МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ

Федеральное государственное бюджетное образовательное учреждение

высшего образования

«Астраханский государственный университет»

Допускается к защите

«7» июня 2018 г.

Заведующий кафедрой ИБ, д.т.н.

_____ Ажмухамедов И.М.

БАКАЛАВРСКАЯ РАБОТА

Разработка схемы защиты от межсайтового скриптинга

Исполнитель:

студент группы ЗИ-41

_____ Носиров З.А.

«1» июня 2018 г.

Согласовано:

Кафедра информационной безопасности

нормоконтролер, к.т.н., доцент

_____ Гурская Т.Г.

«5» июня 2018 г.

Руководитель:

Кафедра информационной безопасности

зав. каф. ИБ, д.т.н., доцент

_____ Ажмухамедов И.М.

«4» июня 2018 г.

Текстовая документация БР 10.03.01.031.2018

Астрахань 2018

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Астраханский государственный университет»

Факультет математики и информационных технологий
Направление подготовки «Информационная безопасность»
Кафедра информационной безопасности

УТВЕРЖДАЮ

Зав. кафедрой _____

«1» декабря 20 17 г.

ЗАДАНИЕ

на выпускную квалификационную работу студента

Носирова Зафаржона Амрулоевича

1 Тема бакалаврской работы Разработка схемы защиты от межсайтового скриптинга утверждена приказом по университету от «27» ноября 2018 г. № 08-01-06/2014

2 Дата выдачи задания по ВКР «30» ноября 2017 г.

3 Исходные данные к проекту:

Одним из опасных видов компьютерных атак является межсайтовый скриптинг, в англоязычной литературе называемый – XSS (cross site scripting, x – используется в данной аббревиатуре для краткости, c – не используется, чтобы избежать путаницы с CSS). Также межсайтовый скриптинг является одним из самых распространенных компьютерных атак, по версии OWASP (открытого проекта обеспечения безопасности веб-приложений). Об этом же свидетельствуют и результаты исследования компании Positive Technologies. Межсайтовый скриптинг основан на эксплуатации XSS-уязвимостей веб-приложения. С помощью XSS-уязвимостей нарушитель внедряет вредоносный программный код в веб-страницу, отправляемую сервером клиенту. Часто уязвимость, позволяющую реализовать данный тип компьютерной атаки, также называют XSS.

4 Функции, реализуемые программой:

- детектирование XSS-уязвимостей;
- создание карты веб-сайта;
- предварительная авторизация в веб-приложении;

- справочник уязвимостей;

- создание отчета о выполнении тестирования;

- формирование рекомендаций по найденным уязвимостям;

- сохранение отчета.

5 Содержание пояснительной записки:

- исследование предметной области: актуальность, цель и задачи бакалаврской области;

- выявление особенностей компьютерных атак вида межсайтовый скриптинг;

- исследование программных решений направленных на обнаружение XSS-уязвимостей: выявление недостатков в существующих решениях;

- разработка программного обеспечения для обнаружения XSS-уязвимостей на основе анализа полной карты веб-приложения;

- расчёт экономической эффективности проектного решения.

6 Перечень графического материала:

- результаты исследования Positive Technologies, 2017 г.;

- классификация компьютерных атак вида межсайтовый скриптинг;

- категории веб-уязвимостей;

- общая схема проведения XSS-атаки;

- DF-диаграмма проектного решения;

- ER-диаграмма проектного решения;

- блок-схема алгоритма поиска «отраженных» XSS-уязвимостей;

- блок-схема алгоритма поиска «XSS-уязвимостей, эксплуатирующих DOM»;

- блок схема алгоритма поиска «хранимых» XSS-уязвимостей;

- блок-схема разработанной программы детектирования XSS-уязвимостей.

Руководитель _____ (подпись) Ажмухамедов И.М.

Задание принял к исполнению _____ (подпись) Носиров З.А.

РЕФЕРАТ

Ключевые слова: МЕЖСАЙТОВЫЙ СКРИПТИНГ, XSS-АТАКА, ВНЕДРЕНИЕ КОДА, XSS-УЯЗВИМОСТЬ, СКРИПТИНГ, ВРЕДОНОСНЫЙ КОД.

Работа изложена на 85страницах и состоит из введения, 3-х глав, заключения, списка использованной литературы, содержащего 33 наименований, и 5 приложений. В работе имеется 25 рисунков и 12 таблиц.

В бакалаврской работе были проанализированы особенности проведения компьютерных атак вида «межсайтовый скриптинг» и проведен анализ существующих программ предназначенных для обнаружения XSS-уязвимостей. Результаты анализа показали, что программные средства осуществляют поиск XSS-уязвимостей в «открытой части» (то есть не рассматриваются те веб-страницы, появляющиеся после авторизации). Поэтому целью бакалаврской работы является повышение эффективности защиты веб-приложения от межсайтового скриптинга, путем разработки программного обеспечения для детектирования уязвимостей на основе анализа полной карты веб-приложения.

Для достижения поставленной цели были разработаны эффективные алгоритмы поиска XSS-уязвимостей, предусматривающие поиск в «закрытой части» веб-приложения. Разработанные алгоритмы были реализованы в виде программного обеспечения, которое успешно апробировано в ООО НПП «ДосЛаб», о чем свидетельствует соответствующий акт внедрения.

Экспериментально доказана конкурентоспособность разработанной программы. Проведена оценка экономической эффективности внедрения программного обеспечения, также рассмотрены вопросы, связанные с техникой безопасности и охраной труда.

Результаты бакалаврской работы опубликованы в 3 печатных изданиях, один из которых входит в перечень ВАК. Работа была представлена и удостоена наград в следующих конкурсах:

- удостоена статуса финалиста в восемнадцатой всероссийской конкурс-конференции среди студентов и аспирантов по информационной безопасности «SIBINFO-2018»;

- удостоена статуса победителя в конкурсе проектных работ имени академика А.А. Бочвара.

STRUCTURAL ABSTRACT

Key words: CROSS SITE SCRIPTING, XSS-ATTACK, CODE INTRODUCTION, XSS-VULNERABILITIES, SCRIPTING, EXPLOIT.

The work is presented on 85 pages and consists of an introduction, 3 chapters, conclusion, list of references containing 33 titles, and 5 applications. The work has 25 figures and 12 tables.

In the bachelor's work, the features of computer attacks of the "cross-site scripting" type were analyzed and the existing programs designed to detect XSS vulnerabilities were analyzed. The results of the analysis showed that the software searches for XSS vulnerabilities in the "open part" (that is, those web pages that appear after authorization are not considered). Therefore, the purpose of the bachelor's work is to increase the effectiveness of protection of web applications from cross-site scripting, by developing software for vulnerability detection based on the analysis of the full map of the web application.

To achieve this goal, we developed effective algorithms for finding XSS-vulnerabilities, providing search in the "closed part" of the web application. The algorithms were implemented as software that has been successfully tested in "DosLab" organization, as evidenced by the act of implementation.

The competitiveness of the developed program is experimentally proved. The economic efficiency of the software implementation was assessed, the issues related to safety and health were also considered.

The results of the bachelor's work are published in 3 publications, one of which is included in the list of hack. The work was presented and awarded in the following competitions:

- awarded the status of finalist in the eighteenth all-Russian competition-conference among students and postgraduates on information security " SIBINFO-2018»;

- awarded the status of the winner in the competition of project works named after academician A. A. Bochvar.

СОДЕРЖАНИЕ

СПИСОК ИСПОЛЬЗУЕМЫХ СОКРАЩЕНИЙ

1) ИБ – информационная безопасность;
2) OWASP – открытый проект по информационной безопасности;
3) XSS – межсайтовый скриптинг;
4) ПО – программное обеспечение;
5) JS-код – JavaScript код;
6) СУБД – система управления базой данных;
7) БД – база данных;
8) ПЭВМ – пользователь персональной электронно-вычислительной машины;
9) ЭЛТ – электроннолучевая трубка;
10) ПК – персональный компьютер;
11) ОС – операционная система;
12) ОЗУ – оперативно запоминающее устройство.

ВВЕДЕНИЕ

Обеспечение информационной безопасности (ИБ) вычислительных систем является одной из приоритетных задач, решаемой любой организацией, в хозяйственной деятельности которой применяются алгоритмы сбора, обработки, хранения, передачи информации. Среди множества угроз ИБ существует категория негативных воздействий, которые могут провоцировать сами пользователи информационных ресурсов компании, подвергая опасности внутреннюю вычислительную сеть и в нее входящие устройства. Эти угрозы стали возможны благодаря широкому распространению сети Интернет.

При этом десять лет назад большинство веб-приложений были статическими и не имели интерактивных интерфейсов взаимодействия с пользователями. В них почти не было уязвимостей, которые могли быть использованы нарушителями. Поэтому многие веб-разработчики игнорировали вопросы безопасности веб-приложений. Однако на сегодняшний день существуют миллионы динамических веб-сайтов с множеством новых технологий, которые выполняются и используются в веб-браузерах. Данные технологии позволяют подключать к веб-приложениям различные модули, которые усиливают взаимодействие посетителей с веб-ресурсом (например, доски объявлений, формы обратной связи и т.д.).

Однако эти новшества имеют и отрицательную сторону. Динамические веб-сайты обеспечивают хорошую платформу нарушителям для внедрения вредоносного кода. С помощью внедренного кода нарушитель может получить доступ к данным авторизации пользователей и, выдавая себя за них, совершать противоправные действия как на локальных компьютерах пользователей, так и в сетевом оборудовании компании, меняя конфигурацию сети и программного обеспечения. Отсутствие должных мер по соблюдению правил и норм информационной безопасности приводит к появлению угроз, которые эксплуатируют уязвимости, связанные с внедрением кода. Тем самым подвергают компании большим финансовым и репутационным рискам.

Одним из таких видов компьютерных атак является межсайтовый скриптинг, в англоязычной литературе называемый – XSS (cross site scripting, x – используется в данной аббревиатуре для краткости, c – не используется, чтобы избежать путаницы с CSS) [1]. Межсайтовый скриптинг является одним из самых распространенных компьютерных атак, по версии OWASP (открытого проекта обеспечения безопасности веб-приложений) [2]. Об этом же свидетельствуют и результаты исследования компании Positive Technologies (Рисунок В.1) [3]. Компьютерная атака вида межсайтовый скриптинг

основана на эксплуатации XSS-уязвимостей веб-приложения. С помощью XSS-уязвимостей нарушитель внедряет вредоносный программный код в веб-страницу, отправляемую сервером клиенту. Данный код может позволить получить доступ к данным авторизации пользователя с целью совершения противоправных действий в отношении защищаемой информации. Часто уязвимость, позволяющую реализовать данный тип компьютерной атаки, также называют XSS.

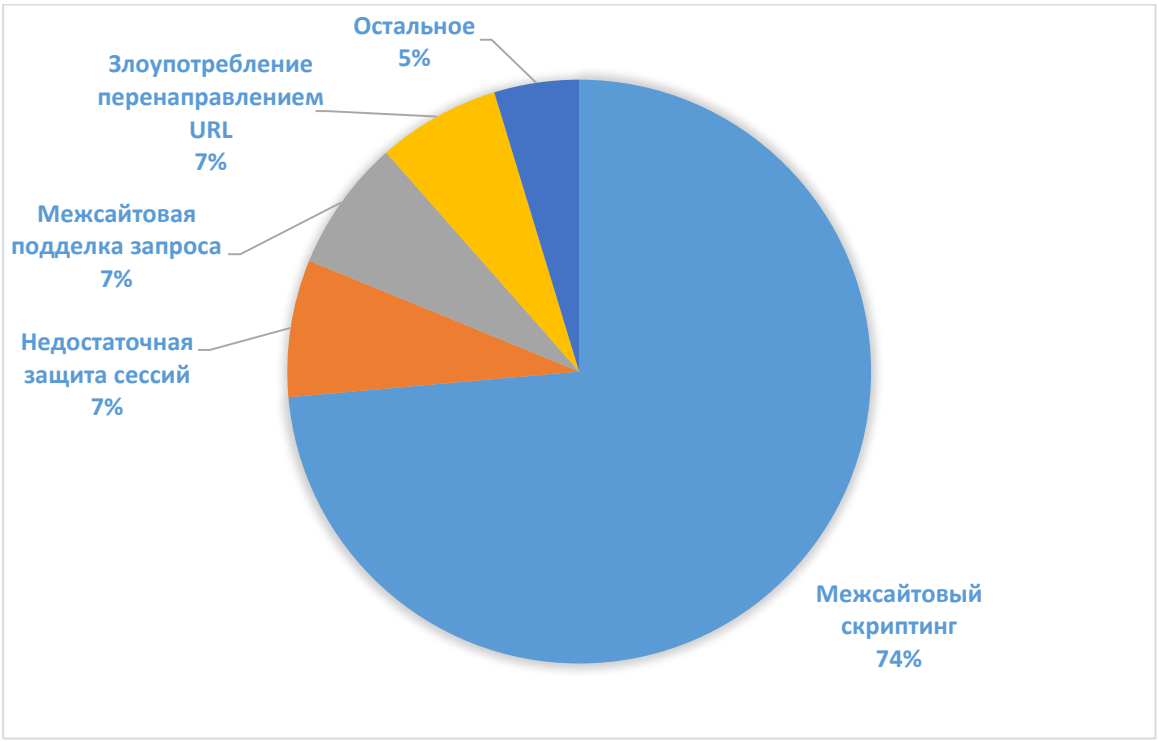


Рисунок В.1 – Результаты исследования Positive Technologies

Современные средства защиты информации предлагают широкий спектр алгоритмов и программного обеспечения, способных предотвращать различные угрозы информационной безопасности. Но каждое решение имеет свои недостатки. Недостатки программ для детектирования XSS связаны с особенностями построения веб-приложений. Большинство веб-приложений предусматривает авторизацию на веб-ресурсе для увеличения привилегий пользователя. То есть авторизованным пользователям будет доступен больший функционал веб-ресурса, чем неавторизованным. Существующие программные решения, предназначенные для обнаружения XSS-уязвимостей, осуществляют поиск только в открытой части веб-ресурса, это негативно сказывается на уровень ИБ веб-приложения.

Исходя из этого, целью работы явилось повышение эффективности защиты веб-приложения от межсайтового скриптинга, путем разработки программного обеспечения для детектирования уязвимостей на основе анализа полной карты веб-приложения.

Разработанное программное обеспечение (ПО) предоставляет пользователю информацию об уязвимостях исследуемого веб-приложения, выдает рекомендации по их устранению.

Для достижения поставленной цели были решены следующие задачи:

- исследовать предметную область и выявить особенности проведения XSS-атак;

- провести сравнительный анализ существующих решений, предназначенных для выявления XSS-уязвимостей, выявить их преимущества и недостатки;

- разработать эффективные алгоритмы поиска XSS, предусматривающие поиск в «закрытой» части веб-приложения и реализовать их в виде ПО;

- провести технико-экономическое обоснование, тестирование и апробацию предложенного решения и наметить пути его дальнейшего совершенствования.

Работа изложена на 82 страницах и состоит из введения, 3-х глав, заключения, списка использованной литературы, содержащего 33 наименований, и 5 приложений.

В работе имеется 25 рисунков и 12 таблиц.

В первой главе на основе анализа технической литературы по ИБ и Интернет-ресурсов посвященным методам и средствам защиты информации проведен анализ предметной области и имеющихся на сегодняшний день решений. Был сделан вывод о том, что XSS-уязвимости обладают рядом особенностей, которые не в должной мере учтены в предлагаемых на рынке решениях средств защиты, а именно, существующие решения, предназначенные для обнаружения XSS-уязвимостей, осуществляют поиск только в «открытой части» веб-ресурса, которая доступна авторизованным пользователям.

Вторая глава посвящена описанию разработки ПО для обнаружения XSS-уязвимостей на основе анализа полной карты веб-приложения. Описывается создание модели функциональных компонентов с использованием диаграмм на языке UML, этапы разработки и алгоритмы программных модулей проекта. Приводится руководство пользователя для созданного ПО.

В третьей главе описаны результаты практического применения созданной программы на основе функционального тестирования. Тестирование осуществляется на примере веб-приложений, содержащих те или иные XSS-уязвимости. Описывается экономический эффект от внедрения программного продукта, учитывающий затраты на разработку и внедрение, а также финансовые издержки от возможных угроз ИБ. Приводятся достоинства и недостатки созданного решения, а также перспективы развития и расширения функционала ПО.

Глава 1 Особенности защиты от XSS-атак на веб-приложения

1.1 Концепция и виды межсайтового скриптинга (XSS)

Благодаря повсеместному распространению сети Интернет все большее количество уязвимостей, угрожают миллиардам пользователей сети. Поэтому возникает необходимость обеспечения безопасной среды для Интернет-серфинга. Специалисты по информационной безопасности разделяют все веб-уязвимости на две большие категории. Первая категория включает защиту веб-приложений, которая состоит из пользовательских и серверных приложений. Вторая категория – это сетевая безопасность, которая включает в себя базы данных, программы, операционные системы и сети (Рисунок 1.1).

Как правило, нарушителя интересует уязвимости на уровне веб-сервера и прикладного программного обеспечения, которые часто эксплуатирует при проведении различных видов атак в частности XSS.

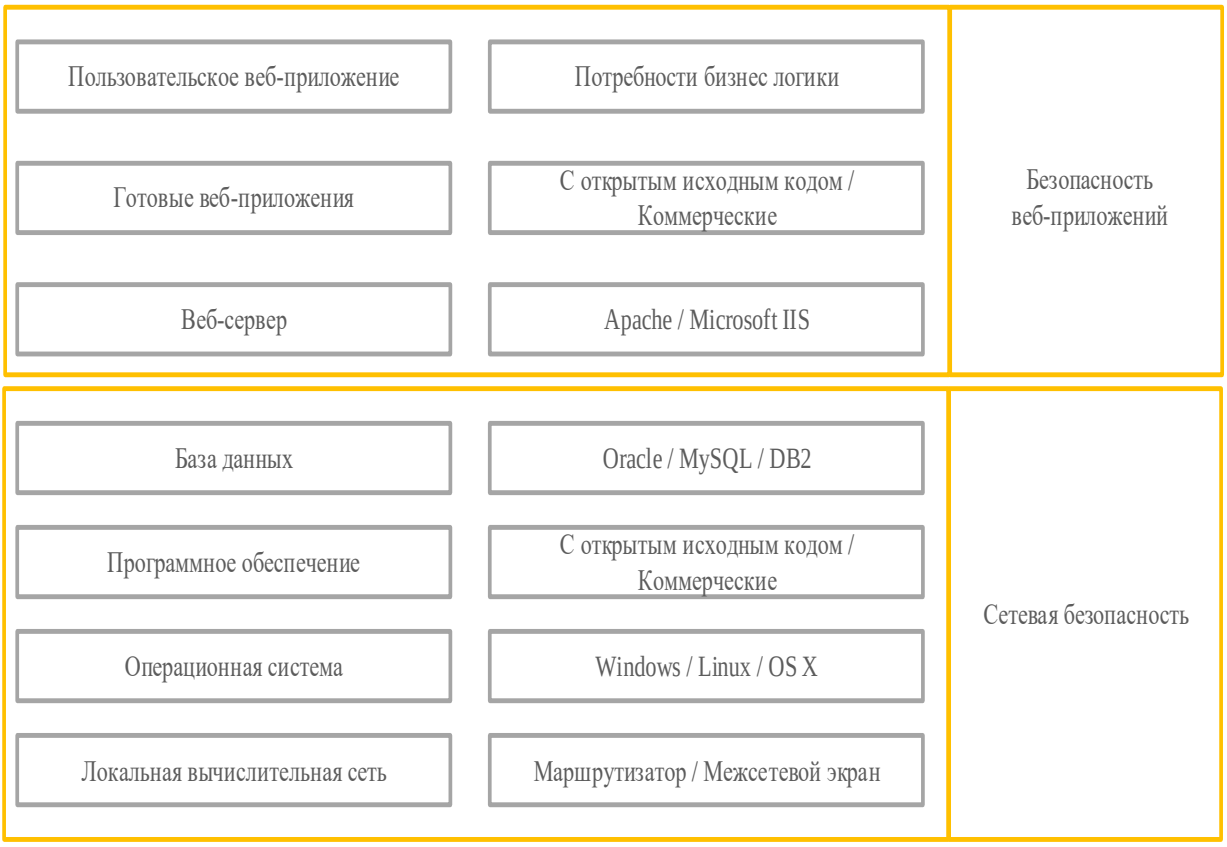


Рисунок 1.1 – Категории веб-уязвимостей

Безопасность сетевых технологий находится на более высоком уровне в отличие от безопасности веб-приложений, поэтому необходимо уделить больше внимания более слабому звену, чтобы избежать ситуаций, наносящих ущерб пользователям и принять

меры по сокращению потерь. Одним из опасных компьютерных атак на веб-приложения является межсайтовый скриптинг.

«Межсайтовый скриптинг» – это компьютерная атака, заключающаяся во внедрении вредоносного кода во входные параметры веб-страницы, отправляемые веб-браузеру пользователя. Внедряемый код похож на SQL-инъекцию и может быть использован различными способами [4]. XSS принято классифицировать по двум критериям: по вектору и способу воздействия (Рисунок 1.2).

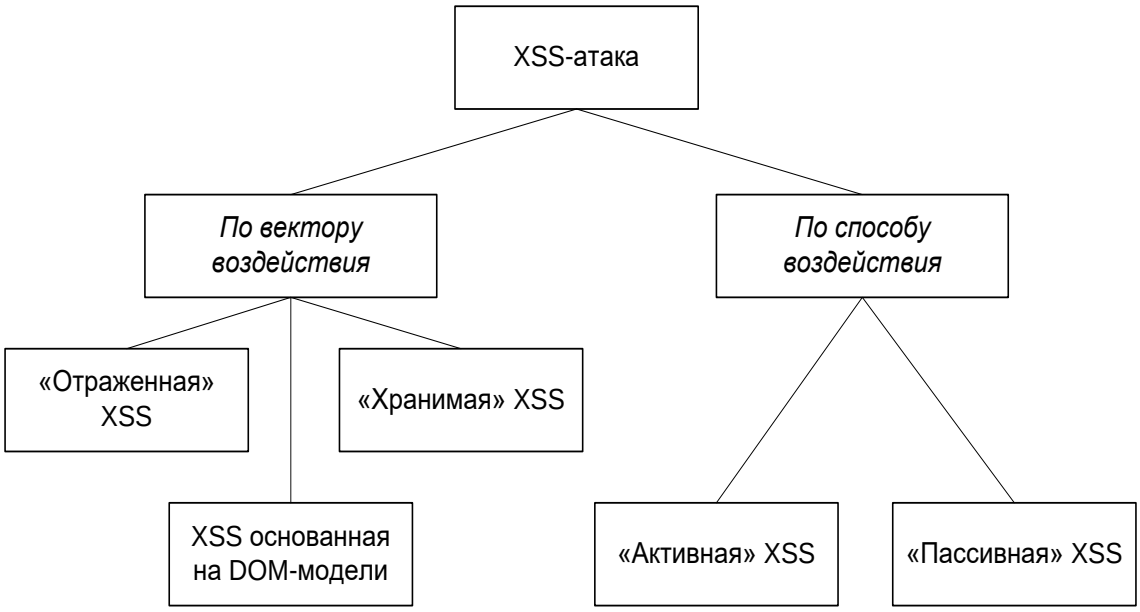


Рисунок 1.2 – Классификация XSS

По способу воздействия XSS-атака может быть [5]:

- «активная» XSS — не требующая каких-либо лишних действий со стороны пользователя с точки зрения функционала веб-приложения;

- «пассивная» XSS — срабатывающая при выполнении пользователем определённого действия («клик» или наведение указателя мыши и т.п.).

По вектору воздействия XSS-атаки подразделяются на следующие виды:

- «отраженная» (reflected) XSS;

- «хранимая» (stored) XSS;

- «XSS основанная на DOM-модели».

На Рисунке 1.3 показана общая схема реализации компьютерной атаки вида межсайтовый скриптинг.

Рисунок 1.3 – Общая схема проведения XSS атаки

«Хранимые» XSS-атаки возможны тогда, когда нарушителю удастся внедрить на сервер вредоносный код, выполняющийся в браузере пользователя каждый раз при обращении к оригинальной странице. Классическим примером этой уязвимости являются форумы, на которых разрешено оставлять комментарии в HTML-формате.

Отличительной особенностью «отраженной» XSS-атаки от «хранимой» является то, что компьютерная атака срабатывает тогда, когда пользователь переходит по ссылке нарушителя. Эти уязвимости появляются, когда данные, предоставленные веб-клиентом исполняются непосредственно серверными скриптами для синтаксического анализа и отображения страницы. Чаще всего эти уязвимости находятся в параметрах HTTP-запроса или в форме HTML из-за некорректной обработки.

XSS-уязвимости основанные на «DOM-модели» также, часто используются злоумышленниками для проведения атак. Прежде всего необходимо дать определение термину DOM. DOM (от англ. Document object model – «объектная модель документа») – объектная модель, используемая для XML/HTML-документов [6]. С помощью DOM можно генерировать древовидную структуру HTML документов. Поэтому с легкостью можно изменять или контролировать объектную модель документа. Этот вид XSS-атаки полностью отличается от хранимой и отраженной так как вредоносный код не внедряется в веб-страницу. XSS-уязвимость, основанная на объектной модели документа, возникает

непосредственно внутри JavaScript сценария. Это является серьезной уязвимостью, и обычная защита от XSS-атак не будет работать при проведении этой атаки.

1.2 Описание и схемы реализаций атак, использующих XSS-уязвимости

«Отраженная» XSS-атака является наиболее распространенной, и уязвимость, которую эксплуатирует данный вид компьютерной атаки, легка в детектировании. При «отраженной» XSS-атаке передача кода серверу и возврат его к пользователю осуществляется в рамках одного HTTP-запроса. Процесс проведения межсайтового скриптинга с использованием «отраженной» XSS-уязвимости сложен. Это связано с тем, что вредоносный код необходимо внедрить в URL адрес, затем URL с внедренным вредоносным кодом отправить пользователю, чтобы он перешел по ссылке для запуска вредоносного кода. Однако сложность реализации не является препятствием для нарушителей.

При проведении «базовой отраженной» XSS-атаки не преследуется цель кражи конфиденциальной информации. Суть атаки заключается в том, что при переходе на веб-сайт с «отраженной» XSS-уязвимостью, выводится предупреждающее окно. Это результат выполнения скриптового кода. Ссылка для перехода на веб-страницу с «базовой отраженной» XSS-уязвимостью выглядит примерно так:

```
http://site.ru/<script>alert (“XSS успешно выполнен”</script>).
```

Если нарушитель отправит данный URL своей жертве в таком виде, то скорее всего продвинутому пользователю веб-адрес покажется странным, и он не перейдет по данной ссылке. Поэтому одним из методов сокрытия вредоносной ссылки является использование кодирования RFC 1738 [7]. Как видно из фрагмента кода, приведенного ниже, после кодирования вредоносного URL скриптовый код становится трудно распознаваемым:

```
URL-адрес с XSS:
http://site.ru/<script>alert('XSS')<script>
URL-адрес с XSS после кодирования:
http:%3a%2f%2fsite.ru%2f%3cscript%3ealert(%27XSS%27)%3cscript%3
e
```

Нарушители используют «отраженную» XSS-уязвимость с другими компьютерными атаками, чтобы повысить эффект от эксплуатации уязвимости. В частности, появился новый вид компьютерной атаки, так называемый «кликджекинг» (от англ. Clickjacking – «угон клика»). Его суть заключается в том, что пользователь, совершая «клик» на легитимной странице, на самом деле производит переход по ссылке, сформированной злоумышленником [8].

Особенности «кликджекинга»:

- кража ссылки происходит, когда пользователь нажимает кнопку, сформированную нарушителем, либо переходит по ссылке;

- нет необходимости в инъецируемых скриптах, что делает атаку более простой в реализации;

- компьютерная атака основана на Dynamic HTML;

- для захвата ссылки злоумышленник всего лишь должен контролировать некоторую часть веб-сайта.

В свою очередь, благодаря «кликджекингу», появился другой вид компьютерной атаки, которая сочетает в себе JavaScript и тег Iframe. Специалисты по ИБ назвали эту компьютерную атаку «Cross-Frame Scripting (XFS)». Эта компьютерная атака, сочетает в себе комбинацию вредоносного JavaScript-кода (JS-код) с HTML тегом Iframe, который загружает легитимную страницу для кражи пользовательских данных. XFS-атака обычно бывает успешной только в сочетании с социальной инженерией.

Рассмотрим пример реализации XFS-атаки [9]. Нарушитель, применяя методы социальной инженерии, убеждает пользователя перейти на специально сформированную страницу. После перехода на страницу нарушителя в веб-браузер пользователя подгружаются вредоносный JavaScript-код (JS-код) и Iframe. Когда пользователь вводит учетные данные в Iframe, указывающий на легитимный сайт, вредоносный JS-код крадет нажатия клавиш.

Существует еще один вид XSS-атаки, эксплуатирующий «отраженную» XSS-уязвимость веб-приложения, так называемый «Redirection XSS». Это вид XSS-атаки, в котором вредоносный код подгружается с ненадежного источника и внедряется в различные скрипты веб-приложения. Атака «Redirection XSS» выглядит следующим образом:

- данные поступают в веб-ресурс с ненадежного источника в виде веб-запроса;

- полученные данные «вклиниваются» в динамический контент веб-приложения, который отправляется веб-пользователю без проверки на наличие вредоносного кода.

Вредоносный контент, отправляемый в веб-браузер пользователя, часто принимает вид JS-кода, но может также включать HTML, Flash или любой другой тип кода, который может быть обработан веб-браузером.

Компьютерные атаки, использующие «хранимые» XSS-уязвимости, являются, по сравнению с «отраженными» XSS, более опасными, так как они могут быть использованы на протяжении длительного времени. «Хранимые» XSS-атаки в основном используются для кражи сессионной информации (cookies или другой информации о сеансе). Также

возможно использовать данную атаку для перенаправления пользователя на веб-ресурс злоумышленника или выполнять другие вредоносные операции на ПК пользователя.

Рассмотрим наиболее опасные компьютерные атаки, эксплуатирующие «хранимые» XSS-уязвимости веб-приложения.

Компьютерная атака под названием «кража куков» (от англ. Stealing cookies). Как известно, cookies – это небольшой фрагмент данных, который хранится в веб-браузере пользователя и включает в себя пользовательские данные и много полезной информации, используемой для распознавания пользователя во время доступа к веб-приложению. Cookies генерируются на стороне веб-сервера и отправляется веб-браузеру пользователя для сохранения ее в локальном хранилище веб-браузера. При последующем обращении пользователя к веб-приложению, сохраненные cookies будут отправлены веб-серверу для распознавания пользователя и выдачи доступа к веб-ресурсу, основываясь на информации, записанной в cookies [10].

Иногда нарушители для совершения долгосрочной компьютерной атаки используют XSS-уязвимость для перенаправления пользователей на загрузку троянского коня. Специалисты по ИБ называют данную атаку «XSS Based Trojan Horse», в переводе с английского означает «XSS атака с внедрением троянского коня» [11].

При проведении данной атаки нарушителю необходимо, чтобы пользователь скачал троянскую программу. Для этого нарушитель использует «хранимую» XSS-уязвимость веб-приложения. В область веб-приложения с «хранимой» XSS-уязвимостью внедряется вредоносный код для скачивания троянской программы. Любой, кто посетит эту страницу, автоматически перейдет по ссылке с вредоносным JS-кодом. JS-код обработается браузером пользователя и в фоновом режиме загрузит оставшуюся часть вредоносного кода на компьютер пользователя. При этом жертва не заметит компьютерную атаку, так как используется Iframe для создания дочернего окна, в настройках которого прописан нулевой размер.

«XSS-worm» для усиления эффективности атаки, объединяет «хранимую» XSS-уязвимость и методику проведения компьютерной атаки с помощью «сетевого червя». Для успешного распространения «червя» необходимо компактно написать код, так как в веб-приложениях устанавливают определенные ограничения ввода данных пользователем. Для обхода ограничений нарушитель использует определенные методы [12]. Иногда, нарушитель вызывает вредоносный код извне если в веб-приложениях отсутствует ограничение прав пользователей. Существует еще один вид компьютерной атаки

эксплуатирующая «хранимую» XSS-уязвимость, так называемая «XSS tunneling» (в переводе с английского «XSS-туннелирование»).

Благодаря компьютерной атаке «XSS tunneling» нарушитель имеет возможность постоянно контролировать компьютер пользователя [13]. На Рисунке 1.4 представлена схема реализации «XSS-туннелирования».

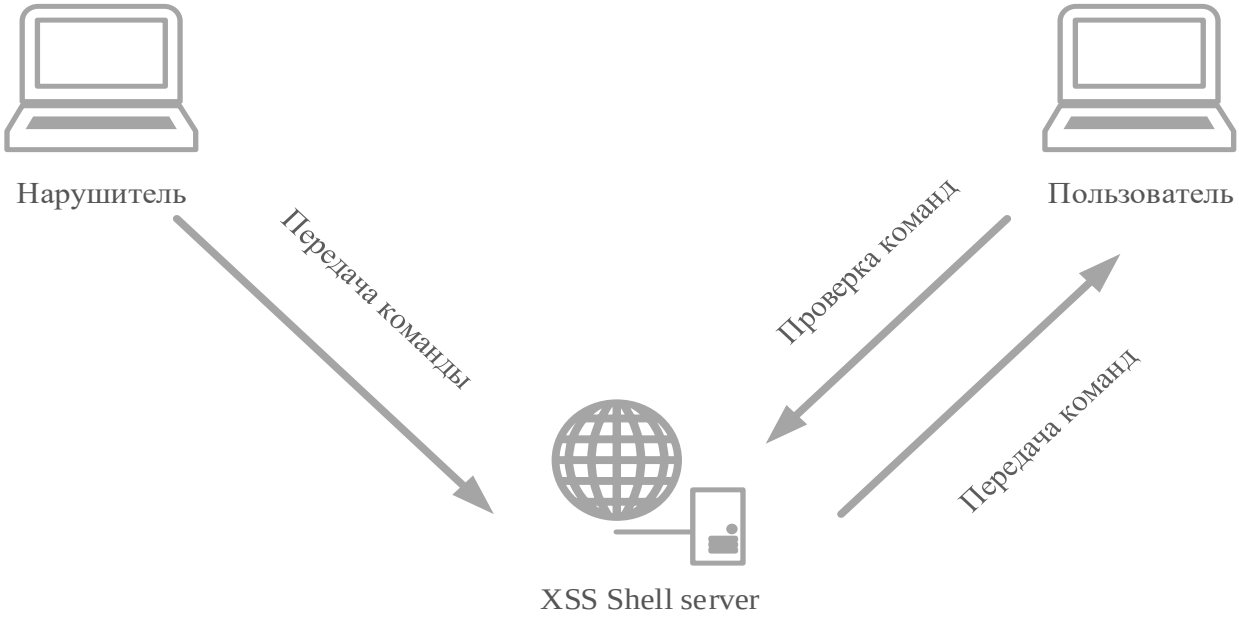


Рисунок 1.4 – Схема реализации «XSS-туннелирования»

Нарушитель через XSS-Shell сервер посылает команды, которые будут выполняться на компьютере жертвы. Таким образом XSS-туннелирование обеспечивает передачу данных HTTP трафика между нарушителем и пользователем.

Следующая компьютерная атака эксплуатирует XSS-уязвимость в DOM-модели, возникающей на стороне пользователя во время обработки данных внутри JavaScript сценария. Данный тип XSS-атаки получил такое название, поскольку реализуется через DOM (Document Object Model), не зависящий от платформы и языка программный интерфейс, позволяющий программам и сценариям получать доступ к содержимому HTML и XML-документов, а также изменять содержимое, структуру и оформление таких документов [14]. При некорректной фильтрации возможно модифицировать DOM атакуемого сайта и добиться выполнения JS-кода в контексте атакуемого сайта.

Рассмотрим более подробно механизм проведения «DOM XSS». Данный вид XSS-атаки использует доверенный, управляемый сервером скрипт, высылаемый пользователю веб-приложения. Например, JS-код осуществляющий проверку работоспособности формы перед отправкой серверу заполненных данных пользователем. Скрипт обрабатывает введенные данные, затем вставляет их обратно в веб-страницу (например, с

Dynamic HTML). Это дает возможность реализовать «DOM XSS» т.е. «вклинить» вредоносный код в сценарий JS-кода.

1.3 Способы защиты от межсайтового скриптинга

Компьютерные атаки на основе межсайтового скриптинга потенциально опасны для большинства веб-приложений и браузеров. Нарушители постоянно изобретают всё новые типы компьютерных атак, но следующие мероприятия смогут помочь защитить системы от этих компьютерных атак. Для защиты от XSS-атак применяются различные методы как на серверах, так и в браузерах клиентов.

Благодаря невнимательности и небрежности пользователей, нарушитель с легкостью может получить несанкционированный доступ к данным пользователя. Поэтому возникает необходимость повышения осведомленности пользователей об уязвимостях и способах защит от XSS-атак в процессе Интернет-серфинга. Основным способом защиты от XSS-атак является отключение всех языков сценариев. К сожалению, пользователи не могут отключить язык сценариев, потому, что он повлечет за собой отключение многих функций веб-сайтов. Однако, пользователь может предпринять несколько следующих советов для снижения рисков XSS-атак.

Пользователям рекомендуется осуществлять своевременное обновление браузеров до новой версии, так как большинство производителей браузеров стали внедрять средства защиты от различных видов XSS-атак в стандартный функционал браузера. Например, актуальная версия браузера Mozilla Firefox достаточно безопасна [15]. Firefox автоматически кодирует символы <и> (в последовательности %3C и %3E) в параметре document.URL в случае, когда URL не введен напрямую в адресную строку. Таким образом, данный браузер не уязвим к атакам, проводимым с использованием модели DOM. Для повышения безопасности работы браузера следует также установить дополнения (расширения), такие, как: NoScript, FlashBlock и панель инструментов Netcraft.

Также не стоит забывать про компьютерные атаки, связанные с социальной инженерией, следует серьезно и тщательно рассматривать все сообщения электронной почты, содержащие длинные, объемные и подозрительные ссылки. Не стоит переходить по этим ссылкам, даже если они ведут на известные сайты, которым вы доверяете. Во многих подобных сообщениях применяются различные хитрости для того, чтобы убедить Вас пройти по ссылке. В некоторых сообщениях содержатся предложения об улучшении вашего материального состояния и независимости, другие сообщения содержат

предупреждения о том, что вы можете потерять свою учетную запись (действительную) на веб-сайте, если "немедленно не подтвердите информацию о вашем имени пользователя и пароле". Необходимо серьезно подумать перед переходом по таким ссылкам. Очевидно, что точно такие же меры предосторожности следует предпринимать по отношению к сообщениям в системах онлайн-сообщений и социальных сетях.

В случае с кодом веб-приложения выполняющимся на веб-сервере необходимо предусмотреть экранирование любых входных данных, для этого применяются алгоритмы очистки входных данных от JS-кода, который может оказаться вредоносным. Большинство платформ веб-разработки реализуют данный функционал в стандартной комплектации поставляемых инструментов.

Для проверки полей при отправке формы следует применять алгоритм фильтрации, основанный на белых списках, проверяющих корректность ввода данных пользователями. Под входными данными понимается любая информация, поступающая на вход веб-приложения заголовки HTTP-запросов, файлы cookies, строки запросов, скрытые поля, синтаксис данных в веб-формах. Так же необходимой мерой для защиты от XSS-атак является явное указание разработчиком веб-приложения кодировки страницы до определения пользовательских полей ввода. В случае если кодировка не указана злоумышленник может встроить вредоносный код в кодировке UTF-7, обойдя тем самым фильтрацию символов угловых скобок и кавычек в формах ввода [16].

Файлы cookies, содержащие данные авторизации пользователя и историю заполнения форм необходимо делать недоступным для чтения с помощью языка сценариев JavaScript данное действие необходимо выполнять, как на веб-страницах, отображаемых в браузере клиента, так и в коде внутренних функций серверной части веб-приложения.

Также необходимо использовать заголовки, которые обеспечивают контроль политик безопасности контента (CSP – Content Security Policy), что позволяет в явном виде указать список источников, с которых будет разрешено осуществлять загрузку JavaScript-кода [17]. Даже в случае успешного внедрения XSS в веб-страницу, выполнение кода будет заблокировано, так как его источник не будет указан в соответствующем списке. Указание браузеру о применении данной политики посылается с помощью HTTP-запроса.

1.4 Анализ существующих решений на рынке

Современные средства защиты информации предлагают широкий спектр алгоритмов и программного обеспечения, способных предотвращать различные угрозы

ИБ. Большинство средств обнаружения уязвимостей громоздкие, обладают избыточным функционалом, который может замедлить работу вычислительной системы и увеличить потребление ресурсов. А простые решения «заточенные» только на поиск XSS-уязвимостей также обладают недостатками. Недостатки программ для детектирования XSS связаны с особенностями построения веб-приложений. Большинство веб-приложений предусматривает авторизацию на веб-ресурсе для увеличения привилегий пользователя. То есть авторизованным пользователям будет доступен больший функционал веб-ресурса, чем неавторизованным.

Рассмотрим наиболее популярные решения, направленные на обнаружение XSS-уязвимостей.

«XSpider» (MAX-Patrol) обеспечивает комплексный мониторинг информационной безопасности предприятия, решая такие задачи, как контроль эффективности процессов ИБ, политик безопасности и их изменения, инвентаризация и оценка защищенности [18]. Осуществляет поиск «отраженных», «хранимых» и «DOM XSS». Помимо этого, «XSpider» имеет функцию формирования отчета и выдачи рекомендаций по найденным уязвимостям. Данный программный комплекс очень громоздкий, обладает избыточным функционалом и сильно нагружает ЛВС организации. Также не осуществляет поиск XSS-уязвимостей в закрытой части веб-приложения.

«Nemesida Scanner» предназначен для выявления уязвимостей в веб-приложениях, таких как SQL injection, XSS, LFI/RFI, XXE, Open-redirect, поиска компонентов с известными уязвимостями и критичных данных в открытом доступе, выявления недостатков конфигурации веб-приложения, сканирования портов, и т.д. Nemesida Scanner предоставляется в виде консольной версии для популярных Linux-дистрибутивов [19]. «Nemesida Scanner» содержит большую базу современных векторов атак и полезных нагрузок (payload), не требователен к аппаратному обеспечению. Осуществляет поиск уязвимостей только в открытой части веб-приложения.

«Acuentix Online Web Security Scanner» обеспечивает автоматический контроль безопасности web-приложений и позволяет выявить уязвимые места в защите web-сайта [20]. Программа автоматически обнаруживает следующие уязвимости: XSS, SQL injection, GHDB (Google hacking Database). Данное ПО осуществляет поиск уязвимостей созданием карты сайта. Проверяет все web-страницы с элементами для ввода данных, моделируя ввод данных с использованием всех возможных комбинаций и анализируя полученные результаты. Недостатком является ее высокая стоимость и отсутствие осуществления поиска уязвимостей в закрытой части веб-приложения.

«XSSF» направлен на выявление потенциальных угроз, связанных с XSS-уязвимостями [21]. Основная функция данного ПО заключается в создании коммуникационного канала (или XSSF-туннеля) с браузером жертвы для выполнения различных компьютерных атак, каждая из которых оформлена отдельным модулем. Существует большое количество модулей, например, модуль похищения файлов, модуль вызова через программные средства связи (например, Skype), сканирование сети и другие.

«XSSF» предлагает множество методов для проведения XSS-атак, однако не предоставляет большого количества методов для нахождения XSS-уязвимостей. Для работы с «XSSF» предварительно необходимо ознакомиться с инструментами, которые входят в ее состав и изучить документацию, так как используемые алгоритмы могут нанести вред приложениям. В «XSSF» невозможно осуществить поиск XSS-уязвимостей, связанных с кражей куков. Также операции программы осуществляются в открытой части веб-приложения, так как программа «заточена» на проведение XSS-атак, а не на обнаружение XSS-уязвимостей.

«XenoTix» – утилита для тестирования на проникновение в систему, обнаруживающая возможности использования межсайтового скриптинга [22]. Осуществляет нагрузочное тестирование с использованием различных видов XSS-нагрузок. Имеет возможность как ручного, так и автоматического запуска алгоритмов, нагружающих веб-приложение. Несмотря на простоту, «XenoTix» обладает несколькими ограничениями. Каждый раз при загрузке файла или атаке через обратную оболочку браузер пользователя блокирует действия программы и выводит сообщения, требующие подтверждения пользователя. Это связано с тем, что приложение использует апплеты, требующие само подписания. Также в данном продукте отсутствует функция авторизации на тестируемом веб-ресурсе, что делает невозможным поиск уязвимостей, связанных с кражей куков.

«Wapiti» является консольным сканером веб-приложений, который в своей основе несет принцип черного ящика [23]. «Wapiti» входит в состав утилит дистрибутива Kali Linux. Можно скачать исходники с SourceForge и использовать на любом дистрибутиве, основанном на ядре Linux. «Wapiti» поддерживает GET и POST HTTP методы запросов. В плане организации тестирования – анализируются не исходники приложения, а ответы сервера на запросы с измененными параметрами. Утилита сначала анализирует структуру сайта, ищет доступные сценарии, анализирует параметры, а затем включает свой фаззер. Касательно XSS-уязвимостей, «Wapiti» поддерживает поиск как «хранимых», так и «отраженные» уязвимостей, но не «XSS основанные на DOM-модели».

Недостатком является то, что «Wapiti» не дает возможности предварительно авторизоваться в системе, и может проверять только функционал, доступный для незарегистрированного пользователя. «Wapiti» написан на скриптовом языке Python, который является однопоточным языком. Таким образом, невозможно параллельное выполнение задач с целью ускорения работы приложения.

Было проведено сравнение по основным характеристикам существующих программ (XSSF, XenoTix, Wapiti, XSpider (MAX-Patrol), Nemesida Scanner, Acunetix Online Web Security Scanner) для обнаружения XSS-уязвимостей. Результаты сравнения программ по основным параметрам, осуществляющие поиск XSS-уязвимостей представлены в Таблице 1.1.

Таблица 1.1 – Результаты сравнения

Характеристики	XSSF	XenoTix	Wapiti	XSpider (MAX-Patrol)	Nemesida Scanner	Acunetix Online Web Security Scanner
Авторизация в системе	–	–	–	–	–	–
Поиск «отраженных» XSS	+	+	+	+	+	+
Поиск «хранимых» XSS	+	+	+	+	+	+
Поиск «DOM» XSS	+	–	+	–	+	+
Наличие ГИП	+	+	–	+	–	+
Формирование отчета	+	+	–	+	–	+
Выдача рекомендаций	–	–	–	+	–	+

1.5 Анализ и выбор средств реализации проекта

Выбор средств и методов реализации проекта будет зависеть от нескольких факторов:

- потребности и возможности пользователей;
- реализуемые функции программного обеспечения.

Решение, разрабатываемое в рамках дипломной работы должно быть доступно широкому кругу пользователей с разным уровнем подготовки и владения веб-технологиями. Данное требование будет влиять на вид ПО, которое будет разрабатываться, это может быть:

- тонкий клиент, предоставляющий пользователю возможность осуществлять поиск уязвимостей с помощью веб-интерфейса;

- толстый клиент.

Рассмотрим достоинства и недостатки каждого из методов, а также доступные способы реализации решения на той или иной платформе.

При клиент-серверном подходе к организации ПО взаимодействие между пользователями и сервером (содержащим бизнес-логику) происходит через браузер, поэтому проблема кроссплатформенности в этом случае не стоит, так как ее берет на себя разработчик браузера. Пользователю достаточно иметь устройство с установленным браузером, чтобы начать работать в системе. В этом заключается основное преимущество по сравнению с толстыми клиентами. При этом, попадая в систему с разных устройств, пользователь будет видеть одинаковый (или практически одинаковый, но содержащий идентичный функционал интерфейс) интерфейс, что облегчит освоение приложения.

Работа через веб-приложение не предполагает дополнительной установки ПО со стороны пользователя, за исключением случаем, когда функционал браузера достаточен в полной мере для работы. В противном случае приходится либо использовать возможности дополнительных платформ, либо устанавливать плагины и агенты. Появляется возможность реализовывать практически полностью идентичный классическому UI-функционал (в частности drag-and-drop, вывод контекстного меню по клику правой кнопки мыши и т.д.) [24]. Часть вычислений может выполняться на машине пользователя без отправки на сервер, и в целом лучше балансируется нагрузка на ресурсы клиента и сервера. Веб-приложения могут быть гибкими, предлагая клиенту тот или иной функционал за ответствующую плату.

За плюсы веб-приложений приходится платить следующими ограничениями:

- ввиду унификации приложение в большей или меньшей степени будет испытывать трудности при интеграции в существующие сервисы;

- необходимо будет учитывать финансовые в временные затраты на развертывание приложения и его техническую поддержку скрипты и код приложения должны загружаться в рабочие папки браузера при подключении, что замедляет работу с системой;

- при запуске скрипта на стороне клиента теряется производительность;

- разворачивание клиента в браузере приводит к ограничениям использования системных ресурсов и устройств;

- могут возникнуть сложности с интеграцией с МФУ, сканерами, сторонним софтом для редактирования документов разных форматов.

К недостаткам толстого клиента можно отнести необходимость в установке специализированного ПО каждому пользователю, данное требование может оказаться не критичным для большинства организаций.

Преимуществами подобного рода ПО можно считать то, что пользователи работают в привычном интерфейсе, свойственном конкретной оболочке операционной системы, приложение легко интегрируется с приложениями и системным программным обеспечением уже установленным и используемым в рамках компании. Толстые клиенты снимают часть вычислительной нагрузки, ложащейся на серверы. К тому же зачастую не все тонкие клиенты позволяют работать с документами без использования интернета, что в некоторых случаях может быть критично. В связи с вышеизложенным, ПО будет реализовано в виде толстого клиента.

Сегодня широко применяются языки программирования высокого уровня, такие как: Delphi, C++, Pascal, C#. А также СУБД на базе SQL, QBE. Подбор языка программирования основывается на выборе типа применяемой БД.

Система управления базами данных (СУБД) — совокупность программных и лингвистических средств общего или специального назначения, обеспечивающих управление созданием и использованием баз данных.

Основные функции СУБД:

- управление данными во внешней памяти (на дисках);

- управление данными в оперативной памяти с использованием дискового кэша;

- журнализация изменений, резервное копирование и восстановление базы данных после сбоев;

- поддержка языков БД (язык определения данных, язык манипулирования данными).

Обычно современная СУБД содержит следующие компоненты:

- ядро, которое отвечает за управление данными во внешней и оперативной памяти и журнализацию;

- процессор языка базы данных, обеспечивающий оптимизацию запросов на извлечение и изменение данных и создание, как правило, машинно-независимого исполняемого внутреннего кода;

- подсистему поддержки времени исполнения, которая интерпретирует программы манипуляции данными, создающие пользовательский интерфейс с СУБД;

- сервисные программы (внешние утилиты), обеспечивающие ряд дополнительных возможностей по обслуживанию информационной системы.

Одним из главных достоинств, встроенных БД можно назвать то, что есть возможность хранить логически сгруппированные данные в различных таблицах и отражать связи между ними, объединяя единую базу. Для задания связи таблицы имеют поля с идентичными именами или хотя бы с идентичными форматами данных. Связь между таблицами указывает отношения между равными значениями в этих полях. При использовании такой организации данных можно уменьшить избыточность хранимых данных, упростить их ввод и реализацию запросов и отчетов.

Выбор системы управления баз данных (СУБД) включает в себя довольно трудную многопараметрическую задачу и становится одним из самых значимых этапов в процессе разработки приложений БД. Определенный программный продукт должен соответствовать как текущим, так и будущим потребностям компании. Для рассмотрения выделено несколько СУБД: SQL Anywhere, InterBase, MS Access.

SQL Anywhere — многофункциональный программный пакет, который обеспечивает управление и синхронизацию данных предприятия, позволяет быстро разрабатывать и внедрять решения в области баз данных для мобильных вычислений и рабочих групп [25]. В состав SQL Anywhere включены надежные механизмы синхронизации корпоративного класса, что обеспечит надежную и защищенную интеграцию локального, работающего с данными приложения и корпоративной БД. Это единственная технология синхронизации, обеспечивающая синхронизацию данных между популярными мобильными устройствами и базами данных Sybase, Microsoft, Oracle, MySQL и IBM на предприятии.

InterBase — это система управления реляционными базами данных, поставляемая корпорацией BORLAND, для построения приложений с архитектурой клиент-сервер произвольного масштаба: от сетевой среды небольшой рабочей группы с сервером под управлением Novell NetWare или Windows NT на базе IBM PC до информационных систем крупного предприятия на базе серверов IBM, Hewlett-Packard, SUN [26].

Microsoft Office Access или просто Microsoft Access — реляционная система управления базами данных (СУБД) корпорации Microsoft [34]. Входит в состав пакета Microsoft Office. Имеет широкий спектр функций, включая связанные запросы, связь с

внешними таблицами и базами данных. Благодаря встроенному языку VBA, в самом Access можно писать приложения, работающие с базами данных.

Access, при работе с базой данных, иначе взаимодействует с жёстким (или гибким) диском, нежели другие программы.

В других программах файл-документ при открытии полностью загружается в оперативную память и новая редакция этого файла (изменённый файл) целиком записывается на диск только при нажатии кнопки «сохранить».

В Access новая редакция содержимого изменённой ячейки таблицы записывается на диск (сохраняется) сразу, как только курсор клавиатуры будет помещён в другую ячейку (или новая редакция изменённой записи записывается на диск сразу, как только курсор клавиатуры будет поставлен в другую запись (строку)). Таким образом, при сбое электропитания потери данных будут минимальными - только в той записи, которая редактировалась на момент сбоя.

Целостность данных в Access обеспечивается также за счёт механизма транзакций.

В работе будут использоваться следующие технологии:

- язык программирования высокого уровня Borland Delphi 7;
- структурированный язык запросов SQL и СУБД MS Access.

Для обращения к web-приложениям будут использованы стандартные средства среды разработки, такие как: компонент «TWebBrowse» (использующий функциональность Microsoft Internet Explorer, который поставляется вместе с Windows и не требует установки и настройки дополнительных программных средств, что делает его более выгодным на фоне его аналогов) и библиотеки MSHTML.

Для представления способов работы и возможностей эксплуатации пользователями разрабатываемого ПО будет представлена в виде Use-Case диаграммы – которая представляет собой наглядное отображение возможных действий пользователей в системе. Основными элементами диаграммы являются актеры (пользователи), которые обозначаются, как показано на Рисунке 1.5.

Рисунок 1.5 – Изображение пользователя на диаграмме прецедентов использования

Следующим элементом является прецедент, действие, обозначаемое символом овала, с текстом, описывающим само действие (Рисунок 1.6).

Рисунок 1.6 – Изображение действия на диаграмме прецедентов использования

Стрелки на подобных диаграммах также могут выглядеть по-разному и показывать различные отношения между пользователями и действиями, например, с их помощью можно отображать свойства наследования, отношения обобщения, включения, расширения, зависимости действий друг от друга (Рисунок 1.7).

Рисунок 1.7 – Обозначение отношений включения и расширения

Ключевое слово <<include>> означает, что какое-либо действие включает в себя другое, обязательное действие, в свою очередь <<extend>> говорит о том, что действие не обязательно к исполнению, однако требует предварительного выполнения другого действия, от которого исходит стрелка на диаграмме.

Так как язык Delphi – объектно-ориентированный, то для описания структуры компонентов программы будет использоваться диаграмма классов языка UML.

С помощью диаграммы классов, для создания которой используется язык UML, можно показать какие атрибуты и методы содержит класс, а также то, в каких отношениях с другими классами он выступает. При проектировании классов принято, что все его атрибуты и методы отображающие внутреннюю структуру скрыты (недоступны для программистов). Отношения, в которые могут вступать классы друг с другом, влияют на работу методов класса и содержимое его атрибутов. Если необходимо, чтобы объекты одного класса вели себя при вызове методов подобно объектам другого класса, то необходимо создать отношение наследования. При этом один из классов становится родителем (базовым классом), а другой наследником (дочерним классом). На диаграмме классов приняты следующие обозначения (см. Таблица 1.2).

Таблица 1.2 – Обозначения зависимостей на диаграмме классов

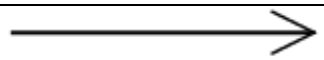
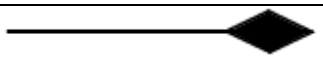
Обозначение	Название отношения
	Ассоциация
	Наследование (от дочернего класса – к родительскому)
	Агрегация
	Композиция

Диаграмма деятельности представляет собой пошаговый процесс характеризующий ту или иную активность, которая может происходить в системе. Диаграммы деятельности могут использоваться для представления одного и того же процесса с разных точек зрения.

1.6 Вывод по 1 главе

Межсайтовый скриптинг является одним из самых распространенных компьютерных атак, по версии OWASP. И для предотвращения компьютерных атак подобного рода, разработано большое количество средств обнаружения XSS-уязвимостей. Но большинство из них громоздкие, обладают избыточным функционалом, который может замедлить работу вычислительной системы и увеличить потребление ресурсов. А простые решения «заточенные» только на поиск XSS-уязвимостей также обладают недостатками. Анализ существующих решений для поиска XSS-уязвимостей показал, что программы осуществляют поиск только в открытой части веб-приложения, которая доступна авторизованным пользователям, тем самым снижая уровень информационной безопасности веб-приложения.

Поэтому целью бакалаврской работы явилось повышение эффективности защиты веб-приложения от межсайтового скриптинга, путем разработки программного обеспечения для детектирования уязвимостей на основе анализа полной карты веб-ресурсов. Разрабатываемое программное обеспечение (ПО) должно предоставлять пользователю информацию об уязвимостях исследуемого веб-приложения и выдавать рекомендации по их устранению. Для достижения поставленной цели необходимо решить следующие задачи:

- исследовать предметную область и выявить особенности проведения XSS-атак;
- провести сравнительный анализ существующих решений, предназначенных для выявления XSS-уязвимостей, выявить их преимущества и недостатки;
- разработать эффективные алгоритмы поиска XSS, предусматривающие поиск в «закрытой» части веб-приложения и реализовать их в виде ПО;

- провести технико-экономическое обоснование, тестирование и апробацию предложенного решения и наметить пути его дальнейшего совершенствования.

Глава 2 Обнаружение XSS-уязвимостей на основе анализа полной карты веб-приложения

2.1 Разработка алгоритмов обнаружения XSS-уязвимостей

Для достижения поставленной цели была использована методика, основанная на последовательном применении наиболее эффективных алгоритмов для обнаружения различных типов XSS-уязвимостей [27]. Разработанное ПО получило название «XSS-Finder».

Функции, реализуемые разрабатываемым программным обеспечением:

- детектирование всех видов XSS-уязвимостей («отраженная XSS», «хранимая XSS» и «XSS основанная на DOM-модели»);
- предварительная авторизация в веб-приложении и хранение кука;
- составление списка всех внутренних URL веб-приложения;
- создание отчета о найденных уязвимостях;
- формирование рекомендаций по найденным уязвимостям.

С помощью диаграммы потоков данных на Рисунке 2.1 описано движение информации в разрабатываемом программном обеспечении.

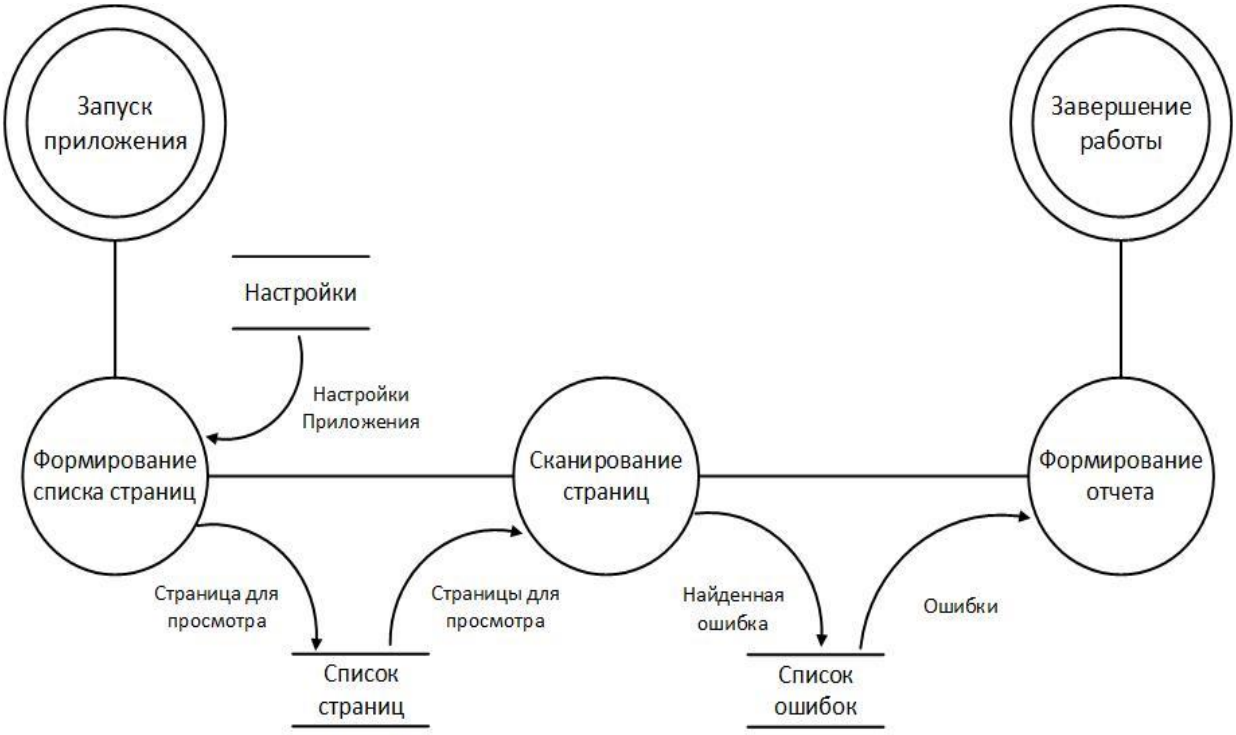


Рисунок 2.1 – DF-диаграмма проектного решения

Алгоритмы обнаружения для каждого вида XSS отличаются друг от друга это связано с особенностями уязвимостей. На Рисунке 2.2 представлена блок-схема,

иллюстрирующая алгоритм поиска «отраженных» XSS. Так как данный вид уязвимостей проявляется только на стороне пользователя при отправке форм, алгоритм провоцирует их отправку методом POST, включая в отправляемые элементы значения и получая ответ в виде HTML сообщения. Перед отправкой инъекции с целью экономии времени и ресурсов анализируется JavaScript-код формы отправки в котором возможно предприняты какие-либо меры защиты (фильтрация специальных символов и т.д.). Затем на основе результатов анализа из базы данных ПО подбирается соответствующая инъекция для отправки.

Если веб-браузер успешно выполняет действие, предусмотренное внедренной XSS-инъекцией, то страница помечается как содержащая потенциальную угрозу соответствующего типа, иначе она помечается как безопасная и не добавляется в итоговый список уязвимостей.

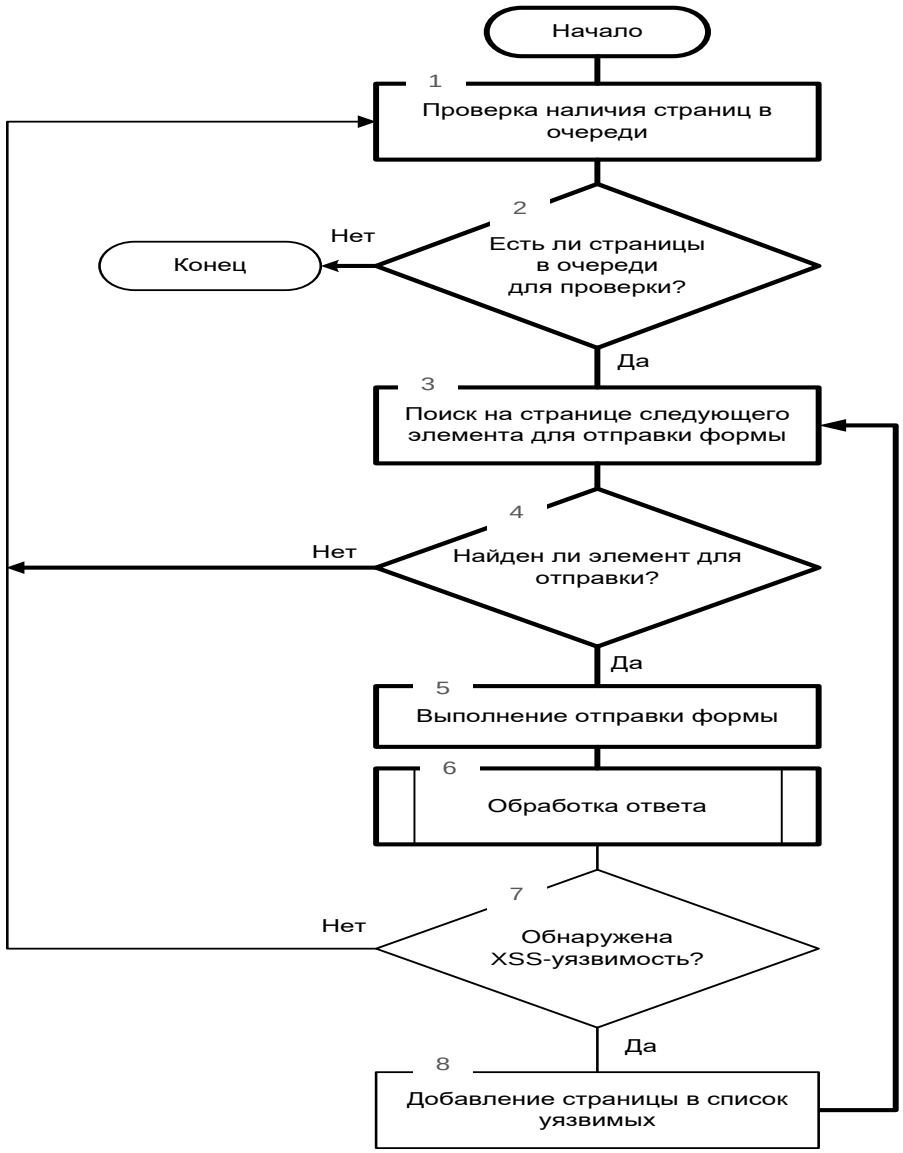


Рисунок 2.2 – Блок-схема алгоритма поиска «отраженных» XSS-уязвимостей

Для поиска XSS-уязвимостей, основанных на использовании объектной модели документа – DOM, используется алгоритм, блок-схема которого показана на Рисунке 2.3. В ходе выполнения алгоритма осуществляется анализ кода страницы на наличие скриптов, «спрятанных» в HTML тегах. После нахождения содержимого всех имеющихся на странице скриптов, в найденных данных осуществляется поиск фрагментов кодов, осуществляющих вызов методов, основанных на объектной модели документа, таких как:

- запись «чистого» HTML;
- прямая модификация модели документа (в том числе события Dynamic HTML);
- прямое выполнение скриптов.

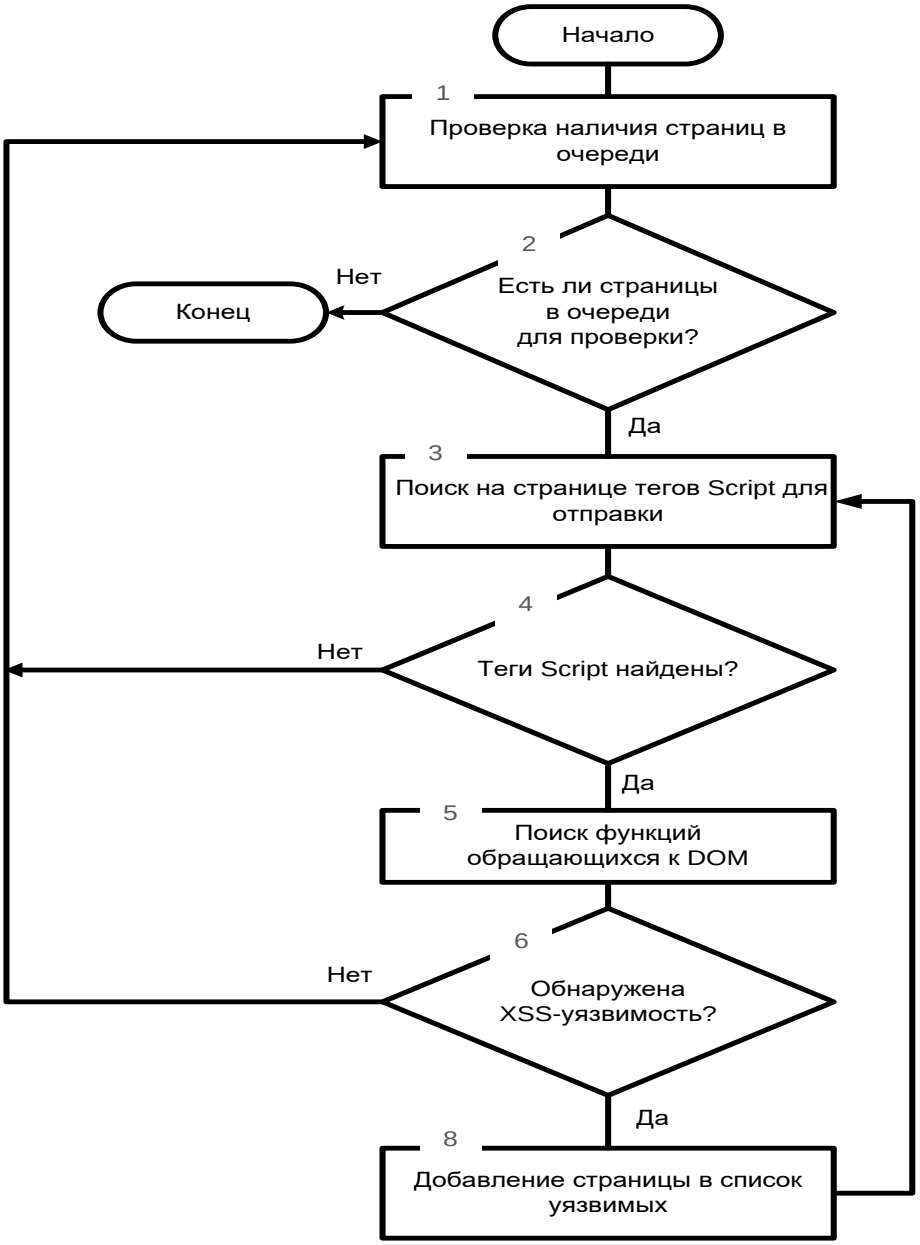


Рисунок 2.3 – Блок-схема алгоритма поиска «XSS-уязвимостей, эксплуатирующих DOM»

Для реализации поиска «хранимых» уязвимостей используется алгоритм, блок схема которого показана на Рисунке 2.4. Работа данного алгоритма имеет свои особенности, так как, в отличие от «отраженных», «хранимые» XSS-уязвимости являются следствием сохранения скрипта в базе данных на сервере. Поиск подобных уязвимостей должен осуществляться с помощью предварительного POST-запроса, чтобы не позволить вредоносному коду внести изменения в базу данных. Алгоритм во многом схож с алгоритмом поиска «отраженных» XSS-уязвимостей. Однако, поскольку данная уязвимость в отличие от «отраженной» XSS проявляется не на стороне пользователя, то необходимо производить отправку инъекции на сервер и ждать ответа.

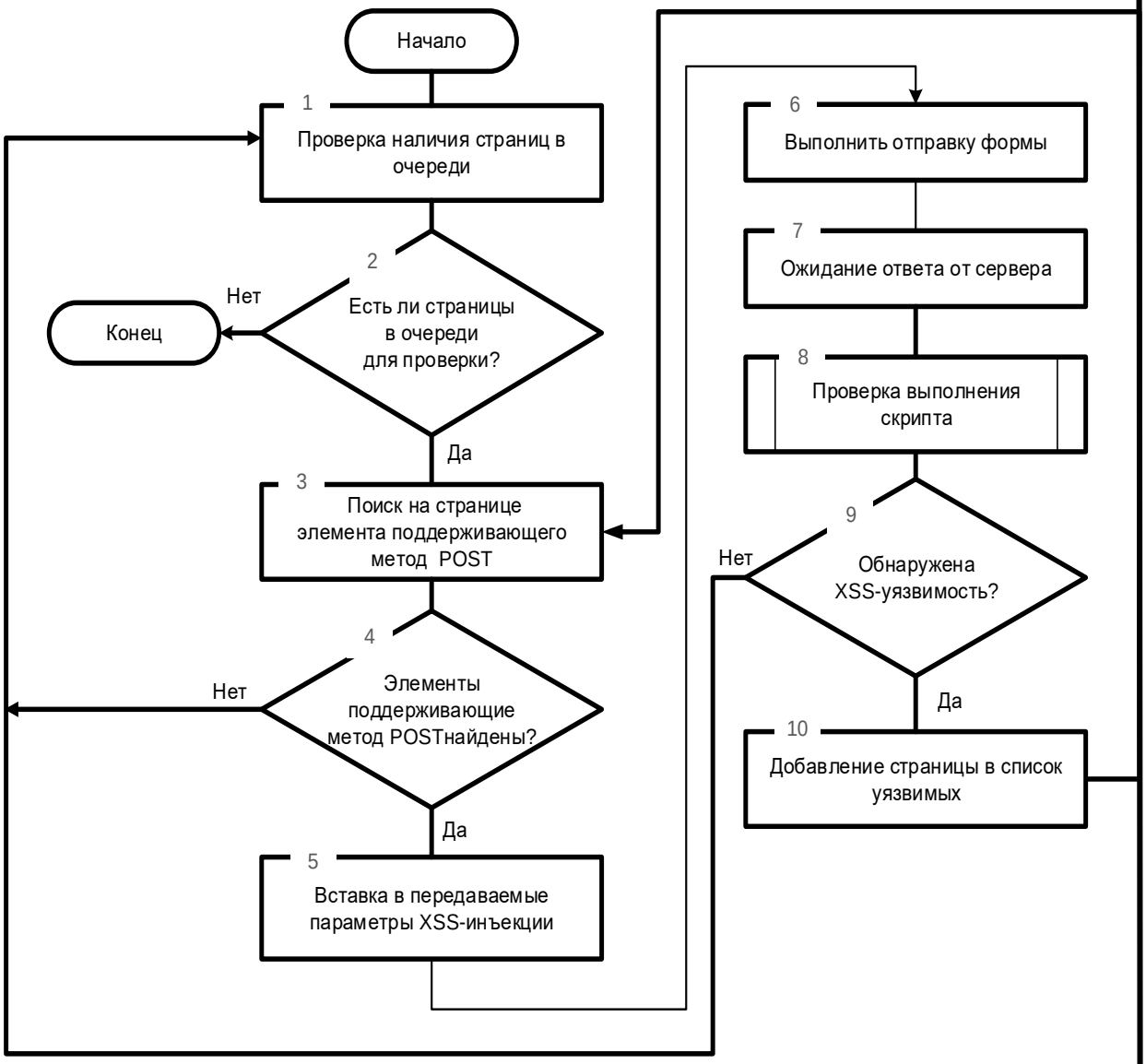


Рисунок 2.4 – Блок схема алгоритма поиска «хранимых» XSS-уязвимостей

В случае успешного внедрения скрипта (инъекции), веб-страница с введенной XSS-инъекцией сохраняется вместе с прочей информацией о найденной уязвимости в базе

данных. При выполнении скрипта продолжение поиска на данной странице не осуществляется, поскольку при отправке формы будет осуществляться выполнение скрипта, находящегося в базе данных веб-приложения. Поэтому данную проверку необходимо запускать повторно после устранения уязвимости. Укрупненная блок-схема разработанной программы показана на Рисунке 2.5.

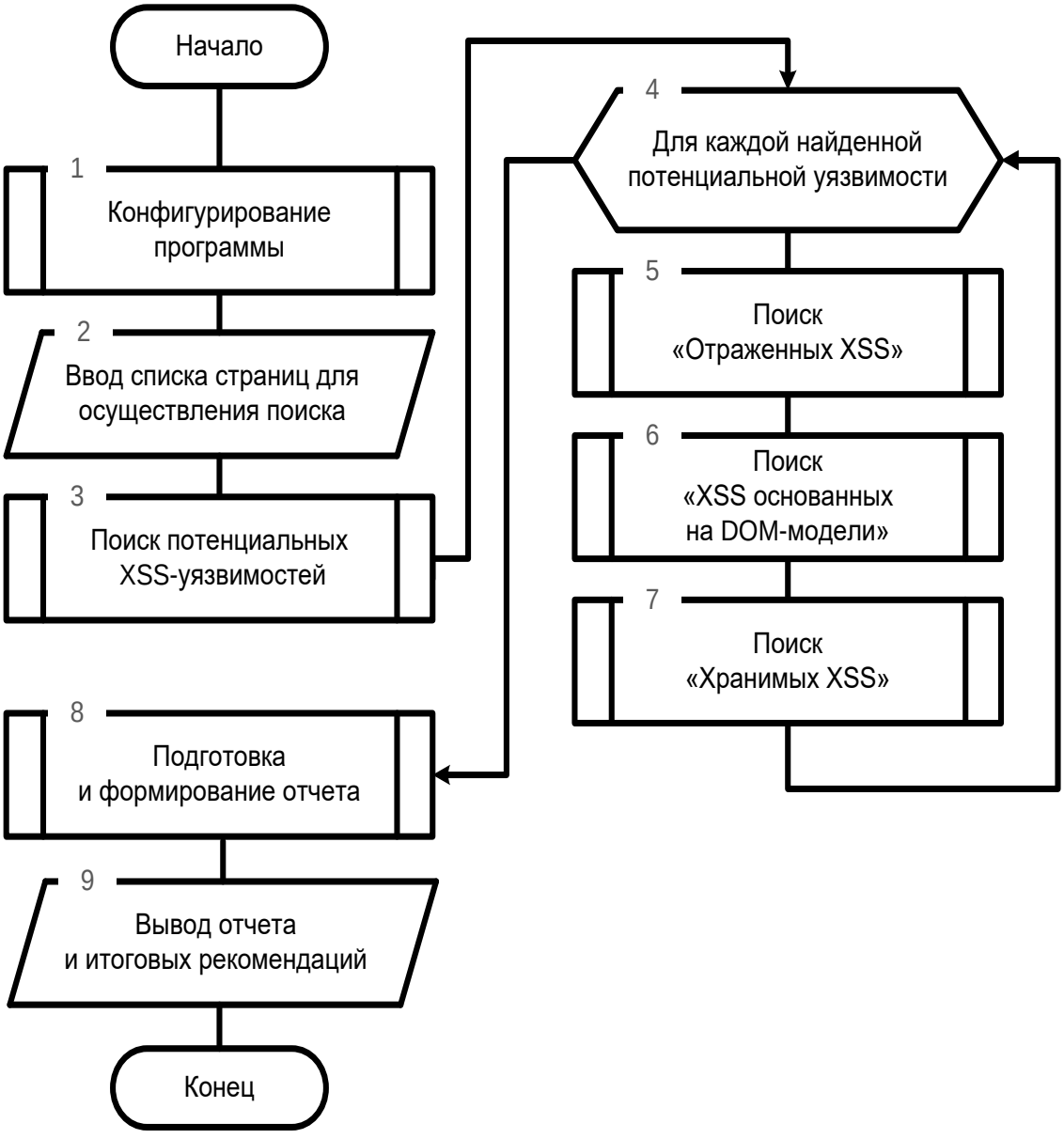


Рисунок 2.5 – Блок-схема разработанной программы детектирования XSS-уязвимостей

Конфигурирование программы включает следующие этапы:

- создание задания для выполнения;
- выбор и добавление допустимых расширений (файлы для поиска);
- авторизация на тестируемом веб-приложении.

2.2 Описание разработки программных компонентов

Для хранения исходных данных и результатов работы модулей системы программное обеспечение использует базу данных в формате MS Access. Основные таблицы базы данных описаны в Таблице 2.1.

Таблица 2.1 – Основные таблицы базы данных

Название	Описание
Problem	Уязвимость (справочник уязвимостей)
Task	Задание
TaskState	Состояние задания
Page	Страница веб-сайта
PageCheck	Результаты проверки страницы веб-сайта
TaskProblem	Список уязвимостей, которые будут проверяться в текущем задании
ProblemType	Тип уязвимости

На Рисунке 2.6 представлена структура базы данных разработанного программного обеспечения.

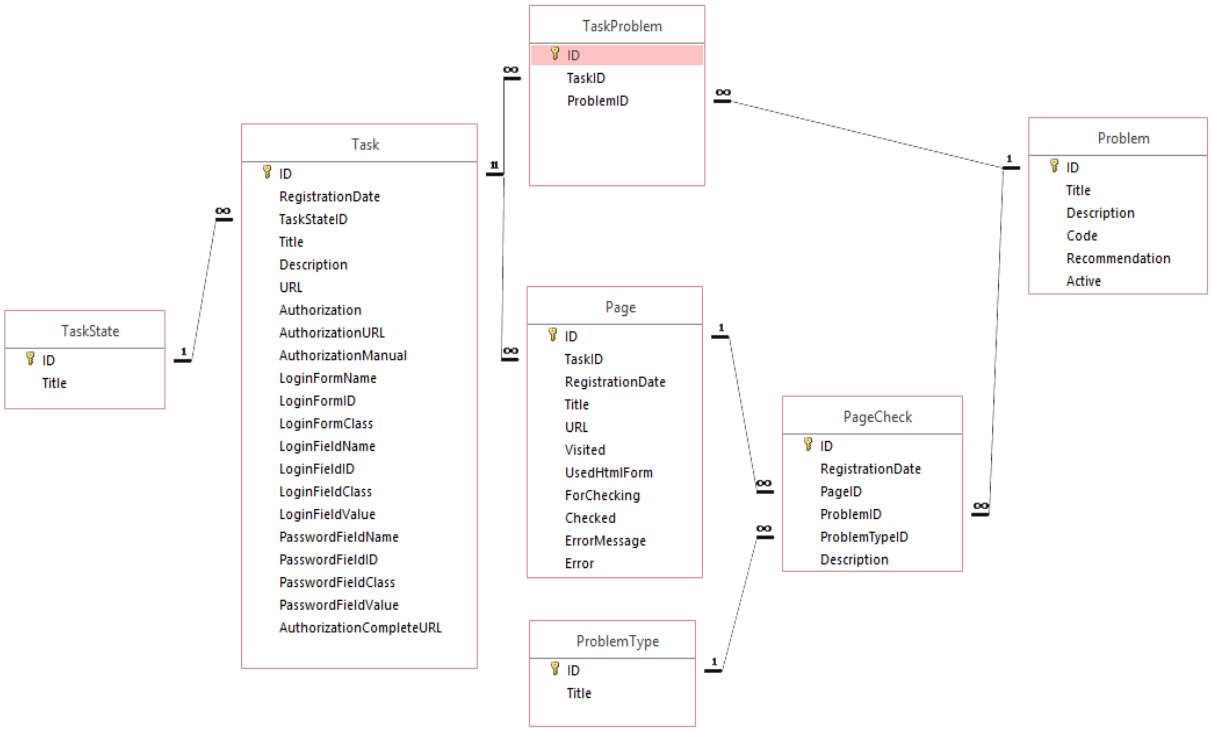


Рисунок 2.6 – Структура базы данных проектного решения

В Таблице 2.2 представлены название и описание основных структур классов.

Таблица 2.2 – Структуры классов

Название	Описание
TWebBrowser	Класс, реализующий механизм доступа к функциональности Microsoft Internet Explorer. Встроенный класс Delphi.
TDatabaseObject	Класс, реализующий базовые функции для обращения к базе данных. Реализация содержится в файле «uDatabaseObject.pas».
TSiteNavigator	Является базовым классом для компонентов, реализующих механизм навигации по содержимому web-приложений. Реализует внутри себя возможность подключения к объектам TWebBrowser. Также наследует функции от класса «TDatabaseObject». Реализация содержится в файле «uSiteNavigator.pas».
TSiteLogin	Является наследником класса «TSiteNavigator». Реализует механизм автоматической авторизации. Реализация содержится в файле «uSiteLogin.pas».
TSiteMapCreator	Является наследником класса «TSiteNavigator». Реализует механизм создания карты сайта. Реализация содержится в файле «uSiteMapCreator.pas».
TSiteReflectedXssFinder	Является наследником класса «TSiteNavigator». Реализует механизм поиска отраженных уязвимостей на основе созданной карты сайта. Реализация содержится в файле «uSiteXssFinder.pas».
TSiteStoredXssFinder	Является наследником класса «TSiteNavigator». Реализует механизм поиска хранимых уязвимостей на основе созданной карты сайта. Реализация содержится в файле «uSiteXssFinder.pas».

«TSiteMapCreator» класс, реализующий построение карты сайта. Работает следующим образом: выполняется загрузка базового адреса, после загрузки страницы автоматически выбираются все ссылки, которые имеются на странице и добавляется в базу данных. Базовый адрес также записывается в базу данных и для него вычисляется ряд показателей (наличие форм ввода на странице, будет ли использоваться данная страница в ходе проверок, и т.д.) После этого, устанавливается «флаг посещения» для базового

адреса, и данная процедура выполняется рекурсивно для каждой полученной ссылки, хранящейся в БД в статусе «еще не посещено».

«TSiteReflectedXssFinder» класс, реализующий поиск «отраженных» уязвимостей. Поиск уязвимостей реализуется следующим образом: выполняется последовательный перебор страниц веб-сайта, у которых имеются GET-параметры. Для URL-адреса текущей (выбранной) страницы выполняется генерация вариантов встраивания XSS-инъекций в URL-адрес (метод «InjectXssInUrl»). После этого, результаты работы метода «InjectXssInUrl» (т.к. результат работы метода представляет собой список ссылок) помещаются в очередь «URLQueue», из которой они последовательно извлекаются, и по ним выполняется переход. После загрузки страницы, выполняется поиск XSS-сигнатур внутри страницы.

«TSiteStoredXssFinder» класс, реализующий поиск «хранимых» уязвимостей. Внутри данного класса в два этапа реализуется механизм поиска уязвимостей:

- встраивание XSS: для встраивания выбираются только те страницы, внутри которых имеются формы ввода данных. Когда «TSiteStoredXssFinder» выполняет переход на страницу с формой, при первой загрузке, он загружает список полей формы в специальную очередь «PageFieldsQueue» и заполняет очередь XSS-сигнатур «XSSQueue». И пока в очереди «XSSQueue» имеются элементы, последовательно извлекает поля из очереди «PageFieldsQueue» и выполняет ввод данных в соответствующее поле и выполняет отправку формы. Когда очередь «PageFieldsQueue» не содержит элементов, удаляется первый элемент очереди «XSSQueue» и так продолжается до тех пор, пока количество элементов в «XSSQueue» не станет равным нулю. После осуществляется переход к следующей странице и операции встраивания повторяются;

- поиск XSS: класс «TSiteStoredXssFinder» выполняет последовательный обход всей карты сайта (т.к. внесенные данные могут отображаться на множестве различных страницах сайта) и выполняется поиск XSS-сигнатур в HTML-коде страницы. Также поиск выполняется после каждой отправки формы на этапе встраивания (это связано с тем, что обычно, на сайтах после сохранения данных выполняется redirect на страницу просмотре внесенных данных).

На Рисунке 2.7 представлена диаграмма классов, реализованных в разработанном программном обеспечении.

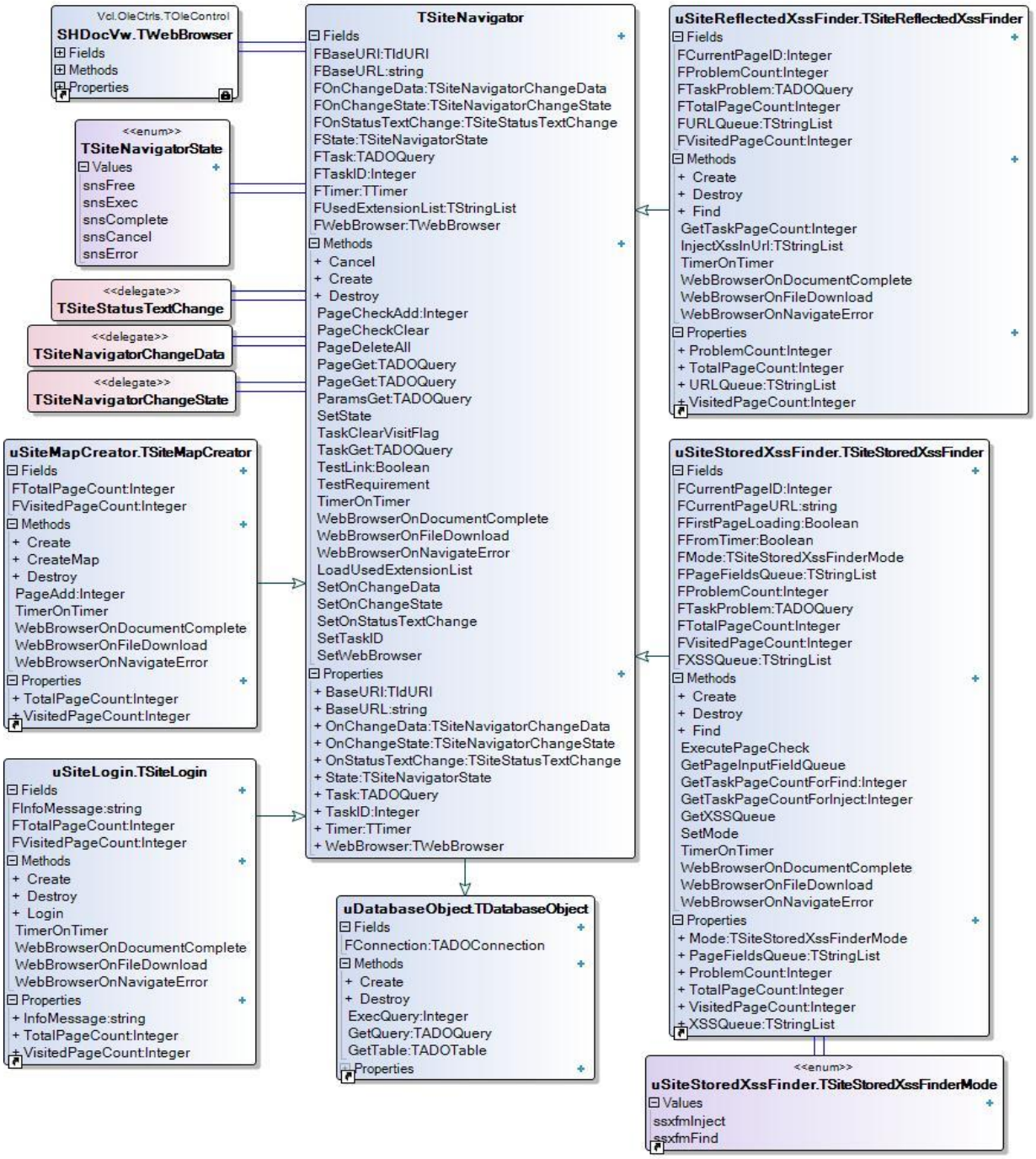


Рисунок 2.7 – Диаграмма классов

2.3 Описание функционала разрабатываемого программного обеспечения

В основные возможности программного обеспечения входят:

- детектирование всех видов XSS-уязвимостей;
- предварительная авторизация в веб-приложении;
- справочник XSS-уязвимостей;
- создание отчета и формирование рекомендаций по найденным уязвимостям.

Детектирование всех видов XSS-уязвимостей

В разработанной программе поиск уязвимостей осуществляется путем создания задания, выделены следующие задания: поиск «хранимых», «отраженных» и «уязвимости, основанные на DOM-модели», также создание карты веб-ресурса и полный цикл сканирования.

Предварительная авторизация в веб-приложении и хранение кука

Авторизация предназначена для осуществления поиска уязвимостей в «закрытой» части веб-приложения, которая доступна авторизованным пользователям. Авторизация может быть выполнена в ручном так и, в автоматическом режиме. Ручная авторизация – это когда пользователю предоставляется возможность выполнить на начальном этапе вход на сайт. Автоматическая авторизация – это когда программа самостоятельно выполняет вход (хранит сессионные данные до конца тестирования), используя заранее определенные параметры задания (Рисунок 2.8).

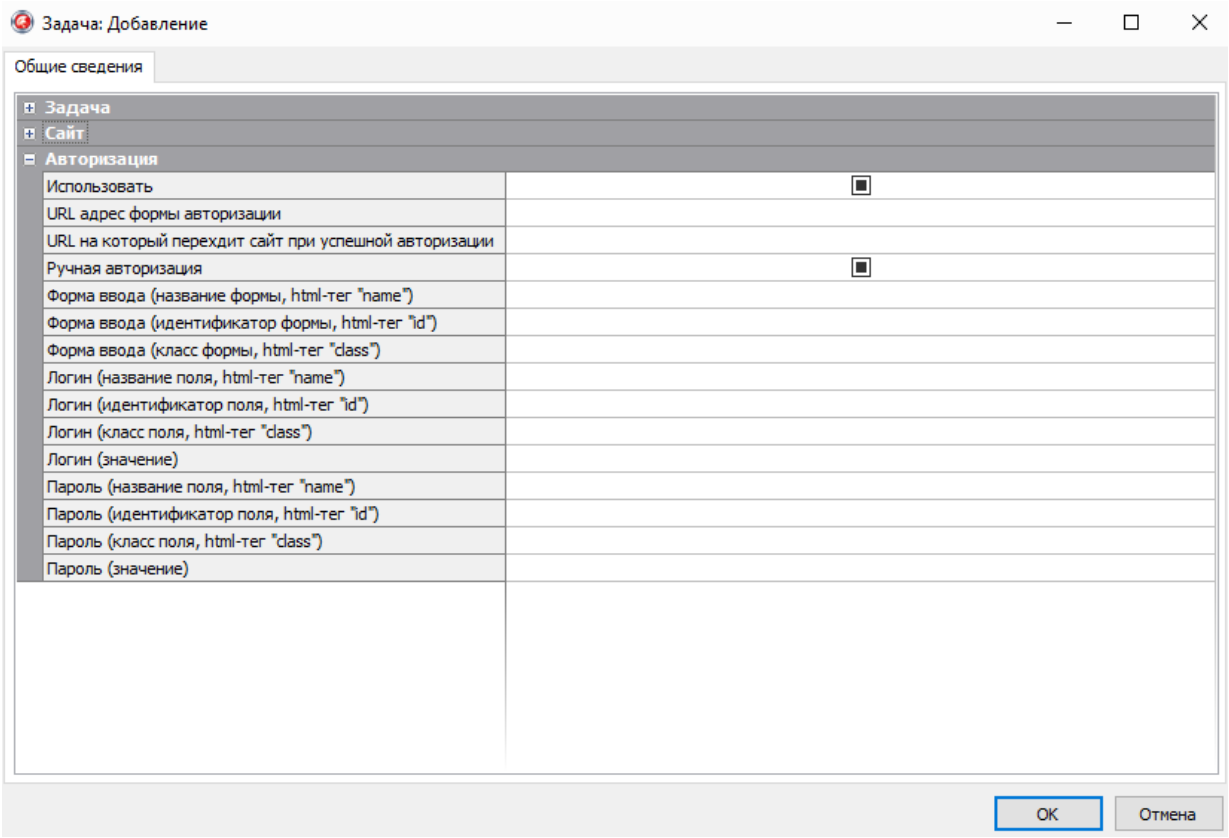


Рисунок 2.8 – Интерфейс формы авторизации

Справочник XSS-уязвимостей

Справочник XSS-уязвимостей представляет собой список с XSS-инъекциями, которые можно редактировать. Также реализована возможность добавления новых инъекций, это связано с тем что нарушители изобретают все новые виды атак. В данном

интерфейсе возможно выбрать те инъекции, которые необходимы для пользователя в процессе тестирования с целью экономия времени. На Рисунке 2.9 представлен интерфейс справочника с уязвимостями.

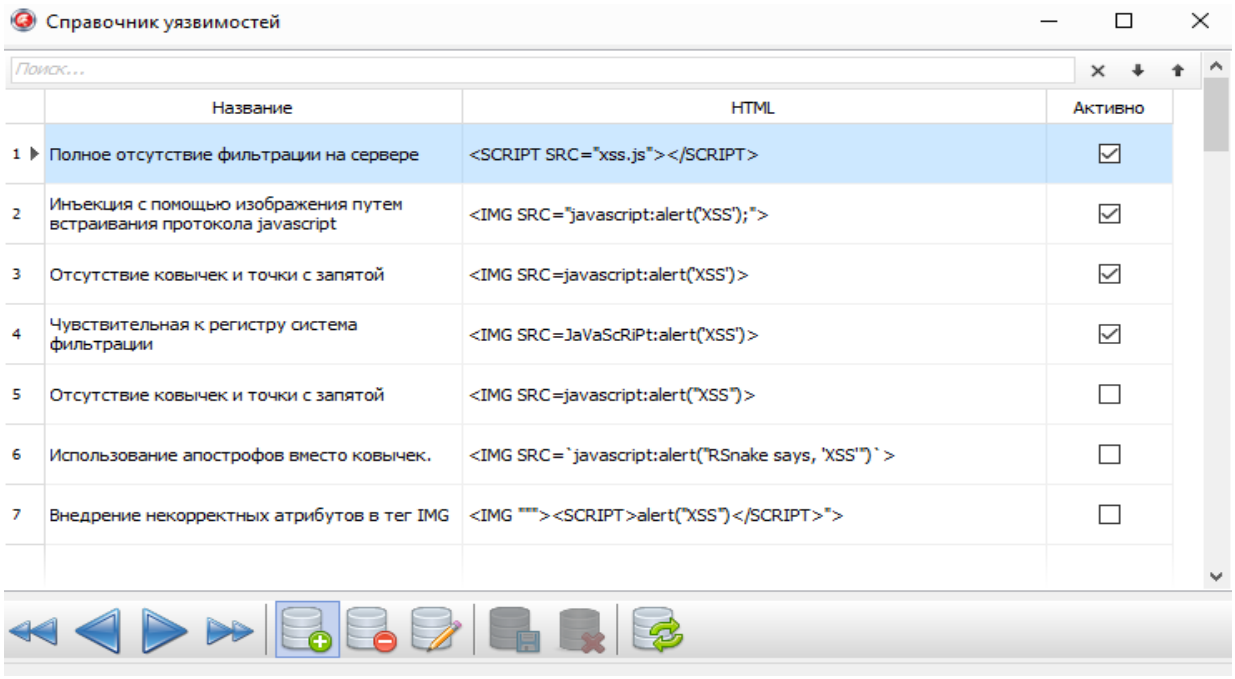


Рисунок 2.9 – Интерфейс справочника с уязвимостями

Создание отчета и формирование рекомендаций по найденным уязвимостям

Отчет предоставляет веб-разработчику полную информацию о тестируемом веб-приложении. На основе результатов отчета автоматически формируются рекомендации по устранению найденных уязвимостей. Рекомендации формируются при наличии записей во внутренней структуре где хранятся найденные уязвимости. На Рисунке 2.10. представлен фрагмент сформированного отчета. Полный отчет приведен в Приложении Г. Отличительной особенностью формирования рекомендации в разработанном программном обеспечении по сравнению с аналогичными решениями, является выдача конкретных рекомендаций (подробно описывается найденный вид уязвимости с примерами устранения и приводится ссылка на литературу).

Отчет по выполнению задачи поиска XSS уязвимостей

Результаты по каждой странице

№	Дата регистрации	Название	URL	Просмот.	Для проверки	Проверен.
1	22.01.2018 15:04:46		http://localhost:7081/basic/web/index.php?r=request%2Fview&id=33	+	+	
2	22.01.2018 15:04:47		http://localhost:7081/basic/web/index.php?r=request%2Fupdate&id=33	+	+	
3	22.01.2018 15:04:47		http://localhost:7081/basic/web/index.php?r=request%2Fdelete&id=33	+	+	
4	22.01.2018 15:04:46	ID	http://localhost:7081/basic/web/index.php?r=request%2Findex&sort=id	+	+	
		Обнаруженные уязвимости				
		1	Инъекция с помощью изображения путем встраивания протокола javascript			
		2	Полное отсутствие фильтрации на сервере			
		Всего обнаружено на странице: 2				
5	22.01.2018 15:04:41	Авторасписание	http://localhost:7081/	+	+	
6	22.01.2018 15:04:41	Выход (admin)	http://localhost:7081/basic/web/index.php?r=site%2Flogout	+	+	
7	22.01.2018 15:04:41	Главная	http://localhost:7081/basic/web/	+	+	
8	22.01.2018 15:04:46	Добавить заявку	http://localhost:7081/basic/web/index.php?r=request%2Fcreate	+	+	
9	22.01.2018 15:04:41	Заявки	http://localhost:7081/basic/web/index.php?r=request%2Findex	+	+	
10	22.01.2018 15:04:54	Заявки	http://localhost:7081/basic/web/index.php?r=request%2Findex&sort=-id	+	+	

Рисунок 2.10 – Фрагмент сформированного отчета

Выдача рекомендаций является одним из преимуществ разработанной программы. Так как она значительно упрощает процесс тестирования веб-ресурса разработчиком.

2.4 Руководство пользователя

Разработанное ПО при наличии NetFramework 4.0 может функционировать со следующими ОС: Windows XP, Windows 7, Windows 8, Windows 10. Перед началом использования разработанной программы пользователю необходимо настроить параметры безопасности зоны Интернет. Для этого необходимо зайти в «Панель управления» - «Свойства обозревателя» - «Безопасность» - «Другой …» после этого, найти параметр «Включить фильтр XSS» и установить его свойство в значение «отключить». Данную операцию необходимо выполнить для каждой зоны (Рисунок 2.11).

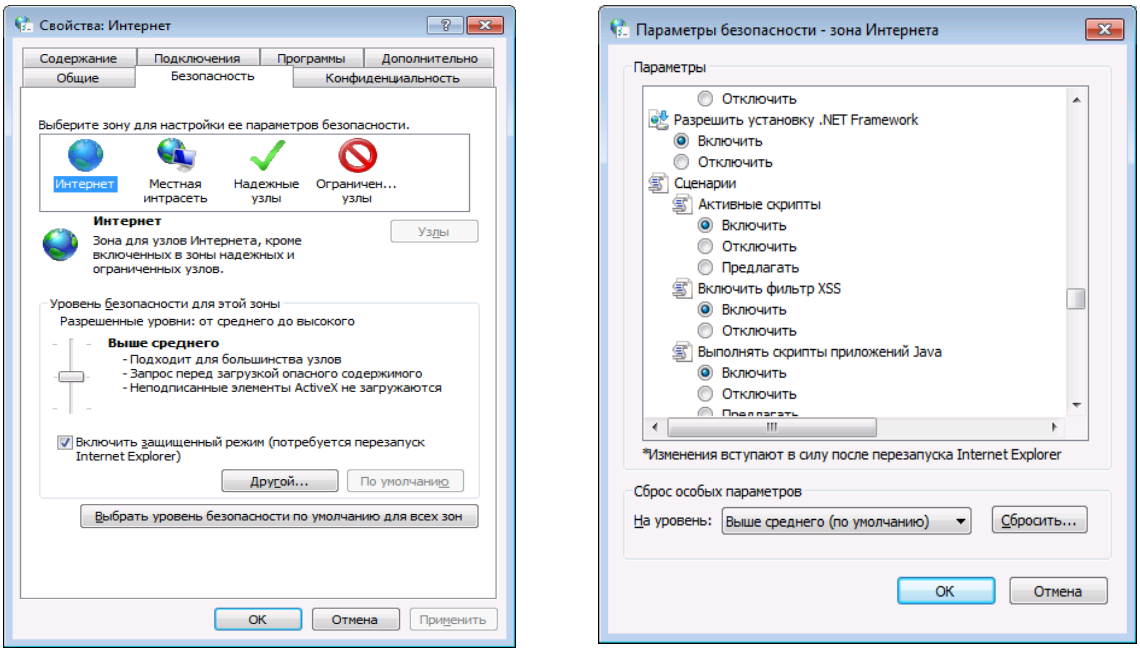


Рисунок 2.11 – Настройка параметров безопасности зоны

После завершения настройки параметров безопасности зон Windows необходимо запустить программу. Запустив программу, пользователь увидит главное меню, где должен выбрать пункт «Задача» – «Создать задание» (Рисунок 2.12).

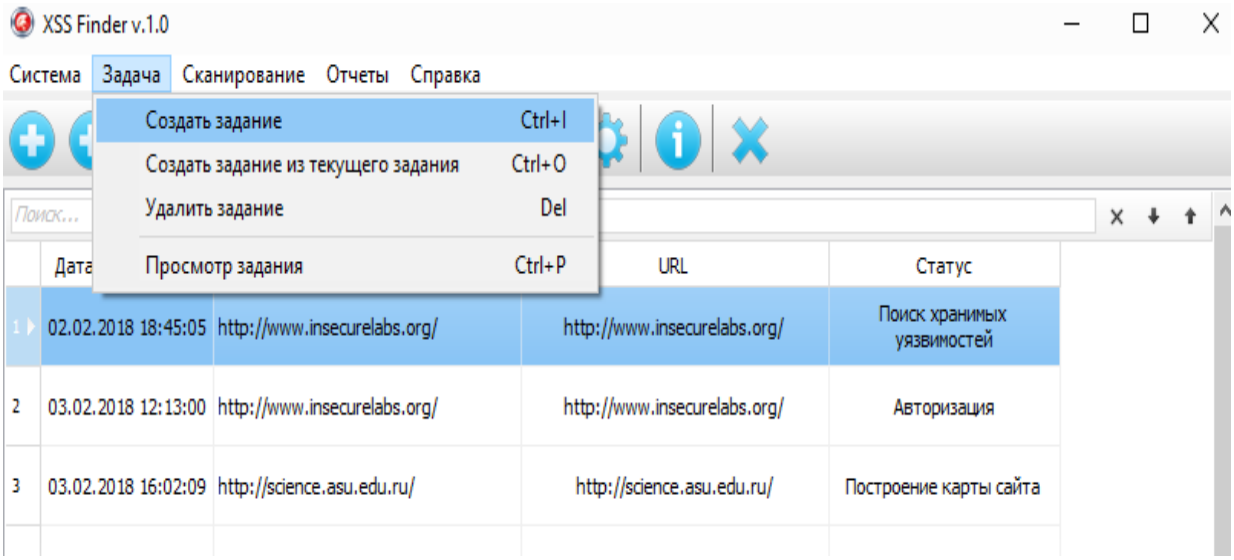


Рисунок 2.12 – Интерфейс создания задания

После создания задания и ввода данных, необходимо перейти в пункт «Сканирование» и выбрать интересующий тип сканирования. Тестирование веб-приложения производится в соответствии с информацией, введённой в задании.

В зависимости от задания, порядок выполнения которых определяет пользователь, появляется определенное окно с отчетами (Рисунок 2.13).

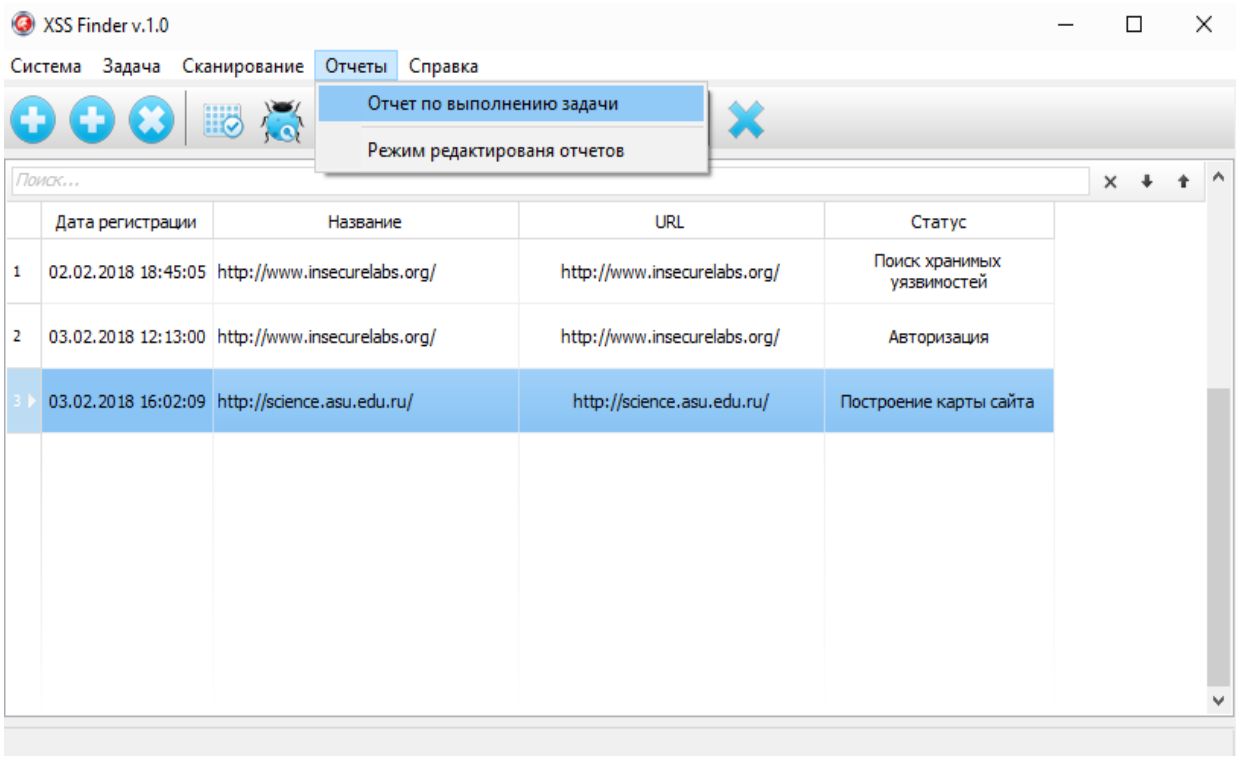


Рисунок 2.13 – Список отчетов о протестированных веб-приложениях

Для просмотра отчета необходимо выбрать из списка нужное задание и перейти в пункт «Отчет по выполнению задачи». Также предусмотрена функция редактирования формата представления отчета, при необходимости можно добавить дополнительные поля, которые будут содержать информацию.

2.5 Выводы по 2 главе

На основе методики последовательного применения наиболее эффективных алгоритмов обнаружения XSS-уязвимостей разработаны соответствующие алгоритмы детектирования, отличающиеся тем, что предусматривают поиск в «закрытой» части веб-ресурса. Алгоритмы детектирования реализованы в виде программного обеспечения, которое успешно апробировано и внедрено в организацию ООО НПП «ДосЛаб» (Приложение Б). Также на разработанное ПО получено свидетельство о государственной регистрации (Приложение В).

Глава 3 Результаты практического применения

3.1 Тестирование разработанного ПО и аналогичных решений

Для оценки качества работы разработанного ПО необходимо было сравнить результаты его работы с результатами аналогичных решений, которые пользуются популярностью среди разработчиков веб-приложений, а именно: XSSF, XenoTix, Wapiti, XSpider (MAX-Patrol), Nemesida Scanner, Acunetix Online Web Security Scanner. Перечисленные программы и разработанное ПО были протестированы на специализированном веб-ресурсе, предназначенный для тренировок и тестирования по поиску XSS-уязвимостей (*www.insecurelabs.org*). На Рисунке 3.1 представлена начальная страница тренировочного веб-ресурса.

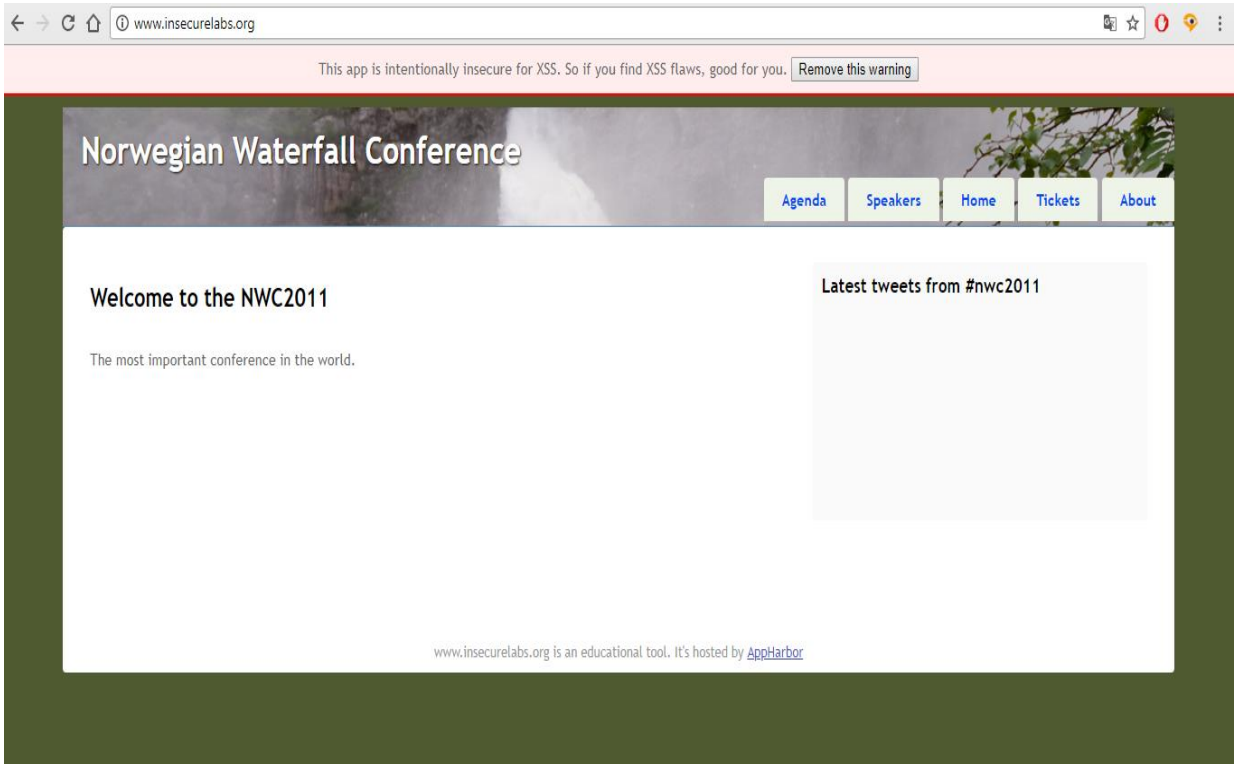


Рисунок 3.1 – Начальная страница веб-ресурса

Веб-сайт (www.insecurelabs.org) содержит 8 XSS-уязвимостей двух видов («хранимые» и «отраженные»). Результаты тестирования свидетельствуют о том, что разработанное ПО минимальное количество времени в отличие от аналогичных решений [28]. В Таблице 3.1 представлены результаты поиска XSS-уязвимостей разработанного ПО и аналогичных решений.

Таблица 3.1 – Результаты тестирования программ на интернет ресурсе, предназначенном для тестирования и поиска XSS-уязвимостей

Наименование показателя	XSSF	Xeno Tix	Wapiti	XSpider	Neme-sida Scanner	Acunetix Web Security Scanner	XSS-Finder
Количество найденных уязвимостей / общее количество	6/8	5/8	6/8	8/8	7/8	8/8	8/8
Общее затраченное время на поиск (сек.)	40	49	37	35	33	30	28
Среднее время нахождения одной уязвимости (сек.)	6,67	9,8	6,17	4,37	4,71	3,75	3,5

Также для проверки работоспособности обнаружения XSS-уязвимостей в закрытой части веб-ресурса, было решено протестировать разработанную программу на реальных веб-ресурсах, где предусмотрена функция авторизации пользователя: www.podarki.ru/, www.yaplakal.com/.

Тестирование перечисленных веб-ресурсов проводилось следующим образом:

1 создано задание с указанием URL тестируемого веб-ресурса, в параметрах введены данные для авторизации;

2 затем запущен процесс «Создание карты сайта»;

3 после создания карты сайта запущен «Полный цикл сканирования»;

4 просканировав веб-ресурс ПО сформировал отчет об уязвимостях и выдал рекомендации по их устранению.

Отчет с результатами тестирования на тренировочном веб-ресурсе представлен в Приложении Г. На основе результатов тестирования были получены данные распространенности XSS-уязвимостей (см. Таблица 3.2).

В Таблице 3.2 приведены результаты тестирования перечисленных веб-ресурсов.

Таблица 3.2 – Результаты тестирования веб-ресурсов

Наименование показателя	XSSF	Xen oTix	Wapiti	XSpider	Neme-sida Scanner	Acunetix Web Security Scanner	XSS-Finder
Количество найденных уязвимостей / общее количество	7/13	8/13	8/13	10/13	9/13	10/13	13/13
Общее затраченное время на поиск (сек.)	53	57	50	55	56	50	56
Среднее время нахождения одной уязвимости (сек.)	7,57	7,12	6,25	5,50	6,22	5,00	4,31

3.2 Расчет экономической эффективности

Расчёт затрат на реализацию программного обеспечения

Общая стоимость затрат на создание программного продукта будет складываться из:

1) затраты на создание программного обеспечения;

2) затраты на внедрение программного обеспечения;

3) эксплуатационные затраты.

Себестоимость разработки программного обеспечения определяется по формуле (3.1):

$$С_{пп} = М+С_{осн,зп} + С_{доп,зп}+С_{н} + С_{эвм}+И_{пп}+Н+Б, \quad (3.1)$$

где М – материальные затраты;

$С_{осн,зп}$ – основная заработная плата персонала (руб.);

$С_{доп,зп}$ – дополнительная заработная плата персонала (руб.);

$С_{н}$ – налоги на заработную плату (руб);

$С_{эвм}$ – затраты, связанные с эксплуатацией ЭВМ;

$И_{пп}$ – износ лицензионных программных продуктов ($И_{пп} = 0$);

Н – накладные расходы;

Б – покупка лицензии Microsoft Access 2010 (5000 руб.).

Основная заработная плата определяется по формуле (3.2):

$$С_{осн,зп} = Т_{разраб}*З, \quad (3.2)$$

где $Т_{разраб}$ – время необходимое для разработки ПП (ч.);

З – основная заработная плата персонала за один час (руб./ч).

Для разработки программного обеспечения необходимо $Т_{разраб}$ = 336 ч. Исполнителем является программист. Оклад программиста 3 категории составляет 40000 руб. в месяц. Основная заработная плата за 1 ч составит:

$$З = \frac{40000}{8*26} = 192 \text{ руб./ч.}$$

Основная заработная плата программиста за весь период разработки по формуле (3.2) равна:

$$С_{осн,зп} = 336*192 = 64512 \text{ руб.}$$

Дополнительная заработная плата составляет 12 % от основной заработной платы. Дополнительная заработная плата программиста за весь период разработки составит:

$$С_{доп,зп} = 64512*0,12 = 7741 \text{ руб.}$$

Налоги на заработную плату составляют 30 % от суммы основной и дополнительной заработной платы:

$$С_{н} = 0,3*4032 = 21675 \text{ руб.}$$

Затраты, связанные с эксплуатацией ЭВМ, определяются по формуле (3.3):

$$С_{эвм} = С_{маш/ч}*Т_{ЭВМ}, \quad (3.3)$$

где $С_{эвм}$ – себестоимость 1 машино-часа работы ЭВМ;

$Т_{ЭВМ}$ – время работы на ЭВМ при разработке программного обеспечения, требуется $Т_{ЭВМ}$=150 ч.

Стоимость 1 машино-часа работы ЭВМ определяется по формуле (3.4):

$$С_{маш/ч} = \frac{С_{эл,эн}+А+С_{рем}}{Т_{вт}}, \quad (3.4)$$

где $С_{эл,эн}$ – стоимость потребляемой в год электроэнергии, руб.;

А – амортизация в год, руб.;

$С_{рем}$ – затраты на ремонт в год, руб.;

$Т_{вт}$ – действительный фонд времени работы вычислительной техники, ч.

Стоимость потребляемой в год электроэнергии рассчитывается по формуле (3.5):

$$С_{эл,эн} = р*Т_{ном}*с_{э}, \quad (3.5)$$

где р – мощность, потребляемая из сети одной ЭВМ, кВт;

$Т_{ном}$ – номинальный фонд времени работы ЭВМ в год, ч;

$c_э$ – стоимость 1 кВт/ч электрической энергии, руб./(кВт·ч).

Мощность, потребляемая из сети одной ЭВМ, p=0,25 кВт. Стоимость 1 кВт·ч электрической энергии $c_э$=4,04 руб./(кВт·ч). Номинальный фонд времени работы ЭВМ равен:

$$T_{ном}=8\cdot 26\cdot 12=2496 \text{ ч.}$$

За год отчисления на электрическую энергию по формуле (3.6) составят:

$$C_{эл,эн} = 0,25*2496*4,04 = 2521 \text{ руб.}$$

Амортизация вычислительной техники равна 25 % от ее стоимости. Стоимость ЭВМ равна 20000 руб. Тогда амортизация вычислительной техники за год составит 5000 рублей.

Затраты на ремонт в год равны 4 % от стоимости ЭВМ и равны 800 рублей.

Действительный фонд времени работы ЭВМ в год рассчитывается по формуле (3.7):

$$T_{вт} = T_{ном} - T_{проф}, \quad (3.7)$$

где $T_{проф}$ – годовые затраты времени на профилактические работы (принимаются 10 % от $T_{ном}$).

Действительный фонд времени работы ЭВМ по формуле (3.8):

$$T_{вт} = 2496 – 2496*0,1 = 2246 \text{ ч.}$$

Стоимость машино-часа по формуле (4.5):

$$C_{маш/ч} = \frac{2521+5000+800}{2246} = 3,7 \text{ руб./ч.}$$

Затраты на содержание и эксплуатацию ЭВМ по выражению (3.4):

$$C_{эвм} = 3,7*150 = 555 \text{ руб.}$$

Накладные расходы равны 30 % от основной заработной платы и составляют:

$$Н = 0,3*64512 = 19353 \text{ руб.}$$

Итого на реализацию программного обеспечения выходит:

$$C_{пп} = 40+64512+7741+1210+555+19353+5000 = 98411 \text{ руб.}$$

Заработная плата персонала, занимающегося тестированием веб-приложений

В месяц заработная плата персонала, занятого выполнением поиска XSS-уязвимостей составляет:

$$ЗП_м = (1+0,3)*(О*Д/К), \quad (3.5)$$

где $ЗП_м$ – заработная плата сотрудника в месяц, руб.;

О – месячный должностной оклад сотрудника, (О = 10 000 руб.);

Д – количество дней в месяц, необходимых для выполнения поставленной задачи, дни;

К – среднее количество рабочих дней за месяц равным 26 дням.

По формуле (3.5) определяется заработная плата сотрудника (Д=12 дней):

$ЗП_м$ = (1+0,3)*(10 000*12/26) = 6 000руб. (месячная заработная плата);

$ЗП_г$ = 6 000 * 12 = 72 000 руб. (годовая заработная плата).

Так как в для тестирования необходимо как минимум 2 сотрудников, то общая сумма заработной платы в год равна 144 000 рублей.

Заработная плата персонала после внедрения программного продукта по формуле (3.5) равна (Д=3):

$ЗП_м$ = (1+0,3)*(3 000*3/26) = 450 руб. (месячная заработная плата).

$ЗП_г$ = 450 * 12 = 5400 руб. (годовая заработная плата).

Для 2 сотрудников: 5400*2=10 800 рублей.

Тогда условно-годовая экономия в заработной плате равна: 144 000-10 800=133 200 руб.

Расчет коэффициента экономической эффективности

Расчетный коэффициент экономической эффективности определяется по формуле:

$$E_р = \frac{Э_{год}}{С_{пп}}, \quad (3.6)$$

где $Э_{год}$ = 133 200 руб. Тогда $E_р$ = 1,35.

Расчетный (фактический) срок окупаемости определяется следующим образом:

$$Т = \frac{1}{E_р} = 0,7$$

Проведенный анализ экономического эффекта от внедрения разработанного программного обеспечения показывает, что его применение достаточно эффективно. Это выражается в следующем:

- затраты времени на поиск и устранение XSS-уязвимости сокращается на 70 %;
- годовая экономия составляет 133 200 рублей;
- расчетный коэффициент эффективности равен 1,35.

3.3 Эргономика проектного решения

Эргономика занимается комплексным изучением и проектированием трудовой деятельности с целью оптимизации орудий, условий и процесса труда, а также профессионального мастерства. Ее предметом является трудовая деятельность, а объектом исследования – системы «человек – орудие труда – предмет труда –

производственная среда» [29]. Эргономика относится к тем наукам, которые можно различать по предмету и специфическому сочетанию методов, применяемых в них.

Эргономика рабочего места

Рабочее место и взаимное расположение всех его элементов должно соответствовать антропометрическим, физическим и психологическим требованиям. Большое значение имеет также характер работы. В частности, при организации рабочего места инженера по информационной безопасности должны быть соблюдены следующие основные условия:

- площадь на одно рабочее место пользователей персональной электронно-вычислительной машины (ПЭВМ) на базе электроннолучевой трубки (ЭЛТ) должна составлять не менее 6 $м^2$;

- высота рабочей поверхности стола для взрослых пользователей должна регулироваться в пределах 680 - 800 мм;

- модульными размерами рабочей поверхности стола для ПЭВМ, на основании которых должны рассчитываться конструктивные размеры, следует считать: ширину 800, 1000, 1200 и 1400 мм, глубину 800 и 1000 мм при нерегулируемой его высоте, равной 725 мм;

- рабочий стол должен иметь высоту - 600 мм, ширину - 500 мм, глубину на уровне согнутых ног - 450 мм, глубину на уровне вытянутых ног –650 мм;

- конструкция рабочего стула должна обеспечивать: ширину и глубину поверхности сиденья (400 мм), поверхность сиденья с закругленным передним краем, регулировку высоты поверхности сиденья (400 - 550 мм) и углам наклона (вперед до 15 град, назад до 5 град)., регулировку подлокотников по высоте над сиденьем в пределах 230 +-30 мм и внутреннего – расстояния между подлокотниками в пределах 350 -500 мм.

Рабочее место пользователя ПЭВМ должно быть оборудовано подставкой для ног, которая имеет ширину не менее 300 мм, глубину не менее 400 мм, регулировку по высоте в пределах до 150 мм и по углу наклона опорной поверхности подставки до 20° [30]. Правильная и неправильная позы за компьютером показаны на Рисунках 3.2 и 3.3.

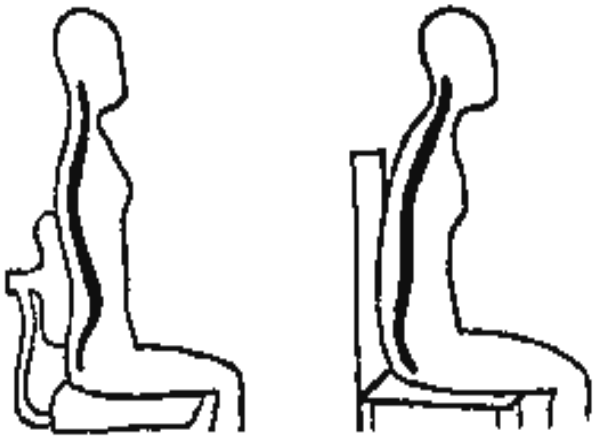


Рисунок 3.2 – Правильная и неправильная позы за компьютером

Рисунок 3.3 – Правильная позиция за компьютером

Также необходимо соблюдать оптимальный режим работы, представленный на Рисунке 3.4.

В последнее время вместо компьютерной мыши используется трекбол. При работе с трекболом объем движений, выполняемых рукой, уменьшается, но объем движений, производимых пальцами, увеличивается. Поэтому никакого преимущества одного манипулятора перед другим нет.

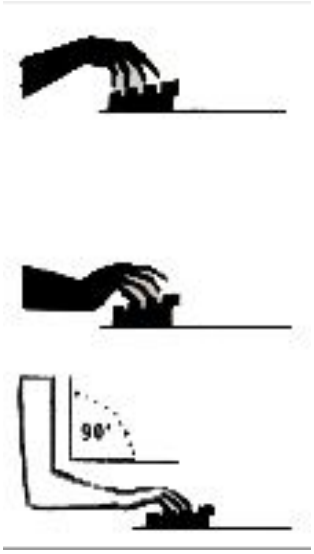


Рисунок 3.4 – Положение запястья и кисти при работе на клавиатуре

Эргономический анализ программного обеспечения

В ходе работы была разработана программа для обнаружения XSS-уязвимостей на основе анализа полной карты веб-приложения. Пользователями программы являются разработчики, аудиторы и тестировщики ИС. Для установки данной программы необходимы следующие требования:

- IBM-совместимый компьютер с МП Intel Core i3 и выше с тактовой частотой 1 ГГц;
- 1 ГБ ОЗУ;
- ОС - Windows XP/Vista/7/8/8.1/10.

Данная программа написана на языке программирования Delphi. Важно также учитывать организацию диалога (Таблицы 3.3, 3.4), пространственную организацию информации (Таблица 3.5) и цветовые решения соотношения «фигура-фон» (Таблица 3.6).

Основные характеристики первичных, вторичных и всплывающих окон приведены в Таблице 3.3.

Таблица 3.3 – Организация диалога

Использование функциональных клавиш для кодирования выбора элементов меню (да\нет)	Использование гипертекста (нет\да)		Использование динамики (анимации) (нет\да)	
	Первичные окна	Вторичные окна	Первичные окна	Вторичные окна
Нет	Нет	Нет	Нет	Нет

Таблица 3.4 – Организация диалога

Степень вмешательства пользователя в ход программы	Режим диалога	Форма вопросов-ответов		Выдача сообщений о состоянии системы, качестве работы пользователя (да\нет)	
		Первичные окна	Вторичные окна	Первичные окна	Вторичные окна
1	2	3	4	5	6
Пользователь может приостановить ход программы для осмысления информации и продолжить работу далее с места паузы	Много экранный	Выбор варианта из предложенного списка	Ввод пользователем в систему числовых и других параметров	Нет	Нет

Таблица 3.5 – Пространственная организация информации

Размер окон (в % к площади экрана)			Возможность менять размер и расположение окон на экране (нет\да)		
Первичные	Вторичные	Всплывающие	Первичные	Вторичные	Всплывающие
1	2	3	4	5	6
60	60	Нет	Нет	Нет	Нет

Как показано в Таблице 3.6 в первичных формах используется неагрессивная цветовая палитра, что способствует лучшему восприятию информации.

Таблица 3.6 – Цветовые решения соотношения «фигура-фон»

Цветовое оформление первичных окон			Общее количество используемых цветов	Тип визуальной среды	Возможность настройки цветовой палитры окон
Фон	Фигура	Текст			
1	2	3	4	5	6
Белый	Прямоугольник	Черный, серый	3	Нормальная	Нет

Все первичные формы представлены в виде прямоугольников, основная палитра содержит три основных цвета: серый, черный и красный.

Вторичные окна – это окна, в которых пользователи и компьютер ведут диалог параллельно диалогу в первичном окне. Вторичные окна, как и первичные, представлены в виде прямоугольников, цветовая гамма включает в себя серый и черный цвета (Таблица 3.7).

Таблица 3.7 – Цветовые решения соотношения «фигура-фон»

Цветовое оформление вторичных окон			Общее количество используе-мых цветов	Тип визуальной среды	Возможность настройки цветовой палитры окон (нет\да)
Фон	Фигура	Текст			
Белый	Прямоу-гольник	Черный, серый, 14 пт	4	Нормальная	Нет

Всплывающие окна представляют собой участок экрана, в котором располагается экранируемая панель, которая расширяет диалог пользователя через первичные и вторичные окна.

В Таблице 3.8 содержится описание цветового решения соотношения «фигура-фон» для всплывающих окон.

Таблица 3.8 – Цветовые решения соотношения «фигура-фон»

Цветовое оформление вторичных окон			Общее количество используемых цветов	Тип визуальной среды	Возможность настройки цветовой палитры окон (нет\да)
Фон	Фигура	Текст			
Белый	Прямоугольник	Черный, серый 14 пт	2	Нормальная	Нет

В данном разделе рассмотрены вопросы, связанные с эргономикой рабочего места, а также был проведен эргономический анализ программного обеспечения.

Благодаря предложенным эргономическим мероприятиям эффективность внедрения программного обеспечения будет выше.

3.4 Пути дальнейшего совершенствования

В дальнейшем работа будет продолжена в виде магистерской диссертации. Намечается разработать кроссплатформенную версию программного обеспечения, в котором помимо поиска XSS, также будет осуществляться поиск следующих актуальных видов веб-уязвимостей из списка OWASP TOP 10 2017: (XXE (eXternal Entity), CSRF (Cross-Site Request Forgery), SQL Injection).

3.5 Вывод по 3 главе

Разработанное программное обеспечение успешно апробировано в организации ООО НПП «ДосЛаб». Для оценки качества разработанного ПО проведено его сравнение по основным параметрам с аналогичными решениями.

Результаты тестирования свидетельствуют о том, что разработанное ПО имеет ряд преимуществ перед существующими продуктами, а именно:

- осуществляет поиск XSS в «закрытой» части веб-ресурса;
- находит большее количество XSS-уязвимостей по сравнению с аналогичными решениями;
- затрачивает минимальное количество времени на поиск;
- формирует более информативные рекомендации по устранению обнаруженных уязвимостей.

Перечисленные преимущества делают разработку конкурентоспособным программным продуктом, направленным на детектирование XSS-уязвимостей.

Проведенный анализ экономического эффекта от внедрения разработанного ПО свидетельствует о том, что затраты времени на поиск и устранение XSS-уязвимостей веб-разработчиком сокращается на 70% и годовая экономия организацией составляет 133 200 рублей.

ЗАКЛЮЧЕНИЕ

При написании бакалаврской работы были проанализированы особенности проведения компьютерных атак вида «межсайтовый скриптинг». Также проведен анализ существующих программ предназначенных для обнаружения XSS-уязвимостей, результаты которого показали, что данные программы не в должной мере справляются с поставленными задачами, тем самым приводят к снижению уровня информационной безопасности веб-приложения.

На основе методики, основанной на последовательном применении наиболее эффективных алгоритмов обнаружения различных типов XSS, разработаны соответствующие алгоритмы поиска уязвимостей. Данные алгоритмы реализованы в виде соответствующего ПО, которое позволяет повысить эффективность защиты веб-приложения от XSS-атак. Программное обеспечение значительно упрощает процесс тестирования веб-ресурса разработчиком, благодаря функционалу формирования отчета с рекомендациями по устранению найденных XSS.

Также разработанное программное обеспечение успешно апробировано в ООО НПП «ДосЛаб», о чем свидетельствует соответствующий акт внедрения.

Экспериментально доказана конкурентоспособность разработанной программы. Проведенный анализ экономического эффекта показал, что внедрение разработанного программного обеспечения эффективно. Также были рассмотрены вопросы, связанные с техникой безопасности и охраной труда.

Результаты бакалаврской работы опубликованы в 3 печатных изданиях, один из которых входит в перечень ВАК. Работа была представлена и удостоена наград в следующих конкурсах:

- удостоена статуса финалиста в восемнадцатой всероссийской конкурс-конференции среди студентов и аспирантов по информационной безопасности «SIBINFO-2018»;
- удостоена статуса победителя в конкурсе проектных работ имени академика А.А. Бочвара.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1 Элхади А. М. Полное пособие по межсайтовому скриптингу // SecurityLab.ru [Электронный ресурс]. 11.12.2012. – URL: www.securitylab.ru/analytics/432835.php?R=1 (дата обращения 15.03.2018).

2 OWASP Top 10 - 2017 The Ten Most Critical Web Application Security Risks // OWASP the free and open software security community [Электронный ресурс]. 2017 – URL: www.owasp.org/images/7/72/OWASP_Top_10-2017_%28en%29.pdf.pdf (дата обращения 20.03.2018).

3 Positive Research. Уязвимости веб-приложений: пора анализировать исходный код // Positive Research Center [Электронный ресурс]. 08.08.2017 – URL: www.blog.ptsecurity.ru/2017/08/web-attacks.html. (дата обращения 20.03.2018).

4 Евтеев Д. SQL Injection от А до Я // Positive Technologies [Электронный ресурс]. 11.10.2014. – URL: www.ptsecurity.com/upload/corporate/ru-ru/analytics/PT-devteev-Advanced-SQL-Injection.pdf (дата обращения 21.03.2018).

5 КБлог Свободного Вебмастера // Межсайтовый скриптинг, как защитить сайт от XSS атаки [Электронный ресурс]. 29.01.2018 – URL: https://webliberty.ru/xss/ (дата обращения 18.03.2018).

6 Klein A. Третий тип XSS: Межсайтовый скриптинг через DOM // SecurityLab.ru [Электронный ресурс]. 09.10.2006. – URL: https://www.securitylab.ru/analytics/275087.php (дата обращения 18.03.2018).

7 Berners-Lee T. Uniform Resource Locators // RFC 1738 – IETF [Электронный ресурс] – URL: www.ietf.org/rfc/rfc1738.txt (дата обращения 23.03.2018).

8 Do Son. Clickjacking Attack // Penetration Testing Security Training Share [Электронный ресурс] 27.07.2017. – URL: https://securityonline.info/clickjacking-attack/ (дата обращения 24.03.2018).

9 Cross-frame-scripting // OWASP the free and open software security community [Электронный ресурс]. 22.06.2017. – URL: www.owasp.org/index.php/Cross_Frame_Scripting (дата обращения 24.03.2018).

10 Types of XSS: Stored XSS, Reflected XSS and DOM-based XSS // Acunetix Blog [Электронный ресурс]. 17.01.2018. – URL: www.acunetix.com/websitesecurity/xss/ (дата обращения 22.03.2018).

11 Cross-site-scripting // OWASP the free and open software security community [Электронный ресурс]. 11.04.2009. – URL: www.owasp.org/index.php/Cross-site-scripting (дата обращения 25.03.2018).

12 How to write a XSS (cross site scripting) Worm for a McCodes Site // Hackbbs.org [Электронный ресурс]. 10.09.2017. – URL: ftp://hackbbs.org/milworm/272 (дата обращения 18.03.2018).

13 Mavituna F. XSS tunneling // Portcullis-security.com [Электронный ресурс]. 06.12.2016. – URL: https://labs.portcullis.co.uk/download/XSS-Tunnelling.pdf (дата обращения 18.03.2018).

14 DOM Based XSS // OWASP the free and open software security com-munity [Электронный ресурс]. 22.06.2017. – URL: www.owasp.org/index.php/DOM_Based_XSS (дата обращения 24.03.2018).

15 Mozilla Firefox // Mozilla Corporation [Электронный ресурс]. 18.03.2017. – URL: https://www.mozilla.org/ru/firefox/ (дата обращения 18.03.2018).

16 Fogie S., Grossman J., Hansen R., Rager A., Petkov P. XSS attacks: Cross Site Scripting exploits and defense – Seth Fogie, Oxford: Elsevier Limited, 2007. – 448 p.

17 CSP (Политика Защиты Контента) // MDN Web Docs [Электронный ресурс]. 21.12.2016. – URL: https://developer.mozilla.org/ru/docs/Web/Security/CSP (дата обращения 18.03.2018).

18 XSpider // Positive Technologies [Электронный ресурс]. 12.05.2013. – URL: www.ptsecurity.com/ru-ru/products/xspider/ (дата обращения 20.03.2018).

19 Nemesida Scanner // PENTESTIT [Электронный ресурс]. 03.01.2017. – URL: www.pentestit.ru/nemesida-scanner/ (дата обращения 20.03.2018).

20 Audit Your Web Security with Acunetix Vulnerability Scanner // Acunetix [Электронный ресурс]. 03.01.2017. – URL: www.acunetix.com/vulnerability-scanner/ (дата обращения 20.03.2018).

21 Джатана Н., Агравал А., Собти К. Пост-эксплуатация XSS: продвинутые методы и способы защиты // SecurityLab.ru [Электронный ресурс]. 12.05.2013. – URL: www.securitylab.ru/analytics/440187.php (дата обращения 30.03.2018).

22 Xenotix XSS Exploit Framework // OWASP [Электронный ресурс]. 22.06.2017. – URL: www.owasp.org/index.php/OWASP_Xenotix_XSS_Exploit_Framework (дата обращения 24.03.2018).

23 Wapiti - сканер уязвимостей веб-приложений // DefconRU [Электронный ресурс]. 29.10.2015. – URL: https://defcon.ru/penetration-testing/1657/ (дата обращения 18.03.2018).

24 Чем толстый клиент отличается от тонкого? // Life 1C [Электронный ресурс]. 07.04.2015. – URL: http://life1c.ru/post/1285 (дата обращения 18.03.2018).

25 SQL Anywhere // Сайбес компания [Электронный ресурс]. 12.10.2016. – URL: https://www.sybase.ru/products/asa (дата обращения 18.03.2018).

26 InterBase // IBase.ru [Электронный ресурс]. 10.12.2015. – URL: http://www.ibase.ru/interbase/ (дата обращения 18.03.2018).

27 Носиров З. А., Ажмухамедов И. М. Обнаружение XSS-уязвимостей на основе анализа полной карты веб-приложения // Системы управления, связи и безопасности. 2018. №1. С. 78-94. [Электронный ресурс]. – URL: http://sccs.intelgr.com/archive/2018-01/03-Nosirov.pdf (дата обращения 13.03.2018).

28 Носиров З.А., Ажмухамедов И.М. Разработка программного обеспечения для выявления XSS-уязвимостей // Проблемы информационной безопасности. Т. 1 — Ростов-на-Дону: РГЭУ (РИНХ), 2018. — С. 216-221.

29 Эргономика рабочего места за компьютером. [Электронный ресурс]. URL: http://zdravyshka.ru/Poleznye-sovety/Sovety-vracha/ergonomika-rabochego-mesta-za-kompyuterom.html (дата обращения: 15.03.2018).

30 Техника безопасности и охрана труда на предприятии. [Электронный ресурс]. URL: http://outsourcing.yourbuhg.ru/autsorsing-ohrany/ohrana-truda-na-predpriatii.html (дата обращения: 20.03.2018).

31 Носиров З.А., Ажмухамедов И.М., Выборнова О.Н. Получение закрытой информации из открытых источников // Проблемы информационной безопасности. Т. 1 — Ростов-на-Дону: РГЭУ (РИНХ), 2018 — С. 154-157.

32 Носиров З.А., Ажмухамедов И.М., Марьенков А.Н. Разработка программного обеспечения для обнаружения XSS-уязвимостей и выдачи рекомендаций по их устранению: свидетельство на ПрЭВМ № 2018610645; заявитель Носиров З.А.; патентообладатели: Носиров З.А., Ажмухамедов И.М., Марьенков А.Н. – № 2017661927 заявл. 21.11.2017; опубл. 15.01.2018.

33 Носиров З.А. Инструмент для обнаружения вредоносного кода в системах управления контентом: свидетельство на ПрЭВМ № 2017662831; заявитель Носиров З.А.; патентообладатель: Носиров З.А. – № 2017619928 заявл. 26.09.2017; опубл. 17.11.2017.

34 Microsoft Access // Wikipedia.org [Электронный ресурс] 14.03.2015. URL: https://ru.wikipedia.org/wiki/Microsoft_Access (дата обращения 18.03.2018).

ПРИЛОЖЕНИЕ А

Листинг программы

```
unit uSiteStoredXssFinder;
interface
uses Classes, SysUtils, DB, ADODB, uDatabaseObject, uSiteNavigator, Forms,
MSHTML, SHDocVw, idUri, Vcl.ExtCtrls, Dialogs;
type
    /// <summary>
    /// Режимы работы XssFinder
    /// </summary>
    TSiteStoredXssFinderMode = (ssxfmInject, ssxfmFind);
    /// <summary>
    /// Структура для хранения полей формы (для размежения в очереди на заполнение)
    /// </summary>
    TSiteStoredXssFinderPageField = class
    private
        /// <summary>
        /// Номер html-формы
        /// </summary>
        FHtmlFormNumber: Integer;
        /// <summary>
        /// Номер html-элемента формы
        /// </summary>

        FHtmlElementNumber: Integer;
        /// <summary>
        /// Задает  номер html-элемента формы
        /// </summary>
        procedure SetHtmlElementNumber(const Value: Integer);
        /// <summary>
        /// Задает  номер html-формы
        /// </summary>
        procedure SetHtmlFormNumber(const Value: Integer);
    published
        /// <summary>
        /// Конструктор класса
        /// </summary>
        constructor Create;
        /// <summary>
        /// Деструктор класса
        /// </summary>
        destructor Destroy; override;
        /// <summary>
        /// Номер html-элемента формы
        /// </summary>
        property HtmlElementNumber: Integer read FHtmlElementNumber write
SetHtmlElementNumber;
        /// <summary>
        /// Номер html-формы
        /// </summary>
        property HtmlFormNumber: Integer read FHtmlFormNumber write SetHtmlFormNumber;
    end;
    /// <summary>
    /// Класс, реализующий поиск хранимых уязвимостей.
    /// </summary>
    TSiteStoredXssFinder = class(TSiteNavigator)
    private
        /// <summary>
        /// Общее количество найденных ссылок
        /// </summary>
```

```
FTotalPageCount: Integer;
        /// <summary>
        /// Количество посещенных ссылок
        /// </summary>
        FVisitedPageCount: Integer;
        /// <summary>
        /// Очередь URL-адресов (очередь содержит варианты URL, в которые встроены XSS
коды)
        /// </summary>
        FXSSQueue: TStringList;
        /// <summary>
        /// Список XSS, которые будут проверяться
        /// </summary>
        FTaskProblem: TADOQuery;
        /// <summary>
        /// Идентфикатор проверяемой страницы
        /// </summary>
        FCurrentPageID: Integer;
        /// <summary>
        /// URL текущей страницы
        /// </summary>
        FCurrentPageURL: string;
        /// <summary>
        /// Количество найденных уязвимостей
        /// </summary>
        FProblemCount: Integer;
        /// <summary>
        /// Режим работы анализатора (инъекция или поиск)
        /// </summary>
        FMode: TSiteStoredXssFinderMode;
        /// <summary>
        /// Флаг "Первая загрузка страницы"
        /// </summary>
        FFirstPageLoading: Boolean;
        /// <summary>
        /// Флаг "вызов страницы из таймера"
        /// </summary>
        FFromTimer: Boolean;
        /// <summary>
        /// Очередь полей для заполнения
        /// </summary>
        FPageFieldsQueue: TStringList;
        /// <summary>
        /// Метод создает очередь полей
        /// </summary>
        procedure GetPageInputFieldQueue;
        /// <summary>
        /// Метод заполняет очередь XSS сигнатур
        /// </summary>
        procedure GetXSSQueue;
        /// <summary>
        /// Обработчик события "Загрузка документа завершена"
        /// </summary>
        procedure WebBrowserOnDocumentComplete(ASender: TObject; const pDisp: IDispatch;
const URL: OleVariant); override;
        /// <summary>
        /// Обработчик события "Загрузка файла"
        /// </summary>
        procedure WebBrowserOnFileDownload(ASender: TObject; ActiveDocument: WordBool;
var Cancel: WordBool); override;
        /// <summary>
```

```
 /// Обработчик события "Ошибка навигации"
        /// </summary>
        procedure WebBrowserOnNavigateError(ASender: TObject; const pDisp: IDispatch;
const URL, Frame, StatusCode: OleVariant; var Cancel: WordBool); override;
        /// <summary>
        /// Обработчик таймера
        /// </summary>
        procedure TimerOnTimer(Sender: TObject); override;
        /// <summary>
        /// Вычисляет количество страниц, в которые будут производиться инъекции
        /// </summary>
        function GetTaskPageCountForInject(TaskID: Integer): Integer;
        /// <summary>
        /// Вычисляет общее количество страниц для поиска уязвимостей
        /// </summary>
        function GetTaskPageCountForFind(TaskID: Integer): Integer;
        /// <summary>
        /// Задаем режим работы XssFinder
        /// </summary>
        procedure SetMode(const Value: TSiteStoredXssFinderMode);
        /// <summary>
        /// Выполняет проверку страницы на наличие хранимых XSS-уязвимостей
        /// </summary>
        procedure ExecutePageCheck(URL: string);
    public
        /// <summary>
        /// Конструктор класса
        /// </summary>
        constructor Create(Connection: TADOConnection);
        /// <summary>
        /// Деструктор класса
        /// </summary>
        destructor Destroy; override;
        /// <summary>
        /// Метод генерации карты сайта
        /// </summary>
        procedure Find;
        /// <summary>
        /// Общее количество найденных ссылок
        /// </summary>
        property TotalPageCount: Integer read FTotalPageCount;
        /// <summary>
        /// Количество посещенных ссылок
        /// </summary>
        property VisitedPageCount: Integer read FVisitedPageCount;
        /// <summary>
        /// Количество найденных уязвимостей
        /// </summary>
        property ProblemCount: Integer read FProblemCount;
        /// <summary>
        /// Очередь URL-адресов
        /// </summary>
        property XSSQueue: TStringList read FXSSQueue;
        /// <summary>
        /// Очередь полей для заполнения
        /// </summary>
        property PageFieldsQueue: TStringList read FPageFieldsQueue;
        /// <summary>
        /// Режим работы анализатора (инъекция или поиск)
        /// </summary>
        property Mode: TSiteStoredXssFinderMode read FMode;
```

```
end;
implementation
{ TSiteMapCreator }
constructor TSiteStoredXssFinder.Create(Connection: TADOConnection);
begin
    inherited Create(Connection);
    FXSSQueue := TStringList.Create;
    FPageFieldsQueue := TStringList.Create;
    FPageFieldsQueue.OwnsObjects := True;
    FMode := ssxfmInject;
end;
destructor TSiteStoredXssFinder.Destroy;
begin
    FXSSQueue.Free;
    FPageFieldsQueue.Free;
    if Assigned(FTaskProblem) then
        FTaskProblem.Free;
    inherited;
end;
procedure TSiteStoredXssFinder.ExecutePageCheck(URL: string);
var
    SubStr, HTML: WideString;
    I, J, HtmlElementNumber, HtmlFormNumber: Integer;
begin
    if Assigned(OnStatusTextChange) then
        OnStatusTextChange(Self, 'Поиск XSS: ' + URL);
    HTML := WideLowerCase((WebBrowser.Document as IHTMLDocument2).body.outerHTML);
    for I := 1 to FTaskProblem.RecordCount do
    begin
        FTaskProblem.RecNo := I;
        SubStr := WideLowerCase(FTaskProblem.FieldByName('Code').AsString);
        if Pos(SubStr, HTML) > 0 then
        begin
            PageCheckAdd(FCurrentPageID,
FTaskProblem.FieldByName('ProblemID').AsInteger, 2, URL);
            Inc(FProblemCount);
            if Assigned(OnStatusTextChange) then
                OnStatusTextChange(Self, 'Найдена XSS: ' + URL);
        end;
    end;
end;
procedure TSiteStoredXssFinder.TimerOnTimer(Sender: TObject);
var
    Page, TaskProblem: TADOQuery;
    URI: TIdURI;
    I: Integer;
    URL: string;
    L: TStringList;
begin
    if Mode = ssxfmInject then
    begin
        if (XSSQueue.Count = 0) and (PageFieldsQueue.Count = 0) then
        begin
            Page := GetQuery('select top 1 * from Page where Visited = False and
ForChecking = True and UsedHtmlForm = True and TaskID = ' + IntToStr(TaskID) +
                ' order by ID asc');
            try
                if ((Page.RecordCount = 0) and (TotalPageCount > 0)) then
                begin
                    SetMode(ssxfmFind);
                    FTotalPageCount := GetTaskPageCountForFind(TaskID);
```

```
                    Exit;
                    end;
                    try
                        FFirstPageLoading := True;
                        FCurrentPageID := Page.FieldByName('ID').AsInteger;
                        FCurrentPageURL := Page.FieldByName('URL').AsString;
                        FFromTimer := True;
                        Page.Edit;
                        Page.FieldByName('Visited').AsBoolean := True;
                        Page.Post;
                        Inc(FVisitedPageCount);
                        WebBrowser.Navigate(FCurrentPageURL);
                    except
                        on E: Exception do
                        begin
                            Page.Edit;
                            Page.FieldByName('ErrorMessage').AsString := E.Message;
                            Page.FieldByName('Error').AsBoolean := True;
                            Page.Post;
                        end;
                    end;
                finally
                    Page.Free;
                end;
            end
            else
            begin
                WebBrowser.Navigate(FCurrentPageURL);
            end;
        end
        else
        begin
            Page := GetQuery('select top 1 * from Page where Visited = False and ForChecking
    = True and TaskID = ' + IntToStr(TaskID) + ' order by ID asc');
            try
                if ((Page.RecordCount = 0) and (TotalPageCount > 0)) then
                begin
                    SetState(snsComplete);
                    Exit;
                end;
                try
                    FFirstPageLoading := True;
                    FCurrentPageID := Page.FieldByName('ID').AsInteger;
                    FCurrentPageURL := Page.FieldByName('URL').AsString;
                    FFromTimer := True;
                    Page.Edit;
                    Page.FieldByName('Visited').AsBoolean := True;
                    Page.Post;
                    Inc(FVisitedPageCount);
                    WebBrowser.Navigate(FCurrentPageURL);
                except
                    on E: Exception do
                    begin
                        Page.Edit;
                        Page.FieldByName('ErrorMessage').AsString := E.Message;
                        Page.FieldByName('Error').AsBoolean := True;
                        Page.Post;
                    end;
                end;
            finally
                Page.Free;
```

```
end;
    end;
    FFromTimer := True;
    Timer.Enabled := False;
end;

procedure TSiteStoredXssFinder.WebBrowserOnDocumentComplete(ASender: TObject; const
pDisp: IDispatch; const URL: OleVariant);
var
    Document: IHTMLDocument2;
    FormElement: IHTMLFormElement;
    Input: IHTMLElement;
    I, J, HtmlElementNumber, HtmlFormNumber: Integer;
    Data: TSiteStoredXssFinderPageField;
begin
    if Mode = ssxfmInject then
    begin
        if FFromTimer = True then
        begin
            if FFirstPageLoading then
            begin
                GetPageInputFieldQueue;
                GetXSSQueue;
                FFirstPageLoading := False;
            end;
            if PageFieldsQueue.Count > 0 then
            begin
                if Assigned(OnStatusTextChange) then
                    OnStatusTextChange(Self, 'Встраивание XSS: ' + URL);
                Data := PageFieldsQueue.Objects[0] as TSiteStoredXssFinderPageField;
                HtmlFormNumber := Data.HtmlFormNumber;
                HtmlElementNumber := Data.HtmlElementNumber;
                PageFieldsQueue.Delete(0);

                Document := WebBrowser.Document as IHTMLDocument2;
                if HtmlFormNumber < Document.Forms.length then
                begin
                    FormElement := Document.Forms.item(OleVariant(HtmlFormNumber),
varEmpty) as IHTMLFormElement;
                    if HtmlElementNumber < FormElement.length then
                    begin
                        Input := FormElement.item(OleVariant(HtmlElementNumber),
varEmpty) as IHTMLElement;
                        if (AnsiLowerCase(Input.tagName) = 'input') then
                        begin
                            Input.setAttribute('value', XSSQueue.Strings[0], 0);
                            FFromTimer := False;
                            FormElement.submit;
                            Exit;
                        end;
                        if (AnsiLowerCase(Input.tagName) = 'textarea') then
                        begin
                            Input.innerText := XSSQueue.Strings[0];
                            FFromTimer := False;
                            FormElement.submit;
                            Exit;
                        end;
                    end;
                end;
            end
            else
```

```
        begin
                if XSSQueue.Count > 0 then
                begin
                    XSSQueue.Delete(0);
                    if XSSQueue.Count = 0 then
                    begin
                        Timer.Enabled := True;
                        Exit;
                    end;
                end;
                GetPageInputFieldQueue;
            end;
        end
        else
        begin
            ExecutePageCheck(URL);
        end;
    end
    else
    begin
        ExecutePageCheck(URL);
    end;
    Timer.Enabled := True;
end;

procedure TSiteStoredXssFinder.GetPageInputFieldQueue;
var
    FormList: IHTMLElementCollection;
    Input: IHTMLElement;
    FormElement: IHTMLFormElement;
    I, J: Integer;
    Data: TSiteStoredXssFinderPageField;
begin
    PageFieldsQueue.Clear;
    FormList := (WebBrowser.Document as IHTMLDocument2).Forms;
    for I := 0 to FormList.length - 1 do
    begin
        FormElement := FormList.item(OleVariant(I), varEmpty) as IHTMLFormElement;
        for J := 0 to FormElement.length - 1 do
        begin
            Input := FormElement.item(OleVariant(J), varEmpty) as IHTMLElement;
            if AnsiLowerCase(Input.getAttribute('type', 0)) = 'text' then
            begin
                Data := TSiteStoredXssFinderPageField.Create;
                Data.HtmlFormNumber := I;
                Data.HtmlElementNumber := J;
                PageFieldsQueue.AddObject('', Data);
            end;

            if AnsiLowerCase(Input.tagName) = 'textarea' then
            begin
                Data := TSiteStoredXssFinderPageField.Create;
                Data.HtmlFormNumber := I;
                Data.HtmlElementNumber := J;
                PageFieldsQueue.AddObject('', Data);
            end;

        end;
    end;
end;
```

```
procedure TSiteStoredXssFinder.WebBrowserOnFileDownload(ASender: TObject;
ActiveDocument: WordBool; var Cancel: WordBool);
begin
    Cancel := True;
    Timer.Enabled := True;
end;
procedure TSiteStoredXssFinder.WebBrowserOnNavigateError(ASender: TObject; const pDisp:
IDispatch; const URL, Frame, StatusCode: OleVariant;
    var Cancel: WordBool);
begin
    Cancel := True;
    Timer.Enabled := True;
end;
procedure TSiteStoredXssFinder.Find;
begin
    FProblemCount := 0;
    FVisitedPageCount := 0;
    SetState(snsFree);
    TestRequirement;
    SetState(snsExec);
    SetMode(ssxfmInject);
    FTotalPageCount := GetTaskPageCountForInject(TaskID);
    if FTotalPageCount = 0 then
        SetState(snsComplete);
    if Assigned(FTaskProblem) then
        FTaskProblem.Free;
    FTaskProblem := GetQuery('select * from ViewTaskProblem where TaskID = ' +
IntToStr(TaskID));
    PageCheckClear(2);
    Timer.Enabled := True;
end;
function TSiteStoredXssFinder.GetTaskPageCountForInject(TaskID: Integer): Integer;
var
    Page: TADOQuery;
begin
    Page := GetQuery('select count(ID) as F1 from Page where ForChecking = True and
UsedHtmlForm = True and TaskID = ' + IntToStr(TaskID));
    try
        Result := Page.FieldByName('F1').AsInteger;
    finally
        Page.Free;
    end;
end;
function TSiteStoredXssFinder.GetTaskPageCountForFind(TaskID: Integer): Integer;
var
    Page: TADOQuery;
begin
    Page := GetQuery('select count(ID) as F1 from Page where ForChecking = True and
TaskID = ' + IntToStr(TaskID));
    try
        Result := Page.FieldByName('F1').AsInteger;
    finally
        Page.Free;
    end;
end;
procedure TSiteStoredXssFinder.GetXSSQueue;
var
    I: Integer;
begin
    XSSQueue.Clear;
    for I := 1 to FTaskProblem.RecordCount do
```

```
begin
        FTaskProblem.RecNo := I;
        XSSQueue.Add(FTaskProblem.FieldByName('Code').AsString);
    end;
end;
procedure TSiteStoredXssFinder.SetMode(const Value: TSiteStoredXssFinderMode);
begin
    TaskClearVisitFlag;
    FVisitedPageCount := 0;
    FMode := Value;
end;
{ TSiteStoredXssFinderPageField }
constructor TSiteStoredXssFinderPageField.Create;
begin
    inherited;
end;
destructor TSiteStoredXssFinderPageField.Destroy;
begin
    inherited;
end;
procedure TSiteStoredXssFinderPageField.SetHtmlElementNumber(const Value: Integer);
begin
    FHtmlElementNumber := Value;
end;
procedure TSiteStoredXssFinderPageField.SetHtmlFormNumber(const Value: Integer);
begin
    FHtmlFormNumber := Value;
end;
end.
procedure TSiteReflectedXssFinder.TimerOnTimer(Sender: TObject);
var
    Page, TaskProblem: TADOQuery;
    URI: TIdURI;
    I: Integer;
    URL: string;
    L: TStringList;
begin
    if URLQueue.Count = 0 then
    begin
        Page := GetQuery('select top 1 * from Page where ForChecking = True and Visited
= False and TaskID = ' + IntToStr(TaskID) + ' order by ID asc');
        try
            if ((Page.RecordCount = 0) and (TotalPageCount > 0)) then
            begin
                SetState(snsComplete);
                Exit;
            end;
            try
                FCurrentPageID := Page.FieldByName('ID').AsInteger;
                Page.Edit;
                Page.FieldByName('Visited').AsBoolean := True;
                Page.Post;
                Inc(FVisitedPageCount);
                for I := 1 to FTaskProblem.RecordCount do
                begin
                    FTaskProblem.RecNo := I;
                    L := InjectXssInUrl(Page.FieldByName('URL').AsString,
FTaskProblem.FieldByName('Code').AsString);
```

```
try
                        URLQueue.AddStrings(L);
                    finally
                        L.Free;
                    end;
                end;
            except
                on E: Exception do
                begin
                    Page.Edit;
                    Page.FieldByName('ErrorMessage').AsString := E.Message;
                    Page.FieldByName('Error').AsBoolean := True;
                    Page.Post;
                end;
            end;
        finally
            Page.Free;
        end;
    end
    else
    begin
        try
            URL := URLQueue.Strings[0];
            URLQueue.Delete(0);
            WebBrowser.Navigate(URL);
            if Assigned(OnStatusTextChange) then
                OnStatusTextChange(Self, URL);
        finally
            Timer.Enabled := False;
        end;
    end;

end;

procedure TSiteReflectedXssFinder.WebBrowserOnDocumentComplete(ASender: TObject; const
pDisp: IDispatch; const URL: OleVariant);
var
    SubStr, HTML: WideString;
    I: Integer;
    SubLen, SrcLen: Integer;
begin
    HTML := WideLowerCase((WebBrowser.Document as IHTMLDocument2).body.outerHTML);
    for I := 1 to FTaskProblem.RecordCount do
    begin
        FTaskProblem.RecNo := I;
        SubStr := WideLowerCase(FTaskProblem.FieldByName('Code').AsString);
        if Pos(SubStr, HTML) > 0 then
        begin
            PageCheckAdd(FCurrentPageID,
FTaskProblem.FieldByName('ProblemID').AsInteger, 1, URL);
            Inc(FProblemCount);
            if Assigned(OnStatusTextChange) then
                OnStatusTextChange(Self, 'Найдена уязвимость: ' + URL);
        end;
    end;
    Timer.Enabled := True;
end;
```

```
implementation
{ TSiteMapCreator }
constructor TSiteMapCreator.Create(Connection: TADOConnection);
begin
    inherited Create(Connection);
end;
destructor TSiteMapCreator.Destroy;
begin
    inherited;
end;

function TSiteMapCreator.PageAdd(URL: string; Title: string): Integer;
var
    Q: TADOQuery;
begin
    Result := 0;
    Q := GetQuery('select * from Page where TaskID = ' + IntToStr(TaskID) + ' and URL =
' + QuotedStr(URL));
    try
        if Q.RecordCount = 0 then
        begin
            Q.Insert;
            Q.FieldByName('TaskID').AsInteger := TaskID;
            Q.FieldByName('RegistrationDate').AsDateTime := Now;
            Q.FieldByName('Title').AsString := Title;
            Q.FieldByName('URL').AsString := URL;
            Q.FieldByName('Visited').AsBoolean := False;
            Q.FieldByName('Checked').AsBoolean := False;
            Q.Post;
            Inc(FTotalPageCount);
        end
        else
        begin
            Q.Edit;
            Q.FieldByName('Title').AsString := Title;
            Q.Post;
        end;

        Result := Q.FieldByName('ID').AsInteger;
    finally
        Q.Free;
    end;
end;

procedure TSiteMapCreator.TimerOnTimer(Sender: TObject);
var
    Page: TADOQuery;
begin
    try
        Page := GetQuery('select top 1 * from Page where Visited = 0 and TaskID = ' +
IntToStr(TaskID));
        try
            if (Page.RecordCount = 0) and (TotalPageCount > 0) then
                SetState(snsComplete);
            try
                WebBrowser.Navigate(Page.FieldByName('URL').AsString);
```

```
 except
                on E: Exception do
                begin
                    Page.Edit;
                    Page.FieldByName('ErrorMessage').AsString := E.Message;
                    Page.FieldByName('Error').AsBoolean := True;
                    Page.Post;
                    Inc(FVisitedPageCount);
                end;
            end;
        finally
            Page.Free;
        end;
    finally
        Timer.Enabled := False;
    end;
end;

procedure TSiteMapCreator.WebBrowserOnDocumentComplete(ASender: TObject; const pDisp:
IDispatch; const URL: OleVariant);
var
    Links: IHTMLElementCollection;
    Link: IHTMLElement;
    LinkUrl, LinkText: string;
    I, PageID: Integer;
    Page: TADOQuery;
begin
    try
        PageID := PageAdd(URL, (WebBrowser.Document as IHTMLDocument2).Title);
        Page := PageGet(PageID);
        try
            Page.Edit;
            Page.FieldByName('UsedHtmlForm').AsBoolean := (WebBrowser.Document as
IHTMLDocument2).Forms.length > 0;
            Page.Post;
            Links := (WebBrowser.Document as IHTMLDocument2).all.tags('a') as
IHTMLElementCollection;
            for I := 0 to Links.length - 1 do
            begin
                Link := Links.item(OleVariant(I), varEmpty) as IHTMLElement;
                LinkUrl := Link.getAttribute('href', varEmpty);
                LinkText := Link.innerText;
                if TestLink(LinkUrl) then
                begin
                    if (PageAdd(LinkUrl, LinkText) > 0) then
                    begin
                        if Assigned(OnStatusTextChange) then
                            OnStatusTextChange(Self, LinkText + ' (' + LinkUrl + ')');
                    end;
                end;
                if State = snsCancel then
                    Exit;
                Application.ProcessMessages;
            end;

            Page.Edit;
            Page.FieldByName('Visited').AsBoolean := True;
```

```
Page.FieldByName('ForChecking').AsBoolean := True;
            Page.Post;
            Inc(FVisitedPageCount);
            if Assigned(OnStatusTextChange) then
                OnStatusTextChange(Self, URL);
        finally
            Page.Free
        end;
    except
    end;
    Timer.Enabled := True;
end;

procedure TSiteMapCreator.WebBrowserOnFileDownload(ASender: TObject; ActiveDocument:
WordBool; var Cancel: WordBool);
begin
    Cancel := True;
    Timer.Enabled := True;
end;

procedure TSiteMapCreator.WebBrowserOnNavigateError(ASender: TObject; const pDisp:
IDispatch; const URL, Frame, StatusCode: OleVariant; var Cancel: WordBool);
begin
    Cancel := True;
    Timer.Enabled := True;
end;

procedure TSiteMapCreator.CreateMap;
begin
    FTotalPageCount := 0;
    FVisitedPageCount := 0;
    SetState(snsFree);
    TestRequirement;
    SetState(snsExec);
    PageDeleteAll;
    WebBrowser.Navigate(BaseURL);
end;

end.
constructor TSiteLogin.Create(Connection: TADOConnection);
begin
  inherited Create(Connection);
end;
destructor TSiteLogin.Destroy;
begin
  inherited;
end;
procedure TSiteLogin.TimerOnTimer(Sender: TObject);
begin
end;
procedure TSiteLogin.WebBrowserOnDocumentComplete(ASender: TObject; const pDisp:
IDispatch; const URL: OleVariant);
var
    FormList, InputList, ButtonList: IHTMLElementCollection;
```

```
 Form, Input, Button: IHTMLElement;
    I, J: Integer;
begin
    if Task.FieldByName('AuthorizationURL').AsString = URL then
    begin
        FormList := (WebBrowser.Document as IHTMLDocument2).Forms;
        InputList := (WebBrowser.Document as IHTMLDocument2).all.tags('input') as
IHTMLElementCollection;
        ButtonList := (WebBrowser.Document as IHTMLDocument2).all.tags('button') as
IHTMLElementCollection;
        for I := 0 to FormList.length - 1 do
        begin
            Form := FormList.item(OleVariant(I), varEmpty) as IHTMLElement;
            if (Task.FieldByName('LoginFormName').AsString = Form.getAttribute('name',
0)) or
              (Task.FieldByName('LoginFormID').AsString = Form.getAttribute('id', 0)) or
              (Task.FieldByName('LoginFormClass').AsString = Form.getAttribute('class',
0)) then
            begin
                for J := 0 to InputList.length - 1 do
                begin
                    Input := InputList.item(OleVariant(J), varEmpty) as IHTMLElement;
                    if Input.getAttribute('type', 0) = 'text' then
                    begin
                        if (Form.contains(Input)) then
                        begin
                            if (Task.FieldByName('LoginFieldName').AsString =
Input.getAttribute('name', 0)) or
                              (Task.FieldByName('LoginFieldID').AsString =
Input.getAttribute('id', 0)) then
                            begin
                                Input.setAttribute('value',
Task.FieldByName('LoginFieldValue').AsString, 0);
                            end;
                        end;
                    end;

                    if Input.getAttribute('type', 0) = 'password' then
                    begin
                        if (Form.contains(Input)) then
                        begin
                            if (Task.FieldByName('PasswordFieldName').AsString =
Input.getAttribute('name', 0)) or
                              (Task.FieldByName('PasswordFieldID').AsString =
Input.getAttribute('id', 0)) then
                            begin
                                Input.setAttribute('value',
Task.FieldByName('PasswordFieldValue').AsString, 0);
                            end;
                        end;
                    end;
                end;

                for J := 0 to ButtonList.length - 1 do
                begin
                    Button := ButtonList.item(OleVariant(J), varEmpty) as IHTMLElement;
                    if (Form.contains(Button)) then
```

```
                    begin
                        if Button.getAttribute('type', 0) = 'submit' then
                        begin
                            Button.click;
                        end;
                    end;
                end;
            end;
        end;
    end
    else
    begin
        if Task.FieldByName('AuthorizationCompleteURL').AsString = URL then
        begin
            SetState(snsComplete);
        end
        else
        begin
            SetState(snsError);
        end;
    end;
end;

procedure TSiteLogin.WebBrowserOnFileDownload(ASender: TObject; ActiveDocument:
WordBool; var Cancel: WordBool);
begin
    Cancel := True;
    Timer.Enabled := True;
end;

procedure TSiteLogin.WebBrowserOnNavigateError(ASender: TObject; const pDisp: IDispatch;
const URL, Frame, StatusCode: OleVariant; var Cancel: WordBool);
begin
    Cancel := True;
    Timer.Enabled := True;
end;
procedure TSiteLogin.Login;
begin
    SetState(snsFree);
    TestRequirement;
    SetState(snsExec);
    if Task.FieldByName('Authorization').AsBoolean = False then
    begin
        SetState(snsComplete);
        Exit;
    end;
    WebBrowser.Navigate(Task.FieldByName('AuthorizationURL').AsString);
end;

end. Form, Input, Button: IHTMLElement;
    I, J: Integer;
begin
    if Task.FieldByName('AuthorizationURL').AsString = URL then
    begin
        FormList := (WebBrowser.Document as IHTMLDocument2).Forms;
        InputList := (WebBrowser.Document as IHTMLDocument2).all.tags('input') as
IHTMLElementCollection;
```

```
ButtonList := (WebBrowser.Document as IHTMLDocument2).all.tags('button') as
IHTMLElementCollection;
        for I := 0 to FormList.length - 1 do
        begin
            Form := FormList.item(OleVariant(I), varEmpty) as IHTMLElement;
            if (Task.FieldByName('LoginFormName').AsString = Form.getAttribute('name',
0)) or
              (Task.FieldByName('LoginFormID').AsString = Form.getAttribute('id', 0)) or
              (Task.FieldByName('LoginFormClass').AsString = Form.getAttribute('class',
0)) then
            begin
                for J := 0 to InputList.length - 1 do
                begin
                    Input := InputList.item(OleVariant(J), varEmpty) as IHTMLElement;
                    if Input.getAttribute('type', 0) = 'text' then
                    begin
                        if (Form.contains(Input)) then
                        begin
                            if (Task.FieldByName('LoginFieldName').AsString =
Input.getAttribute('name', 0)) or
                              (Task.FieldByName('LoginFieldID').AsString =
Input.getAttribute('id', 0)) then
                            begin
                                Input.setAttribute('value',
Task.FieldByName('LoginFieldValue').AsString, 0);
                            end;
                        end;
                    end;
```

ПРИЛОЖЕНИЕ Б

Акт внедрения в опытную эксплуатацию результатов бакалаврской работы в ООО НПП «ДосЛаб»

ООО НПП «ДосЛаб»

УТВЕРЖДАЮ

Директор

М.П. _____ Б. Р. Досмухамедов

«10» января 2018 г.

Акт

о внедрении в опытную эксплуатацию результатов бакалаврской работы

студента 4 курса направления «Информационная безопасность»

З. А. Носирова

1. Предмет внедрения: результат бакалаврской работы по теме «Разработка схемы защиты от межсайтового скриптинга».

2. Состав внедряемых результатов:

- программное обеспечение для обнаружения XSS-уязвимостей и выдачи рекомендаций по их устранению.

3. Место внедрения: ООО НПП «ДосЛаб».

4. Анализ работоспособности внедренных результатов бакалаврской работы подтверждает высокий уровень защиты информационной системы от актуальных компьютерных атак вида «Межсайтовый скриптинг».

Вывод: программное обеспечение для обнаружения XSS-уязвимостей и выдачи рекомендаций по их устранению успешно внедрен в информационную систему организации.

Проектный руководитель _____ Кравченко К. А.

(должность) (подпись) (Ф.И.О.)

ПРИЛОЖЕНИЕ В

Свидетельство о государственной регистрации программы для ЭВМ

РОССИЙСКАЯ ФЕДЕРАЦИЯ

СВИДЕТЕЛЬСТВО

о государственной регистрации программы для ЭВМ

№ 2018610645

ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ ДЛЯ ОБНАРУЖЕНИЯ XSS-УЯЗВИМОСТЕЙ И ВЫДАЧИ РЕКОМЕНДАЦИЙ ПО ИХ УСТРАНЕНИЮ

Правообладатели: ***Носиров Зафаржон Амрулоевич (RU), Ажмухамедов Искандар Маратович (RU), Марьенков Александр Николаевич (RU)***

Авторы: ***Носиров Зафаржон Амрулоевич (RU), Ажмухамедов Искандар Маратович (RU), Марьенков Александр Николаевич (RU)***

Заявка № **2017661927**
Дата поступления **21 ноября 2017 г.**
Дата государственной регистрации
в Реестре программ для ЭВМ ***15 января 2018 г.***

Руководитель Федеральной службы по интеллектуальной собственности

Г.П. Ивлиев

ПРИЛОЖЕНИЕ Г

Сформированный отчет

Отчет по выполнению задачи поиска XSS уязвимостей

Общие сведения задачи

Описание задания	
Дата регистрации	02.02.2018 14:28:05
Название	http://www.insecurelabs.org/
URL	http://www.insecurelabs.org/
Статус	Полный цикл тестирования
Итого уязвимостей	8

Список проверяемых уязвимостей	
1	Полное отсутствие фильтрации на сервере
2	Инъекция с помощью изображения путем встраивания протокола JavaScript
3	Отсутствие кавычек и точки с запятой
4	Чувствительная к регистру система фильтрации

Результаты по каждой странице

№	Дата регистраци	Названи е	URL	Просм отрено	Для проверки	Пров ер
1	02.02.2018 14:29:26	About	http://www.insecurelabs.org/Home/About	+	+	
2	02.02.2018 14:29:26	Agenda	http://www.insecurelabs.org/Talk	+	+	
		Обнаруженные уязвимости				
		1	Инъекция с помощью изображения путем встраивания протокола JavaScript	XSS через форму ввода (POST)		
		2	Полное отсутствие фильтрации на	XSS через форму ввода		
		Всего обнаружено на странице: 2				
3	02.02.2018 14:29:26	Home	http://www.insecurelabs.org/	+	+	
4	02.02.2018 14:29:29	NWC - Details	http://www.insecurelabs.org/Speaker/Details/Brill%20Grates	+	+	
5	02.02.2018 14:29:29	NWC - Details	http://www.insecurelabs.org/Speaker/Details/Geoff%20Sutherland	+	+	
6	02.02.2018 14:29:29	NWC - Details	http://www.insecurelabs.org/Speaker/Details/Greg%20Old	+	+	
7	02.02.2018 14:29:29	NWC - Details	http://www.insecurelabs.org/Speaker/Details/Guth%20Scottrie	+	+	
8	02.02.2018 14:29:29	NWC - Details	http://www.insecurelabs.org/Speaker/Details/Hansel%20Scottsman	+	+	
9	02.02.2018 14:29:29	NWC - Details	http://www.insecurelabs.org/Speaker/Details/Jeff%20Geoff	+	+	
10	02.02.2018 14:29:29	NWC - Details	http://www.insecurelabs.org/Speaker/Details/Kevin%20Hennley	+	+	
11	02.02.2018 14:29:29	NWC - Details	http://www.insecurelabs.org/Speaker/Details/Poppy%20Marendieck	+	+	
		Обнаруженные уязвимости				
		1	Инъекция с помощью изображения путем встраивания протокола JavaScript	XSS через форму ввода (POST)		
		2	Полное отсутствие фильтрации на	XSS через форму ввода		
		Всего обнаружено на странице: 2				
13	02.02.2018 14:29:28	NWC - View	http://www.insecurelabs.org/Talk/Details/2		+	
		Обнаруженные уязвимости				
		1	Инъекция с помощью изображения путем встраивания протокола JavaScript	XSS через форму ввода (POST)		
		4	Чувствительная к регистру система	XSS через форму ввода		
		Всего обнаружено на странице: 2				
14	02.02.2018 14:29:28	NWC - View	http://www.insecurelabs.org/Talk/Details/3		+	
		Обнаруженные уязвимости				
		1	Инъекция с помощью изображения путем встраивания протокола JavaScript	XSS через форму ввода (POST)		

		3	Отсутствие кавычек и точки с запятой		XSS через форму ввода	
		Всего обнаружено на странице: 2				
15	02.02.2018 14:29:28	NWC - View	http://www.insecurelabs.org/Talk/Details/4		+	
		Обнаруженные уязвимости				

		Всего обнаружено на странице: 0				
16	02.02.2018 14:29:28	NWC - View	http://www.insecurelabs.org/Talk/Details/5	+	+	
		Обнаруженные уязвимости				
		Всего обнаружено на странице: 0				
17	02.02.2018 14:29:28	NWC - View	http://www.insecurelabs.org/Talk/Details/6		+	
		Обнаруженные уязвимости				
		Всего обнаружено на странице: 0				
18	02.02.2018 14:29:28	NWC - View	http://www.insecurelabs.org/Talk/Details/7		+	
		Всего обнаружено на странице: 0				
19	02.02.2018 14:29:26	Speakers	http://www.insecurelabs.org/Speaker		+	
20	02.02.2018 14:29:26	Tickets	http://www.insecurelabs.org/Home/Tickets		+	

Итоги поиска

№	Уязвимость	Найдено (кол-во)	Код проверки	Описание уязвимости	Рекомендации
1	Полное отсутствие фильтрации на сервере	4	<SCRIPT SRC="xss.js"></SCRIPT>	Данная уязвимость возникает в тех ситуациях, когда данные введенные пользователем выводятся без надлежащей фильтрации выводятся в тексте сгенерированного html документа. К примеру, может быть ситуация, когда данные, отправленные одним пользователем без фильтрации выводятся другим пользователям. Типичной системой такого	Необходимо использовать следующее: кодирование управляющих HTML-символов, JavaScript, CSS.
2	Инъекция с помощью изображения путем встраивания протокола	2	<IMG SRC="javascript:alert('XSS');">	Инъекция с помощью изображения путем встраивания протокола JavaScript. А если учесть возможности этого языка и стиль JSFuck. Оперируя только символами (,), [,], +, ! можно получить доступ ко всему алфавиту и сконструировать почти любую инъекцию	Кодирование управляющих HTML-символов, JavaScript, CSS и URL перед отображением в браузере.
3	Отсутствие кавычек и точки с запятой	1	<IMG SRC=javascript:alert('XSS')>	Уязвимость, является ситуация, когда часть HTTP GET запроса выводится на этой же html странице тому же пользователю без надлежащей фильтрации. Как правило – это ситуации, когда без надлежащей фильтрации выводится идентификатор сессии или другие	Обеспечение безопасности cookies, которая может быть реализована путём ограничения домена и пути для принимаемых
4	Чувствительная к регистру система фильтрации	1	<IMG SRC=JaVaScRiPt:alert('XSS')>	Данная уязвимость возникает в тех ситуациях, когда данные введенные пользователем выводятся без надлежащей фильтрации выводятся в тексте сгенерированного html документа.	Использование заголовка Content Security Policy, позволяющего задавать список, в который заносятся

ПРИЛОЖЕНИЕ Д

Материалы на электронном носителе