МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ

РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение высшего образования «Самарский национальный исследовательский университет имени академика С.П. Королева»

(Самарский университет)

Институт информатики, математики и электроники

Факультет информатики

Кафедра геоинформатики и информационной безопасности

КУРСОВОЙ ПРОЕКТ ПО ДИСЦИПЛИНЕ

«Инженерия программного обеспечения»

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

«Статический анализатор кода программ, написанных на С++, на уязвимости по информационной безопасности»

ЧАСТЬ 1

Разработка программы. Разработка структуры и общего

алгоритма программы. Распределение работ.

|  |  |
| --- | --- |
| Студент \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  *(подпись)* | К.А. Ковалев |
|  |  |
| Руководитель работы \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  *(подпись)* | Я.А. Мостовой |
|  |  |

САМАРА 2020

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ

РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение высшего образования «Самарский национальный исследовательский университет имени академика С.П. Королева»

(Самарский университет)

Институт информатики, математики и электроники

Факультет информатики

Кафедра геоинформатики и информационной безопасности

**ЗАДАНИЕ НА КУРСОВОЙ ПРОЕКТ**

Студенту ***Ковалеву К.А.*** группы 6411-100503D

Тема проекта: ***«Статический анализатор кода программ, написанных на С++, на уязвимости по информационной безопасности», часть № 1. Разработка программы. Разработка структуры и общего алгоритма программы. Распределение работ.***

|  |  |  |
| --- | --- | --- |
| Планируемые результаты освоения образовательной программы (компетенции) | Планируемые результаты | Содержание задания |
| ПК-5 | **Знать:**  Этапы оценки и управления рисками, современные методики оценки рисков, смысловое содержание численных показателей оценок рисков.  **Уметь:**  определять необходимую и достаточную совокупность средств защиты информации, организационных мер направленных на снижение рисков ИБ, разработать архитектуру СИБ, направленную на снижение рисков информационной безопасности автоматизированных систем.  **Владеть:**  навыками определения приемлемого для организации уровня риска; идентификации, анализа и оценки рисков; ранжирование рисков; принятия решения по рискам и разработка плана реагирования на риски; реализации мероприятий и оценки эффективности реализованных мер. | 1. Провести анализ этапов оценки и управления рисками. Узнать современные методики для оценки рисков, а также смысловое содержание показателей оценок рисков. 2. Использовать умения по обеспечению безопасности, направленную на снижение рисков ИБ. 3. Разработать статический анализатор кода, направленный на снижение рисков информационной безопасности систем. |
| ПК-12 | **Знать:**  критерии и методы оценки качества политики безопасности АС; основные методы управления информационной безопасностью; средства поддержки процессов управления информационной безопасностью АС.  **Уметь:**  проводить аудит информационной безопасности АС; применять методы синтеза системы защиты информации в АС; формировать политику информационной безопасности АС; вести мониторинг информационной безопасности АС.  **Владеть:**  навыками определения класса АС; организации обеспечения информационной безопасности АС; применения средств поддержки процессов управления информационной безопасностью АС. | 1. Определить критерии, методы оценки, управления и средств поддержки безопасности АС. 2. Разработать политику, средства мониторинга и поддержки процессов информационной безопасности АС. |
| ПСК-7.5 | **Знать:**  задачи, решаемые подразделениями по защите информации, виды угроз безопасности информации.  **Уметь:**  координировать деятельность подразделений и специалистов по защите информации в организациях, в том числе на предприятии и в учреждении.  **Владеть:** навыками формализации постановки задач по защите информации, контроля хода выполнения задач. | 1. Использовать научно-публицистическую литературу для определения задач по защите информации. Узнать виды угроз безопасности информации. 2. Уметь координировать деятельность на предприятии и в учреждении. 3. Сформулировать задачи для обеспечения безопасности. |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Дата выдачи задания | 6 февраля 2020 г. | | | |
| Срок представления на кафедру пояснительной записки | | | 24 мая 2020 г. | |
| Руководитель курсового проекта профессор кафедры ГИиИБ, д.т.н. | |  | | Я.А. Мостовой |
|  | | *(подпись)* | |  |
| Задание принял к исполнению  студент группы № 6411-100503D | |  | | К.А. Ковалев |
|  | | *(подпись)* | |  |

РЕФЕРАТ

**Пояснительная записка к курсовому проекту:** 37 c., 12 рисунков, 3 таблицы, 10 источников, 2 приложения.

АНАЛИЗАТОР, СТАТИЧЕСКИЙ АНАЛИТОР, УЯЗВИМОСТЬ, ИНФОРМАЦИОННАЯ БЕЗОПАСНОСТЬ, АНАЛИЗ КОДА, ЯЗЫК ПРОГРАММИРОВАНИЯ.

Цель работы – разработка статического анализатора кода программ, написанных на C++, для анализа уязвимостей информационной безопасности, а также разработка средств отладки разрабатываемого анализатора и средств, подтверждающих его эффективность таким образом, чтобы все специально внесенные виды уязвимостей во фрагменты ПО в различных сочетаниях могли бы быть обнаружены анализатором.

В итоге были получены следующие результаты:

1. Выполнен анализ существующих статических анализаторов кода C++;
2. Выполнен обзор существующих методов и принципов, на которых основывается статический анализ кода;
3. Разработан и протестирован собственный анализатор статического кода C++, написанный на языке Python в среде разработке PyCharm.

Содержание

[ВВЕДЕНИЕ 7](#_Toc40622782)

[1 АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ 9](#_Toc40622783)

[1.1 Теоретические основы 9](#_Toc40622784)

[1.2 Анализ существующих реализаций 10](#_Toc40622785)

[1.3 Выводы и результаты 12](#_Toc40622786)

[2 ОПИСАНИЕ РАЗРАБОТАННОГО ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ 13](#_Toc40622787)

[2.1 Выбор средств разработки и программных средств 13](#_Toc40622788)

[2.2 Требования к разрабатываемой программе 13](#_Toc40622789)

[2.3 Структура программы 15](#_Toc40622790)

[2.4 Описание модулей и их составных частей 16](#_Toc40622791)

[2.4.1 Модуль первичного лексического анализа 16](#_Toc40622792)

[2.4.2 Модуль обработчиков уязвимостей 17](#_Toc40622793)

[2.4.3 Модуль пользовательского интерфейса 25](#_Toc40622794)

[2.4.4 Агрегирующий модуль 30](#_Toc40622795)

[2.4.5 Модуль тестирования и генерации тестового кода С++ 30](#_Toc40622796)

[2.5 Вывод и результаты 31](#_Toc40622797)

[3 ТЕСТИРОВАНИЕ 32](#_Toc40622798)

[ЗАКЛЮЧЕНИЕ 36](#_Toc40622799)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 37](#_Toc40622800)

[ПРИЛОЖЕНИЕ А 38](#_Toc40622801)

[Индивидуальная часть 43](#_Toc40622802)

[1. Распределение работ 43](#_Toc40622803)

[2. Разработка структуры и общего алгоритма программы 43](#_Toc40622804)

[3. Разработка программы 44](#_Toc40622805)

[ПРИЛОЖЕНИЕ Б 45](#_Toc40622806)

ВВЕДЕНИЕ

Одним из общепринятых и эффективных методов выявления дефектов является обзор кода. Его суть в совместном внимательном чтении исходного кода с целью обнаружения ошибок и дефектов. В этом процессе выявляются ошибки или участки кода, которые могут стать ошибочными в будущем. Обзор кода является эффективным, но очень трудозатратным методом, так как отнимает время от основной разработки у программистов, а также требует от них большой концентрации. Альтернативным решением являются инструменты статического анализа кода. Они обрабатывают исходные тексты программ и выдают программисту рекомендации обратить повышенное внимание на определенные участки кода. Несмотря на то, что существует большое количество статических анализаторов кода, зачастую в них опускается возможность обнаружения проблем, связанных с уязвимостями информационной безопасности. Это очень весомое опущение, так как состояния информационной безопасности разрабатываемой программы является одним из главных столпов, на котором стоит архитектура приложения. В связи с этим разработка статического анализатора, учитывающего возможности обнаружения уязвимостей информационной безопасности является важной и актуальной задачей.

Цель работы – разработка статического анализатора кода программ, написанных на C++, для анализа уязвимостей информационной безопасности, а также разработка средств отладки разрабатываемого анализатора и средств, подтверждающих его эффективность таким образом, чтобы все специально внесенные виды уязвимостей во фрагменты ПО в различных сочетаниях могли бы быть обнаружены анализатором.

В соответствии с поставленной целью были сформулированы и решены следующие задачи:

1. анализ предметной области;
2. разработка анализатора программ, написанных на С++, на уязвимости информационной безопасности;
3. программирование разработанных алгоритмов;
4. разработка проверочных тестов;
5. разработка генератора случайного кода С++;
6. автономная отладка программ;
7. комплексная отладка программ.

# АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ

## Теоретические основы

Принцип работы статических анализаторов основывается на разборе предоставленного к анализу кода без его выполнения.

В ходе исследования предметной области, мы выяснили, что существует два основных подхода при реализации статического анализатора.

Первый подход – использование регулярных выражений. В этом случае анализ кода ведется с помощью базы, в которой находятся шаблоны уязвимостей. Шаблоны представляют из себя регулярные выражения, составленные по сигнатуре уязвимости. Такой подход удобен в реализации и имеет право на существование, но его возможности ограничены. Так, например, с его помощью практически невозможно осуществлять комплексный анализ, то есть анализ уязвимостей, которые распределены по коду или пересекаются с другими уязвимостями.

Второй подход решает проблему комплексного анализа. В этом случае анализ начинается с разбиения кода анализируемой программы на лексемы – составные части, такие как функции, переменные и т.д. Далее из этих лексем строится объект, характеризующий анализируемую программу. После этого начинается анализ полученного объекта на наличие уязвимостей с помощью реализованных для этого алгоритмов.

После изучения двух подходов было решено использовать гибридный подход, то есть использование комплексного анализа с разбиением совместно с регулярными выражениями. Такой подход сделал возможным разработку приложения, способного к комплексному анализу уязвимостей в рамках курсового проекта.

## Анализ существующих реализаций

В рамках анализа предметной области был произведен обзор существующих решений статического анализа. При исследовании конкретной реализации статического анализатора мы изучали его принцип работы, функциональные возможности, а также способность к переиспользованию для добавления собственного функционала, позволяющего осуществлять поиск необходимых нам уязвимостей. Мы выделили три наиболее распространенных анализатора.

**CppСheck** — статический анализатор кода для языка C/C++, предназначенный для поиска ошибок, которые не обнаруживаются компиляторами. Главной целью проекта является сведение до минимума количества ложных срабатываний при поиске ошибок. Анализатор способен проверять нестандартные участки кода, включающие использование расширений компилятора, встраиваемый ассемблер и т. п.

Перечень некоторых обрабатываемых уязвимостей:

1. утечки памяти;
2. утечки ресурсов;
3. выход за границы массивов;
4. использование неинициализированных переменных;
5. использование устаревших функций;
6. разыменование нулевого указателя.

**PVS-Studio** — инструмент для выявления ошибок и потенциальных уязвимостей в исходном коде программ, написанных на языках С, C++, C# и Java. Работает в 64-битных системах на Windows, Linux и macOS и может анализировать код, предназначенный для 32-битных, 64-битных и встраиваемых ARM платформ. PVS-Studio выполняет статический анализ кода и генерирует отчёт, помогающий программисту находить и устранять ошибки. PVS-Studio выполняет широкий спектр проверок кода, но наиболее силён в поисках опечаток и последствий неудачного переиспользования кода методом «копировать-вставить».

Перечень некоторых обрабатываемых уязвимостей:

1. утечки памяти;
2. утечки ресурсов;
3. выход за границы массивов;
4. использование устаревших функций;
5. проверка операций ввода/вывода;
6. разыменование нулевого указателя.

**Clang Static Analyzer** - инструмент анализа исходного кода, который находит ошибки в программах на C, C ++ и Objective-C. Анализатор является проектом с полностью открытым исходным кодом и является частью проекта Clang. Как и остальная часть Clang, анализатор реализован в виде библиотеки C ++, которая может использоваться другими инструментами и приложениями.

Перечень некоторых обрабатываемых уязвимостей:

1. утечки памяти;
2. поиск неиспользуемых переменных;
3. проверка возвращаемого значения;
4. выход за границы массивов;
5. использование устаревших функций;
6. проверка операций ввода/вывода;
7. разыменование нулевого указателя.

Таблица 1 – Обоснования отказа от имеющихся анализаторов

|  |  |  |
| --- | --- | --- |
| Название анализатора | Расположение | Причина отказа от анализатора |
| СppCheck | http://cppcheck.source  forge.net/ | Отсутствие анализа уязвимостей многопоточности.  Цельная структура, анализатора, не позволяющая полноценно интегрировать свои версии шаблонов уязвимостей. |
| PVS-Studio | https://www.viva64.com  /ru/pvs-studio/ | Отсутствие анализа уязвимостей многопоточности.  Исходный код не является открытым, невозможно вносить свои изменения. |
| Clang Static  Analyzer | https://clang-analyzer.llvm.org/ | Отсутствует анализа уязвимостей многопоточности, уязвимостей, связанных с выделением памяти. |

## Выводы и результаты

В ходе проведенного анализа предметной области были изучены и выбраны подходы к реализации статического анализатора. Были исследованы существующие решения в области статического анализа, а также была рассмотрена возможность их переиспользования для добавления собственного функционала, позволяющего осуществлять поиск необходимых уязвимостей.

# ОПИСАНИЕ РАЗРАБОТАННОГО ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

## Выбор средств разработки и программных средств

Для разработки нашего приложения был выбран язык Python3. Выбор был обусловлен двумя основными причинами:

1. наличие гибкой и функциональной библиотеки для работы с регулярными выражениями - Lib/re.py, а также расположенность языка для работы с ними;
2. наличие библиотеки для создания удобного и дружественного интерфейса - Lib/tkinter;

В качестве среды разработки выступила программа PyCharm, специально предназначенная для создания приложений на языке Python. Она имеет удобный интерфейс и большое количество полезных функций, позволяющих проводить отладку кода, связываться с удаленным репозиторием для синхронизации изменений в разрабатываемом приложение и т.д.

## Требования к разрабатываемой программе

Разрабатываемое приложение должно соответствовать ряду требований, которые были описаны в методическом указании к курсовому проекту, а также сформулированы в ходе предреализационного планирования.

1. Программа должна иметь удобный и понятный графический интерфейс.
2. Интерфейс программы должен содержать:
3. список загруженных на проверку программ;
4. список загруженных поисковых программ;
5. список загруженных тест примеров;
6. указатель на исследуемые уязвимости с изменением его состояния по мере проверки уязвимостей;
7. кнопку «произвести проверку» программы №… или последовательно все;
8. кнопку произвести проверку на тест- примерах №… или на всех.
9. выводимые результаты работы программы для каждой проверяемой программы или тест примера должны содержать виды уязвимостей с отметкой обнаружена или не обнаружена. В случае обнаружения указывается точка обнаружения.
10. Должен быть реализован поиск следующих уязвимостей:
11. переполнение буфера;
12. пренебрежение обработок ошибок;
13. проблемы, связанные с динамическим выделением памяти;
14. внедрение команд;
15. внедрение SQL команд;
16. состояние гонки;
17. ошибка форматной строки;
18. некорректный доступ к файлам;
19. читатели – писатели;
20. случайные числа криптографического характера;
21. переполнение целых чисел;
22. пренебрежение безопасным хранением данных;
23. утечка информации.
24. Программа должна быть проверена набором ручных тестов, а также тестами, созданными автогенератором тестов.

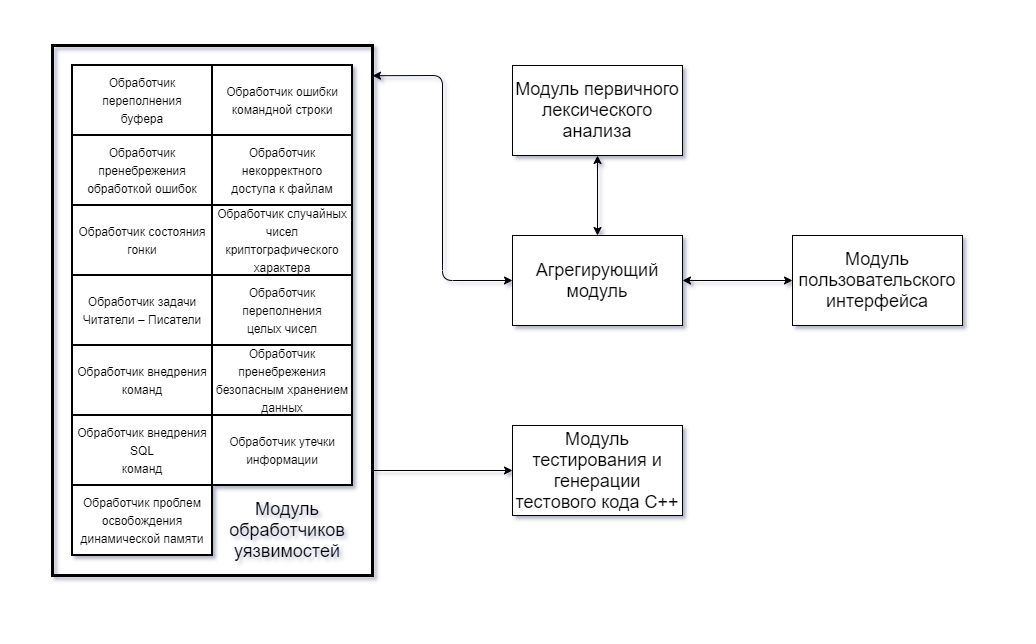
## Структура программы

Разработанную программу можно разделить на пять основных модулей (рисунок 1).

1. Модуль первичного лексического анализа.
2. Модуль обработчиков уязвимостей, содержащий в себе 13 частей:

* обработчик переполнения буфера;
* обработчик пренебрежения обработкой ошибок;
* обработчик состояния гонки;
* обработчик задачи читатели-писатели;
* обработчик внедрения команд;
* обработчик внедрения sql команд;
* обработчик проблем освобождения динамической памяти;
* обработчик ошибки командной строки;
* обработчик некорректного доступа к файлам;
* обработчик случайных чисел криптографического характера;
* обработчик переполнения целых чисел;
* обработчик пренебрежения безопасным хранением данных;
* обработчик утечки информации.

1. Модуль пользовательского интерфейса.
2. Агрегирующий модуль.
3. Модуль тестирования и генерации тестового кода C++.

Рисунок 1 – Структурная схема ПО

## Описание модулей и их составных частей

### Модуль первичного лексического анализа

Как мы отметили раннее нами был выбран гибридный подход к статическому анализу кода. Модуль первичного лексического анализа обеспечивает начальный анализ кода, который позволяет получить объектную модель исследуемой программы. В эту модель входят все объявленные в анализируемом коде функции, их параметры, переменные и потоки. При анализе очень важно иметь полное представление о структуре анализируемой программы, так как, например, в разных функциях могут быть объявлены переменные или потоки с одинаковым именем, и это необходимо учитывать.

Для представления модели используются классы:

1. Variable – данный класс отвечает за хранение информации о найденной переменной. Он содержит полную декларацию переменной, номер строки, на которой она была объявлена, ее имя, тип и значение.
2. FunctionContext – данный класс отвечает за хранение информации о найденной функции. Класс содержит тип возвращаемого значения функции, ее имя, список ее потоков, параметров и переменных, которые являются объектами класса Variable, а также список всех строк, которые включает в себя функция.
3. Thread - данный класс отвечает за хранение информации о найденном потоке из стандартной библиотеки С++. Он содержит полную декларацию потока, номер строки, на которой он была объявлен, его имя, имя исполняемой функции и список переданных параметров.

После завершения первичного лексического анализа сформированная объектная модель анализируемой программы поступает к обработчикам уязвимостей. Полученная ими информация расширяет возможности анализа и делает его проще.

### Модуль обработчиков уязвимостей

Модуль обработчиков уязвимостей состоит из 13 классов, отвечающих за поиск и обработку конкретных уязвимостей.

Таблица 2 – Соответствие уязвимости, ее обработчика и краткого описания метода обнаружения

|  |  |  |
| --- | --- | --- |
| Уязвимость | Метод обнаружения | ПО обнаружения |
| Переполнение буфера | Поиск объявленных буферов и функций, в которых они были использованы. | BufferOverflowHandler |
| Переполнение целых чисел | Поиск арифметических выражений, в которых используются объявленные целочисленные переменные и проверка этих выражений на переполнение. | IntegerOverflowHandler |
| Внедрение SQL-кода | Поиск уязвимых функций вызова SQL запросов с телом запроса, в котором отсутствует экранирование символа единичной кавычки. | SqlInjectionHandler |
| Внедрение команд | Поиск уязвимых функции, которые позволяют выполнить внедрение команд. | CommandsIntoductionHandler |
| Пренебрежение безопасным хранением данных | Поиск уязвимых функций, которые позволяют изменять уровень доступа к данным и задавать привилегии для определенных объектов  ОС. | DataStorageManagmentHandler |
| Утечка информации | Поиск уязвимых функций, которые могут привести к потере или утечке данных. | DataLeakHandler |
| Некорректный доступ к файлам | Поиск уязвимых функций, позволяющие проводить операций над файлами (создание ссылок, файлов, директорий, изменение файлов и т.д. | IncorrectFileAccessHandler |
| Случайные числа криптографического характера. | Поиск функций генерации случайных чисел, инициализированных константным значениями, а также других не криптостойких генераторов из стандартных библиотек С++. | RandomGeneratorHandler |
| Ошибка форматной строки | Поиск функции функций, использующих форматную строку. | FormatStringHandler |
| Состояние гонки | Поиск потоков, у которых используется одна и та же исполняемая функция. | RaceConditionHandler |
| Читатели-писатели | Поиск потоков, использующих одинаковые переменные в качестве параметров исполняемых функций, а также анализ этих исполняемых функций на наличие и структуру блоков синхронизации. | ReadersWritersHandler |
| Проблемы с динамическим выделением памяти | Поиск переменных, для которых была выделена, но не освобождена память. | MemoryLeakHandler |
| Пренебрежение обработкой ошибок | Поиск «try-catch» выражений с отсутствием обработки исключений. | EmptyCatchHandler |

Рассмотрим каждую уязвимость и ее обработчик в отдельности. Заранее отметим, что все обработчики сконструированы так, чтобы иметь возможность находить уязвимости, даже, если они перекрыты или пересекаются с другими уязвимостями и с произвольным кодом.

1. *Переполнения буфера*

Данная уязвимость возникает, например, при получении данных извне и дальнейшем копировании их во внутренний буфер без проверки размера копируемых данных. При необходимом объеме записываемых данных адрес возврата, находящийся в стеке, может быть перезаписан. При этом на его место можно перезаписать необходимый адрес, который указывает на вредоносный код.

Обработчик «BufferOverflowHandler» руководствуется следующим шаблоном при поиске уязвимости:

1. в коде ведется поиск всех объявленных переменных, которые являются буферами, то есть поиск массивов и указателей;
2. поиск функции, предназначенных для работы с буферами;
3. проверка того, используются ли найденные раннее переменные в обнаруженных потенциально опасных функциях.
4. Пренебрежение обработкой ошибок

Безопасность подвергается серьезной угрозе, когда не происходит должного уделения внимания обработке ошибок. Опасность заключается в том, что в некоторых ситуациях программа может оказаться в некорректном состоянии, что ведет к отказу обслуживания, а, следовательно, создает угрозу безопасности.

Обработчик «EmptyCatchHandler» руководствуется следующим шаблоном при поиске уязвимости:

1. ведется поиск «try-catch» конструкции;
2. осуществляется проверка, является ли тело блока «catch» пустым, при этом обработчик учитывает возможность вложенных конструкций «try-catch» (рисунок 2), в этом случае будет осуществлен рекурсивный анализ таких конструкций.

try {

#код программы

}

catch {

try {

#код программы

}

catch {} #пустой блок «catch»

}

Рисунок 2 – вложенная конструкция «try-catch»

1. Переполнение целых чисел

Уязвимость в компьютерной арифметике, при которой вычисленное в результате операции значение не может быть помещено в n-битный целочисленный тип данных. Различают переполнение через верхнюю границу представления и через нижнюю.

Обработчик IntegerOverflowHandler руководствуется следующим шаблоном при поиске уязвимости:

1. поиск всех инициализированных, целочисленных переменных;
2. поиск всех арифметических выражений с найденными раннее переменными;
3. проверка результата арифметического выражения на возможность переполнения.

1. Пренебрежение обработкой ошибок

Данная уязвимость возникает при работе с динамическим выделением памяти. Например, программа завершается без освобождения памяти.

Обработчик MemoryLeakHandler руководствуется следующим шаблоном при поиске уязвимости:

1. поиск всех переменных, для которых была выделена память;
2. поиск операторов высвобождения памяти, по всем найденным переменным;
3. если высвобождение памяти не было обнаружена, появляется предупреждение о возможной ошибке.
4. Внедрение SQL команд

Эта уязвимость предполагает, что любая строка, значение которой не может быть определена во время компиляции, может содержать данные, вводимые пользователем. Созданная из введенных пользователем данных командная строка SQL уязвима перед атаками путем внедрения кода SQL. При атаке путем внедрения кода SQL нарушитель предоставляет входные данные, которые изменяют структуру запроса с целью повредить или получить несанкционированный доступ к основной базе данных. Типичные методы включают в себя введение одинарной кавычки или апострофа, который является разделителем строкового литерала SQL; два тире, обозначающие комментарий SQL; и точка с запятой, которая указывает, что следует новая команда.

Обработчик SqlInjectionHandler руководствуется следующим шаблоном при поиске уязвимости:

1. поиск функций в которых выполняется sql – запрос;
2. проверка тела sql – запроса, переданного функции на наличие неэкранированной одинарной кавычки.
3. Внедрение команд

Если в отношении пользовательского ввода не осуществляется должная проверка, то может быть осуществлена атака путем внедрения команды. Используя эту уязвимость, атакующий может формировать ввод таким образом, что он будет содержать команды операционной системы, которые будут выполняться с привилегиями уязвимого приложения.

Обработчик CommandsIntroductionHandler руководствуется следующим шаблоном при поиске уязвимости:

1. поиск функций, позволяющие выполнять исполняемые файлы и системные команды, а также создавать дочерние процессы, которым можно передать команду на выполнение.
2. Состояние гонки

Уязвимость, при которой работа системы или приложения зависит от того, в каком порядке выполняются части кода. Состояние гонки возникает тогда, когда несколько потоков многопоточного приложения пытаются одновременно получить доступ к одним и тем же ресурсам, причем хотя бы один поток выполняет запись. Кроме того, ошибки такого рода невероятно сложно воспроизвести повторно.

Обработчик RaceConditionHandler работает с любым количеством потоков и руководствуется следующим шаблоном при поиске уязвимости:

1. получение всех объявленных потоков;
2. анализ найденных потоков на наличие одинаковых исполняемых функций.
3. Ошибка форматной строки

Данная уязвимость заключается в использовании одной из функций форматного вывода (printf, fprintf, sprintf). Эти функции имеют неопределенное количество параметров, число и интерпретация которых определяется форматной строкой. Это позволяет злоумышленнику задать свою собственную форматную строку, что ведет к возможности чтения значений из стека, а при указании специального формата, и к записи в память.

Обработчик FormatStringHandler руководствуется следующим шаблоном при поиске уязвимости:

1. поиск функций, использующие форматную строку в качестве параметра.
2. Некорректный доступ к файлам

Можно выделить типичные проблемы безопасности, связанные с файловыми операциями. Первый тип - является ли файл файлом вообще и стоит ли с ним работать. Второй вид уязвимости состоит в вопросе существования файла, с которым будет вестись работа.

Обработчик IncorrectFileAccess руководствуется следующим шаблоном при поиске уязвимости:

1. поиск объявленных файловых переменных;
2. поиск функций, отвечающий за создание директорий, файлов.
3. Задача читатели писатели

Данная уязвимость возникает при одновременном доступе нескольких потоков к одной и той же области памяти, допускающей чтение и запись. При этом одновременно могут читать сколько угодно потоков, но писать — только один. При одновременном чтении проблем не возникает, но изменение памяти во время ее чтения некоторым потоком может привести к серьезным ошибкам.

Обработчик ReadersWritersHandler работает с любым количеством потоков и руководствуется следующим шаблоном при поиске уязвимости:

1. получение всех объявленных потоков;
2. анализ потоков на использование одних и тех же переменных;
3. анализ конструкций синхронизаций в исполняемых функциях потоков, а именно проверка наличия мьютексов, а также проверка освобождения мьютексов.
4. Случайные числа криптографического характера

Зачастую при решении криптографических задач используются случайные числа, которые получают с помощью генераторов случайных чисел. Алгоритмы их получения должны быть криптостойкими, иначе это может привести к возникновению угроз безопасности. Частой ошибкой является инициализация генератора кода константным значением, так как в этом случае криптостойкость алгоритма значительно уменьшается, и значения последовательности можно вычислить. Рекомендуется использовать в качестве значения инициализации уникальное значение, например, текущее время.

Обработчик RandomGeneratorHandler руководствуется следующим шаблоном при поиске уязвимости:

1. поиск функций генерации случайных чисел;
2. проверка на то, что найденная функция генерации инициализирована константным значением.
3. Пренебрежение безопасным хранением данных

Уязвимость возникает при встраивании конфиденциальной информации в код программы и некорректном использовании элементов управления доступом к файлам, дескрипторам и другим данным.

Обработчик DataStorageManagmentHandler руководствуется следующим шаблоном при поиске уязвимости:

1. поиск функций, которые изменяют уровень доступа к данным.

1. Утечка информации

Злоумышленник получает данные, которые могут привести к нарушению политики безопасности. Утечка информации может быть случайной и намеренной. При случайной утечке, информация может просочиться по неочевидному каналу, либо из-за алгоритмической ошибки в программы. Намеренная же возникает, когда команда программистов и конечных пользователей неодинаково понимают, что нужно защищать. Яркий пример утечки – слишком подробное логирование действий, происходящих в программе или чрезмерное количество информации об ошибках.

Обработчик DataLeakHandler руководствуется следующим шаблоном при поиске уязвимости:

1. поиск функций, которые могут приводить к утечке информации.

### Модуль пользовательского интерфейса

Модуль отвечает за реализацию графического интерфейса, с помощью которого пользователь использует функционал анализатора.

**1. Главное окно.**

Главное окно приложения (рисунок 3) можно поделить на три основополагающие части:

1) Блок доступных к анализу уязвимостей, расположенный в левой части главного окна (рисунок 3).

Данный блок предназначен для выбора тех уязвимостей, которые будут исследоваться при анализе выбранной программы. Можно выбрать как одну отдельную уязвимость, так и несколько сразу.

2) Блок, в котором располагается список всех тестов, написанных участниками бригады, согласно составленной таблице ответственности по написанию тестов. Блок расположен в центре главного окна (рисунок 2). Здесь же располагаются две активные кнопки: «Проверить все» - запускает анализ всех тестов в данном блоке по выбранным уязвимостям, и «Проверить» - запускает анализ только выбранных тестов в данном блоке по выбранным уязвимостям.

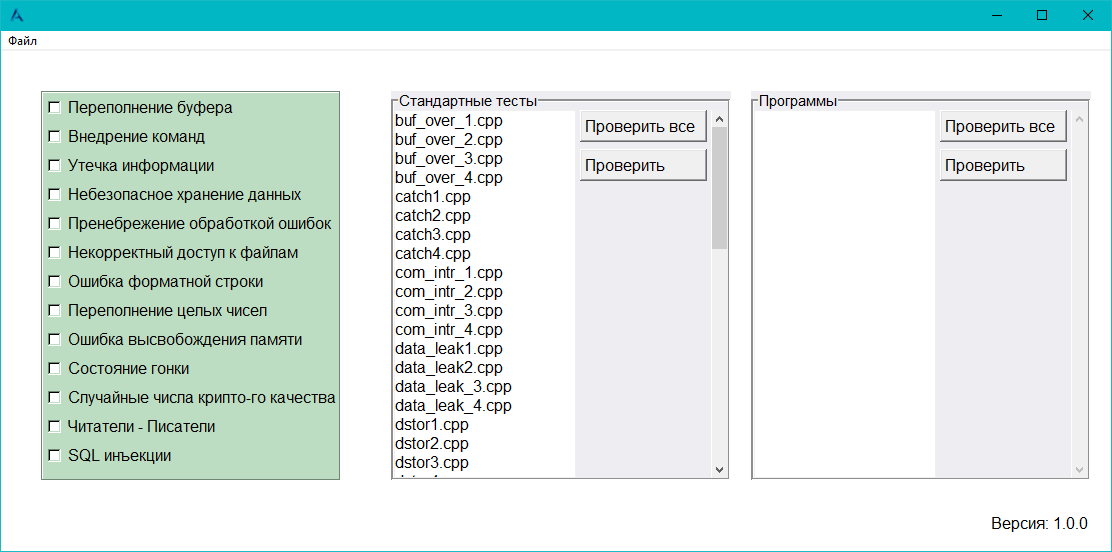
****3) Блок, в котором располагается список программ, которые пользователь самостоятельно загружает в анализатор. Расположен в правой части главного окна (рисунок 2). Функциональность кнопок «Проверить все» и «Проверить» аналогична центральному блоку.

Рисунок 3 – Главное окно

Загрузка собственных программ в анализатор происходит путем нажатия кнопки «Файл», расположенного в левом верхнем углу главного окна (рисунок 4), и далее кнопки «Открыть». После этого появится окно выбора программ (рисунок 5).

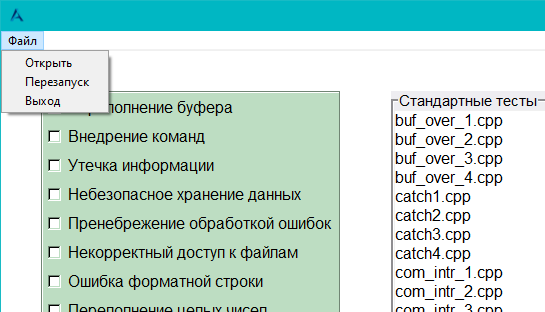


Рисунок 4 – Кнопка «Файл»

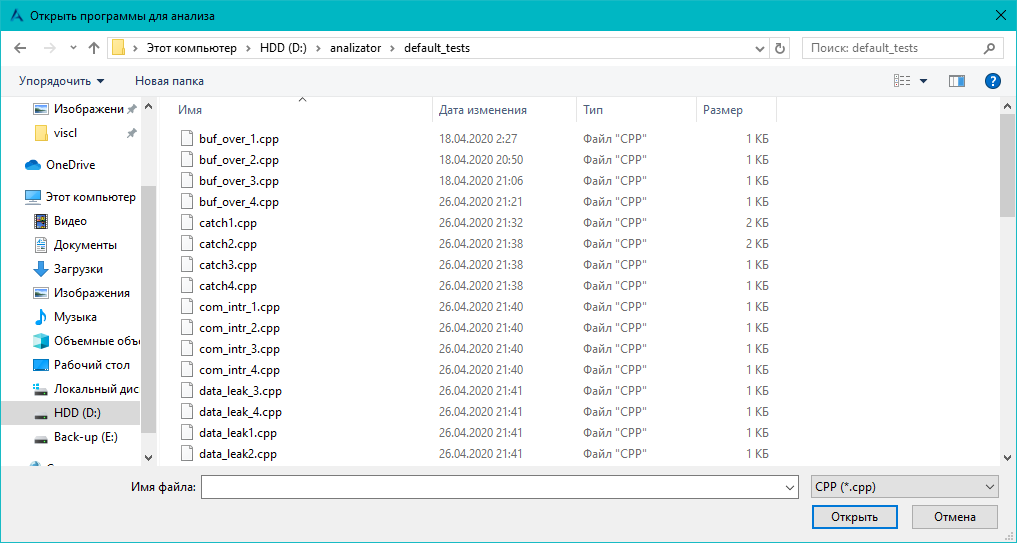


Рисунок 5 – открытие файлов для анализа

Также в меню «Файл» доступны пункты: «Перезапуск» - перезапуск программы и «Выход» - выход из программы.

**2. Окно анализа программы.**

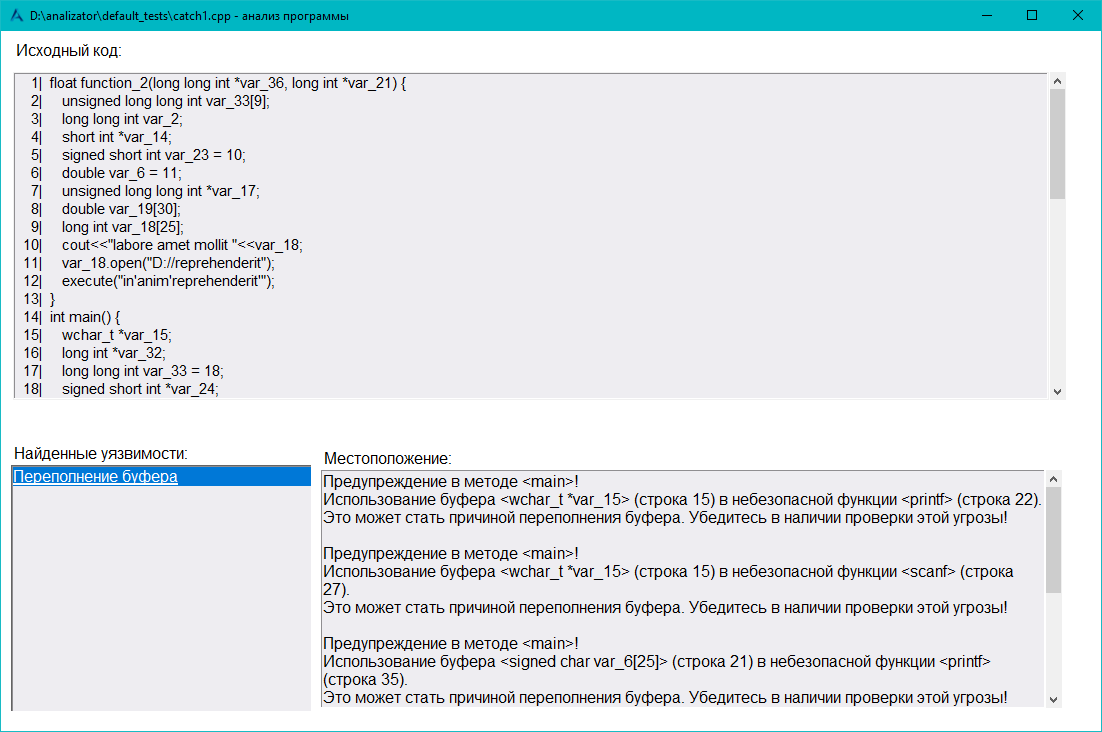
При нажатии на одну из кнопок «Проверить все» или «Проверить» откроется окно анализа выбранной программы или сразу несколько окон в случае выбора нескольких программ, по одному окну на каждую программу (рисунок 6).

Рисунок 6 – окно анализа программы.

В верхней части данного окна для более удобного просмотра результатов анализа располагается анализируемый код. В нижней части – список найденных уязвимостей и их подробные описания, а именно информация по каждой найденной уязвимости каждого типа и ее местоположение (номер строки в коде).

В случае, если в программе не будут найдены выбранные уязвимости появится информационное окно (рисунок 7).

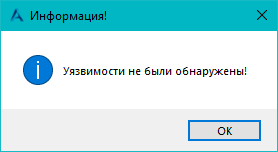


Рисунок 7 – информационное окно

Был рассмотрен и реализован ряд действий при возникновении внештатных ситуаций при работе программы.

1. Если пользователь не выберет ни одного теста и начнет анализ, то появится информационное окно (рисунок 8), после нажатия на которое пользователь сможет продолжить работу.

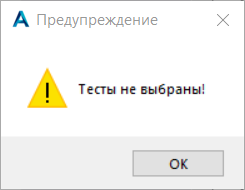


Рисунок 8 – информационное окно

1. В случае с отсутствием выбора программ ситуация – такая же (рисунок 9).

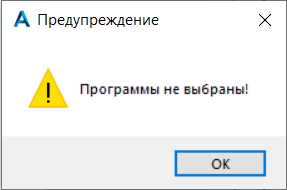


Рисунок 9 – информационное окно

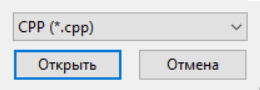
1. Во избежание нештатной ситуации, связанной с открытием файла неизвестного формата, вместо программы предусмотрен контроль открытия файлов (рисунок 10). Разрешенный формат – «.cpp»

Рисунок 10 – разрешенный формат для анализа

1. В случае с отсутствием выбора уязвимости аналогична ситуация с 1 и 2 (рисунок 11)

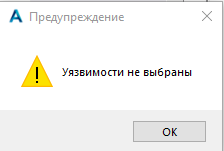


Рисунок 11 – информационное окно

### Агрегирующий модуль

Данный модуль отвечает за координацию и объединение всех предыдущих модулей в одно целое.

### Модуль тестирования и генерации тестового кода С++

Разработанный статический анализатор прошел отладку и ручные тесты, написанные участниками бригады (подробнее об этом в 3 главе), но важным показателем правильности его работы является тестирование с помощью автоматического генератора кода, так как он позволяет провести статистический анализ количества сгенерированных ошибок к количеству найденных ошибок.

Модуль генератор кода разработан как консольное приложение, обособленное от остальных частей проекта. Оно позволяет генерировать произвольный код с заданными уязвимостями, которые будут пересекаться и чередоваться друг с другом, произвольным кодом и другими уязвимостями. При генерации осуществляется подсчет каждой уязвимости. Для проведения статистического анализа указывается количество тестов к генерации.

Алгоритм статистического анализа:

1. Генерация кода;
2. Анализ сгенерированного кода;
3. Вывод отчета о количестве сгенерированных и найденных ошибок.

С помощью полученного отчета можно вести отладку приложения и делать вывод о правильности его работы.

## Вывод и результаты

В результате был разработан статический анализатор кода, способный обнаруживать все 13 заявленных уязвимостей. Также для проверки работоспособности программы был разработан автоматический генератора кода С++.

# ТЕСТИРОВАНИЕ

Для тестирования статического анализатора использовались два подхода:

1. Тесты, написанные членами бригады. По два теста на разрабатываемую уязвимость и по два теста на уязвимость «соседа». Итого по 4 теста на все 13 уязвимостей, то есть 52 теста. Все тесты указаны в индивидуальных частях отчета каждого из участников.

Таблица 3 - распределение тестов по исполнителям

|  |  |
| --- | --- |
| Ковалев К.А. | 1) Тесты для обработчика переполнения буфера;  2) Тесты для обработчика пренебрежения обработок ошибок;  3) Тесты для обработчика задачи Читатели – Писатели;  4) Тесты для обработчика состояния гонки; |
| Терентьев Е.А. | 5) Тесты для обработчика внедрения SQL команд;  6) Тесты для обработчика внедрения команд.  7) Тесты для обработчика проблем, связанных с динамическим выделением памяти; |
| Щекочихин Д.А. | 8) Тесты для обработчика ошибки форматной строки;  9) Тесты для обработчика некорректного доступа к файлам; |
| Оганисян Т.С. | 10) Тесты для обработчика случайных чисел криптографического характера;  11) Тесты для обработчика переполнения целых чисел. |
| Шабакаев А.Р. | 12) Тесты для обработчика пренебрежения безопасным хранением данных;  13) Тесты для обработчика утечки информации. |

Каждый из членов команды пишет тесты на свою уязвимость и на «уязвимость соседа», т.е. Шабакаев пишет тесты для уязвимостей Оганисяна, Оганисян для Щекочихина, Щекочихин для Терентьева, Терентьев для Ковалева, Ковалев для Шабакаева.

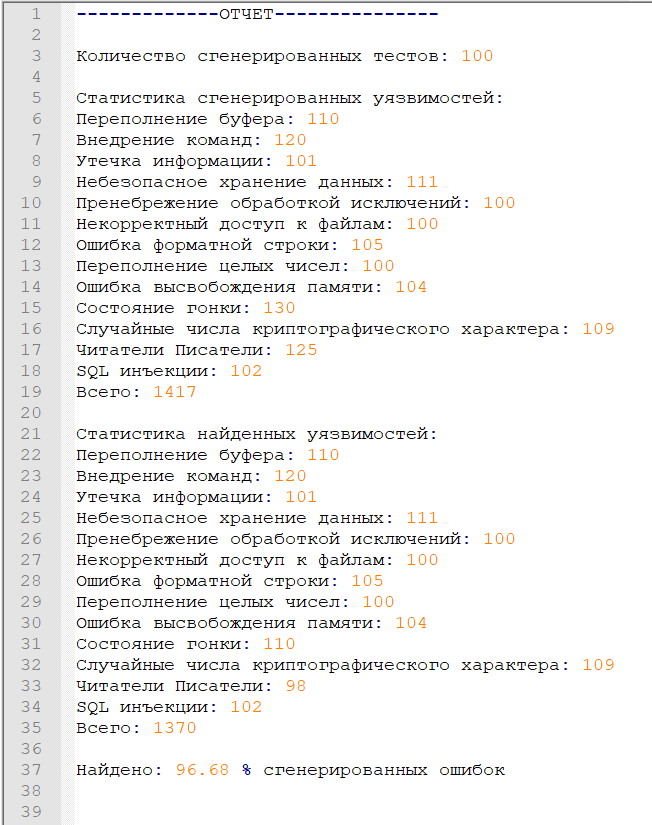
1. Автоматический генератор тестового кода C++. С помощью формируемого им отчета (рисунок 12) производилась отладка и делался вывод о правильности работы программы.

Рисунок 12 – статистический отчет о работе анализатора

Следует отметить, что количество найденных ошибок многопоточности всегда меньше количества сгенерированных, это обусловлено случайностью генерации потоков. Невозможно точно подсчитать сколько сгенерированных поток будут использовать одну и туже исполняемую функцию или одни и те же параметры, то есть будут действительно уязвимы.

В результате использования двух вышеприведенных подходов программа была полностью отлажена и протестирована на соответствие поставленным целям и задачам.

ЗАКЛЮЧЕНИЕ

В процессе выполнения курсового проекта мы проанализировали раннее разработанные анализаторы кода и их методы реализации, на основе полученных знаний сами реализовали статический анализатор кода, написанного на языке «С++», написали тесты, а также генератор случайного кода С++ для проверки работоспособности нашей программы.

Статический анализатор кода проверяет код на наличие уязвимостей информационной безопасности, согласованных на этапе проектирования.

В течение выполнения проекта мы запрограммировали разработанные алгоритмы по поиску уязвимостей и провели тестирование на тестах, написанных членами бригады, и тестах, сгенерированных автоматическим генератором.

Также на протяжении разработки статического анализатора, мы проводили отладку, устраняли ошибки и недоработки.

В результате, мы изучили все согласованные на этапе проектирования уязвимости информационной безопасности, которые могут привести к непоправимым ошибкам при разработке приложения на языке С++, а также методы их определения, реализовали статический анализатор кода С++ и получили бесценный опыт работы в команде.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1 Регулярные выражения / Джеффри Фридл, под ред. А. Галунова, Н. Макаровой, Б. Попова, Ю. Бочиной. - 3-е изд. - СПб.: Символ-Плюс, 2008. - 608 с.

2 Ховард М., Лебланк Д., Виега Д. Как написать безопасный код на С++, Java, Perl, PHP, ASP.NET. - Москва: ДМК Пресс, 2014. - 288 с.

3 Архипова М.В. Генерация тестов для семантических анализаторов // Вычислительные методы и программирование. - 2006. - №3. - С. 55-70.

4 Статический анализ кода C++ // habr.com URL: https://habr.com/ru/post/75123/ (дата обращения: 06.04.2020).

5 Энтони Уильямс Параллельное программирование на С++ в действии. Практика разработки многопоточных программ. - Москва: ДМК Пресс, 2012. - 672 с.

6 Brian Chess, Jacob West Secure Programming with Static Analysis. Addison-Wesley Professional, 2007. - 624 с.

7 Регулярные выражения в Python от простого к сложному. Подробности, примеры, картинки, упражнения // habr.com URL: https://habr.com/ru/post/349860/ (дата обращения: 15.04.2020).

8 М. И. Глухих, В. М. Ицыксон Программная инженерия. Обеспечение качества программных средств методами статистического анализа: учебное пособие. - СПб: Изд-во Политехнического ун-та, 2011. - 149 с.

9 Ф.М. Пучков Методы и средства автоматизированного обнаружения уязвимостей в программах на языке C на основе статического анализа их исходных текстов: автореф. дис. ... канд. физ.-мат. наук: 05.13.19. - Москва, 2010. - 24 с.

10 Златопольский Д. М. Основы программирования на языке Python. - 1 изд. - Москва: ДМК Пресс, 2017. - 284 с.

ПРИЛОЖЕНИЕ А

Статический анализатор кода С++ (рис. 1) состоит из 5 основных модулей.

1. Модуль первичного лексического анализа кода предназначен для разбиения анализируемого кода на составляющие части и построения структуры анализируемого кода. Это позволяет передавать важную информацию об анализируемом коде обработчикам уязвимостей.
2. Модуль обработчиков уязвимостей содержит в себе 13 python-модулей, каждый из которых отвечает за обработку отдельной уязвимости.

Список модулей:

* Обработчик переполнения буфера;
* Обработчик пренебрежения обработок ошибок;
* Обработчик проблем, связанных с динамическим выделением памяти;
* Обработчик внедрения SQL команд;
* Обработчик внедрения команд;
* Обработчик состояния гонки;
* Обработчик ошибки форматной строки;
* Обработчик некорректного доступа к файлам;
* Обработчик задачи Читатели – Писатели;
* Обработчик случайных чисел криптографического характера;
* Обработчик переполнения целых чисел;
* Обработчик пренебрежения безопасным хранением данных;
* Обработчик утечки информации.

1. Агрегирующий модуль отвечает за агрегацию и управление всеми используемыемыми модулями программы.

4) Пользовательский интерфейс отвечает за графическую оболочку нашей программы, принимает и отображает в удобном, читаемом виде информацию, полученную в ходе анализа.

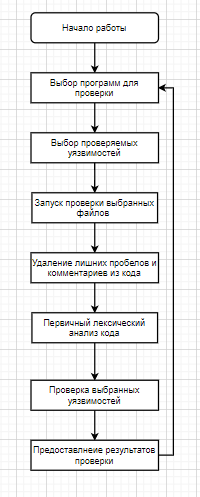
5) Модуль тестирования и генерации тестового кода С++ предназначен для тестирования и отладки разработанного анализатора.

Рисунок 1 – Временная диаграмма статического анализатора

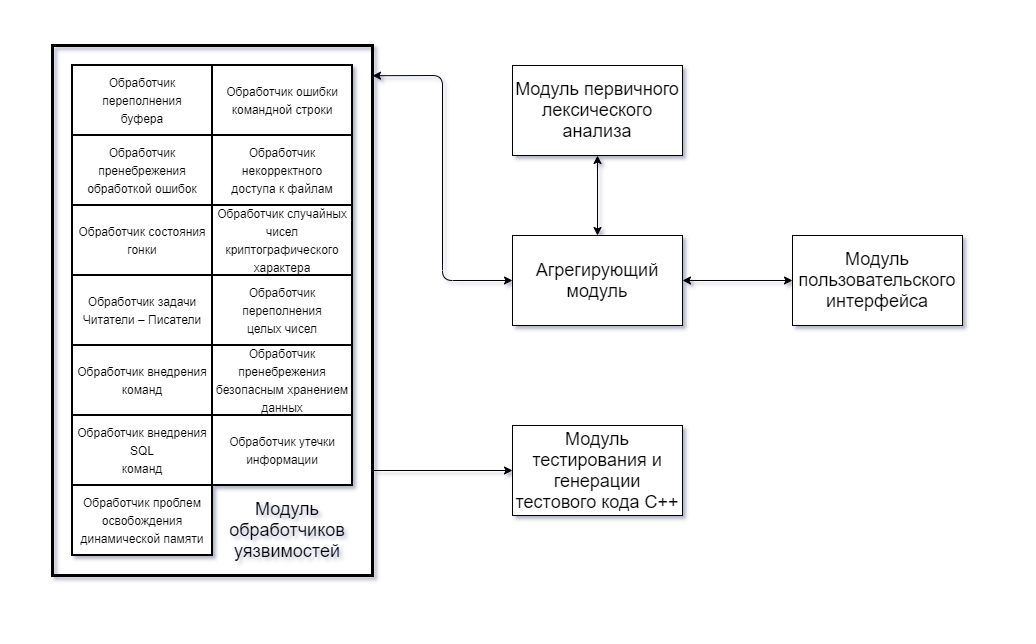


Рисунок 2 - Структурная схема статического анализатора

Таблица 1 - распределения частей проекта среди участников бригады.

|  |  |  |  |
| --- | --- | --- | --- |
| Часть проекта и должность | Общие задачи и наименования. | Ответственный | Конкретные задачи |
| Часть №1 - Бригадир | Разработка программы, структуры и общего алгоритма, распределение работ. | Ковалев К.А. | 1) Обработчик переполнения буфера;  2) Обработчик пренебрежения обработкой ошибок;  3) Обработчик состояния гонки;  4) Обработчик задачи Читатели – Писатели;  5) Модуль первичного лексического анализа кода;  6) Агрегирующий модуль;  7) Модуль тестирования и генерации тестового кода С++. |
| Часть №2 – Зам бригадира | Разработка программы, общего алгоритма. | Терентьев Е.А. | 1) Обработчик внедрения команд;  2) Обработчик внедрения SQL команд;  3) Обработчик проблем, связанных с динамическим выделением памяти. |
| Часть №3 – Разработчик программы | Разработка программы, разработка графического интерфейса. | Щекочихин Д.А. | 1) Обработчик ошибки командной строки;  2) Обработчик некорректного доступа к файлам;  3) Пользовательский интерфейс. |
| Часть №4 – Разработчик программы | Разработка программы, разработка инструкций пользователя и другой ЭТД. | Оганисян Т.С. | 1) Обработчик случайных чисел криптографического характера;  2) Обработчик переполнения целых чисел. |
| Часть №5 – Разработчик программы | Разработка программы. Разработка перечня тестов. Разработка перечня внештатных ситуаций. | Шабакаев А.Р. | 1) Обработчик пренебрежения безопасным хранением данных;  2) Обработчик утечки информации. |

## Индивидуальная часть

В рамках должности бригадира мною были решены следующие задачи:

1. Распределение работ;
2. Разработка структуры и общего алгоритмы программы;
3. Разработка программы.
4. Распределение работ

Перед началом реализации курсового проекта мною были определены основные задачи в соответствие с поставленной целью – разработкой статического анализатора.

Далее был составлен план по выполнению поставленных задач. Все его детали были доведены до участников бригады.

Была составлена таблица распределения частей проекта среди участников бригады. В этой таблице указана область ответственности и решаемые задачи каждого из членов бригады.

Была составлена таблица распределения по составлению тестов среди участников бригады.

В течение всего курсового я проводил комплексный контроль по реализации поставленных задач.

1. Разработка структуры и общего алгоритма программы

После изучения предметной области и выбора подхода к реализации анализатора я начал проработку структуры программы. В результате была создана модель, состоящая из пяти модулей:

1. Модуль первичного лексического анализа кода;
2. Модуль обработчиков уязвимостей;
3. Агрегирующий модуль;
4. Модуль пользовательского интерфейса;
5. Модуль тестирования и генерации тестового кода.

Модульный подход упрощает разработку проекта, так как позволяет вносить меньше правок при изменении логики какого-либо функционала. В случае же с монолитным, цельным приложением, зависимости между частями ПО настолько сложны, что изменение хотя бы одной функции может повлечь изменения во всем проекте.

После определения основных модулей началась разработка общего алгоритма программы. Согласно выбранному гибридному подходу (сочетание построения объектной модели анализируемой программы и использования механизма регулярных выражений) общий алгоритм программы выглядит следующим образом:

1. Анализатор принимает программу для анализа в качестве входных данных;
2. Для удобства анализа принятая программа преобразуется в набор строк;
3. Происходит очистка кода от лишних пробелов и комментариев;
4. Происходит первичный лексический анализ программы, составляется ее объектная модель;
5. Пользователь выбирает необходимые ему уязвимости;
6. Объектная модель программы передается обработчикам выбранных уязвимостей, происходит анализ;
7. Вся информация о найденных уязвимостях обрабатывается, агрегируется и отображается в окне пользовательского интерфейса.
8. Разработка программы

В рамках разработки программы передо мной были поставлены следующие задачи:

1. Обработчик переполнения буфера;
2. Обработчик пренебрежения обработкой ошибок;
3. Обработчик состояния гонки;
4. Обработчик задачи Читатели – Писатели;
5. Модуль первичного лексического анализа кода;
6. Агрегирующий модуль;
7. Модуль тестирования и генерации тестового кода С++.

3.1 Обработчик уязвимости «Переполнения буфера»

Уязвимость переполнения буфера возникает, например, при получении данных извне и дальнейшем копировании их во внутренний буфер без проверки размера копируемых данных. При необходимом объеме записываемых данных адрес возврата, находящийся в стеке, может быть перезаписан. При этом на его место можно перезаписать необходимый адрес, который указывает на вредоносный код.

Поиском и анализом данной уязвимости занимается класс «BufferOverflowHandler», листинг которого представлен на рисунке 3. Как и каждый из всех обработчиков, подчиненных одному интерфейсу, класс «BufferOverflowHandler» имеет функцию «parse», принимающую на вход объектную модель анализируемой программы, построенной в модуле первичной лексической обработки.

В начале функция ищет среди всех переменных предоставленной объектный модели переменные, являющиеся массивами или указателями, так как они потенциально могут быть использованы в будущем в качестве буферов.

from typing import List  
from core import main\_code\_parser  
from core.base\_handler import BaseHandler  
from core.variable import \*  
from core.function\_context import FunctionContext  
  
  
class BufferOverflowHandler(BaseHandler):  
  
 vulnerability\_name = 'Переполнение буфера'  
  
 """  
 - char \* strcpy ( char \* destination, const char \* source ); - Копирует строку <source> в буфер <destination>.  
   
 - int printf ( const char \* format, ... ); - Выводит строку указанную в формате в стандартный поток вывода.  
   
 - char \* strcat ( char \* destination, const char \* source ); - Добавляет к строке <destination> копию <source>  
   
 - void \* memcpy ( void \* destination, const void \* source, size\_t num ); - копирует <num> байт из <source> в   
 <destination>  
   
 - char \* gets ( char \* str ); - читает символы из потока ввода и записывает их в <str>  
   
 - int sprintf ( char \* str, const char \* format, ... ); - аналогично <printf>, но вывод происходит в буфер <str>  
   
 - int vsprintf (char \* s, const char \* format, va\_list arg ); - аналогично <sprintf>, но принимает только один <arg>  
   
 - char \* strncpy ( char \* destination, const char \* source, size\_t num ); - <strcpy>, но копирует только <num> байт  
 """  
  
 def \_\_init\_\_(self):  
 self.pattern = r"(strcpy|printf|strcat|memcpy|gets|sprintf|vsprintf|strncpy|scanf)" \  
 r"\(.\*\)"  
 self.output = []  
  
 def parse(self, contexts: List[FunctionContext]):  
 total\_errors = 0  
 for context in contexts:  
 declared\_variables = context.variables  
 for line\_number, line in context.source\_code.items():  
 matches = re.finditer(self.pattern, line)  
 for match in matches:  
 used\_variables = main\_code\_parser.get\_parameters(match.group(0), declared\_variables)  
 for used\_variable in used\_variables:  
 declaration = used\_variable.full\_declaration  
 if is\_pointer(declaration) or is\_array(declaration):  
 total\_errors += 1  
 self.output.append(  
 f"{total\_errors}) Предупреждение в методе <{context.name}>!\n"  
 f"Использование буфера <{declaration[:-1]}> (строка {used\_variable.line\_appeared}) "  
 f"в небезопасной функции <{match.group(1)}> (строка {line\_number}).\n"  
 f"Это может стать причиной переполнения буфера. "  
 f"Убедитесь в наличии проверки этой угрозы!\n")  
 self.output.append(self.vulnerability\_name + ": " + str(total\_errors))  
 return self.output

Рисунок 3 – класс «BufferOverflowHandler»

Затем с помощью регулярных выражений ведется поиск всех функций, которые могут использовать буферы в качестве параметров и приводить их к переполнению. Ведется поиск следующих функций:

1. char \* strcpy (char \* destination, const char \* source) - Копирует строку <source> в буфер <destination>;
2. int printf (const char \* format, ...) - Выводит строку указанную в формате в стандартный поток вывода;
3. char \* strcat (char \* destination, const char \* source) - Добавляет к строке <destination> копию <source>;
4. void \* memcpy (void \* destination, const void \* source, size\_t num) - копирует <num> байт из <source> в <destination>;
5. char \* gets (char \* str) - читает символы из потока ввода и записывает их в <str>;
6. int sprintf (char \* str, const char \* format, ...) - аналогично <printf>, но вывод происходит в буфер <str>;
7. int vsprintf (char \* s, const char \* format, va\_list arg) - аналогично <sprintf>, но принимает только один <arg>;
8. char \* strncpy (char \* destination, const char \* source, size\_t num) - <strcpy>, но копирует только <num> байт.

Предупреждение о возможном переполнении буфера происходит в том случае, если найдена функция из списка, и в качестве параметра она использует один из раннее обнаруженных буферов.

Для проверки работы обработчика были составлены и успешно пройдены тесты.

**Тесты Ковалева К.А. на уязвимость «Переполнение буфера»**

В тесте на рисунке 4 содержится четыре потенциальные уязвимости переполнения буфера. На рисунке 5 приведен результат работы программы.

1. На 12 строке в функции «strcpy» используется буфер «param\_1»;
2. На 13 строке в функции «memcpy» используется буфер «param\_2»;
3. На 20 строке в функции «strcpy» используется буфер «c»;
4. На 20 строке в функции «strcpy» используется буфер «arg [3]».

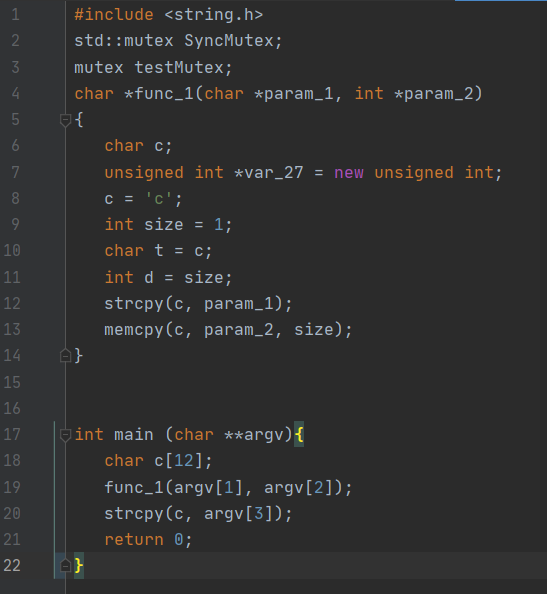


Рисунок 4 – тест Ковалева К.А. на переполнение буфера

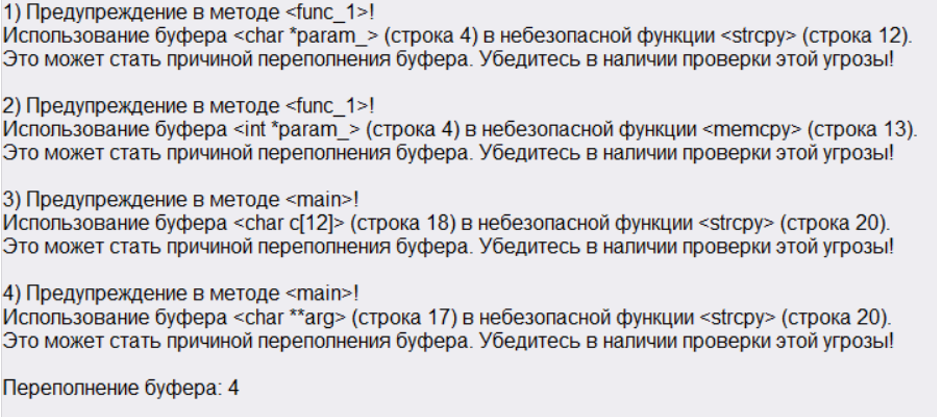
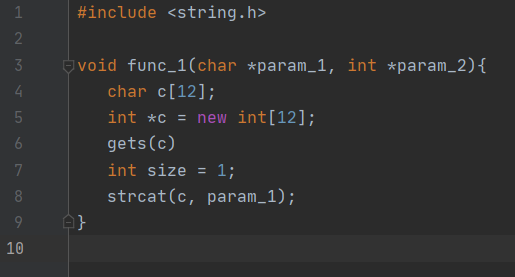


Рисунок 5 – результат работы программы

В тесте на рисунке 6 содержится три потенциальные уязвимости переполнения буфера. На рисунке 7 приведен результат работы программы.

1. На 6 строке в функции «gets» используется буфер «c»;
2. На 8 строке в функции «strcat» используется буфер «c»;
3. На 8 строке в функции «strcat» используется буфер «param\_1».



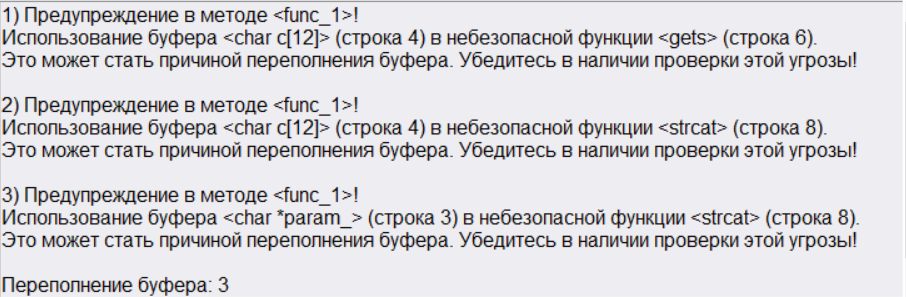
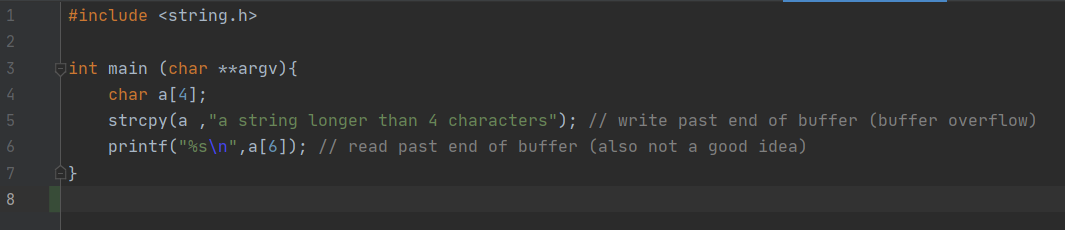
Рисунок 6 - тест Ковалева К.А. на переполнение буфера

Рисунок 7 – результат работы программы

**Тесты Терентьева Е.А. на уязвимость «Переполнение буфера»**

В тесте на рисунке 8 содержится две потенциальные уязвимости переполнения буфера. На рисунке 9 приведен результат работы программы.

1. На 5 строке в функции «strcpy» используется буфер «a»;
2. На 6 строке в функции «printf» используется буфер «a».

Рисунок 8 - тест Терентьева Е.А. на переполнение буфера 

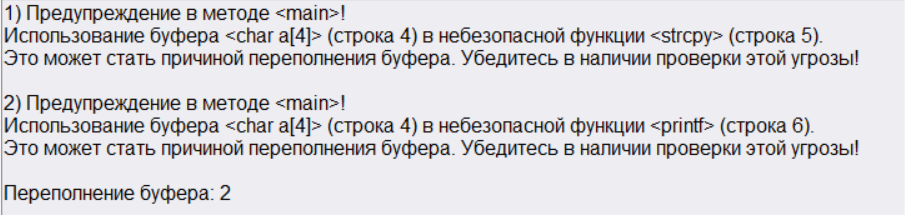


Рисунок 9 – результат работы программы

В тесте на рисунке 10 содержится пять потенциальных уязвимостей переполнения буфера. На рисунке 11 приведен результат работы программы.

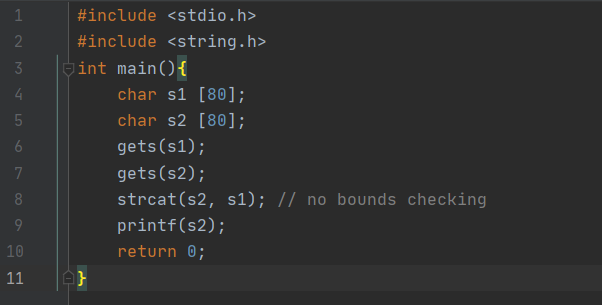
1. На 6 строке в функции «gets» используется буфер «s1»;
2. На 7 строке в функции «gets» используется буфер «s1»;
3. На 8 строке в функции «strcat» используется буфер «s2»;
4. На 8 строке в функции «strcat» используется буфер «s1»;
5. На 9 строке в функции «printf» используется буфер «s2».

Рисунок 10 - тест Терентьева Е.А. на переполнение буфера

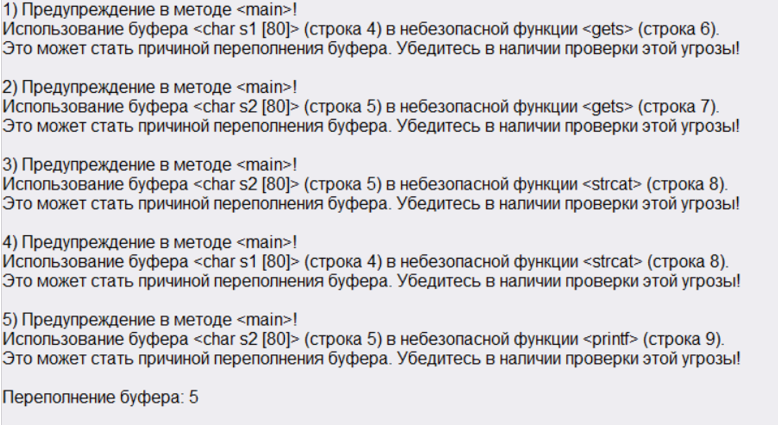


Рисунок 11 – результат работы программы

3.2 Обработчик уязвимости «Пренебрежения обработкой ошибок»

Безопасность подвергается серьезной угрозе, когда не происходит должного уделения внимания обработке ошибок. Опасность заключается в том, что в некоторых ситуациях программа может оказаться в некорректном состоянии, что ведет к отказу обслуживания, а, следовательно, создает угрозу безопасности.

Поиском и анализом данной уязвимости занимается класс «EmptyCatchHandler», листинг которого представлен на рисунке 12. Как и каждый из всех обработчиков, подчиненных одному интерфейсу, класс «EmptyCatchHandler» имеет функцию «parse», принимающую на вход объектную модель анализируемой программы, построенной в модуле первичной лексической обработки.

В начале функция ищет объявление блока «try-catch», затем переходит в функцию «analyze\_catch», которая предназначена для поиска вложенных конструкций «try-catch». Анализируя каждый блок, функция выдает предупреждение, если обнаружен пустой блок «catch», то есть не происходит обработки ошибок.

import re  
from typing import List  
from core.base\_handler import BaseHandler  
from core.function\_context import FunctionContext  
  
  
def is\_empty(catch\_body: List[str]):  
 for line in catch\_body:  
 tmp = line.replace("", "")  
 if tmp == "": continue  
 else: return False  
 return True  
  
  
class EmptyCatchHandler(BaseHandler):  
 vulnerability\_name = 'Пренебрежение обработкой исключений'  
  
 def \_\_init\_\_(self):  
 self.pattern = r'catch\s\*' \  
 r'\(.\*\)\s\*\{'  
 self.output = []  
 self.total\_errors = 0  
  
 def analyze\_catch(self, cur\_line\_number, global\_line\_number, context):  
 body = []  
 close\_br = 0  
 open\_br = 1  
 dict\_lines = context.source\_code  
 while cur\_line\_number < len(dict\_lines):  
 cur\_line = list(dict\_lines.items())[cur\_line\_number][1]  
 if re.match(self.pattern, cur\_line) is not None:  
 return self.analyze\_catch(cur\_line\_number + 1, global\_line\_number + 1, context)  
  
 if re.match(r'}', cur\_line) is None:  
 if re.match(r'.\*{', cur\_line) is not None:  
 open\_br += 1  
 body.append(cur\_line)  
  
 if re.match(r'}', cur\_line):  
 close\_br += 1  
 if open\_br != close\_br:  
 body.append(cur\_line)  
 elif is\_empty(body):  
 self.total\_errors += 1  
 self.output.append(f"{self.total\_errors}) Предупреждение в методе <{context.name}>!\n"  
 f"Отстутсвует обработка исключения или ошибки! (строка {global\_line\_number - len(body)})\n")  
 return cur\_line\_number  
 elif not is\_empty(body):  
 return cur\_line\_number  
 global\_line\_number += 1  
 cur\_line\_number += 1  
  
 def parse(self, contexts: List[FunctionContext]):  
 index = 0  
 prev\_index = 0  
 for context in contexts:  
 prev\_index += index  
 index = 0  
 while index < len(context.source\_code):  
 processed\_line = list(context.source\_code.items())[index][1]  
 if re.match(self.pattern, processed\_line) is not None:  
 index = self.analyze\_catch(index + 1, index + prev\_index + 1, context)  
 else:  
 index += 1  
 self.output.append(self.vulnerability\_name + ": " + str(self.total\_errors))  
 self.total\_errors = 0  
 return self.output

Рисунок 12 – Класс EmptyCatchHandler

Для проверки работы обработчика были составлены и успешно пройдены тесты.

**Тесты Ковалева К.А. на уязвимость «Пренебрежение обработкой ошибок»**

В тесте на рисунке 13 содержится одна потенциальная уязвимость пренебрежения обработкой исключений. На рисунке 14 приведен результат работы программы.

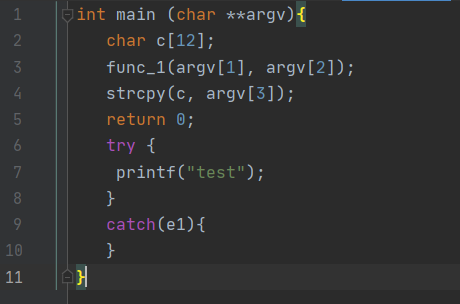
1. На 9 строке объявлен пустой блок «catch».

Рисунок 13 – тест Ковалева К.А. на пренебрежение обработкой исключений

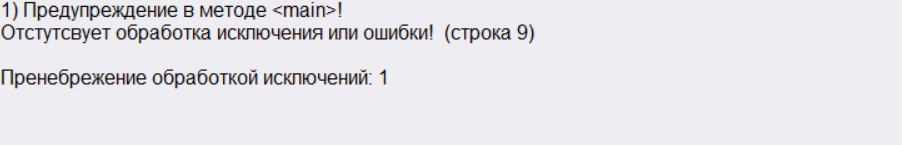


Рисунок 14 – результат работы программы

В тесте на рисунке 15 содержится две потенциальные уязвимости пренебрежения обработкой исключений. В этом примере демонстрируется поиск вложенных блоков «try-catch». На рисунке 16 приведен результат работы программы.

1. На 9 строке объявлен пустой блок «catch»;
2. На строке 15 объявлен пустой блок «catch».

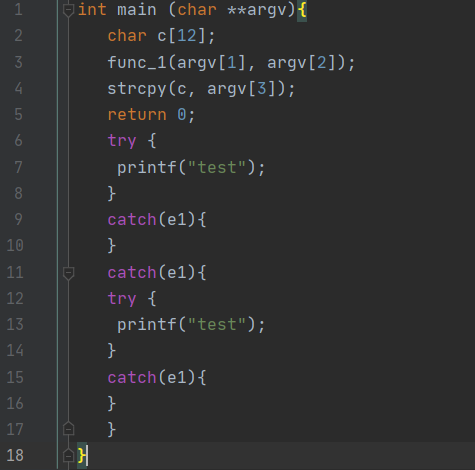


Рисунок 15 – тест Ковалева К.А. на пренебрежение обработкой исключений

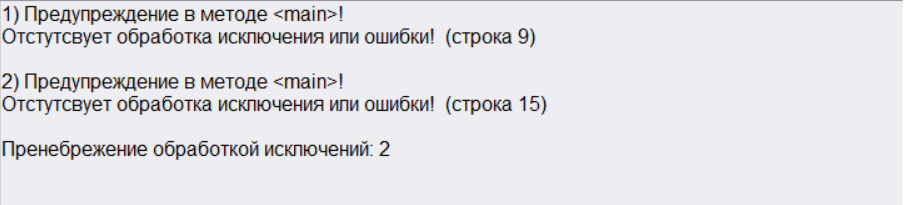


Рисунок 16 – результат работы программы

**Тесты Терентьева Е.А. на уязвимость «Пренебрежение обработкой ошибок»**

В тесте на рисунке 17 содержится одна потенциальная уязвимость пренебрежения обработкой исключений. На рисунке 18 приведен результат работы программы.

1. На 8 строке объявлен пустой блок «catch».

Рисунок 17 – тест Терентьева Е.А. на пренебрежение обработкой исключений

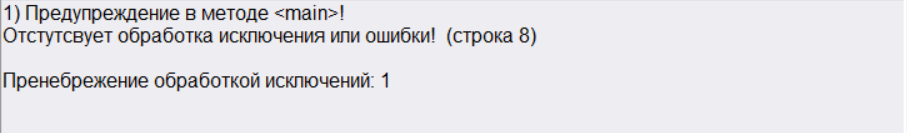


Рисунок 18 – результат работы программы

В тесте на рисунке 19 содержится три потенциальные уязвимости пренебрежения обработкой исключений. На рисунке 20 приведен результат работы программы.

1. На 5 строке объявлен пустой блок «catch»;
2. На 11 строке объявлен пустой блок «catch»;
3. На 18 строке объявлен пустой блок «catch».

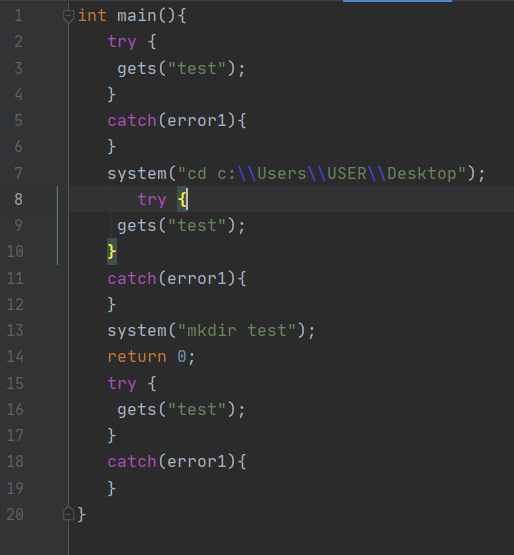


Рисунок 19 – тест Терентьева Е.А. на пренебрежение обработкой исключений

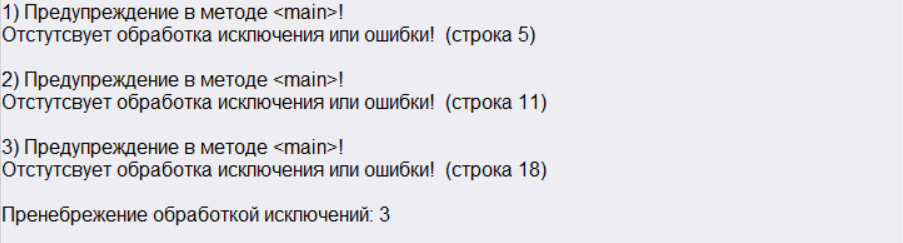


Рисунок 20 – результат работы программы

3.3 Обработчик уязвимости «Состояние гонки»

Уязвимость, при которой работа системы или приложения зависит от того, в каком порядке выполняются части кода. Состояние гонки возникает тогда, когда несколько потоков многопоточного приложения пытаются одновременно получить доступ к одним и тем же ресурсам, причем хотя бы один поток выполняет запись. Кроме того, ошибки такого рода невероятно сложно воспроизвести повторно.

Поиском и анализом данной уязвимости занимается класс «RaceConditionHandler», листинг которого представлен на рисунке 21. Как и каждый из всех обработчиков, подчиненных одному интерфейсу, класс «RaceConditionHandler» имеет функцию «parse», принимающую на вход объектную модель анализируемой программы, построенной в модуле первичной лексической обработки.

В начале функция получает из объектной модели все объявленные потоки. Затем получает список всех исполняемых функций этих потоков. Предупреждение появляется в том случае, если у двух и более потоков используется одна и та же исполняемая функция.

from typing import List  
from core.base\_handler import BaseHandler  
from core.function\_context import FunctionContext  
  
  
class RaceConditionHandler(BaseHandler):  
 vulnerability\_name = 'Состояние гонки'  
  
 def \_\_init\_\_(self):  
 self.output = []  
  
 def parse(self, contexts: List[FunctionContext]):  
 total\_errors = 0  
 for context in contexts:  
 if len(context.threads) != 0:  
 declared\_threads = context.threads  
 # part for detecting same runnables  
 funcs\_to\_threads = [[], []]  
 func\_in\_threads = []  
 # creating list of all runnables  
 for thread in declared\_threads:  
 if thread.runnable\_function not in func\_in\_threads:  
 func\_in\_threads.append(thread.runnable\_function)  
 # filling list of func - threads  
 for index, func in enumerate(func\_in\_threads):  
 for thread in declared\_threads:  
 if func == thread.runnable\_function:  
 funcs\_to\_threads[index].append(thread)  
  
 # checking where function is used more than one time as runnable  
 for func\_usage in funcs\_to\_threads:  
 if len(func\_usage) > 1:  
 total\_errors += 1  
 warning = f"{total\_errors}) Предупреждение в методе <{context.name}>!\n"  
 warning += "Потоки:\n"  
 cur\_func = func\_usage[0].runnable\_function  
 for thread in func\_usage:  
 warning += f"\"<{thread.thread\_name}> (строка {thread.line\_appeared})\"\n"  
 warning += f"используют одну и туже исполняемую функцию <{cur\_func}>, " \  
 f"это может вызвать состояние гонки!\n"  
 self.output.append(warning)  
 self.output.append(self.vulnerability\_name + ": " + str(total\_errors))  
 return self.output

Рисунок 21 – класс RaceConditionHandler

Для проверки работы обработчика были составлены и успешно пройдены тесты.

**Тесты Ковалева К.А. на уязвимость «Состояние гонки»**

В тесте на рисунке 22 содержится одна потенциальная уязвимость состояния гонки. На рисунке 23 приведен результат работы программы.

1. Объявленные потоки t1, t2, t3 используют одну и ту же исполняемую функцию «function\_9».

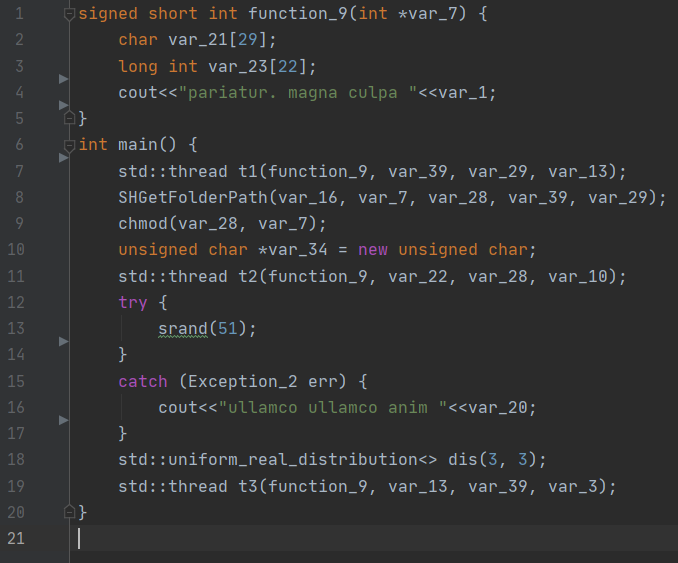


Рисунок 22 – тест Ковалева К.А. на состояние гонки

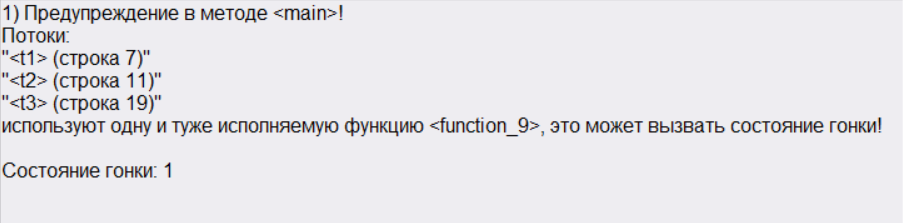


Рисунок 23 – результат работы программы

В тесте на рисунке 24 содержится одна потенциальная уязвимость состояния гонки. На рисунке 25 приведен результат работы программы.

1. Объявленные потоки t0, t1, t2 используют одну и ту же исполняемую функцию «function\_19».

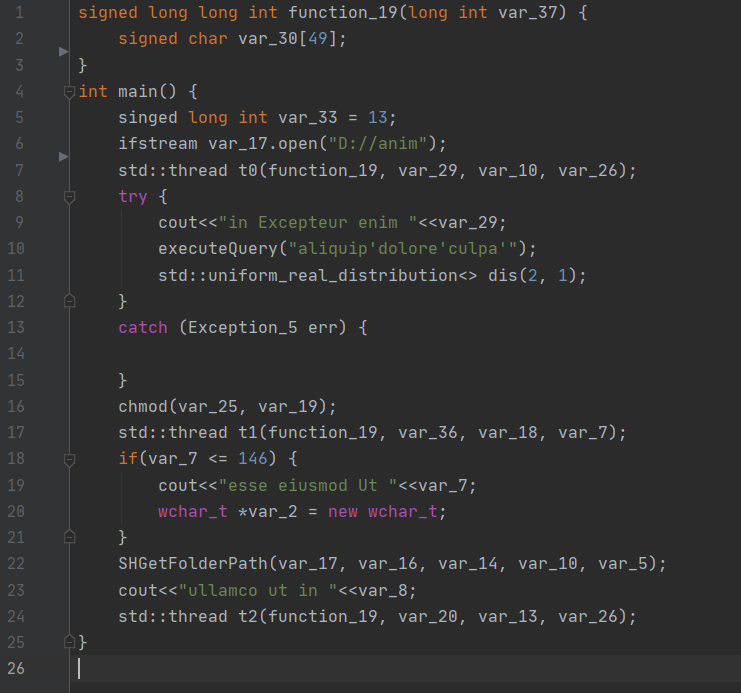


Рисунок 24 – тест Ковалева К.А. на состояние гонки

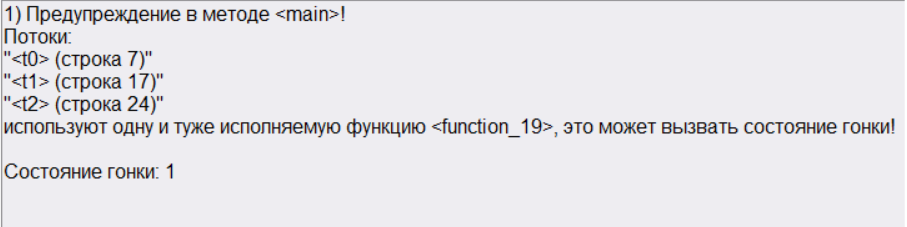


Рисунок 25 – результат работы программы

**Тесты Терентьева Е.А. на уязвимость «Состояние гонки»**

В тесте на рисунке 26 содержится две потенциальные уязвимости состояния гонки. На рисунке 27 приведен результат работы программы.

1. Объявленные потоки t0, t4, t5 используют одну и ту же исполняемую функцию «function3»;
2. Объявленные потоки t1, t2 используют одну и ту же исполняемую функцию «function4»;

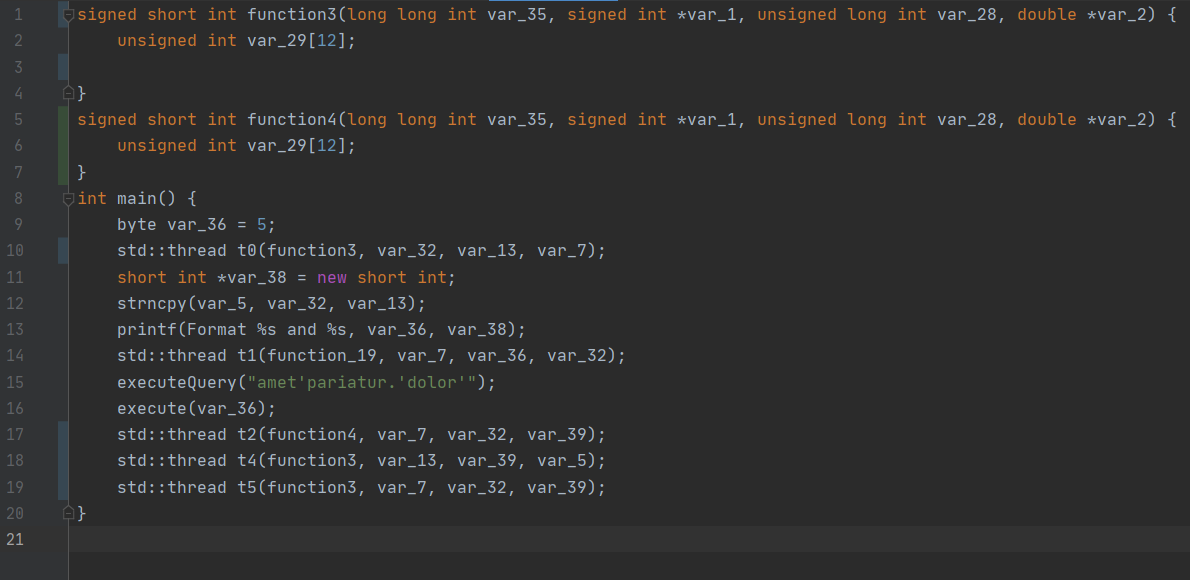


Рисунок 26 – тест Терентьева Е.А. на состояние гонки

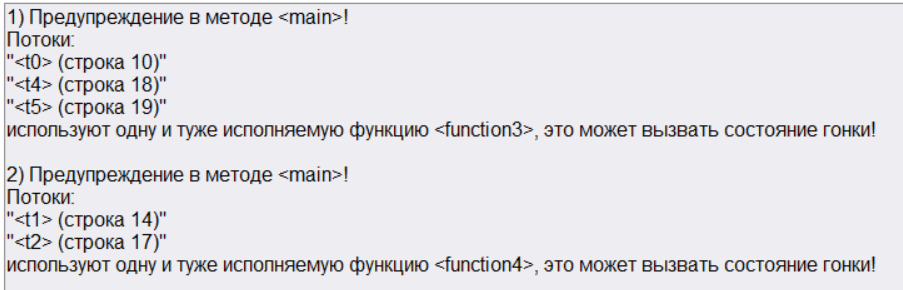


Рисунок 27 – результат работы программы

В тесте на рисунке 28 содержится одна потенциальная уязвимость состояния гонки. На рисунке 29 приведен результат работы программы.

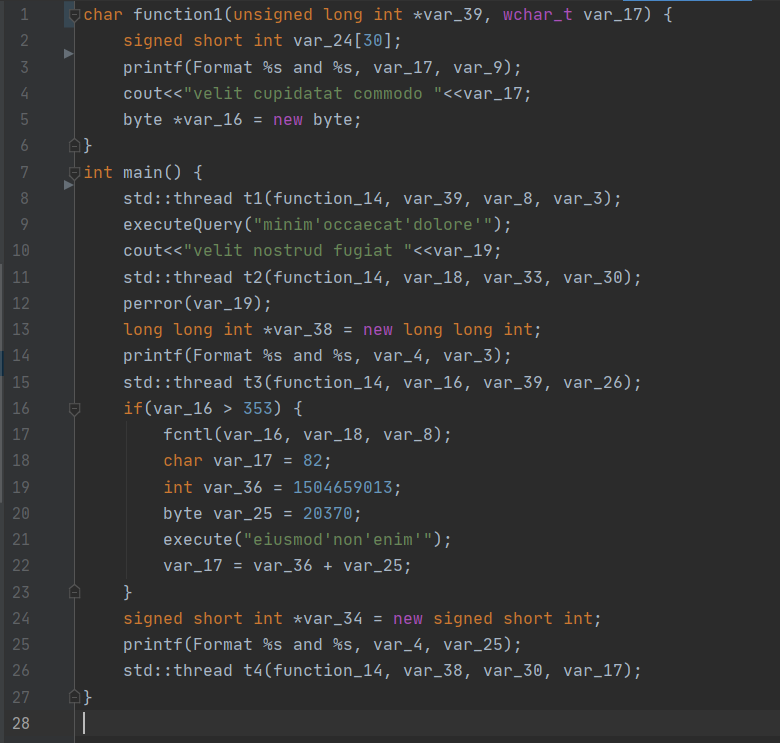
1. Объявленные потоки t1, t2, t3, t4 используют одну и ту же исполняемую функцию «function\_14».

Рисунок 28 – тест Терентьева Е.А. на состояние гонки

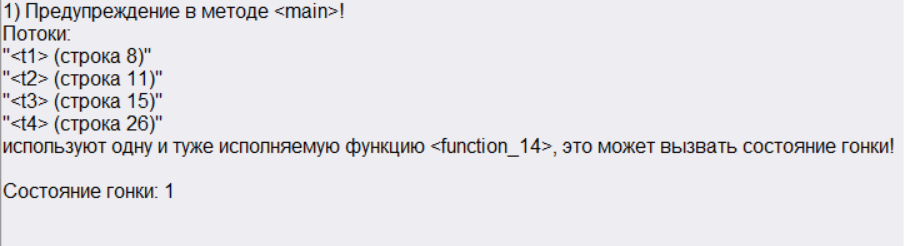


Рисунок 29 – результат работы программы

3.4 Обработчик уязвимости «Задача читатели-писатели»

Данная уязвимость возникает при одновременном доступе нескольких потоков к одной и той же области памяти, допускающей чтение и запись. При этом одновременно могут читать сколько угодно потоков, но писать — только один. При одновременном чтении проблем не возникает, но изменение памяти во время ее чтения некоторым потоком может привести к серьезным ошибкам.

Поиском и анализом данной уязвимости занимается класс «ReadersWritersHandler», листинг которого представлен на рисунке 30. Как и каждый из всех обработчиков, подчиненных одному интерфейсу, класс «ReadersWritersHandler» имеет функцию «parse», принимающую на вход объектную модель анализируемой программы, построенной в модуле первичной лексической обработки.

В начале функция получает из объектной модели все объявленные потоки. Затем получает список всех используемых параметров этих потоков. Предупреждение появляется в том случае, если у двух и более потоков используются одинаковые параметры. Помимо этого, проверяется есть ли у двух и более потоков с одинаковыми парметрами в исполняемой функции блок синхронизации (семафор). Если блока синхронизации нет, выдается предупреждение, если есть, то проверяется освобождается ли блок синхронизации в исполняемой функции. Если он не освобождается, то также выдается предупреждение.

import re  
from typing import List  
  
from core.base\_handler import BaseHandler  
from core.function\_context import FunctionContext  
  
  
class ReadersWritersHandler(BaseHandler):  
 vulnerability\_name = 'Читатели Писатели'  
  
 def \_\_init\_\_(self):  
 self.output = []  
 self.total\_errors = 0  
  
 def check\_mutex(self, context: FunctionContext, thread):  
 mutex\_cnt = len(FunctionContext.global\_mutexes)  
 mutex\_locked = [0 for \_ in range(mutex\_cnt)]  
 mutex\_unlocked = [0 for \_ in range(mutex\_cnt)]  
 context\_beginning = None  
 for line\_number, line in context.source\_code.items():  
 context\_beginning = line\_number + 1  
 break  
 for line\_number, line in context.source\_code.items():  
 for index, mutex in enumerate(FunctionContext.global\_mutexes):  
 if re.match(r'' + mutex + r'\.lock\s\*\(.\*\);', line):  
 mutex\_locked[index] += 1  
 continue  
 for index, mutex in enumerate(FunctionContext.global\_mutexes):  
 if re.match(r'' + mutex + r'\.unlock\s\*\(.\*\);', line):  
 mutex\_unlocked[index] += 1  
 for index in range(mutex\_cnt):  
 if mutex\_locked[index] == mutex\_unlocked[index]:  
 continue  
 elif mutex\_locked[index] > mutex\_unlocked[index]:  
 self.total\_errors += 1  
 return f"{self.total\_errors - 1}) Отсутствует освобождение блока синхронизации " \  
 f"(семафор - {FunctionContext.global\_mutexes[index]}) в исполняемой функции <{context.name}> " \  
 f"(строка {context\_beginning}) потока <{thread.thread\_name}>"  
 elif mutex\_locked[index] < mutex\_unlocked[index]:  
 self.total\_errors += 1  
 return f"{self.total\_errors - 1}) Отсутстввует захват блока синхронизации " \  
 f"(семафор - {FunctionContext.global\_mutexes[index]}) в исполняемой функции <{context.name}> " \  
 f"(строка {context\_beginning}) потока <{thread.thread\_name}>"  
 is\_no\_mutex = True  
 for index, \_ in enumerate(mutex\_locked):  
 if mutex\_locked[index] != 0 and mutex\_unlocked[index] != 0:  
 is\_no\_mutex = False  
 if is\_no\_mutex:  
 self.total\_errors += 1  
 return f"{self.total\_errors - 1}) Отсутстввует блок синхронизации " \  
 f"в исполняемой функции <{context.name}> (строка {context\_beginning}) потока <{thread.thread\_name}>"  
  
 return ""  
  
 def parse(self, contexts: List[FunctionContext]):  
 for context in contexts:  
 if len(context.threads) != 0:  
 declared\_threads = context.threads  
 # part for detecting same variables  
 vars\_to\_threads = [] # list of dicts - List[{var\_name: List[Threads]}]  
 var\_in\_threads = [] # list of variables - List[Variable]  
 # creating list of all used params  
 for thread in declared\_threads:  
 for parameter in thread.parameter\_list:  
 if parameter not in var\_in\_threads:  
 var\_in\_threads.append(parameter)  
 # filling list of params - threads  
 for index, var in enumerate(var\_in\_threads):  
 t\_list = []  
 dict\_tmp = {var.var\_name: t\_list}  
 vars\_to\_threads.append(dict\_tmp)  
 for thread in declared\_threads:  
 for parameter in thread.parameter\_list:  
 if var == parameter:  
 vars\_to\_threads[index][var.var\_name].append(thread)  
 # checking where same parameters has usage in several threads  
 thread\_with\_same\_vars = []  
 for var\_usage in vars\_to\_threads:  
 warning = ""  
 for key in var\_usage:  
 if len(var\_usage[key]) > 1:  
 self.total\_errors += 1  
 warning = "---Анализ параметров потоков---\n"  
 warning += f"{self.total\_errors}) Предупреждение в методе <{context.name}>!\n"  
 warning += "Потоки:\n"  
 for thread in var\_usage[key]:  
 warning += f"\"<{thread.thread\_name}> (строка {thread.line\_appeared})\"\n"  
 warning += f"используют одну и ту же переменную <{key}> " \  
 f"в качестве параметра для исполняемой функции, " \  
 f"это может привести к проблеме Читатели - Писатели!\n"  
 self.output.append(warning)  
 self.output.append("---Анализ объектов синхронизации в исполняемых функциях потоков, "  
 "использующих одни и теже переменные---\n")  
 thread\_list = list(var\_usage.items())[0][1]  
 for thread in thread\_list:  
 for c in contexts:  
 if c.name == thread.runnable\_function:  
 self.output.append(self.check\_mutex(c, thread))  
 self.output.append("\n")  
 self.output.append(self.vulnerability\_name + ": " + str(self.total\_errors))  
 self.total\_errors = 0  
 return self.output

Рисунок 30 – класс ReadersWritersHandler

Для проверки работы обработчика были составлены и успешно пройдены тесты.

**Тесты Ковалева К.А. на уязвимость «Задача читатели-писатели»**

В тесте на рисунке 31 содержится три потенциальных уязвимости задачи читатели-писатели. На рисунке 32 приведен результат работы программы.

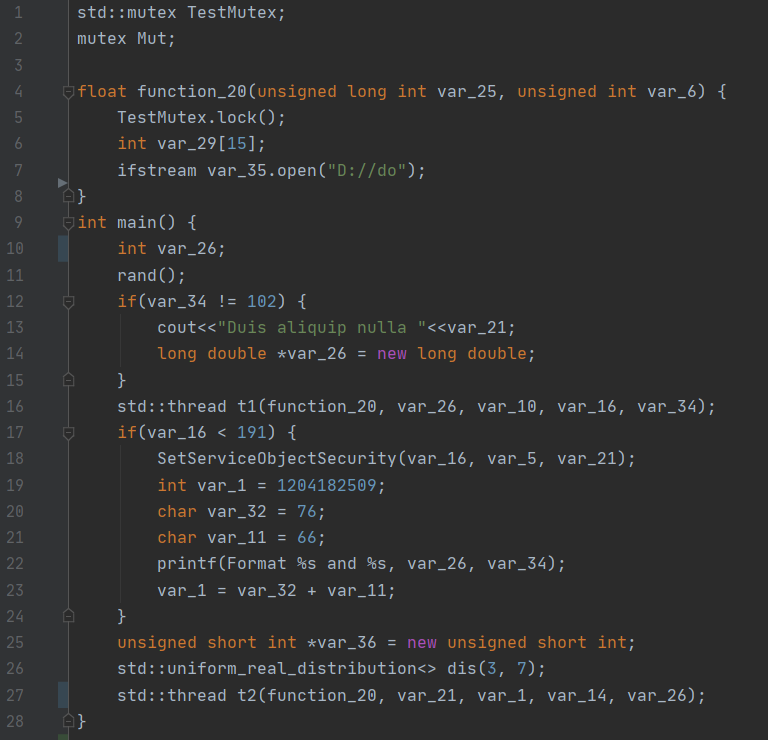
1. Объявленные потоки t1, t2 используют одну и ту же переменную «var\_26»;
2. В исполняемой функции «function20» потока t1, объявленный мьютекс «TestMutex» не освобождается;
3. В исполняемой функции «function20» потока t2, объявленный мьютекс «TestMutex» не освобождается;

Рисунок 31 – тест Ковалева К.А. на задачу читатели-писатели

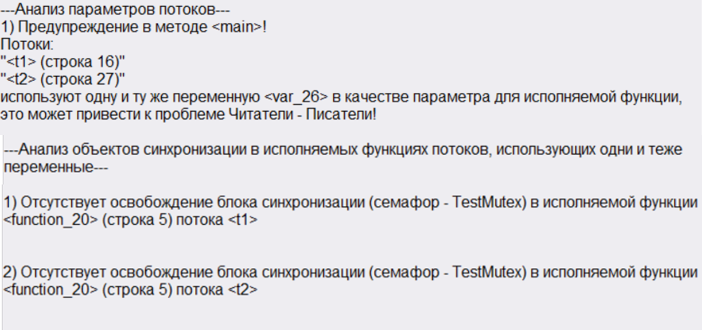


Рисунок 32 – результат работы программы

В тесте на рисунке 33 содержится одна потенциальная уязвимость задачи читатели-писатели. На рисунке 34 приведен результат работы программы.

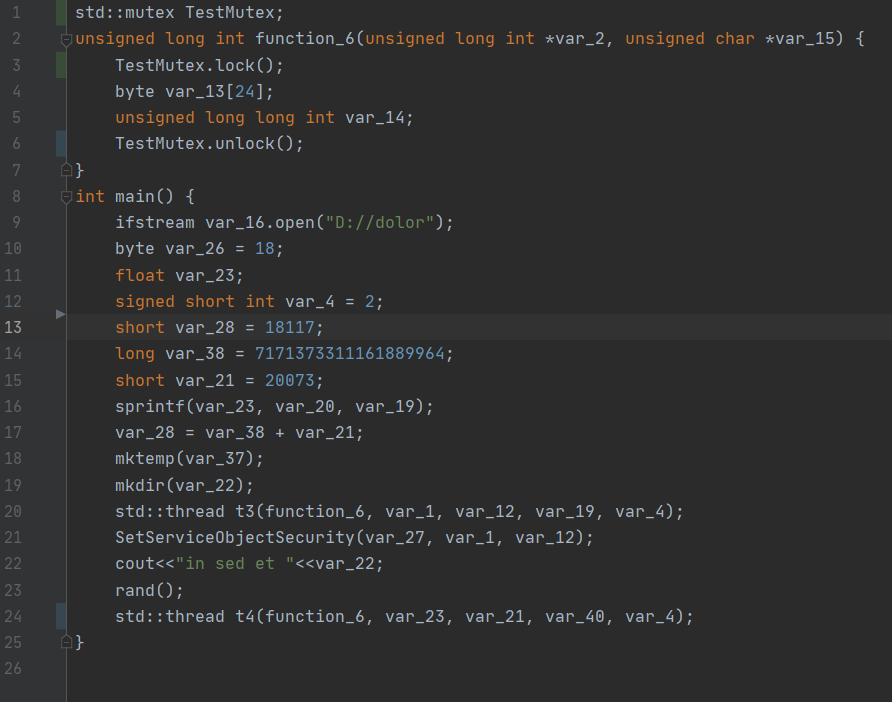
1. Объявленные потоки t4, t3 используют один и тот же параметр «var\_4».

Рисунок 33 – тест Ковалева К.А. на задачу читатели-писатели

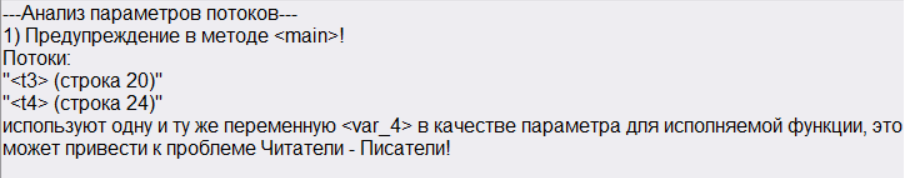


Рисунок 34 – результат работы программы

**Тесты Терентьева Е.А. на уязвимость «Задача читатели-писатели».**

В тесте на рисунке 35 содержится три потенциальные уязвимость задачи читатели-писатели. На рисунке 36 приведен результат работы программы.

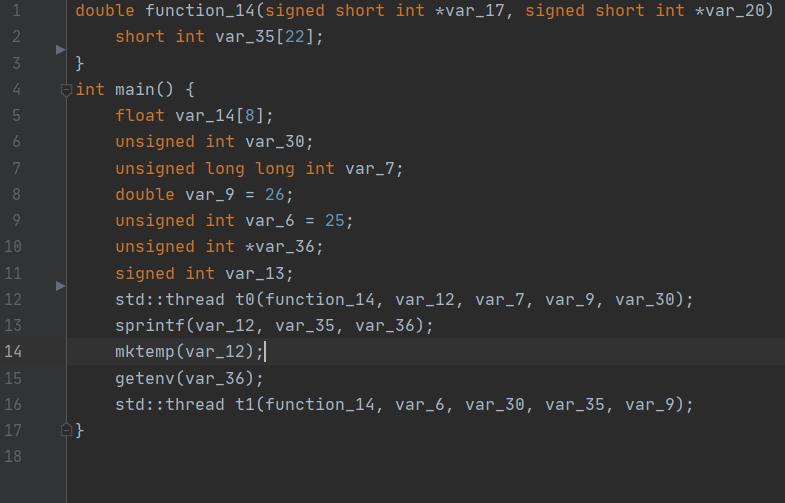
1. Объявленные потоки t0, t1 используют один и тот же параметр «var\_9»;
2. В исполняемой функции «function\_9» потока t1 отсутствует блок синхронизации;
3. В исполняемой функции «function9» потока t0 отсутствует блок синхронизации.

Рисунок 35 – тест Тереньтьева Е.А. на задачу читатели-писатели

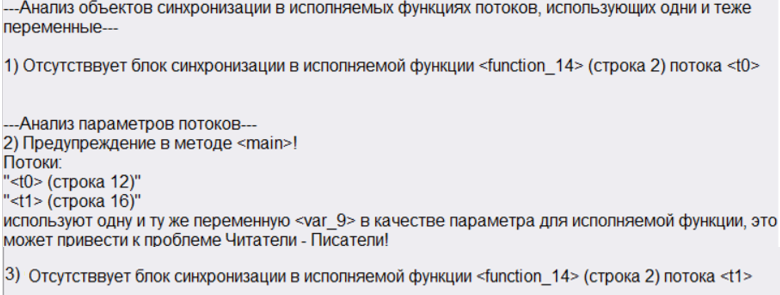


Рисунок 36 – результат работы программы

В тесте на рисунке 37 содержится одна потенциальная уязвимость задачи читатели-писатели. На рисунке 38 приведен результат работы программы.

1. Объявленные потоки t1, t0 используют один и тот же параметр «var\_25».

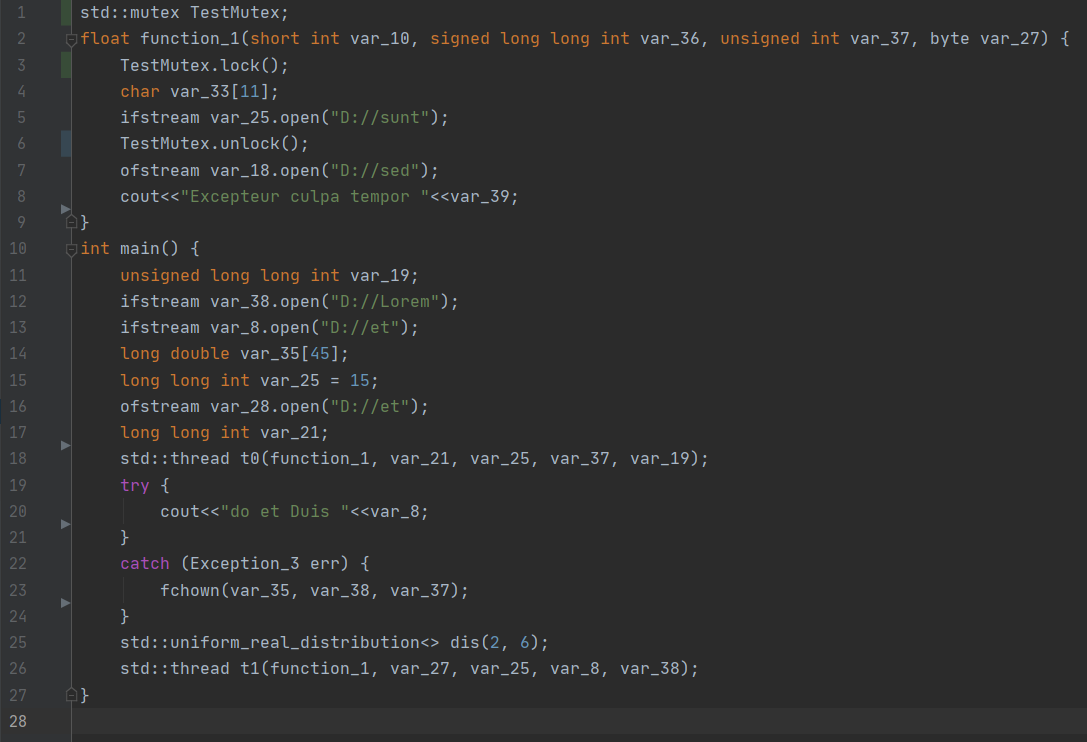


Рисунок 37 – тест Терентьева Е.А. на задачу читатели-писатели

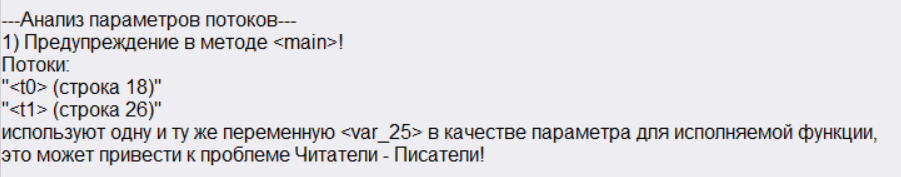


Рисунок 38 – результат работы программы

* 1. Модуль первичного лексического анализа кода

Данный модуль отвечает за формирование объектной модели анализируемой программы, которая в дальнейшем используется всеми обработчиками при поиске уязвимостей.

Объектная модель представляет из себя список таких структур данных как «FunctionContext» (рисунок 39).

from collections import OrderedDict  
from typing import List  
from core.variable import Variable  
  
  
class FunctionContext:  
 global\_mutexes = []  
  
 def \_\_init\_\_(self, return\_type, name, parameters):  
 self.return\_type = return\_type  
 self.name = name  
 self.parameters: List[Variable] = parameters  
 self.variables: List[Variable] = []  
 self.threads = []  
 self.source\_code = OrderedDict()

Рисунок 39 – класс FunctionContext

Этот класс является объектным представлением функции. Его поля содержат информацию о возвращаемом значении функции, ее имени, параметрах, всех переменных и потоках, объявленных в области ее видимости. Также класс хранит все строки коды, занимаемые функцией.

Как я отметил выше, в FunctionContext хранится ифномарция о параметрах, переменных и потоках в ее области видимости. Все они также представлены в объектном виде.

Класс «Variable» (рисунок 40) отвечает за представление переменных или параметров. Он хранит полную декларацию переменной, ее имя, тип, значение и номер строки, на которой она была объявлена. Также в классе есть две вспомогательные функции, позволяющие определить является ли переменная указателем или массивом.

import re  
  
  
def is\_pointer(variable\_declaration):  
 return True if re.search(r"\\*", variable\_declaration) else False  
  
  
def is\_array(variable\_declaration):  
 return True if re.search(r"\[.\*]", variable\_declaration) else False  
  
  
class Variable:  
 def \_\_init\_\_(self, full\_declaration, line\_appeared, var\_name, var\_type, value=None):  
 self.full\_declaration = full\_declaration  
 self.line\_appeared = line\_appeared  
 self.var\_name = var\_name  
 self.var\_type = var\_type  
 self.value = value

Рисунок 40 – класс Variable

Класс «Thread» (рисунок 41) отвечает за представление потоков. Он хранит полную декларацию потока, его имя, имя исполняемой функции, список параметров, а также номер строки, на котором он был объявлен.

from typing import List  
  
from core.variable import Variable  
  
  
class Thread:  
 def \_\_init\_\_(self, full\_declaration, line\_appeared, thread\_name, runnable\_function, parameter\_list: List[Variable]):  
 self.full\_declaration = full\_declaration  
 self.line\_appeared = line\_appeared  
 self.thread\_name = thread\_name  
 self.runnable\_function = runnable\_function  
 self.parameter\_list = parameter\_list

Рисунок 41 – класс Thread

Для наполнения информацией представленных объектов используется python-модуль «main\_code\_parser.py» (рисунок 42)

import re  
from typing import List  
from core import regexs\_constants as rc  
from core.function\_context import FunctionContext  
from core.thread import Thread  
from core.variable import Variable  
  
  
def find\_contexts(source\_code):  
 *"""  
 Метод для поиска и анализа контекстов в текущем файле.  
 Контексты в данном случае - глобальные лексемы языка, такие как функции, структуры и классы.  
 В данный момент реализован поиск функций.  
 Изначально ведется разбиение всего кода на контексты, а затем анализ каждого из них, например, на наличие  
 объявленных переменных поток и т.д.* ***:rtype*** *List[FunctionContext]  
 """* func\_declarations = get\_func\_declarations(source\_code)  
 for func in func\_declarations:  
 func.variables += func.parameters  
 func.variables += get\_declared\_variables(func.source\_code)  
 func.threads = get\_declared\_threads(func.source\_code, func.variables)  
 return func\_declarations  
  
  
def get\_func\_declarations(source\_code):  
 *"""  
 Поиск сигнатур функций и формирование из них объектов.* ***:rtype*** *List[FunctionContext]  
 """* func\_declarations = []  
 cur\_func = None  
 open\_br = 0  
 close\_br = 0  
  
 for line\_number, line in enumerate(source\_code):  
 line\_number += 1  
  
 open\_bracer\_matches = re.finditer(r"{", line)  
 for \_ in open\_bracer\_matches:  
 open\_br += 1  
  
 close\_bracer\_matches = re.finditer(r"}", line)  
 for \_ in close\_bracer\_matches:  
 close\_br += 1  
 mutex\_match = re.match(rc.mutex\_regexp, line, re.MULTILINE)  
 if mutex\_match is not None:  
 FunctionContext.global\_mutexes.append(mutex\_match.group(2))  
 func\_match = re.match(rc.func\_decl\_regexp, line, re.MULTILINE)  
 if func\_match is not None:  
 pointer = func\_match.group(2)  
 return\_type = func\_match.group(1) + pointer.rstrip() if pointer is not None else func\_match.group(1)  
 name = func\_match.group(3)  
 parameters = get\_func\_params(func\_match.group(4), line\_number)  
 cur\_func = FunctionContext(return\_type, name, parameters)  
 cur\_func.source\_code[line\_number] = line  
  
 elif cur\_func is not None:  
 if re.match(r"}", line) is not None:  
 if open\_br != close\_br:  
 cur\_func.source\_code[line\_number] = line  
  
 elif open\_br == close\_br:  
 cur\_func.source\_code[line\_number] = line  
 func\_declarations.append(cur\_func)  
 cur\_func = None  
 close\_br = 0  
 open\_br = 0  
 else:  
 cur\_func.source\_code[line\_number] = line  
  
 return func\_declarations  
  
  
def get\_func\_params(params\_in\_bracket, line\_number):  
 *"""  
 Поиск параметров функции.* ***:rtype*** *List[Variable]  
 """* parameters\_list = []  
 if params\_in\_bracket[1:-1] == "":  
 return parameters\_list  
 raw\_param\_list = re.split(r",\s\*", params\_in\_bracket[1:-1])  
 for raw\_parameter in raw\_param\_list:  
 param\_match = re.match(rc.func\_param\_regexp, raw\_parameter)  
 pointer = param\_match.group(2)  
 param\_type = param\_match.group(1) + pointer.rstrip() if pointer is not None else param\_match.group(1)  
 param\_name = param\_match.group(3)  
 parameters\_list.append(Variable(raw\_parameter, line\_number, param\_name, param\_type))  
 return parameters\_list  
  
  
def get\_declared\_variables(source\_code):  
 *"""  
 Поиск и инициализация всех объявленных переменных в функции.* ***:rtype*** *List[Variables]  
 """* variables\_list = []  
 for line\_number, line in source\_code.items():  
 var\_matches = re.finditer(rc.variable\_regexp, line, re.MULTILINE)  
 for match in var\_matches:  
 var = Variable(match.group(0).strip(), line\_number, match.group(5), match.group(3))  
 if match.group(7) is not None:  
 var.value = match.group(7)[1:].strip()  
 variables\_list.append(var)  
  
 for \_, line in source\_code.items():  
 var\_matches = re.finditer(rc.assignment\_regexp, line, re.MULTILINE)  
 for match in var\_matches:  
 v1 = None  
 v2 = None  
 for var in variables\_list:  
 if match.group(1) == var.var\_name:  
 v1 = var  
 if match.group(2) == var.var\_name:  
 v2 = var  
 if v1 is not None and v2 is not None:  
 v1.value = v2.value  
 if v1 is not None and v2 is None:  
 v1.value = match.group(2)  
  
 return variables\_list  
  
  
def get\_declared\_threads(source\_code, declared\_variables):  
 *"""  
 Поиск и инициализация всех объявленных потоков в функции.* ***:rtype****: List[Thread]  
 """* threads\_list = []  
 for line\_number, line in source\_code.items():  
 matches = re.finditer(rc.thread\_regexp, line)  
 for match in matches:  
 cur\_thread\_params = get\_parameters(match.group(0), declared\_variables)  
 threads\_list.append(  
 Thread(match.group(0), line\_number, match.group(2), match.group(3), cur\_thread\_params))  
 return threads\_list  
  
  
def get\_parameters(raw\_parameters, declared\_variables) -> List[Variable]:  
 *"""  
 Поиск уже объявленных переменных, которые используются, например, при вызове функции или объявлении потока.* ***:rtype****: List[Variable]  
 """* tmp = re.search(r"\(.\*\)", raw\_parameters).group(0)[1:-1]  
 parameters\_list = re.split(r",", tmp)  
 v\_parameters\_list = []  
 for index, parameter in enumerate(parameters\_list):  
 for var in declared\_variables:  
 if re.sub(r'\[.\*\]', '', parameter.strip()) == var.var\_name:  
 v\_parameters\_list.append(var)  
 break  
 return v\_parameters\_list

Рисунок 42 – main\_code\_parser

Главная функция в «main\_code\_parser» – «find\_contexts», она отвечает за инициализацию объектной модели программы. После ее запуска, сначала, вызывается функция «get\_func\_declarations», которая проводит поиск сигнатур функций и формирование из них объектов. Затем методами «get\_declared\_variables» и «get\_declared\_threads» происходит поиск и инициализация объектов переменных и потоков. С помощью функции «get\_func\_parameters» происходит инициализация параметров функции. Также в модуле «main\_code\_parser» есть вспомогательная функция «get\_parameters», которая позволяет обрабатывать параметры вызываемых в анализируемом коде функций.

3.6 Агрегирующий модуль

Данный модуль (рисунок 43) отвечает за координацию и объединение всех предыдущих модулей в одно целое. Здесь, сначала, происходит передача объектной модели анализируемой программы обработчикам, а затем результат анализа предоставляется графическому интерфейсу.

from UI.UI import UI  
from core.main\_code\_parser import find\_contexts  
from handlers.buffer\_overflow\_handler import BufferOverflowHandler  
from handlers.commands\_introduction\_hadler import CommandsIntroductionHandler  
from handlers.data\_leak\_handler import DataLeakHandler  
from handlers.data\_storage\_handler import DataStorageManagementHandler  
from handlers.empty\_catch\_handler import EmptyCatchHandler  
from handlers.file\_access\_handler import IncorrectFileAccessHandler  
from handlers.format\_string\_handler import FormatStringHandler  
from handlers.integer\_overflow\_handler import IntegerOverflowHandler  
from handlers.memory\_leak\_handler import MemoryLeakHandler  
from handlers.race\_condition\_handler import RaceConditionHandler  
from handlers.random\_generator\_handler import RandomGeneratorHandler  
from handlers.readers\_writers\_handler import ReadersWritersHandler  
from handlers.sql\_injection\_handler import SQLInjectionHandler  
from utils.initial\_parse import clean\_code  
  
handlers\_list = {  
 BufferOverflowHandler.vulnerability\_name: BufferOverflowHandler,  
  
 CommandsIntroductionHandler.vulnerability\_name: CommandsIntroductionHandler,  
  
 DataLeakHandler.vulnerability\_name: DataLeakHandler,  
  
 DataStorageManagementHandler.vulnerability\_name: DataStorageManagementHandler,  
  
 EmptyCatchHandler.vulnerability\_name: EmptyCatchHandler,  
  
 IncorrectFileAccessHandler.vulnerability\_name: IncorrectFileAccessHandler,  
  
 FormatStringHandler.vulnerability\_name: FormatStringHandler,  
  
 IntegerOverflowHandler.vulnerability\_name: IntegerOverflowHandler,  
  
 MemoryLeakHandler.vulnerability\_name: MemoryLeakHandler,  
  
 RaceConditionHandler.vulnerability\_name: RaceConditionHandler,  
  
 RandomGeneratorHandler.vulnerability\_name: RandomGeneratorHandler,  
  
 ReadersWritersHandler.vulnerability\_name: ReadersWritersHandler,  
  
 SQLInjectionHandler.vulnerability\_name: SQLInjectionHandler,  
}  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 ui = UI(handlers\_list.keys())  
 ui.start\_main(  
 lambda vulnerability, program: handlers\_list[vulnerability]().parse(find\_contexts(clean\_code(program))))

Рисунок 43 – агрегирующий модуль

3.7 Модуль тестирования и генерации тестового кода С++.

Данный модуль предназначен для генерации случайного тестового кода С++ и запуска его проверки для получения статистики о количестве сгенерированных и найденных уязвимостей. Это позволяет производить отладку и оценивать работоспособность анализатора.

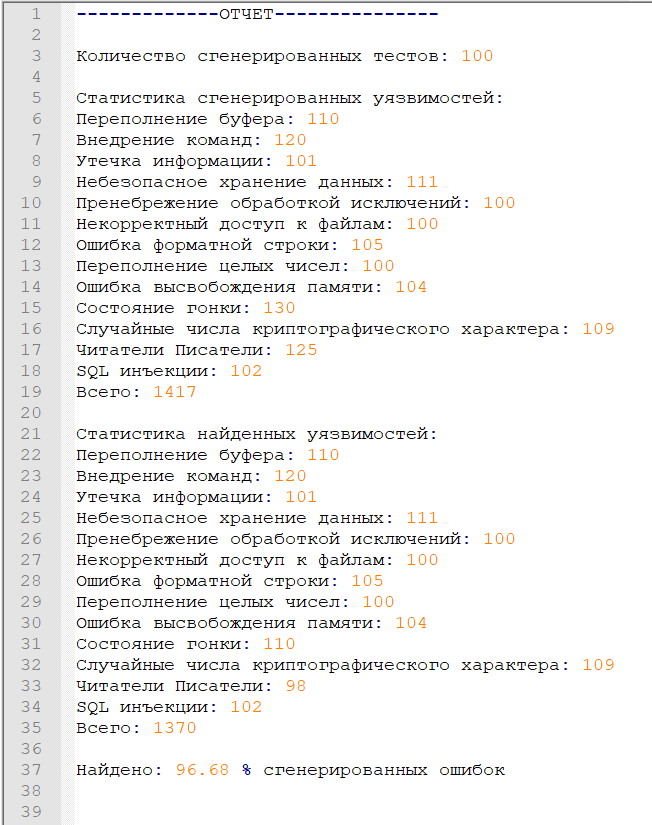
Главный класс отвечающий за генерацию – «CodeGenerator» (рисунок 44). В его возможности входит генерация программ в заданном количестве с случайным кодом С++ с указанными уязвимостями. Принцип работы основан на случайном вызове функций, которые генерируют либо произвольный код, либо код указанной уязвимости. То есть на каждую уязвимость существует отдельная функция, которая ее генерирует. Также в классе присутствуют методы, отвечающие за генерацию блоков «if» и «catch» случайного уровня вложенности. Сгенерированные уязвимости могут пересекаться друг с другом или с обычным произвольным кодом. Это сделано для максимального приближения сгенерированного кода к реальному.

После завершения генерации запускается проверка кода анализатором, а затем формируется общий отчет, в котором приводится статистика сгенерированных и найденных уязвимостей. Используя эту статистику можно производить отладку анализатора, а также делать вывод о степени правильности его работы. Пример сгенерированного отчета приведен на рисунке 45.

Для генерации различных видов переменных используется python-модуль «var\_generator» (рисунок 46).

import os  
import re  
import main  
from code\_generator.var\_genetor import \*  
from core.main\_code\_parser import find\_contexts  
from core.variable import is\_pointer, is\_array  
from utils.constants import \*  
from utils.initial\_parse import clean\_code  
  
v\_funcs = [gen\_var, gen\_var\_with\_value, gen\_buffer, gen\_pointer]  
p\_funcs = [gen\_var, gen\_pointer]  
  
  
class CodeGenerator:  
 cur\_available\_funcs = []  
 mutexes = []  
 final\_output = [["Переполнение буфера", 0], ["Внедрение команд", 0], ["Утечка информации", 0],  
 ["Небезопасное хранение данных", 0], ["Пренебрежение обработкой исключений", 0],  
 ["Некорректный доступ к файлам", 0], ["Ошибка форматной строки", 0], ["Переполнение целых чисел", 0],  
 ["Ошибка высвобождения памяти", 0], ["Состояние гонки", 0],  
 ["Случайные числа криптографического характера", 0], ["Читатели Писатели", 0],  
 ["SQL инъекции", 0]]  
 final\_output\_for\_test = [["Переполнение буфера", 0], ["Внедрение команд", 0], ["Утечка информации", 0],  
 ["Небезопасное хранение данных", 0], ["Пренебрежение обработкой исключений", 0],  
 ["Некорректный доступ к файлам", 0], ["Ошибка форматной строки", 0],  
 ["Переполнение целых чисел", 0],  
 ["Ошибка высвобождения памяти", 0], ["Состояние гонки", 0],  
 ["Случайные числа криптографического характера", 0], ["Читатели Писатели", 0],  
 ["SQL инъекции", 0]]  
 line\_number = 1  
  
 # генерирует рандомный cout  
 # def gen\_cout(self, indent, params):  
 # sample\_text = ""  
 # param = r\_v(params)  
 # for \_ in range(3):  
 # sample\_text += r\_v(sample\_words) + " "  
 # code = indent + "cout<<\"" + sample\_text + "\"<<" + param + ";\n"  
 # return code  
  
 # генерирует if statement с рандомным содержанием  
 def gen\_cond(self, indent, params):  
 own\_indent = "\t"  
 declaration = indent + "if(" + r\_v(params)[0] + " " + r\_v(cond) + " " + str(rng(0, 500)) + ") {\n"  
 self.line\_number += 1  
 body = ""  
 for index in range(2):  
 body += r\_v(self.cond\_generators)(self, own\_indent + indent, params)  
 code = declaration + body + indent + "}\n"  
 self.line\_number += 1  
 return code  
  
 # генерирует блок try-catch с рандомным содержанием  
 def gen\_try\_catch(self, indent, params):  
 own\_indent = "\t"  
 try\_declaration = indent + "try {\n"  
 self.line\_number += 1  
 try\_body = ""  
 for index in range(3):  
 try\_body += r\_v(self.catch\_generators)(self, own\_indent + indent, params)  
 catch\_declaration = indent + "catch (Exception\_" + str(rng(1, 20)) + " err) {\n"  
 self.line\_number += 1  
 catch\_body = ""  
 if rng(0, 1) == 1:  
 for index in range(2):  
 catch\_body += r\_v(self.catch\_generators)(self, own\_indent + indent, params)  
 else:  
 catch\_body = "\n"  
 for i in self.final\_output:  
 if "Пренебрежение обработкой исключений" in i[0]:  
 i[0] += f"\nСтрока {self.line\_number}\n"  
 i[1] += 1  
 self.line\_number += 1  
 code = try\_declaration + try\_body + indent + "}\n" + catch\_declaration + catch\_body + indent + "}\n"  
 self.line\_number += 2  
 return code  
  
 # генерирует ошибку переполнения буфера  
 def buff\_error(self, indent, params):  
 funcs = [  
 {"strcpy(": 2},  
 {"printf(Overflow %s and %s, ": 2},  
 {"memcpy(": 3},  
 {"strcat(": 2},  
 {"gets(": 1},  
 {"sprintf(": 3},  
 {"vsprintf(": 3},  
 {"strncpy(": 3},  
 {"scanf(%s, ": 1},  
 ]  
 cur\_func\_dict = r\_v(funcs)  
 cur\_func = list(cur\_func\_dict.keys())[0]  
 n = list(cur\_func\_dict.values())[0]  
 is\_nice = False  
 chosen\_param\_list = []  
 if "printf" in cur\_func:  
 for i in self.final\_output:  
 if "Ошибка форматной строки" in i[0]:  
 i[0] += f"\nСтрока {self.line\_number}\n"  
 i[1] += 1  
 while not is\_nice:  
 chosen\_param\_list = []  
 for p in gen\_n\_rands(n, 0, len(params) - 1):  
 if is\_pointer(params[p][1]) or is\_array(params[p][1]):  
 for i in self.final\_output:  
 if "Переполнение буфера" in i[0]:  
 i[0] += f"\nСтрока {self.line\_number}\n"  
 i[1] += 1  
 is\_nice = True  
 chosen\_param\_list.append(params[p][0])  
 code = indent + cur\_func + ", ".join(chosen\_param\_list) + ");\n"  
 self.line\_number += 1  
 return code  
  
 # генерирует ошибку встраивания команд  
 def c\_intr\_error(self, indent, params):  
 funcs = [  
 {"system(": 1},  
 {"popen(": 2},  
 {"execlp(": 3},  
 {"execvp(": 2},  
 {"ShellExecute(": 6},  
 ]  
 cur\_func\_dict = r\_v(funcs)  
 cur\_func = list(cur\_func\_dict.keys())[0]  
 n = list(cur\_func\_dict.values())[0]  
 chosen\_param\_list = [params[param][0] for param in gen\_n\_rands(n, 0, len(params) - 1)]  
 code = indent + cur\_func + ", ".join(chosen\_param\_list) + ");\n"  
 for i in self.final\_output:  
 if "Внедрение команд" in i[0]:  
 i[0] += f"\nСтрока {self.line\_number}\n"  
 i[1] += 1  
 self.line\_number += 1  
 return code  
  
 # генерирует ошибку утечки информации  
 def data\_leak(self, indent, params):  
 funcs = [  
 {"GetLastError(": 0},  
 {"SHGetFolderPath(": 5},  
 {"GetEnvironmentStrings(": 0},  
 {"GetEnvironmentVariable(": 1},  
 {"errno": 0},  
 {"getenv(": 1},  
 {"strerror(": 1},  
 {"perror(": 1}  
 ]  
 cur\_func\_dict = r\_v(funcs)  
 cur\_func = list(cur\_func\_dict.keys())[0]  
 if cur\_func == "errno":  
 code = indent + "cout<<" + cur\_func + ";\n"  
 for i in self.final\_output:  
 if "Утечка информации" in i[0]:  
 i[0] += f"\nСтрока {self.line\_number}\n"  
 i[1] += 1  
 self.line\_number += 1  
 return code  
 n = list(cur\_func\_dict.values())[0]  
 chosen\_param\_list = [params[param][0] for param in gen\_n\_rands(n, 0, len(params) - 1)]  
 code = indent + cur\_func + ", ".join(chosen\_param\_list) + ");\n"  
 for i in self.final\_output:  
 if "Утечка информации" in i[0]:  
 i[0] += f"\nСтрока {self.line\_number}\n"  
 i[1] += 1  
 self.line\_number += 1  
 return code  
  
 def storage\_error(self, indent, params):  
 funcs = [  
 {"SetFileSecurity(": 3},  
 {"SetKernelObjectSecurity(": 3},  
 {"SetServiceObjectSecurity(": 3},  
 {"chmod(": 2},  
 {"fchmod(": 2},  
 {"fchown(": 3},  
 {"fcntl(": 3},  
 {"setgroups(": 2}  
 ]  
 cur\_func\_dict = r\_v(funcs)  
 cur\_func = list(cur\_func\_dict.keys())[0]  
 n = list(cur\_func\_dict.values())[0]  
 chosen\_param\_list = [params[param][0] for param in gen\_n\_rands(n, 0, len(params) - 1)]  
 code = indent + cur\_func + ", ".join(chosen\_param\_list) + ");\n"  
 if cur\_func == "chmod(":  
 for i in self.final\_output:  
 if "Некорректный доступ к файлам" in i[0]:  
 i[0] += f"\nСтрока {self.line\_number}\n"  
 i[1] += 1  
 for i in self.final\_output:  
 if "Небезопасное хранение данных" in i[0]:  
 i[0] += f"\nСтрока {self.line\_number}\n"  
 i[1] += 1  
 self.line\_number += 1  
 return code  
  
 def file\_error(self, indent, params):  
 funcs = [  
 {"mkdir(": 1},  
 {"mktemp(": 1},  
 {"rmdir(": 1},  
 {"chmod(": 2},  
 {"utime(": 2}  
 # {"open": 0}  
 ]  
 cur\_func\_dict = r\_v(funcs)  
 cur\_func = list(cur\_func\_dict.keys())[0]  
 n = list(cur\_func\_dict.values())[0]  
 # if cur\_func == "open":  
 # code = indent + r\_v(params)[0] + ".open(\"D://" + r\_v(sample\_words) + "\");\n"  
 # for i in self.final\_output:  
 # if "Некорректный доступ к файлам" in i[0]:  
 # i[0] += f"\nСтрока {self.line\_number}\n"  
 # i[1] += 1  
 # self.line\_number += 1  
 # return code  
 if cur\_func == "chmod(":  
 for i in self.final\_output:  
 if "Небезопасное хранение данных" in i[0]:  
 i[0] += f"\nСтрока {self.line\_number}\n"  
 i[1] += 1  
 chosen\_param\_list = [params[param][0] for param in gen\_n\_rands(n, 0, len(params) - 1)]  
 code = indent + cur\_func + ", ".join(chosen\_param\_list) + ");\n"  
 for i in self.final\_output:  
 if "Некорректный доступ к файлам" in i[0]:  
 i[0] += f"\nСтрока {self.line\_number}\n"  
 i[1] += 1  
 self.line\_number += 1  
 return code  
  
 def format\_error(self, indent, params):  
 is\_nice = False  
 chosen\_param\_list = []  
 while not is\_nice:  
 chosen\_param\_list = []  
 for p in gen\_n\_rands(2, 0, len(params) - 1):  
 if is\_pointer(params[p][1]) or is\_array(params[p][1]):  
 for i in self.final\_output:  
 if "Переполнение буфера" in i[0]:  
 i[0] += f"\nСтрока {self.line\_number}\n"  
 i[1] += 1  
 is\_nice = True  
 chosen\_param\_list.append(params[p][0])  
 code = indent + "printf(Format %s and %s, " + ", ".join(chosen\_param\_list) + ");\n"  
 for i in self.final\_output:  
 if "Ошибка форматной строки" in i[0]:  
 i[0] += f"\nСтрока {self.line\_number}\n"  
 i[1] += 1  
 self.line\_number += 1  
 return code  
  
 def iover\_error(self, indent, params):  
 chosen\_var\_list = []  
 for \_ in range(3):  
 chosen\_var\_list.append(gen\_integer\_with\_value(indent, params))  
 self.line\_number += 1  
 sample\_code = ""  
 for index in range(1):  
 sample\_code += r\_v(self.iover\_generators)(self, indent, params)  
 code = "".join(chosen\_var\_list) + sample\_code + indent + chosen\_var\_list[0].split()[1] + " = " + \  
 chosen\_var\_list[1].split()[1] + \  
 " + " + chosen\_var\_list[2].split()[1] + ";\n"  
 for i in self.final\_output:  
 if "Переполнение целых чисел" in i[0]:  
 i[0] += f"\nСтрока {self.line\_number}\n"  
 i[1] += 1  
 self.line\_number += 1  
 return code  
  
 def free\_error(self, indent, params):  
 var\_type = r\_v(constants.types)  
 var\_name = "var\_" + str(rng(1, max\_val))  
 while var\_name in params:  
 var\_name = "var\_" + str(rng(1, max\_val))  
 code = indent + var\_type + " \*" + var\_name + " = new " + var\_type + ";\n"  
 params.append((var\_name, code))  
  
 for i in self.final\_output:  
 if "Ошибка высвобождения памяти" in i[0]:  
 i[0] += f"\nСтрока {self.line\_number}\n"  
 i[1] += 1  
 self.line\_number += 1  
 return code  
  
 def race\_error(self, indent, params):  
 code = ""  
 for index in range(rng(2, 5)):  
 for i in range(rng(2, 3)):  
 code += r\_v(self.random\_code\_generators)(self, indent, params)  
 chosen\_param\_list = [params[param][0] for param in gen\_n\_rands(3, 0, len(params) - 1)]  
 code += indent + "std::thread t" + str(index) + "(" + r\_v(self.cur\_available\_funcs) + ", " + ", ".join(  
 chosen\_param\_list) + ");\n"  
 for i in self.final\_output:  
 if "Состояние гонки" in i[0]:  
 i[0] += f"\nСтрока {self.line\_number}\n"  
 i[1] += 1  
 self.line\_number += 1  
 return code  
  
 def rng\_error(self, indent, params):  
 funcs = [  
 {"srand(": 1},  
 {"rand(": 0},  
 {"uniform\_real\_distribution(": 0}  
 ]  
 cur\_func\_dict = r\_v(funcs)  
 cur\_func = list(cur\_func\_dict.keys())[0]  
 if cur\_func == "uniform\_real\_distribution(":  
 code = indent + "std::uniform\_real\_distribution<> dis(" + str(rng(1, 3)) + ", " + str(rng(1, 9)) + ");\n"  
 for i in self.final\_output:  
 if "Случайные числа криптографического характера" in i[0]:  
 i[0] += f"\nСтрока {self.line\_number}\n"  
 i[1] += 1  
 self.line\_number += 1  
 return code  
 if cur\_func == "srand(":  
 code = indent + cur\_func + str(rng(3, 100)) + ");\n"  
 for i in self.final\_output:  
 if "Случайные числа криптографического характера" in i[0]:  
 i[0] += f"\nСтрока {self.line\_number}\n"  
 i[1] += 1  
 self.line\_number += 1  
 return code  
 else:  
 code = indent + cur\_func + ");\n"  
 for i in self.final\_output:  
 if "Случайные числа криптографического характера" in i[0]:  
 i[0] += f"\nСтрока {self.line\_number}\n"  
 i[1] += 1  
 self.line\_number += 1  
 return code  
  
 def readers\_error(self, indent, params):  
 code = ""  
 for index in range(rng(2, 5)):  
 for i in range(rng(2, 3)):  
 code += r\_v(self.random\_code\_generators)(self, indent, params)  
 chosen\_param\_list = [params[param][0] for param in gen\_n\_rands(4, 0, len(params) - 1)]  
 code += indent + "std::thread t" + str(index) + "(" + r\_v(self.cur\_available\_funcs) + ", " + ", ".join(  
 chosen\_param\_list) + ");\n"  
 for i in self.final\_output:  
 if "Читатели Писатели" in i[0]:  
 i[0] += f"\nСтрока {self.line\_number}\n"  
 i[1] += 1  
 self.line\_number += 1  
 return code  
  
 def sql\_error(self, indent, params):  
 funcs = ["execute(", "executeQuery("]  
 cur\_func = r\_v(funcs)  
 sample\_text = ""  
 for \_ in range(3):  
 sample\_text += r\_v(sample\_words) + "'"  
 random = rng(0, 1)  
 if random == 0:  
 code = indent + cur\_func + "\"" + sample\_text + "\");\n"  
 else:  
 code = indent + cur\_func + r\_v(params)[0] + ");\n"  
 for i in self.final\_output:  
 if "SQL инъекции" in i[0]:  
 i[0] += f"\nСтрока {self.line\_number}\n"  
 i[1] += 1  
 self.line\_number += 1  
 return code  
  
 def gen\_sample\_function(self, vulnerabilities, file):  
 indent = "\t"  
 # Генерация сигнатуры функции  
 func\_type = r\_v(func\_types)  
 func\_name = "function\_" + str(rng(0, 20))  
 generated\_vars = []  
 func\_params = []  
 for index in range(rng(1, 4)):  
 func\_params.append(r\_v(p\_funcs)(indent, generated\_vars))  
 file.write(func\_type + " " + func\_name + "(" + ", ".join([param[1:-2] for param in func\_params]) + ") {\n")  
 self.line\_number += 1  
 # Генерация переменных  
 if "Читатели Писатели" in vulnerabilities:  
 mutex = r\_v(self.mutexes)  
 if rng(0, 1) == 1:  
 file.write("\t" + mutex + ".lock();\n")  
 self.line\_number += 1  
 if "Переполнение буфера" in vulnerabilities:  
 file.write(gen\_buffer(indent, generated\_vars))  
 self.line\_number += 1  
 for index in range(rng(6, 7)):  
 code = r\_v(v\_funcs)(indent, generated\_vars)  
 file.write(code)  
 self.line\_number += 1  
 # Генерация произвольного кода  
 for index in range(rng(1, 2)):  
 file.write(r\_v(self.random\_code\_generators)(self, indent, generated\_vars))  
  
 # Конец функции  
 file.write("}\n")  
 self.line\_number += 1  
 return func\_name  
  
 def gen\_function(self, vuln\_generators, file):  
 indent = "\t"  
 func\_type = "int"  
 func\_name = "main"  
 generated\_vars = []  
 func\_params = []  
 for index in range(rng(1, 4)):  
 func\_params.append(r\_v(p\_funcs)(indent, generated\_vars))  
 file.write(func\_type + " " + func\_name + "(" + ", ".join([param[1:-2] for param in func\_params]) + ") {\n")  
 self.line\_number += 1  
 for index in range(rng(6, 7)):  
 code = r\_v(v\_funcs)(indent, generated\_vars)  
 file.write(code)  
 self.line\_number += 1  
 for generator in vuln\_generators:  
 for index in range(rng(2, 3)):  
 #file.write(r\_v(self.random\_code\_generators)(self, indent, generated\_vars))  
 file.write(generator(self, indent, generated\_vars))  
 file.write("}\n")  
 self.line\_number += 1  
 return func\_name  
  
 def gen\_code(self, vulnerabilities: List[str], test\_numbers):  
 folder = "../generated\_tests"  
 for the\_file in os.listdir(folder):  
 file\_path = os.path.join(folder, the\_file)  
 try:  
 if os.path.isfile(file\_path):  
 os.unlink(file\_path)  
 except Exception as e:  
 print(e)  
 for index in range(test\_numbers):  
 self.line\_number = 1  
 file = open('../generated\_tests/' + "+".join(vulnerabilities)[0] + str(index + 1) + '.cpp', 'w')  
 generators = []  
 for vuln in vulnerabilities:  
 generators.append(self.name\_generator[vuln])  
 if "Читатели Писатели" in vulnerabilities:  
 mutex\_cnt = rng(1, 3)  
 for i in range(mutex\_cnt):  
 mutex\_name = "testMutex" + str(r\_v(gen\_n\_rands(mutex\_cnt, 0, 10)))  
 file.write("mutex " + mutex\_name + ";\n")  
 self.line\_number += 1  
 self.mutexes.append(mutex\_name)  
 for i in range(rng(1, 3)):  
 self.cur\_available\_funcs.append(self.gen\_sample\_function(vulnerabilities, file))  
 self.gen\_function(generators, file)  
 self.cur\_available\_funcs = []  
 file.write(str(self.line\_number))  
 file.close()  
 # review\_file = open('../generated\_tests/' + "+".join(vulnerabilities)[0] + str(index + 1) + 'обзор.cpp', 'w',  
 # encoding="utf-8")  
 # review\_file.write("Сгенерированные и найденные ошибки:\n\n")  
 # for vulnerability in vulnerabilities:  
 # for key, value in self.final\_output:  
 # if vulnerability in key:  
 # review\_file.write("Сгенерированные:\n")  
 # review\_file.write(key + ": " + str(value) + "\n")  
 # review\_file.write("\nНайденные:\n")  
 # for vulnerability in vulnerabilities:  
 # to\_write = main.handlers\_list[vulnerability]().parse((find\_contexts(  
 # clean\_code('../generated\_tests/' + "+".join(vulnerabilities) + str(index + 1) + '.cpp'))))[  
 # -1] + "\n"  
 # review\_file.write(to\_write)  
 for vulnerability in vulnerabilities:  
 to\_write = main.handlers\_list[vulnerability]().parse((find\_contexts(  
 clean\_code('../generated\_tests/' + "+".join(vulnerabilities)[0] + str(index + 1) + '.cpp'))))[  
 -1] + "\n"  
 m = re.search(r'(\d+)', to\_write)  
 tmp\_cnt = int(m.group(1))  
 for i in self.final\_output\_for\_test:  
 if vulnerability in i[0]:  
 i[1] += tmp\_cnt  
 stat\_file = open('общая\_статистика.cpp', 'w',  
 encoding="utf-8")  
 stat\_file.write("Статистика сгенерированных уязвимостей:\n")  
 for vulnerability in vulnerabilities:  
 cur\_vuln\_cnt = 0  
 for key, value in self.final\_output:  
 if vulnerability in key:  
 cur\_vuln\_cnt += value  
 stat\_file.write(vulnerability + ": " + str(cur\_vuln\_cnt) + "\n")  
  
 stat\_file.write("\nСтатистика найденных уязвимостей:\n")  
 for vulnerability in vulnerabilities:  
 for key, value in self.final\_output\_for\_test:  
 if vulnerability in key:  
 stat\_file.write(vulnerability + ": " + str(value) + "\n")  
  
 random\_code\_generators = [gen\_cond, gen\_try\_catch, buff\_error, c\_intr\_error, data\_leak, storage\_error,  
 file\_error, format\_error, iover\_error, free\_error, rng\_error, sql\_error]  
  
 catch\_generators = [gen\_cond, buff\_error, c\_intr\_error, data\_leak, storage\_error,  
 file\_error, format\_error, iover\_error, free\_error, rng\_error, sql\_error]  
  
 cond\_generators = [gen\_try\_catch, buff\_error, c\_intr\_error, data\_leak, storage\_error,  
 file\_error, format\_error, iover\_error, free\_error, rng\_error, sql\_error]  
  
 iover\_generators = [gen\_cond, gen\_try\_catch, buff\_error, c\_intr\_error, data\_leak, storage\_error,  
 file\_error, format\_error, free\_error, rng\_error, sql\_error]  
  
 name\_generator = {  
 "Переполнение буфера": buff\_error,  
 "Внедрение команд": c\_intr\_error,  
 "Утечка информации": data\_leak,  
 "Небезопасное хранение данных": storage\_error,  
 "Пренебрежение обработкой исключений": gen\_try\_catch,  
 "Некорректный доступ к файлам": file\_error,  
 "Ошибка форматной строки": format\_error,  
 "Переполнение целых чисел": iover\_error,  
 "Ошибка высвобождения памяти": free\_error,  
 "Состояние гонки": race\_error,  
 "Случайные числа криптографического характера": rng\_error,  
 "Читатели Писатели": readers\_error,  
 "SQL инъекции": sql\_error  
 }  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 gen = CodeGenerator()  
 gen.gen\_code(["Переполнение буфера", "Внедрение команд", "Утечка информации", "Небезопасное хранение данных",  
 "Пренебрежение обработкой исключений"], 1)

Рисунок 44 – класс CodeGenerator

Рисунок 45 – пример отчета

from typing import List  
from utils import constants  
from code\_generator.rng\_utils import \*  
  
max\_val = 40  
  
  
def gen\_buffer(indent, generated\_vars: List[tuple]):  
 var\_type = r\_v(constants.types)  
 var\_name = "var\_" + str(rng(1, max\_val))  
 while var\_name in [gen\_vars[0] for gen\_vars in generated\_vars]:  
 var\_name = "var\_" + str(rng(1, max\_val))  
 code = indent + var\_type + " " + var\_name + "[" + str(rng(1, 50)) + "];\n"  
 generated\_vars.append((var\_name, code))  
 return code  
  
  
def gen\_pointer(indent, generated\_vars: List[tuple]):  
 var\_type = r\_v(constants.types)  
 var\_name = "var\_" + str(rng(1, max\_val))  
 while var\_name in [gen\_vars[0] for gen\_vars in generated\_vars]:  
 var\_name = "var\_" + str(rng(1, max\_val))  
 code = indent + var\_type + " \*" + var\_name + ";\n"  
 generated\_vars.append((var\_name, code))  
 return code  
  
  
def gen\_var(indent, generated\_vars: List[tuple]):  
 var\_type = r\_v(constants.types)  
 var\_name = "var\_" + str(rng(1, max\_val))  
 while var\_name in [gen\_vars[0] for gen\_vars in generated\_vars]:  
 var\_name = "var\_" + str(rng(1, max\_val))  
 code = indent + var\_type + " " + var\_name + ";\n"  
 generated\_vars.append((var\_name, code))  
 return code  
  
  
def gen\_var\_with\_value(indent, generated\_vars: List[tuple]):  
 var\_type = r\_v(constants.types)  
 var\_name = "var\_" + str(rng(1, max\_val))  
 while var\_name in [gen\_vars[0] for gen\_vars in generated\_vars]:  
 var\_name = "var\_" + str(rng(1, max\_val))  
 code = indent + var\_type + " " + var\_name + " = " + str(rng(1, 30)) + ";\n"  
 generated\_vars.append((var\_name, code))  
 return code  
  
  
def gen\_stream(indent, generated\_vars: List[tuple]):  
 var\_type = r\_v(constants.stream\_types)  
 var\_name = "var\_" + str(rng(1, max\_val))  
 while var\_name in [gen\_vars[0] for gen\_vars in generated\_vars]:  
 var\_name = "var\_" + str(rng(1, max\_val))  
 code = indent + var\_type + " " + var\_name + ".open(\"D://" + r\_v(constants.sample\_words) + "\");\n"  
 generated\_vars.append((var\_name, code))  
 return code  
  
  
def gen\_integer\_with\_value(indent, generated\_vars: List[tuple]):  
 var\_type = r\_v(constants.int\_types)  
 var\_name = "var\_" + str(rng(1, max\_val))  
 while var\_name in [gen\_vars[0] for gen\_vars in generated\_vars]:  
 var\_name = "var\_" + str(rng(1, max\_val))  
 max\_over = constants.integer\_limits[var\_type].max  
 code = indent + var\_type + " " + var\_name + " = " + str(rng(int(max\_over/2), max\_over)) + ";\n"  
 generated\_vars.append((var\_name, code))  
 return code  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 for \_ in range(10):  
 print(gen\_integer\_with\_value("", []))

Рисунок 46 – var\_generator

ПРИЛОЖЕНИЕ Б

ПОЛНЫЙ ЛИСТИНГ ПРОГРАММЫ

**Обработчик переполнения буфера.**

Автор: Ковалев К.А.

from typing import List

from core import main\_code\_parser

from core.base\_handler import BaseHandler

from core.variable import \*

from core.function\_context import FunctionContext

class BufferOverflowHandler(BaseHandler):

vulnerability\_name = 'Переполнение буфера'

"""

- char \* strcpy ( char \* destination, const char \* source ); - Копирует строку <source> в буфер <destination>.

- int printf ( const char \* format, ... ); - Выводит строку указанную в формате в стандартный поток вывода.

- char \* strcat ( char \* destination, const char \* source ); - Добавляет к строке <destination> копию <source>

- void \* memcpy ( void \* destination, const void \* source, size\_t num ); - копирует <num> байт из <source> в

<destination>

- char \* gets ( char \* str ); - читает символы из потока ввода и записывает их в <str>

- int sprintf ( char \* str, const char \* format, ... ); - аналогично <printf>, но вывод происходит в буфер <str>

- int vsprintf (char \* s, const char \* format, va\_list arg ); - аналогично <sprintf>, но принимает только один <arg>

- char \* strncpy ( char \* destination, const char \* source, size\_t num ); - <strcpy>, но копирует только <num> байт

"""

def \_\_init\_\_(self):

self.pattern = r"(strcpy|printf|strcat|memcpy|gets|sprintf|vsprintf|strncpy|scanf)" \

r"\(.\*\)"

self.output = []

def parse(self, contexts: List[FunctionContext]):

total\_errors = 0

for context in contexts:

declared\_variables = context.variables

for line\_number, line in context.source\_code.items():

matches = re.finditer(self.pattern, line)

for match in matches:

used\_variables = main\_code\_parser.get\_parameters(match.group(0), declared\_variables)

for used\_variable in used\_variables:

declaration = used\_variable.full\_declaration

if is\_pointer(declaration) or is\_array(declaration):

total\_errors += 1

self.output.append(

f"{total\_errors}) Предупреждение в методе <{context.name}>!\n"

f"Использование буфера <{declaration[:-1]}> (строка {used\_variable.line\_appeared}) "

f"в небезопасной функции <{match.group(1)}> (строка {line\_number}).\n"

f"Это может стать причиной переполнения буфера. "

f"Убедитесь в наличии проверки этой угрозы!\n")

self.output.append(self.vulnerability\_name + ": " + str(total\_errors))

return self.output

**Обработчик пренебрежения обработок ошибок.**

Автор: Ковалев К.А.

import re

from typing import List

from core.base\_handler import BaseHandler

from core.function\_context import FunctionContext

def is\_empty(catch\_body: List[str]):

for line in catch\_body:

tmp = line.replace("", "")

if tmp == "": continue

else: return False

return True

class EmptyCatchHandler(BaseHandler):

vulnerability\_name = 'Пренебрежение обработкой исключений'

def \_\_init\_\_(self):

self.pattern = r'catch\s\*' \

r'\(.\*\)\s\*\{'

self.output = []

self.total\_errors = 0

def analyze\_catch(self, cur\_line\_number, global\_line\_number, context):

body = []

close\_br = 0

open\_br = 1

dict\_lines = context.source\_code

while cur\_line\_number < len(dict\_lines):

cur\_line = list(dict\_lines.items())[cur\_line\_number][1]

if re.match(self.pattern, cur\_line) is not None:

return self.analyze\_catch(cur\_line\_number + 1, global\_line\_number + 1, context)

if re.match(r'}', cur\_line) is None:

if re.match(r'.\*{', cur\_line) is not None:

open\_br += 1

body.append(cur\_line)

if re.match(r'}', cur\_line):

close\_br += 1

if open\_br != close\_br:

body.append(cur\_line)

elif is\_empty(body):

self.total\_errors += 1

self.output.append(f"{self.total\_errors}) Предупреждение в методе <{context.name}>!\n"

f"Отстутсвует обработка исключения или ошибки! (строка {global\_line\_number - len(body)})\n")

return cur\_line\_number

elif not is\_empty(body):

return cur\_line\_number

global\_line\_number += 1

cur\_line\_number += 1

def parse(self, contexts: List[FunctionContext]):

index = 0

prev\_index = 0

for context in contexts:

prev\_index += index

index = 0

while index < len(context.source\_code):

processed\_line = list(context.source\_code.items())[index][1]

if re.match(self.pattern, processed\_line) is not None:

index = self.analyze\_catch(index + 1, index + prev\_index + 1, context)

else:

index += 1

self.output.append(self.vulnerability\_name + ": " + str(self.total\_errors))

self.total\_errors = 0

return self.output

**Обработчик проблем, связанных с динамическим выделением памяти.**

Автор: Терентьев Е.А.

from typing import List

from core.base\_handler import BaseHandler

from core.function\_context import FunctionContext

from core.variable import \*

class MemoryLeakHandler(BaseHandler):

vulnerability\_name = 'Ошибка высвобождения памяти'

def \_\_init\_\_(self):

self.pattern = r'^\s\*delete(\(.\*\))'

self.output = []

def parse(self, contexts: List[FunctionContext]):

total\_errors = 0

for context in contexts:

m\_vars = []

for var in context.variables:

if var.value is not None:

if "new" in var.full\_declaration:

m\_vars.append(var)

free\_matches = []

for \_, line in context.source\_code.items():

matches = re.finditer(self.pattern, line, re.IGNORECASE)

free\_matches += matches

for var in m\_vars:

if var.var\_name not in list(map(lambda x: x.group(1)[1:-1].strip(), free\_matches)):

total\_errors += 1

self.output.append(f"{total\_errors}) Угроза в методе <{context.name}>!\n"

f"Отсутствует высвобождение памяти для переменной <{var.var\_name}> (строка {var.line\_appeared})")

self.output.append(self.vulnerability\_name + ": " + str(total\_errors))

return self.output

**Обработчик внедрения SQL команд.**

Автор: Терентьев Е.А.

import re

from typing import List

from core.base\_handler import BaseHandler

from core.function\_context import FunctionContext

class SQLInjectionHandler(BaseHandler):

vulnerability\_name = 'SQL инъекции'

def \_\_init\_\_(self):

self.pattern = r'(executeQuery|execute)(\(.\*\))'

self.output = []

def parse(self, contexts: List[FunctionContext]):

total\_errors = 0

for context in contexts:

for line\_number, line in context.source\_code.items():

matches = re.finditer(self.pattern, line)

for match in matches:

tmp = match.group(2)[1:-1]

if tmp[0] == '\"': # check if parameter is literally string like "test string"

if re.search(r'[^\\]\'', tmp) is not None:

total\_errors += 1

self.output.append(f"{total\_errors}) Угроза в методе <{context.name}>!\n"

f"Тело sql запроса <{tmp[1:-1]}>, используемое в функции <{match.group(1)}> (строка {line\_number}) "

f"имеет неэкранированный символ(ы) <'>\nЭто может привести к sql инъекции")

else:

total\_errors += 1

self.output.append(f"{total\_errors}) Предупреждение в методе <{context.name}>!\n"

f"Проверьте тело sql запроса <{tmp}>, используемое в функции <{match.group(1)}> (строка {line\_number}) "

f"на наличие неэкранированного символа <'>'\nЭто может привести к sql инъекции")

self.output.append(self.vulnerability\_name + ": " + str(total\_errors))

return self.output

**Обработчик внедрения команд.**

Автор: Терентьев Е.А.

import re

from typing import List

from core.base\_handler import BaseHandler

from core.function\_context import FunctionContext

class CommandsIntroductionHandler(BaseHandler):

vulnerability\_name = 'Внедрение команд'

"""

- int system (const char\* command); - выполняет системную команду.

- FILE \*popen(const char \*command, const char \*type); - создает дочерний процесс, которому можно передать команду.

- int execlp(const char \*file, const char \*arg, ...); - выполнение исполняемого файла по пути <file> с передачей

параметра в <arg>.

- int execvp(const char \*file, char \*const argv[]); - тоже что и execlp, но параметры передаются в <argv>.

- HINSTANCE ShellExecute(HWND hwnd, LPCSTR lpOperation, LPCSTR lpFile, LPCSTR lpParameters, LPCSTR lpDirectory, INT nShowCmd); -

выполняет команду <lpOperation> над указанным файлом <lpFile>.

"""

def \_\_init\_\_(self):

self.pattern = r'(system|popen|execlp|execvp|ShellExecute)'

self.output = []

def parse(self, contexts: List[FunctionContext]):

total\_errors = 0

for context in contexts:

for line\_number, line in context.source\_code.items():

matches = re.finditer(self.pattern, line)

for match in matches:

total\_errors += 1

self.output.append(f"{total\_errors}) Предупреждение в методе <{context.name}>!\n"

f"Использование функции <{match.group(1)}>, в которую может быть внедрена"

f" вредоносная команда (строка {line\_number}) "

f"Убедитесь в наличии проверки этой угрозы!\n")

self.output.append(self.vulnerability\_name + ": " + str(total\_errors))

return self.output

**Обработчик состояния гонки.**

Автор: Ковалев К.А.

from typing import List

from core.base\_handler import BaseHandler

from core.function\_context import FunctionContext

class RaceConditionHandler(BaseHandler):

vulnerability\_name = 'Состояние гонки'

def \_\_init\_\_(self):

self.output = []

def parse(self, contexts: List[FunctionContext]):

total\_errors = 0

for context in contexts:

if len(context.threads) != 0:

declared\_threads = context.threads

# part for detecting same runnables

funcs\_to\_threads = [[], []]

func\_in\_threads = []

# creating list of all runnables

for thread in declared\_threads:

if thread.runnable\_function not in func\_in\_threads:

func\_in\_threads.append(thread.runnable\_function)

# filling list of func - threads

for index, func in enumerate(func\_in\_threads):

for thread in declared\_threads:

if func == thread.runnable\_function:

funcs\_to\_threads[index].append(thread)

# checking where function is used more than one time as runnable

for func\_usage in funcs\_to\_threads:

if len(func\_usage) > 1:

total\_errors += 1

warning = f"{total\_errors}) Предупреждение в методе <{context.name}>!\n"

warning += "Потоки:\n"

cur\_func = func\_usage[0].runnable\_function

for thread in func\_usage:

warning += f"\"<{thread.thread\_name}> (строка {thread.line\_appeared})\"\n"

warning += f"используют одну и туже исполняемую функцию <{cur\_func}>, " \

f"это может вызвать состояние гонки!\n"

self.output.append(warning)

self.output.append(self.vulnerability\_name + ": " + str(total\_errors))

return self.output

**Обработчик ошибки форматной строки.**

Автор: Щекочихин Д.А.

import re

from typing import List

from core.base\_handler import BaseHandler

from core.function\_context import FunctionContext

class FormatStringHandler(BaseHandler):

vulnerability\_name = 'Ошибка форматной строки'

def \_\_init\_\_(self):

self.pattern = r'(printf\(.\*)\)'

self.output = []

def parse(self, contexts: List[FunctionContext]):

total\_errors = 0

for context in contexts:

for line\_number, line in context.source\_code.items():

matches = re.finditer(self.pattern, line, re.IGNORECASE)

for \_ in matches:

total\_errors += 1

self.output.append(f"Предупреждение в методе <{context.name}>!\n"

f"Возможна ошибка форматной строки (строка {line\_number})\n")

self.output.append(self.vulnerability\_name + ": " + str(total\_errors))

return self.output

**Обработчик некорректного доступа к файлам.**

Автор: Щекочихин Д.А.

import re

from typing import List

from core.base\_handler import BaseHandler

from core.function\_context import FunctionContext

class IncorrectFileAccessHandler(BaseHandler):

vulnerability\_name = 'Некорректный доступ к файлам'

"""

- ofstream.open(const char \*fname); - открывает поток вывода в файл <fname>.

- ifstream.open(const char \*fname); - открывает поток ввода в файл <fname>.

- int mkdir(const char \*path); - создает директорию по указанному пути.

- int rmdir(const char \*path); - удаляет директорию по указанному пути.

- char \*mktemp(char \*fname); - создает уникальное имя файла и сам файл.

- int utime(char \*fname, struct utimbuf \*t); - изменяет время последней модификации файла <fname>.

"""

def \_\_init\_\_(self):

self.pattern = r'(^mkdir|^mktemp|^rmdir|^utime|^chmod)'

self.output = []

def parse(self, contexts: List[FunctionContext]):

total\_errors = 0

for context in contexts:

streams = list(filter(lambda v: 1 if v.var\_type == "ofstream" or v.var\_type == "ifstream" else 0,

context.variables))

streams = [v.var\_name for v in streams]

for line\_number, line in context.source\_code.items():

matches = re.finditer(r"(ifstream|ofstream)\s\*([\w]\*[\w\d\_]\*)\.open\(.\*\)", line)

for match in matches:

total\_errors += 1

self.output.append(f"{total\_errors}) Предупреждение в методе <{context.name}>!\n"

f"Использование функции открытия потока ввода/вывода <{match.group(1)}> "

f"(строка {line\_number}). Проверьте доступность открываемого файла")

# matches = re.finditer(r"(" + "|".join(streams) + r")\.open", line)

# for match in matches:

# total\_errors += 1

# self.output.append(f"{total\_errors}) Предупреждение в методе <{context.name}>!\n"

# f"Использование функции открытия потока ввода/вывода <{match.group(1)}> "

# f"(строка {line\_number}). Проверьте доступность открываемого файла")

matches = re.finditer(self.pattern, line)

for match in matches:

total\_errors += 1

self.output.append(f"{total\_errors}) Предупреждение в методе <{context.name}>!\n"

f"Использование функции <{match.group(0)}>, которая осуществляет доступ к файлам"

f" (line {line\_number}). Отсутствие проверки существования файла может "

f"привести к ошибке")

self.output.append(self.vulnerability\_name + ": " + str(total\_errors))

return self.output

**Обработчик задачи Читатели – Писатели.**

Автор: Ковалев К.А.

import re

from typing import List

from core.base\_handler import BaseHandler

from core.function\_context import FunctionContext

class ReadersWritersHandler(BaseHandler):

vulnerability\_name = 'Читатели Писатели'

def \_\_init\_\_(self):

self.output = []

self.total\_errors = 0

def check\_mutex(self, context: FunctionContext, thread):

mutex\_cnt = len(FunctionContext.global\_mutexes)

mutex\_locked = [0 for \_ in range(mutex\_cnt)]

mutex\_unlocked = [0 for \_ in range(mutex\_cnt)]

context\_beginning = None

for line\_number, line in context.source\_code.items():

context\_beginning = line\_number + 1

break

for line\_number, line in context.source\_code.items():

for index, mutex in enumerate(FunctionContext.global\_mutexes):

if re.match(r'' + mutex + r'\.lock\s\*\(.\*\);', line):

mutex\_locked[index] += 1

continue

for index, mutex in enumerate(FunctionContext.global\_mutexes):

if re.match(r'' + mutex + r'\.unlock\s\*\(.\*\);', line):

mutex\_unlocked[index] += 1

for index in range(mutex\_cnt):

if mutex\_locked[index] == mutex\_unlocked[index]:

continue

elif mutex\_locked[index] > mutex\_unlocked[index]:

self.total\_errors += 1

return f"{self.total\_errors - 1}) Отсутствует освобождение блока синхронизации " \

f"(семафор - {FunctionContext.global\_mutexes[index]}) в исполняемой функции <{context.name}> " \

f"(строка {context\_beginning}) потока <{thread.thread\_name}>"

elif mutex\_locked[index] < mutex\_unlocked[index]:

self.total\_errors += 1

return f"{self.total\_errors - 1}) Отсутстввует захват блока синхронизации " \

f"(семафор - {FunctionContext.global\_mutexes[index]}) в исполняемой функции <{context.name}> " \

f"(строка {context\_beginning}) потока <{thread.thread\_name}>"

is\_no\_mutex = True

for index, \_ in enumerate(mutex\_locked):

if mutex\_locked[index] != 0 and mutex\_unlocked[index] != 0:

is\_no\_mutex = False

if is\_no\_mutex:

self.total\_errors += 1

return f"{self.total\_errors - 1}) Отсутстввует блок синхронизации " \

f"в исполняемой функции <{context.name}> (строка {context\_beginning}) потока <{thread.thread\_name}>"

return ""

def parse(self, contexts: List[FunctionContext]):

for context in contexts:

if len(context.threads) != 0:

declared\_threads = context.threads

# part for detecting same variables

vars\_to\_threads = [] # list of dicts - List[{var\_name: List[Threads]}]

var\_in\_threads = [] # list of variables - List[Variable]

# creating list of all used params

for thread in declared\_threads:

for parameter in thread.parameter\_list:

if parameter not in var\_in\_threads:

var\_in\_threads.append(parameter)

# filling list of params - threads

for index, var in enumerate(var\_in\_threads):

t\_list = []

dict\_tmp = {var.var\_name: t\_list}

vars\_to\_threads.append(dict\_tmp)

for thread in declared\_threads:

for parameter in thread.parameter\_list:

if var == parameter:

vars\_to\_threads[index][var.var\_name].append(thread)

# checking where same parameters has usage in several threads

thread\_with\_same\_vars = []

for var\_usage in vars\_to\_threads:

warning = ""

for key in var\_usage:

if len(var\_usage[key]) > 1:

self.total\_errors += 1

warning = "---Анализ параметров потоков---\n"

warning += f"{self.total\_errors}) Предупреждение в методе <{context.name}>!\n"

warning += "Потоки:\n"

for thread in var\_usage[key]:

warning += f"\"<{thread.thread\_name}> (строка {thread.line\_appeared})\"\n"

warning += f"используют одну и ту же переменную <{key}> " \

f"в качестве параметра для исполняемой функции, " \

f"это может привести к проблеме Читатели - Писатели!\n"

self.output.append(warning)

self.output.append("---Анализ объектов синхронизации в исполняемых функциях потоков, "

"использующих одни и теже переменные---\n")

thread\_list = list(var\_usage.items())[0][1]

for thread in thread\_list:

for c in contexts:

if c.name == thread.runnable\_function:

self.output.append(self.check\_mutex(c, thread))

self.output.append("\n")

self.output.append(self.vulnerability\_name + ": " + str(self.total\_errors))

self.total\_errors = 0

return self.output

**Обработчик случайных чисел криптографического характера.**

Автор: Оганисян Т.С.

import re

from typing import List

from core.base\_handler import BaseHandler

from core.function\_context import FunctionContext

class RandomGeneratorHandler(BaseHandler):

vulnerability\_name = 'Случайные числа криптографического характера'

"""

void srand(unsigned seed); - используется для установки начала последовательности, генерируемой функ­цией rand().

int rand (void); - вовращает псевдо-случайное челочисленное число в интервале от 0 до RAND\_MAX.

uniform\_real\_distribution - класс используемый для генерации равномернораспределенного нецелочисленного числа

равномерно распределенного на интервале [a, b)

"""

def \_\_init\_\_(self):

self.pattern = [r'srand\((.\*?)\)', r'rand\(\)|uniform\_real\_distribution']

self.output = []

def parse(self, contexts: List[FunctionContext]):

*"""*

*1)Находит использование функции генерации srand(), в случаях когда она инициализирована const числовым значением*

*2)Находит использование небезопасных функций генерации rand() и uniform\_real\_distribution*

*"""*

total\_errors = 0

for context in contexts:

for line\_number, line in context.source\_code.items():

srand\_matches = re.finditer(self.pattern[0], line, re.IGNORECASE)

for match in srand\_matches:

if match.group(1).isdigit():

total\_errors += 1

self.output.append(f"{total\_errors}) Угроза в методе <{context.name}>!\n"

f"Использование функции <srand>, которая инициализирована константным значением - "

f"{match.group(1)} (строка {line\_number})")

rand\_matches = re.finditer(self.pattern[1], line, re.IGNORECASE)

for match in rand\_matches:

total\_errors += 1

self.output.append(f"{total\_errors}) Предупреждение в методе {context.name}!\n"

f"Использования не крипто-безопасной функции генерации <{match.group(0)}> (строка {line\_number})")

self.output.append(self.vulnerability\_name + ": " + str(total\_errors))

return self.output

**Обработчик переполнения целых чисел.**

Автор: Оганисян Т.С.

import re

from typing import List

from core.base\_handler import BaseHandler

from core.function\_context import FunctionContext

from utils import constants

class IntegerOverflowHandler(BaseHandler):

vulnerability\_name = 'Переполнение целых чисел'

def \_\_init\_\_(self):

self.output = []

self.regex\_arithmetic = r"^\s\*([\w:<>]\*)\s\*=\s\*([\w:<>]+|\d)\s\*(\+|\-|\\*|\/)?\s\*([\w:<>]+|\d).\*;"

def parse(self, contexts: List[FunctionContext]):

*"""*

*Проверяет переполнение объявленных переменных, учавствующих в арифметических операциях*

*"""*

total\_errors = 0

for context in contexts:

declared\_vars = context.variables.copy()

int\_vars = []

for i in declared\_vars:

if i.var\_type in constants.int\_types:

int\_vars.append(i)

for line\_number, line in context.source\_code.items():

arithmetic\_matches = re.finditer(self.regex\_arithmetic, line, re.IGNORECASE)

for matchNum, match in enumerate(arithmetic\_matches):

left = None

right = None

if match.group(2).isdigit():

left = int(match.group(2))

else:

for i in int\_vars:

if i.var\_name == match.group(2): left = int(i.value)

if match.group(4).isdigit():

right = int(match.group(4))

else:

for ind in int\_vars:

if ind.var\_name == match.group(4): right = int(ind.value)

if match.group(3) == '\*':

tmp = left \* right

elif match.group(3) == '+':

tmp = left + right

elif match.group(3) == '-':

tmp = left - right

else:

tmp = left // right

tmp\_var = None

for k in int\_vars:

if k.var\_name == match.group(1):

k.value = tmp

tmp\_var = k

try:

assert (constants.integer\_limits[tmp\_var.var\_type].min <= int(tmp\_var.value) <=

constants.integer\_limits[tmp\_var.var\_type].max)

except AssertionError:

total\_errors += 1

self.output.append(f"{total\_errors}) Угроза в методе {context.name}!\n"

f"Переполнение челочисленной переменной <{tmp\_var.var\_name}> типа "

f"<{tmp\_var.var\_type}> (строка {line\_number}) "

f"Выход за границы типа!")

self.output.append(self.vulnerability\_name + ": " + str(total\_errors))

return self.output

**Обработчик пренебрежения безопасным хранением данных.**

Автор: Шабакаев А.Р.

import re

from typing import List

from core.base\_handler import BaseHandler

from core.function\_context import FunctionContext

class DataStorageManagementHandler(BaseHandler):

vulnerability\_name = 'Небезопасное хранение данных'

"""

- BOOL SetFileSecurity(LPCTSTR lpFileName, SECURITY\_INFORMATION SecurityInformation,

PSECURITY\_DESCRIPTOR pSecurityDescriptor); - устанавливает защиту объекта файла или каталога.

- BOOL SetKernelObjectSecurity(HANDLE Handle, SECURITY\_INFORMATION SecurityInformation,

PSECURITY\_DESCRIPTOR SecurityDescriptor); - устанавливает защиту объекта ядра.

- BOOL SetServiceObjectSecurity(SC\_HANDLE hService, SECURITY\_INFORMATION dwSecurityInformation,

PSECURITY\_DESCRIPTOR lpSecurityDescriptor); - устанавливает дескриптор безопасности объекта службы.

- int chmod(char \*pathname,int pmode); - изменяет разрешенный доступ для файла <pathname>.

- int fchmod(int fildes, mode\_t mode); - изменяет разрешенный доступ.

- int fchown(int fd, uid\_t owner, gid\_t group); - изменяет владельца файла.

- int fcntl(int fd, int cmd, ... /\* arg \*/ ); - манипуляции над файловым дескриптором.

- int setgroups(int size, gid\_t list[]); - устанавливает список дополнительных идентификаторов групп.

"""

def \_\_init\_\_(self):

self.pattern = r'(SetFileSecurity|SetKernelObjectSecurity|SetServiceObjectSecurity|chmod|fchmod|fchown|' \

r'fcntl|setgroups)'

self.output = []

def parse(self, contexts: List[FunctionContext]):

total\_errors = 0

for context in contexts:

for line\_number, line in context.source\_code.items():

matches = re.finditer(self.pattern, line)

for match in matches:

total\_errors += 1

self.output.append(f"{total\_errors}) Предупреждение в методе <{context.name}>!\n"

f"Использование функции, регулирующей настройки безопасности системы или файлов "

f"<{match.group(0)}>, которая может привести к проблеме "

f"ненадежного хранения данных (строка {line\_number})\n")

self.output.append(self.vulnerability\_name + ": " + str(total\_errors))

return self.output

**Обработчик утечки информации.**

Автор: Шабакаев А.Р.

import re

from typing import List

from core.base\_handler import BaseHandler

from core.function\_context import FunctionContext

# *TODO: можно добавить поиск вывода исключений в конструкциях try-catch*

class DataLeakHandler(BaseHandler):

vulnerability\_name = 'Утечка информации'

"""

- DWORD GetLastError(); - возваращет код последней ошибки, вызывающего потока.

- HRESULT SHGetFolderPath(HWND hwndOwner, int nFolder, HANDLE hToken, DWORD dwFlags, LPTSTR pszPath); - возвращает

путь к определённой системной директории, тип которой указан в идентификаторе CSIDL.

- LPVOID GetEnvironmentStrings(); - возвраащет переменные окружения для текущего процесса.

- string GetEnvironmentVariable (string variable); - возвращает из текущего процесса значение переменной среды.

- errno - макрос, который возвращает последний номер ошибки.

- char \*getenv(const char \*name); - возвращает указатель на информацию об окружении, ассоциированную с <name>

в таблице информации окружения. Окружение программы может включать такие вещи, как имена путей и устройств.

- char \*strerror(int num); - возвращает указатель на сообщение об ошибке, связанное с номером ошибки <num>.

- void perror(const char \*str); - помещает значение глобальной переменной errno в строку и записывает эту строку в

поток stderr.

"""

def \_\_init\_\_(self):

self.pattern = r'(GetLastError|SHGetFolderPath|GetEnvironmentStrings|GetEnvironmentVariable|errno|getenv|' \

r'strerror|perror)'

self.output = []

def parse(self, contexts: List[FunctionContext]):

total\_errors = 0

for context in contexts:

for line\_number, line in context.source\_code.items():

matches = re.finditer(self.pattern, line)

for match in matches:

total\_errors += 1

if match.group(0) == "errno":

self.output.append(f"{total\_errors}) Предупреждение в методе <{context.name}>!\n"

f"Использование макроса <{match.group(0)}>, который может привести к утечке "

f"системных данных или к расскрытию важной информации (строка {line\_number})\n")

else:

self.output.append(f"{total\_errors}) Предупреждение в методе <{context.name}>!\n"

f"Использование функции <{match.group(0)}>, которая может привести к утечке "

f"системных данных или к расскрытию важной информации (строка {line\_number})\n")

self.output.append(self.vulnerability\_name + ": " + str(total\_errors))

return self.output

**Агрегирующий модуль.**

Автор: Ковалев К.А.

from UI.UI import UI

from core.main\_code\_parser import find\_contexts

from handlers.buffer\_overflow\_handler import BufferOverflowHandler

from handlers.commands\_introduction\_hadler import CommandsIntroductionHandler

from handlers.data\_leak\_handler import DataLeakHandler

from handlers.data\_storage\_handler import DataStorageManagementHandler

from handlers.empty\_catch\_handler import EmptyCatchHandler

from handlers.file\_access\_handler import IncorrectFileAccessHandler

from handlers.format\_string\_handler import FormatStringHandler

from handlers.integer\_overflow\_handler import IntegerOverflowHandler

from handlers.memory\_leak\_handler import MemoryLeakHandler

from handlers.race\_condition\_handler import RaceConditionHandler

from handlers.random\_generator\_handler import RandomGeneratorHandler

from handlers.readers\_writers\_handler import ReadersWritersHandler

from handlers.sql\_injection\_handler import SQLInjectionHandler

from utils.initial\_parse import clean\_code

handlers\_list = {

BufferOverflowHandler.vulnerability\_name: BufferOverflowHandler,

CommandsIntroductionHandler.vulnerability\_name: CommandsIntroductionHandler,

DataLeakHandler.vulnerability\_name: DataLeakHandler,

DataStorageManagementHandler.vulnerability\_name: DataStorageManagementHandler,

EmptyCatchHandler.vulnerability\_name: EmptyCatchHandler,

IncorrectFileAccessHandler.vulnerability\_name: IncorrectFileAccessHandler,

FormatStringHandler.vulnerability\_name: FormatStringHandler,

IntegerOverflowHandler.vulnerability\_name: IntegerOverflowHandler,

MemoryLeakHandler.vulnerability\_name: MemoryLeakHandler,

RaceConditionHandler.vulnerability\_name: RaceConditionHandler,

RandomGeneratorHandler.vulnerability\_name: RandomGeneratorHandler,

ReadersWritersHandler.vulnerability\_name: ReadersWritersHandler,

SQLInjectionHandler.vulnerability\_name: SQLInjectionHandler,

}

if \_\_name\_\_ == '\_\_main\_\_':

ui = UI(handlers\_list.keys())

ui.start\_main(

lambda vulnerability, program: handlers\_list[vulnerability]().parse(find\_contexts(clean\_code(program))))

import re

def replace(match):

return None if match.group(0).startswith('/') else match.group(0)

def clean\_code(program, mode='r'):

pattern = re.compile(r'//.\*?$|/\\*.\*?\\*/|\'(?:\\.|[^\\\'])\*\'|"(?:\\.|[^\\"])\*"', re.DOTALL | re.MULTILINE)

tmp = []

a = open(program, mode)

text = a.read()

cleaned\_code = re.sub(pattern, replace, text)

for line in cleaned\_code.splitlines():

tmp.append(line.lstrip())

a.close()

return tmp

**Модуль первичного лексического анализа кода.**

Автор: Ковалев К.А.

class BaseHandler:

def parse(self, contexts):

pass

from collections import OrderedDict

from typing import List

from core.variable import Variable

class FunctionContext:

global\_mutexes = []

def \_\_init\_\_(self, return\_type, name, parameters):

self.return\_type = return\_type

self.name = name

self.parameters: List[Variable] = parameters

self.variables: List[Variable] = []

self.threads = []

self.source\_code = OrderedDict()

import re

from typing import List

from core import regexs\_constants as rc

from core.function\_context import FunctionContext

from core.thread import Thread

from core.variable import Variable

def find\_contexts(source\_code):

*"""*

*Метод для поиска и анализа контекстов в текущем файле.*

*Контексты в данном случае - глобальные лексемы языка, такие как функции, структуры и классы.*

*В данный момент реализован поиск функций.*

*Изначально ведется разбиение всего кода на контексты, а затем анализ каждого из них, например, на наличие*

*объявленных переменных поток и т.д.*

***:rtype*** *List[FunctionContext]*

*"""*

func\_declarations = get\_func\_declarations(source\_code)

for func in func\_declarations:

func.variables += func.parameters

func.variables += get\_declared\_variables(func.source\_code)

func.threads = get\_declared\_threads(func.source\_code, func.variables)

return func\_declarations

def get\_func\_declarations(source\_code):

*"""*

*Поиск сигнатур функций и формирование из них объектов.*

***:rtype*** *List[FunctionContext]*

*"""*

func\_declarations = []

cur\_func = None

open\_br = 0

close\_br = 0

for line\_number, line in enumerate(source\_code):

line\_number += 1

open\_bracer\_matches = re.finditer(r"{", line)

for \_ in open\_bracer\_matches:

open\_br += 1

close\_bracer\_matches = re.finditer(r"}", line)

for \_ in close\_bracer\_matches:

close\_br += 1

mutex\_match = re.match(rc.mutex\_regexp, line, re.MULTILINE)

if mutex\_match is not None:

FunctionContext.global\_mutexes.append(mutex\_match.group(2))

func\_match = re.match(rc.func\_decl\_regexp, line, re.MULTILINE)

if func\_match is not None:

pointer = func\_match.group(2)

return\_type = func\_match.group(1) + pointer.rstrip() if pointer is not None else func\_match.group(1)

name = func\_match.group(3)

parameters = get\_func\_params(func\_match.group(4), line\_number)

cur\_func = FunctionContext(return\_type, name, parameters)

cur\_func.source\_code[line\_number] = line

elif cur\_func is not None:

if re.match(r"}", line) is not None:

if open\_br != close\_br:

cur\_func.source\_code[line\_number] = line

elif open\_br == close\_br:

cur\_func.source\_code[line\_number] = line

func\_declarations.append(cur\_func)

cur\_func = None

close\_br = 0

open\_br = 0

else:

cur\_func.source\_code[line\_number] = line

return func\_declarations

def get\_func\_params(params\_in\_bracket, line\_number):

*"""*

*Поиск параметров функции.*

***:rtype*** *List[Variable]*

*"""*

parameters\_list = []

if params\_in\_bracket[1:-1] == "":

return parameters\_list

raw\_param\_list = re.split(r",\s\*", params\_in\_bracket[1:-1])

for raw\_parameter in raw\_param\_list:

param\_match = re.match(rc.func\_param\_regexp, raw\_parameter)

pointer = param\_match.group(2)

param\_type = param\_match.group(1) + pointer.rstrip() if pointer is not None else param\_match.group(1)

param\_name = param\_match.group(3)

parameters\_list.append(Variable(raw\_parameter, line\_number, param\_name, param\_type))

return parameters\_list

def get\_declared\_variables(source\_code):

*"""*

*Поиск и инициализация всех объявленных переменных в функции.*

***:rtype*** *List[Variables]*

*"""*

variables\_list = []

for line\_number, line in source\_code.items():

var\_matches = re.finditer(rc.variable\_regexp, line, re.MULTILINE)

for match in var\_matches:

var = Variable(match.group(0).strip(), line\_number, match.group(5), match.group(3))

if match.group(7) is not None:

var.value = match.group(7)[1:].strip()

variables\_list.append(var)

for \_, line in source\_code.items():

var\_matches = re.finditer(rc.assignment\_regexp, line, re.MULTILINE)

for match in var\_matches:

v1 = None

v2 = None

for var in variables\_list:

if match.group(1) == var.var\_name:

v1 = var

if match.group(2) == var.var\_name:

v2 = var

if v1 is not None and v2 is not None:

v1.value = v2.value

if v1 is not None and v2 is None:

v1.value = match.group(2)

return variables\_list

def get\_declared\_threads(source\_code, declared\_variables):

*"""*

*Поиск и инициализация всех объявленных потоков в функции.*

***:rtype****: List[Thread]*

*"""*

threads\_list = []

for line\_number, line in source\_code.items():

matches = re.finditer(rc.thread\_regexp, line)

for match in matches:

cur\_thread\_params = get\_parameters(match.group(0), declared\_variables)

threads\_list.append(

Thread(match.group(0), line\_number, match.group(2), match.group(3), cur\_thread\_params))

return threads\_list

def get\_parameters(raw\_parameters, declared\_variables) -> List[Variable]:

*"""*

*Поиск уже объявленных переменных, которые используются, например, при вызове функции или объявлении потока.*

***:rtype****: List[Variable]*

*"""*

tmp = re.search(r"\(.\*\)", raw\_parameters).group(0)[1:-1]

parameters\_list = re.split(r",", tmp)

v\_parameters\_list = []

for index, parameter in enumerate(parameters\_list):

for var in declared\_variables:

if re.sub(r'\[.\*\]', '', parameter.strip()) == var.var\_name:

v\_parameters\_list.append(var)

break

return v\_parameters\_list

func\_decl\_regexp = \

r"^(char|unsigned char|signed char|int|byte|unsigned int|signed int|short int|unsigned short int|" \

r"signed short int|long int|singed long int|unsigned long int|long long int|signed long long int|" \

r"unsigned long long int|float|double|long double|wchar\_t|short|long|void)\s+" \

r"(\\*+\s\*)?" \

r"(\w+)\s\*" \

r"(\(.\*\))"

func\_param\_regexp = \

r"(char|unsigned char|signed char|int|byte|unsigned int|signed int|short int|unsigned short int|" \

r"signed short int|long int|singed long int|unsigned long int|long long int|signed long long int|" \

r"unsigned long long int|float|double|long double|wchar\_t|short|long)\s+" \

r"(\\*+\s\*)?" \

r"(\w+)\s\*" \

r"(\[.\*\])\*"

variable\_regexp = \

r"^(extern\s+)?" \

r"(const\s+)?" \

r"(char|unsigned char|signed char|int|byte|short|long|ofstream|ifstream||wchar\_t" \

r"|unsigned int|signed int|short int|unsigned short int|signed short int|long int|singed long int" \

r"|unsigned long int|long long int|signed long long int|unsigned long long int|float|double|long double)\s+" \

r"(\\*+\s\*)?" \

r"(\w+)\s\*" \

r"(\[.+]\s\*)?" \

r"(=.+)?\s\*;"

assignment\_regexp = r"(\w+)\s\*=\s\*(.+)\s\*;"

thread\_regexp = r'^(std::)?thread\s+(\w+)\s\*\(\s\*(\w+)\s\*(,\s\*[\w,"]\*\s\*)\*\)\s\*;'

mutex\_regexp = r'^(std::)?mutex\s+(\w+);'

from typing import List

from core.variable import Variable

class Thread:

def \_\_init\_\_(self, full\_declaration, line\_appeared, thread\_name, runnable\_function, parameter\_list: List[Variable]):

self.full\_declaration = full\_declaration

self.line\_appeared = line\_appeared

self.thread\_name = thread\_name

self.runnable\_function = runnable\_function

self.parameter\_list = parameter\_list

import collections

import re

def is\_pointer(variable\_declaration):

return True if re.search(r"\\*", variable\_declaration) else False

def is\_array(variable\_declaration):

return True if re.search(r"\[.\*]", variable\_declaration) else False

class Variable:

def \_\_init\_\_(self, full\_declaration, line\_appeared, var\_name, var\_type, value=None):

self.full\_declaration = full\_declaration

self.line\_appeared = line\_appeared

self.var\_name = var\_name

self.var\_type = var\_type

self.value = value

**Пользовательский интерфейс.**

Автор: Щекочихин Д.А.

from tkinter import \*

FONT = ("Helvetica", 12)

BUTTON\_WIDTH = 13

BUTTON\_HEIGHT = 1

MAIN\_COLOR = "#DDC2BD"

def create\_check\_button(parent: Frame, bg, text: str, height: int, width: int, anchor, value: int = 0):

value = IntVar(value=value)

ch\_button = Checkbutton(parent, bg=bg, font=("Helvetica", 12), activebackground=bg, variable=value, text=text, height=height,

width=width, anchor=anchor, fg="black")

ch\_button.pack()

return value, ch\_button

def create\_button(parent: Widget, text, ui, func):

return Button(parent, width=BUTTON\_WIDTH, height=BUTTON\_HEIGHT, state=NORMAL if ui.default\_tests

else DISABLED, font=FONT, text=text, bd=2, anchor=W,

command=func)

def create\_listbox(parent: Widget, select\_mode, border\_width: int, action=None, font=FONT):

listbox = Listbox(parent, font=font, width=20, selectmode=select\_mode, exportselection=False,

borderwidth=border\_width)

listbox.pack(fill=Y)

listbox.bind('<<ListboxSelect>>', action)

return listbox

def create\_scroll\_box(frame, item, orient=HORIZONTAL):

scrollbar = Scrollbar(frame, orient=orient)

scrollbar.config(command=item.xview if orient == HORIZONTAL else item.yview)

scrollbar.pack(side=BOTTOM if orient == HORIZONTAL else RIGHT, fill=X if orient == HORIZONTAL else Y)

item.config(xscrollcommand=scrollbar.set if orient == HORIZONTAL else NONE,

yscrollcommand=scrollbar.set if orient == VERTICAL else NONE)

item.pack(side=TOP if orient == HORIZONTAL else LEFT)

return scrollbar

def create\_code\_scroll\_box(frame, item, orient=HORIZONTAL):

scrollbar = Scrollbar(frame, orient=orient)

scrollbar.config(command=item.xview if orient == HORIZONTAL else item.yview)

item.config(xscrollcommand=scrollbar.set if orient == HORIZONTAL else NONE,

yscrollcommand=scrollbar.set if orient == VERTICAL else NONE)

return scrollbar

def create\_code\_listbox(parent: Widget, select\_mode, border\_width: int, action=None, font=FONT, bg="white"):

listbox = Listbox(parent, font=font, width=129, height=18, selectmode=select\_mode, exportselection=False,

borderwidth=border\_width, bg=bg)

listbox.pack(fill=BOTH, expand=TRUE)

listbox.bind('<<ListboxSelect>>', action)

return listbox

import threading

def create\_workers(number\_of\_threads, work, queue, \*args):

for \_ in range(number\_of\_threads):

t = threading.Thread(target=work, args=(queue, \*args,))

t.daemon = True

t.start()

return

def create\_jobs(args, queue):

if not args:

from queue import Empty

raise Empty()

for arg in args:

queue.put(arg)

queue.join()

return

import \_tkinter

from tkinter import Listbox

def get\_selected\_t(listbox: Listbox):

try:

selected = [listbox.get(idx) for idx in listbox.curselection()]

if not len(selected):

raise \_tkinter.TclError

return selected

except \_tkinter.TclError:

from tkinter import messagebox

messagebox.showwarning("Предупреждение", "Тесты не выбраны!")

return ()

def get\_selected\_p(chosen\_programs, listbox: Listbox):

try:

programs\_path = []

selected = [listbox.get(idx) for idx in listbox.curselection()]

for select in selected:

for program in chosen\_programs:

if select in program: programs\_path.append(program)

if not len(programs\_path):

raise \_tkinter.TclError

return programs\_path

except \_tkinter.TclError:

from tkinter import messagebox

messagebox.showwarning("Предупреждение", "Программы не выбраны!")

return ()

def get\_selected\_v(check\_buttons\_array: tuple or list):

try:

vulnerabilities = [item[1].cget('text') for item in check\_buttons\_array if item[0].get()]

if not len(vulnerabilities):

raise \_tkinter.TclError

return vulnerabilities

except \_tkinter.TclError:

from tkinter import messagebox

messagebox.showwarning("Предупреждение", "Уязвимости не выбраны")

return ()

#!/usr/bin/env python3  
# -\*- coding: UTF-8 -\*-  
import os  
from os.path import basename  
from queue import Queue, Empty  
from tkinter import messagebox  
from tkinter.filedialog import askopenfilenames  
from UI.create\_utils import \*  
from UI.multithreading import create\_workers, create\_jobs  
from UI.select\_utils import \*  
  
FONT = ("Helvetica", 12)  
BUTTON\_WIDTH = 13  
BUTTON\_HEIGHT = 1  
MAIN\_COLOR = "white"  
  
  
def fill\_listbox(listbox: Listbox, items: tuple or list):  
 for item in items:  
 listbox.insert(END, item)  
  
  
def write\_to\_text(text\_field: Text, text: str or list or tuple, option: str = NONE):  
 text\_field.configure(state=NORMAL)  
 if option == 'CLEAR':  
 text\_field.delete('1.0', END)  
 if isinstance(text, (list, tuple)):  
 for line in text:  
 text\_field.insert(END, str(line) + '\n')  
 else:  
 text\_field.insert(END, text)  
 text\_field.configure(state=DISABLED)  
  
  
class Form(object):  
 def \_\_init\_\_(self, title: str, geometry: str, font: (str, int)):  
 self.root = Tk()  
 self.root.iconbitmap(os.getcwd() + "\\resources\\icon.ico")  
 self.root.configure(bg=MAIN\_COLOR)  
 self.root.update()  
 self.root.title(title)  
 x = int((self.root.winfo\_screenwidth() - self.root.winfo\_reqwidth()) / 2) - 555  
 y = int((self.root.winfo\_screenheight() - self.root.winfo\_reqheight()) / 2) - 250  
 self.root.geometry(geometry + "+" + str(x) + "+" + str(y))  
 self.root.resizable(width=True, height=True)  
 self.root.minsize(self.root.winfo\_width(), self.root.winfo\_height())  
 self.root.protocol('WM\_DELETE\_WINDOW', self.exit)  
 self.font = font  
  
 def error(self, message):  
 self.root.withdraw()  
 messagebox.showerror("Error", message)  
  
 def start(self):  
 try:  
 self.root.mainloop()  
 except Exception as e:  
 self.error(e.args)  
  
 def exit(self):  
 try:  
 self.root.destroy()  
 except Exception as e:  
 self.error(e.args)  
  
  
def restart\_program():  
 executable = sys.executable  
 os.execl(executable, executable, \*sys.argv)  
  
  
def find\_vulnerabilities(programs: tuple or list, vulnerabilities: tuple or list, handler):  
 *"""* ***:param*** *programs: список путей к файлам, которые будут анализироваться* ***:param*** *vulnerabilities: список выбранных уязвимостей, которые будут искаться \  
 (их названия согласно декларации в классе)* ***:param*** *handler: лямба функция, которая объявлена в main.py при вызове start\_main* ***:return****:  
 """* def get\_vuln\_out(cur\_p):  
 vuln\_out = []  
 for vulnerability in vulnerabilities:  
 if handler(vulnerability, cur\_p):  
 vuln\_out.append((vulnerability, handler(vulnerability, cur\_p)))  
 return dict(vuln\_out)  
  
 if not (programs and vulnerabilities):  
 return 0  
 try:  
 queue = Queue()  
 create\_workers(len(programs), show\_vulns, queue)  
 job\_args = []  
 for program in programs:  
 if get\_vuln\_out(program):  
 job\_args.append((program, get\_vuln\_out(program), open(program, 'r').read().splitlines()))  
 create\_jobs(job\_args, queue)  
 except Empty:  
 from tkinter import messagebox  
 messagebox.showinfo("Информация!", "Уязвимости не были обнаружены!")  
  
  
def show\_vulns(queue):  
 def get\_rank(var):  
 count = 1  
 var //= 10  
 while var > 0:  
 var //= 10  
 count += 1  
 return count  
  
 v\_form = None  
 try:  
 program, content, program\_code = queue.get(block=True)  
 queue.task\_done()  
 f = open(program, 'a') # дозапись в файл  
 for \_, value in content.items():  
 f.write(value[1])  
 v\_form = Form(program + ' - анализ программы', '1100x700', FONT)  
 x = int((v\_form.root.winfo\_screenwidth() - v\_form.root.winfo\_reqwidth()) / 2) - 550  
 y = int((v\_form.root.winfo\_screenheight() - v\_form.root.winfo\_reqheight()) / 2) - 300  
 v\_form.root.geometry("1100x700" + "+" + str(x) + "+" + str(y))  
  
 code\_frame = Frame(v\_form.root, bg=MAIN\_COLOR, width=1100, height=400, borderwidth=2)  
 vuln\_frame = Frame(v\_form.root,  
 relief="ridge", bg=MAIN\_COLOR, width=300, height=270)  
 loc\_frame = Frame(v\_form.root,  
 relief="ridge", bg=MAIN\_COLOR, width=1100, height=270)  
  
 code\_lbl = Label(code\_frame, text='Исходный код:', font=v\_form.font, anchor='w', bg=MAIN\_COLOR)  
 loc\_lbl = Label(loc\_frame, text='Местоположение:', font=v\_form.font, anchor='w', bg=MAIN\_COLOR)  
 v\_lbl = Label(vuln\_frame, text='Найденные уязвимости:', font=v\_form.font, width=28,  
 anchor='w', bg=MAIN\_COLOR)  
  
 code\_extra\_frame = Frame(code\_frame, bg=MAIN\_COLOR)  
 code\_lb = create\_code\_listbox(code\_extra\_frame, EXTENDED, 1, font=("Helvetica", 11), bg="#EEEDF1")  
 create\_scroll\_box(code\_extra\_frame, code\_lb, VERTICAL)  
  
 pr\_len = len(program)  
 max\_rank = get\_rank(pr\_len)  
 tmp\_list = []  
 for index, line in enumerate(program\_code):  
 cur\_rank = get\_rank(index + 1)  
 cur\_indent = 2 + (max\_rank - cur\_rank) \* 2  
 tmp\_list.append(' ' \* cur\_indent + str(index + 1) + "| " + line.replace("\t", " "))  
 fill\_listbox(code\_lb, tmp\_list)  
  
 code\_extra\_frame.grid(row=1, column=0, columnspan=2, sticky=E, padx=10, pady=10)  
 code\_lbl.grid(row=0, column=0, sticky=NW, padx=10, pady=(5, 0))  
  
 loc\_extra\_frame = Frame(loc\_frame, bg=MAIN\_COLOR)  
 where\_tb = Text(loc\_extra\_frame, borderwidth=1, width=80, height=13, relief="groove", wrap=WORD,  
 state='disabled', font=FONT, bg="#EEEDF1")  
 create\_scroll\_box(loc\_extra\_frame, where\_tb, VERTICAL)  
 loc\_lbl.grid(row=0, column=0, sticky=NW, padx=10, pady=(5, 0))  
 loc\_extra\_frame.grid(row=1, column=0, sticky=EW, padx=10)  
  
 code\_frame.grid\_propagate(0)  
 vuln\_frame.grid\_propagate(0)  
 loc\_frame.grid\_propagate(0)  
  
 found\_lb = Listbox(vuln\_frame, font=FONT, width=70, height=40, selectmode=SINGLE, exportselection=False, bg="#EEEDF1", borderwidth=1)  
 found\_lb.bind('<<ListboxSelect>>', lambda event: write\_to\_text(where\_tb,  
 content[  
 found\_lb.get(  
 found\_lb.curselection())],  
 'CLEAR'))  
  
 fill\_listbox(found\_lb, content.keys())  
  
 v\_lbl.grid(row=0, column=0, sticky=W)  
 found\_lb.grid(row=1, column=0, sticky=EW)  
  
 code\_frame.grid(row=0, column=0, columnspan=2, sticky=EW, pady=(0, 10))  
 vuln\_frame.grid(row=1, column=0, pady=(0, 5), padx=(10, 0), sticky=W)  
 loc\_frame.grid(row=1, column=1, pady=(0, 5), sticky=W)  
  
 v\_form.start()  
 except Exception as e:  
 from tkinter import messagebox  
 v\_form.error(e.args)  
 v\_form.exit()  
  
  
def upload\_chosen\_p():  
 *"""  
 Возвращает список путей к выбранным программам* ***:return****: List[path\_to\_file]  
 """* programs = []  
 try:  
 programs = list(askopenfilenames(title='Открыть программы для анализа', initialdir=os.getcwd(),  
 filetypes=[("CPP", ".cpp")]))  
 except IOError:  
 messagebox.showwarning("Предупреждение", "Папка \"default\_tests\" не существует")  
 return programs  
  
  
def upload\_default\_t():  
 *"""  
 Возвращает список путей к стандартным тестам* ***:return****: List[path\_to\_file]  
 """* tests = []  
 try:  
 tests = [os.getcwd() + '\\default\_tests\\' + file for file in os.listdir(os.getcwd() + '\\default\_tests') if  
 file.endswith(".cpp")]  
 except IOError:  
 from tkinter import messagebox  
 messagebox.showwarning("Предупреждение", "Директория \"default\_tests\" не существует")  
 except Exception as e:  
 from tkinter import messagebox  
 messagebox.showwarning("Предупреждение", e.args)  
 return tests  
  
  
class UI(object):  
  
 def \_\_init\_\_(self, vulnerability\_names):  
 self.vulnerability\_names = vulnerability\_names  
 self.default\_tests = upload\_default\_t()  
 self.chosen\_programs = []  
  
 def fill\_programs\_list(self, list\_box: Listbox):  
 self.chosen\_programs = upload\_chosen\_p()  
 fill\_listbox(list\_box, [basename(file) for file in self.chosen\_programs])  
  
 def start\_main(self, handler):  
 main\_form = None  
 try:  
 main\_form = Form('', '1110x500', FONT)  
 main\_form.root.focus\_force()  
  
 main\_menu = Menu()  
 file\_menu = Menu(tearoff=0)  
 file\_menu.add\_command(label="Открыть", command=lambda: self.fill\_programs\_list(programs\_listb))  
 file\_menu.add\_command(label="Перезапуск", command=lambda: restart\_program())  
 file\_menu.add\_command(label="Выход", command=lambda: main\_form.root.quit())  
 main\_menu.add\_cascade(label="Файл", menu=file\_menu)  
 main\_form.root.config(menu=main\_menu)  
  
 v\_frame = Frame(main\_form.root, borderwidth=2, relief="groove", bg="#BDDDC2", width=300, height=390)  
  
 t\_frame = LabelFrame(main\_form.root, font=("Helvetica", 11), text="Стандартные тесты", borderwidth=3,  
 relief="groove", bg="#EEEDF1", width=340, height=390)  
 p\_frame = LabelFrame(main\_form.root, font=("Helvetica", 11), text="Программы", borderwidth=3,  
 relief="groove", bg="#EEEDF1", width=340, height=390)  
 b\_frame1 = Frame(t\_frame, pady=7, bg="#EEEDF1")  
 b\_frame2 = Frame(p\_frame, pady=7, bg="#EEEDF1")  
  
 tests\_listb = create\_listbox(t\_frame, EXTENDED, 0)  
 programs\_listb = create\_listbox(p\_frame, EXTENDED, 0)  
  
 create\_scroll\_box(t\_frame, tests\_listb, VERTICAL)  
 create\_scroll\_box(p\_frame, programs\_listb, VERTICAL)  
  
 v\_to\_check = [create\_check\_button(v\_frame, bg="#BDDDC2", text=vulnerability, height=1, width=30, anchor=W)  
 for vulnerability in self.vulnerability\_names]  
  
 l1 = lambda: find\_vulnerabilities(  
 [os.getcwd() + '\\default\_tests\\' + file for file in get\_selected\_t(tests\_listb)],  
 get\_selected\_v(v\_to\_check), handler)  
 l2 = lambda: find\_vulnerabilities(self.default\_tests, get\_selected\_v(v\_to\_check), handler)  
 l3 = lambda: find\_vulnerabilities(get\_selected\_p(self.chosen\_programs, programs\_listb),  
 get\_selected\_v(v\_to\_check), handler)  
 l4 = lambda: find\_vulnerabilities(self.chosen\_programs, get\_selected\_v(v\_to\_check), handler)  
  
 check1 = create\_button(b\_frame1, "Проверить", self, l1)  
 check2 = create\_button(b\_frame2, "Проверить", self, l3)  
 check\_all1 = create\_button(t\_frame, "Проверить все", self, l2)  
 check\_all2 = create\_button(p\_frame, "Проверить все", self, l4)  
  
 version\_lbl = Label(main\_form.root, text='Версия: 1.0.0', font=FONT, anchor='w', bg=MAIN\_COLOR)  
  
 v\_frame.grid(row=0, column=0, padx=(40, 0), pady=(40, 0), sticky=N)  
 t\_frame.grid(row=0, column=1, padx=(50, 20), pady=(40, 0), sticky=N)  
 p\_frame.grid(row=0, column=2, pady=(40, 0), sticky=N)  
 version\_lbl.grid(row=1, column=2, pady=(30, 0), sticky=SE)  
  
 check\_all1.pack(side=TOP)  
 check1.pack(side=TOP)  
 check\_all2.pack(side=TOP)  
 check2.pack(side=TOP)  
  
 b\_frame1.pack()  
 b\_frame2.pack()  
  
 v\_frame.pack\_propagate(0)  
 t\_frame.pack\_propagate(0)  
 p\_frame.pack\_propagate(0)  
  
 fill\_listbox(tests\_listb, [basename(file) for file in self.default\_tests])  
  
 main\_form.start()  
 except Exception as e:  
 from tkinter import messagebox  
 main\_form.error(e.args)  
 main\_form.exit()

**Генератор тестового кода С++.**

Автор: Ковалев К.А.

import os  
import re  
import main  
from code\_generator.var\_genetor import \*  
from core.main\_code\_parser import find\_contexts  
from core.variable import is\_pointer, is\_array  
from utils.constants import \*  
from utils.initial\_parse import clean\_code  
  
v\_funcs = [gen\_var, gen\_var\_with\_value, gen\_buffer, gen\_pointer]  
p\_funcs = [gen\_var, gen\_pointer]  
  
  
class CodeGenerator:  
 cur\_available\_funcs = []  
 mutexes = []  
 final\_output = [["Переполнение буфера", 0], ["Внедрение команд", 0], ["Утечка информации", 0],  
 ["Небезопасное хранение данных", 0], ["Пренебрежение обработкой исключений", 0],  
 ["Некорректный доступ к файлам", 0], ["Ошибка форматной строки", 0], ["Переполнение целых чисел", 0],  
 ["Ошибка высвобождения памяти", 0], ["Состояние гонки", 0],  
 ["Случайные числа криптографического характера", 0], ["Читатели Писатели", 0],  
 ["SQL инъекции", 0]]  
 final\_output\_for\_test = [["Переполнение буфера", 0], ["Внедрение команд", 0], ["Утечка информации", 0],  
 ["Небезопасное хранение данных", 0], ["Пренебрежение обработкой исключений", 0],  
 ["Некорректный доступ к файлам", 0], ["Ошибка форматной строки", 0],  
 ["Переполнение целых чисел", 0],  
 ["Ошибка высвобождения памяти", 0], ["Состояние гонки", 0],  
 ["Случайные числа криптографического характера", 0], ["Читатели Писатели", 0],  
 ["SQL инъекции", 0]]  
 line\_number = 1  
  
 # генерирует рандомный cout  
 # def gen\_cout(self, indent, params):  
 # sample\_text = ""  
 # param = r\_v(params)  
 # for \_ in range(3):  
 # sample\_text += r\_v(sample\_words) + " "  
 # code = indent + "cout<<\"" + sample\_text + "\"<<" + param + ";\n"  
 # return code  
  
 # генерирует if statement с рандомным содержанием  
 def gen\_cond(self, indent, params):  
 own\_indent = "\t"  
 declaration = indent + "if(" + r\_v(params)[0] + " " + r\_v(cond) + " " + str(rng(0, 500)) + ") {\n"  
 self.line\_number += 1  
 body = ""  
 for index in range(2):  
 body += r\_v(self.cond\_generators)(self, own\_indent + indent, params)  
 code = declaration + body + indent + "}\n"  
 self.line\_number += 1  
 return code  
  
 # генерирует блок try-catch с рандомным содержанием  
 def gen\_try\_catch(self, indent, params):  
 own\_indent = "\t"  
 try\_declaration = indent + "try {\n"  
 self.line\_number += 1  
 try\_body = ""  
 for index in range(3):  
 try\_body += r\_v(self.catch\_generators)(self, own\_indent + indent, params)  
 catch\_declaration = indent + "catch (Exception\_" + str(rng(1, 20)) + " err) {\n"  
 self.line\_number += 1  
 catch\_body = ""  
 if rng(0, 1) == 1:  
 for index in range(2):  
 catch\_body += r\_v(self.catch\_generators)(self, own\_indent + indent, params)  
 else:  
 catch\_body = "\n"  
 for i in self.final\_output:  
 if "Пренебрежение обработкой исключений" in i[0]:  
 i[0] += f"\nСтрока {self.line\_number}\n"  
 i[1] += 1  
 self.line\_number += 1  
 code = try\_declaration + try\_body + indent + "}\n" + catch\_declaration + catch\_body + indent + "}\n"  
 self.line\_number += 2  
 return code  
  
 # генерирует ошибку переполнения буфера  
 def buff\_error(self, indent, params):  
 funcs = [  
 {"strcpy(": 2},  
 {"printf(Overflow %s and %s, ": 2},  
 {"memcpy(": 3},  
 {"strcat(": 2},  
 {"gets(": 1},  
 {"sprintf(": 3},  
 {"vsprintf(": 3},  
 {"strncpy(": 3},  
 {"scanf(%s, ": 1},  
 ]  
 cur\_func\_dict = r\_v(funcs)  
 cur\_func = list(cur\_func\_dict.keys())[0]  
 n = list(cur\_func\_dict.values())[0]  
 is\_nice = False  
 chosen\_param\_list = []  
 if "printf" in cur\_func:  
 for i in self.final\_output:  
 if "Ошибка форматной строки" in i[0]:  
 i[0] += f"\nСтрока {self.line\_number}\n"  
 i[1] += 1  
 while not is\_nice:  
 chosen\_param\_list = []  
 for p in gen\_n\_rands(n, 0, len(params) - 1):  
 if is\_pointer(params[p][1]) or is\_array(params[p][1]):  
 for i in self.final\_output:  
 if "Переполнение буфера" in i[0]:  
 i[0] += f"\nСтрока {self.line\_number}\n"  
 i[1] += 1  
 is\_nice = True  
 chosen\_param\_list.append(params[p][0])  
 code = indent + cur\_func + ", ".join(chosen\_param\_list) + ");\n"  
 self.line\_number += 1  
 return code  
  
 # генерирует ошибку встраивания команд  
 def c\_intr\_error(self, indent, params):  
 funcs = [  
 {"system(": 1},  
 {"popen(": 2},  
 {"execlp(": 3},  
 {"execvp(": 2},  
 {"ShellExecute(": 6},  
 ]  
 cur\_func\_dict = r\_v(funcs)  
 cur\_func = list(cur\_func\_dict.keys())[0]  
 n = list(cur\_func\_dict.values())[0]  
 chosen\_param\_list = [params[param][0] for param in gen\_n\_rands(n, 0, len(params) - 1)]  
 code = indent + cur\_func + ", ".join(chosen\_param\_list) + ");\n"  
 for i in self.final\_output:  
 if "Внедрение команд" in i[0]:  
 i[0] += f"\nСтрока {self.line\_number}\n"  
 i[1] += 1  
 self.line\_number += 1  
 return code  
  
 # генерирует ошибку утечки информации  
 def data\_leak(self, indent, params):  
 funcs = [  
 {"GetLastError(": 0},  
 {"SHGetFolderPath(": 5},  
 {"GetEnvironmentStrings(": 0},  
 {"GetEnvironmentVariable(": 1},  
 {"errno": 0},  
 {"getenv(": 1},  
 {"strerror(": 1},  
 {"perror(": 1}  
 ]  
 cur\_func\_dict = r\_v(funcs)  
 cur\_func = list(cur\_func\_dict.keys())[0]  
 if cur\_func == "errno":  
 code = indent + "cout<<" + cur\_func + ";\n"  
 for i in self.final\_output:  
 if "Утечка информации" in i[0]:  
 i[0] += f"\nСтрока {self.line\_number}\n"  
 i[1] += 1  
 self.line\_number += 1  
 return code  
 n = list(cur\_func\_dict.values())[0]  
 chosen\_param\_list = [params[param][0] for param in gen\_n\_rands(n, 0, len(params) - 1)]  
 code = indent + cur\_func + ", ".join(chosen\_param\_list) + ");\n"  
 for i in self.final\_output:  
 if "Утечка информации" in i[0]:  
 i[0] += f"\nСтрока {self.line\_number}\n"  
 i[1] += 1  
 self.line\_number += 1  
 return code  
  
 def storage\_error(self, indent, params):  
 funcs = [  
 {"SetFileSecurity(": 3},  
 {"SetKernelObjectSecurity(": 3},  
 {"SetServiceObjectSecurity(": 3},  
 {"chmod(": 2},  
 {"fchmod(": 2},  
 {"fchown(": 3},  
 {"fcntl(": 3},  
 {"setgroups(": 2}  
 ]  
 cur\_func\_dict = r\_v(funcs)  
 cur\_func = list(cur\_func\_dict.keys())[0]  
 n = list(cur\_func\_dict.values())[0]  
 chosen\_param\_list = [params[param][0] for param in gen\_n\_rands(n, 0, len(params) - 1)]  
 code = indent + cur\_func + ", ".join(chosen\_param\_list) + ");\n"  
 if cur\_func == "chmod(":  
 for i in self.final\_output:  
 if "Некорректный доступ к файлам" in i[0]:  
 i[0] += f"\nСтрока {self.line\_number}\n"  
 i[1] += 1  
 for i in self.final\_output:  
 if "Небезопасное хранение данных" in i[0]:  
 i[0] += f"\nСтрока {self.line\_number}\n"  
 i[1] += 1  
 self.line\_number += 1  
 return code  
  
 def file\_error(self, indent, params):  
 funcs = [  
 {"mkdir(": 1},  
 {"mktemp(": 1},  
 {"rmdir(": 1},  
 {"chmod(": 2},  
 {"utime(": 2}  
 # {"open": 0}  
 ]  
 cur\_func\_dict = r\_v(funcs)  
 cur\_func = list(cur\_func\_dict.keys())[0]  
 n = list(cur\_func\_dict.values())[0]  
 # if cur\_func == "open":  
 # code = indent + r\_v(params)[0] + ".open(\"D://" + r\_v(sample\_words) + "\");\n"  
 # for i in self.final\_output:  
 # if "Некорректный доступ к файлам" in i[0]:  
 # i[0] += f"\nСтрока {self.line\_number}\n"  
 # i[1] += 1  
 # self.line\_number += 1  
 # return code  
 if cur\_func == "chmod(":  
 for i in self.final\_output:  
 if "Небезопасное хранение данных" in i[0]:  
 i[0] += f"\nСтрока {self.line\_number}\n"  
 i[1] += 1  
 chosen\_param\_list = [params[param][0] for param in gen\_n\_rands(n, 0, len(params) - 1)]  
 code = indent + cur\_func + ", ".join(chosen\_param\_list) + ");\n"  
 for i in self.final\_output:  
 if "Некорректный доступ к файлам" in i[0]:  
 i[0] += f"\nСтрока {self.line\_number}\n"  
 i[1] += 1  
 self.line\_number += 1  
 return code  
  
 def format\_error(self, indent, params):  
 is\_nice = False  
 chosen\_param\_list = []  
 while not is\_nice:  
 chosen\_param\_list = []  
 for p in gen\_n\_rands(2, 0, len(params) - 1):  
 if is\_pointer(params[p][1]) or is\_array(params[p][1]):  
 for i in self.final\_output:  
 if "Переполнение буфера" in i[0]:  
 i[0] += f"\nСтрока {self.line\_number}\n"  
 i[1] += 1  
 is\_nice = True  
 chosen\_param\_list.append(params[p][0])  
 code = indent + "printf(Format %s and %s, " + ", ".join(chosen\_param\_list) + ");\n"  
 for i in self.final\_output:  
 if "Ошибка форматной строки" in i[0]:  
 i[0] += f"\nСтрока {self.line\_number}\n"  
 i[1] += 1  
 self.line\_number += 1  
 return code  
  
 def iover\_error(self, indent, params):  
 chosen\_var\_list = []  
 for \_ in range(3):  
 chosen\_var\_list.append(gen\_integer\_with\_value(indent, params))  
 self.line\_number += 1  
 sample\_code = ""  
 for index in range(1):  
 sample\_code += r\_v(self.iover\_generators)(self, indent, params)  
 code = "".join(chosen\_var\_list) + sample\_code + indent + chosen\_var\_list[0].split()[1] + " = " + \  
 chosen\_var\_list[1].split()[1] + \  
 " + " + chosen\_var\_list[2].split()[1] + ";\n"  
 for i in self.final\_output:  
 if "Переполнение целых чисел" in i[0]:  
 i[0] += f"\nСтрока {self.line\_number}\n"  
 i[1] += 1  
 self.line\_number += 1  
 return code  
  
 def free\_error(self, indent, params):  
 var\_type = r\_v(constants.types)  
 var\_name = "var\_" + str(rng(1, max\_val))  
 while var\_name in params:  
 var\_name = "var\_" + str(rng(1, max\_val))  
 code = indent + var\_type + " \*" + var\_name + " = new " + var\_type + ";\n"  
 params.append((var\_name, code))  
  
 for i in self.final\_output:  
 if "Ошибка высвобождения памяти" in i[0]:  
 i[0] += f"\nСтрока {self.line\_number}\n"  
 i[1] += 1  
 self.line\_number += 1  
 return code  
  
 def race\_error(self, indent, params):  
 code = ""  
 for index in range(rng(2, 5)):  
 for i in range(rng(2, 3)):  
 code += r\_v(self.random\_code\_generators)(self, indent, params)  
 chosen\_param\_list = [params[param][0] for param in gen\_n\_rands(3, 0, len(params) - 1)]  
 code += indent + "std::thread t" + str(index) + "(" + r\_v(self.cur\_available\_funcs) + ", " + ", ".join(  
 chosen\_param\_list) + ");\n"  
 for i in self.final\_output:  
 if "Состояние гонки" in i[0]:  
 i[0] += f"\nСтрока {self.line\_number}\n"  
 i[1] += 1  
 self.line\_number += 1  
 return code  
  
 def rng\_error(self, indent, params):  
 funcs = [  
 {"srand(": 1},  
 {"rand(": 0},  
 {"uniform\_real\_distribution(": 0}  
 ]  
 cur\_func\_dict = r\_v(funcs)  
 cur\_func = list(cur\_func\_dict.keys())[0]  
 if cur\_func == "uniform\_real\_distribution(":  
 code = indent + "std::uniform\_real\_distribution<> dis(" + str(rng(1, 3)) + ", " + str(rng(1, 9)) + ");\n"  
 for i in self.final\_output:  
 if "Случайные числа криптографического характера" in i[0]:  
 i[0] += f"\nСтрока {self.line\_number}\n"  
 i[1] += 1  
 self.line\_number += 1  
 return code  
 if cur\_func == "srand(":  
 code = indent + cur\_func + str(rng(3, 100)) + ");\n"  
 for i in self.final\_output:  
 if "Случайные числа криптографического характера" in i[0]:  
 i[0] += f"\nСтрока {self.line\_number}\n"  
 i[1] += 1  
 self.line\_number += 1  
 return code  
 else:  
 code = indent + cur\_func + ");\n"  
 for i in self.final\_output:  
 if "Случайные числа криптографического характера" in i[0]:  
 i[0] += f"\nСтрока {self.line\_number}\n"  
 i[1] += 1  
 self.line\_number += 1  
 return code  
  
 def readers\_error(self, indent, params):  
 code = ""  
 for index in range(rng(2, 5)):  
 for i in range(rng(2, 3)):  
 code += r\_v(self.random\_code\_generators)(self, indent, params)  
 chosen\_param\_list = [params[param][0] for param in gen\_n\_rands(4, 0, len(params) - 1)]  
 code += indent + "std::thread t" + str(index) + "(" + r\_v(self.cur\_available\_funcs) + ", " + ", ".join(  
 chosen\_param\_list) + ");\n"  
 for i in self.final\_output:  
 if "Читатели Писатели" in i[0]:  
 i[0] += f"\nСтрока {self.line\_number}\n"  
 i[1] += 1  
 self.line\_number += 1  
 return code  
  
 def sql\_error(self, indent, params):  
 funcs = ["execute(", "executeQuery("]  
 cur\_func = r\_v(funcs)  
 sample\_text = ""  
 for \_ in range(3):  
 sample\_text += r\_v(sample\_words) + "'"  
 random = rng(0, 1)  
 if random == 0:  
 code = indent + cur\_func + "\"" + sample\_text + "\");\n"  
 else:  
 code = indent + cur\_func + r\_v(params)[0] + ");\n"  
 for i in self.final\_output:  
 if "SQL инъекции" in i[0]:  
 i[0] += f"\nСтрока {self.line\_number}\n"  
 i[1] += 1  
 self.line\_number += 1  
 return code  
  
 def gen\_sample\_function(self, vulnerabilities, file):  
 indent = "\t"  
 # Генерация сигнатуры функции  
 func\_type = r\_v(func\_types)  
 func\_name = "function\_" + str(rng(0, 20))  
 generated\_vars = []  
 func\_params = []  
 for index in range(rng(1, 4)):  
 func\_params.append(r\_v(p\_funcs)(indent, generated\_vars))  
 file.write(func\_type + " " + func\_name + "(" + ", ".join([param[1:-2] for param in func\_params]) + ") {\n")  
 self.line\_number += 1  
 # Генерация переменных  
 if "Читатели Писатели" in vulnerabilities:  
 mutex = r\_v(self.mutexes)  
 if rng(0, 1) == 1:  
 file.write("\t" + mutex + ".lock();\n")  
 self.line\_number += 1  
 if "Переполнение буфера" in vulnerabilities:  
 file.write(gen\_buffer(indent, generated\_vars))  
 self.line\_number += 1  
 for index in range(rng(6, 7)):  
 code = r\_v(v\_funcs)(indent, generated\_vars)  
 file.write(code)  
 self.line\_number += 1  
 # Генерация произвольного кода  
 for index in range(rng(2, 5)):  
 file.write(r\_v(self.random\_code\_generators)(self, indent, generated\_vars))  
  
 # Конец функции  
 file.write("}\n")  
 self.line\_number += 1  
 return func\_name  
  
 def gen\_function(self, vuln\_generators, file):  
 indent = "\t"  
 func\_type = "int"  
 func\_name = "main"  
 generated\_vars = []  
 func\_params = []  
 for index in range(rng(1, 4)):  
 func\_params.append(r\_v(p\_funcs)(indent, generated\_vars))  
 file.write(func\_type + " " + func\_name + "(" + ", ".join([param[1:-2] for param in func\_params]) + ") {\n")  
 self.line\_number += 1  
 for index in range(rng(6, 7)):  
 code = r\_v(v\_funcs)(indent, generated\_vars)  
 file.write(code)  
 self.line\_number += 1  
 for generator in vuln\_generators:  
 for index in range(rng(2, 3)):  
 file.write(r\_v(self.random\_code\_generators)(self, indent, generated\_vars))  
 file.write(generator(self, indent, generated\_vars))  
 file.write("}\n")  
 self.line\_number += 1  
 return func\_name  
  
 def gen\_code(self, vulnerabilities: List[str], test\_numbers):  
 folder = "../generated\_tests"  
 for the\_file in os.listdir(folder):  
 file\_path = os.path.join(folder, the\_file)  
 try:  
 if os.path.isfile(file\_path):  
 os.unlink(file\_path)  
 except Exception as e:  
 print(e)  
 for index in range(test\_numbers):  
 self.line\_number = 1  
 file = open('../generated\_tests/' + "+".join(vulnerabilities) + str(index + 1) + '.cpp', 'w')  
 generators = []  
 for vuln in vulnerabilities:  
 generators.append(self.name\_generator[vuln])  
 if "Читатели Писатели" in vulnerabilities:  
 mutex\_cnt = rng(1, 3)  
 for i in range(mutex\_cnt):  
 mutex\_name = "testMutex" + str(r\_v(gen\_n\_rands(mutex\_cnt, 0, 10)))  
 file.write("mutex " + mutex\_name + ";\n")  
 self.line\_number += 1  
 self.mutexes.append(mutex\_name)  
 for i in range(rng(1, 3)):  
 self.cur\_available\_funcs.append(self.gen\_sample\_function(vulnerabilities, file))  
 self.gen\_function(generators, file)  
 self.cur\_available\_funcs = []  
 file.write(str(self.line\_number))  
 file.close()  
 # review\_file = open('../generated\_tests/' + "+".join(vulnerabilities)[0] + str(index + 1) + 'обзор.cpp', 'w',  
 # encoding="utf-8")  
 # review\_file.write("Сгенерированные и найденные ошибки:\n\n")  
 # for vulnerability in vulnerabilities:  
 # for key, value in self.final\_output:  
 # if vulnerability in key:  
 # review\_file.write("Сгенерированные:\n")  
 # review\_file.write(key + ": " + str(value) + "\n")  
 # review\_file.write("\nНайденные:\n")  
 # for vulnerability in vulnerabilities:  
 # to\_write = main.handlers\_list[vulnerability]().parse((find\_contexts(  
 # clean\_code('../generated\_tests/' + "+".join(vulnerabilities) + str(index + 1) + '.cpp'))))[  
 # -1] + "\n"  
 # review\_file.write(to\_write)  
 for vulnerability in vulnerabilities:  
 to\_write = main.handlers\_list[vulnerability]().parse((find\_contexts(  
 clean\_code('../generated\_tests/' + "+".join(vulnerabilities) + str(index + 1) + '.cpp'))))[  
 -1] + "\n"  
 m = re.search(r'(\d+)', to\_write)  
 tmp\_cnt = int(m.group(1))  
 for i in self.final\_output\_for\_test:  
 if vulnerability in i[0]:  
 i[1] += tmp\_cnt  
 stat\_file = open('общая\_статистика.cpp', 'w',  
 encoding="utf-8")  
 stat\_file.write("Статистика сгенерированных уязвимостей:\n")  
 for vulnerability in vulnerabilities:  
 cur\_vuln\_cnt = 0  
 for key, value in self.final\_output:  
 if vulnerability in key:  
 cur\_vuln\_cnt += value  
 stat\_file.write(vulnerability + ": " + str(cur\_vuln\_cnt) + "\n")  
  
 stat\_file.write("\nСтатистика найденных уязвимостей:\n")  
 for vulnerability in vulnerabilities:  
 for key, value in self.final\_output\_for\_test:  
 if vulnerability in key:  
 stat\_file.write(vulnerability + ": " + str(value) + "\n")  
  
 random\_code\_generators = [gen\_cond, gen\_try\_catch, buff\_error, c\_intr\_error, data\_leak, storage\_error,  
 file\_error, format\_error, iover\_error, free\_error, rng\_error, sql\_error]  
  
 catch\_generators = [gen\_cond, buff\_error, c\_intr\_error, data\_leak, storage\_error,  
 file\_error, format\_error, iover\_error, free\_error, rng\_error, sql\_error]  
  
 cond\_generators = [gen\_try\_catch, buff\_error, c\_intr\_error, data\_leak, storage\_error,  
 file\_error, format\_error, iover\_error, free\_error, rng\_error, sql\_error]  
  
 iover\_generators = [gen\_cond, gen\_try\_catch, buff\_error, c\_intr\_error, data\_leak, storage\_error,  
 file\_error, format\_error, free\_error, rng\_error, sql\_error]  
  
 name\_generator = {  
 "Переполнение буфера": buff\_error,  
 "Внедрение команд": c\_intr\_error,  
 "Утечка информации": data\_leak,  
 "Небезопасное хранение данных": storage\_error,  
 "Пренебрежение обработкой исключений": gen\_try\_catch,  
 "Некорректный доступ к файлам": file\_error,  
 "Ошибка форматной строки": format\_error,  
 "Переполнение целых чисел": iover\_error,  
 "Ошибка высвобождения памяти": free\_error,  
 "Состояние гонки": race\_error,  
 "Случайные числа криптографического характера": rng\_error,  
 "Читатели Писатели": readers\_error,  
 "SQL инъекции": sql\_error  
 }  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 gen = CodeGenerator()  
 gen.gen\_code(["Переполнение буфера", "SQL инъекции"], 3)

from random import randint as rng

# возвращает рандомный элемент указанного множества

def r\_v(params):

return params[rng(0, len(params) - 1)]

# генерирует n неповторяющихся значений от min до max

def gen\_n\_rands(n, min\_v, max\_v):

rands = []

if n == 0:

return rands

for index in range(n):

cur\_rng = rng(min\_v, max\_v)

while cur\_rng in rands:

cur\_rng = rng(min\_v, max\_v)

rands.append(cur\_rng)

return rands

from typing import List

from utils import constants

from code\_generator.rng\_utils import \*

max\_val = 40

def gen\_buffer(indent, generated\_vars: List[tuple]):

var\_type = r\_v(constants.types)

var\_name = "var\_" + str(rng(1, max\_val))

while var\_name in [gen\_vars[0] for gen\_vars in generated\_vars]:

var\_name = "var\_" + str(rng(1, max\_val))

code = indent + var\_type + " " + var\_name + "[" + str(rng(1, 50)) + "];\n"

generated\_vars.append((var\_name, code))

return code

def gen\_pointer(indent, generated\_vars: List[tuple]):

var\_type = r\_v(constants.types)

var\_name = "var\_" + str(rng(1, max\_val))

while var\_name in [gen\_vars[0] for gen\_vars in generated\_vars]:

var\_name = "var\_" + str(rng(1, max\_val))

code = indent + var\_type + " \*" + var\_name + ";\n"

generated\_vars.append((var\_name, code))

return code

def gen\_var(indent, generated\_vars: List[tuple]):

var\_type = r\_v(constants.types)

var\_name = "var\_" + str(rng(1, max\_val))

while var\_name in [gen\_vars[0] for gen\_vars in generated\_vars]:

var\_name = "var\_" + str(rng(1, max\_val))

code = indent + var\_type + " " + var\_name + ";\n"

generated\_vars.append((var\_name, code))

return code

def gen\_var\_with\_value(indent, generated\_vars: List[tuple]):

var\_type = r\_v(constants.types)

var\_name = "var\_" + str(rng(1, max\_val))

while var\_name in [gen\_vars[0] for gen\_vars in generated\_vars]:

var\_name = "var\_" + str(rng(1, max\_val))

code = indent + var\_type + " " + var\_name + " = " + str(rng(1, 30)) + ";\n"

generated\_vars.append((var\_name, code))

return code

def gen\_stream(indent, generated\_vars: List[tuple]):

var\_type = r\_v(constants.stream\_types)

var\_name = "var\_" + str(rng(1, max\_val))

while var\_name in [gen\_vars[0] for gen\_vars in generated\_vars]:

var\_name = "var\_" + str(rng(1, max\_val))

code = indent + var\_type + " " + var\_name + ".open(\"D://" + r\_v(constants.sample\_words) + "\");\n"

generated\_vars.append((var\_name, code))

return code

def gen\_integer\_with\_value(indent, generated\_vars: List[tuple]):

var\_type = r\_v(constants.int\_types)

var\_name = "var\_" + str(rng(1, max\_val))

while var\_name in [gen\_vars[0] for gen\_vars in generated\_vars]:

var\_name = "var\_" + str(rng(1, max\_val))

max\_over = constants.integer\_limits[var\_type].max

code = indent + var\_type + " " + var\_name + " = " + str(rng(int(max\_over/2), max\_over)) + ";\n"

generated\_vars.append((var\_name, code))

return code

if \_\_name\_\_ == '\_\_main\_\_':

for \_ in range(10):

print(gen\_integer\_with\_value("", []))