ISSN 2500-316X (Online)

https://doi.org/10.32362/2500-316X-2020-8-6-9-33

УДК 004.056.53

Модели и сценарии реализации угроз для интернет-ресурсов

С.А. Лесько

МИРЭА – Российский технологический университет, Москва 119454, Россия
@Автор для переписки, e-mail: lesko@mirea.ru

Для облегчения обнаружения различных уязвимостей существует множество различных инструментов (сканеров), которые могут помочь в анализе безопасности веб-приложений и облегчить разработку их защиты. Но данные инструменты в большинстве своём могут только идентифицировать проблемы и не способны устранять их. Поэтому знания разработчика безопасности являются ключевым фактором при построении безопасного веб-ресурса. Для устранения проблем безопасности приложений разработчики должны знать все пути и векторы проведения различных атак для того, чтобы иметь возможность разрабатывать различные механизмы защиты. В данном обзоре рассмотрены две из наиболее опасных уязвимостей в сфере веб-технологий: SQL-инъекции и XSS-атаки (межсайтовый скриптинг, англ. CrossSite Scripting - XSS), а также конкретные случаи и примеры их применения, различные подходы к выявлению уязвимостей в приложениях и предотвращению угроз. Межсайтовый скриптинг также, как и атаки SQL-инъекций, связан с проверкой входных данных. Механизмы этих атак очень схожи, но в XSS-атаках жертвой является сам пользователь, а в атаках SQL-инъекцией – сервер базы данных (БД) веб-приложения. При XSS-атаках вредоносный контент доставляется пользователям с использованием языка программирования, выполняемого на стороне клиента, такого как JavaScript, а при использовании SQL-инъекции используется язык запросов к БД SQL. При этом XSS-атаки, в отличие от SQL-инъекций, наносят вред исключительно клиентской стороне, оставляя сервер приложения в рабочем состоянии. Разработчики должны разрабатывать защиту, как для серверных составляющих, так и для клиентской части веб-приложения.

Ключевые слова: компьютерная и сетевая безопасность, модели и сценарии угроз, интернет-ресурсы, SQL-инъекции, XSS-атаки (межсайтовый скриптинг).

Для цитирования: Лесько С.А. Модели и сценарии реализации угроз для интернет-ресурсов. *Российский технологический журнал*. 2020;8(6):9-33. https://doi.org/10.32362/2500-316X-2020-8-6-9-33

Models and scenarios of implementation of threats for internet resources

Sergey A. Lesko

MIREA – Russian Technological University, Moscow 119454, Russia
@Corresponding author, e-mail: lesko@mirea.ru

To facilitate the detection of various vulnerabilities, there are many different tools (scanners) that can help analyze the security of web applications and facilitate the development of their protection. But these tools for the most part can only identify problems, and they are not capable of fixing them. Therefore, the knowledge of the security developer is a key factor in building a secure Web resource. To resolve application security problems, developers must know all the ways and vectors of various attacks in order to be able to develop various protection mechanisms. This review discusses two of the most dangerous vulnerabilities in the field of Web technologies: SQL injections and XSS attacks (cross-site scripting – XSS), as well as specific cases and examples of their application, as well as various approaches to identifying vulnerabilities in applications and threat prevention. Cross-site scripting as well as SQL-injection attacks are related to validating input data. The mechanisms of these attacks are very similar, but in the XSS attacks the user is the victim, and in the SQL injection attacks, the database server of the Web application. In XSS attacks, malicious content is delivered to users by means of a client-side programming language such as JavaScript, while using SQL injection, the SQL database query language is used. At the same time, XSS attacks, unlike SQL injections, harm only the client side leaving the application server operational. Developers should develop security for both server components and the client part of the web application.

Keywords: computer and network security, threat models and scenarios, Internet resources, SQL injections, XSS attacks (crossite scripting).

For citation: Lesko S.A. Models and scenarios of implementation of threats for internet resources. *Rossiiskii tekhnologicheskii zhurnal = Russian Technological Journal.* 2020;8(6):9-33 (in Russ.). https://doi.org/10.32362/2500-316X-2020-8-6-9-33

Введение

Перед обсуждением безопасности веб-ресурсов или атак на них, важно понять эволюцию веб-приложений, их возрастающую сложность и первостепенную важность, которую они играют сегодня в жизни более чем миллиарда людей.

Появление веб-приложений первого поколения было строго ограничено их способностью предоставлять больше информации, чем брошюра, которую вы могли бы получить по почте. Статический HTML был предоставлен как инструмент для отображения изображений и текстовой информации.

По мере того, как доступ к Интернету становился все более повсеместным, потребности пользователей, обращающихся к веб-приложениям, также возрастали. В результате веб-приложения эволюционировали, обеспечив пользователям такие удобства, как поиск информации, её публикацию и загрузку.

CGI (англ. Common Gateway Interface – общий интерфейс шлюза) был первым скачком вперед в этом прогрессе. CGI предоставил пользователям возможность взаимодействовать с веб-страницами путем отправки данных в формы. После отправки на сервер CGI сценарии обрабатывают эти данные и представляют сгенерированный HTML-код обратно конечному пользователю. Благодаря взаимодействию с конечными пользователями, CGI фактически стал одним из первых известных векторов атак на веб-приложения.

Разработчики веб-приложений не ограничились работой со сценариями CGI, вместо этого появлялись новые более развитые платформы и фреймворки. PHP, ASP.NET, J2EE, AJAX, Ruby and Rails и другие языки появились, включив в себя больше интерактивных возможностей, предоставив пользователям больше гибкости и мощности при управлении данными и рабочим процессом в веб-приложениях.

Обеспечение безопасности веб-ресурсов стало невероятно важным аспектом, поскольку информация, обрабатываемая веб-приложениями, стала критически важной для клиентов, организаций и стран.

Веб-приложения управляют широким спектром информации, включая финансовые данные, медицинские записи, номера социального страхования, интеллектуальную собственность и данные национальной безопасности. Веб-приложения должны обрабатывать эту информацию надежно, сохраняя при этом эффективность и доступность.

В целом под безопасностью веб-приложений можно понимать защиту конфиденциальности, целостности и доступности ресурсов в интернет-пространстве. Веб-службы и приложения сформировали новый ландшафт обмена информацией, который повысил производительность электронного бизнеса. Однако с другой стороны, также значительно возросло и количество киберугроз из-за растущей популярности веб-приложений.

Предпринимаются огромные усилия по смягчению последствий этих атак с помощью различных механизмов безопасности в виде сканеров, систем обнаружения вторжений, устройств шифрования и брандмауэров веб-приложений. Атаки на веб-приложения могут включать в себя неправильные конфигурации безопасности, нарушение аутентификации и управления сеансом или другие проблемы.

Однако одни из наиболее опасных и распространенных атак на веб-приложения используют уязвимости, связанные с неправильной проверкой или фильтрацией именно ненадежных входов, что приводит к инъекции вредоносного скрипта или кода языка, специфичного для домена. Атаки данного типа могут включать в себя, например, Cross-Site Scripting (XSS), SQL-инъекции и другие.

В течение последних нескольких лет данные атаки возглавляют списки наиболее опасных уязвимостей, опубликованных в OWASP, MITRE и другими подобными организациями.

Рассмотрим пример популярного проекта «Top-10» OWASP, целью которого является повышение осведомленности о методах защиты веб-приложений, а также выявление некоторых из наиболее важных рисков, с которыми могут столкнуться разработчики и

пользователи. В трех последних списках «Top-10» 2010, 2013, 2017 годов в пятерке лучших позиций доминируют различные инъекционные атаки.

В то же время злоумышленники находят новые пути, чтобы обойти защитные механизмы с использованием различных методов, несмотря на многочисленные вводимые меры противодействия. Например, уже к 2013 году было предложено более 20 средств защиты от атак SQL-инъекциями. С тех пор данное число удвоилось, в то время как исследователи указывают, что в последние годы и количество атак SQL-инъекциями неуклонно возрастает.

В данном литературном обзоре, основанном на классификации атак из списка OWASP «TOP-10», будут рассмотрены следующие детали, связанные с угрозами веб-ресурсов:

1) несколько наиболее опасных по версии OWASP векторов атак;

2) методы и сценарии проведения данных атак;

3) наиболее действенные методы практической защиты веб-приложений и веб-сервисов.

1. Внедрение вредоносного кода (Code injection)

Инъекции кода являются одними из наиболее важных и распространённых типов атак, направленных на веб-ресурсы. Уже несколько лет подряд именно данный тип атак занимает самую верхнюю строчку в списках OWASP.

Атаки данного типа имеют большое количество различных вариаций (например, инъекция в командах ОС, XPath-инъекция, LDAP-инъекция), но на сегодняшний момент наиболее используемой и действенной из них является SQL-инъекция, применяемая злоумышленниками на протяжении многих лет. В статье [1] исследователи, проанализировав множество Web-атак, приводят именно SQL-инъекции как наиболее популярный тип атак.

Главная идея, заложенная в SQL-инъекции, заключается в том, что злоумышленник при помощи полей ввода данных, расположенных на веб-ресурсе, внедряет свои зловредные SQL-команды. Если в коде веб-приложения не предусмотрена фильтрация вводимых данных, то на веб-сервере становится возможным запустить SQL-команды и выполнять прямые запросы к внутренней базе данных (БД) в обход сетевых средств защиты.

При использовании атак SQL-инъекциями злоумышленник имеет возможность, к примеру:

– заполучить различные таблицы с данными (например, таблицы, содержащие идентификационные данные зарегистрированных пользователей);

– изменять записи в данных таблицах;

– полностью удалить базу данных, таким образом полностью вывести из строя веб-ресурс.

Вредоносные SQL-операторы могут быть введены в уязвимое приложение с использованием множества различных механизмов и сценариев ввода. В работе [2] были классифицированы различные сценарии. Рассмотрим наиболее распространенные из них.

1) Инъекция через пользовательский ввод: в этом случае злоумышленники вводят команды SQL, предоставляя соответствующим образом, созданный пользовательский ввод. Веб-приложение может читать пользовательский ввод несколькими способами на основе среды, в которой развертывается приложение.

В большинстве атак SQL-инъекциями, ориентированными на веб-приложения, пользовательские входные данные обычно поступают из форм, отправляемых в него через

HTTP GET или POST-запросы. Веб-приложения обычно имеют доступ к пользовательскому вводу, содержащемуся в этих запросах, поскольку они будут обращаться к любой другой переменной в среде.

2) Инъекция через файлы Cookie: Cookie – это файлы, содержащие информацию о каком-либо состоянии, создаваемые веб-приложениями и хранящиеся на клиентской машине. Когда клиент возвращается обратно на веб-ресурс, файлы Cookie могут использоваться для восстановления информации о состоянии клиента (например, авторизован он или нет).

Поскольку клиент имеет контроль над хранением файлов Cookie, злоумышленник может подменить их содержимое. Если веб-приложение использует содержимое файла Cookie для создания некоторых SQL-запросов, злоумышленник может легко отправить атаку, встроив ее в файл Cookie.

3) Инъекция через переменные сервера: переменные сервера представляют собой набор переменных, которые содержат HTTP заголовки, заголовки сетевого уровня и переменные среды. Веб-приложения используют эти переменные сервера различными способами, такими как статистика использования журналов и определение тенденций просмотра. Если эти переменные регистрируются в базе данных без какой-либо «очистки», это может привести к уязвимости SQL-инъекциями.

Поскольку злоумышленники могут подделывать значения, помещенные в заголовки HTTP, таким образом они могут использовать эту уязвимость, подставив вредоносный SQL-запрос непосредственно в заголовки.

4) Инъекция второго порядка: во вторичных инъекциях злоумышленники вводят вредоносные данные в систему или базу данных, чтобы косвенно вызвать SQL-инъекцию при более позднем использовании этих данных. Цель такого рода атаки существенно отличается от обычной (т.е. первого порядка) инъекционной атаки.

Инъекции второго порядка не пытаются вызвать атаку, когда вредоносный ввод первоначально достигает базы данных. Вместо этого злоумышленники полагаются на знания о том, где будут впоследствии использоваться входные данные, и создают атаки таким образом, чтобы они происходили во время такого использования.

В работе [3] исследователи также рассмотрели сценарии и дополнили их некоторыми примерами. Для уточнения приведем классический пример инъекционной атаки второго порядка. Например, злоумышленник регистрируется на сайте, используя имя пользователя, такое как «admin’ --». Приложение должным образом экранирует одинарную кавычку во входных данных, прежде чем сохранить их в базе данных, предотвращая потенциально вредоносный эффект.

Далее злоумышленник, пытается выполнить обход входа на ресурс через форму восстановления пароля. На данном этапе пользователю требуется выполнить операцию по изменению своего пароля, которая обычно включает следующие действия:

1. Проверку того, что пользователь знает текущий (старый) пароль.

2. Изменение пароля, если проверка выполнена успешно.

Злоумышленник, может предположить, что данная форма веб-приложения может создать команду SQL следующим образом:

queryString="UPDATE users SET password=’" + newPassword + "’ WHERE userName=’" + userName + "’ AND password=’" + oldPassword + "’",

где newPassword и oldPassword – это новый и старый пароли, соответственно, а userName – это имя пользователя, вошедшего в систему. Злоумышленник вводит в поле имени заведомо зарегистрированного пользователя «admin' --» и новый пароль.

Поэтому строка запроса, отправленная в базу данных, будет выглядеть следующим образом (предположим, что newPassword и oldPassword это «newpwd» и «oldpwd»):

UPDATE users SET password='newpwd' WHERE userName= 'admin'--' AND password='oldpwd'.

Поскольку «--» – это оператор однострочного комментария в SQL, то все, что расположено после него, будет игнорироваться базой данных. Поэтому результатом этого запроса является то, что база данных изменяет пароль администратора («admin») на значение, указанное злоумышленником, не учитывая при этом старый пароль.

Инъекции второго порядка особенно трудно обнаружить и предотвратить, потому что путь выполнения инъекции отличается от пути, где атака на самом деле проявляется.

Разработчик может предусмотреть защиту от известных путей SQL-инъекций, экранируя, проверяя тип и фильтруя входные данные, которые приходят от пользователя и предполагать, что веб-ресурс защищён. Но позднее, когда эти данные будут использоваться в другом контексте или для построения запроса другого типа, такие ранее обработанные (дезинфицированные) входные данные могут также привести к атаке инъекцией. Поэтому подробные знания о том, как дальше будут использоваться введённые данные, позволяют разработчику предусмотреть и защиту от инъекций второго порядка.

1.1. Анализ процедуры проведения SQL-инъекций

Методы атаки SQL-инъекциями могут быть различными. В работе [4] авторы исследовали процедуры их проведения. Процедура SQL-инъекции обычно делится на следующие пять шагов, как показано на рис. 1.

Рис. 1. Процедура SQL-инъекции.

1) Поиск точек инъекции.

Точка инъекции – это веб-страницы, имеющие уязвимости к инъекциям. На статических URL страницах, заканчивающихся на «.html» или «.shtml», инъекции производиться не могут, так как эти страницы являются обычными статическими, не имеющими доступа к базе данных. Инъекции могут быть проведены на ресурсах, у которых есть некое хранилище информации, то есть база данных, где хранится вся информация с ресурса.

Инъекция может производиться только на динамических страницах с помощью знака вопроса «?». Это связано с тем, что информация о странице генерируется путем получения данных из базы в соответствии с различными параметрами запроса. Следовательно, доступ к базе данных должен поддерживаться самой базой данных.

Поскольку HTTP-запрос может выполняться двумя способами: либо через метод GET, либо через POST, то и SQL-инъекция также имеет два способа выполнения: инъекция, проводимая через метод GET, и инъекция через метод POST, соответственно.

Можно определить, выполняется ли какая-либо SQL-инъекция в соответствии с информацией, возвращаемой браузером, путем ввода, например, в адресную строку браузера некоторых специальных команд, таких как «and 1=1» и «and 1=2». Таким образом, можно попробовать обнаружить точку инъекции.

2) Получение информации об используемой системы управления базой данных (СУБД)

Получение информации, такой как тип используемой СУБД, является важной частью при проведении SQL-инъекции. Узнать тип применяемой СУБД можно, рассматривая сообщения об ошибках, генерируемые из-за заведомо неправильно сформированного и отправленного SQL-запроса со специальными структурами.

В статье [5] был предложен следующий пример для определения используемой СУБД. Предположим, что у нас есть некий URL-адрес *http://www.site.com/show.php?id=6.* Если в конце данного адреса добавить одинарную кавычку, то есть *http://www.site.com/show.php?id=6'*, это будет считаться заведомо неверным значением, отправляемым на сервер и тип СУБД можно оценить в соответствии с возвращенной ошибкой. Если имеется уязвимость и не настроен обработчик ошибок, то появится ошибка примерно следующего содержания: «*mysql_fetch_assoc(): supplied argument is not a valid MySQL result resource*». Если же ошибки обрабатываются, то о присутствии уязвимости может свидетельствовать пустой результат. Первые два этапа являются подготовительными перед проведением последующей SQL-инъекции.

3) Построение предположений о таких сведениях, как имена таблиц, имена полей, имена пользователей и пароли базы данных.

Существуют правила для имен таблиц и имен полей, хранящихся в базе данных. Например, информация о других базах данных, доступных на сервере MySQL, хранится в базе данных «Information_Schema», которая является системной БД и поставляется вместе с различными СУБД, например, MySQL. Данная уязвимость описана в статье [6] и подразумевает следующие аспекты.

В современных версиях MySQL вся метаинформация относительно других баз данных и данных таблиц хранятся в таблице «SCHEMATA» и таблице «TABLE», соответственно. Более того, вся информация о столбцах хранится в таблице «COLUMNS».

Далее производится обращение к таблице «SHEMATA», из которой можно узнать название БД Web-ресурса. Например, запрос может выглядеть следующим образом:

http://www.site.com/show.php?id=-1 SELECT 1,SCHEMA_NAME,3,4 FROM INFORMATION_SCHEMA.SCHEMATA.

В ответ может быть возвращено следующее: information_schema, mysql, SiteDB. Из полученных названий БД можно сделать вывод, что БД, с которой необходимо производить некие действия, это «SiteDB». Далее таким же образом можно определить и названия таблиц, находящихся в выбранной БД.

Злоумышленники могут последовательно запрашивать длину и содержимое имени таблицы, имени колонок, имени пользователя и пароля в базе данных, создавая специальные запросы доступа к базе данных.

В частности, соответствующие тестовые SQL-запросы добавляются в конец обычного URL. Затем после отправки запроса рассуждения производятся в соответствии с ответом, возвращаемым веб-сервером.

Если после отправки было получено обычное содержимое веб-страницы, это значит, что имена таблиц или имена колонок, предположительно подобранные злоумышленниками, являются правильными, в противном случае они ошибочны. Затем злоумышленники должны продолжать и дальше строить догадки и подставлять тестовые запросы до тех пор, пока содержимое веб-страницы не будет успешно выявлено. Данный процесс построения догадок может быть реализован с помощью большого количества существующих инструментов инъекции в Интернете, и таким образом пароли пользователей рано или поздно будут взломаны.

4) Поиск входа в систему управления веб-ресурса.

В общем случае, зарегистрированные пользователи веб-системы могут получить доступ к веб-серверу, авторизуясь на ресурсе. Как правило, зарегистрированные пользователи веб-ресурса могут получить доступ к каким-либо действиям с веб-сервером только путем входа в систему. Интерфейс системы управления недоступен для обычных пользователей. Чтобы найти адрес для входа в данную систему, можно использовать различные средства сканирования для быстрого поиска, где все возможные адреса входа в систему будут рассматриваться последовательно.

После того, как станет известен адрес входа для системы управления, информация, полученная на предыдущих этапах (например, данные администратора), используется для входа в систему. Это означает, что теперь злоумышленник получает все полномочия администратора веб-ресурса и может производить действия от его лица.

5) Вторжение и уничтожение.

После входа в систему управления злоумышленники могут разрушить работу веб-ресурса, например, уничтожив различные информационные публикации, произведя искажение веб-страниц, загрузив на сервер какой-либо вирус, произведя кражу и утечку профилей пользователей. В итоге все данные будут фальсифицированы, что приведёт к различным неприятным последствиям.

1.2. Обнаружение атаки SQL-инъекцией.

Можно вручную проверять все формы на ресурсе на предмет уязвимости SQL-инъекциям, например, теми же подстановками конструкций вида «and 1=1» или подобных. Но недостатком ручного способа может быть недосмотр или невнимательность разработчика. Когда веб-приложение имеет множество различных кодов, очень сложно и неэффективно вручную находить уязвимости SQL-инъекций. Для этого существуют различные автоматизированные инструменты, которые являются более систематическими и тщательными. Они не понимают логику самого веб-приложения, но могут очень быстро протестировать множество потенциальных точек инъекции, которые человек не может отследить полностью и последовательно.

Для таких задач были разработаны некоторые коммерческие инструменты и инструменты с открытым исходным кодом, такие как Paros Proxy, IBM Rational AppScan, SQLiX,

HP Web Inspect и так далее. Хотя данные средства автоматического обнаружения могут не идентифицировать все существующие уязвимости, они обеспечивают достаточно высокую оценку безопасности веб-сайта, включая тестирование на уязвимости SQL-инъекциями.

Так или иначе, если злоумышленник нашёл способ провести инъекцию, то следует определить и выявить точку ее проведения, проанализировать, как ему это удалось и закрыть этот недостаток в будущем.

Существует два способа атаки SQL-инъекций. Одним из них является ручной способ встраивания некоторых ненормальных входных данных, другой – применение различных инструментов проведения инъекции.

Независимо от того, какой способ атаки используется, следы так или иначе будут оставлены в системе после успешно проведенной атаки SQL-инъекцией. В статье [7] авторы определили, как можно отследить была ли проведена атака или нет. Существует четыре эффективных способа оценить, была ли атакована веб-система с помощью SQL-инъекции.

1) Проверить, есть ли какие-либо «ненормальные» таблицы в базе данных. Могут быть созданы некоторые временные таблицы в базе данных после атак SQL-инъекций выполняемых с использованием таких программ, как HDSI и NBSI.

2) Проверить журналы базы данных. Если запустить службу управления журналом с помощью СУБД, все обращения в базу данных будут записываться в него. В частности, SQL-инъекция может быть обнаружена в журналах, если ложно выполненные SQL-запросы в них записаны.

3) Проверка журналов веб-сервера. На веб-сервере может сохраняться подробная информация, такая как IP-адреса, время обращения, к каким файлам зафиксировано обращение и способы доступа посетителя, записанные в журнал. При атаке путем внедрения SQL-кода обычно обращаются к файлам страниц, на которых существует точки инъекции. Размер файла журнала большой, поэтому можно судить о том, существует ли атака SQL-инъекцией путем проверки размера и содержимого файлов журнала.

4) Можно также определить, существуют ли какие-либо вторжения с помощью проверки таких сведений, как номер учетной записи системного администратора, состояние открытия порта, файлы, созданные в последнее время в системе, вирусы и журналы брандмауэра.

1.3. Рекомендации по устранению атак SQL-инъекцией.

Атаки SQL-инъекций, к сожалению, очень распространены, и это связано с двумя факторами:

– значительная распространенность уязвимостей SQL-инъекций;

– привлекательность цели (т.е. база данных обычно содержит все интересные и важные данные веб-приложения).

Весьма прискорбно, что существует так много успешных атак SQL-инъекцией, в то время как достаточно просто избежать данных уязвимостей на этапе разработки, в том числе используя автоматизированный контроль программного кода.

SQL-инъекции могут возникать, когда разработчики программного обеспечения создают динамические запросы к базе данных, включая в них вводимые пользователем данные.

Избежать ошибок SQL-инъекций достаточно просто. При этом разработчики должны:

а) либо прекратить писать динамические запросы;

б) либо запретить ввод данных, предоставленных пользователем, который содержит вредоносный SQL-код и влияет на логику выполненного запроса.

Далее представлен набор простых методов предотвращения уязвимостей SQL-инъекций, позволяющий избежать этих двух проблем. Данные методы могут использоваться практически на любом языке программирования и с любой базой данных.

Существуют и другие типы баз данных, такие как базы данных XML, которые могут иметь схожие проблемы (например, XPath и XQuery инъекции). Нижеприведенные методы также могут быть использованы и для их защиты.

Методы для первичной защиты:

1. Применение подготовленных выражений/связываемых переменных (англ. Prepared Statements).

2. Использование хранимых процедур.

3. Проверка входных данных по «белому списку» (англ. White List).

4. Экранирование всех входных данных пользователя.

Можно также рассмотреть несколько методов, применяемых в качестве дополнительной защиты совместно с каким-либо методом из списка выше:

1. Обеспечение минимальной привилегии.

2. Выполнение проверки «белого списка» в качестве вторичной защиты.

Применение подготовленных выражений. Использование подготовленных выражений со связываемыми переменными (так называемые параметризованные запросы) проще в написании и понимании, чем динамические запросы. Параметризованные запросы заставляют разработчика сначала определить весь SQL-код, а затем передать каждый параметр в запрос. Этот стиль кодирования позволяет базе данных различать код и данные независимо от того, какие пользовательские данные были предоставлены [8].

Использование подготовленных выражений гарантирует, что злоумышленник не сможет изменить задачи и намерения запроса, даже если он вставит зловредные SQL- команды. Приведём пример. Если злоумышленник введёт идентификатор пользователя «tom 'or' 1 '=' 1», параметризованный запрос не выполнит зловредный код, а вместо этого будет искать имя пользователя, которое буквально соответствует всей введённой строке.

Практически в каждом современном языке программирования есть свои расширения, позволяющие использовать подготовленные выражения, например:

– Java EE – применяет PreparedStatement() со связываемыми переменными.

– NET – использует параметризованные запросы, такие как SqlCommand() или OleDbCommand () со связанными переменными.

– PHP применяется PDO со строго типизированными параметризованными запросами (с использованием bindParam()).

В редких случаях подготовленные выражения могут нанести ущерб производительности. При столкновении с такой ситуацией лучше всего:

а) тщательно проверять все данные;

б) вместо использования подготовленных выражений применять обезвреживание введенных данных всех пользователей, используя специальные процедуры экранирования, специфичные для той базы данных с которой работаете (данный метод будет описан ниже).

Хранимые процедуры. Хранимые процедуры (англ. Stored Procedures) не всегда являются достаточными для того, чтобы избавиться от SQL-инъекций. Тем не менее, неко-

торые стандартные конструкции программирования хранимых процедур имеют тот же эффект, что и применение параметризованных запросов, когда они «безопасно реализованы», что является нормой для большинства языков хранимых процедур.

Они требуют от разработчика просто создавать SQL-запросы с параметрами, которые автоматически параметризуются, если только разработчик не делает что-то в значительной степени вне нормы. Разница между подготовленными выражениями и хранимыми процедурами заключается в том, что SQL-код для хранимой процедуры определяется и сохраняется в самой базе данных, а затем вызывается из веб-приложения.

Термин «безопасно реализованы» означает, что хранимая процедура не включает какую-либо небезопасную динамическую генерацию SQL. Разработчики обычно не генерируют динамический SQL внутри хранимых процедур, что можно сделать, но этого следует избегать. Если избежать нельзя, хранимая процедура должна использовать проверку ввода или правильное экранирование, чтобы убедиться, что все введенные в нее пользователем входные данные не могут использоваться для ввода SQL-кода в динамически сгенерированный запрос.

При аудите кода необходимо отслеживать использование sp_execute, execute или exec в хранимых процедурах SQL-сервера. Такие же рекомендации по аудиту необходимы для аналогичных функций в других СУБД.

Существует также несколько случаев, когда хранимые процедуры могут повысить риск. Например, на сервере MS SQL имеется 3 основных роли по умолчанию: db_datareader, db_datawriter и db_owner. Перед использованием хранимых процедур администраторы базы данных предоставляют пользователям db_datareader или db_datawriter права в зависимости от требований. Однако хранимым процедурам требуются права на выполнение, роль которых по умолчанию недоступна. Данные риски хранимых процедур описаны в статье [9].

Некоторые системы, в которых управление пользователями централизовано, но ограничено этими тремя ролями, приводят к тому, что все веб-приложения работают с правами db_owner, чтобы хранимые процедуры могли выполняться.

Естественно, это означает, что если сервер будет взломан, то злоумышленник будет иметь полные права на базу данных, а ранее он мог иметь доступ только на чтение.

Данный и предыдущий методы имеют одинаковую эффективность в предотвращении SQL-инъекций, поэтому разработчики должны сами выбирать, какой из этих подходов является наиболее предпочтительным.

Проверка входных данных. Различные части SQL-запросов не являются законными расположениями для использования связанных переменных, таких как имена таблиц или столбцов, а также указателей порядка сортировки (ASC или DESC). В таких ситуациях наиболее подходящей защитой является проверка входных данных или перепроектирование запроса. Имена таблиц или столбцов правильнее закладывать и брать из кода, а не из пользовательских параметров [10].

Но если пользовательские параметры используются для изменения имен таблиц и столбцов, то значения параметров должны быть сопоставлены с законными/ожидаемыми именами таблиц или столбцов (т.е. сравниваемы с заранее заданным «белым списком» имен), чтобы убедиться, что непроверенные данные пользователя не попадают в запрос. Стоит обратить внимание на то, что это признак плохой конструкции, и скорее всего необходимо учитывать полную переработку запроса, если позволяет время.

Проверка ввода по «белым спискам» также рекомендуется в качестве вторичной защиты во всех случаях, даже при использовании связываемых переменных.

Экранирование данных. Этот метод следует использовать только в качестве крайней меры, когда ни один из вышеперечисленных вариантов невозможно применить.

Предыдущий метод проверки входных данных, вероятно, является лучшим выбором, поскольку метод «Экранирование данных» является самым слабым способом защиты по сравнению с другими средствами защиты, и нельзя гарантировать, что он позволит предотвратить все пути SQL-инъекции.

Суть метода заключается в том, что необходимо произвести замену всех управляющих символов (экранирование) входных данных пользователя перед помещением их в тело запроса [11]. Обычно таким способом рекомендуется модифицировать устаревший код, когда внедрение той же проверки входных данных не является целесообразной и экономически эффективной.

Приложения, разрабатываемые с нуля или приложения с низкой степенью риска должны быть построены или переписаны с использованием параметризованных запросов, хранимых процедур или какого-либо объектного-реляционного отображения (ORM).

Метод работает следующим образом. Когда пользователь вводит какие-либо входные данные в какую-либо форму на веб-ресурсе, некоторые символы впоследствии могут восприниматься, как управляющие и быть использованы как продолжение SQL-команды. Для таких символов необходимо использовать специальные схемы экранирования.

Каждая СУБД поддерживает одну или несколько схем экранирования символов, специфичных для определенных типов запросов. Если вы защитите все входные данные, вводимые пользователем, используя правильную схему экранирования для применяемой БД, то в СУБД не будет возникать путаница входных данных с SQL-кодом, написанным разработчиком, что позволит избежать возможных атак SQL-инъекции.

1.4. Выводы

Атаки SQL-инъекции представляют собой одну из самых больших угроз для базы данных и клиентов. Поскольку они обычно используются для непосредственного воздействия на информацию, которую видит клиент, ее можно легко использовать для попытки проникновения вредоносного кода на компьютеры клиентов.

Данный тип атак очень популярен у злоумышленников, потому что он является относительно простым в реализации способом и не требует особых условий для проведения. Он также популярен, потому что их легко провести на ресурсе, который просто поддается компрометации, поскольку разработчикам обычно требуется много времени, чтобы обнаружить и исправить все потенциальные точки атаки.

Это оставляет сам веб-ресурс и его базу данных беззащитными и открытыми для атаки в течение длительного периода времени, поскольку компании обычно не желают закрывать его, пока производится настройка и исправление ошибок.

Анализ атак SQL-инъекции показывает, что администратор базы данных, разработчик приложения и системный администратор должны работать вместе, чтобы обеспечить правильную защиту данных. По мере того как разработчики баз данных и приложений начнут вводить в привычку создавать более безопасный код, который не подвержен атакам SQL-инъекций, их программный продукт станет более безопасным, равно как и их будущие проекты.

При правильном планировании (например, надлежащем построении защиты объектов) и надлежащем мониторинге (например, при использовании xEvents для фиксации подозрительных запросов) можно предотвратить множество потенциальных проблем, связанных с SQL-инъекциями. Когда же данные проблемы обнаруживаются, можно отслеживать запросы и видеть, что случилось.

2. Межсайтовый скриптинг

Межсайтовый скриптинг (англ. CrossSite Scripting – XSS) – это ещё один из типов атак, также связанный с проверкой входных данных. Механизм данных атак очень схож с атаками SQL-инъекции, но в отличие от второго все зловредные действия проводятся с клиентом веб-ресурса, то есть код выполняется в браузере посетителя.

Таким образом, в XSS-атаках жертвой является сам пользователь, а не веб-приложение. При XSS-атаках вредоносный контент доставляется пользователям с использованием языка программирования, выполняемого на стороне клиента, такого как JavaScript. К тому же XSS-атаки, в отличие от SQL-инъекций, наносят вред исключительно клиентской стороне, оставляя сервер приложения полностью в рабочем состоянии.

При применении XSS-атак злоумышленник может преследовать, например, следующие цели:

– завладеть сессионной Cookie пользователя и нанести вред от его лица;

– украсть входные данные пользователя, вводимые через формы на веб-ресурсе (например, данные кредитной карты и т.п.);

– изменить входные данные, подменив их подставными (например, реквизиты счета и т.п.);

– использовать недостаток для внедрения зловредной рекламы и т.д.

2.1. Основные понятия и принципы

Чтобы понять принцип XSS-атак, для начала необходимо понять основную концепцию Same Origin Policy (SOP). Правило ограничения домена (SOP) является базовым механизмом защиты, предлагаемым веб-браузерами против данных атак. Это политика контроля доступа, обеспечивающая строгое разделение между содержимым, предоставляемым различными веб-источниками. Концепция SOP описана в статье [12].

Данная политика разрешает сценариям, находящимся на страницах одного сайта, доступ к методам и свойствам друг друга без ограничений, но предотвращает доступ к большинству методов и свойств для страниц на разных сайтах. Одинаковые источники – это источники, у которых совпадают три признака: домен, порт, протокол.

В результате действия SOP, например, скрипты, запущенные на странице, загруженной с «http://attacker.com», не позволяют получить доступ к файлам, установленным на сайте «http://trusted.com». Это является предварительным условием для того, чтобы злоумышленники не могли захватывать различные файлы, например, сессионные Cookie, с надежного веб-ресурса, раскрывать их и использовать в своих целях.

Под сессионными Cookie понимаются временные данные, частично хранящиеся на стороне клиента (может храниться только ID сессии) и доступные на период, пока пользователь находится на ресурсе. Такие данные могут содержать логин и зашифрованный пароль пользователя. Когда злоумышленник заполучает данную Cookie он может обмануть сервер и выдать себя за жертву.

К сожалению, применения политики SOP недостаточно для предотвращения множества распространенных атак, таких как XSS.

Межсайтовый скриптинг является способом обхода концепции SOP. Всякий раз, когда HTML-код динамически генерируется, а входные данные пользователя не подвергаются какой-либо проверке или фильтрации и отражаются на странице, злоумышленник может использовать данный недостаток для того, чтобы встроить свой собственный HTML-код.

В таком случае злоумышленник может легко вставить JavaScript-код, который будет запускаться в контексте сайта. Таким образом, он сможет получить доступ к другим страницам в одном домене и будет иметь право читать данные, такие как CSRF-токены или установленные файлы Cookie.

Если файлы Cookie, которые обычно содержат информацию идентификатора сеанса, могут быть прочитаны с помощью JavaScript, злоумышленник имеет возможность применить их в своем браузере и войти в веб-приложение под идентификационными данными жертвы. Если данный вектор не работает, злоумышленник может считывать личную информацию со страниц, например, читать CSRF-токены и делать запросы от имени пользователя. Такой тип атаки называется CSRF (англ. Cross Site Request Forgery – Межсайтовая подделка запроса). Данная уязвимость также может эксплуатироваться совместно с XSS-атаками. В работе [13] авторы дали определения XSS, CSRF и описали их сходства и различия.

Сходство между CSRF и XSS – использование в качестве вектора атаки клиентов веб-приложений. Хотя на первый взгляд CSRF-атаки очень похожи на атаки на основе межсайтового скриптинга (XSS), они отличаются своими целями.

В то время, как целью XSS-атаки является вставка активного кода в HTML-документ либо для эксплуатации уязвимостей среды исполнения сценариев на стороне клиента, либо для отправки важной информации (такой, как данные сессионных Cookie) на неизвестный сайт злоумышленника. Целью CSRF-атаки является выполнение нежелательных для пользователя действий с помощью веб-сайта, на котором он имеет учетную запись и с которым он работал ранее.

Более того, там, где с помощью XSS-атаки злоумышленники ищут возможность для похищения данных из Cookie для манипуляции пользовательской учетной записью, с помощью CSRF-атаки ищут возможность использования пользовательских Cookie для выполнения действий без информирования и получения согласия пользователя.

Приведём пример классического сценария проведения CSRF-атаки. Например, пользователь авторизован на каком-либо сервисе электронной почты, и при этом в браузере запомнена сессионная Cookie. Если пользователь каким-либо образом (методами социальной инженерии и т.п.) попадет на подготовленную злоумышленником страницу, на которой размещён вредоносный скрипт, браузер жертвы совершает некий запрос к серверу электронной почты. Сайт электронной почты проверяет есть ли Cookie, видит, что посетитель авторизован и обрабатывает вредоносный запрос, тем самым выполняя нужные злоумышленнику действия.

Таким образом, можно сказать, что в случае XSS-атак используется доверие, оказываемое пользователем вед-ресурсу, а в случае CSRF-атак – доверие веб-сайта своему пользователю.

2.2. Участники XSS-атаки

Перед тем, как приступить к описанию работы XSS-атак, необходимо определить все стороны, участвующие в них. В статье [14] авторы дали определение всем основным участникам данной атаки. При ее проведении задействованы три участника: веб-ресурс, жертва и атакующий.

Веб-ресурс выводит HTML-страницы для пользователей, которые запрашивают их. В представленных примерах он находится по адресу http://website/.

База данных веб-ресурса – это часть приложения, которая хранит вводимые пользовательские данные на страницах сайта.

Жертва – это обычный посетитель ресурса, запрашивающий его страницы, используя свой браузер.

Атакующий – это злоумышленник, намеревающийся провести атаку на жертву используя XSS-уязвимости на ресурсе.

Сервер злоумышленника – это ресурс, находящийся под контролем злоумышленника, применяемый для кражи конфиденциальной информации жертвы. В приведенных примерах он находится по адресу http://attacker/.

2.3. Типы межсайтовых скриптов

Сначала были обнаружены и идентифицированы два основных вектора XSS-атак: хранимый и отраженный XSS, но в 2005 году Амит Клейн определил и ввёл третий вектор данной атаки, который был назван DOM-ориентированный XSS. В работе [15] авторами был описан каждый из трех векторов данной атаки.

Эти три типа XSS-атак определяются следующим образом:

1) Хранимые (постоянные) XSS (англ. Stored (Persistent) XSS).

Данный тип атаки является самым разрушительным. Хранимые XSS обычно совершаются, когда пользовательские входные данные хранятся на целевом сервере, в виде сообщений на форуме, в журнале посетителей, в поле комментариев и т.д.

Классическим примером является вредоносный скрипт, вставленный злоумышленником в поле комментариев в блоге или в сообщения на форуме. Когда жертва переходит на уязвимую веб-страницу в браузере, полезная нагрузка XSS будет использоваться как часть самой веб-страницы (точно также, как и сами законные комментарии) [16]. Это означает, что жертвы, в конечном итоге, случайно выполнят вредоносный сценарий, как только страница будет просмотрена в браузере.

Пример проведения данного типа XSS-атаки, изображён на рис. 2. В данном и последующих примерах мы будем считать, что конечной целью злоумышленника является кража Cookie жертвы с использованием XSS-уязвимости веб-ресурса. Данная операция будет совершена, если браузер жертвы обработает и выполнит следующий код:

```
<script>
window.location='http://attacker/?cookie='+document.cookie
</script>
```

Этот сценарий выполнит HTTP-запрос на другой URL-адрес, который перенаправит браузер пользователя на сервер атакующего. При этом данный URL-адрес включает в себя Cookie жертвы в качестве параметра запроса. Когда HTTP-запрос поступит на сервер злоумышленника, он сможет извлечь полученные Cookie из самого запроса и дальше использовать их своих целях.

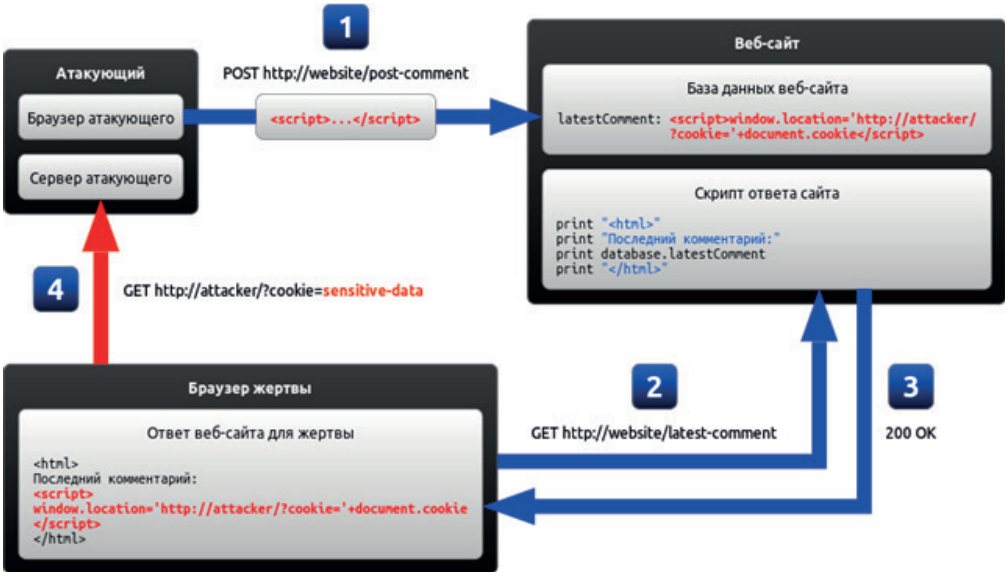

Рис. 2. Пример постоянного XSS.

На рис. 2, представлена следующая последовательность действий: шаги:

1. Злоумышленник использует какую-либо из форм на ресурсе (например, форму комментариев) для того, чтобы отправить вредоносный код в базу данных веб-ресурса.

2. Жертва запрашивает страницу с веб-сайта.

3. Веб-ресурс в ответ включает вредоносный код из базы данных и отправляет его обратно жертве.

4. Браузер жертвы считывает JavaScript код и выполняет вредоносный сценарий внутри ответа, отправляя Cookie на сервер злоумышленника.

Таким образом любой из пользователей, посетивший данную страницу с внедрённым зловредным сценарием, будет подвержен хранимой XSS-атаке.

С появлением HTML5 и других технологий браузера можно предположить, что полезная нагрузка атаки будет постоянно храниться в браузере жертвы, такой как база данных HTML5 (Web SQL Database), и никогда не будет отправлена на сервер приложения. Таким образом, хранимые XSS не обязательно выполняются через сервер БД, а могут быть выполнены и через локальную базу данных, хранимую на клиенте.

2.4. Отражённые (непостоянные) XSS (англ. Reflected (Non-Persistent) XSS).

Вторым распространённым типом XSS-атаки является отраженные XSS. Данный тип атаки происходит, когда входные данные пользователя немедленно возвращаются веб-приложением в сообщении об ошибке, результатах поиска или любом другом ответе, который включает в себя часть или все данные, предоставленные пользователем в рамках запроса.

Здесь сценарий полезной нагрузки злоумышленника должен быть частью запроса, который отправляется на веб-сервер и отражается обратно таким образом, что HTTP-ответ включает полезную нагрузку из HTTP-запроса [17].

Используя фишинговые электронные письма и другие методы социальной инженерии, злоумышленник заманивает жертву непреднамеренно сделать запрос на сервер, который содержит полезную нагрузку XSS, и завершает выполнение сценария, который отражается и выполняется внутри браузера. Пример данной атаки представлен на рис. 3.

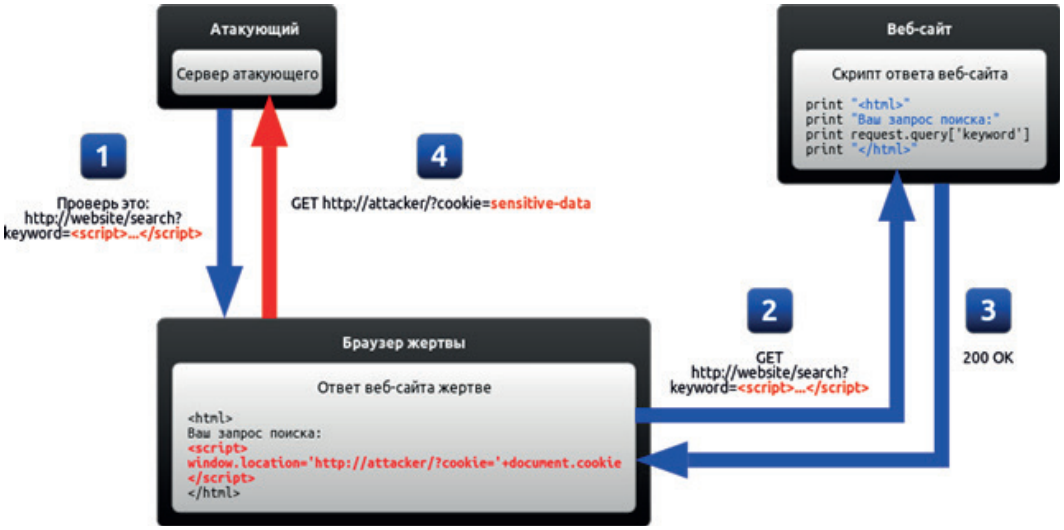

Рис. 3. Пример отраженного XSS.

1. Злоумышленник создает и отправляет жертве URL-адрес, содержащий вредоносную строку.

2. Жертва обманным путем злоумышленника, переходя по ссылке, отправляет URL-запрос на веб-ресурс.

3. Веб-сайт в ответ включает вредоносный код из полученного запроса.

4. Браузер жертвы выполняет вредоносный сценарий, содержащийся в ответе, тем самым отправляя Cookie жертвы на сервер злоумышленника.

Поскольку «Отраженный XSS» не является постоянной атакой, злоумышленник должен доставлять полезную нагрузку каждой жертве, поэтому, например, социальные сети часто используются для распространения данных атак.

Существует множество способов, которыми злоумышленник может побудить жертву инициировать отраженный XSS-запрос. Например, злоумышленник может отправить жертве вводящее в заблуждение электронное письмо со ссылкой, содержащей вредоносный JavaScript. Чтобы жертва не смогла распознать строку адреса с зловредным скриптом, злоумышленники могут прибегать к применению служб, позволяющих «укоротить» ссылку, тем самым маскируя всю вредоносную строку.

Если жертва нажимает на такую ссылку, HTTP-запрос инициируется из браузера жертвы и отправляется в уязвимое веб-приложение. Затем вредоносный JavaScript возвращается к браузеру жертвы, где он выполняется в контексте сессии пользователя.

2) XSS в DOM (англ. DOM Based XSS).

Как определил Амит Клейн, который опубликовал первую статью об этой проблеме, «DOM Based XSS» представляет собой форму XSS, при которой в браузере происходит весь нарушенный поток данных от источника к приёмнику, то есть источник данных находится в DOM, приемник также находится в DOM, и поток данных никогда не покидает браузер.

Данный тип атаки не зависит от вредоносного содержимого, отправляемого на сервер. В хранимом и отраженном XSS-скрипт включен в ответ сервера. Браузер жертвы прини-

мает его, полагая, что он является законной частью веб-страницы и выполняет его при загрузке страницы. В «DOM Based XSS» выполняется только законный сценарий, предоставляемый сервером. В статье [18] был описан третий тип XSS, известный как «DOM Based XSS».

XSS в DOM является расширенным типом атаки XSS, которая становится возможной, когда сценарии клиентской стороны веб-приложения записывают предоставленные пользователем данные в объектную модель документа (англ. Document Object Model – DOM).

Затем данные считываются из DOM с помощью веб-приложения и выводятся в браузер. Если данные некорректно обрабатываются, злоумышленник может ввести полезную нагрузку, которая будет храниться как часть DOM и будет выполняться при чтении данных из DOM. Пример данного типа XSS, представлен на рис. 4.

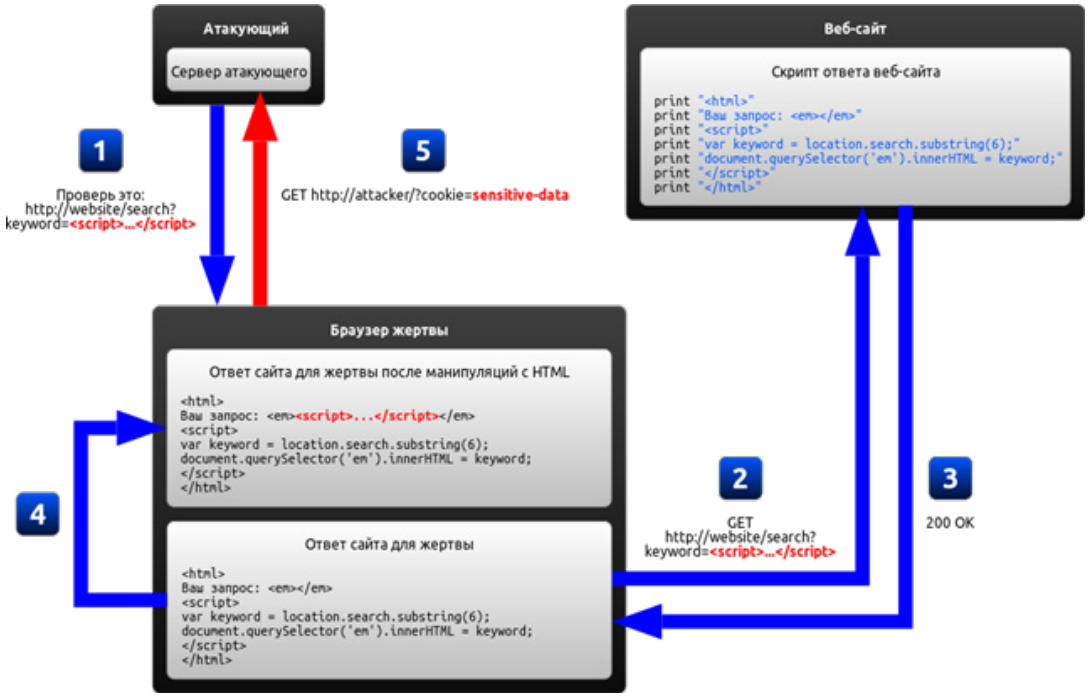

Рис. 4. XSS в Document Object Model.

1. Злоумышленник создаёт URL-адрес, содержащий вредоносную строку, и отправляет его жертве.

2. Жертва обманным путем злоумышленника, переходя по ссылке, отправляет запрос на веб-ресурс.

3. Веб-ресурс принимает запрос, но не включает в ответ вредоносный код.

4. Браузер жертвы выполняет законный сценарий, содержащийся в ответе, в результате чего вредоносный скрипт будет внедрён в страницу.

5. Браузер жертвы выполняет вредоносный сценарий, вставленный в страницу, и отправляет данные Cookie жертвы на сервер злоумышленника.

Самой опасной вариацией данного типа атаки является атака на стороне клиента, при этом полезная нагрузка злоумышленника никогда не отправляется на сервер [19]. Это происходит тогда, когда вредоносный код содержится в фрагменте идентификатора

URL-адреса (что-либо после символа #). Браузеры не отправляют эту часть адреса на сервер, так что веб-ресурс не имеет доступа к нему на стороне сервера. Однако, код со стороны клиента имеет доступ к нему, и таким образом возможно проведение XSS-атаки.

Это затрудняет обнаружение такой атаки различными брандмауэрами веб-приложений (WAF) и разработчиками средств безопасности, анализирующими журналы сервера, так как они не смогут увидеть проведенную атаку.

Данный случай не ограничивается идентификатором фрагмента. Существуют и другие пользовательские данные, которые будут невидимыми для сервера, например, новые функции HTML5, такие как LocalStorage и IndexedDB.

Среди различных объектов, составляющих DOM, есть некоторые объекты, которыми злоумышленник может манипулировать, чтобы генерировать условие XSS. К таким объектам относятся URL (document.URL), часть URL-адреса за хэшем (location.hash) и Referrer (document.referrer).

3) Отличие XSS в DOM от других типов XSS-атак.

В первых двух вышеописанных типах XSS-атак (Хранимые и Отраженные) сервер веб-приложения вставляет вредоносный сценарий на страницу, которая затем отправляется в ответе обратно жертве. Когда браузер жертвы получает ответ, он подразумевает, что вредоносный сценарий – это одна из частей законного содержания страницы, и далее автоматически выполняет его во время загрузки страницы, как и любой другой сценарий.

Рассматривая третий тип «DOM Based XSS» можно заметить, что вредоносный код не вставляется как часть страницы. Единственный сценарий, который будет автоматически выполняться во время загрузки страницы, является законной частью страницы. Проблема состоит в том, что данный законный сценарий напрямую использует пользовательские входные данные для того, чтобы добавить HTML на страницу. Поскольку вредоносный код встраивается в страницу с помощью innerHTML, он и будет рассматриваться как HTML, в результате чего данный зловредный скрипт будет работать [20]. Данное различие хоть и не существенное, но является очень важным.

Итак, главное отличие XSS в DOM состоит в следующем. В хранимых и отражённых XSS-атаках вредоносный JavaScript выполняется при загрузке страницы, как часть HTML, отправленного сервером. В случае с «XSS в DOM-модели» зловредный JavaScript выполняется после загрузки страницы, в результате данная страница с закономерным JavaScript обращается небезопасным способом к пользовательскому вводу, включающему вредоносный код.

2.5. Рекомендации по устранению XSS-атак

Для предотвращения уязвимостей, связанных с межсайтовым скриптингом, очень важно применять контекстно-зависимую кодировку выходных данных. В некоторых случаях этого может быть достаточно для кодирования специальных символов HTML, таких как открытие и закрытие тегов.

В других случаях необходимо правильно применять кодирование URL. Ссылки, как правило, должны быть запрещены, если их адреса не начинаются с протоколов из «белого списка», таких как «http://» или «https://», таким образом предотвращая использование URI схем, таких как «javascript://». В работе [21] описаны некоторые из способов устранения XSS.

Несмотря на то, что большинство современных веб-браузеров имеют встроенный фильтр XSS, их не следует рассматривать как альтернативу дезинфекции. Они не могут улавливать все виды межсайтовых скриптовых атак и не являются достаточно точными, чтобы не приводить к ложным срабатываниям, которые помешали бы правильной загрузке некоторых страниц ресурса. Фильтр XSS веб-браузера должен быть использован только как «вторая линия защиты», их главная цель состоит в том, чтобы минимизировать воздействие уже существующих уязвимостей.

Разработчики не должны использовать «черные списки», поскольку для них существует множество обходов. Еще одна вещь, которую им следует избегать – это удаление опасных функций и символов, поскольку фильтры XSS браузеров не могут распознавать опасные полезные нагрузки, когда результат подделывается с учетом возможных обходов. При этом единственной рекомендацией предотвращения XSS является кодирование, как было указано выше.

2.6. Способы защиты от XSS-атак

Рассмотрим подробнее способы защиты от XSS-атак. Далее будет приведена простая модель для предотвращения XSS, использующая экранирование/кодирование выходных данных должным образом. Хотя существует огромное количество векторов атаки XSS, следующие несколько простых правил могут полностью защитить от этой серьезной атаки.

Напомним, что XSS является атакой встраивания кода, то есть входные данные пользователя ошибочно интерпретируются как зловредный код. Для того, чтобы не допустить данной атаки, требуется безопасная обработка входных данных. Для веб-разработчиков существуют два способа выполнения безопасной обработки вводимых данных:

Кодирование – позволяет обрабатывать входные данные пользователя только как текст и не позволяющий браузеру обрабатывать входные параметры как код.

Кодирование подразумевает под собой замену всех управляющих символов на эквивалентные. Самый популярный тип кодирования – это маскирование HTML, который преобразует символы, такие как < и > в эквивалентные < и > соответственно.

Если, например, пользователь введёт <script>...</script>, то результирующий HTML станет выглядеть следующим образом: <script>...</script> [22]. Таким образом, если все управляющие символы будут замаскированы, то браузер не воспримет введённые данные как зловредный код, а выдаст их обезвреженными как обычный текст.

Валидация – позволяет производить фильтрацию пользовательских данных так, что браузер будет интерпретировать их как код без вредоносных команд.

Один из применяемых способов проверки в веб-разработке позволяет использовать некоторые HTML-элементы (например, и), но запрещает другие (например, <script>). Фильтрация может производиться двумя способами: через «чёрный список» либо через «белый список». Хотя это достаточно различные методы предотвращения XSS, они имеют несколько общих черт, которые являются важными для понимания при использовании любого из них.

Концепция «чёрного списка» рассматривает запрещённые шаблоны, которые не должны появляться в пользовательском вводе. Например, разрешить пользователям отправлять ссылки с любыми протоколами, кроме «javascript:».

Но главным недостатком данного подхода является сложность реализации, так как приходится учитывать множество различных факторов. Например, для примера выше

первую букву можно закодировать как «javascript:», тогда поиск по подстроке пропустит данный вариант, и атака окажется выполнимой [23].

Подход же «белого списка» определяет разрешенный шаблон, в противоположность «чёрному списку», и отмечает вводимые данные недействительными, если они не соответствует данному шаблону. Как пример, можно разрешить пользователям использовать лишь протоколы «http:» и «https:» и ничего более. «Белый список» гораздо проще в реализации, и он не устаревает, так как, если появится новая функция, то её просто необходимо внести, дополнив таким образом список.

Контекст. Безопасная обработка входных данных должна быть выполнена по-разному в зависимости от того, где на странице используется пользовательский ввод.

Есть множество контекстов на странице, где может быть применен пользовательский ввод. Для каждого из них должны быть соблюдены особые правила для того, чтобы пользовательский ввод не мог рассматриваться вне своего контекста и не мог быть интерпретирован как вредоносный код. В таблице приведены наиболее распространенные контексты.

Контексты

Контекст	**Пример кода**
Контент в виде HTML элемента	Пользовательские данные
Атрибуты значений в HTML	<input value=" Пользовательские данные">
Значения в URL-запросе	http://example.com/?parameter=Пользовательские данные
Значения в JavaScript	var name = "Пользовательские данные"

В рассмотренных контекстах уязвимость, приводящая к XSS, может возникнуть, если вводимые пользователем данные были вставлены до первого кодирования или проверки. Злоумышленник может внедрить вредоносный код, просто дописав закрывающий разделитель для данного контекста и следом за ним вредоносный код [24].

Рассмотрим пример для пользовательских данных, вводимых в атрибуты значений HTML. Исходя из данного контекста, строка зловредного кода должна выглядеть следующим образом:

"><script>...</script><input value=".

Тогда в итоге, будет получен код следующего вида:

<input value=""><script>...</script><input value="">.

Данную уязвимость можно устранить путём удаления кавычек, используемых в пользовательских данных, и тогда XSS не сможет быть выполнена, но это применимо лишь к данному контексту. Поэтому безопасная обработка вводимых данных всегда должна быть применена к контексту, где будут вставляться пользовательские данные.

Проверка входящих/исходящих данных. Безопасная обработка данных может быть выполнена когда на ресурс поступают входные данные, или прямо перед тем, как веб-приложение вставляет пользовательские данные в содержимое страницы (исходящие данные).

Защита может быть выполнена путём обезвреживания данных, как только они поступают на сервер веб-ресурса, и таким образом можно обеспечить, что все вредоносные коды будут нейтрализованы. Но проблема заключается в том, что пользовательские данные могут быть вставлены в несколько контекстов, как было описано выше. Тогда нет простого способа отловить все возможные пути внедрения XSS и обезопасить их.

Вместо этого необходимо использовать защиту на уже исходящих данных от сервера, потому что средства защиты могут принимать во внимание каждый конкретный контекст и то, какие вводимые пользователем данные будут вставлены.

Обработка на Клиенте/Сервере. Безопасная обработка может быть проведена либо на стороне клиента, либо на стороне сервера, данные варианты стоит рассматривать при различных обстоятельствах.

В современных веб-приложениях проверка данных на наличие всех типов XSS-атак должна производиться как на стороне сервера, так и на стороне клиента.

При защите от хранимых и отражённых XSS безопасная обработка входных данных должна быть выполнена в коде на стороне сервера. Это производится при помощи различных инструментов языка программирования, используемого сервером, например, PHP.

При защите от атак типа «XSS в DOM», при которых сервер никогда не получает зловредный код (например, рассмотренная ранее атака через идентификатор #), обработка входных данных должна быть выполнена в коде на стороне клиента [25]. Это производится с помощью методов и свойств клиентского языка JavaScript.

В качестве второй линии обороны можно использовать Политики безопасности контента (англ. Content Security Policy – CSP). Они позволяют вносить ограничения на использование ресурсов только из надёжных источников. Это означает, что если даже злоумышленнику удастся внедрить вредоносный контент (скрипт) на веб-ресурс, CSP предотвратит его исполнение.

2.7. Вывод

Межсайтовый скриптинг является одной из самых опасных уязвимостей веб-ресурса, равно как и SQL-инъекции. Он осуществляется различными способами, чтобы навредить пользователям ресурса.

В основном он используется для атаки на захват сессии (кража сессионной Cookie). Защита от уязвимостей XSS возможна, но никогда нельзя на 100% сказать, что никто не сможет сломать разработанную защиту или фильтр.

Всемирная паутина развивается и появляются новые функции и технологии, поэтому злоумышленники всегда находят способы обойти защиту. Все рассмотренные векторы атаки XSS выполняются по-разному, но имеют один и тот же эффект, если выполняются успешно. Так как атака XSS основана на вводимых пользователем данных, то стоит учитывать безопасную обработку входных данных.

Кодирование должно выполняться везде, где используется пользовательский ввод. В различных ситуациях кодирование должно быть дополнено фильтрацией данных (валидацией). Для предотвращения всех типов XSS-атак обработка вводимых данных должна выполняться в коде как на стороне клиента, так и на стороне сервера.

В качестве дополнительного уровня защиты стоит рассмотреть Политики безопасности контента (CSP), которые позволят избежать выполнения различных вредоносных сценариев в случае, если безопасная обработка вводимых данных содержит ошибку.

Заключение

Существует множество подходов и конструкций, реализованных в различных веб-приложениях, но безопасность по-прежнему является одним из основных вопросов во всем мире. В данной статье были рассмотрены две из наиболее опасных уязвимостей

в сфере веб-технологий, а также конкретные случаи и примеры SQL-инъекций и XSS-атак на веб-приложения. Наличие уязвимостей XSS и SQL-инъекций в веб-приложениях, имеет огромный риск не только для веб-приложений, но и для самих пользователей.

Также в статье были рассмотрены различные существующие подходы к выявлению и предотвращению этих уязвимостей в приложении. Разработчик должен использовать предоставленные механизмы и стараться разрабатывать безопасный код.

Для облегчения обнаружения различных уязвимостей существуют различные инструменты (сканеры), которые могут помочь в анализе безопасности веб-приложений и облегчить разработку защиты. Но данные инструменты в большинстве своём могут только идентифицировать проблемы, но не способны устранять их. Поэтому знания разработчика безопасности являются ключевым фактором при построении безопасного веб-ресурса.

Для устранения проблем безопасности приложений, разработчики должны знать все пути и векторы проведения различных атак, чтобы иметь возможность разрабатывать механизмы защиты.

Межсайтовый скриптинг, также как и атаки SQL-инъекций, связан с проверкой входных данных. Механизмы таких атак очень схожи, но в XSS-атаках жертвой является сам пользователь, а в атаках SQL-инъекцией – сервер базы данных веб-приложения. При XSS-атаках вредоносный контент доставляется пользователям с использованием языка программирования, выполняемого на стороне клиента, такого как JavaScript, а при использовании SQL-инъекции используется язык запросов к БД SQL.

При этом XSS-атаки, в отличие от SQL-инъекций, наносят вред исключительно клиентской стороне, оставляя сервер приложения полностью в рабочем состоянии.

Разработчики должны применять защиту, как для серверных составляющих, так и для клиентской части веб-приложения. В большинстве языков программирования, ориентированных на веб-разработку, есть множество средств и инструментов, позволяющих защитить веб-приложение и избежать большинства атак.

Таким образом, правильно спроектированные и реализованные механизмы защиты позволят веб-приложению оставаться неуязвимым для большинства существующих атак.

Литература / References:

1. Kaur D., Kaur P. Empirical Analysis of Web Attacks. *Procedia Computer Science.* 2016;78:298-306. https://doi.org/10.1016/j.procs.2016.02.057

2. Nagpal B., Chauhan N., Singh N. A Survey on the detection of SQL injection attacks and their countermeasures. *J. Inf. Process. Syst.* 2017;13(4):689-702. https://doi.org/10.3745/JIPS.03.0024

3. Wang K. and Hou Y. Detection method of SQL injection attack in cloud computing environment. In: *2016 IEEE Advanced Information Management, Communicates, Electronic and Automation Control Conference (IMCEC).* P. 487-493. https://doi.org/10.1109/IMCEC.2016.7867260

4. Hu H. Research on the technology of detecting the SQL injection attack and non-intrusive prevention in WEB system. In: 2017 *AIP Conference Proceedings.* 2017;1839(1):020205. https://doi.org/10.1063/1.4982570

5. Lounis O., Guermeche S.E.B., Saoudi L., Benaicha S.E. A new algorithm for detecting SQL injection attack in Web application. In: 2014 *Science and Information Conference (SAI)* 2014. P. 589-594. https://doi.org/10.1109/SAI.2014.6918246

6. Voitovych O.P., Yuvkovetskyi O.S., Kupershtein L.M. SQL injection prevention system. In (2016) *International Conference Radio Electronics & Info Communications (UkrMiCo) 2016.* P. 1-4. https://doi.org/10.1109/UkrMiCo.2016.7739642

7. Razzaq A., Anwar Z., Ahmad H.F., Latif K., Munir F. Ontology for attack detection: An intelligent approach to Web application security. *Computers & Security.* 2014;45:124-146.
https://doi.org/10.1016/j.cose.2014.05.005

8. Vibhandik R., Bose A.K. Vulnerability assessment of Web applicat*ions – a testing approach. In:* Forth International Conference on e-Technologies and Networks for Development (ICeND). 2015. P. 16-21.
https://doi.org/10.1109/ICeND.2015.7328531

9. Sahu D.R., Tomar D.S. Analysis of Web application code vulnerabilities using secure coding standards. *Arab. J. Sci. Eng.* 2017;42(2):885-895.
https://doi.org/10.1007/s13369-016-2362-5

10. Shar L.K., Tan H.B.K. Predicting SQL injection and Cross site scripting vulnerabilities through mining input sanitization patterns. *Inform. Software Tech.* 2013;55(10):1767-1780.
http://dx.doi.org/10.1016/j.infsof.2013.04.002

11. Canfora G., Visaggio C.A. A set of features to detect Web security threats. *Journal of Computer Virology and Hacking Techniques.* 2016;12(4):243-261.
https://doi.org/10.1007/s11416-016-0266-2

12. Wang C.-H., Zhou Y.-S. A new Cross-Site scripting detection mechanism integrated with HTML5 and CORS properties by using browser extensions. In: *International Computer Symposium (ICS).* 2016. P. 264-269.
https://doi.org/10.1109/ICS.2016.0060

13. Alvarez E.D., Correa B.D., Arango I.F. An analysis of XSS, CSRF and SQL injection in colombian software and web site development. In: *8th Euro American Conference on Telematics and Information Systems (EATIS).* 2016. P. 1-5.
https://doi.org/10.1109/EATIS.2016.7520140

14. Gupta S., Gupta B. B. Automated Discovery of JavaScript Code Injection Attacks in PHP Web Applications. *Procedia Computer Science.* 2016;78:82-87.
https://doi.org/10.1016/j.procs.2016.02.014

15. Vishnu B.A., Jevitha K.P. Prediction of Cross-Site Scripting Attack Using Machine Learning Algorithms. In: *Procc. International Conference on Interdisciplinary Advances in Applied Computing 2014 (ICONIAAC '14).* Article 55.
https://doi.org/10.1145/2660859.2660969

16. Shrivastava A., Choudhary S., Kumar A. XSS vulnerability assessment and prevention in web application. In: *2nd International Conference on Next Generation Computing Technologies (NGCT-2016).* P. 850-853.
https://doi.org/10.1109/NGCT.2016.7877529

17. Sivakorn S., Keromytis A.D., Polakis J. That's the way the Cookie crumbles: Evaluating HTTPS enforcing mechanisms. In: *Proceedings of the 2016 ACM on Workshop on Privacy in the Electronic Society (WPES '16)*. P. 71-81.
http://dx.doi.org/10.1145/2994620.2994638

18. Wang R., Xu G., Zeng X., Li X., Feng Z. TT-XSS: A novel taint tracking based dynamic detection framework for DOM Cross-Site Scripting. *J. Parallel Distrib. Comput.* 2017;118(1):100-106.
https://doi.org/10.1016/j.jpdc.2017.07.006

19. Zalbina M.R., Septian T.W., Stiawan D., Idris M.Y., Heryanto A., Budiarto R. Payload recognition and detection of Cross Site Scripting attack. In: *2nd International Conference on Anti-Cyber Crimes (ICACC-17).* 2017. P. 172-176.
https://doi.org/10.1109/Anti-Cybercrime.2017.7905285

20. Jerkovic H., Vranesic P., Dadic S. Securing web content and services in open source content management systems. In: *39th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO) 2016.* P. 1402-1407.
https://doi.org/10.1109/MIPRO.2016.7522359

21. Zhang S., Wang W., Chen Z., Gu H., Liu J., Wang C. A web page malicious script detection system. In: *3rd International Conference on Cloud Computing and Intelligence Systems.* IEEE 2014. P. 394-399.
https://doi.org/10.1109/CCIS.2014.7175767

22. Rexha B., Halili A., Rrmoku K., Imeraj D. Impact of secure programming on web application vulnerabilities. In: *International Conference on Computer Graphics, Vision and Information Security (CGVIS).* 2015 IEEE. P. 61-66.
https://doi.org/10.1109/cgvis.2015.7449894

23. Filipe R., Araujo F. Client-side monitoring techniques for web sites. In: *15th International Symposium on Network Computing and Applications (NCA).* 2016 IEEE. P. 363-366.
https://doi.org/10.1109/NCA.2016.7778642

24. Zachara M. Identification of scanning and attacks against web applications with graph-based modeling of users' behavior. In: *3rd IEEE International Conference on Cybernetics 2017 (CYBCONF).* 8 p.
https://doi.org/10.1109/CYBConf.2017.7985783

25. Jemi Hazel J., Valarmathie P., Saravanan R. Guarding web application with multi-angled attack detection. In: *International Conference on Soft-Computing and Network Security (ICSNS -2015).* 4 p.
https://doi.org/10.1109/ICSNS.2015.7292382

Об авторе:

Лесько Сергей Александрович, кандидат технических наук, доцент кафедры «Прикладные информационные технологии» Института комплексной безопасности и специального приборостроения ФГБОУ ВО «МИРЭА – Российский технологический университет» (119454, Россия, Москва, пр-т Вернадского, д. 78). Scopus Author ID: 57189664364, https://orcid.org/0000-0002-6641-1609

About the author:

Sergey A. Lesko, Cand. Sci. (Engineering), Associate Professor of the Department «Applied information technology», Institute of Integrated Security and Special Instrumentation, MIREA – Russian Technological University (78, Vernadskogo pr., Moscow 119454, Russia). Scopus Author ID: 57189664364, https://orcid.org/0000-0002-6641-1609

Поступила: 18.05.2020; получена после доработки: 02.09.2020; принята к опубликованию: 10.10.2020.