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, 2 ,6 ,9, 14 ,15 ,17

Результаты проектирования. Разработка

общего алгоритма. Разработка модулей ПО

|  |  |
| --- | --- |
| 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, 2, 6, 9, 14, 15, 17. Результаты проектирования. Разработка общего алгоритма. Разработка модулей ПО.***

|  |  |  |
| --- | --- | --- |
| 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 такжа 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 учереждении. 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:** 79 c., 6 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

[Общий отчет по проекту 8](#_Toc39595054)

[ВВЕДЕНИЕ 8](#_Toc39595055)

[1 МЕТОДЫ И БИБЛИОТЕКИ ИСПОЛЬЗУЕМЫЕ ПРИ РАЗРАБОТКЕ ПО ………………………………………………………………………………...10](#_Toc39595056)

[1.1 Анализ методов 10](#_Toc39595057)

[1.2 Существующие библиотеки и программные средства 10](#_Toc39595058)

[1.3 Разработка алгоритма 13](#_Toc39595059)

[1.4 Выводы и результаты 13](#_Toc39595060)

[2 ОПИСАНИЕ РАЗРАБОТАННОГО ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ 15](#_Toc39595061)

[2.1 Выбор средств разработки и системных программных средств 15](#_Toc39595062)

[2.2 Требования к разрабатываемой программе 15](#_Toc39595063)

[2.3 Структура программы 17](#_Toc39595064)

[2.3.1 Основные модули 18](#_Toc39595065)

[2.4 Описание интерфейса пользователя 30](#_Toc39595066)

[2.5 Тестирование 34](#_Toc39595067)

[2.6 Выводы и результаты 36](#_Toc39595068)

[ЗАКЛЮЧЕНИЕ 37](#_Toc39595069)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 38](#_Toc39595070)

[ИНДИВИДУАЛЬНЫЙ ОТЧЕТ ПО ПРОЕКТУ 39](#_Toc39595071)

[ПРИЛОЖЕНИЕ А 41](#_Toc39595072)

[ПРИЛОЖЕНИЕ Б 44](#_Toc39595073)

Общий отчет по проекту

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. разработка проверочных тестов;
4. программирование разработанных алгоритмов;
5. автономная отладка программ;
6. комплексная отладка программ;

# МЕТОДЫ И БИБЛИОТЕКИ ИСПОЛЬЗУЕМЫЕ ПРИ РАЗРАБОТКЕ ПО

## Анализ методов

Принцип работы статических анализаторов основывается на разборе предоставленного к анализу кода, без его компиляции или интерпретации. Существует два основных подхода к реализации статического анализатора. Первый – разбиение кода анализируемой программы на отдельные части - лексемы и использование их при анализе уязвимостей. Второй – использование механизма регулярных выражений, для поиска уязвимостей по заданным шаблонам. Третий – комбинация двух предыдущих. При разработке собственного анализатора мы остановились на третьем варианте.

## Существующие библиотеки и программные средства

Разработка нашего курсового проекта начиналась с обзора существующих решений и программных средств в области статического анализа. Мы выделили три статических анализатора кода, изучили их функциональные возможности, а также выделили недостатки, из-за которых мы не стали использовать их в качестве основы для создания собственного 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.

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. |

Основой нашего приложения стал язык Python. Это решение было принято исходя из двух преимуществ этого языка:

1. Наличие гибкой библиотеки для создания графического интерфейса приложения;
2. Расположенность языка к работе с регулярными выражениями, а также наличие функциональной библиотеки для решения нашей задачи.

Ключевую роль в разработке анализатора сыграла библиотека - Lib/re.py. Она предназначена для с работы с текстом и с регулярными выражениями.

## Разработка алгоритма

Работа нашего анализатора построена на разбиение исследуемого исходного кода на отдельные части – лексемы и их дальнейший анализ на наличие уязвимостей по построенным шаблонам. Разбиение кода является основополагающей частью статического анализа, поэтому алгоритму выделения отдельных частей кода было выделено большая часть времени разработки. Ниже приведен алгоритм разбиения исходного кода на составные части, а именно функции.

1. Поиск начала объявления функции;
2. Поиск окончания объявления функции;
3. Анализ тела функции.
4. Анализ возвращаемого значения функции;
5. Анализ параметров функции;
6. Анализ объявленных переменных внутри функции;
7. Анализ объявленных потоков внутри функции;
8. Построение структуры данных, полностью описывающей состояние и наполнение функции.

## Выводы и результаты

Был произведен анализ предметной области, рассмотрены существующие решения и программные средства статического анализа. В результате был выбран подход анализа, сочетающий в себе использование разбиения исходного кода на отдельные части – лексемы, а также использование регулярных выражений.

# ОПИСАНИЕ РАЗРАБОТАННОГО ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

## Выбор средств разработки и системных программных средств

Основой нашего приложения стал язык Python из-за наличия гибкой библиотеки для создания графического интерфейса приложения и расположенность языка к работе с регулярными выражениями.

В качестве среды разработки выступила PyCharm. Такой выбор обусловлен направленностью этой среды разработки на язык Python, функциональность и удобный пользовательский интерфейс.

## Требования к разрабатываемой программе

- программа должна иметь дружественны интерфейс

- интерфейс программы должен содержать:

1. Список загруженных на проверку программ;

2.Список загруженных поисковых программ;

3. Список загруженных тест примеров;

4. Указатель на исследуемые уязвимости с изменением его состояния по мере проверки уязвимостей;

5. Кнопку «произвести проверку» программы №… или последовательно все;

6. Кнопку произвести проверку на тест- примерах №… или на всех.

7. 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

- должен быть реализован поиск следующих уязвимостей:

1. Переполнение буфера;

2. Пренебрежение обработок ошибок;

3. Проблемы, связанные с динамическим выделением памяти;

4. Внедрение команд;

5. Внедрение SQL команд;

6. Состояние гонки;

7. Ошибка форматной строки;

8. Некорректный доступ к файлам;

9. Читатели – Писатели;

10. Случайные числа криптографического характера;

11. Переполнение целых чисел;

12. Пренебрежение безопасным хранением данных;

13. Утечка информации.

- программа должна быть проверена набором ручных тестов, а также тестами, созданными автогенератором тестов.

## Структура программы

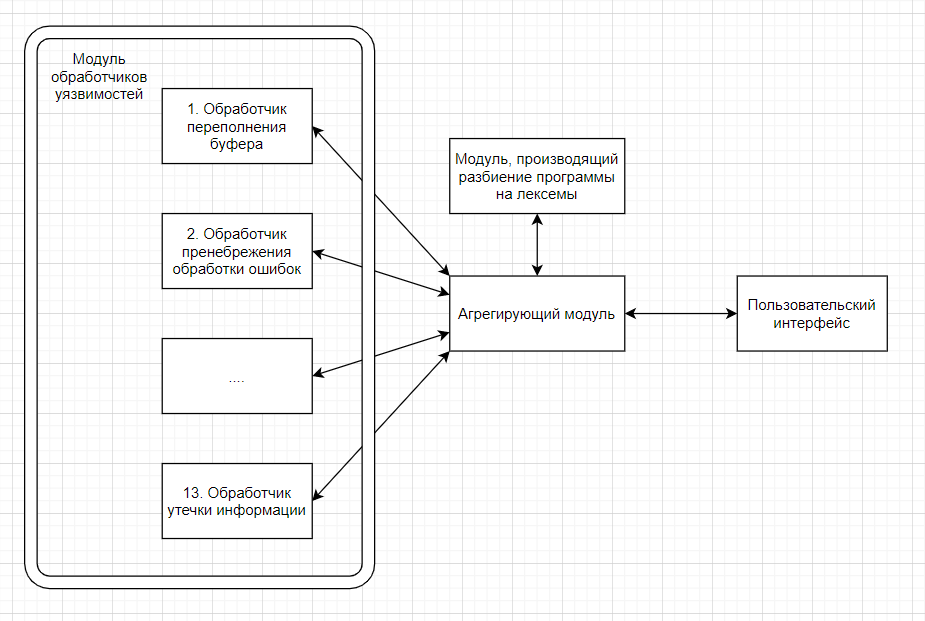


Рисунок 1 – Структурная схема ПО

Программа состоит из четырех основных модулей:

1. Модуль обработчиков уязвимостей;
2. Модуль, отвечающий за разбиение исходного кода на лексемы;
3. Пользовательский интерфейс;
4. Агрегирующий модуль;

*Модуль обработчиков уязвимостей* содержит в себе все классы, отвечающие за обработку конкретных уязвимостей.

*Модуль, отвечающий за разбиение исходного кода на лексемы*, проводит разбиение анализируемых программ на составные части, для дальнейшего анализа.

*Модуль пользовательского интерфейса* отвечает за графическое отображение программы, предоставляет дружественный способ взаимодействия с пользователем.

*Агрегирующий модуль* отвечает за совместную организацию всех остальных модулей программы.

### Основные модули

**Модуль обработчиков уязвимостей**

Данный модуль состоит из 13 классов, отвечающих за поиск и обработку отдельной уязвимости. В анализаторе рассматриваются следующие уязвимости.

**1. Переполнение 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.

Пример уязвимого кода:

int main**(**int argc**,** char **\***argv**[])** **{**

char buf**[**256**];**

**...**

**...**

strcpy**(**buf**,** argv**[**1**]);**

**}**

Логическая структура уязвимости:

<объявление переменной, расцениваемой как возможный буфер>

<использование объявленных буферов в небезопасных относительно

переполнения функциях>

Между строками шаблона возможно наличие постороннего кода. Ведется поиск использования ранее объявленных буферов в небезопасных, относительно переполнения, функциях.

**2. 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.

Пример уязвимого кода:

public class WebServlet**{**

protected void Page\_Load**(**HttpRequest req**) {**

**try** **{**

string input **=** req**.**Form**[**"in"**];**

Process p **=** Process**.**Start**(**input**);**

**}** **catch{}**

**}**

**}**

В данном примере отсутствует обработчик исключения, что может привести к большой угрозе безопасности в случае возникновения ошибки.

Логическая структура уязвимости:

<объявление блока «try»>

<пустой «catch», связанный с вышеупомянутым «try»>

Между строками шаблона возможно наличие постороннего кода. Ведется поиск конструкций «try catch» с пустым блоком «catch».

**3. Проблемы с динамическим выделением памяти.**

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

Пример уязвимого кода:

int main**(){**

int **\***a**;**

int i**,** n**;**

scanf**(**"%d"**,** **&**n**);**

a **=** **new int;**

**...**

**...**

exit**();**

**}**

Логическая структура уязвимости:

<выделение памяти>

<отсутствие освобождения памяти>

Между строками шаблона возможно наличие постороннего кода. Сначала ведется поиск переменных, которые указывают на выделенную память, затем ведется поиск операторов освобождения этой памяти.

**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.

Пример уязвимого кода:

SqlConn conn **=** **new** SqlConn**(**connection**);**

SqlCommand command **=** **new** SqlCommand**();**

command**.**Connection **=** connection**;**

connection**.**Open**();**

Object queryRes **=** command**.**Execute**(**"SELECT AccountNumber FROM Users " **+**

"WHERE Username='" **+** name **+**

"' AND Password='" **+** password **+** "'"**);**

// SELECT AccountNumber FROM Users WHERE Username='' OR 1=1

В данном фрагменте кода запрос к базе данных выполняется не в параметризованном виде или без какой-либо проверки на входные данные. Подставив вместо параметра name строковое значение «' OR 1=1 --», злоумышленник сможет получить данные всех пользователей, обойдя алгоритм защиты.

Логическая структура уязвимости:

<использование в функции запроса неэкранированной строки>

Между строками шаблона возможно наличие постороннего кода. Ведется поиск кода, где выполняются запросы к базе данных без экранирования символа <’>.

**5. 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.

Пример уязвимого кода:

public class WebForm **{**

void Page\_Load**(**object sender**,** EventArgs e**){**

string input **=** Request**.**Form**[**"in"**];**

Process p **=** Process**.**Start**(**input**);**

**}**

**}**

В данном примере уязвимый код запрашивает выполнение процесса на удаленном сервере, не проверяя передаваемые параметры.

Логическая структура уязвимости:

<использование системной функции без проверки входных параметров>

Между строками шаблона возможно наличие постороннего кода. Ведется поиск использования системных функций, в которые можно передать небезопасные параметры.

**6. 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. Состояние гонки можно обнаружить, проверяя обращение потенциальных потоков к одним и тем же переменным или при использовании потоками одинаковых объявленных функций.

Пример уязвимого кода:

int param1**;**

int param3**;**

func1**{**

**while** **(!**stop**){**

x**++;**

**}**

**}**

func2**{**

**while** **(!**stop**){**

**if(**x**%**2 **==** 0**){...}**

**}**

**}**

std::thread t1(func1, param1)

std::thread t2(func2, param1)

std::thread t3(func1, param3)

...

Пусть изначально x=0. При случайном определении приоритета доступа к переменной может быть получена такая ситуация.

Оператор if в потоке 2 проверяет *x* на чётность. Оператор «*x++*» в потоке 1 увеличивает *x* на единицу. В потоке 2 выполняет код, находящийся под условием четности в то время как переменная уже нечетная.

Логическая структура уязвимости:

<объявление переменных, используемых потоками>

<объявление функций, используемых потоками>

<объявление потоков>

Между строками шаблона возможно наличие постороннего кода. Допускается неограниченное количество потоков. Сначала ведется поиск всех объявленных потоков, переменных и функций, используемых потоками. Затем, определяется использование разными потоками одной и той же функции.

**7. 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.

Пример уязвимого кода:

int main() {

char input[100];

scanf(“%s”, &input);

printf(input);} //отсутствие формата вывода

Логическая структура уязвимости:

<использование функции с форматной строкой>

Между строками шаблона возможно наличие постороннего кода. Ведется поиск функций, в которых используется форматная строка.

**8. Некорректный доступ к файлам.**

Можно выделить типичные проблемы безопасности, связанные с файловыми операциями.

Первый тип уязвимости – является ли файл файлом вообще и стоит ли с ним работать.

Второй вид уязвимости состоит в вопросе существования файла, с которым будет вестись работа.

Пример уязвимого кода:

Const char **\***filename **=**”**/**this\_is\_not\_a\_file”**;**

File **\***f\_pointer**=**open**(**filename**,** "rw"**)**

f\_pointer**.**read**(**buf**[])**

В данном примере, происходит работа с файловым указателем, без проверки того, что файл вообще существует.

Логическая структура уязвимости:

<объявление файловых переменных>

<обращение к файловым переменным>

Между строками шаблона возможно наличие постороннего кода. Сначала ведется поиск объявлений файловых переменных, затем поиск функций, в которых используются файловые переменные.

**9. 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.

Данную уязвимость можно обнаружить, отслеживая используемые потоками параметры и отслеживая наличие блоков синхронизации в исполняемых потоками функциях.

Пример уязвимого кода:

res param1**;**

res param2**;**

mutex mutex\_1;

func1**{**

param1.read();

**}**

func2**{**

param1.write();

**}**

std::thread t1(func1, param1)

std::thread t2(func1, param1)

std::thread t3(func2, param1)

...

В данном примере три потока используют одну и ту же переменную param1, два из них производят чтение, а один запись. При этом в примере отсутствуют блоки синхронизации на основе мьютексов.

Логическая структура уязвимости:

<объявление переменных, используемых потоками>

<объявление функций, используемых потоками>

<объявление потоков>

Между строками шаблона возможно наличие постороннего кода. Допускается неограниченное количество потоков. Сначала ведется поиск всех объявленных потоков, переменных и функций, используемых потоками. Затем, определяется использование разными потоками одних и тех же переменных. Затем происходит проверка наличия блоков синхронизации (мьютексов) в исполняемых потоками функциях.

**10. Случайные числа криптографического качества.**

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.

Пример уязвимого кода:

mt19937 rnd**;**

rnd**.**seed**(**500**);**

int value **=** gen**(**rnd**);**

Логическая структура уязвимости:

<объявление переменных, используемых генераторами случайных чисел>

<объявление функций генерации случайных чисел>

Между строками шаблона возможно наличие постороннего кода. Ведется поиск не криптостойких функций генерации случайных чисел, а также поиск функций, которые используют в качестве seed (значения необходимого для генерации) неизменяемые значения.

**11. 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.

Пример уязвимого кода:

short a**;**

int b **=** 2 147 483 647**;**

int c **=** 2 147 483 647**;**

someFunction**(**b**+**c**);**

Логическая структура уязвимости:

<объявление переменных, используемых в арифметических операциях>

<объявление операций>

Между строками шаблона возможно наличие постороннего кода. Производится поиск переменных участвующих в выражениях. Затем проводится проверка возможности переполнения объявленных переменных.

**12. Пренебрежение безопасным хранением данных.**

Уязвимость возникает при встраивании секретного кода в текст программы, некорректное использование элементов управления доступом.

Пример уязвимого кода:

int test(char \*params){  
    int result = SetKernelObjectSecurity(params);  
    if (result ==0) return True  
    return False;  
}

Логическая структура уязвимости:

<объявление функций, которые могут изменять права доступа>

Между строками шаблона возможно наличие постороннего кода.

Происходит поиск уязвимых функций, которые позволяют задать уровень привилегий для определенных объектов ОС.

**13. 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.

Пример уязвимого кода:

**try{...}**

**catch** **(**Exception se**){**

foreach **(**SecurityErr e in se**.**Errors**){**

printf**(**e**.**getMessage**);**

GetEnvironmentVariable(str);

**}**

**}**

Логическая структура уязвимости:

<использование функций, приводящих к утечке информации>

Между строками шаблона возможно наличие постороннего кода. Ведется поиск функций, в которых может происходить утечка информации.

За каждую уязвимость отвечает отдельный обработчик согласно таблице.

Таблица 2 – Уязвимость – Метод обнаружения – ПО обнаружения

|  |  |  |
| --- | --- | --- |
| Уязвимость | Метод обнаружения | ПО обнаружения |
| Переполнение буфера | Поиск объявленных буферов памяти ифункий, связанных с их использованием | BufferOverflowHandler |
| Ошибка форматной строки | Поиск функции printf() | FormatStringHandler |
| Внедрение SQL-кода | Поиск уязвимых функций вызова SQL запросов с телом запроса в котором отсутствует экранирование символа <’> | SQLInjectionHandler |
| Внедрение команд | Поиск уязвимых функции, которые позволяют выполнять системные команды | CommandsIntoductionHandler |
| Пренебрежение безопасным хранением данных | Поиск уязвимых функций, которые позволяют задать уровень привилегий для определенных объектов  ОС | DataStorageManagmentHandler |
| Утечка информации | Поиск уязвимых функций, которые могут привести к потере\утечке данных | DataLeakHandler |
| Некорректный доступ к файлам | Поиск уязвимых функций, позволяющие проводить операций над файлами (создание ссылок, файлов, изменение файлов и т.д. | IncorrectFileAccessHandler |
| Случайные числа криптографического характера. | Поиск функций генерации случайных чисел, инициализированных константным значением, а также других не криптостойких генераторов из стандартной библиотеки. | RandomGeneratorHandler |
| Переполнение целых чисел | Поиск объявлений целочисленных переменных и арифметических выражений,приводящих к их переполнению | IntegerOverflowHandler |
| Состояние гонки | Поиск потоков, у которых используется одна и таже исполняемая функция. | RaceConditionHandler |
| Читатели Писатели | Поиск потоков, использующих одинаковые переменные в качестве параметров исполняемых функций, а также анализ этих исполняемых функций на наличие блоков синхронизации. | ReadersWritersHandler |
| Проблемы с динамическим выделением памяти | Поиск переменных, для которых была выделена, но не освобождена память. | DataLeakHandler |

**Агрегирующий модуль**

Модуль отвечает за агрегацию всех остальных модулей и их совместное использования для анализа исходного кода программы.

**Модуль, производящий разбиение программы на лексемы**

Модуль отвечает за разбиение исходного кода анализируемой программы на лексемы или составные части, которые впоследствии используются обработчиками уязвиостей для анализа исходного кода.

**Пользовательский интерфейс**

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

## Описание интерфейса пользователя

**1. Главное окно.**

Главное окно приложения (рис. 2) можно поделить на три основополагающие части:

1) Блок доступных к анализу уязвимостей, расположенный в левой части главного окна (рис. 2).

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 таблицы ответственности по написанию тестов. Блок расположен в центре главного окна (рис. 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.

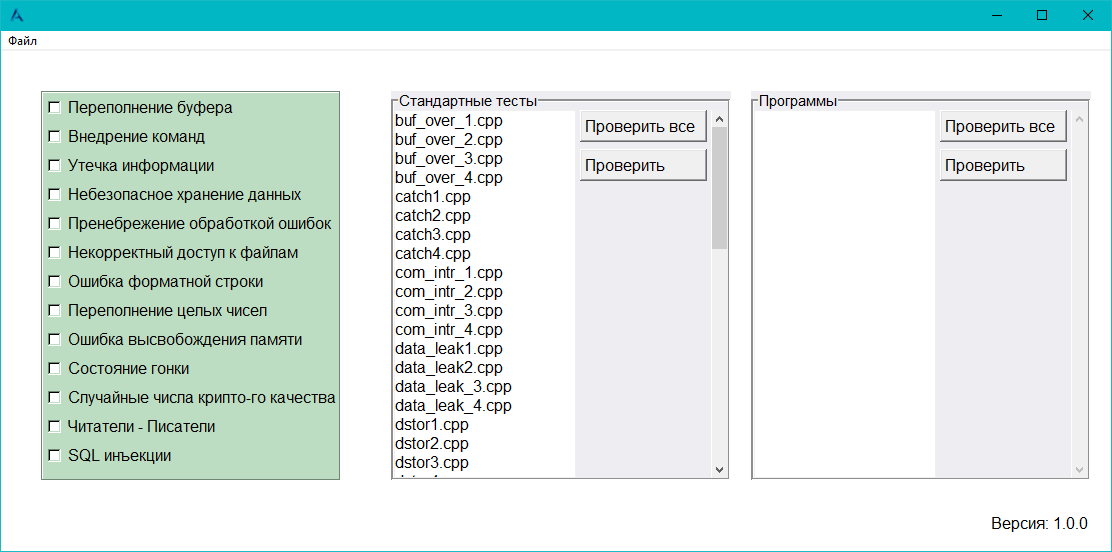
****3) Блок, в котором располагается список программ, которые пользователь самостоятельно загружает в анализатор. Расположен в правой части главного окна (рис. 2). Функциональность кнопок «Проверить все» и «Проверить» аналогична центральному блоку.

Рисунок 2 – Главное окно

Загрузка собственных программ в анализатор происходит путем нажатия на кнопку «Файл», расположенного в левом верхнем углу главного окна (рис. 3), и далее на кнопку «Открыть». После этого появится окно выбора программ (рис. 4).

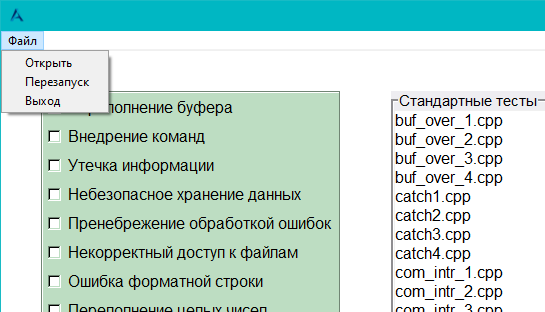


Рисунок 3 – Кнопка «Файл»

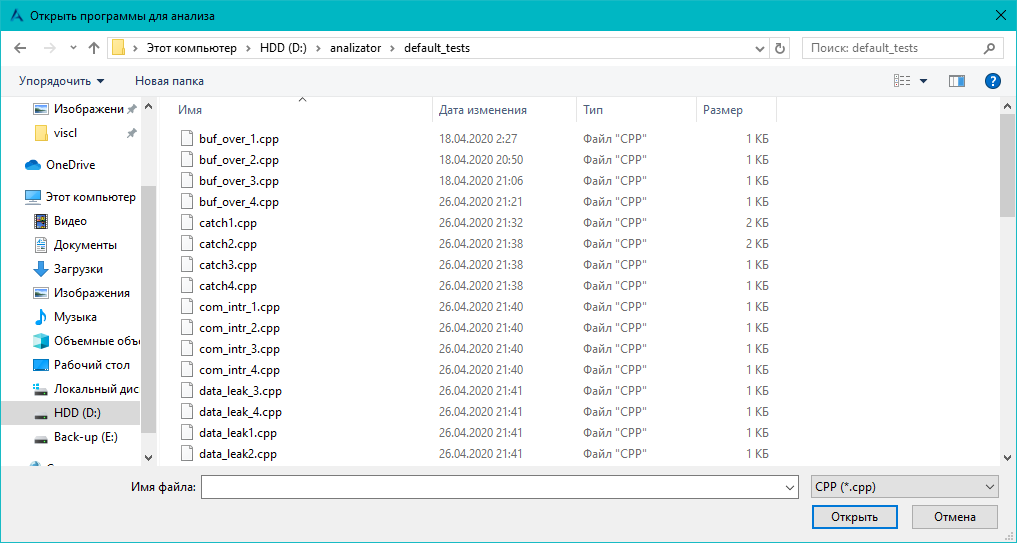


Рисунок 4 – открытие файлов для анализа

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

**2. Окно анализа программы.**

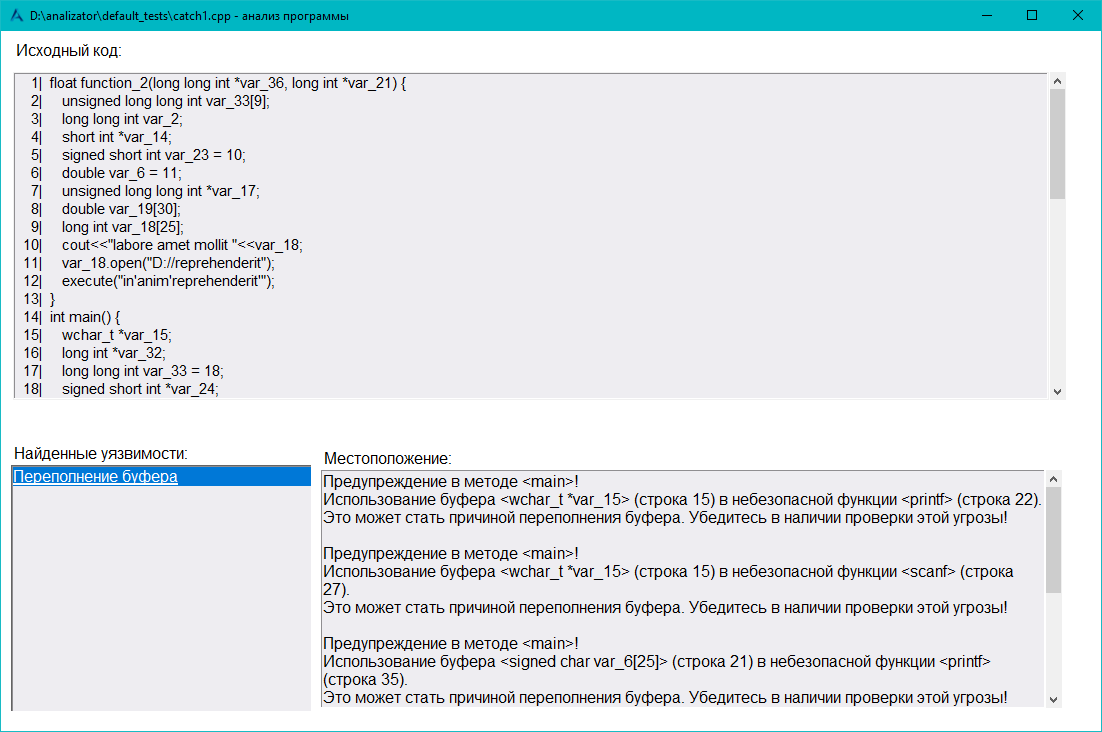
При нажатии на одну из кнопок «Проверить все» или «Проверить» откроется окно анализа выбранной программы или несколько окон в случае выбора нескольких программ, по одному окну на каждую программу (рис. 5).

Рисунок 5 – окно анализа программы.

В верхней части данного окна 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 (рис. 6).

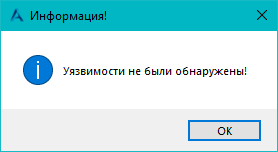


Рисунок 6 – информационное окно

## 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 оработчика 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 Тереньтьева, Терентьев для Ковалева, Ковалев для Шабакаева.

1. Автоматический генератор тестового кода C++.

Реализация данной программы выполналсь в рамках отдельной части курсового проекта. Генератор представляет собой консольное приложение, способное генерировать случайный код C++, в котором буду сгенерированы указанные уязвимости или их комбинации. Пример автоматически сгенерированного кода указан в приложении Б. В тестовом генераторе можно указать количество генерируемых тестов, а также уязвимости, которые должны быть сгенерированы. В результате работы генератора, выдается текстовый отчет, в котором указано количество сгенерированных ошибок и количество найденных анализатором ошибок

## Выводы и результаты

В результате был разработан статический анализатор кода, способный обнаруживать все 13 заявленных уязвимостей. Работоспособность программы была проверена на наборе тестов, по 4 теста на каждую уязвимость, а также при помощи реализованного автоматического генератора кода.

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 проектирвоания.

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 сатического анализатора, мы проводили отладку, устраняли ошибки и недоработки.

В результате, мы изучули все согласованные на этапе проектирования уязвимости информационной безпоасности, которые могут привести к непоправимым ошибкам безопасности при разработке приложения на языке С++, а также методы их определения, реализовали статический анализтор 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.

индивидуальный отчет по проекту

ПРИЛОЖЕНИЕ А

Статический анализатор кода С++ (рис. 1) состоит из 4 основных модулей, в состав которых входит 16 частей.

1) Модуль обработчиков уязвимостей:

Часть 1 - Обработчик переполнения буфера;

Часть 2 - Обработчик пренебрежения обработок ошибок;

Часть 3 - Обработчик проблем, связанных с динамическим выделением памяти;

Часть 4 - Обработчик внедрения SQL команд;

Часть 5 - Обработчик внедрения команд;

Часть 6 - Обработчик состояния гонки;

Часть 7 - Обработчик ошибки форматной строки;

Часть 8 - Обработчик некорректного доступа к файлам;

Часть 9 - Обработчик задачи Читатели – Писатели;

Часть 10 - Обработчик случайных чисел криптографического характера;

Часть 11 - Обработчик переполнения целых чисел;

Часть 12 - Обработчик пренебрежения безопасным хранением данных;

Часть 13 - Обработчик утечки информации.

2) Агрегирующий модуль – часть 14.

3) Модуль, производящий разбиение программы на лексемы – часть 15.

4) Пользовательский интерфейс – часть 16.

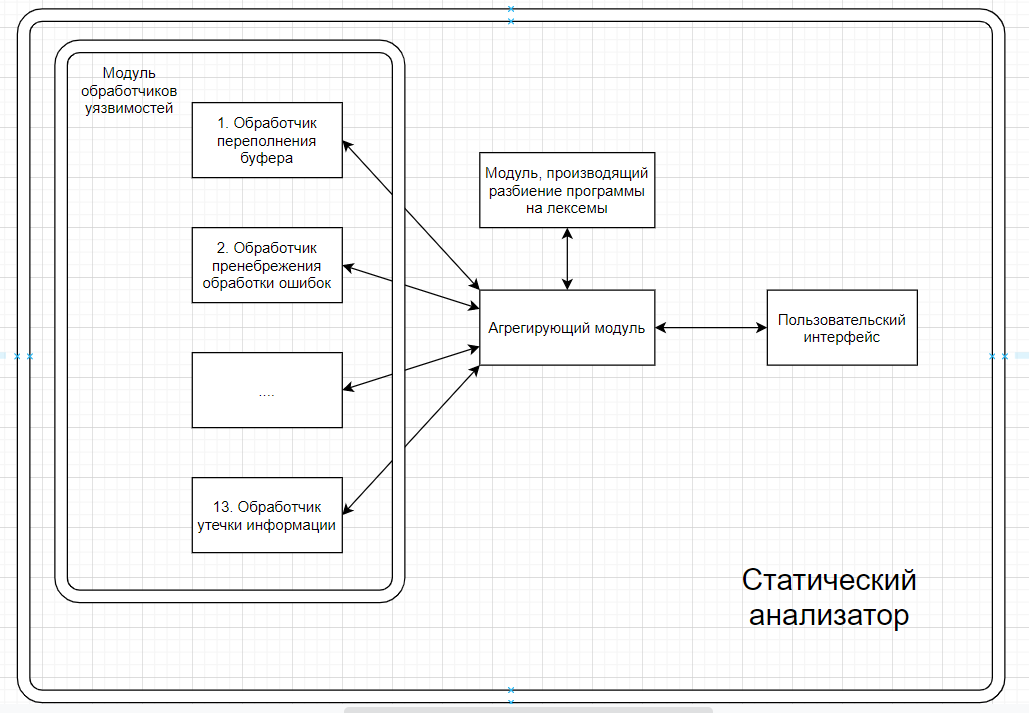


Рисунок 1 - Структурная схема статического анализатора

Также был разработан генератор тестового кода С++ для тестирования анализатора.

Генератор тестового кода С++ - часть 17

Таблица 1 - распределения частей проекта среди участников бригады.

|  |  |
| --- | --- |
| Ковалев К.А. | Часть 1 - Обработчик переполнения буфера;  Часть 2 - Обработчик пренебрежения обработок ошибок;  Часть 9 - Обработчик задачи Читатели – Писатели;  Часть 6 - Обработчик состояния гонки;  Часть 15 - Модуль, производящий разбиение программы на лексемы;  Часть 14 – Агрегирующий модуль;  Часть 17 – Генератор тестового кода С++. |
| Терентьев Е.А. | Часть 4 - Обработчик внедрения SQL команд;  Часть 5 - Обработчик внедрения команд.  Часть 3 - Обработчик проблем, связанных с динамическим выделением памяти; |
| Щекочихин Д.А. | Часть 7 - Обработчик ошибки форматной строки;  Часть 8 - Обработчик некорректного доступа к файлам;  Часть 16 – Пользовательский интерфейс. |
| Оганисян Т.С. | Часть 10 - Обработчик случайных чисел криптографического характера;  Часть 11 - Обработчик переполнения целых чисел. |
| Шабакаев А.Р. | Часть 12 - Обработчик пренебрежения безопасным хранением данных;  Часть 13 - Обработчик утечки информации. |

ПРИЛОЖЕНИЕ Б

Исходный код программы:

Часть 1 - 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

Часть 2 - 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

Часть 3 - 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

Часть 4 - 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

Часть 5 - 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

Часть 6 - 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

Часть 7 – 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

Часть 8 - 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

Часть 9 - 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

Часть 10 - 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

Часть 11 - 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

Часть 12 - 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

Часть 13 - 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

Часть 14 – 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

Часть 15 - Модуль, производящий разбиение программы на лексемы.

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

Часть 16 – 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()

Часть 17 – 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("", []))

Пример теста, созданного автогенератором тестового кода С++.

signed char function\_18(byte \*var\_11, char \*var\_24) {  
 float var\_25[23];  
 byte var\_15 = 30;  
 long int var\_29;  
 byte var\_32[6];  
 unsigned int var\_6 = 15;  
 long int var\_31[33];  
 long long int var\_26 = 25;  
 SetServiceObjectSecurity(var\_26, var\_11, var\_6);  
 if(var\_11 != 321) {  
 executeQuery("exercitation'irure'ut'");  
 sprintf(var\_31, var\_29, var\_24);  
 }  
}  
unsigned int function\_13(signed char var\_20) {  
 signed int var\_9[9];  
 unsigned char var\_32;  
 signed int var\_21;  
 unsigned short int var\_22;  
 unsigned char var\_39;  
 unsigned char \*var\_25;  
 signed int var\_14;  
 signed long long int var\_4;  
 strncpy(var\_39, var\_14, var\_25);  
}  
int main(signed int var\_28, long double \*var\_5) {  
 unsigned short int \*var\_13;  
 unsigned long int var\_2 = 16;  
 char var\_36 = 19;  
 int var\_31[27];  
 wchar\_t \*var\_15;  
 int var\_9;  
 long long int var\_29;  
 strcpy(var\_13, var\_5);  
 sprintf(var\_15, var\_28, var\_31);  
 sprintf(var\_15, var\_36, var\_5);  
 execvp(var\_13, var\_2);  
 popen(var\_2, var\_9);  
 perror(var\_9);  
 getenv(var\_2);  
 perror(var\_29);  
 fchown(var\_15, var\_36, var\_31);  
 fcntl(var\_15, var\_13, var\_5);  
 try {  
 printf(Format %s and %s, var\_31, var\_13);  
 gets(var\_5);  
 fcntl(var\_31, var\_36, var\_29);  
 }  
 catch (Exception\_17 err) {  
 chmod(var\_2, var\_31);  
 SetServiceObjectSecurity(var\_36, var\_15, var\_29);  
 }  
 try {  
 signed char \*var\_7 = new signed char;  
 if(var\_2 >= 424) {  
 strcpy(var\_5, var\_29);  
 char \*var\_35 = new char;  
 }  
 mktemp(var\_29);  
 }  
 catch (Exception\_13 err) {  
 printf(Format %s and %s, var\_28, var\_7);  
 execute(var\_15);  
 }  
 try {  
 int var\_24 = 1326806408;  
 long var\_12 = 8202466402154759295;  
 long var\_33 = 5791971425614649244;  
 if(var\_12 <= 171) {  
 utime(var\_28, var\_24);  
 executeQuery(var\_33);  
 }  
 var\_24 = var\_12 + var\_33;  
 if(var\_29 != 309) {  
 wchar\_t \*var\_33 = new wchar\_t;  
 execute(var\_24);  
 }  
 executeQuery("adipiscing'aute'laborum.'");  
 }  
 catch (Exception\_6 err) {  
  
 }  
}  
84

Отчет о поиске уязвимостей в этом тесте.

Статистика сгенерированных уязвимостей:  
Переполнение буфера: 14  
Внедрение команд: 2  
Утечка информации: 3  
Небезопасное хранение данных: 6  
Пренебрежение обработкой исключений: 1  
  
Статистика найденных уязвимостей:  
Переполнение буфера: 14  
Внедрение команд: 2  
Утечка информации: 3  
Небезопасное хранение данных: 6  
Пренебрежение обработкой исключений: 1