безопасности решениях.

FB.SERVLET_SERVER_NAME

Заголовок имени хоста может контролироваться клиентом. Таким образом, его значение не следует использовать в каких-либо критических для безопасности решениях. И ServletRequest.getServerName(), и HttpServletRequest.getHeader("Host") имеют одинаковое поведение: извлекают заголовок Host. Веб-контейнер, обслуживающий ваше приложение, по умолчанию может перенаправлять запросы в ваше приложение. Это позволит злоумышленнику разместить любое значение в заголовке Host. Рекомендуется не доверять этому значению при принятии каких-либо решений по обеспечению безопасности, принимаемых в отношении запроса.

FB.SERVLET_SESSION_ID

Метод HttpServletRequest.getRequestedSessionId() обычно возвращает значение JSESSIONID файла cookie. Обычно это значение доступно только логике управления сеансом, а не обычному коду разработчика. Таким образом, JSESSIONID следует использовать только для того, чтобы проверить, соответствует ли его значение существующему идентификатору

сеанса. Если это не так, пользователя следует считать неаутентифицированным пользователем. Кроме того, значение идентификатора сеанса никогда не должно регистрироваться. Если это так, то файл журнала может содержать действительные идентификаторы активных сеансов, что позволяет инсайдеру перехватывать любые сеансы, идентификаторы которых были зарегистрированы и все еще активны.

FB.SERVLET_QUERY_STRING

Строка запроса представляет собой объединение имен и значений параметров GET. Могут быть переданы и другие параметры, отличные от предполагаемых. Как и в случае отдельных значений параметров, полученных с помощью таких методов, как HttpServletRequest.getParameter(), значение, полученное с помощью HttpServletRequest.getQueryString(), следует считать небезопасным. Возможно, вам придется проверить или очистить все, что получено из строки запроса, прежде чем передавать его конфиденциальным API.

FB.SERVLET_HEADER

Заголовки запросов могут быть легко изменены запрашивающим пользователем. В общем, не следует предполагать, что запрос поступил из обычного браузера без изменений злоумышленником. Таким образом, рекомендуется не доверять этому значению при принятии любых решений по безопасности, которые вы принимаете в отношении запроса.

FB.SERVLET_HEADER_REFERER

Детектор указывает на то, что или этому заголовку можно присвоить любое значение, если запрос исходит от злоумышленника, или «Referer» не будет присутствовать, если запрос был инициирован из другого безопасного источника (HTTPS). Рекомендуется сделать так, чтобы никакое управление доступом не основывалось на значении этого заголовка. Никакая защита CSRF не должна основываться только на этом значении.

FB.SERVLET_HEADER_USER_AGENT

Заголовок «User-Agent» может быть легко подделан клиентом. Принятие различных вариантов поведения на основе User-Agent (для UA-сканера) не рекомендуется.

FB.COOKIE_USAGE

Информация, хранящаяся в пользовательском файле cookie, не должна быть конфиденциальной или связана с сеансом. В большинстве случаев конфиденциальные данные должны храниться только в сеансе и на них ссылаются файлы cookie сеанса пользователя. См. HttpSession (HttpServletRequest.getSession()). Пользовательские файлы cookie могут

использоваться для информации, которая должна храниться дольше, чем конкретный сеанс, и не зависит от него.

FB.COMMAND_INJECTION

Выделенный API используется для выполнения системной команды. Если в этот API передаются нефильтрованные входные данные, это может привести к выполнению произвольной команды.

FB.WEAK_FILENAMEUTILS

Некоторые методы FilenameUtils не фильтруют NULL-байты (0x00). Если в имя файла вводится нулевой байт, если это имя файла передается в базовую ОС, полученный файл будет именем файла, указанным до нулевого байта, поскольку на уровне ОС все строки завершаются нулевой байт, хотя сама Java не заботится о нулевых байтах и не относится к ним по-особенному. Такое поведение ОС можно использовать для обхода проверки имени файла, которая проверяет конец имени файла (например, заканчивается на «.log»), чтобы убедиться, что доступ к файлу безопасен. Чтобы это исправить, рекомендуется сделать две вещи: обновите Java 7, обновление 40 или более поздней версии, или Java 8+, поскольку в этих версиях исправлено внесение нулевых байтов в имена файлов. Строго проверяйте любые имена файлов, предоставленные ненадежными пользователями, чтобы убедиться, что они действительны (т. е. не содержат нулевых значений, не содержат символов пути и т. д.). Если вы знаете, что используете современную версию Java, невосприимчивую к внедрению NULL-байтов, вы, вероятно, можете отключить это правило.

FB.WEAK_TRUST_MANAGER

Пустые реализации TrustManager часто используются для простого подключения к хосту, который не подписан корневым центром сертификации. Как следствие, это уязвимо для атак типа «человек посередине», поскольку клиент будет доверять любому сертификату. Необходимо создать TrustManager, позволяющий использовать определенные сертификаты (например, на основе TrustStore).

FB.JAXWS_ENDPOINT

Этот метод является частью веб-службы SOAP (JSR224). Необходимо проанализировать безопасность этого веб-сервиса. Например: аутентификация, если она применяется, должна быть проверена, контроль доступа, если он применяется, должен быть проверен, входные данные следует отслеживать на предмет потенциальных уязвимостей. В идеале связь должна осуществляться через SSL.

FB.JAXRS_ENDPOINT

Этот метод является частью веб-службы REST (JSR311). Необходимо проанализировать безопасность этого веб-сервиса. Например: аутентификация, если она применяется, должна быть проверена, контроль доступа, если он применяется, должен быть проверен, входные данные следует отслеживать на предмет потенциальных уязвимостей. В идеале связь должна осуществляться через SSL. Если служба поддерживает запись (например, через POST), следует изучить ее уязвимость к CSRF.

FB.TAPESTRY_ENDPOINT

Конечная точка Tapestry была обнаружена при запуске приложения. Каждая страница Tapestry в этом приложении должна быть исследована, чтобы убедиться, что все входные данные, автоматически отображаемые таким образом, правильно проверены перед их использованием.

FB.WICKET_ENDPOINT

Этот класс представляет веб-страницу Wicket. Ввод автоматически считывается из экземпляра PageParameters, передаваемого конструктору. Текущая страница сопоставляется с представлением /package/WebPageName.html. Каждая страница Wicket в этом приложении должна быть исследована, чтобы убедиться, что все входные данные, которые автоматически сопоставляются таким образом, правильно проверены перед их использованием.

FB.WEAK_MESSAGE_DIGEST

Алгоритмы MD2, MD4 и MD5 не рекомендуются для MessageDigest. PBKDF2 следует использовать, например, для хэширования пароля.

FB.CUSTOM_MESSAGE_DIGEST

Реализация пользовательского MessageDigest подвержена ошибкам. NIST рекомендует использовать SHA-224, SHA-256, SHA-384, SHA-512, SHA-512/224 или SHA-512/256.

FB.FILE_UPLOAD_FILENAME

Имя файла, предоставленное API FileUpload, может быть изменено клиентом для ссылки на неавторизованные файлы. Поэтому такие значения не следует передавать напрямую в API файловой системы. Если это приемлемо, приложение должно генерировать собственные имена файлов и использовать их. В противном случае предоставленное имя файла должно быть правильно проверено, чтобы убедиться, что оно правильно структурировано, не содержит несанкционированных символов пути (например, / \) и относится к авторизованному файлу.

FB.REDOS

Регулярные выражения (Regex) часто подвергаются атакам типа «отказ в обслуживании» (DOS) (называемым ReDOS). Это связано с тем, что механизмам регулярных выражений может потребоваться много времени при анализе определенных строк, в зависимости от того, как определено регулярное выражение. Поэтому вполне возможно, что один запрос может вызвать большой объем вычислений на стороне сервера. Проблема с этим регулярным выражением и другими подобными ему заключается в том, что существует два разных способа, которыми один и тот же входной символ может быть принят регулярным выражением из-за + (или *) внутри скобок и + (или *) снаружи скобка. В том виде, в каком это написано, любой + может содержать символ «a». Чтобы исправить это, регулярное выражение следует переписать, чтобы устранить двусмысленность. Например, это можно было бы просто переписать как: ^a+$, что, по-видимому, и имел в виду автор (любое количество a). Если предположить, что исходное регулярное выражение означало именно это, то новое регулярное выражение можно быстро оценить и оно не подлежит ReDOS.

FB.XXE_SAXPARSER

Атаки внешних объектов XML (XXE) могут возникать, когда анализатор XML поддерживает сущности XML при обработке XML, полученного из ненадежного источника.

Этот параметр защитит вас от атак типа «отказ в обслуживании» и удаленного доступа к файлам.

```
SAXParserFactory spf = SAXParserFactory.newInstance();
spf.setFeature(XMLConstants.FEATURE_SECURE_PROCESSING, true);
SAXParser parser = spf.newSAXParser();
parser.parse(inputStream, customHandler);
```

Отключив DTD, можно предотвратить почти все атаки XXE.

```
SAXParserFactory spf = SAXParserFactory.newInstance();
spf.setFeature("http://apache.org/xml/features/disallow-doctype-decl", true);
SAXParser parser = spf.newSAXParser();
parser.parse(inputStream, customHandler);
```

FB.XXE_XMLREADER

Атаки внешних объектов XML (XXE) могут возникать, когда анализатор XML поддерживает сущности XML при обработке XML, полученного из ненадежного источника.

Этот параметр защитит вас от атак типа «отказ в обслуживании» и удаленного доступа к файлам.

```
XMLReader reader = XMLReaderFactory.createXMLReader();
reader.setFeature(XMLConstants.FEATURE_SECURE_PROCESSING,
true);
reader.setContentHandler(customHandler);
reader.parse(new InputSource(inputStream));
```

Отключив DTD, можно предотвратить почти все атаки XXE.

```
XMLReader reader = XMLReaderFactory.createXMLReader();
reader.setFeature("http://apache.org/xml/features/disallow-doctype-decl",
true);
reader.setContentHandler(customHandler);
reader.parse(new InputSource(inputStream));
```

FB.XXE_DOCUMENT

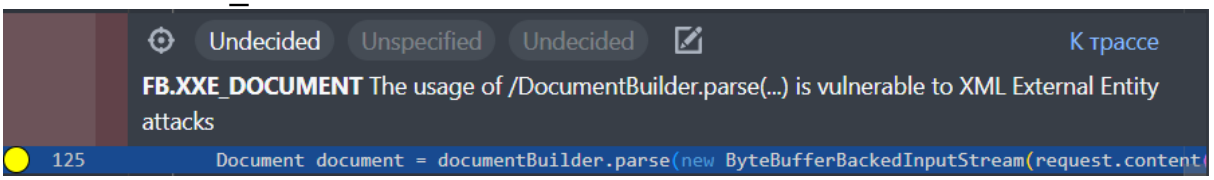


Рисунок 51 –Детектор FB.XXE_DOCUMENT в интерфейсе Svacer

Атаки внешних объектов XML (XXE) могут возникать, когда анализатор XML поддерживает сущности XML при обработке XML, полученного из ненадежного источника.

Этот параметр защитит вас от атак типа «отказ в обслуживании» и удаленного доступа к файлам.

```
DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();
dbf.setFeature(XMLConstants.FEATURE_SECURE_PROCESSING, true);
DocumentBuilder db = dbf.newDocumentBuilder();
Document doc = db.parse(input);
```

Отключив DTD, можно предотвратить почти все атаки XXE.

```
DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();
dbf.setFeature("http://apache.org/xml/features/disallow-doctype-decl", true);
DocumentBuilder db = dbf.newDocumentBuilder();
Document doc = db.parse(input);
```

FB.STRUTS1_ENDPOINT

Как только запрос будет перенаправлен на этот контроллер, автоматически будет создан экземпляр объекта Form, содержащий параметры

HTTP. Использование этих параметров следует пересмотреть, чтобы убедиться в их безопасности.

FB.STRUTS2_ENDPOINT

Когда запрос направляется к его контроллеру (например, к выбранному классу), предоставленные параметры HTTP автоматически сопоставляются с установщиками для класса. Следовательно, все установщики этого класса следует рассматривать как ненадежные входные данные, даже если форма не включает эти значения. Злоумышленник может просто предоставить дополнительные значения в запросе, и они в любом случае будут установлены в объекте, если у этого объекта есть такой установщик. Использование этих параметров следует пересмотреть, чтобы убедиться в их безопасности.

FB.SPRING_ENDPOINT

Этот класс является контроллером Spring. Все методы, помеченные RequestMapping (а также его ярлыки GetMapping, PostMapping, PutMapping, DeleteMapping и PatchMapping), доступны удаленно. Этот класс следует проанализировать, чтобы убедиться, что удаленно доступные методы безопасны для потенциальных злоумышленников.

FB.CUSTOM_INJECTION

Идентифицированный метод восприимчив к инъекциям. Ввод должен быть проверен и правильно экранирован.

FB.SQL_INJECTION_HIBERNATE

Входные значения, включенные в запросы SQL, должны передаваться безопасно. Связывание переменных в подготовленных операторах можно использовать для легкого снижения риска внедрения SQL. В качестве альтернативы для подготовки операторов можно использовать критерии Hibernate.

Уязвимый код:

```
Session session = sessionFactory.openSession();
Query q = session.createQuery("select t from UserEntity t where id = " + input);
q.execute();
```

Решение:

```
Session session = sessionFactory.openSession();
Query q = session.createQuery("select t from UserEntity t where id = :userId");
q.setString("userId",input);
```

```
q.execute();

Solution for dynamic queries (with Hibernate Criteria):
Session session = sessionFactory.openSession();
Query q = session.createCriteria(UserEntity.class)
   .add( Restrictions.like("id", input) )
   .list();
q.execute();
```

FB.SQL_INJECTION_JDO

Входные значения, включенные в запросы SQL, должны передаваться безопасно. Связывание переменных в подготовленных операторах можно использовать для легкого снижения риска внедрения SQL.

FB.SQL_INJECTION_JPA

Входные значения, включенные в запросы SQL, должны передаваться безопасно. Связывание переменных в подготовленных операторах можно использовать для легкого снижения риска внедрения SQL.

FB.SCRIPT_ENGINE_INJECTION

Динамический код оценивается. Необходимо провести тщательный анализ конструкции кода. Выполнение вредоносного кода может привести к утечке данных или компрометации операционной системы. Если предполагается оценка пользовательского кода, следует применить правильную изолированную программную среду. Безопаснее - оценка кода JavaScript с использованием библиотеки Cloudbees Rhino Sandbox.

FB.SPEL_INJECTION

Выражение Spring создается с динамическим значением. Источник значений должен быть проверен, чтобы избежать попадания нефильтрованных значений в эту рискованную оценку кода.

FB.BAD_HEXA_CONVERSION

При преобразовании массива байтов, содержащего хеш-подпись, в удобочитаемую строку может быть допущена ошибка преобразования, если массив читается побайтно. В следующем примере показано использование метода Integer.toHexString(), который удаляет все ведущие нули из каждого байта вычисленного хеш-значения.

```
MessageDigest md = MessageDigest.getInstance("SHA-256");
byte[] resultBytes = md.digest(password.getBytes("UTF-8"));

StringBuilder stringBuilder = new StringBuilder();
```

```
for(byte b :resultBytes) {
   stringBuilder.append( Integer.toHexString( b & 0xFF ) );
}

return stringBuilder.toString();
```

Эта ошибка ослабляет вычисленное значение хеш-функции, поскольку приводит к большему количеству коллизий. Например, хеш-значения «0x0679» и «0x6709» будут выводиться как «679» для вышеуказанной функции.

В этой ситуации метод Integer.toHexString() следует заменить на String.format() следующим образом:

```
stringBuilder.append( String.format( "%02X", b ) );
```

FB.HAZELCAST_SYMMETRIC_ENCRYPTION

Сетевая связь для Hazelcast настроена на использование симметричного шифра (вероятно, DES или Blowfish). Сами по себе эти шифры не обеспечивают целостность или безопасную аутентификацию. Предпочтительно использование асимметричного шифрования.

FB.NULL_CIPHER

NullCipher редко используется намеренно в производственных приложениях. Он реализует интерфейс Cipher, возвращая зашифрованный текст, идентичный предоставленному открытому тексту. В некоторых контекстах, например при тестировании, может подойти NullCipher. Избегайте использования NullCipher. Его случайное использование может создать значительный риск конфиденциальности.

FB.DES_USAGE

DES считается ненадежным шифром для современных приложений. В настоящее время NIST рекомендует использовать блочные шифры AES вместо DES.

FB.RSA_NO_PADDING

Программное обеспечение использует алгоритм RSA, но не включает оптимальное асимметричное заполнение шифрования (OAEP), которое может ослабить шифрование.

FB.HARD_CODE_PASSWORD

Пароли не должны храниться в исходном коде. Исходный код может широко распространяться в корпоративной среде и, безусловно, является открытым исходным кодом. Для безопасного управления паролями и секретными ключами следует хранить их в отдельных файлах конфигурации или хранилищах ключей. (Жестко запрограммированные ключи сообщаются отдельно по шаблону жестко закодированных ключей)

FB.STRUTS_FORM_VALIDATION

Входные данные формы должны иметь минимальную проверку ввода. Превентивная проверка помогает обеспечить глубокую защиту от различных рисков. Проверка может быть введена путем реализации метода проверки.

FB.XSS_REQUEST_WRAPPER

Реализация HttpServletRequestWrapper под названием XSSRequestWrapper была опубликована в различных блогах. Фильтрация слабая по нескольким причинам: она охватывает только параметры, а не заголовки и входы побочных каналов. Цепочку функций замены можно легко обойти.

Для более надежной защиты выберите решение, которое автоматически кодирует символы в представлении (шаблоне или JSP) в соответствии с правилами защиты XSS, определенными в шпаргалке по предотвращению XSS OWASP.

FB.BLOWFISH_KEY_SIZE

Шифр Blowfish поддерживает размеры ключей от 32 до 448 бит. Небольшой размер ключа делает зашифрованный текст уязвимым для атак методом перебора. При генерации ключа следует использовать не менее 128 бит энтропии, если требуется использование Blowfish. Если алгоритм можно изменить, вместо него следует использовать блочный шифр AES.

FB.RSA_KEY_SIZE

NIST рекомендует использовать для алгоритма RSA ключи длиной 2048 бит и выше.

FB.UNVALIDATED_REDIRECT

Непроверенные перенаправления происходят, когда приложение перенаправляет пользователя на целевой URL-адрес, указанный предоставленным пользователем параметром, который не проверен. Такие уязвимости могут быть использованы для облегчения фишинговых атак.

Решение/контрмеры:

Не принимать места назначения перенаправления от пользователей

Примите ключ назначения и используйте его для поиска целевого (законного) пункта назначения.

Принимать только относительные пути

URL-адреса белого списка (если возможно)

Убедитесь, что начало URL-адреса входит в белый список.

FB.XSS_JSP_PRINT

Обнаружен потенциальный XSS. Его можно использовать для выполнения нежелательного JavaScript в браузере клиента. (См. ссылки)

Лучшая защита от XSS — контекстно-зависимое кодирование вывода, как в примере выше. Обычно необходимо учитывать 4 контекста: HTML, JavaScript, CSS (стили) и URL-адреса. Пожалуйста, следуйте правилам защиты от XSS, определенным в шпаргалке по предотвращению XSS OWASP, в которой эти меры защиты описаны очень подробно.

FB.XSS_SERVLET

Обнаружен потенциальный XSS. Его можно использовать для выполнения нежелательного JavaScript в браузере клиента. Обычно необходимо учитывать 4 контекста: HTML, JavaScript, CSS (стили) и URL-адреса. Пожалуйста, следуйте правилам защиты от XSS, определенным в шпаргалке по предотвращению XSS OWASP, в которой эти меры защиты описаны очень подробно. Обратите внимание, что это правило XSS в сервлете ищет аналогичные проблемы, но ищет их иначе, чем существующие правила «XSS: сервлет отражает уязвимость межсайтового скриптинга» и «XSS: сервлет отражает уязвимость межсайтового скриптинга на странице ошибок» в FindBugs .

FB.XML_DECODER

XMLDecoder не следует использовать для анализа ненадежных данных. Десериализация пользовательского ввода может привести к выполнению произвольного кода. Это возможно, поскольку XMLDecoder поддерживает произвольный вызов метода. Эта возможность предназначена для вызова методов установки, но на практике можно вызвать любой метод.

Решение состоит в том, чтобы избежать использования XMLDecoder для анализа содержимого из ненадежного источника.

FB.ECB_MODE

Вместо режима электронной кодовой книги (ECB), который не обеспечивает хорошую конфиденциальность, следует использовать режим шифрования аутентификации, который обеспечивает лучшую конфиденциальность зашифрованных данных. В частности, режим ECB каждый раз выдает один и тот же результат для одного и того же входа. Так, например, если пользователь отправляет пароль, зашифрованное значение

каждый раз одно и то же. Это позволяет злоумышленнику перехватить и воспроизвести данные.

Чтобы исправить это, вместо этого следует использовать что-то вроде режима Галуа/Счетчика (GCM).

FB.PADDING_ORACLE

Этот конкретный режим CBC с PKCS5Padding подвержен атакам оракула заполнения. Злоумышленник потенциально может расшифровать сообщение, если система обнаружит разницу между открытым текстом с недопустимым заполнением и действительным заполнением. Различие между действительным и недопустимым заполнением обычно проявляется через отдельные сообщения об ошибках, возвращаемые для каждого условия.

FB.CIPHER_INTEGRITY

Полученный зашифрованный текст может быть изменен злоумышленником. Это означает, что шифр не дает возможности обнаружить, что данные были подделаны. Если злоумышленник может контролировать зашифрованный текст, он может быть изменен без обнаружения.

Решение состоит в том, чтобы использовать шифр, включающий хеш-код аутентификации сообщения (HMAC) для подписи данных. Объединение функции HMAC с существующим шифром подвержено ошибкам [1]. В частности, всегда рекомендуется сначала проверить HMAC, и только если данные не изменены, затем выполнять какие-либо криптографические функции над данными.

Следующие режимы уязвимы, поскольку они не предоставляют HMAC:
- CBC
- OFB
- CTR
- ECB

FB.ESAPI_ENCRYPTOR

ESAPI имеет небольшую историю уязвимостей в криптографическом компоненте. Вот краткий список проверок, позволяющий убедиться, что аутентифицированное шифрование работает должным образом.

1. Библиотечная версия

Эта проблема исправлена в ESAPI версии 2.1.0. Версии <= 2.0.1 уязвимы для обхода MAC-адреса (CVE-2013-5679).

2. Конфигурация:

Библиотека версии 2.1.0 по-прежнему уязвима из-за изменения размера ключа в определении зашифрованного текста (CVE-2013-5960). Необходимо принять некоторые меры предосторожности.

FB.ANDROID_EXTERNAL_FILE_ACCESS

Приложение записывает данные на внешнее хранилище (возможно, SD-карту). Это действие имеет множество последствий для безопасности. Во-первых, хранилище файлов на SD-карте будет доступно приложению, имеющему разрешение READ_EXTERNAL_STORAGE. Кроме того, если сохраненные данные содержат конфиденциальную информацию о пользователе, потребуется шифрование.

FB.ANDROID_BROADCAST

Намерения трансляции могут прослушиваться любым приложением, имеющим соответствующее разрешение. Рекомендуется по возможности избегать передачи конфиденциальной информации.

FB.ANDROID_WORLD_WRITABLE

Файл, написанный в этом контексте, использует режим создания MODE_WORLD_READABLE. Возможно, это не ожидаемое поведение — раскрытие записываемого контента.

FB.ANDROID_GEOLOCATION

Предлагается запросить у пользователя подтверждение получения его геолокации.

FB.ANDROID_WEB_VIEW_JAVASCRIPT

Включение JavaScript для WebView означает, что он теперь подвержен XSS. Рендеринг страницы должен быть проверен на наличие потенциально отраженного XSS, сохраненного XSS и DOM XSS.

FB.ANDROID_WEB_VIEW_JAVASCRIPT_INTERFACE

Использование интерфейса JavaScript может подвергнуть WebView риску использования API. Если в WebView срабатывает XSS, класс может быть вызван вредоносным кодом JavaScript.

FB.BSHIFT_WRONG_ADD_PRIORITY

Код выполняет операцию типа (x << 8 + y). Хотя это может быть и правильно, вероятно, оно должно было выполнять (x << 8) + y, но операция сдвига имеет более низкий приоритет, поэтому на самом деле она анализируется как x << (8 + y).

FB.AM_CREATES_EMPTY_JAR_FILE_ENTRY

Код вызывает putNextEntry(), за которым сразу же следует вызов closeEntry(). В результате получается пустая запись JarFile. Содержимое записи должно быть записано в JarFile между вызовами putNextEntry() и closeEntry().

FB.AM_CREATES_EMPTY_ZIP_FILE_ENTRY

Код вызывает putNextEntry(), за которым сразу же следует вызов closeEntry(). В результате получается пустая запись ZipFile. Содержимое записи должно быть записано в ZipFile между вызовами putNextEntry() и closeEntry().

FB.CN_IDIOM_NO_SUPER_CALL

Этот нефинальный класс определяет метод clone(), который не вызывает super.clone(). Если этот класс («A») расширен подклассом («B»), а подкласс B вызывает super.clone(), то вполне вероятно, что метод clone() класса B вернет объект типа A, что нарушает стандартный контракт для clone().

Если все методы clone() вызывают super.clone(), то они гарантированно используют Object.clone(), который всегда возвращает объект правильного типа.

FB.CO_ABSTRACT_SELF

Этот класс определяет ковариантную версию метода CompareTo(). Чтобы правильно переопределить метод CompareTo() в интерфейсе Comparable, параметр CompareTo() должен иметь тип java.lang.Object.

FB.CO_SELF_NO_OBJECT

Этот класс определяет ковариантную версию метода CompareTo(). Чтобы правильно переопределить метод CompareTo() в интерфейсе Comparable, параметр CompareTo() должен иметь тип java.lang.Object.

FB.DE_MIGHT_DROP

Этот метод может исключить исключение. В общем, исключения должны каким-либо образом обрабатываться или сообщаться о них, либо их следует выбрасывать из метода.

FB.DMI_ENTRY_SETS_MAY_REUSE_ENTRY_OBJECTS

Методу entrySet() разрешено возвращать представление базовой карты, в которой один объект Entry используется повторно и возвращается во время итерации. Начиная с Java 6, это делали как IdentityHashMap, так и EnumMap. При итерации по такой карте значение Entry действительно только до тех пор, пока вы не перейдете к следующей итерации. Если, например, вы

попытаетесь передать такой набор записей в метод addAll, все пойдет совсем не так.

FB.EQ_ABSTRACT_SELF

Этот класс определяет ковариантную версию метода equals(). Чтобы правильно переопределить метод equals() в java.lang.Object, параметр equals() должен иметь тип java.lang.Object.

FB.EQ_SELF_NO_OBJECT

Этот класс определяет ковариантную версию метода equals(). Чтобы правильно переопределить метод equals() в java.lang.Object, параметр equals() должен иметь тип java.lang.Object.

FB.FI_FINALIZER_ONLY_NULLS_FIELDS

Этот финализатор ничего не делает, кроме полей с нулевым значением. Это совершенно бессмысленно и требует, чтобы объект был собран мусором, финализирован, а затем снова собран мусором. Вам следует просто удалить метод Finalize.

FB.FI_MISSING_SUPER_CALL

Этот метод Finalize() не вызывает метод Finalize() своего суперкласса. Таким образом, любые действия финализатора, определенные для суперкласса, выполняться не будут. Добавьте вызов super.finalize().

FB.GC_UNCHECKED_TYPE_IN_GENERIC_CALL

Этот вызов универсального метода коллекции передает аргумент при компиляции типа Object, где ожидается определенный тип из параметров универсального типа. Таким образом, ни стандартная система типов Java, ни статический анализ не могут предоставить полезную информацию о том, принадлежит ли объект, передаваемый в качестве параметра, к соответствующему типу.

FB.HE_HASHCODE_NO_EQUALS

Этот класс определяет метод hashCode(), но не метод Equals(). Следовательно, класс может нарушить инвариант, согласно которому равные объекты должны иметь равные хэш-коды.

FB.HE_HASHCODE_USE_OBJECT_EQUALS

Этот класс определяет метод hashCode(), но наследует его метод equals() от java.lang.Object (который определяет равенство путем сравнения ссылок на объекты). Хотя это, вероятно, будет соответствовать контракту, согласно которому равные объекты должны иметь равные хэш-коды, переопределение метода hashCode(), вероятно, не является тем, что

предполагалось. (Переопределение hashCode() подразумевает, что идентичность объекта основана на критериях, более сложных, чем простое равенство ссылок.)

FB.ISC_INSTANTIATE_STATIC_CLASS

Этот класс выделяет объект, основанный на классе, который предоставляет только статические методы. Этот объект не нужно создавать, просто получите доступ к статическим методам напрямую, используя имя класса в качестве квалификатора.

FB.J2EE_STORE_OF_NON_SERIALIZABLE_OBJECT_INTO_SESSION

Кажется, этот код сохраняет несериализуемый объект в HttpSession. Если этот сеанс пассивирован или перенесен, возникнет ошибка.

FB.JCIP_FIELD_ISNT_FINAL_IN_IMMUTABLE_CLASS

Класс аннотирован с помощью net.jcip.annotations.Immutable или javax.annotation.concurrent.Immutable, и правила для этих аннотаций требуют, чтобы все поля были конечными.

FB.NM_CLASS_NOT_EXCEPTION

Этот класс не является производным от другого исключения, а заканчивается на «Exception». Это будет сбивать с толку пользователей этого класса.

FB.NM_CONFUSING

У упомянутых методов есть имена, которые отличаются только заглавными буквами.

FB.NM_FIELD_NAMING_CONVENTION

Имена полей, которые не являются окончательными, должны быть написаны в смешанном регистре, причем первая буква должна быть строчной, а первые буквы последующих слов — заглавными. Имена последних полей должны быть написаны прописными буквами, а слова разделены подчеркиванием («_»).

FB.NM_FUTURE_KEYWORD_USED_AS_IDENTIFIER

Идентификатор — это слово, которое зарезервировано как ключевое слово в более поздних версиях Java, и ваш код необходимо будет изменить, чтобы скомпилировать его в более поздних версиях Java.

FB.NM_FUTURE_KEYWORD_USED_AS_MEMBER_IDENTIFIER

Этот идентификатор используется как ключевое слово в более поздних версиях Java. Этот код и любой код, ссылающийся на этот API, необходимо будет изменить, чтобы скомпилировать его в более поздних версиях Java.

FB.NP_METHOD_RETURN_RELAXING_ANNOTATION

Метод всегда должен реализовывать контракт метода, который он переопределяет. Таким образом, если метод take помечен как возвращающий значение @Nonnull, не следует переопределять этот метод в подклассе методом, помеченным как возвращающий значение @Nullable или @CheckForNull. Это нарушает договор, согласно которому метод не должен возвращать значение null.

FB.NP_OPTIONAL_RETURN_NULL

Использование необязательного типа возврата (java.util.Optional или com.google.common.base.Optional) всегда означает, что явные нулевые возвраты не были желательны по замыслу. Возврат нулевого значения в таком случае является нарушением контракта и, скорее всего, приведет к поломке клиентского кода.

FB.NM_SAME_SIMPLE_NAME_AS_SUPERCLASS

Этот класс имеет простое имя, идентичное имени его суперкласса, за исключением того, что его суперкласс находится в другом пакете (например, Alpha.Foo расширяет beta.Foo). Это может сбивать с толку, создавать множество ситуаций, в которых вам придется просматривать операторы импорта для разрешения ссылок, и создает множество возможностей для случайного определения методов, которые не переопределяют методы в своих суперклассах.

FB.NM_VERY_CONFUSING_INTENTIONAL

У упомянутых методов есть имена, которые отличаются только заглавными буквами. Это очень сбивает с толку, поскольку если бы капитализация была идентична, один из методов переопределял бы другой. Судя по существованию других методов, кажется, что существование обоих этих методов является преднамеренным, но это определенно сбивает с толку. Вам следует постараться устранить один из них, если только вы не вынуждены иметь оба из-за замороженных API.

FB.NM_WRONG_PACKAGE_INTENTIONAL

Метод в подклассе не переопределяет аналогичный метод в суперклассе, поскольку тип параметра не совсем соответствует типу соответствующего параметра в суперклассе.

FB.OS_OPEN_STREAM

Метод создает объект потока ввода-вывода, не присваивает его никаким полям, не передает другим методам, которые могли бы его закрыть или вернуть, и не закрывает поток на всех путях выхода из метода. Это может привести к утечке файлового дескриптора. Обычно рекомендуется использовать блокfinally, чтобы гарантировать закрытие потоков.

FB.OS_OPEN_STREAM_EXCEPTION_PATH

Метод создает объект потока ввода-вывода, не назначает его никаким полям, не передает другим методам и не возвращает, а также не закрывает его на всех возможных путях исключений из метода. Это может привести к утечке файлового дескриптора. Обычно рекомендуется использовать блокfinally, чтобы гарантировать закрытие потоков.

FB.RR_NOT_CHECKED

Этот метод игнорирует возвращаемое значение одного из вариантов java.io.InputStream.read(), который может возвращать несколько байтов. Если возвращаемое значение не проверено, вызывающая сторона не сможет правильно обработать случай, когда было прочитано меньше байтов, чем запрошено вызывающей стороной. Это особенно коварный вид ошибки, поскольку во многих программах при чтении из входных потоков обычно считывается весь объем запрошенных данных, что приводит к сбою программы лишь время от времени.

FB.SR_NOT_CHECKED

Этот метод игнорирует возвращаемое значение java.io.InputStream.skip(), которое может пропускать несколько байтов. Если возвращаемое значение не проверено, вызывающая сторона не сможет правильно обработать случай, когда было пропущено меньше байтов, чем запрошено вызывающей стороной. Это особенно коварный тип ошибки, поскольку во многих программах пропуски входных потоков обычно пропускают весь объем запрошенных данных, что приводит к сбою программы лишь время от времени. Однако в буферизованных потоках метод Skip() будет пропускать только данные в буфере и обычно не сможет пропустить запрошенное количество байтов.

FB.SW_SWING_METHODS_INVOKED_IN_SWING_THREAD

Методы Swing show(), setVisible() и package() создадут связанный одноранговый узел для кадра. При создании узла система создает поток отправки событий. Это усложняет ситуацию, поскольку поток отправки событий может уведомлять слушателей, пока обработка упаковки и проверки еще продолжается. Эта ситуация может привести к тому, что два потока будут проходить через графический интерфейс на основе компонентов Swing — это

серьезный недостаток, который может привести к взаимоблокировкам или другим связанным с потоками проблемам. Вызов пакета приводит к реализации компонентов. По мере их реализации (то есть не обязательно видимых) они могут вызвать уведомление прослушивателя в потоке отправки событий.

FB.SE_NONFINAL_SERIALVERSIONID

Этот класс определяет поле SerialVersionUID, которое не является окончательным. Поле следует сделать окончательным, если оно предназначено для указания UID версии для целей сериализации.

FB.SE_NONLONG_SERIALVERSIONID

Этот класс определяет поле SerialVersionUID, которое не является длинным. Поле следует сделать длинным, если оно предназначено для указания UID версии для целей сериализации.

FB.SE_NONSTATIC_SERIALVERSIONID

Этот класс определяет поле SerialVersionUID, которое не является статическим. Поле следует сделать статическим, если оно предназначено для указания UID версии в целях сериализации.

FB.SE_READ_RESOLVE_MUST_RETURN_OBJECT

Чтобы метод readResolve распознавался механизмом сериализации, необходимо объявить, что он имеет возвращаемый тип Object.

FB.BIT_AND

Этот метод сравнивает выражение формы (e и C) с D, которое всегда будет неравным из-за определенных значений констант C и D. Это может указывать на логическую ошибку или опечатку.

FB.BOA_BADLY_OVERRIDDEN_ADAPTER

Этот метод переопределяет метод, найденный в родительском классе, где этот класс является адаптером, реализующим прослушиватель, определенный в пакете java.awt.event или javax.swing.event. В результате этот метод не будет вызываться при возникновении события.

FB.BX_UNBOXED_AND_COERCED_FOR_TERNARY_OPERATOR

Обернутое примитивное значение распаковывается и преобразуется в другой тип примитива как часть вычисления условного тернарного оператора (оператор b?e1:e2). Семантика Java требует, чтобы, если e1 и e2 являются обернутыми числовыми значениями, значения распаковываются и преобразуются/приводятся к их общему типу (например, если e1 имеет тип Integer, а e2 имеет тип Float, тогда e1 распаковывается и преобразуется в значение с плавающей запятой и упакованное в коробку.

FB.CO_COMPARETO_RESULTS_MIN_VALUE

В некоторых ситуациях этот метод CompareTo или Compare возвращает константу Integer.MIN_VALUE, что является исключительно плохой практикой. Единственное, что имеет значение для возвращаемого значения CompareTo, — это знак результата. Но люди иногда отрицают возвращаемое значение CompareTo, ожидая, что это изменит знак результата. Так и будет, за исключением случая, когда возвращаемое значение — Integer.MIN_VALUE. Поэтому просто верните -1, а не Integer.MIN_VALUE.

FB.DLS_DEAD_STORE_OF_CLASS_LITERAL

Эта инструкция присваивает переменной литерал класса, а затем никогда его не использует. Поведение этого отличается в Java 1.4 и Java 5. В Java 1.4 и более ранних версиях ссылка на Foo.class приведет к принудительному выполнению статического инициализатора Foo, если он еще не был выполнен. В Java 5 и более поздних версиях этого нет.

FB.DLS_OVERWRITTEN_INCREMENT

Код выполняет операцию увеличения/уменьшения (например, i++/i--), а затем немедленно перезаписывает ее. Например, i = i++ / i = i — немедленно перезаписывает увеличенное/уменьшенное значение исходным значением.

FB.DMI_ARGUMENTS_WRONG_ORDER

Аргументы вызова этого метода расположены в неправильном порядке. Например, вызов Preconditions.checkNotNull("message", message) имеет зарезервированные аргументы: проверяемое значение является первым аргументом.

FB.DMI_CALLING_NEXT_FROM_HASNEXT

Метод hasNext() вызывает метод next(). Это почти наверняка неверно, поскольку метод hasNext() не должен изменять состояние итератора, а метод next должен изменять состояние итератора.

FB.DMI_DOH

Этот конкретный вызов метода не имеет смысла по причинам, которые должны быть очевидны при рассмотрении.

FB.DMI_LONG_BITS_TO_DOUBLE_INVOKED_ON_INT

Вызывается метод Double.longBitsToDouble, но в качестве аргумента передается 32-битное целое число. Это почти наверняка не предназначено и вряд ли даст желаемый результат.

FB.DMI_FUTILE_ATTEMPT_TO_CHANGE_MAXPOOL_SIZE_OF_SCHEDULED_THREAD_POOL_EXECUTOR

Хотя ScheduledThreadPoolExecutor наследует от ThreadPoolExecutor, некоторые унаследованные методы настройки для него бесполезны. В частности, поскольку он действует как пул фиксированного размера, использующий потоки corePoolSize и неограниченную очередь, корректировка MaximumPoolSize не имеет никакого полезного эффекта.

FB.DMI_SCHEDULED_THREAD_POOL_EXECUTOR_WITH_ZERO_CORE_THREADS

ScheduledThreadPoolExecutor с нулевым потоком ядра никогда ничего не выполнит; изменения максимального размера пула игнорируются.

FB.DMI_VACUOUS_CALL_TO_EASYMOCK_METHOD

Этот вызов не передает никаких объектов методу EasyMock, поэтому он ничего не делает.

FB.EC_INCOMPATIBLE_ARRAY_COMPARE

Этот метод вызывает .equals(Object o) для сравнения двух массивов, но массивов несовместимых типов (например, String[] и StringBuffer[] или String[] и int[]). Они никогда не будут равны. Кроме того, когда метод равенства(...) используется для сравнения массивов, он только проверяет, являются ли они одним и тем же массивом, и игнорирует содержимое массивов.

FB.EQ_DONT_DEFINE_EQUALS_FOR_ENUM

Этот класс определяет перечисление, а равенство перечислений определяется с использованием идентификатора объекта. Определение метода ковариантного равенства для значения перечисления является исключительно плохой практикой, поскольку это, скорее всего, приведет к тому, что два разных значения перечисления будут сравниваться как равные при использовании ковариантного метода перечисления и как неравные при обычном сравнении.

FB.EQ_OTHER_NO_OBJECT

Этот класс определяет метод Equals(), который не переопределяет обычный метод Equals(Object), определенный в базовом классе java.lang.Object. Вместо этого он наследует метод Equals(Object) от суперкласса. Класс, вероятно, должен определить логический метод равенства (Object).

FB.EQ_OTHER_USE_OBJECT

Этот класс определяет метод Equals(), который не переопределяет обычный метод Equals(Object), определенный в базовом классе

java.lang.Object. Класс, вероятно, должен определить логический метод равенства (Object).

FB.HE_SIGNATURE_DECLARES_HASHING_OF_UNHASHABLE_CLASS

Метод, поле или класс объявляет общую подпись, в которой нехешируемый класс используется в контексте, где требуется хешируемый класс. Класс, который объявляет метод равенства, но наследует метод hashCode() от Object, является нехешируемым, поскольку он не удовлетворяет требованию, чтобы равные объекты имели равные хеш-коды.

FB.IJU_BAD_SUITE_METHOD

Класс представляет собой JUnit TestCase и определяет метод suite(). Однако метод набора необходимо объявить.

FB.IJU_NO_TESTS

Класс представляет собой JUnit TestCase, но не реализовал никаких методов тестирования.

FB.IJU_SETUP_NO_SUPER

Класс представляет собой JUnit TestCase и реализует метод setUp. Метод setUp должен вызывать super.setUp(), но этого не происходит.

FB.IJU_SUITE_NOT_STATIC

Класс представляет собой JUnit TestCase и реализует метод suite(). Метод набора должен быть объявлен как статический, но это не так.

FB.IJU_TEARDOWN_NO_SUPER

Класс представляет собой JUnit TestCase и реализует метод TearDown. Метод TearDown должен вызывать super.tearDown(), но этого не происходит.

FB.IM_MULTIPLYING_RESULT_OF_IREM

Код умножает результат оставшегося целого числа на целочисленную константу. Убедитесь, что вы не перепутали приоритет операторов. Например, i % 60 * 1000 — это (i % 60) * 1000, а не i % (60 * 1000).

FB.INT_BAD_COMPARISON_WITH_INT_VALUE

Этот код сравнивает значение int с длинной константой, находящейся за пределами диапазона значений, которые могут быть представлены как значение int. Это сравнение бессмысленно и, возможно, неверно.

FB.IO_APPENDING_TO_OBJECT_OUTPUT_STREAM

Этот код открывает файл в режиме добавления, а затем переносит результат в поток вывода объекта.

Это не позволит вам добавлять данные к существующему потоку вывода объекта, хранящемуся в файле. Если вы хотите иметь возможность добавлять данные к потоку вывода объекта, вам необходимо держать поток вывода объекта открытым.

Единственная ситуация, в которой может работать открытие файла в режиме добавления и запись потока вывода объекта, — это если при чтении файла вы планируете открыть его в режиме произвольного доступа и искать смещение байта, с которого началось добавление.

FB.MF_METHOD_MASKS_FIELD

Этот метод определяет локальную переменную с тем же именем, что и поле в этом классе или суперклассе. Это может привести к тому, что метод прочитает неинициализированное значение из поля, оставит поле неинициализированным или и то, и другое.

FB.NP_ALWAYS_NULL

Здесь разыменовывается нулевой указатель. Это приведет к исключению NullPointerException при выполнении кода.

FB.NP_ALWAYS_NULL_EXCEPTION

Здесь разыменовывается указатель, имеющий значение null на пути исключения. Это приведет к исключению NullPointerException при выполнении кода. Обратите внимание: поскольку SpotBugs в настоящее время не отсекает невозможные пути исключений, это может быть ложным предупреждением.

Также обратите внимание, что SpotBugs считает вариант оператора переключателя по умолчанию путем исключения, поскольку случай по умолчанию часто невозможен.

FB.NP_ARGUMENT_MIGHT_BE_NULL

Параметр этого метода был идентифицирован как значение, которое всегда следует проверять, чтобы определить, является ли оно нулевым, но оно разыменовывается без предварительной проверки на нулевое значение.

FB.NP_CLOSING_NULL

close() вызывается для значения, которое всегда равно нулю. Если этот оператор будет выполнен, возникнет исключение нулевого указателя. Но здесь большой риск: вы никогда не закроете то, что должно быть закрыто.

FB.NP_GUARANTEED_DEREF

Существует оператор или ветвь, выполнение которой гарантирует, что значение в этот момент равно нулю, и это значение гарантированно будет разыменовано (за исключением прямых путей, включающих исключения во время выполнения).

Обратите внимание, что такая проверка, как if (x == null) генерирует new NullPointerException(); рассматривается как разыменование x.

FB.NP_GUARANTEED_DEREF_ON_EXCEPTION_PATH

На пути исключения существует оператор или ветвь, выполнение которой гарантирует, что значение в этот момент равно нулю, и это значение гарантированно будет разыменовано (за исключением прямых путей, включающих исключения во время выполнения).

FB.NP_NONNULL_FIELD_NOT_INITIALIZED_IN_CONSTRUCTOR

Поле помечено как ненулевое, но конструктор не записал его. Поле может быть инициализировано в другом месте во время конструктора или всегда может быть инициализировано перед использованием.

FB.NP_NULL_INSTANCEOF

Этот тест экземпляра всегда будет возвращать значение false, поскольку проверяемое значение гарантированно будет нулевым. Хотя это и безопасно, убедитесь, что это не является признаком какого-то недопонимания или какой-либо другой логической ошибки.

FB.NP_NULL_ON_SOME_PATH

Существует ветвь инструкции, которая в случае выполнения гарантирует, что нулевое значение будет разыменовано, что приведет к генерации исключения NullPointerException при выполнении кода. Конечно, проблема может заключаться в том, что ветвь или оператор невозможны и что исключение нулевого указателя никогда не может быть выполнено; решить это за пределами возможностей SpotBugs.

FB.NP_NULL_ON_SOME_PATH_EXCEPTION

Здесь разыменовывается ссылочное значение, равное нулю на некотором пути управления исключениями. Это может привести к исключению NullPointerException при выполнении кода. Обратите внимание: поскольку SpotBugs в настоящее время не отсекает невозможные пути исключений, это может быть ложным предупреждением.

Также обратите внимание, что SpotBugs считает вариант оператора переключателя по умолчанию путем исключения, поскольку случай по умолчанию часто невозможен.

FB.NP_NULL_PARAM_DEREF

Этот вызов метода передает нулевое значение для параметра метода, отличного от NULL. Либо параметр помечен как параметр, который всегда должен быть ненулевым, либо анализ показал, что он всегда будет разыменован.

FB.NP_NULL_PARAM_DEREF_ALL_TARGETS_DANGEROUS

Значение, возможно, нулевое, передается на сайте вызова, где все известные целевые методы требуют, чтобы параметр был ненулевым. Либо параметр помечен как параметр, который всегда должен быть ненулевым, либо анализ показал, что он всегда будет разыменован.

FB.NP_NULL_PARAM_DEREF_NONVIRTUAL

Возможно, нулевое значение передается ненулевому параметру метода. Либо параметр помечен как параметр, который всегда должен быть ненулевым, либо анализ показал, что он всегда будет разыменован.

FB.NP_STORE_INTO_NONNULL_FIELD

Значение, которое может быть нулевым, сохраняется в поле, помеченном как @Nonnull.

FB.NM_BAD_EQUAL

Этот класс определяет метод равный(Object). Этот метод не переопределяет метод Equals(Object) в java.lang.Object, что, вероятно, и было задумано.

FB.NM_LCASE_HASHCODE

Этот класс определяет метод hashcode(). Этот метод не переопределяет метод hashCode() в java.lang.Object, что, вероятно, и было задумано.

FB.NM_LCASE_TOSTRING

Этот класс определяет метод tostring(). Этот метод не переопределяет метод toString() в java.lang.Object, что, вероятно, и было задумано.

FB.NM_METHOD_CONSTRUCTOR_CONFUSION

Этот обычный метод имеет то же имя, что и класс, в котором он определен. Вероятно, он задумывался как конструктор. Если он должен был быть конструктором, удалите объявление возвращаемого значения void. Если вы случайно определили этот метод, осознали ошибку, определили

правильный конструктор, но не можете избавиться от этого метода из-за обратной совместимости, объявите этот метод устаревшим.

FB.QBA_QUESTIONABLE_BOOLEAN_ASSIGNMENT

Этот метод присваивает буквальное логическое значение (истина или ложь) логической переменной внутри выражения if или while. Скорее всего, это должно было быть логическое сравнение с использованием ==, а не присваивание с использованием =.

FB.RCN_REDUNDANT_NULLCHECK_WOULD_HAVE_BEEN_A_NPE

Здесь проверяется значение, чтобы определить, является ли оно нулевым, но это значение не может быть нулевым, поскольку ранее оно было разыменовано, и если бы оно было нулевым, при предыдущем разыменовании возникло бы исключение нулевого указателя. По сути, этот код и предыдущее разыменование расходятся во мнениях относительно того, может ли это значение быть нулевым. Либо проверка избыточна, либо предыдущее разыменование ошибочно.

FB.RE_CANT_USE_FILE_SEPARATOR_AS_REGULAR_EXPRESSION

В приведенном здесь коде используется File.separator, где требуется регулярное выражение. Это не удастся на платформах Windows, где File.separator представляет собой обратную косую черту, которая интерпретируется в регулярном выражении как escape-символ. Среди других опций вы можете просто использовать File.separatorChar=='\\' ? "\\\\" : File.separator вместо File.separator.

FB.RV_01_TO_INT

Случайное значение от 0 до 1 приводится к целочисленному значению 0. Вероятно, вы захотите умножить случайное значение на что-то еще, прежде чем приводить его к целому числу, или использовать метод Random.nextInt(n).

FB.RV_CHECK_COMPARETO_FOR_SPECIFIC_RETURN_VALUE

Этот код вызывает метод CompareTo или Compare и проверяет, является ли возвращаемое значение конкретным значением, например 1 или -1. При вызове этих методов следует проверять только знак результата, а не какое-либо конкретное ненулевое значение. Хотя многие или большинство методов CompareTo и Compare возвращают только -1, 0 или 1, некоторые из них возвращают другие значения.

FB.SA_FIELD_SELF_COMPUTATION

Этот метод выполняет бессмысленное вычисление поля с другой ссылкой на то же поле (например, x&x или x-x). Из-за характера вычислений эта операция кажется бессмысленной и может указывать на опечатку или логическую ошибку. Дважды проверьте расчет.

FB.SIC_THREADLOCAL_DEADLY_EMBRACE

Этот класс является внутренним классом, но, вероятно, должен быть статическим внутренним классом. В действительности существует серьезная опасность смертельного столкновения между внутренним классом и потоком, локальным во внешнем классе. Поскольку внутренний класс не является статическим, он сохраняет ссылку на внешний класс. Если локальный поток содержит ссылку на экземпляр внутреннего класса, внутренний и внешний экземпляры будут доступны и не подлежат сборке мусора.

FB.SIO_SUPERFLUOUS_INSTANCEOF

Проверка типа выполняется с помощью оператора экземпляра, где можно статически определить, относится ли объект к запрошенному типу.

FB.SQL_BAD_RESULTSET_ACCESS

Вызов методов getXXX или updateXXX набора результатов был выполнен там, где индекс поля равен 0. Поскольку поля ResultSet начинаются с индекса 1, это всегда является ошибкой.

FB.STI_INTERRUPTED_ON_CURRENTTHREAD

Этот метод вызывает вызов Thread.currentThread() только для вызова метода прерывания(). Поскольку прерывание() является статическим методом, проще и понятнее использовать Thread.interrupted().

FB.SE_READ_RESOLVE_IS_STATIC

Чтобы метод readResolve распознавался механизмом сериализации, его нельзя объявлять как статический метод.

FB.TQ_ALWAYS_VALUE_USED_WHERE_NEVER_REQUIRED

Значение, указанное как содержащее аннотацию квалификатора типа, используется в месте или местах, требующих, чтобы значение не содержало эту аннотацию.

Точнее, значение, помеченное квалификатором типа, указывающим, when=ALWAYS, гарантированно достигнет использования или применений, где тот же квалификатор типа указывает, when=NEVER.

FB.TQ_COMPARING_VALUES_WITH_INCOMPATIBLE_TYPE_QUALIFIERS

Значение, указанное как содержащее аннотацию квалификатора типа, сравнивается со значением, которое никогда не содержит этот квалификатор.

Точнее, значение, аннотированное квалификатором типа, указывающим, when=ALWAYS, сравнивается со значением, где тот же квалификатор типа указывает, when=NEVER.

FB.TQ_MAYBE_SOURCE_VALUE_REACHES_ALWAYS_SINK

Значение, которое помечено как возможно не являющееся экземпляром значений, обозначенных квалификатором типа, и это значение гарантированно будет использоваться таким образом, который требует значений, обозначенных этим квалификатором типа.

FB.TQ_MAYBE_SOURCE_VALUE_REACHES_NEVER_SINK

Значение, которое помечено как возможное экземпляр значений, обозначенных квалификатором типа, и это значение гарантированно будет использоваться таким образом, который запрещает значения, обозначаемые этим квалификатором типа.

FB.TQ_NEVER_VALUE_USED_WHERE_ALWAYS_REQUIRED

Значение, указанное как не содержащее аннотацию квалификатора типа, гарантированно будет использовано в месте или местах, где требуется, чтобы значение содержало эту аннотацию.

Точнее, значение, помеченное квалификатором типа, указывающим, when=NEVER, гарантированно достигнет использования или использования, где тот же квалификатор типа указывает, when=ALWAYS.

FB.TQ_UNKNOWN_VALUE_USED_WHERE_ALWAYS_STRICTLY_REQUIRED

Значение используется таким образом, что требуется аннотация значения квалификатором типа. Квалификатор типа является строгим, поэтому инструмент отклоняет любые значения, не имеющие соответствующей аннотации.

Чтобы заставить значение иметь строгую аннотацию, определите функцию идентификации, в которой возвращаемое значение будет помечено строгой аннотацией. Это единственный способ превратить неаннотированное значение в значение со строгой аннотацией квалификатора типа.

FB.LG_LOST_LOGGER_DUE_TO_WEAK_REFERENCE

OpenJDK представляет потенциальную несовместимость. В частности, изменилось поведение java.util.logging.Logger. Вместо использования

сильных ссылок теперь внутри системы используются слабые ссылки. Это разумное изменение, но, к сожалению, некоторый код использует старое поведение — при изменении конфигурации регистратора он просто удаляет ссылку на регистратор. Это означает, что сборщик мусора может освободить эту память, а это означает, что конфигурация регистратора потеряна.

FB.OBL_UNSATISFIED_OBLIGATION

Этот метод может не выполнить очистку (закрытие, удаление) потока, объекта базы данных или другого ресурса, требующего явной операции очистки.

Как правило, если метод открывает поток или другой ресурс, метод должен использовать блок try/finally, чтобы гарантировать очистку потока или ресурса до возврата метода.

Этот шаблон ошибок по существу аналогичен шаблонам ошибок OS_OPEN_STREAM и ODR_OPEN_DATABASE_RESOURCE, но основан на другой (и, мы надеемся, лучшей) технике статического анализа.

FB.OBL_UNSATISFIED_OBLIGATION_EXCEPTION_EDGE

Этот метод может не выполнить очистку (закрытие, удаление) потока, объекта базы данных или другого ресурса, требующего явной операции очистки.

Как правило, если метод открывает поток или другой ресурс, метод должен использовать блок try/finally, чтобы гарантировать очистку потока или ресурса до возврата метода.

Этот шаблон ошибок по существу аналогичен шаблонам ошибок OS_OPEN_STREAM и ODR_OPEN_DATABASE_RESOURCE, но основан на другой (и, мы надеемся, лучшей) технике статического анализа.

FB.DM_CONVERT_CASE

Строка преобразуется в верхний или нижний регистр с использованием кодировки платформы по умолчанию. Это может привести к неправильным преобразованиям при использовании международных символов. Используйте String.toUpperCase(Locale l), String.toLowerCase(Locale l) версии вместо этого.

FB.DP_DO_INSIDE_DO_PRIVILEGED

Этот код вызывает метод, требующий проверки разрешений безопасности. Если этому коду будут предоставлены разрешения

безопасности, но он может быть вызван кодом, у которого нет разрешений безопасности, тогда вызов должен происходить внутри блока doPrivileged.

FB.EI_EXPOSE_STATIC_REP2

Этот код сохраняет ссылку на внешне изменяемый объект в статическом поле. Если непроверенные изменения изменяемого объекта поставят под угрозу безопасность или другие важные свойства, вам придется сделать что-то другое. Во многих ситуациях лучше хранить копию объекта.

FB.MS_CANNOT_BE_FINAL

Изменяемое статическое поле может быть изменено вредоносным кодом или случайно из другого пакета. К сожалению, способ использования поля не позволяет легко решить эту проблему.

FB.MS_FINAL_PKGPROTECT

Изменяемое статическое поле может быть изменено вредоносным кодом или случайно из другого пакета. Поле можно сделать защищенным пакетом и/или сделать окончательным, чтобы избежать этой уязвимости.

FB.MS_MUTABLE_ARRAY

Последнее статическое поле ссылается на массив, и к нему может получить доступ вредоносный код или случайно из другого пакета. Этот код может свободно изменять содержимое массива.

FB.MS_MUTABLE_HASHTABLE

Последнее статическое поле ссылается на хэш-таблицу и может быть доступно вредоносному коду или случайно из другого пакета. Этот код может свободно изменять содержимое Hashtable.

FB.MS_OOI_PKGPROTECT

Последнее статическое поле, определенное в интерфейсе, ссылается на изменяемый объект, например массив или хеш-таблицу. Этот изменяемый объект может быть изменен вредоносным кодом или случайно из другого пакета. Чтобы решить эту проблему, поле необходимо переместить в класс и сделать пакет защищенным, чтобы избежать этой уязвимости.

FB.MS_PKGPROTECT

Изменяемое статическое поле может быть изменено вредоносным кодом или случайно. Поле можно сделать защищенным пакетом, чтобы избежать этой уязвимости.

FB.MS_SHOULD_BE_FINAL

Это общедоступное статическое или защищенное статическое поле не является окончательным и может быть изменено вредоносным кодом или случайно из другого пакета. Чтобы избежать этой уязвимости, поле можно сделать окончательным.

FB.MS_SHOULD_BE_REFACTORED_TO_BE_FINAL

Это общедоступное статическое или защищенное статическое поле не является окончательным и может быть изменено вредоносным кодом или случайно из другого пакета. Чтобы избежать этой уязвимости, поле можно сделать окончательным. Однако статический инициализатор содержит более одной записи в поле, поэтому для этого потребуется некоторый рефакторинг.

FB.DC_DOUBLECHECK

Этот метод может содержать экземпляр блокировки с двойной проверкой. Эта идиома неверна с точки зрения семантики модели памяти Java.

FB.DM_MONITOR_WAIT_ON_CONDITION

Этот метод вызывает wait() для объекта java.util.concurrent.locks.Condition. Ожидание условия должно выполняться с использованием одного из методов await(), определенных интерфейсом Condition.

FB.DM_USELESS_THREAD

Этот метод создает поток без указания метода запуска либо путем наследования от класса Thread, либо путем передачи объекта Runnable. Таким образом, это не приносит ничего, кроме траты времени.

FB.IS2_INCONSISTENT_SYNC

Доступ к полям этого класса осуществляется непоследовательно с точки зрения синхронизации. В этом отчете об ошибке указано, что детектор шаблонов ошибок определил, что

Класс содержит смесь заблокированных и разблокированных доступов,

Класс не аннотирован как javax.annotation.concurrent.NotThreadSafe.

По крайней мере один заблокированный доступ был выполнен одним из собственных методов класса, и

Количество несинхронизированных обращений к полям (чтение и запись) составляло не более трети всех обращений, при этом вес операций записи в два раза превышал вес операций чтения.

Типичная ошибка, соответствующая этому шаблону, — забывание синхронизировать один из методов в классе, который должен быть потокобезопасным.

Вы можете выбрать узлы с надписью «Несинхронизированный доступ», чтобы отобразить места кода, в которых детектор считает, что доступ к полю осуществлялся без синхронизации.

Обратите внимание, что в этом детекторе существуют различные источники погрешностей; например, детектор не может статически обнаружить все ситуации, в которых удерживается замок. Кроме того, даже если детектор точно различает заблокированный и незаблокированный доступ, рассматриваемый код все равно может быть правильным.

FB.IS_FIELD_NOT_GUARDED

Это поле помечено с помощью net.jcip.annotations.GuardedBy или javax.annotation.concurrent.GuardedBy, но доступ к нему может осуществляться таким образом, что это нарушает эти аннотации.

FB.JML_JSR166_CALLING_WAIT_RATHER_THAN_AWAIT

Этот метод вызывает wait(), notify() или notifyAll() для объекта, который также предоставляет методы await(), signal(), signalAll() (например, объекты util.concurrent Condition). Вероятно, это не то, что вам нужно, и даже если вы этого хотите, вам следует подумать об изменении вашего дизайна, поскольку другие разработчики найдут его исключительно запутанным.

FB.MSF_MUTABLE_SERVLET_FIELD

Веб-сервер обычно создает только один экземпляр сервлета или класса JSP (т. е. рассматривает класс как синглтон) и будет иметь несколько потоков, вызывающих методы в этом экземпляре для обслуживания нескольких одновременных запросов. Таким образом, наличие изменяемого поля экземпляра обычно создает условия гонки.

FB.NO_NOTIFY_NOT_NOTIFYALL

Этот метод вызывает notify(), а не notifyAll(). Мониторы Java часто используются в различных условиях. Вызов notify() пробуждает только один поток, а это означает, что пробуждённый поток может быть не тем, который ожидает условия, которое только что выполнил вызывающий объект.

FB.STCAL_STATIC_CALENDAR_INSTANCE

Несмотря на то, что JavaDoc не содержит никаких намеков на это, календари по своей сути небезопасны для многопоточного использования. Совместное использование одного экземпляра за границами потоков без надлежащей синхронизации приведет к нестабильному поведению приложения. В версии 1.4 проблемы возникают реже, чем в Java 5, где вы, вероятно, увидите случайные исключения ArrayIndexOutOfBoundsException

или IndexOutOfBoundsExceptions в sun.util.calendar.BaseCalendar.getCalendarDateFromFixedDate().

У вас также могут возникнуть проблемы с сериализацией.

Рекомендуется использовать поле экземпляра.

FB.VO_VOLATILE_INCREMENT

Этот код увеличивает/уменьшает изменяемое поле. Приращения/уменьшения изменчивых полей не являются атомарными. Если несколько потоков увеличивают/уменьшают поле одновременно, приращения/уменьшения могут быть потеряны.

FB.VO_VOLATILE_REFERENCE_TO_ARRAY

Это объявляет изменчивую ссылку на массив, что может быть не тем, что вам нужно. При использовании изменчивой ссылки на массив операции чтения и записи ссылки на массив считаются энергозависимыми, но элементы массива являются энергонезависимыми. Чтобы получить изменчивые элементы массива, вам нужно будет использовать один из классов атомарных массивов в java.util.concurrent (представленный в Java 5.0).

FB.WS_WRITEOBJECT_SYNC

Этот класс имеет метод writeObject(), который синхронизируется; однако ни один другой метод класса не синхронизируется.

FB.DM_BOXED_PRIMITIVE_TOSTRING

Упакованный примитив выделяется только для вызова toString(). Более эффективно использовать статическую форму toString, которая принимает примитивное значение.

FB.DM_FP_NUMBER_CTOR

Использование new Double(double) гарантированно всегда приводит к созданию нового объекта, тогда как Double.valueOf(double) позволяет кэшировать значения, выполняемые компилятором, библиотекой классов или JVM. Использование кэшированных значений позволяет избежать выделения объектов, и код будет работать быстрее.

Если класс не должен быть совместим с JVM, предшествующими Java 5, используйте либо автоупаковку, либо метод valueOf() при создании экземпляров Double и Float.

FB.ITA_INEFFICIENT_TO_ARRAY

Этот метод использует метод toArray() производного класса коллекции и передает аргумент массива прототипов нулевой длины. Более эффективно использовать myCollection.toArray(new Foo[myCollection.size()]). Если переданный массив достаточно велик для хранения всех элементов коллекции, он заполняется и возвращается напрямую. Это позволяет избежать необходимости создавать второй массив (путем отражения) для возврата в качестве результата.

FB.SIC_INNER_SHOULD_BE_STATIC_ANON

Этот класс является внутренним классом, но не использует встроенную ссылку на объект, который его создал. Эта ссылка увеличивает размер экземпляров класса и может сохранять ссылку на объект-создатель дольше, чем необходимо. Если возможно, класс следует превратить в статический внутренний класс. Поскольку анонимные внутренние классы нельзя пометить как статические, для этого потребуется рефакторинг внутреннего класса, чтобы он стал именованным внутренним классом.

FB.SIC_INNER_SHOULD_BE_STATIC_NEEDS_THIS

Этот класс является внутренним классом, но он не использует встроенную ссылку на объект, который его создал, за исключением случаев создания внутреннего объекта. Эта ссылка увеличивает размер экземпляров класса и может сохранять ссылку на объект-создатель дольше, чем необходимо. Если возможно, класс следует превратить в статический внутренний класс. Поскольку ссылка на внешний объект требуется во время создания внутреннего экземпляра, внутренний класс необходимо будет реорганизовать, чтобы передать ссылку на внешний экземпляр конструктору внутреннего класса.

FB.UM_UNNECESSARY_MATH

Этот метод использует статический метод из java.lang.Math для постоянного значения. Результат этого метода в этом случае может быть определен статически, и быстрее, а иногда и точнее, если просто использовать константу.

FB.DMI_CONSTANT_DB_PASSWORD

Этот код создает соединение с базой данных с использованием жестко запрограммированного постоянного пароля. Любой, у кого есть доступ к исходному или скомпилированному коду, может легко узнать пароль.

FB.DMI_EMPTY_DB_PASSWORD

Этот код создает подключение к базе данных с использованием пустого или пустого пароля. Это указывает на то, что база данных не защищена паролем.

FB.HRS_REQUEST_PARAMETER_TO_COOKIE

Этот код создает файл cookie HTTP, используя ненадежный параметр HTTP. Если этот файл cookie будет добавлен в ответ HTTP, это приведет к уязвимости разделения ответа HTTP.

FB.HRS_REQUEST_PARAMETER_TO_HTTP_HEADER

Этот код напрямую записывает параметр HTTP в заголовок HTTP, что допускает уязвимость разделения ответа HTTP.

FB.PT_ABSOLUTE_PATH_TRAVERSAL

Программное обеспечение использует параметр HTTP-запроса для создания пути, который должен находиться в ограниченном каталоге, но оно не нейтрализует должным образом последовательности абсолютных путей, такие как «/abs/path», которые могут разрешаться в местоположение, находящееся за пределами этого каталога.

FB.PT_RELATIVE_PATH_TRAVERSAL

Программное обеспечение использует параметр HTTP-запроса для создания пути, который должен находиться в ограниченном каталоге, но оно не нейтрализует должным образом последовательности, такие как «..», которые могут разрешаться в местоположение, находящееся за пределами этого каталога.

FB.SQL_NONCONSTANT_STRING_PASSED_TO_EXECUTE

Этот метод вызывает метод выполнения или addBatch для оператора SQL со строкой, которая создается динамически. Вместо этого рассмотрите возможность использования подготовленного оператора. Он более эффективен и менее уязвим для атак с использованием SQL-инъекций.

FB.SQL_PREPARED_STATEMENT_GENERATED_FROM_NONCONSTANT_STRING

Код создает подготовленный оператор SQL из непостоянной строки. Если при построении этой строки используются непроверенные испорченные данные пользователя, то можно использовать SQL-инъекцию, чтобы заставить подготовленный оператор сделать что-то неожиданное и нежелательное.

FB.XSS_REQUEST_PARAMETER_TO_JSP_WRITER

Этот код напрямую записывает параметр HTTP в выходные данные JSP, что допускает уязвимость межсайтового скриптинга.

FB.XSS_REQUEST_PARAMETER_TO_SEND_ERROR

Этот код напрямую записывает параметр HTTP на страницу ошибок сервера (используя HttpServletResponse.sendError). Повторение этих ненадежных входных данных допускает отражение уязвимости межсайтового скриптинга.

FB.XSS_REQUEST_PARAMETER_TO_SERVLET_WRITER

Этот код напрямую записывает параметр HTTP в выходные данные сервлета, что допускает отражение уязвимости межсайтового скриптинга.

FB.BC_UNCONFIRMED_CAST_OF_RETURN_VALUE

Этот код выполняет непроверяемое приведение возвращаемого значения метода. Код может вызывать метод таким образом, что приведение гарантированно будет безопасным, но SpotBugs не сможет проверить безопасность приведения. Убедитесь, что логика вашей программы гарантирует, что это приведение не завершится неудачно.

FB.CI_CONFUSED_INHERITANCE

Этот класс объявлен финальным, но объявляет поля, подлежащие защите. Поскольку класс является окончательным, его нельзя получить из него, а использование protected сбивает с толку. Модификатор доступа к полю должен быть изменен на частный или общедоступный, чтобы отображать истинное использование поля.

FB.DB_DUPLICATE_SWITCH_CLAUSES

Этот метод использует один и тот же код для реализации двух предложений оператора переключения. Это может быть случай дублирования кода, но это также может указывать на ошибку кодирования.

FB.MTIA_SUSPECT_SERVLET_INSTANCE_FIELD

Этот класс является наследником класса сервлета и использует переменную-член экземпляра. Поскольку в рамках J2EE создается только один экземпляр класса сервлета и используется в многопоточном режиме, эта парадигма крайне нежелательна и, скорее всего, проблематична. Рассмотрите возможность использования только локальных переменных метода.

FB.MTIA_SUSPECT_STRUTS_INSTANCE_FIELD

Этот класс является наследником класса действий Struts и использует переменную-член экземпляра. Поскольку в среде Struts создается только один экземпляр класса Action Struts и используется в многопоточном режиме, эта парадигма крайне не рекомендуется и, скорее всего, проблематична. Рассмотрите возможность использования только локальных переменных метода. Сообщаются только поля экземпляров, записанные вне монитора.

FB.NP_DEREFERENCE_OF_READLINE_VALUE

Результат вызова readLine() разыменовывается без проверки того, является ли результат нулевым. Если больше нет строк текста для чтения, readLine() вернет значение null и разыменование, которое сгенерирует исключение нулевого указателя.

FB.NP_IMMEDIATE_DEREFERENCE_OF_READLINE

Результат вызова readLine() немедленно разыменовывается. Если больше нет строк текста для чтения, readLine() вернет значение null и разыменование, которое сгенерирует исключение нулевого указателя.

FB.NP_LOAD_OF_KNOWN_NULL_VALUE

Известно, что переменная, на которую ссылаются в этот момент, имеет значение NULL, поскольку ранее была проведена проверка на значение NULL. Хотя это действительно так, это может быть ошибкой (возможно, вы намеревались сослаться на другую переменную, или, возможно, предыдущая проверка того, имеет ли переменная значение NULL, должна была быть проверкой того, не является ли она ненулевой).

FB.NP_NULL_ON_SOME_PATH_FROM_RETURN_VALUE

Возвращаемое значение метода разыменовывается без проверки на нулевое значение, а возвращаемое значение этого метода обычно должно проверяться на нулевое значение. Это может привести к исключению NullPointerException при выполнении кода.

FB.NP_NULL_ON_SOME_PATH_MIGHT_BE_INFEASIBLE

Существует ветвь инструкции, которая в случае выполнения гарантирует, что нулевое значение будет разыменовано, что приведет к генерации исключения NullPointerException при выполнении кода. Конечно, проблема может заключаться в том, что ветвь или оператор невозможны и что исключение нулевого указателя никогда не может быть выполнено; решить это за пределами возможностей SpotBugs. Поскольку это значение ранее проверялось на нулевое значение, такая вероятность вполне возможна.

FB.PZLA_PREFER_ZERO_LENGTH_ARRAYS

Зачастую лучше возвращать массив нулевой длины, чем нулевую ссылку, чтобы указать, что результатов нет (т. е. пустой список результатов). Таким образом, клиентам метода не требуется явная проверка на значение null.

С другой стороны, использование null для обозначения «нет ответа на этот вопрос», вероятно, уместно. Например, File.listFiles() возвращает пустой

список, если указан каталог, не содержащий файлов, и возвращает значение null, если файл не является каталогом.

FB.RCN_REDUNDANT_COMPARISON_OF_NULL_AND_NONNULL_VALUE

Этот метод сравнивает ссылку, которая, как известно, не равна NULL, с другой ссылку, которая, как известно, имеет значение NULL.

FB.RI_REDUNDANT_INTERFACES

Этот класс заявляет, что реализует интерфейс, который также реализуется суперклассом. Это избыточно, поскольку, как только суперкласс реализует интерфейс, все подклассы по умолчанию также реализуют этот интерфейс. Это может указывать на то, что иерархия наследования изменилась с момента создания этого класса, и следует уделить внимание владельцу реализации интерфейса.

FB.RV_CHECK_FOR_POSITIVE_INDEXOF

Метод вызывает String.indexOf и проверяет, является ли результат положительным или неположительным. Гораздо более типично проверить, является ли результат отрицательным или неотрицательным. Он положителен только в том случае, если проверяемая подстрока встречается не в начале строки.

FB.RV_REM_OF_HASHCODE

Этот код вычисляет хэш-код, а затем вычисляет остаток этого значения по модулю другого значения. Поскольку hashCode может быть отрицательным, результат операции остатка также может быть отрицательным.

Предполагая, что вы хотите гарантировать, что результат ваших вычислений неотрицательен, вам может потребоваться изменить свой код. Если вы знаете, что делитель представляет собой степень 2, вместо этого вы можете использовать побитовый оператор и (т. е. вместо использования x.hashCode()%n используйте x.hashCode()&(n-1)). Это, вероятно, быстрее, чем вычисление остатка. Если вы не знаете, что делитель представляет собой степень 2, возьмите абсолютное значение результата операции с остатком (т. е. используйте Math.abs(x.hashCode()%n)).

FB.RV_REM_OF_RANDOM_INT

Этот код генерирует случайное целое число со знаком, а затем вычисляет остаток этого значения по модулю другого значения. Поскольку случайное число может быть отрицательным, результат операции с остатком также может быть отрицательным. Убедитесь, что это так и было задумано, и

настоятельно рассмотрите возможность использования вместо этого метода Random.nextInt(int).

FB.RV_RETURN_VALUE_IGNORED_INFERRED

Этот код вызывает метод и игнорирует возвращаемое значение. Возвращаемое значение имеет тот же тип, что и тип, для которого вызывается метод, и наш анализ показывает, что возвращаемое значение может быть важным (например, как игнорирование возвращаемого значения String.toLowerCase()).

На основании простого анализа тела метода мы предполагаем, что игнорирование возвращаемого значения может быть плохой идеей. Вы можете использовать аннотацию @CheckReturnValue, чтобы указать SpotBugs, является ли игнорирование возвращаемого значения этого метода важным или приемлемым.

Пожалуйста, изучите это внимательно, чтобы решить, можно ли игнорировать возвращаемое значение.

FB.SE_TRANSIENT_FIELD_OF_NONSERIALIZABLE_CLASS

Поле помечено как временное, но класс не является сериализуемым, поэтому пометка его как временного не имеет абсолютно никакого эффекта. Это может быть остаток маркировки от предыдущей версии кода, в которой класс был сериализуемым, или это может указывать на непонимание того, как работает сериализация.

FB.TQ_EXPLICIT_UNKNOWN_SOURCE_VALUE_REACHES_ALWAYS_SINK

Значение используется таким образом, что оно всегда должно быть значением, обозначенным квалификатором типа, но существует явная аннотация, указывающая, что неизвестно, где значение должно иметь этот квалификатор типа. Либо использование, либо аннотация неверны.

FB.TQ_EXPLICIT_UNKNOWN_SOURCE_VALUE_REACHES_NEVER_SINK

Значение используется таким образом, что оно никогда не должно быть значением, обозначаемым квалификатором типа, но существует явная аннотация, указывающая, что неизвестно, где этому значению запрещено иметь этот квалификатор типа. Либо использование, либо аннотация неверны.

FB.UWF_FIELD_NOT_INITIALIZED_IN_CONSTRUCTOR

Это поле никогда не инициализируется ни в одном конструкторе и поэтому может иметь значение null после создания объекта. В другом месте

он загружается и разыменовывается без проверки на ноль. Это может быть либо ошибка, либо сомнительный дизайн, поскольку это означает, что будет сгенерировано исключение нулевого указателя, если это поле будет разыменовано перед инициализацией.

FB.XFB_XML_FACTORY_BYPASS

Этот метод выделяет конкретную реализацию интерфейса XML. Предпочтительно использовать предоставленные фабричные классы для создания этих объектов, чтобы можно было изменить реализацию во время выполнения.

FB.UC_USELESS_VOID_METHOD

Наш анализ показывает, что этот метод непустой пустоты на самом деле не выполняет никакой полезной работы. Пожалуйста, проверьте его: возможно, в его коде ошибка или его тело можно полностью удалить.

Мы стараемся максимально сократить количество ложных срабатываний, но в некоторых случаях это предупреждение может быть ошибочным. К частым ложноположительным случаям относятся:

Этот метод предназначен для запуска загрузки некоторого класса, что может иметь побочный эффект.

Этот метод предназначен для неявной генерации какого-то непонятного исключения.

FB.BAC_BAD_APPLET_CONSTRUCTOR

Этот конструктор вызывает методы родительского апплета, которые полагаются на AppletStub. Поскольку AppletStub не инициализируется до тех пор, пока не будет вызван метод init() этого апплета, эти методы не будут работать корректно.

FB.BC_IMPOSSIBLE_CAST_PRIMITIVE_ARRAY

Это приведение всегда будет вызывать исключение ClassCastException.

FB.CAA_COVARIANT_ARRAY_ELEMENT_STORE

Значение сохраняется в массиве, а тип значения не соответствует типу массива. Из анализа известно, что фактический тип массива уже объявленного типа его переменной или поля и это присвоение не удовлетворяет исходному типу массива. Это назначение может вызвать исключение ArrayStoreException во время выполнения.

FB.CAA_COVARIANT_ARRAY_FIELD

Полю присваивается массив ковариантного типа. Это сбивает с толку и может привести к ArrayStoreException во время выполнения, если ссылка какого-либо другого типа будет сохранена в этом массиве позже. Рассмотрите возможность изменения типа создаваемого массива или типа поля.

FB.CAA_COVARIANT_ARRAY_LOCAL

Локальной переменной присваивается массив ковариантного типа. Это сбивает с толку и может привести к ArrayStoreException во время выполнения, если ссылка какого-либо другого типа будет сохранена в этом массиве позже. Рассмотрите возможность изменения типа создаваемого массива или типа локальной переменной.

FB.CAA_COVARIANT_ARRAY_RETURN

Из метода возвращается массив ковариантного типа. Это сбивает с толку и может привести к ArrayStoreException во время выполнения, если вызывающий код попытается сохранить ссылку какого-либо другого типа в возвращаемом массиве. Рассмотрите возможность изменения типа создаваемого массива или типа возвращаемого значения метода.

FB.CD_CIRCULAR_DEPENDENCY

Этот класс имеет циклическую зависимость от других классов. Это затрудняет создание этих классов, поскольку правильность построения каждого зависит от другого. Рассмотрите возможность использования интерфейсов, чтобы разорвать жесткую зависимость.

FB.CO_COMPARETO_INCORRECT_FLOATING

Этот метод сравнивает значения типа double или float, используя такой шаблон: val1 > val2 ? 1: значение1 <значение2? -1 : 0. Этот шаблон работает некорректно для значений -0.0 и NaN, что может привести к неправильному результату сортировки или нарушению сбора (если в качестве ключей используются сравниваемые значения). Рассмотрите возможность использования статических методов Double.compare или Float.compare, которые правильно обрабатывают все особые случаи.

FB.DC_PARTIALLY_CONSTRUCTED

Похоже, этот метод использует ленивую инициализацию поля с двойной проверкой блокировки. Хотя поле правильно объявлено как изменчивое, возможно, что внутренняя структура объекта изменится после назначения поля, поэтому другой поток может увидеть частично инициализированный объект.

Чтобы решить эту проблему, рассмотрите возможность сначала сохранить объект в локальной переменной и сохранить его в изменчивом поле только после того, как он будет полностью создан.

FB.DLS_DEAD_LOCAL_INCREMENT_IN_RETURN

Этот оператор имеет возврат, например return x++; / вернуть x--;. Постфиксное увеличение/уменьшение не влияет на значение выражения, поэтому это увеличение/уменьшение не имеет никакого эффекта. Пожалуйста, убедитесь, что это утверждение соответствует действительности.

FB.DM_BOXED_PRIMITIVE_FOR_COMPARE

Упакованный примитив создается только для вызова метода CompareTo(). Более эффективно использовать метод статического сравнения (для двойных значений и чисел с плавающей запятой, начиная с Java 1.4, для других примитивных типов, начиная с Java 7), который работает непосредственно с примитивами.

FB.DM_INVALID_MIN_MAX

Этот код пытается ограничить границы значений, используя такую конструкцию, как Math.min(0, Math.max(100, value)). Однако порядок констант неправильный: он должен быть Math.min(100, Math.max(0, значение)). В результате этот код всегда выдает один и тот же результат (или NaN, если значение равно NaN).

FB.DMI_UNSUPPORTED_METHOD

Все цели вызова этого метода вызывают исключение UnsupportedOperationException.

FB.FL_MATH_USING_FLOAT_PRECISION

Метод выполняет математические операции, используя точность с плавающей запятой. Точность с плавающей запятой очень неточна. Например, 16777216.0f + 1.0f = 16777216.0f. Вместо этого рассмотрите возможность использования двойной математики.

FB.IIL_ELEMENTS_GET_LENGTH_IN_LOOP

Метод вызывает NodeList.getLength() внутри цикла, и NodeList создается вызовом getElementsByTagName. Этот NodeList не хранит свою длину, а вычисляет ее каждый раз не самым оптимальным образом. Рассмотрите возможность сохранения длины переменной перед циклом.

FB.IIL_PATTERN_COMPILE_IN_LOOP_INDIRECT

Метод создает одно и то же регулярное выражение внутри цикла, поэтому оно будет компилироваться на каждой итерации. Было бы более оптимально предварительно скомпилировать это регулярное выражение, используя Pattern.compile вне цикла.

FB.IIL_PATTERN_COMPILE_IN_LOOP

Метод вызывает Pattern.compile внутри цикла, передавая константные аргументы. Если шаблон необходимо использовать несколько раз, нет смысла компилировать его для каждой итерации цикла. Переместите этот вызов за пределы цикла или даже в статическое конечное поле.

FB.IIL_PREPARE_STATEMENT_IN_LOOP

Метод вызывает Connection.prepareStatement внутри цикла, передавая постоянные аргументы. Если подготовленный оператор должен выполняться несколько раз, нет смысла создавать его заново для каждой итерации цикла. Переместите этот вызов за пределы цикла.

FB.IMA_INEFFICIENT_MEMBER_ACCESS

Этот метод внутреннего класса считывает или записывает в частную переменную-член владеющего класса или вызывает закрытый метод владеющего класса. Компилятор должен создать специальный метод для доступа к этому частному члену, что делает его менее эффективным. Ослабление защиты переменной-члена или метода позволит компилятору рассматривать это как обычный доступ.

FB.IS_INCONSISTENT_SYNC

Доступ к полям этого класса осуществляется непоследовательно с точки зрения синхронизации. Класс содержит смесь заблокированных и разблокированных доступов. По крайней мере один заблокированный доступ был выполнен одним из собственных методов класса. Количество несинхронизированных обращений к полям (чтение и запись) составило не более одной трети всех обращений, при этом операции записи имеют вдвое больший вес, чем операции чтения. Типичная ошибка, соответствующая этому шаблону ошибок, — забывание синхронизации. Обратите внимание, что в этом детекторе существуют различные источники погрешностей; например, детектор не может статически обнаруживать все ситуации, в которых замок удерживается. Кроме того, даже если детектор точен в различая заблокированный и разблокированный доступ, рассматриваемый код все равно может будь прав.

FB.ME_ENUM_FIELD_SETTER

Этот общедоступный метод, объявленный в общедоступном перечислении, безоговорочно устанавливает поле перечисления, поэтому это

поле может быть изменено вредоносным кодом или случайно из другого пакета. Хотя изменяемые поля перечислений могут использоваться для ленивой инициализации, открывать их для внешнего мира — плохая практика. Рассмотрите возможность удаления этого метода или объявления его частным для пакета.

FB.ME_MUTABLE_ENUM_FIELD

Изменяемое общедоступное поле определяется внутри общедоступного перечисления, поэтому оно может быть изменено вредоносным кодом или случайно из другого пакета. Хотя изменяемые поля перечислений могут использоваться для ленивой инициализации, открывать их для внешнего мира — плохая практика. Рассмотрите возможность объявления этого поля окончательным и/или частным для пакета.

FB.MS_MUTABLE_COLLECTION

Экземпляр изменяемой коллекции присваивается конечному статическому полю, поэтому он может быть изменен вредоносным кодом или случайно из другого пакета. Рассмотрите возможность переноса этого поля в Collections.unmodifyingSet/List/Map/etc. чтобы избежать этой уязвимости.

FB.MS_MUTABLE_COLLECTION_PKGPROTECT

Экземпляр изменяемой коллекции присваивается конечному статическому полю, поэтому он может быть изменен вредоносным кодом или случайно из другого пакета. Поле можно сделать защищенным пакетом, чтобы избежать этой уязвимости. В качестве альтернативы вы можете обернуть это поле в Collections.unmodifyingSet/List/Map/etc. чтобы избежать этой уязвимости.

FB.NP_METHOD_PARAMETER_RELAXING_ANNOTATION

Метод всегда должен реализовывать контракт метода, который он переопределяет. Таким образом, если метод принимает параметр который помечен как @Nullable, вам не следует переопределять этот метод в подклассе методом, где этот параметр равен @Nonnull. Это нарушает контракт, согласно которому метод должен обрабатывать нулевой параметр.

FB.PS_PUBLIC_SEMAPHORES

Этот класс использует синхронизацию вместе с функциями wait(), notify() или notifyAll() для себя (ссылка на this). Клиентские классы, использующие этот класс, могут, кроме того, использовать экземпляр этого класса в качестве объекта синхронизации. Поскольку два класса используют один и тот же объект для синхронизации, корректность многопоточности сомнительна. Не следует синхронизировать или вызывать методы семафора

по общедоступной ссылке. Рассмотрите возможность использования внутренней переменной-члена для управления синхронизацией.

FB.RANGE_ARRAY_INDEX

Операция с массивом выполняется, но индекс массива выходит за пределы, что приведет к исключению ArrayIndexOutOfBoundsException во время выполнения.

FB.RANGE_ARRAY_LENGTH

Метод вызывается с параметром массива и параметром длины, но длина выходит за пределы. Это приведет к исключению IndexOutOfBoundsException во время выполнения.

FB.RANGE_ARRAY_OFFSET

Метод вызывается с параметром массива и параметром смещения, но смещение выходит за пределы. Это приведет к исключению IndexOutOfBoundsException во время выполнения.

FB.RANGE_STRING_INDEX

Вызывается строковый метод, и указанный индекс строки выходит за пределы. Это приведет к исключению StringIndexOutOfBoundsException во время выполнения.

FB.UC_USELESS_CONDITION

Это условие всегда дает тот же результат, что и значение задействованной переменной, которое было сужено ранее. Вероятно имелось в виду что-то другое или условие можно убрать.

FB.UC_USELESS_CONDITION_TYPE

Это условие всегда дает один и тот же результат из-за диапазона типов задействованной переменной. Вероятно имелось в виду что-то другое или условие можно убрать.

FB.UC_USELESS_OBJECT

Наш анализ показывает, что этот объект бесполезен. Он создается и модифицируется, но его значение никогда не выходит за пределы метода и не приводит к какому-либо побочному эффекту. Либо произошла ошибка и объект предназначен для использования, либо его можно удалить.

Этот анализ редко дает ложноположительные результаты. К частым ложноположительным случаям относятся: Этот объект используется для неявной генерации какого-то непонятного исключения. Этот объект

используется как заглушка для обобщения кода. Этот объект использовался для хранения сильных ссылок на объекты со слабыми/мягкими ссылками.

FB.UC_USELESS_OBJECT_STACK

Этот объект создан только для выполнения некоторых модификаций, не имеющих побочных эффектов. Вероятно имелось в виду что-то другое или объект можно удалить.

FB.USM_USELESS_ABSTRACT_METHOD

Этот абстрактный метод уже определен в интерфейсе, реализованном этим абстрактным классом. Этот метод можно удалить, поскольку он не несет никакой дополнительной ценности.

FB.USM_USELESS_SUBCLASS_METHOD

Этот производный метод просто вызывает тот же метод суперкласса, передавая точные полученные параметры. Этот метод можно удалить, поскольку он не несет никакой дополнительной ценности.

FB.VR_UNRESOLVABLE_REFERENCE

Этот класс ссылается на класс или метод, который невозможно разрешить с помощью библиотек, с которыми он анализируется.

USER.BAD_KEYWORD_DEMO

Демонстрационный детектор, который указывает на возможные проблемы, связанные с использованием пользователем ключевых слов (if, try, break, …)

USER.BAD_STRING_LIGHT_DEMO

Демонстрационный (light demo) детектор, который указывает на возможный ввод пользователем строк, которые могут уязвимыми для работы программы

FB.BIT_SIGNED_CHECK_HIGH_BIT

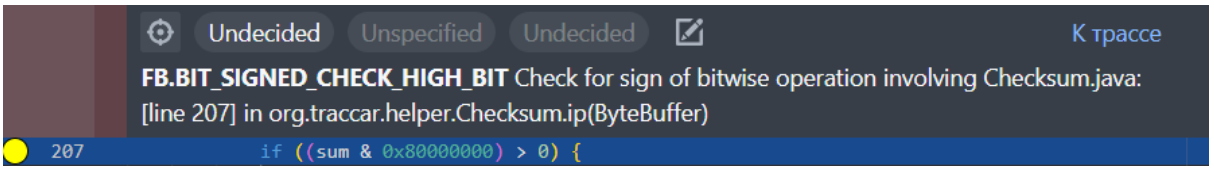


Рисунок 52 –Детектор FB.BIT_SIGNED_CHECK_HIGH_BIT в интерфейсе Svacer

Ошибка "Check for sign of bitwise operation" (Проверьте знак битовой операции) указывает на то, что в вашем коде выполняется битовая операция (например, &, |, ^, ~), и компилятор предупреждает вас о том, что это может привести к неожиданным результатам, если вы применяете ее к знаковым типам данных (типам данных, представляющим значения с учетом знака).

Битовые операции часто используются с целочисленными типами данных. Если вы применяете их к знаковым числам, это может привести к проблемам, связанным с представлением отрицательных чисел в двоичной системе.

FB.EI_EXPOSE_REP

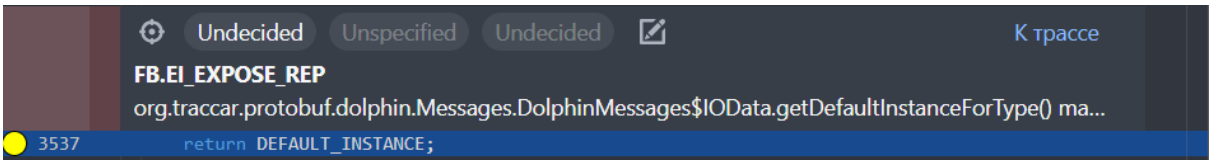


Рисунок 53 –Детектор FB.EI_EXPOSE_REP в интерфейсе Svacer

Ошибка "... may expose internal representation by returning ..." (или подобные предупреждения) указывает на то, что ваш код возвращает объект или структуру данных, которая может раскрывать внутреннее представление (internal representation) своих данных. Это может создать проблемы в том случае, если другие части программы зависят от внутреннего представления и изменения внутреннего представления могут повлиять на их корректность.

FB.EI_EXPOSE_REP2

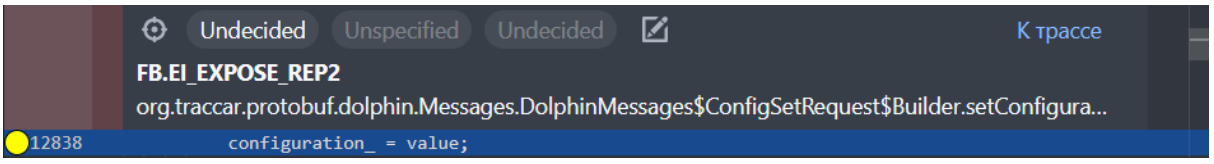


Рисунок 54 –Детектор FB.EI_EXPOSE_REP2 в интерфейсе Svacer

Ошибка "... may expose internal representation by storing an externally mutable object into ..." (или подобные предупреждения) указывает на то, что в вашем коде объект или структура данных хранит ссылку на объект, который может быть изменен извне. Это может создать проблемы, поскольку изменения внешнего объекта могут негативно повлиять на внутреннее состояние вашего объекта или на его корректное функционирование.

FB.LDAP_INJECTION

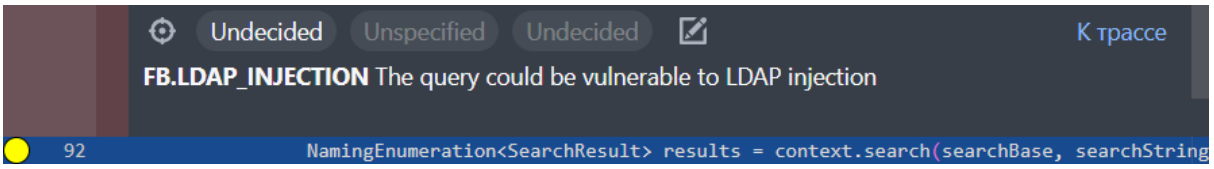


Рисунок 55 –Детектор FB.LDAP_INJECTION в интерфейсе Svacer

Ошибка "The query could be vulnerable to LDAP injection" указывает на потенциальную уязвимость вашего кода к атаке LDAP-инъекции. Это означает, что ваш запрос к LDAP-серверу формируется таким образом, что злоумышленник может внедрить вредоносный код или изменить его выполнение, используя специально сформированные строки.

LDAP (Lightweight Directory Access Protocol) инъекция аналогична SQL-инъекции, но применяется к LDAP-запросам. Злоумышленник может внедрять команды LDAP в ваш запрос, чтобы получить несанкционированный доступ к данным, выполнить поддельные запросы или воздействовать на работу LDAP-сервера.

FB.MS_EXPOSE_REP

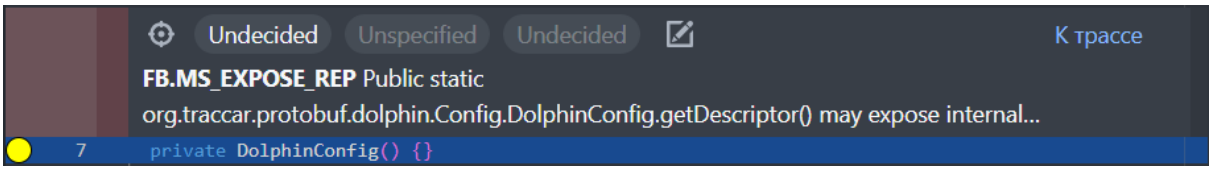


Рисунок 56 –Детектор FB.MS_EXPOSE_REP в интерфейсе Svacer

Ошибка "Public ... may expose internal representation by returning..." (или подобные предупреждения) указывает на то, что публичный метод вашего класса возвращает объект или структуру данных, которые представляют внутреннее состояние объекта. Это может быть потенциальной проблемой, так как внешний код может получить доступ к и изменять внутреннее состояние объекта напрямую, что нарушает инкапсуляцию и может привести к неожиданным побочным эффектам.

FB.NM_CLASS_NAMING_CONVENTION

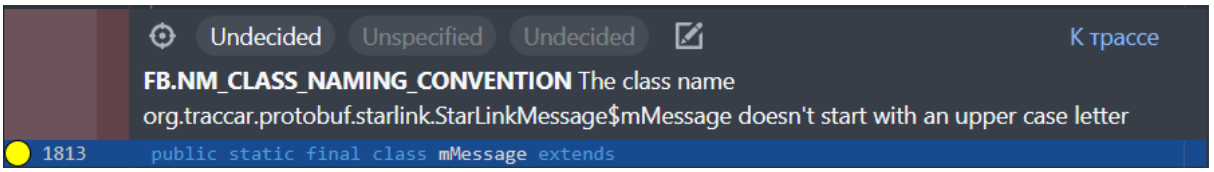


Рисунок 57 –Детектор FB.NM_CLASS_NAMING_CONVENTION в интерфейсе Svacer

Ошибка "The class name ... doesn't start with an upper case letter" указывает на то, что в вашем коде имя класса начинается с символа в нижнем регистре, что не соответствует общепринятым соглашениям о наименовании классов во многих языках программирования.

Обычно в большинстве языков программирования принято именовать классы с использованием "CamelCase" (также известным как "PascalCase"), где каждое новое слово в имени класса начинается с заглавной буквы. Это делается для лучшей читаемости кода и соответствия стандартам стиля.

FB.NM_SAME_SIMPLE_NAME_AS_INTERFACE

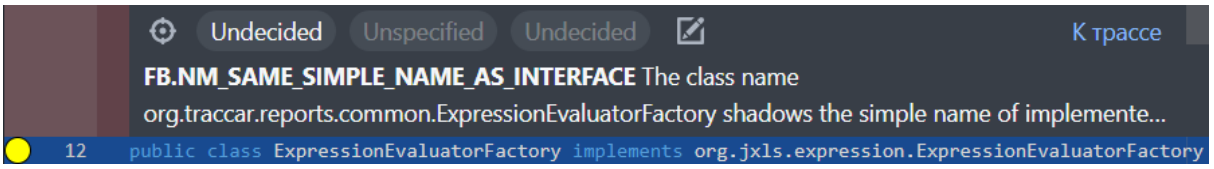


Рисунок 58 –Детектор FB.NM_SAME_SIMPLE_NAME_AS_INTERFACE в интерфейсе Svacer

Ошибка "The class name ... shadows the simple name of implemented interface" указывает на то, что в вашем коде имя класса перекрывает (shadowing) простое имя реализованного интерфейса. Это может быть причиной путаницы и может вызвать проблемы при компиляции или поддержке кода.

FB.NS_NON_SHORT_CIRCUIT

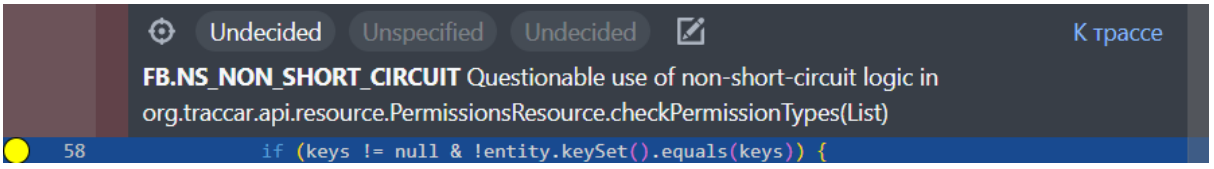


Рисунок 59 –Детектор FB.NS_NON_SHORT_CIRCUIT в интерфейсе Svacer

Ошибка "Questionable use of non-short-circuit logic" указывает на потенциально проблемное использование логических операторов, которые не являются короткозамкнутыми (non-short-circuit logic) в контексте языков программирования, где используется короткозамкнутая логика.

В короткозамкнутой логике выражение останавливается вычисляться, как только результат может быть определен. Например, в выражении A && B,

если A является ложным, B не будет вычислено, так как результат всего выражения уже известен (ложь). То же самое относится к выражению A || B - если A является истиной, B не будет вычислено, так как результат всего выражения уже известен (истина).

Если в вашем коде используется логика, которая не является короткозамкнутой, это может привести к ненужным вычислениям и неэффективному использованию ресурсов.

FB.ODR_OPEN_DATABASE_RESOURCE

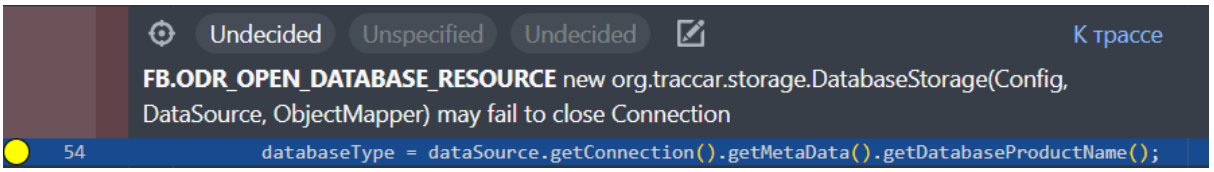


Рисунок 60 –Детектор FB.ODR_OPEN_DATABASE_RESOURCE в интерфейсе Svacer

Ошибка "new ... may fail to close Connection" указывает на потенциальную проблему с управлением ресурсами, такими как соединение с базой данных. Это предупреждение обычно возникает, когда создается объект, который использует ресурс, требующий явного закрытия, но в коде отсутствует соответствующий вызов метода закрытия (close) или освобождения ресурсов.

FB.PATH_TRAVERSAL_IN

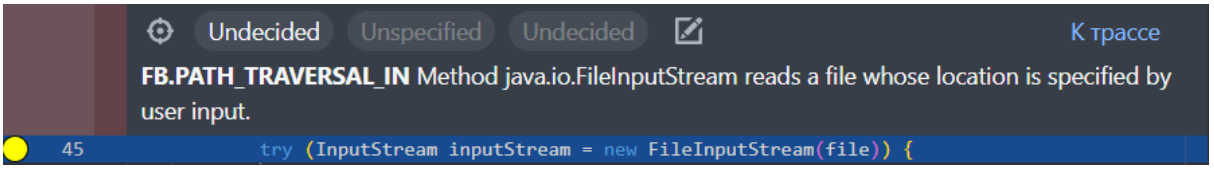


Рисунок 61 –Детектор FB.PATH_TRAVERSAL_IN в интерфейсе Svacer

Ошибка "Method reads a file whose location is specified by user input" указывает на потенциальную уязвимость в вашем коде, связанную с чтением файла, путь к которому указывается пользовательским вводом. Это может стать источником опасности, таких как чтение конфиденциальных файлов, выполнение нежелательных операций или атаки вида "Path Traversal" (перемещение по пути).

FB.PATH_TRAVERSAL_OUT

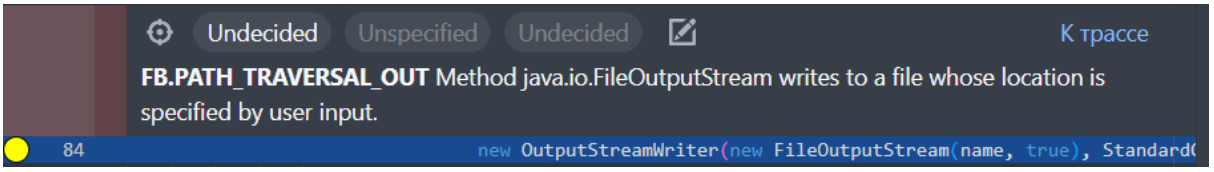


Рисунок 62 –Детектор FB.PATH_TRAVERSAL_OUT в интерфейсе Svacer

Ошибка "Method writes to a file whose location is specified by user input" указывает на потенциальную уязвимость в вашем коде, связанную с записью в файл, путь к которому указывается пользовательским вводом. Это может стать источником опасности, таких как запись в конфиденциальные файлы, выполнение нежелательных операций или атаки вида "Path Traversal" (перемещение по пути).

FB.RV_RETURN_VALUE_IGNORED_NO_SIDE_EFFECT

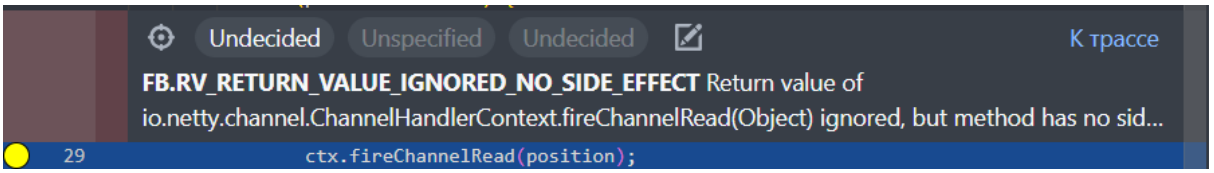


Рисунок 63 –Детектор FB.RV_RETURN_VALUE_IGNORED_NO_SIDE_EFFECT в интерфейсе Svacer

Ошибка "Return value of ... ignored, but method has no side effect" указывает на то, что в вашем коде вызывается метод, который возвращает значение, но это возвращаемое значение не используется, и метод не имеет побочных эффектов. Это может быть индикатором либо ненужного вызова метода, либо упущенной логики, если метод предназначен для выполнения каких-то действий (побочных эффектов).

FB.SF_SWITCH_NO_DEFAULT

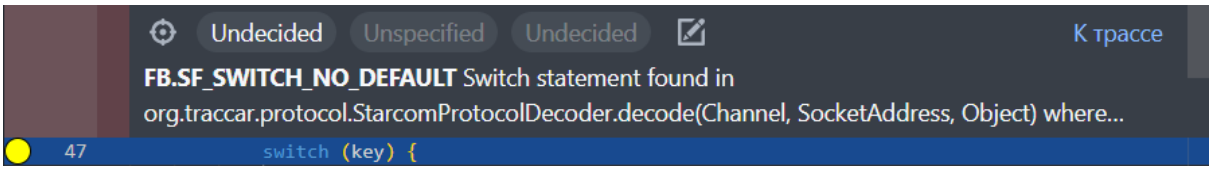


Рисунок 64 –Детектор FB.SF_SWITCH_NO_DEFAULT в интерфейсе Svacer

Ошибка "Switch statement found, where default case is missing" указывает на то, что в вашем операторе switch не предусмотрен блок default. В языках программирования, поддерживающих оператор switch, default представляет собой необязательный блок, который выполняется, если ни один из case не соответствует проверяемому значению.

FB.SKIPPED_CLASS_TOO_BIG

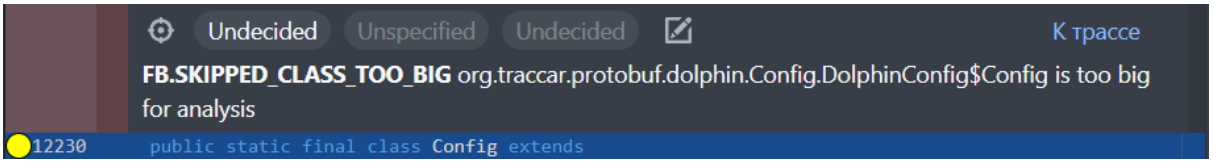


Рисунок 65 –Детектор FB.SKIPPED_CLASS_TOO_BIG в интерфейсе Svacer

Ошибка "object is too big for analysis" указывает на то, что объект, который пытается быть проанализирован в процессе выполнения программы, является слишком большим для обработки. Это может произойти, например, при попытке провести анализ объекта слишком большого размера во время выполнения или при использовании инструментов статического анализа кода, которые сталкиваются с очень большими структурами данных.

Причины такой ошибки могут варьироваться в зависимости от контекста и инструмента анализа, который вы используете. В некоторых случаях это может быть вызвано неэффективным использованием ресурсов, а в других случаях просто ограничением на размер объектов, которые могут быть обработаны.

FB.STATIC_IV

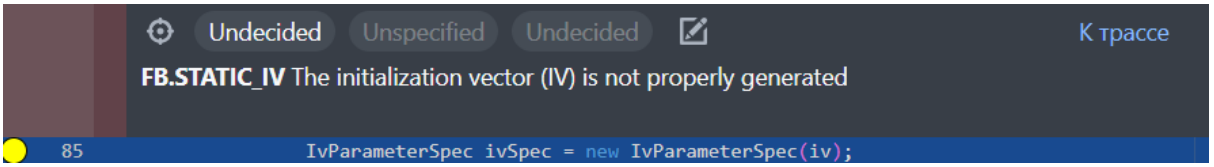


Рисунок 66 –Детектор FB.STATIC_IV в интерфейсе Svacer

Ошибка "The initialization vector (IV) is not properly generated" указывает на проблемы с созданием правильного вектора инициализации (IV) в криптографических операциях, чаще всего связанных с использованием алгоритмов симметричного шифрования, таких как AES (Advanced Encryption Standard).

В симметричном шифровании ключ используется как для шифрования, так и для дешифрования данных. Вместе с ключом часто используется вектор инициализации (IV), который добавляется к ключу перед шифрованием. Неправильная генерация IV может привести к уязвимостям в системе шифрования.

FB.UNENCRYPTED_SOCKET

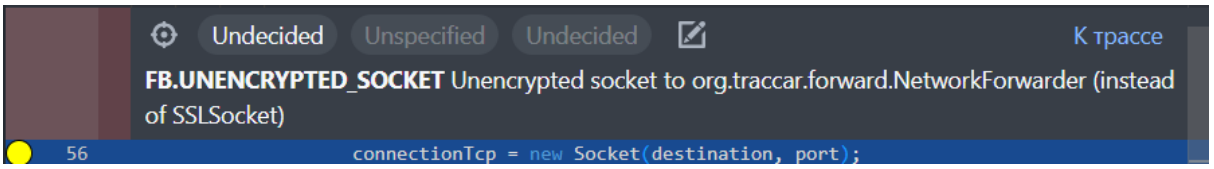


Рисунок 67 –Детектор FB.UNENCRYPTED_SOCKET в интерфейсе Svacer

Ошибка "Unencrypted socket to ..." указывает на использование незашифрованного сокета для обмена данными с указанным местом (например, сервером, хостом). Это может представлять серьезную уязвимость с точки зрения безопасности, поскольку данные, передаваемые по незашифрованному каналу связи, могут быть подвергнуты перехвату и прочтению злоумышленниками.

Проблема также может возникнуть, если вы пытаетесь установить незашифрованное соединение с сервером, который ожидает зашифрованные подключения (например, сервер HTTPS, который ожидает зашифрованные запросы).

FB.XPATH_INJECTION

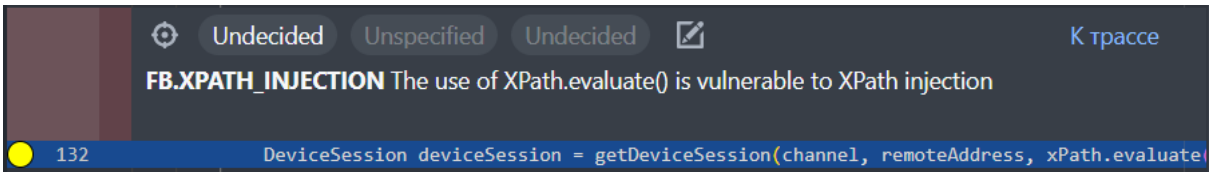


Рисунок 68 –Детектор FB.XPATH_INJECTION в интерфейсе Svacer

Ошибка "The use of XPath.evaluate() is vulnerable to XPath injection" указывает на потенциальную уязвимость в вашем коде, связанную с использованием метода XPath.evaluate() при обработке пользовательского ввода, что может привести к атаке XPath Injection.

XPath (XML Path Language) используется для навигации и запросов в XML-документах. Атака XPath Injection возникает, когда злоумышленник вводит данные в поле запроса (например, форма на веб-сайте), и эти данные непосредственно внедряются в XPath-запрос, что может привести к несанкционированному доступу к данным или выполнению нежелательных действий.

NO_CATCH.STRICT

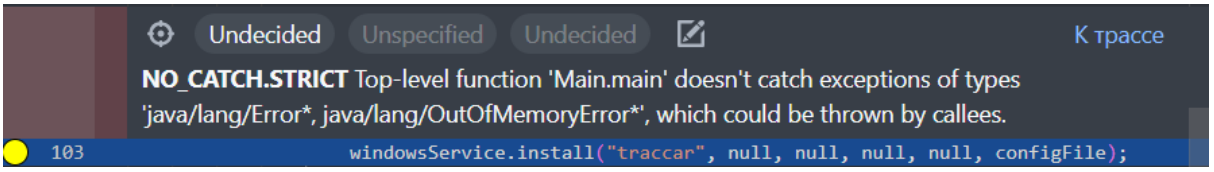


Рисунок 69 –Детектор NO_CATCH.STRICT в интерфейсе Svacer

Ошибка "Top-level function 'Main.main' doesn't catch exceptions of types 'java/lang/Error*, java/lang/OutOfMemoryError*', which could be thrown by callees" указывает на то, что в вашем коде отсутствует обработка исключений типа java/lang/Error и java/lang/OutOfMemoryError. Эти ошибки представляют собой серьезные проблемы, которые могут возникнуть в процессе выполнения программы и указывают на фундаментальные проблемы среды выполнения.

Класс java/lang/Error является базовым классом для всех ошибок времени выполнения, которые обычно связаны с проблемами виртуальной машины Java (JVM) или среды выполнения. java/lang/OutOfMemoryError возникает, когда JVM не может выделить достаточное количество памяти для выполнения операции.

PROC_USE.VULNERABLE

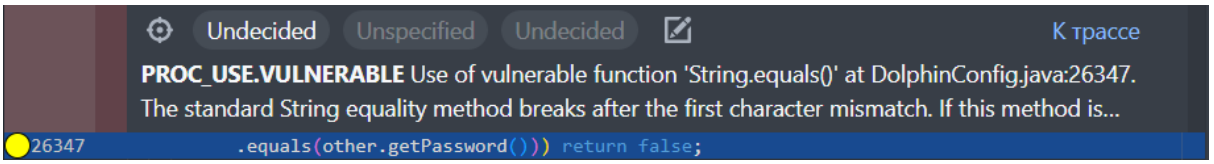


Рисунок 70 –Детектор PROC_USE.VULNERABLE в интерфейсе Svacer

Ошибка "Use of vulnerable function 'String.equals()'. The standard String equality method breaks after the first character mismatch. If this method is used to work with sensitive data, it can lead to timing attacks" указывает на потенциальную уязвимость, связанную с использованием метода String.equals() для сравнения строк, особенно если эти строки содержат чувствительные данные.

Метод String.equals() в Java сравнивает символы строк поочередно от первого до последнего и возвращает false при обнаружении первого несовпадения. Однако, если строки разной длины, сравнение может быть быстрее, если первые символы совпадают.

Это поведение может быть использовано для атак типа "timing attack". Timing attack может произойти, когда злоумышленник измеряет время, затраченное на выполнение операции сравнения строк и на основе этого времени пытается вывести символ за символом содержимое строки. Такие атаки могут быть особенно серьезными, если строки содержат чувствительные данные, такие как пароли.

7.1.5. UNDEFINED ошибки

Ошибки, класс которых не определен, а потому к ним нужно относиться с повышенным вниманием. (Рисунок 70)

Рисунок 71 – Обозначение UNDEFINED ошибок в интерфейсе Svacer

VN_FUTURE_DEREF_OF_NULL

Этот детектор указывает на потенциальное разыменование NULL - указателя в будущем

VN_FUTURE_DEREF_AFTER_NULL

Этот детектор указывает на потенциальное разыменование указателя с ранее обозначенным NULL - значением

SENSITIVE_LEAK

Указывает на потенциальный утечки конфиденциальных данных в программе. Этот детектор обычно обнаруживает участки кода, где чувствительная информация, такая как пароли, персональные данные пользователя, ключи шифрования и другие конфиденциальные данные, могут быть неадекватно обработаны или переданы без должных мер защиты.

SENSITIVE_LEAK.SYSTEM

Указывает на утечки конфиденциальных данных, связанных с системными ресурсами или нарушениями безопасности операционной системы. Такой детектор обычно предназначен для выявления мест в коде, где происходит некорректная работа с системными данными, что может привести к утечкам информации и рискам безопасности.

HARDCODED_PASSWORD

Этот детектор указывает на использование встроенных в программу фиксированных паролей, что небезопасно

UNREACHABLE_CODE.EX

Этот детектор указывает на наличие кода, который не выполняется ни при каких условиях работы программы

INVARIANT_RESULT.BIT_SHIFT

Этот детектор указывает на наличие выражения, которое при любых входных данных принимает одно и то же значение, что ассоциировано с битового сдвига в операциях над переменными

INFO.SVACE_EXCEPTION

Предоставляет информацию о специфических исключениях или ошибках, выявленных в процессе анализа кода инструментом Svace.

Для предложенного тестового материала на Java ошибки класса undefined отсутствуют.

8. Список уязвимостей по классификации ФСТЭК

Уязвимости прикладного программного обеспечения (далее ППО) согласно федеральной службе по техническому и экспортному контролю (далее ФСТЭК) могут представлять собой:

- функции и процедуры, относящиеся к разным прикладным программам и несовместимые между собой (не функционирующие в одной операционной среде) из-за конфликтов, связанных с распределением ресурсов системы;
- функции, процедуры, изменение определенным образом параметров которых позволяет использовать их для проникновения в операционную среду ИСПДн и вызова штатных функций операционной системы, выполнения несанкционированного доступа без обнаружения таких изменений операционной системой;
- **фрагменты кода программ ("дыры", "люки"),** введенные разработчиком, позволяющие обходить процедуры идентификации, аутентификации, проверки целостности и др., предусмотренные в операционной системе;
- **отсутствие необходимых средств защиты** (аутентификации, проверки целостности, проверки форматов сообщений, блокирования несанкционированно модифицированных функций и т.п.);
- **ошибки в программах** (в объявлении переменных, функций и процедур, в кодах программ), **которые при определенных условиях** (например, при выполнении логических переходов) **приводят к сбоям**, в том числе к сбоям функционирования средств и систем защиты информации, к возможности несанкционированного доступа к информации.

9. Интерпретация обнаруженных детекторов как уязвимостей согласно ФСТЭК

Используя характеристику уязвимостей ППО ФСТЭК, согласно пункту 8.2 «Характеристика уязвимостей прикладного программного обеспечения» приложения 1 приказа ФНС России от 21 декабря 2011 г. N ММВ-7-4/959@, возможно идентифицировать дефекты, обнаруживаемые Svace, как уязвимости и недекларированные возможности. То есть соотнести список обнаруживаемых детекторов с списком уязвимостей ППО. Такой анализ позволит улучшить качество проводимой разметки и в будущем создать практикум по автоматизированному поиску уязвимостей.

Таблица 3 - Интерпретация обнаруженных детекторов как уязвимостей согласно ФСТЭК

	функции и процедуры, относящиеся к разным прикладным программам и несовместимые между собой (не функционирующие в одной операционной среде) из-за конфликтов, связанных с распределением ресурсов системы	**функции, процедуры, изменение определенным образом параметров которых позволяет использовать их для проникновения в операционную среду ИСПДн и вызова штатных функций операционной системы, выполнения несанкционированного доступа без обнаружения таких изменений операционной системой;**	**фрагменты кода программ ("дыры", "люки"), введенные разработчиком, позволяющие обходить процедуры идентификации, аутентификации, проверки целостности и др., предусмотренные в операционной системе**	**отсутствие необходимых средств защиты (аутентификации, проверки целостности, проверки форматов сообщений, блокирования несанкционированно модифицированных функций и т.п.)**	**ошибки в программах (в объявлении переменных, функций и процедур, в кодах программ), которые при определенных условиях (например, при выполнении логических переходов) приводят к сбоям, в том числе к сбоям функционирования средств и систем защиты информации, к возможности несанкционированного доступа к информации**

critical	STATIC_OVERFLOW STATIC_OVERFLOW.PROC BUFFER_OVERFLOW BUFFER_OVERFLOW.DFA BUFFER_OVERFLOW.LOOP BUFFER_OVERFLOW.ARGV STATIC_OVERFLOW.LOCAL BUFFER_OVERFLOW.EX BUFFER_OVERFLOW.PROC DYNAMIC_OVERFLOW DYNAMIC_OVERFLOW.PROC DYNAMIC_OVERFLOW.EX BUFFER_OVERFLOW.LIB.EX DEADLOCK	BUFFER_UNDERFLOW	OVERFLOW_AFTER_CHECK OVERFLOW_AFTER_CHECK.EX FB.ML_SYNC_ON_UPDATED_FIELD	TAINTED_ARRAY_INDEX.EX TAINTED_ARRAY_INDEX TAINTED_DYNAMIC_OVERFLOW OVERFLOW_UNDER_CHECK OVERFLOW_UNDER_CHECK.LIB OVERFLOW_UNDER_CHECK.PROC TAINTED_INT TAINTED_INT.LOOP TAINTED_INT.LOOP.MIGHT TAINTED_INT.PTR TAINTED_PTR TAINTED_PTR.MIGHT TAINTED_PTR.FORMAT_STRING FB.IL_INFINITE_RECURSIVE_LOOP FB.ML_SYNC_ON_FIELD_TO_GUARD_CHANGING_THAT_FIELD	DEREF_OF_NULL DEREF_OF_NULL.CONST DEREF_AFTER_NULL DEREF_AFTER_NULL.EX DEREF_OF_NULL.EX AUTHENTICATE_IN_LOOP FB.UL_UNRELEASED_LOCK_EXCEPTION_PATH

major	BUFFER_OVERFLOW.PROC. STRICT DYNAMIC_SIZE_MISMATCH DYNAMIC_SIZE_MISMATCH.MACRO BUFFER_OVERFLOW.VALUES HANDLE_LEAK HANDLE_LEAK.EX DEADLOCK.CLASS_LOADING NO_LOCK.STAT NO_LOCK.STAT.EX NO_LOCK.GUARD WRONG_LOCK.STATIC DOUBLE_LOCK EMPTY_SYNC WRONG_LOCK		NO_UNLOCK BAD_WAIT_OF_COND	TAINTED_ARRAY_INDEX.MIGHT TAINTED_ARRAY_INDEX.LOOP CHECK_AFTER_OVERFLOW OVERFLOW_AFTER_CHECK.MACRO TAINTED_INT.MIGHT TAINTED_INT.COND TAINTED_INT.MIGHT.COND TAINTED_INT.INFINITE_LOOP TAINTED_INT.INFINITE_LOOP.MIGHT TAINTED.INT_OVERFLOW TAINTED_INT.PTR.MIGHT TAINTED_PTR.COND TAINTED_PTR.MIGHT.COND FORMAT_STRING.PARAM_LACK UNCHECKED_FUNC_RES.LIB CHECK_AFTER_PASS_TO_PROC INFINITE_LOOP.EX INFINITE_LOOP.STRING INFINITE_LOOP.NEW INFINITE_LOOP.INT_OVERFLOW INFINITE_LOOP.INT_OVERFLOW.STRICT INFINITE_ LOOP.INT_ OVERFLOW.ARRAY INFINITE_LOOP.INT_OVERFLOW.ARRAY.STRICT DIVISION_BY_ZERO DIVISION_BY_ZERO.EX DIVISION_BY_ZERO.UNDER_CHECK DEFAULT_CASE_MISSING CATCH.NULL_POINTER_EXCEPTION CATCH.GENERIC_EXCEPTION CATCH.NO_BODY THROW.GENERIC_EXCEPTION	DEREF_OF_NULL.ASSIGN NULL_AFTER_DEREF NULL_AFTER_DEREF.MIGHT NULL_AFTER_DEREF.INL DEREF_AFTER_NULL.COND DEREF_AFTER_NULL.EX.COND DEREF_OF_NULL.EX.COND DEREF_OF_NULL.RET.LIB DEREF_OF_NULL.RET.STAT DEREF_AFTER_NULL.MIGHT DEREF_OF_NULL.ANNOT.CONST DEREF_OF_NULL.ANNOT.ASSIGN DEREF_OF_NULL.ANNOT.EX DEREF_OF_NULL.ANNOT.EX.COND UNINIT.BASE_CTOR NEGATIVE_CODE_ERROR NEGATIVE_CODE_ERROR.EX NEGATIVE_CODE_ERROR.EX.CONST INVARIANT_RESULT SIMILAR_BRANCHES SIMILAR_BRANCHES.SWITCH RETURN_IN_FINALLY WRONG_SEMICOLON BAD_COPY_PASTE WRONG_ARGUMENTS_ORDER FALL_THROUGH CONFUSING_INDENTATION

orm ala	DYNAMIC_SIZE_MISMATCH.STRICT HANDLE_LEAK.FRUGAL HANDLE_LEAK.CLOSEABLE HANDLE_LEAK.EXCEPTION HANDLE_LEAK.FRUGAL.EXCEPTION HANDLE_LEAK.CLOSEABLE.EXCEPTION HANDLE_LEAK.EX.EXCEPTION INT_OVERFLOW INT_OVERFLOW.AFTER_CHECK INT_OVERFLOW.CONST INT_OVERFLOW.LOOP INT_OVERFLOW.LOOP.STRICT INT_OVERFLOW.LIB INT_OVERFLOW.SHIFT INT_OVERFLOW.ZERO.WRAP INT_OVERFLOW.COND INT_OVERFLOW.TRUNC.COND INT_OVERFLOW.TRUNC.UNDER_BITMASK INT_OVERFLOW.TRUNC.UNDER_BITMASK.LONG LOCK_INCONSISTENT SHADOWED_NAME HANDLE_LEAK.EX.EXCEPTION	PROC_USE.HASH_FUNC		TAINTED.INT_OVERFLOW.TRUNC UNCHECKED_FUNC_RES.USER UNCHECKED_FUNC_RES.USER.STRICT UNCHECKED_FUNC_RES.STAT PASS_TO_PROC_AFTER_CHECK NO_CHECK_IN_LOCK NO_CATCH NO_CATCH.LIBRARY TERNARY_OPERATOR_PRECEDENCE UNCHECKED_FUNC_RES.STAT	DOUBLE_CLOSE DOUBLE_CLOSE.PROC USE_AFTER_RELEASE DEREF_OF_NULL.UNCLEAR DEREF_OF_NULL.STRICT DEREF_AFTER_NULL.RET DEREF_AFTER_NULL.UNCLEAR DEREF_AFTER_NULL.LOOP DEREF_OF_NULL.RET DEREF_OF_NULL.RET.ASSERT DEREF_OF_NULL.COND DEREF_OF_NULL.ANNOT DEREF_OF_NULL.ANNOT.STRICT DEREF_OF_NULL.ANNOT.COND DEREF_OF_NULL.RET.ANNOT NO_BASE_CALL.STAT NO_BASE_CALL.LIB UNREACHABLE_CODE UNREACHABLE_CODE.ENUM UNREACHABLE_CODE.EXCEPTION UNREACHABLE_CODE.RET UNREACHABLE_CODE.GLOBAL REDUNDANT_COMPARISON REDUNDANT_COMPARISON.ALWAYS_FALSE UNREACHABLE_CODE.SWITCH UNREACHABLE_CODE.CATCH DEREF_OF_NULL.COND DEREF_OF_NULL.RET

inor	PASSED_TO_PROC_AFTER_RELEASE HANDLE_LEAK.STRICT HANDLE_LEAK.EXCEPTION.STRICT PROC_USE.VULNERABLE		NO_UNLOCK.STRICT	OVERFLOW_AFTER_CHECK.VAR UNCHECKED_FUNC_RES.LIB.STRICT UNCHECKED_FUNC_RES.LIB.MINOR NO_CATCH.STRICT	NULL_AFTER_DEREF.RET UNREACHABLE_CODE.DEFAULT UNREACHABLE_CODE.TERMINATION INVARIANT_RESULT.EX REDUNDANT_NULL_CHECK BAD_KEYWORD
undefined			SENSITIVE_LEAK SENSITIVE_LEAK.SYSTEM HARDCODED_PASSWORD		NEW.DEREF_OF_NULL.INCONSISTENT UNREACHABLE_CODE.EX INVARIANT_RESULT.BIT_SHIFT NEW.DEREF_OF_NULL.INCONSISTENT VN_FUTURE_DEREF_OF_NULL VN_FUTURE_DEREF_AFTER_NULL PAST_DEREF_OF_NULL

10. Рекомендации по освоению

Перед началом работы необходимо:

- внимательно ознакомиться с Руководством администратора Svace;
- внимательно ознакомиться с Руководством оператора Svace;
- внимательно ознакомиться с Руководством пользователя;