УДК 004. 41:004.056 **doi: 10.26906/SUNZ.2018.1.114**

А.В. Коваленко

Центральноукраинский национальный технический университет, Кропивницкий

ИМИТАЦИОННАЯ МОДЕЛЬ ТЕХНОЛОГИИ ТЕСТИРОВАНИЯ БЕЗОПАСНОСТИ WEB-ПРИЛОЖЕНИЙ

Разработана имитационная модель технологии тестирования безопасности на основе положений теории масштабирования имитационных моделей, отличающаяся от известных адаптацией выбора входных операторов управления и данных к повышению требований оперативности разработки и реализации модели, что выразилось в реализации процедуры взаимодействия с реальным браузером с использованием средств автоматизации браузера и формировании данных для атаки на нескольких диалектах.

Ключевые слова: *оценка рисков, разработка программного обеспечения, уязвимости безопасности*

Введение

Предлагаемый в работе подход к масштабированию имитационной модели основывается на положениях известных теорий масштабирования [7, 8] в которых зафиксированы основные этапы. В частности это выявление операторов, не влияющих на генерацию и ход событий, наблюдение которых требуется для проверки заданных свойств поведения системы (незначимых операторов) и абстрагирование описания имитационной модели от таких операторов [7, 8].

В разрабатываемой имитационной модели технологии тестирования уязвимостей предлагается не только простое абстрагирование от несущественного оператора, а его замена на более эффективный, с точки зрения точности конечных результатов проверки, оператор. Для имитационной модели технологии тестирования уязвимостей Web-приложений разработано программное приложение, проводящее атаки, на которые будет указывать предоставленная пользователем URL ссылка, по приведенным алгоритмам, и анализирующее результат атак на наличие определенной уязвимости в Web-приложении.

1. Общие требования и структура имитационной модели

Как было указано выше, для снижения временных затрат на имитационное моделирование было решено провести масштабирование путем замены процедур информационного обмена с сервером с помощью HTTP клиента, на процедуры взаимодействия с реальным браузером с использованием средств автоматизации браузера. Кроме основной цели масштабирования эта замена позволит повысить достоверность результата тестирования атаки с инъекцией Javascript кода. Как средство автоматизации браузера было выбрано библиотеку Selenium Webdriver.

Анализ литературы показал, что Selenium Webdriver представляет собой инструмент автоматизации браузера, который позволяет разрабатывать программные продукты, имеющие возможность управлять поведением браузера [1-6]. Для работы с Selenium Webdriver, необходимы следующие программные компоненты:

– Web-браузер (будет автоматизирован);

– драйвер для установленного браузера, который позволяет автоматизировать поведение браузера;

– непосредственно программный продукт, который состоит из набора команд определенного языка программирования для драйвера браузера, предоставляемых специальными библиотеками для многих языков программирования.

При реализации имитационной модели в качестве браузера был выбран Web-браузер Google Chrome. Google Chrome – браузер, разработанный компанией Google на основе браузера с открытым кодом Chromium и другого открытого программного обеспечения [1-6]. Это выбор обусловлен тем, что компанией Google в стратегии развития архитектуры определен факт, что сегодня большинство веб-сайтов являются не просто веб-страницами, но веб-приложениями. В качестве драйвера браузера использован ChromeDriver, который является имплементацией Webdriver для браузера Google Chrome.

Для разработки программного продукта, который будет управлять браузером и выполнять атаки на Web-приложения, необходимо выбрать язык программирования. Для реализации необходимого функционала к языку программирования выдвинуто следующие требования:

– наличие библиотеки команд Selenium Webdriver для данного языка программирования;

– возможность работы с HTTP с помощью нативных или посторонних HTTP клиентов;

– наличие нативной или сторонней системы журналирования с выводом лога в консоль и с сохранением лога в текстовом файле.

Также к языку программирования выдвинуто следующие требования для повышения качества результирующего продукта и повышение эффективности процесса программирования:

– независимость языка программирования от окружения, где будет выполняться программный продукт;

– наличие сторонних библиотек общего назначения;

– наличие инструмента для автоматизированного сбора программы и управления внешними зависимостями.

Существует несколько языков программирования, удовлетворяющие данным требованиям, например, Python, Ruby, Java. Однако, для программной реализации метода тестирования уязвимостей Web-приложений избран язык программирования Java.

Анализ литературы показал, что язык программирования Java соответствует требованиям к независимости от окружения с помощью компиляции исходного Java кода в байт-код, который является упрощенным машинными командами. Затем программу можно выполнить на любой платформе с Java, которая интерпретирует байткод в код, приспособленный к специфике конкретной операционной системы и процессора [1-14]. Сейчас виртуальные машины Java существуют для большинства процессоров и операционных систем.

Для сбора программ, разработанных с использованием языка программирования Java, существует три наиболее распространенные инструменты автоматизированного сбора – Apache Ant, Apache Maven и Gradle. Для реализации данного программного продукта избран инструмент Apache Maven.

Apache Maven – это средство автоматизации работы с программными проектами, который используется для Java проектов для управления и сбора программ. Для описания программного проекта который нужно построить (build), Maven использует конструкцию известную как Project Object Model (POM), зависимости от внешних модулей, компонентов и порядка построения. Maven базируется на плагин-архитектуре, позволяет сделать использование любой программы контролируемым через стандартный вход. Данный инструмент имеет не только функционал со сбором программ, а и по управлению внешними зависимостями и разбиение программного продукта на отдельные независимые модули с последующим сбором в единый модуль [1, 2].

Стандартная библиотека Java предоставляет возможность работы с сетями, в частности, протоколом HTTP [1-14]. Стандартная библиотека Java содержит классы для работы текстом, URL, коллекциями данных и другими функциями общего назначения.

Для языка программирования Java существует библиотека команд Selenium Webdriver [1-6]. Наиболее актуальную версию можно получить из репозитория Maven как отдельную библиотеку или использовать ее как внешнюю зависимость в Java проекте.

Язык программирования Java имеет большое количество библиотек для журнала с возможностями расширенного конфигурации. Для использования в программной реализации метода тестирования уязвимостей Web-приложений выбрано библиотеку журнала Apache Log4j Библиотека Apache Log4j2 имеет большое количество функций и возможностей конфигурации журнала, а именно возможность переадресации журналов выполнения программы к любому потока вывода, например, в текстовый файл на экран, в сетевой сокет и другие [1-14].

Таким образом, язык программирования Java удовлетворяет всем требованиям, которые были выдвинуты к языкам программирования для возможности реализации метода тестирования уязвимостей Web-приложений. Следует отметить, что для разработки приложения будет использовано последнюю версию Java, а именно Java 8.

Для разработки структуры программной реализации метода тестирования уязвимостей Web-приложений нужно обозначить множество функциональных требований, которые должна реализовывать приложение:

– получение входных параметров, а именно типа уязвимости для тестирования и URL веб-страницы для тестирования, в качестве параметров командной строки;

– для типа уязвимости DOM XSS, программный продукт должен получить код отдельной веб-страницы и код всех внешних Javascript файлов, которые используются. После этого, провести статический анализ всего присутствующего Javascript-кода на наличие определенных маркеров, которые могут приводить к уязвимости;

– для каждого GET-параметра в URL страницы нужно определить тип рефлексии параметра на странице и произвести XSS атаку с соответствующими для типа рефлексии данными и проверить ее результат;

– для типа уязвимости SQL Injection, программный продукт должен для каждого GET-параметра в URL страницы провести Boolean blind based SQL инъекцию и определить ее результат на основе анализа текста веб-страниц, который возвращает сервер;

– детальное логирование хода работы программного продукта на консоль и в файл.

Учитывая данные требования к приложению, общая структура программной реализации будет выглядеть как многоуровневый проект со следующими модулями:

– модуль интерфейса пользователя;
– модуль анализа уязвимостей DOM XSS;
– модуль анализа уязвимости к SQL инъекции;
– модуль функций общего назначения.

2. Структура Maven проекта

Из основных источников литературы [1-6], используемых в работе известно, что Java-программа представляет собой набор скомпилированных Java классов, собранные в исполняемый архив типа .jar.

Проведенный анализ литературы показал, что для сборки программного продукта в .jar файл существует несколько способов.

В данном проекте в качестве инструмента сбора проекта был выбран Apache Maven.

Исследования показали, что жизненный цикл сбора Maven проекта содержит фиксированный набор фаз, выполняемых одна за одной. Жизненный цикл по умолчанию состоит из следующих фаз:

– validate (выполняется проверка корректности проекта с точки зрения Maven);

– compile (выполняется компиляция файлов исходного кода);

– test (осуществляется тестирование скомпилированного кода с помощью предоставленного набора модульных тестов);

– package (реализуется упаковка скомпилированного кода в определенную форму дистрибуции, например в JAR архив);

– install (перемещается результат упаковки в локальный репозиторий для использования в локальных проектах);

– deploy (перемещается результат упаковки к удаленному репозиторию для использования другими разработчиками).

Apache Maven использует один или несколько файлов pom.xml для представления в формате XML структуры Java проекта. Project Object Model (POM) и содержит информацию о структуре проекта, внутренних и внешних зависимостях, информации о процессе сбора проекта и плагинов, которые будут использованы при определенных фазах сбора проекта [2].

Maven поддерживает структуру многомодульных проектов посредством подражания и агрегации проектов.

Это достигается введением дополнительных артефактов со специальным маркером в файлах *pom.xml*.

При использовании такой конфигурации, общие характеристики и настройки свойства можно выносить в базовый файл *pom.xml*, который будет унаследован проектами-потомками.

Минимальными необходимыми элементами файла *pom.xml* есть координаты проекта, содержат следующие элементы:

– groupId – уникальный идентификатор проекта или организации, которая разрабатывает проект. Используется как аналог пакета в языке Java.

– artifactId – уникальный идентификатор артефакта в проекте. Артефактом может быть как сам проект, так и модуль многомодульных проекте.

– version – версия артефакта. Используется для Версионирование артефакив в репозитории.

– packaging – тип артефакта проекта Maven. В общем случае имеет значение jar для обычных проектов или отдельных модулей, и pom для многомодульных проекта (агрегатора).

Для программной реализации метода тестирования уязвимостей Web-приложений был избран многомодульная структуру, следовательно, Maven проект также будет разработан как многомодульных проект с несколькими модулями.

Таким образом, учитывая общую структуру приложения, Maven проект будет состоять из корневого проекта, проекта модулю интерфейса, проекта модулю функций общего назначения, корневого проекта модулей анализа уязвимостей и непосредственно проектов модулей анализа уязвимостей.

Предложенная структура Maven проекта является расширяемой, так как наличие корневых проектов позволяет легко подсоединить дополнительные модули анализа уязвимостей.

Корневой Maven проект (*dxss-sqli-framework*) определяет идентификатор группы для всех дочерних проектов как *com.ntukhpi.kit.serg_sem*. Проект является агрегатором, поэтому параметр *packaging* имеет значение *pom*.

```
<groupId> com.ntukhpi.kit.serg_sem </groupId>
<artifactId>dxss-sqli-framework</artifactId>
<version>1.0-SNAPSHOT</version>
<packaging>pom</packaging>
```

Корневой проект является агрегатором модулей, то есть проект содержит ссылки на все дочерние проекты, используемых в приложении.

```
<modules>
    <module>framework-core</module>
    <module>attack-modules</module>
    <module>core-utils</module>
  </modules>
```

Проект также содержит зависимости, используемые всеми дочерними проектами, а также указание Maven использовать версию 8 языка программирования Java при компиляции исходных файлов.

```
<build>
    <plugins>
    <plugin>
    <groupId>org.apache.maven.plugins</groupId>
    <artifactId>maven-compiler-plugin</artifactId>
    <version>1</version>
    <configuration>
                <source>1.8</source>
                <target>1.8</target>
    </configuration>
    </plugin>
    </plugins>
</build>
```

Проект модуля функций общего назначения называется *core-utils*. Проект имеет тип упаковки *jar* и является дочерним к проекту *dxss-sqli-fraework*.

```
<artifactId>core-utils</artifactId>
<packaging>jar</packaging>
<parent>
   <artifactId>dxss-sqli-framework</artifactId>
   <groupId>com.ntukhpi.kit.mark_dubrovskyi</groupId>
   <version>1.0-SNAPSHOT</version>
</parent>
```

Корневой проект модулей анализа уязвимостей называется *attack-modules*. Проект является агрегатором, поэтому параметр *packaging* имеет значение *pom*. Данный проект является дочерним для *dxss-sqli-fraework* и содержит ссылки на проект *core-utils* как зависимость.

```
<artifactId>attack-modules</artifactId>
<packaging>pom</packaging>
<parent>
    <artifactId>dxss-sqli-framework</artifactId>
    <groupId> com.ntukhpi.kit.serg_sem </groupId>
    <version>1.0-SNAPSHOT</version>
```

```
</parent>
<modules>
    <module>sqli-module</module>
    <module>dxss-module</module>
</modules>
<dependencies>
    <dependency>
    <groupId> com.ntukhpi.kit.serg_sem </groupId>
              <artifactId>core-utils</artifactId>
             <version>1.0-SNAPSHOT</version>
    </dependency>
</dependencies>
```

Проект модуля анализа уязвимости DOM XSS называется *dxss-module*. Проект имеет тип упаковки *jar* и является дочерним к проекту *attack-modules*.

Проект модуля анализа уязвимости к SQL инъекции называется *sqli-module*. Проект имеет тип упаковки *jar* и является дочерним к проекту *attack-modules*.

Проект модулю интерфейса называется *framework-core*. Проект имеет тип упаковки *jar*, является дочерним для *dxss-sqli-fraework* и содержит ссылки на проекты *sqli-module* и *dxss-module* как зависимости.

Общая иерархия проектов в многомодульных Maven проекте ghtlcnfdktyf на рис. 1.

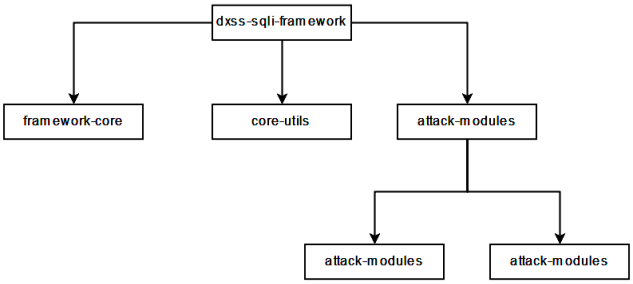

Рис. 1. Иерархия проектов
в многомодульном Maven проекте

3. Структура модуля интерфейса

В разработанной имитационной модели технологии тестирования уязвимостей, модуль интерфейса отвечает за запуск приложения, обработку параметров командной строки, и выполнение теста Web-приложения на наличие определенной уязвимости в зависимости от переданных параметров командной строки. Для запуска приложения, разработанного на языке программирования Java, нужен класс, который содержит статический метод *main*, аргументами к которому передаются параметры командной строки. В программной реализации имитационной модели метода тестирования уязвимостей Web-приложений данный класс называется *Application*. Данный класс передает параметры командной строки другим классам на обработку и обрабатывает исключительные ситуации во время выполнения программы.

Для обработки параметров командной строки с целью максимальной масштабируемости использован шаблон проектирования «Команда». Команда – это шаблон проектирования, который относится к классу шаблонов поведения [3]. Он инкапсулирует запрос в форме объекта, что, в свою очередь, позволяет использовать единый интерфейс выполнения различных действий. Данный шаблон проектирования целесообразно использовать при работе с командной строкой, когда более чем один вариант работы приложения. UML диаграмма шаблона проектирования «Команда» приведена на рис. 2.

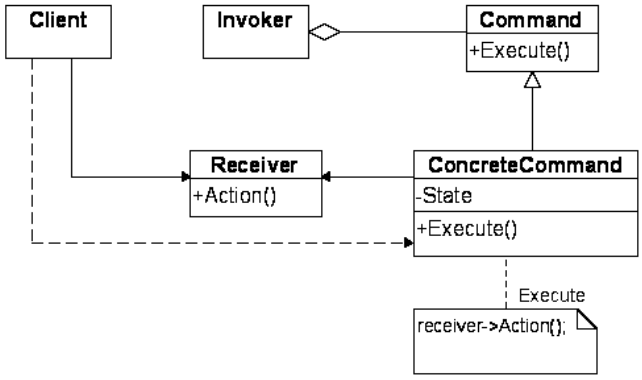

Рис. 2. UML диаграмма
шаблона проектирования «Команда»

Структурные элементы шаблона выполняют следующие функции:

– Command – объявляет интерфейс выполнения операции;

– ConcreteCommand – реализует конкретную команду, вызывая определенные методы объекта Receiver;

– Client – создает объекты ConcreteCommand и устанавливает ее получателей;

– Invoker – обращается к командам с целью выполнения.

UML диаграмма реализации данного шаблона проектирования приведена на рис. 3. В данной реализации шаблона «Команда» присутствует интерфейс Command, его имплементации SqlInjectionAttack и DomXssAttack, а также контейнер команд CommandHolder.

В качестве структурного элемента Invoker шаблона проектирования «Команда» выступает класс Shell, который принимает параметры командной строки от класса Application и использует их для получения нужного объекта – команды и его выполнение. Таким образом, общая UML диаграмма модуля интерфейса изображена на рис. 4.

4. Структура модулей анализа уязвимостей к SQL инъекции и DOM XSS

Как было указано выше модуль анализа уязвимостей DOM XSS предназначен для анализа отдельной Web-страницы на наличие уязвимости DOM XSS. В соответствии с алгоритмом тестирования уязвимости на DOM XSS, модуль имеет классы для получения Javascript кода Web-страницы, классы для анализа Javascript кода и класс для проведения атак.

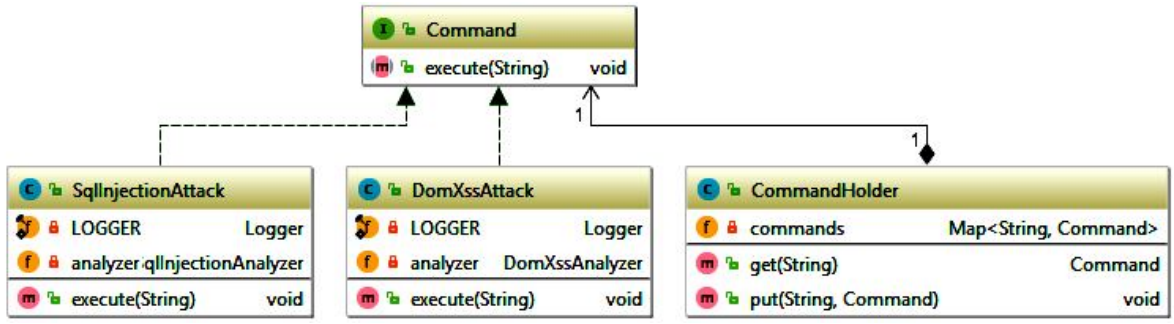

Рис. 3. UML диаграмма реализации шаблона проектирования «Команда»

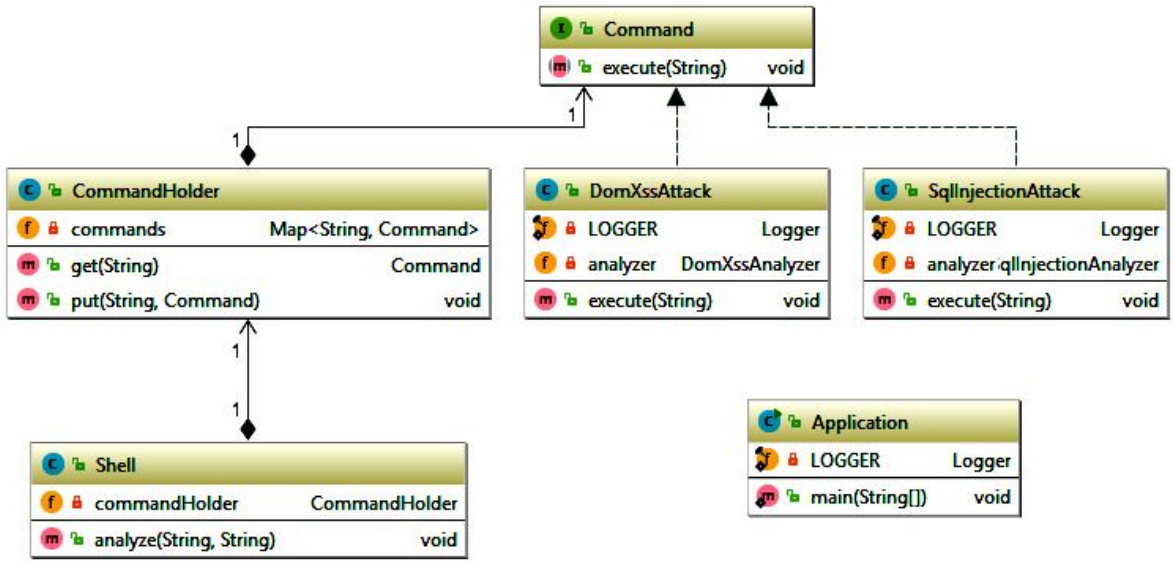

Рис. 4. Общая UML диаграмма модуля интерфейса

Главным классом модуля является класс *DomXssAnalyzer*, который отвечает за непосредственно анализ Web-страницы, на которую указывает переданный URL на наличие уязвимости DOM XSS. Данный класс компонует все классы модуля для выполнения анализа и является ответственным за определение типа рефлексии GET параметра URL, а также непосредственно за проведение атаки.

Класс DomScanner используется для анализа исходного HTML кода страницы и получения списка всех тегов *script* с их содержанием. Класс содержит потоковый HTML «парсер» в виде класса *ResponseHtmlHandler*, который формирует список тегов *script*.

Для выполнения анализа Javascript кода страницы на наличие маркеров уязвимости к DOM XSS атак, используются классы Javascript и JavascriptScanner. Класс Javascript используется для описания характеристик единицы Javascript кода Web-страницы, а именно его тип (внешний или внутренний), ссылку на внешний файл, непосредственно Javascript код и список маркеров уязвимости, который заполняется после анализа. Класс JavascriptScanner используется для получения содержания внешних Javascript файлов и для анализа Javascript кода на наличие маркеров уязвимости в виде стоков и истоков.

Класс PayloadGenerator используется для генерации данных для атаки в зависимости от типа рефлексии параметра GET запроса URL.

Общая UML диаграмма модуля анализа уязвимостей DOM XSS приведена на рис. 5.

Модуль анализа уязвимости к SQL инъекции предназначен для анализа отдельной Web-страницы на наличие уязвимости к SQL инъекции через GET параметр URL Web-страницы. Главным классом модуля является класс SqlInjectionAnalyzer, который отвечает за проведение Boolean blind SQL инъекции и анализа результата атаки с целью определения наличия потенциальной уязвимости.

Одной из сложностей тестирования уязвимости к SQL инъекций является большое количество диалектов SQL в зависимости от используемой системы управления базами данных. Этот факт заставляет формировать данные для атаки на нескольких диалектах для максимального покрытия векторов атаки. С одной стороны это увеличивает количество входных данных имитационной модели, повышает сложность проекта и негативно влияет на общие временные характеристики реализации и проведения тестирования уязвимости. С другой стороны, используя предложенный подход масштабирования можно существенно снизить сложность проекта, не ухудшая качественных характеристик.

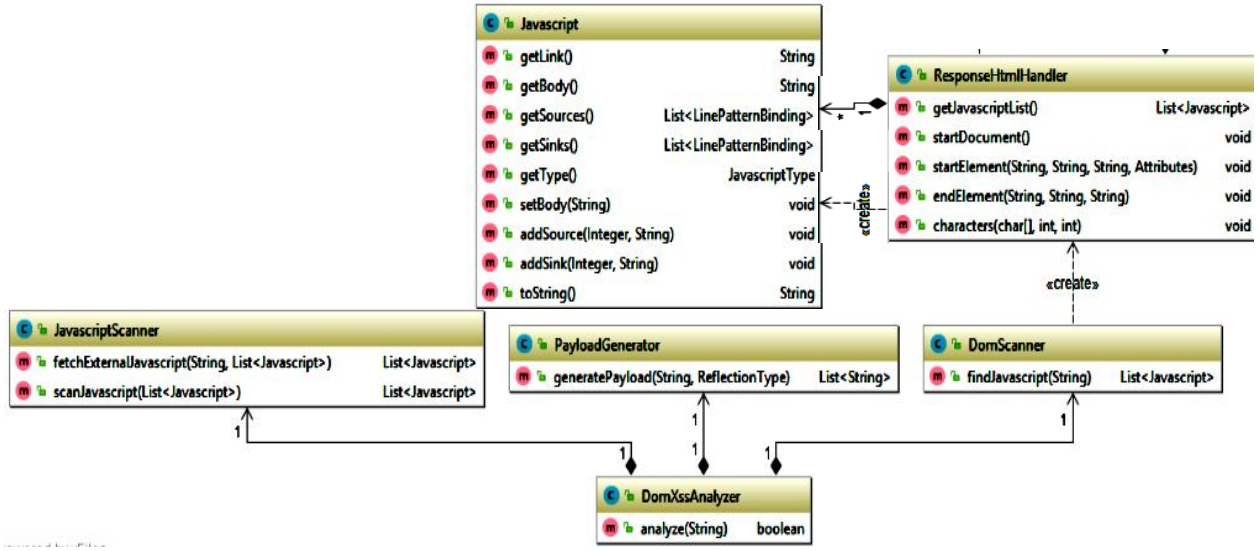

Рис. 5. UML диаграмма модуля анализа уязвимостей DOM XSS

Учитывая это и с целью устранения дублирования кода, в приложении реализован интерфейс DBMSSpecificPayload, который описывает метод, возвращающий список данных для проведения атаки с учетом конкретной системы управления базами данных. Классы, реализующие данный интерфейс, является Common, Oracle и MySql для общих данных атаки, данных, специфичных для систем управления базами данных Oracle и MySQL соответственно. Данные для атаки хранятся в приложении в виде класса BooleanBlindPayload, который включает в себя необходимые для проведения атаки данные.

Общая UML диаграмма модуля анализа уязвимости к SQL инъекции приведена на рис. 6.

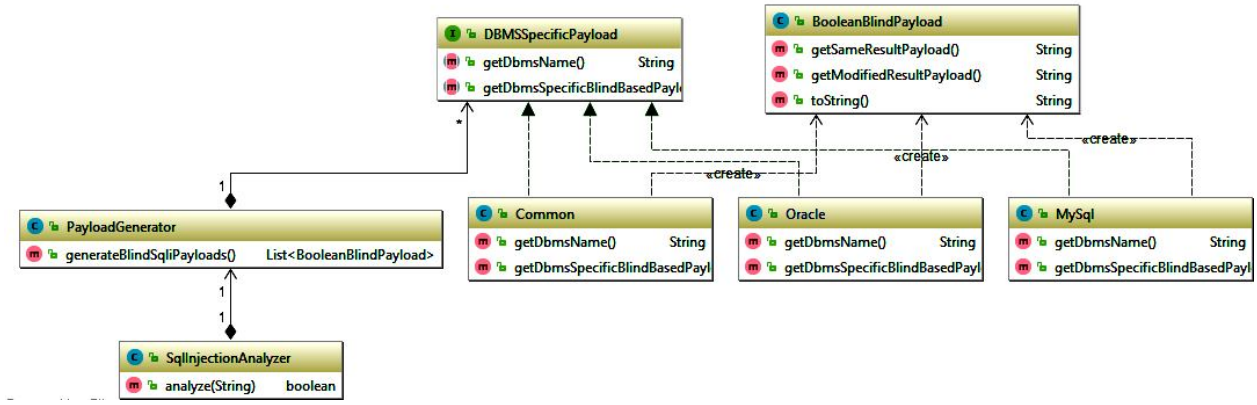

Рис. 6. UML диаграмма модуля анализа уязвимости к SQL инъекции

5. Структура модуля функций общего назначения

Модуль функций общего назначения предоставляет набор функций для работы с URL ссылками и их параметрами, а также для взаимодействия с сервером непосредственно с помощью протокола HTTP.

Для преобразования URL и манипуляции параметрами URL используется класс UrlUtils. Поскольку функции преобразования URL и манипуляции их параметрам не имеют состояния, они реализованы как статические методы.

Для взаимодействия с сервером по протоколу HTTP используется класс HttpUtils. Данный класс реализует шаблон проектирования «Строитель» для строительства внутреннего класса HttpGetRequest. Строитель – образующая шаблон проектирования, позволяет отделить конструирование комплексных объектов от их реализации [4]. В данном случае, шаблон использован для конструирования сложного объекта класса HttpGetRequest, который инкапсулирует результат выполнения GET запроса протокола HTTP. UML диаграмма модуля функций общего назначения изображена на рис. 7.

Имитационная модель технологии тестирования уязвимостей Web-приложений является комплексным программным обеспечением с большим количеством модулей и классов, что в свою очередь, создает проблему компоновки классов между собой. Этой проблемы можно избежать уменьшением количества классов, но это приведет к увеличению, сильно связанных между собой классов, в свою очередь, усложнит разработку, поддержку, читабельность и масштабируемость программного обеспечения.

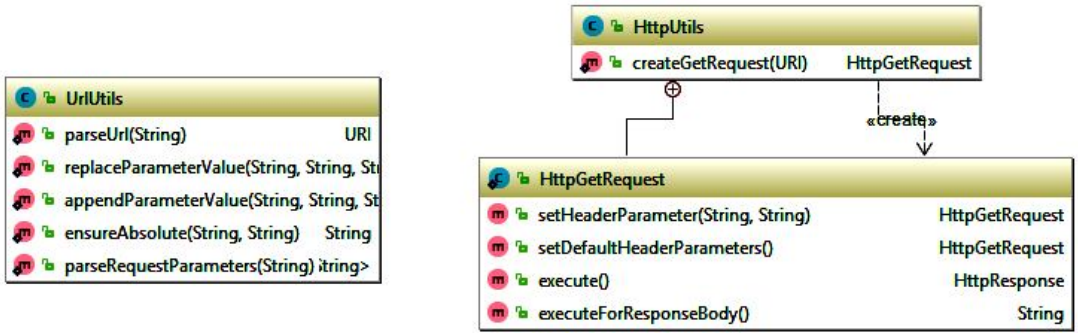

Рис. 7. UML диаграмма модуля функций общего предназначения

Для решения задачи компоновки классов использовано механизмы «внедрения зависимостей» [1-14]. Из литературы [1-14] известно, что «внедрение зависимостей» – это шаблон проектирования, в котором зависимости (или сервисы) внедряются, или передаются по ссылке в зависимый объект (клиент) и становятся частью клиентского состояния. Шаблон отделяет создания зависимостей клиента от собственной логики клиента, позволяет компонентам быть слабо связанными и придерживаться принципов инверсии зависимостей и единого долга [1-14]. Существует три основных типа «внедрения зависимостей» в класс:

– внедрение в конструктор;
– внедрение в свойство;
– внедрение в метод.

В общем случае шаблон «внедрения зависимостей» не привязан к программной реализации и не требует отдельных библиотек или фреймворков для реализации, однако, использование контейнеров для «внедрения зависимостей» значительно упрощает процесс разработки.

Одним из самых популярных контейнеров для «внедрения зависимостей» в экосистеме Java является Spring Framework. Spring Framework – фреймворк для приложений, разработанных на языке программирования Java, который предоставляет контейнер для внедрения зависимостей, поддержку аспектно-ориентированного программирования и многие другие функции [5].

Проведенные исследования показали, что актуальной на момент разработки является версия 4.3.8 от 18.04.2017, что свидетельствует о постоянном развитии фреймворка.

Контейнер Spring позволяет централизовать создания объектов приложения и автоматически решить все зависимости созданных компонентов. При этом, контейнер берет на себя задачи по созданию компонентов, управлению их жизненных циклов и уничтожению компонентов при уничтожении контейнера. Spring также поддерживает еще один вид «внедрения зависимостей», а именно инъекция в частную поле.

Для решения зависимостей Spring использует внешний конфигурационный файл в виде XML файла или Java класса. Использование контейнера Spring имеет следующие этапы:

1. Конфигурация контейнера во внешнем файле.

2. Создание контекста приложения со ссылкой на файл конфигурации.

3. Получение необходимых компонентов из контекста.

Для разработки имитационной модели технологии тестирования Web-приложений было выбрано конфигурацию контейнера для «внедрения зависимостей» в виде Java класса. Каждый модуль многомодульного проекта содержит свой класс с конфигурацией, который отвечает за создание компонентов только своего модуля, после чего все конфигурационные файлы будут использованы при создании контекста приложения, что, в свою очередь, приведет к созданию контейнера для «внедрения зависимостей» Spring и созданию всех необходимых компонентов приложения.

Класс конфигурации компонентов Spring имеет определенную структуру. Минимальным требованием к классу является аннотирования класса аннотацией @Configuration, что позволяет использовать класс для конфигурации контейнера для внедрения зависимостей при запуске приложения. Класс конфигурации должен содержать методы, создают компоненты приложения. Данные методы должны быть аннотированные аннотацией @Bean. При использовании Spring, для полного внедрения всех зависимостей все компоненты должны быть созданы в конфигурационных классах. В случае использования нескольких конфигурационных классов, должен существовать главный конфигурационный класс, аннотированный кроме аннотации @Configuration также аннотацией @Import, содержащей ссылку всем остальным конфигурационных классов.

Таким образом, получила дальнейшее развитие имитационная модель технологий тестирования безопасности Web-приложений. В основу разработки положены основные положения теории масштабирования имитационных моделей в рамках алгоритмического упрощения на основе оценки транзитивной зависимости по управлению и данным. Отличительной особенностью разработанной имита-

ционной модели является адаптация выбора входных операторов управления и данных к повышению требований оперативности разработки и реализации модели, выраженная в реализации процедуры взаимодействия с реальным браузером с использованием средств автоматизации браузера и формировании данных для атаки на нескольких диалектах.

Такие изменения позволили снизить вычислительную сложность процедур тестирования уязвимостей к SQL инъекции и DOM XSS до 1,5 раз.

6. Тестирование имитационной модели разработанной технологии

Опишем процедуру тестирования имитационной модели на примере анализа уязвимостей DOM XSS. Для тестирования имитационной модели в соответствующем режиме нужно запустить собранный проект с параметрами командной строки *dxss* [URL]. Для тестирования было выбрано Web-приложение *https://xss-game.appspot.com/level1*, которое предоставлено фирмой Google для тренировки навыков инженеров по информационной безопасности. Проведенный анализ показал, что данное Web-приложение создано с известной уязвимостью к DOM XSS атакам.

Разработанное приложение в режиме тестирования имеет следующие параметры командной строки:

dxss https://xss-game.appspot.com/level1/frame?query=123

После запуска приложения открывается окно браузера с соответствующим URL (рис. 8).

После открытия Web-страницы выполняется анализ Javascript кода на странице. При анализе в журнале приложения выводится информация о найденных Javascript файлах и их типах (рис. 9).

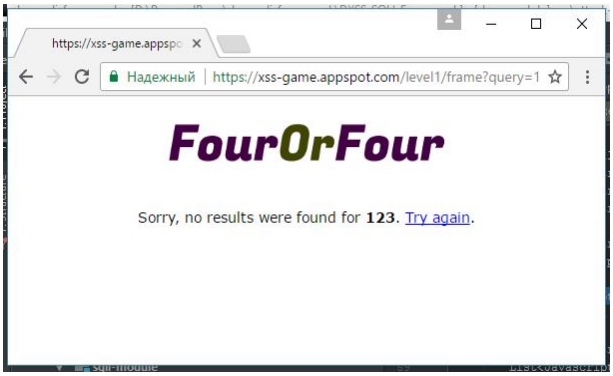

Рис. 8 Внешний вид окна браузера с соответствующим URL

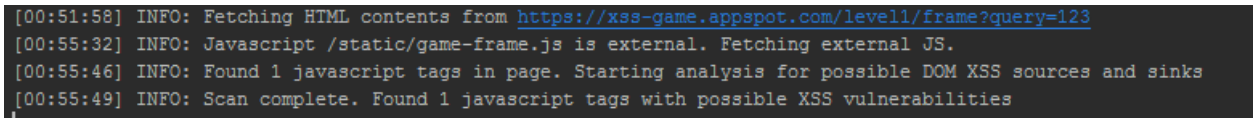

Рис. 9. Внешний вид информации о найденных Javascript файлах и их типах

После анализа Javascript кода на наличие маркеров уязвимости выполняется определение типа рефлексии параметра URL. Для этого генерируется уникальное значение и вставляется как значение параметра URL, после чего выполняется переход по ссылке (рис. 10).

Рис. 10 Внешний вид окна определения типа рефлексии параметра URL

Затем, исходя из типа рефлексии параметра, определяется набор данных для атаки и проводится атака с каждым набором данных. Для простоты тестирования в качестве маркера было использовано диалоговое окно Alert.

В случае успешной атаки набор данных записывается (логируется) в текстовый файл (рис. 11).

Выводы

В работе получила дальнейшее развитие имитационная модель технологии тестирования безопасности на основе положений теории масштабирования имитационных моделей. Отличительной особенностью разработанной имитационной модели является адаптация выбора входных операторов управления и данных к повышению требований оперативности разработки и реализации модели, выраженная в реализации процедуры взаимодействия с реальным браузером с использованием средств автоматизации браузера и формировании данных для атаки на нескольких диалектах.

В основу предложенного подхода алгоритмического упрощения имитационного моделирования проложены усовершенствованные процедуры оценки транзитивной зависимости по управлению и данным. Определено допустимость и целесообразность использования оценки транзитивной зависимости, что снизит вычислительную сложность реализуемых алгоритмов по сравнению с алгоритмами оценки прямой зависимости до 1,5 раз.

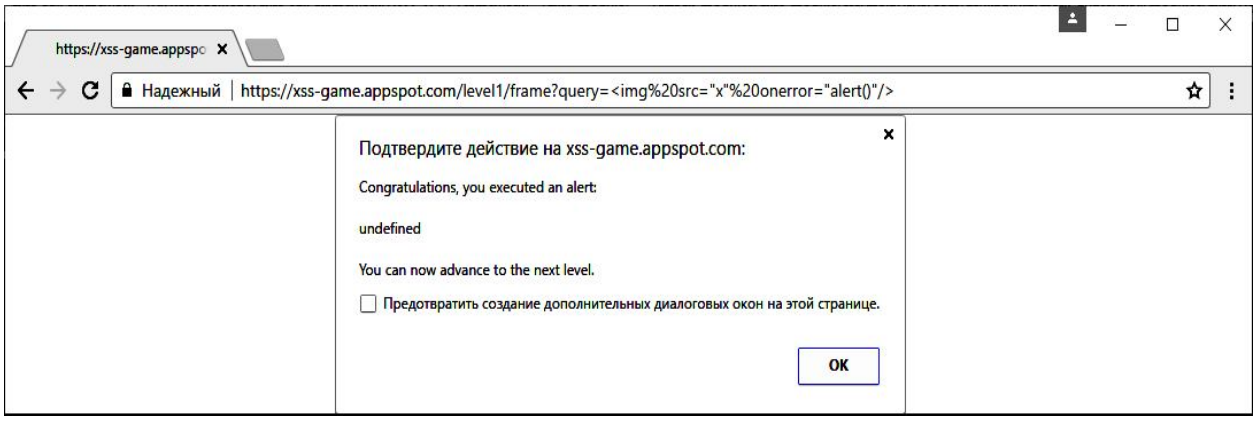

Рис. 11 Внешний вид окна сигнализирующего об успешно проведенной DOM XSS атаке

Список литературы

1. Maven – Introduction: [Електронний ресурс]. – Режим доступу: https://maven.apache.org/what-is-maven.html.

2. Maven – POM Reference: [Електронний ресурс]. – Режим доступу: https://maven.apache.org/pom.html.

3. Gamma E. Design Patterns: Elements of Reusable Object-Oriented Software. / Erich Gamma. – Reading, Mass. : Addison-Wesley, 1995.

4. Fowler M. Inversion of Control Containers and the Dependency Injection pattern: [Електронний ресурс] / Martin Fowler. – Режим доступу: https://martinfowler.com/articles/injection.html.

5. Spring Framework: [Електронний ресурс]. – Режим доступу: http://projects.spring.io/spring-framework/.

6. Ranganath V., Amtoft T., Banerjee A., Dwyer M., Hatcliff J. A new foundation for controldependence and slicing for modern program structures. Technical report 8, santos lab, Kansas State University, 2004.

7. Савенков К. О. Использование зависимостей при масштабировании имитационных моделей. In Методы и средства обработки информации. Труды второй Всероссийской научной конференции, pages 428–434. - М.: Издательский отдел факультета Вычислительной Мате- матики и Кибернетики МГУ им. М.В. Ломоносова, 2005.

8. Семенов С.Г., Швачич Г.Г., Карпова Т.П., Волнянський В.В. Застосування багатопроцесорних систем для удосконалення технологічних процесів // Системи обробки інформації. – Х.: ХУПС, 2016. – Вип. 3(140) С. 221-226.

9. Коваленко А.В. Методы качественного анализа и количественной оценки рисков разработки программного обеспечения / А.А. Смирнов, А.В. Коваленко // Системи обробки інформації. – Вип.. 5(142). – Х.: ХУПС, 2016. – С. 153-157.

10. Проблемы анализа и оценки рисков информационной деятельности / А.А. Смирнов, А.В. Коваленко, Н.Н. Якименко, А.П. Доренский // Системи обробки інформації. – Вип 3(140). – Х.: ХУПС, 2016. – С. 40-42.

11. Метод качественного анализа рисков разработки программного обеспечения / А.А. Смирнов, А.В. Коваленко, Н.Н. Якименко, А.П. Доренский // Наука і техніка Повітряних Сил Збройних Сил України. – Випуск 2(23). – Харків: ХУПС. – 2016. – С. 150-158.

12. Метод количественной оценки рисков разработки программного обеспечения / А.А. Смирнов, А.В. Коваленко, Н.Н. Якименко, А.П. Доренский // Збірник наукових праць ХУПС. Вип. 2 (47). – Харків: ХУПС, 2016. – С. 128-133.

13. Коваленко А.В. Использование псевдобулевых методов бивалентного программирования для управления рисками разработки программного обеспечения / А.А. Смирнов, А.В. Коваленко // Системи управління, навігації та зв'язку. – Випуск 1 (37). – Полтава: ПолтНТУ, 2016. – С. 98-103.

14. Коваленко А.В. Метод управления рисками разработки программного обеспечения / А.В. Коваленко // Системи управління, навігації та зв'язку. – Вип. 2 (38). – Полтава: ПолтНТУ, 2016. – С. 93-100.

Надійшла до редколегії 30.11.2017

Рецензент: д-р техн. наук, с.н.с. С.Г. Семенов, Національний технічний університет «Харківський політехнічний інститут», Харків.

ІМІТАЦІЙНА МОДЕЛЬ ТЕХНОЛОГІЇ ТЕСТУВАННЯ БЕЗПЕКИ WEB-ДОДАТКІВ

О.В. Коваленко

У даній роботі розроблено імітаційну модель технології тестування безпеки на основі положень теорії масштабування імітаційних моделей, що відрізняється від відомих адаптацією вибору вхідних операторів управління і даних до підвищення вимог оперативності розробки та реалізації моделі, що виразилося в реалізації процедури взаємодії з реальним браузером з використанням засобів автоматизації браузера і формуванні даних для атаки на декількох діалектах.

Ключові слова: *оцінка ризиків, розробка програмного забезпечення, уразливості безпеки.*

IMITATION MODEL OF TECHNOLOGY OF SAFETY TESTING OF WEB-APPLICATIONS

O.V. Kovalenko

In this paper, a simulation model of security testing technology has been developed based on the theory of scaling of simulation models, which differs from those known for adapting the choice of input control and data operators to the increase in the requirements for the rapid development and implementation of the model, which resulted in the implementation of the procedure for interacting with a real browser using browser automation tools And the formation of data for attack in several dialects.

Keywords: *risk assessment, software development, security vulnerability.*