РАЗРАБОТКА СТАТИЧЕСКОГО АНАЛИЗАТОРА ИСХОДНЫХ ТЕКСТОВ ПРОГРАММНОГО КОДА ДЛЯ ВЫЯВЛЕНИЯ ДЕФЕКТОВ БЕЗОПАСНОСТИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

А. В. Балуев, А. А. Кузнецов, П. А. Храпунов, Л. Ю. Застылова, Е. С. Алексеева, О. А. Сергеева, М. В. Шестова

ФГУП «РФЯЦ-ВНИИЭФ», г. Саров Нижегородской обл.

Введение

С целью обеспечения информационной безопасности (ИБ)программного обеспечения (ПО) необходимо, чтобы ПО, используемое на предприятии, имело сертификат соответствия.

Проведение анализа уязвимостей ПО в настоящее время является обязательной деятельностью экспертов испытательных лабораторий при проведении сертификационных испытаний, выполняемых по линии системы сертификации Федеральной службы по техническому и экспертному контролю (ФСТЭК) России. Данный вид работ выполняется как при сертификации на соответствие требованиям утвержденных ФСТЭК России профилей защиты, в которых в явном виде включены требования семейства AVA_VAN «Анализ уязвимостей», так и при испытаниях на соответствие требованиям технических условий.

Неотъемлемой частью сертификации ПО,на отсутствие недекларируемых возможностей (НДВ), является статический анализ.Его результаты (в частности, результаты сигнатурного анализа) используются при проведении вышеуказанного анализа уязвимостей. Для оптимизации статического анализа в процессе сертификации применяют средства автоматизации. При текущем уровне сложности ПО, проведение сертификации на уровни контроля отсутствия НДВ 3 и выше (гостайна), без использования средств автоматизации, зачастую невозможно. Это обусловлено большими объемами исходных кодов, широким спектром используемых языков, средств разработки и т. д.

Теоретическая часть

Сертификация – процесс проверки соответствия средств защиты информации (далее – СЗИ), входящих в состав проверяемого продукта, предъявляемым требованиям. Сертификация может проводиться на соответствие руководящим документам (РД) – «Средства вычислительной техники. Защита от несанкционированного доступа к информации. Показатели защищенности от несанкционированного доступа к информации» [1], техническим условиям (ТУ) и общим критериям (ОК).

Методы статического анализа

В информационной безопасности выделяют следующие категории методов проведения статического анализа уязвимостей программного кода [3]:

- анализ структуры и декомпозиция программы;
- проверка свойств (ограничений) моделей представления программного кода;
- сигнатурный анализ.

2.1. Анализ структуры и декомпозиция программы

Данная категория методов ориентирована на выявление нарушений полноты и избыточности и представляет собой процесс выделения части программного продукта, необходимого и достаточного для нормального функционирования программы.

В зависимости от требований, предъявляемых к продукту в процессе сертификации, контроль полноты и избыточности исходных текстов программы может осуществляться на уровне файлов или на уровне функциональных объектов [2].

Проверка свойств (ограничений) моделей представления программного кода

Анализ свойств моделей представления программного кода, получаемых в процессе компиляции, позволяет выявить разного рода некорректности кодирования, например: ошибки указателей, типов данных, неиспользуемые или не инициализированные переменные, утечки памяти и др.

Наиболее известными алгоритмами выявления некорректности кодирования являются:

– интервальный анализ;
– анализ указателей;
– анализ зависимостей по данным;
– контроль типов данных и др.

Сигнатурный анализ

Сигнатурный анализ представляет собой поиск программных дефектов в программном коде путем сопоставления фрагментов программного кода с образцами из базы данных наиболее часто встречающихся дефектов безопасности.

В сети интернет в открытом доступе имеется ряд баз данных, содержащих наборы шаблонов небезопасных сигнатур. Среди них можно выделить следующие:

- MITRECWE – общая классификация дефектов ПО.
- Fortify Seven Pernicious Kingdoms – 7 разрушительных «царств» компании HP Fortify.
- MITRECVE – общие уязвимости и «незащищенности».
- OSVDB – база уязвимостей открытого доступа.
- US-CERT Vulnerability Notes Database – база уязвимостей.
- БДУ ФСТЭК – банк данных угроз безопасности информации.

К недостаткам сигнатурного подхода относят большое число ложных срабатываний, поэтому фактически он направлен на снижение трудоемкости работы эксперта, предоставляя для его внимания наиболее критичные фрагменты кода.

Практическая часть

В соответствии с руководящим документом ФСТЭК России «Защита от несанкционированного доступа к информации Часть 1. Программное обеспечение средств защиты информации. Классификация по уровню контроля отсутствия недекларированных возможностей» [2], в зависимости от уровня контроля отсутствия НДВ, на который сертифицируется исследуемый программный продукт, к содержанию испытаний, проводимых в ходе статического анализа, могут предъявляться различные требования. В частности, одним из неотъемлемых требований выступает контроль наличия заданных конструкций в исходных текстах программ или сигнатурный анализ.

В ходе выполнения сигнатурного анализа выполняется сопоставление фрагментов программного кода с образцами из базы данных потенциально опасных конструкций. Каждый элемент базы проверяется на присутствие в исследуемом программном коде. В случае положительного результата, потенциально опасный участок кода должен быть более подробно исследован экспертом на предмет наличия уязвимости.

Структура и функции

Статический анализатор кода реализует следующие основные функции:

- поиск ошибок программного кода;
- формирование перечней функциональных и информационных объектов;
- представление результатов анализа в табличном виде.

Функциональная схема анализатора представлена на рис. 1.

Для реализации заявленных функций разработана модульная структура анализатора. Данный подход обеспечивает гибкость разрабатываемого продукта и не отягощает доработку и расширение функционала. Подробнее данный подход описан в «Большой энциклопедии нефти и газа» [4].

Архитектура

Для обеспечения многопользовательского режима функционирования, а также распределения вычислительных ресурсов, программная утилита была реализована в рамках клиент-серверной архитектуры. Основной функционал, включая аудит программного кода и логирование, реализуется на серверной части. Хранение баз данных выполняемых проектов, а также логов программы также обеспечивается ресурсами сервера.

При этом взаимодействие с пользователем обеспечивается посредством web-интерфейса, позволяющего задать входные данные и базовые настройки для выполнения анализа. Кроме того, клиентский интерфейс обеспечивает представление всех результирующих данных.

Средства реализации

В качестве основного языка программирования был выбран Java, как наиболее удобный и эффективный инструмент для создания высокопроизводительных приложений. Основными преимуществами использования Java являются кроссплатформенность и многопоточность создаваемого продукта.

На языке Java реализуются основные функции серверной части программы по обработке и анализу исходных текстов проекта, а также ведение логов.

Представление результирующих данных, а также обработка запросов пользователя на стороне сервера и загрузка данных, обеспечивается средствами PHP и СУБД MySQL.

Клиентская часть, представляющая собой web-интерфейс, реализована с использованием стандарта HTML5, имеющего в своей основе сам HTML, а также CSS и JavaScript. Браузеры, поддерживающие HTML5, делают это без необходимости устанавливать дополнительные плагины.

Схема средств реализации функционала представлена на рис. 2.

Принцип работы

Программный анализатор представляет собой ряд структурных компонент, каждый из которых

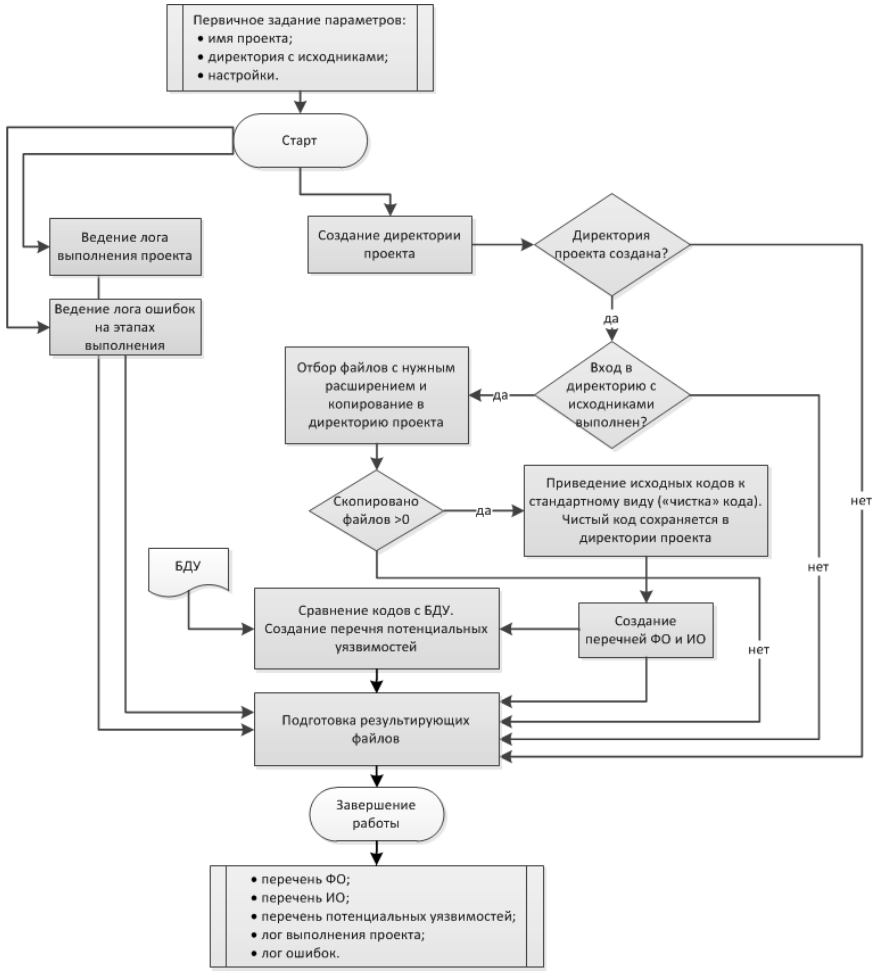

Рис. 1. Функциональная схема

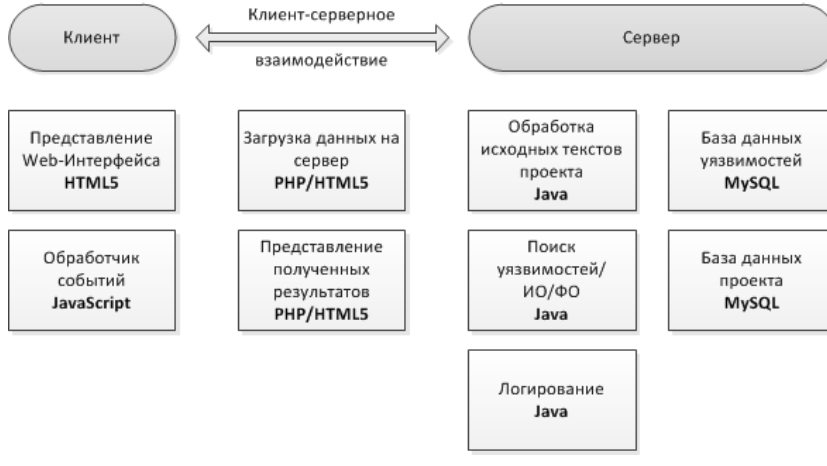

Рис. 2. Средства реализации функционала

реализует специфический набор сервисов. Схема компонент представлена на рис. 3.

Исходные данные, задаваемые пользователем посредством использования модуля web-интерфейса, передаются на сервер.

Пройдя первичную обработку в модуле обработки исходных текстов, «очищенные» коды анализируются на наличие функциональных и информационных объектов.

Результаты анализа поступают на вход модуля работы с базой данных проекта, где они обрабатываются и представляются в модуль web-интерфейса в табличном виде для возможности анализа со стороны экспертов.

На каждом этапе работы модуль логирования собирает информацию о ходе выполнения проекта, а так же возникающих ошибках. Полученные данные также доступны пользователю.

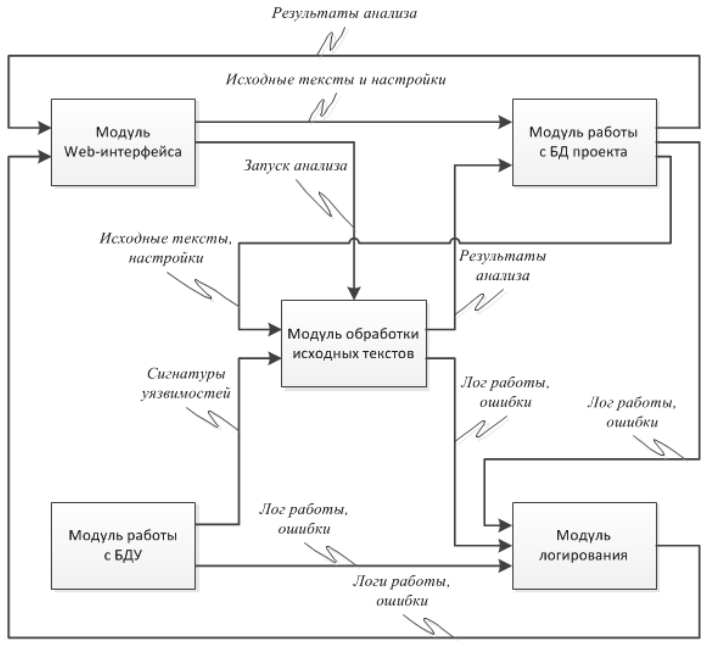

Рис. 3. Структурная схема

Модуль обработки исходных текстов

Модуль обработки исходных текстов представляет собой совокупность предварительной обработки кода, его лексического анализа, синтаксического анализа, семантического анализа и сигнатурного анализа.

На вход лексического анализатора поступают «очищенные» исходные тексты, с разрешёнными директивами препроцессора. На выходе формируется массив лексем, полученный из исходных текстов лексическим анализатором.

Лексический анализатор реализован с использованием автоматного стиля программирования. Главный цикл работает подобно конечному автомату. Переходы между состояниями происходят в зависимости от поданного на вход символа и текущего состояния автомата (в некоторых случаях рассматриваются предыдущий и последующий символ, т. к. язык C не является полностью контекстно-свободным).

Автомат имеет следующие состояния:

- before – начальное состояние автомата. В него автомат так же возвращается после завершения распознавания очередной лексемы;
- ID – идёт распознавание идентификатора (имя функции, класса, переменной и т.д.);
- str_const – идёт распознавание строковой константы;
- oper – идёт распознавание оператора;
- keyword – идёт распознавание ключевого слова;
- num_const – идёт распознавание числовой константы.

Во всех состояниях, кроме начального, входной символ проверяется на принадлежность терминальному символу функцией IsTerm. На вход функции подаётся текущее состояние и очередной прочитанный символ. Если символ не терминал, происходит дальнейшее чтение символов, пока не встретится терминальный. После этого происходит преобразование считанной подстроки символов в очередной элемент массива экземпляров класса Lexem, в соответствии с принадлежностью лексемы к определённому виду.

Синтаксический анализ зависит от работы лексического анализатора. На вход синтаксического анализатора поступает массив лексем со входа лексического анализатора. На выходе строится абстрактное синтаксическое дерево (далее по тексту АСД).

Данный функционал реализуется разбором потока входных лексем в соответствии с правилами приоритетов операторов, используя стек. Просмотр входного потока лексем осуществляется рекурсивно, в качестве выходной очереди выступает упорядоченное дерево, динамически изменяющее свой размер при поступлении нового листа. Каждый узел имеет указатель на родительский узел, а каждый узел, не являющийся листом – ещё и на дочерние узлы. Родительский указатель корня дерева равен NULL.

Если узел не является листом (символьные и числовые константы, операторы, передаваемые при вызове функций, L-values, терминалы и разделители), то при разборе входного потока лексем, ему будут созданы дочерние узлы, которые также будут иметь потомков, если они сами не являются листами.

Также, каждый узел дерева хранит данные – экземпляр класса Lexem.

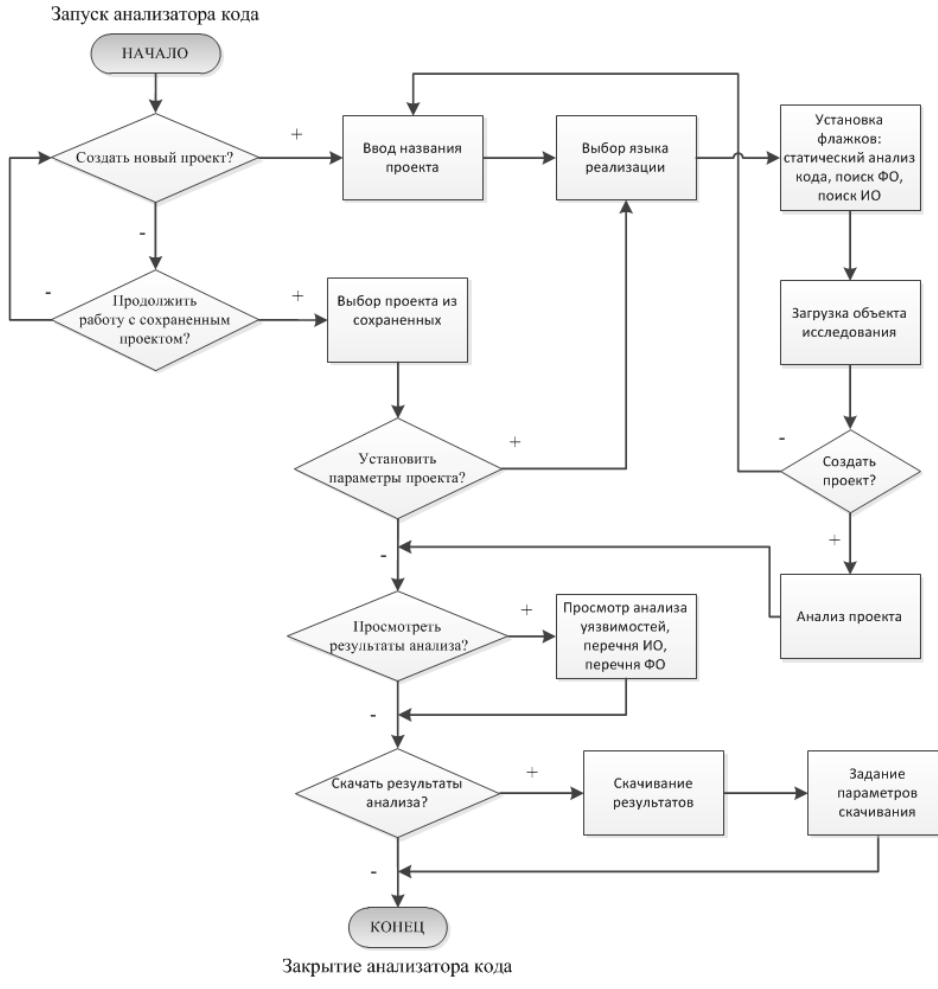

Рис. 2. Алгоритм работы web-интерфейса

Модуль Web-интерфейса

Взаимодействие с пользователем обеспечивается посредством web-интерфейса, позволяющего задать входные данные и базовые настройки для выполнения анализа.

Клиентская часть, представляющая собой web-интерфейс, реализуется с использованием стандарта HTML5, а также CSS и JavaScript.

В ходе разработки статического анализатора кода был разработан алгоритм работы web-интерфейса, представленный на рис. 4.

При запуске анализатора кода, пользователю открывается стартовая страница. Пользователь может выбрать проект из ранее сохраненных, либо создать новый проект.

Нажатием на кнопку «создать новый проект», пользователь переходит к форме, в которой должен указать название проекта и язык реализации, а также установить необходимые параметры анализа и загрузить сам объект исследования и нажать кнопку «создать проект».

Объект исследования будет загружен и на этом этапе есть возможность изменить параметры проекта и нажать «начать анализ».

По завершению анализа, пользователю представляется страница с результатами проведения анализа программного кода, поиска информационных и функциональных объектов. Результаты анализа доступны для просмотра и скачивания.

Пользователю всегда доступна кнопка «о программе», нажав на которую, пользователю представляется информация о статическом анализаторе кода (назначение анализатора кода, его функции, а также лаборатория-разработчик).

Выводы

В ходе выполнения данной работы разработана программная утилита, автоматизирующая процесс проведения статического анализа для программных продуктов, построенных на С-подобных языках программирования. В дальнейшем планируется развитие проекта в части расширения списка языков программирования, доступных к анализу, и функциональных возможностей для покрытия набора требований ФСТЭК России, предъявляемых к проведению статического анализа исходных текстов. В перспективе утилита может быть использована экспертами испытательной лаборатории при проведении сертифика-

ционных испытаний, а также разработчиками программных продуктов в ходе проведения отладки кода.

Литература

1. Руководящий документ. Средства вычислительной техники. Защита от несанкционированного доступа к информации. Показатели защищенности от несанкционированного доступа к информации.fstec.ru/component/attachments/download/297.

2. Руководящий документ. Защита от несанкционированного доступа к информации. Часть 1. Программное обеспечение средств защиты информации. Классификация по уровню контроля отсутствия недекларированных возможностей. fstec.ru/component/attachments/download/294.

3. Статический сигнатурный анализ безопасности программ. www.npo-echelon.ru/doc/stat-analisys.pdf. Марков А. С. Фадин А. А.

4. «Модульная структура программ». Большая энциклопедия нефти и газа. ngpedia.ru / id490497p1.html.