Министерство образования Калининградской области

государственное автономное учреждение

Калининградской области

профессиональная образовательная организация

«Колледж предпринимательства»

**ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ (ДИПЛОМНАЯ) РАБОТА**

**Тема: Анализ современных угроз информационной безопасности на веб-сайтах и способы их предотвращения**

|  |  |
| --- | --- |
| Выпускная квалификационная  (дипломная) работа  допущена к защите  Заместитель директора по УМР  \_\_\_\_\_\_\_\_\_\_\_\_ Бурыкина Ю.И.  «\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_2025 г. | Выполнил:  обучающийся группы ОБС 21-2  специальность 10.02.05 Обеспечение информационной безопасности автоматизированных систем  \_\_\_\_\_\_\_\_\_\_\_ Кудин А.Ю.  Руководитель:  \_\_\_\_\_\_\_\_\_\_\_ Бычай А.П. |

Калининград

2025

**СОДЕРЖАНИЕ**

[ВВЕДЕНИЕ 3](#_Toc199813843)

[1. ВИДЫ УЯЗВИМОСТЕЙ НА ВЕБ-САЙТАХ И СПОСОБЫ ИХ ПРЕДОТВРАЩЕНИЯ 5](#_Toc199813844)

[1.1 Безопасность веб-ресурсов 5](#_Toc199813845)

[1.2 Актуальные угрозы веб-безопасности (OWASP) 6](#_Toc199813846)

[1.3 Протоколы HTTP и HTTPS 10](#_Toc199813847)

[1.4 Межсайтовый скриптинг (XSS) 11](#_Toc199813848)

[1.5 SQL инъекции 13](#_Toc199813849)

[1.6 CSRF (Cross-Site Request Forgery) 16](#_Toc199813850)

[2. ИМИТАЦИЯ АТАК И ТЕСТИРОВАНИЕ СПОСОБОВ ИХ ПРЕДОТВРАЩЕНИЯ 21](#_Toc199813851)

[2.1 Используемые технологии 21](#_Toc199813852)

[2.2 Подготовка тестовой среды 22](#_Toc199813853)

[2.3 Применение XSS и защита от атаки 22](#_Toc199813854)

[2.4 Применение SQLI и защита от атаки 28](#_Toc199813855)

[2.5 Применение CSRF и защита от атаки 31](#_Toc199813856)

[ЗАКЛЮЧЕНИЕ 36](#_Toc199813857)

[СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ 37](#_Toc199813858)

[ПРИЛОЖЕНИЯ 38](#_Toc199813859)

# **ВВЕДЕНИЕ**

В современном информационном обществе, где веб-сайты стали не только платформой для предоставления информации, но и основным каналом коммуникации, совершения покупок, проведения финансовых операций и многих других деятельностей, обеспечение информационной безопасности (ИБ) становится жизненно важным аспектом. Все больше организаций и частных лиц привлекаются к виртуальному пространству, что увеличивает риск возникновения различных угроз безопасности данных.

Сложившаяся ситуация поднимает ряд важных вопросов о защите данных, сохранности конфиденциальной информации и обеспечении непрерывной работоспособности веб-ресурсов. Угрозы информационной безопасности веб-сайтов могут варьироваться от традиционных атак, таких как SQL инъекции, к краже личных данных и распространению вредоносного программного обеспечения, такого как вредоносные скрипты и вирусы.

Для эффективной защиты веб-сайтов необходимо не только анализировать существующие угрозы, но и разрабатывать и применять соответствующие методы и средства их предотвращения. Это может включать в себя использование современных методов шифрования данных, регулярное обновление программного обеспечения и патчей безопасности, реализацию многоуровневой архитектуры защиты, обучение сотрудников по вопросам информационной безопасности, а также установку и настройку средств обнаружения и предотвращения атак.

Целью работы является проанализировать современные угрозы безопасности на веб-сайтах и выявить способы их предотвращения.

Для решение поставленной цели необходимо реализовать задачи:

* Изучить теоретическую составляющую современных угроз;
* Проанализировать методы реализации угроз на веб-сайты, а также их предотвращения;
* Проанализировать какие методы атак и защиты используются злоумышленниками;
* Создать среду для тестирования методов защиты от атак;
* Провести атаки на тестовую среду;
* Обеспечить защиту тестовой среды от атак.

# **ВИДЫ УЯЗВИМОСТЕЙ НА ВЕБ-САЙТАХ И СПОСОБЫ ИХ ПРЕДОТВРАЩЕНИЯ**

## 1.1 Безопасность веб-ресурсов

Временами можно услышать о ситуациях, когда веб-сайты становятся недоступными из-за атак, таких как DDoS (атака на отказ в обслуживании), или из-за изменения отображаемой информации на страницах, часто с повреждением контента. В других случаях, миллионы учетных записей, включая пароли, адреса электронной почты и данные кредитных карт, могут становиться общедоступными, подвергая пользователей веб-сайта личному недовольству, чаще к финансовым рискам.

Цель веб-безопасности заключается в предотвращении любых видов атак. Более формальным определением веб-безопасности является – способ защиты веб-сайтов от несанкционированного доступа, использования, изменения, уничтожения или нарушения работы.

Для эффективной безопасности веб-сайта необходимо уделять особое внимание к разработке: веб-приложения, конфигурации веб-сервера, при написании политик создания и обновления паролей, а также кода на стороне клиента. Хотя все это звучит очень зловеще, хорошая новость заключается в том, что если используется веб-фреймворк для серверной части, то он обеспечит «по умолчанию» надёжные и продуманные механизмы защиты от ряда наиболее распространённых атак. Другие атаки можно смягчить с помощью конфигурации персонального веб-сервера, например, включив HTTPS. Наконец, есть общедоступные инструменты для сканирования уязвимостей, которые могут помочь определить, если были допущены какие-либо очевидные ошибки.

## 1.2 Актуальные угрозы веб-безопасности (OWASP)

В современном веб-пространстве важнейшим аспектом становится защитаприложений от наиболее распространённых уязвимостей. Проект OWASP (Open Web Application Security Project), регулярно публикующий перечень критичных угроз для веб-приложений, остаётся авторитетным источником в этой области. Актуальная редакция OWASP Top 10 охватывает целый спектр уязвимостей, которые нередко приводят к серьёзным последствиям: утечке данных, нарушению конфиденциальности, несанкционированному доступу, а также разрушению целостности цифровых систем. Среди этих рисков выделяются десять ключевых направлений, каждое из которых заслуживает отдельного внимания. Понимание этих угроз необходимо не только для программистов, но и для системных администраторов, архитекторов решений, а также для специалистов по кибербезопасности.

Одной из наиболее опасных проблем остаётся нарушение контроля доступа. Это происходит, когда ограничения по правам пользователей реализованы формально или частично. Например, при ручной подстановке URL злоумышленник может получить доступ к закрытому ресурсу, если проверка прав осуществляется только на клиентской стороне или недостаточно строго на сервере. Такие ошибки встречаются как в небольших проектах, так и в крупных корпоративных системах. Часто уязвимости связаны с отсутствием централизованной авторизации или с прямыми ID-идентификаторами в URL. Чтобы этого избежать, необходимо применять многоуровневую валидацию прав, внедрять шаблоны безопасной разработки (например, RBAC), использовать проверку привилегий на всех уровнях логики приложения и внедрять системные журналы аудита.

Не менее серьёзную угрозу представляет неправильное применение криптографических механизмов. Использование слабых алгоритмов, устаревших протоколов или хранение ключей в доступных местах делает систему уязвимой для дешифровки. Часто разработчики допускают ошибки, полагаясь на устаревшие методы хеширования или, что ещё хуже, хранят пароли в открытом виде. Такие упущения дают злоумышленникам возможность перехватывать или подбирать информацию, предназначенную для защиты. Чтобы обезопасить данные, необходимо использовать современные алгоритмы, такие как bcrypt, scrypt или Argon2, строго разграничивать доступ к ключевым файлам, не допускать хранения ключей в открытом коде и использовать TLS последних версий. Особое внимание должно уделяться сертификации и управлению жизненным циклом ключей.

Инъекции — ещё один опасный класс уязвимостей. К ним относятся SQL-инъекции, внедрение команд операционной системы, XSS и другие типы атак, при которых пользовательские данные интерпретируются как исполняемый код. Причиной их возникновения становится недостаточная фильтрация входных данных, отсутствие параметризации запросов и доверие к неконтролируемым источникам. Последствия таких атак могут быть разрушительными: от получения полного доступа к базе данных до внедрения вредоносного кода на клиентскую сторону. Эффективная защита от инъекций строится на строгой фильтрации всех внешних входов, использовании ORM или средств подготовки запросов, отказе от прямого формирования SQL-команд и постоянном аудите кода.

В современных условиях необходимо уделять внимание не только конкретным реализациям, но и архитектуре самого приложения. Под «небезопасным дизайном» подразумеваются ошибки, допущенные ещё на стадии проектирования, когда безопасность не была встроена в основу решения. К подобным недостаткам можно отнести отсутствие ограничения по числу запросов, недостаточную сегментацию сетей, публичный доступ к административным интерфейсам, чрезмерно доверительные политики CORS и другие недочёты. Чтобы избежать проблем, связанных с архитектурной уязвимостью, необходимо проводить моделирование угроз, использовать шаблоны безопасного проектирования, регулярно пересматривать архитектурные решения и интегрировать безопасность в каждый этап жизненного цикла разработки.

Ошибки конфигурации — одна из самых распространённых причин взлома веб-приложений. Часто разработчики оставляют включённым режим отладки, используют стандартные логины и пароли, не отключают неиспользуемые модули или оставляют конфиденциальные данные в открытых конфигурационных файлах. Всё это предоставляет злоумышленникам широкие возможности для атак. Примером может быть доступ к административной панели без пароля или с использованием известного дефолтного аккаунта. Эффективной мерой защиты служит жёсткий контроль конфигураций на всех этапах развертывания, автоматизация безопасных шаблонов (Infrastructure as Code), проверка параметров с помощью CI-инструментов и минимизация уровня привилегий.

Особое место занимает проблема управления компонентами с известными уязвимостями. Современные веб-приложения часто используют десятки, а порой и сотни сторонних библиотек, многие из которых не обновляются должным образом. Из-за этого даже обновлённое приложение может оказаться уязвимым, если его зависимость содержит известную брешь. Чтобы минимизировать подобные риски, важно регулярно использовать автоматические сканеры уязвимостей, отслеживать CVE-отчёты, поддерживать актуальность зависимостей и не допускать использования устаревших или заброшенных проектов в продакшене.

Проблемы аутентификации и идентификации пользователей всё ещё являются причиной множества успешных атак. К наиболее распространённым относятся подбор паролей, атаки типа «credential stuffing», а также использование украденных сессий. Недостатки в логике входа могут привести к захвату учётной записи или эскалации привилегий. Устранить эти угрозы позволяет многофакторная аутентификация, внедрение алгоритмов анализа поведения (например, географических аномалий), а также строгий контроль сессий и их сроков действия.

Не стоит упускать из вида и вопрос защищённости цепочки поставки программного обеспечения. Внедрение вредоносного кода на стадии сборки или доставки приложения способно привести к заражению даже тех систем, где пользователи соблюдают все меры предосторожности. Подобные атаки всё чаще фиксируются в практике крупных компаний. Решением становится внедрение DevSecOps-практик, контроль целостности бинарных файлов, ограничение доступа к средам сборки, а также цифровая подпись артефактов.

Кроме того, качественная защита невозможна без эффективного логирования и мониторинга. Отсутствие механизмов отслеживания действий пользователей и системных событий приводит к тому, что атаки остаются незамеченными. Надёжная система журналирования должна фиксировать все значимые действия, сохранять логи в защищённом хранилище и обеспечивать их своевременный анализ. А автоматизированный мониторинг должен обнаруживать аномалии в поведении и своевременно уведомлять администраторов о возможных инцидентах.

Наконец, стоит отметить уязвимость, связанную с SSRF (Server-Side Request Forgery). Если приложение позволяет указывать внешние адреса, по которым затем осуществляется запрос, это может быть использовано для обращения к внутренним ресурсам инфраструктуры, включая локальные метаданные облачных сервисов. Такие атаки особенно опасны в виртуализированных и облачных средах. Предотвращение SSRF требует использования валидации вводимых URL, строгой фильтрации адресов, изоляции сетевых компонентов и внедрения списков разрешённых IP-диапазонов.

Топ OWASP служит практическим ориентиром для специалистов в области кибербезопасности и разработки, позволяя сосредоточить усилия на устранении наиболее опасных и распространённых уязвимостей. Постоянный анализ, тестирование и внедрение надёжных архитектурных решений остаются основой устойчивости веб-приложений перед лицом эволюционирующих угроз.

## 1.3 Протоколы HTTP и HTTPS

HTTP – это протокол прикладного уровня модели сетевого взаимодействия Open Systems Interconnection (OSI). Она определяет несколько типов запросов и ответов. Например, если необходимо просмотреть данные с веб-сайта, отправляется запрос HTTP GET. Если необходимо отправить информацию, например, заполнить контактную форму, отправляется запрос HTTP POST.

HTTP передает незашифрованные данные, а это означает, что информация, отправленная из браузера, может быть перехвачена и прочитана третьими лицами. Этот процесс не идеален, поэтому он был расширен до HTTPS, чтобы повысить уровень безопасности взаимодействия. HTTPS объединяет HTTP-запросы и ответы с технологиями SSL и TLS.

Веб-сайты HTTPS должны получить сертификат SSL/TLS от независимого центра сертификации (CA). Веб-ресурсы передают сертификат браузеру, а затем обмениваются данными для установления доверия. Также SSL-сертификат содержит криптографическую информацию, поэтому сервер и веб-браузеры могут обмениваться зашифрованными данными. Процесс работает следующим образом:

1. Пользователь открывает веб-сайт HTTPS, введя формат URL-адреса «https://» в адресной строке браузера;
2. Браузер пытается проверить подлинность сайта, запросив SSL-сертификат сервера;
3. В ответ сервер отправляет сертификат SSL, содержащий открытый ключ;
4. SSL-сертификат веб-сайта подтверждает личность сервера. Как только браузер удовлетворен, используется открытый ключ для шифрования и отправки сообщения, содержащего секретный ключ сеанса;
5. Веб-сервер использует персональный закрытый ключ для расшифровки сообщения и получения ключа сеанса. Затем шифрует сеансовый ключ и отправляет подтверждающее сообщение в браузер;
6. После завершения и браузер, и веб-сервер переходят на использование одинакового сеансового ключа для безопасного обмена сообщениями.

## 1.4 Межсайтовый скриптинг (XSS)

Межсайтовый скриптинг (XSS или *Cross-Site Scripting* ) это термин, используемый для описания типа атак, которые позволяют злоумышленнику внедрять вредоносный код через веб-сайт в браузеры других пользователей. Поскольку внедрённый код поступает в браузер с сайта, он является доверенным и может выполнять действие отправки авторизационного файла *cookie* пользователя злоумышленнику. Когда у злоумышленника есть файл *cookie*, он имеет возможность войти на сайт, как обычный пользователь, и проделать необходимые операции, например, получить доступ к данным кредитной карты, просмотреть контактные данные или изменить пароли.

Существует множество различных видов XSS атак, рассмотрим некоторые из них:

1. **Отражённые XSS-уязвимости.** Злоумышленник внедряет вредоносный код, обычно в виде скрипта в параметры URL, формы или другие места, где ввод пользователя отображается на странице. Вредоносный код отражается обратно на пользователя при выполнении запроса к серверу. Например, злоумышленник может создать вредоносную ссылку, и если пользователь перейдет по ней, то вредоносный код выполнится в браузере. Атаки этого типа часто используют социальную инженерию для убеждения пользователя в выполнении определенных действий;
2. **Хранимые XSS-уязвимости.** Внедряется вредоносный код, например, в комментарии, формы, или другие места, где данные сохраняются на сервере и впоследствии отображаются для других пользователей. Когда иные пользователи просматривают эти данные, вредоносный код выполняется в браузерах владельцев. Примером может быть атака, при которой злоумышленник добавляет вредоносный скрипт в комментарии на форуме. Когда другие пользователи просматривают комментарий, скрипт выполняется в браузерах, что может привести к утечке конфиденциальной информации;
3. **DOM-based XSS.** Это форма атаки скриптинга, которая происходит на стороне клиента (в браузере пользователя). Вместо того чтобы направленно внедрять вредоносный код на сервере, как это происходит в других типах XSS, злоумышленник эксплуатирует уязвимости в дереве объектной модели документа (DOM) браузера. Простыми словами, когда веб-сайт не безопасно обрабатывает пользовательский ввод и позволяет злоумышленнику вставить вредоносный код в адресную строку или другие элементы страницы, браузер может неправильно интерпретировать этот код. Злоумышленник имеет возможность создать специальные ссылки или ввод, которые при выполнении приводят к изменению содержимого страницы или совершению вредоносных действий в контексте пользователя.

Пример DOM-based XSS может быть реализован, когда сайт использует JavaScript для динамического обновления содержимого страницы без должной фильтрации ввода. Если пользователь вставляет в текст ссылки вредоносный код, который впоследствии выполнится в браузере других пользователей при переходе по этой ссылке, то операция может привести к небезопасному выполнению скрипта на стороне клиента.

Также, рассмотрим некоторые способы предотвращения XSS атак:

* **Экранирование и кодирование вывода данных**: Все данные, вводимые пользователем и отображаемые на веб-страницах, должны быть экранированы или закодированы перед тем, как они будут отображены в HTML, JavaScript или других контекстах. Это поможет предотвратить внедрение вредоносного кода на страницу;
* **Использование HTTP заголовков Content Security Policy (CSP):** CSP позволяет вам указать браузеру, какие источники содержимого разрешены для конкретных ресурсов на странице. Это помогает предотвратить выполнение вредоносного JavaScript-кода, внедряемого через XSS;
* **Валидация и фильтрация ввода данных:** на серверной стороне следует осуществлять строгую валидацию и фильтрацию ввода данных, чтобы исключить возможность внедрения вредоносного кода. Это особенно важно для данных, которые могут быть отображены на веб-странице;
* **Использование HTTPOnly и Secure флагов для куки:** Установка флага HTTPOnly для куки помогает предотвратить доступ к куки через JavaScript, тем самым снижая риск XSS атак. Флаг Secure обеспечивает передачу куки только через защищенное соединение HTTPS;
* **Регулярное обновление и защита браузеров и фреймворков:** Регулярное обновление используемых браузеров и фреймворков важно для получения исправлений уязвимостей, в том числе уязвимостей, связанных с XSS;
* **Мониторинг и журналирование:** Регулярный мониторинг веб-приложений и журналирование активности могут помочь выявить и предотвратить XSS атаки на ранних стадиях.

## 1.5 SQL инъекции

SQL-инъекция (SQL Injection, SQLI) является одной из наиболее распространённых и опасных уязвимостей веб-приложений.

Атака заключается во внедрении злоумышленником вредоносных фрагментов SQL-кода в поля ввода веб-приложения. В результате исходный SQL-запрос модифицируется таким образом, что выполняются дополнительные нежелательные команды. Целью SQL-инъекции может быть получение несанкционированного доступа к данным или выполнение произвольных команд на сервере.

SQL-инъекции становятся возможными из-за недостаточной проверки пользовательского ввода. Когда веб-приложение формирует SQL-запросы конкатенацией строк, злоумышленник может включить в ввод конструкции, изменяющие логику запроса. Такое поведение связано с тем, что многие приложения напрямую подставляют пользовательские данные в SQL-запросы без разделения структуры команды и её параметров. Из-за этого даже случайный символ или сочетание символов может изменить смысл запроса на выгодный атакующему.

Существуют несколько основных разновидностей SQL-инъекций. Классическая (внутриполосная, In-band) SQL-инъекция предполагает, что злоумышленник использует один и тот же канал связи для внедрения вредоносного кода и получения результатов его выполнения. В эту категорию входят, в частности, UNION-инъекции и инъекции на основе сообщений об ошибках (error-based). При error-based-инъекции злоумышленник использует текстовые сообщения об ошибках СУБД для получения сведений о структуре и данных базы. При UNION-инъекции злоумышленник объединяет результаты своих запросов с ответом легитимного запроса, что позволяет извлекать дополнительную информацию из базы данных.

Слепая (Inferential) SQL-инъекция применяется в ситуациях, когда приложение не возвращает результаты запросов напрямую. В этом случае злоумышленник задаёт серверу СУБД условные запросы и анализирует реакцию приложения (например, изменение текста страницы или времени отклика) для извлечения данных. Выделяют два подхода: булевую (boolean-based) и основанную на времени (time-based) инъекции. При булевой инъекции сервер возвращает «да» или «нет» в зависимости от истинности условия, что позволяет постепенно восстановить данные по битам. При инъекции на основе времени злоумышленник добавляет в запрос условные задержки (например, использование функции SLEEP). По разнице во времени отклика он определяет истинность условия и, таким образом, получает информацию о содержимом базы данных.

К отдельной категории относятся внеполосные (Out-of-band) SQL-инъекции, при которых злоумышленник настраивает передачу результата по альтернативному каналу (например, посредством DNS-запроса или HTTP-запроса к удалённому серверу). Эта техника применяется реже и требует особых условий (например, наличия сетевых возможностей в СУБД), но позволяет получать информацию, не полагаясь на стандартный ответ веб-приложения.

Для защиты от SQL-инъекций важен принцип наименьших привилегий при работе с базой данных. Это означает, что веб-приложение должно использовать учётную запись с ограниченными правами, имеющую доступ только к необходимым таблицам и операциям. Например, такая учётная запись не должна обладать правами DROP или DELETE на критичные таблицы, если их удаление не требуется для работы приложения. Ограничение прав доступа существенно снижает потенциал вреда в случае успешной атаки, поскольку злоумышленнику не удастся выполнить недопустимые операции.

Важным механизмом защиты является проверка и фильтрация пользовательского ввода (санитизация). При формировании SQL-запросов нельзя напрямую использовать необработанные данные. Применяются методы экранирования специальных символов, удаление неподходящих конструкций и ограничение формата вводимых значений. Часто используют белые списки — заранее определённые наборы допустимых значений или шаблоны. При таком подходе принимается только заранее разрешённый ввод, что исключает любые неподходящие символы или структуры.

Использование параметризованных запросов и ORM также эффективно снижает риск SQL-инъекций. В параметризованных запросах структура SQL-оператора отделена от значений параметров (так называемых «подготовленных операторов»), что исключает прямую подстановку пользовательских данных в текст команды. ORM (Object-Relational Mapping) и многие современные фреймворки веб-разработки автоматически формируют безопасные запросы, используя параметры и экранирование. Благодаря этому пользовательские данные передаются в базу отдельно и не интерпретируются как часть SQL-кода.

Несмотря на относительную простоту реализации, атаки типа SQL-инъекции до сих пор нередко приводят к компрометации баз данных, нарушению конфиденциальности и потере управления над информационными системами. Недооценка этой уязвимости может обернуться серьёзными последствиями как для пользователей, так и для бизнеса в целом. Поэтому при разработке и сопровождении программного обеспечения необходимо системно подходить к контролю ввода данных, использовать безопасные методы обращения к базам данных и применять проверенные механизмы защиты.

## 1.6 CSRF (Cross-Site Request Forgery)

CSRF (Cross-Site Request Forgery), или межсайтовая подделка запроса, представляет собой тип атаки на веб-приложения, при котором злоумышленник инициирует выполнение нежелательных действий от имени авторизованного пользователя. Основная особенность данной уязвимости заключается в том, что пользователь, находясь в активной сессии на целевом сайте, переходит на вредоносную страницу, содержащую скрытую форму или скрипт. Эта страница незаметно для пользователя отправляет поддельный HTTP-запрос к доверенному сайту, где пользователь уже прошёл аутентификацию. В результате, сервер, получив запрос с валидными cookie или токенами, ошибочно считает его легитимным и выполняет требуемое действие.

Особую опасность CSRF представляет в тех случаях, когда веб-приложение опирается исключительно на cookies для подтверждения личности пользователя, не применяя дополнительных мер валидации запроса. Например, если пользователь авторизован в интернет-банкинге и в это время посещает вредоносный сайт, тот может отправить от его имени запрос на перевод средств. Браузер автоматически приложит авторизационные cookie к такому запросу, и сервер выполнит его, не заподозрив подлога. Это делает CSRF особенно эффективной атакой против пользователей, не подозревающих о происходящем, поскольку вся активность происходит в рамках их собственной сессии.

Существуют разные разновидности атак CSRF. Одной из наиболее коварных является так называемый login CSRF. Суть её заключается в том, что злоумышленник принуждает пользователя войти в приложение под его, атакующего, учётными данными. Жертва не осознаёт произошедшего, продолжает пользоваться приложением, полагая, что работает в собственной учётной записи, а тем временем сохраняет свои персональные данные или другие конфиденциальные сведения в чужом аккаунте. Позже злоумышленник входит в этот аккаунт и получает доступ ко всей введённой информации.

Ещё одним видом CSRF считается persistent, или хранимая CSRF-атака. В этом случае вредоносный код размещается непосредственно на целевом сайте, например, в комментариях или пользовательском контенте. Каждый пользователь, посещающий соответствующую страницу, автоматически инициирует CSRF-запрос, встроенный в HTML или JavaScript. Такая атака крайне опасна, поскольку масштабируется на всех посетителей ресурса и может работать длительное время, пока вредоносный фрагмент не будет удалён.

Последствия CSRF-атак для пользователей и владельцев веб-приложений могут быть критическими. На пользовательском уровне возможно несанкционированное изменение пароля, электронного адреса, выполнение операций с финансами, передача сообщений и другие действия, которые изменяют состояние учётной записи. В административных интерфейсах последствия могут быть ещё более разрушительными, поскольку атака может привести к полной компрометации системы, изменению настроек сайта, удалению пользователей, загрузке вредоносного кода и так далее. Отдельной угрозой является невозможность пользователя обнаружить атаку, так как все действия формально выполняются от его имени и в пределах законной сессии.

Для эффективной защиты от CSRF необходимо внедрение комплекса технических и организационных мер. Наиболее распространённым и надёжным методом считается использование CSRF-токенов. Это случайные уникальные значения, создаваемые сервером для каждой пользовательской сессии и встраиваемые в формы или заголовки запросов. При получении запроса сервер проверяет наличие и корректность токена. Если токен отсутствует или отличается от ожидаемого, запрос отвергается. Это делает невозможным выполнение поддельного запроса, так как злоумышленник не может заранее узнать действительный токен, если только не имеет доступа к браузеру жертвы.

Существует несколько вариантов реализации CSRF-токенов. Один из них — синхронизирующие токены, передающиеся в теле POST-запроса. Ещё один — модель двойной отправки, при которой токен помещается одновременно в cookie и в теле запроса. Сервер сверяет их между собой, и в случае несовпадения блокирует запрос. Такой подход особенно удобен в одностраничных приложениях (SPA), где широко применяются JavaScript и API-запросы.

Кроме токенов, хорошей практикой считается проверка заголовков Referer и Origin. Эти заголовки позволяют серверу убедиться, что запрос был отправлен с того же источника, на который он адресован. Если источник не совпадает или заголовки отсутствуют, сервер может отказать в выполнении запроса. Однако стоит отметить, что некоторые браузеры или сетевые устройства могут блокировать или изменять заголовки, поэтому полагаться исключительно на них не рекомендуется.

Современные браузеры также поддерживают механизм ограничения отправки cookie по признаку источника запроса, реализованный через атрибут SameSite. Значение Strict запрещает отправку cookies при любых переходах с внешних сайтов, а Lax допускает только некоторые безопасные сценарии. Использование SameSite существенно повышает уровень защиты, так как предотвращает автоматическую передачу cookie в междоменных запросах.

Ещё одной дополнительной мерой может быть внедрение подтверждения действия со стороны пользователя. Это может быть, например, CAPTCHA, повторное требование ввода пароля или SMS-подтверждение. Такие механизмы не только исключают автоматическую обработку запросов, но и позволяют убедиться, что инициатор действия действительно является тем, за кого себя выдаёт. Особенно актуальны такие меры при выполнении операций, изменяющих состояние аккаунта или касающихся персональных данных.

Важной частью борьбы с CSRF остаётся грамотная архитектура веб-приложения. Использование фреймворков и библиотек, в которых предусмотрены встроенные механизмы защиты, позволяет минимизировать риск ошибок при реализации собственных решений. Также не следует игнорировать настройку заголовков Content Security Policy (CSP), контроль CORS и ограничение разрешённых методов HTTP-запросов. Всё это вместе формирует многоуровневую защиту, которая позволяет надёжно предотвращать попытки подделки запросов между сайтами.

Проблема CSRF особенно актуальна в условиях, когда пользователи всё чаще взаимодействуют с критически важными функциями через веб-интерфейсы. Отсутствие достаточного уровня защиты в таких сценариях может привести к серьёзным последствиям — от кражи данных до финансовых потерь. Повышение уровня осведомлённости среди разработчиков и внедрение современных механизмов валидации запросов позволяют существенно сократить вероятность успешной атаки и обеспечить стабильную защиту пользователей.

# **ИМИТАЦИЯ АТАК И ТЕСТИРОВАНИЕ СПОСОБОВ ИХ ПРЕДОТВРАЩЕНИЯ**

## 2.1 Используемые технологии

Чтобы разобраться с уязвимостями на веб-сайтах, необходимо сначала понять, что из себя представляет веб-сайт, и с помощью каких технологий, языков программирования или иных инструментов злоумышленники могут взаимодействовать с веб-сайтами.

**HTML:** HTML (Hypertext Markup Language) является языком разметки для создания структуры веб-страниц. Он использует теги для определения различных элементов на странице, таких как заголовки, параграфы, изображения и ссылки. На базе HTML строится дерево элементов DOM (Document Object Model) которое представляет структуру документа. Он предоставляет программам (например, JavaScript) интерфейс для манипуляции содержимым и структурой веб-страницы.

**CSS**: CSS (Cascading Style Sheets) отвечает за стилизацию и внешний вид веб-страниц. С его помощью можно определить цвета, шрифты, расположение элементов и другие аспекты дизайна. Таблица стилей помогает расширять возможности языка HTML, добавляя различные методы манипуляции с текстовой, графической или видео информацией.

**JavaScript:** JavaScript является языком программирования, который выполняется в браузере пользователя. Он добавляет интерактивность и динамическое поведение на веб-страницах, позволяя создавать анимации, обрабатывать события и взаимодействовать с пользователем.

**MySQL:** MySQL – это свободная реляционная система управления базами данных.

**PHP:** PHP – язык программирования для серверной части веб-приложения.

## 2.2 Подготовка тестовой среды

Для реализации веб-приложения был выбран хостинг-провайдер под названием “SpaceWeb”, который предоставляет бесплатный сервер для хостинга одного сайта. Данный хостинг также предоставляет базы данных такие как “MySQL” И “PostgreSQL”, которые будут участвовать в практической работе.

С помощью FTP-клиента “FileZilla” было установлено подключение к удаленному FTP серверу, который содержит файлы тестируемого веб-приложения. Все файлы проекта содержатся на сервере. После вышеописанных действий, сайт готов к работе под доменом выделенном этим же хостингом: “<http://artem39rma.temp.swtest.ru>/”

Для разработки проекта был развернут локальный сервер с помощью программы “Open Server”. Open Server предоставляет доступ к СУБД “MySQL” и “PostgreSQL”, а также к веб-интерфейсам данных СУБД. В данном проекте было принято решение использовать СУБД “MySQL”, а используемый серверный язык программирования “PHP”.

## 2.3 Применение XSS и защита от атаки

Для работы с XSS была разработана веб-страница, которая имитирует привычный блок комментариев в интернете (Рисунок 1). В приложениях Б, В, Г, можно ознакомиться с кодом, который задействован на данной странице. Также, была создана база данных под названием “main” в которой, в свою очередь, была создана таблица с названием “comments”. Подключение к базе данных описано в приложении А. Структура данной таблицы представляет собой:

* **id:** Уникальный идентификатор записи в таблице;
* **textData:** Текст комментария введенный пользователем.

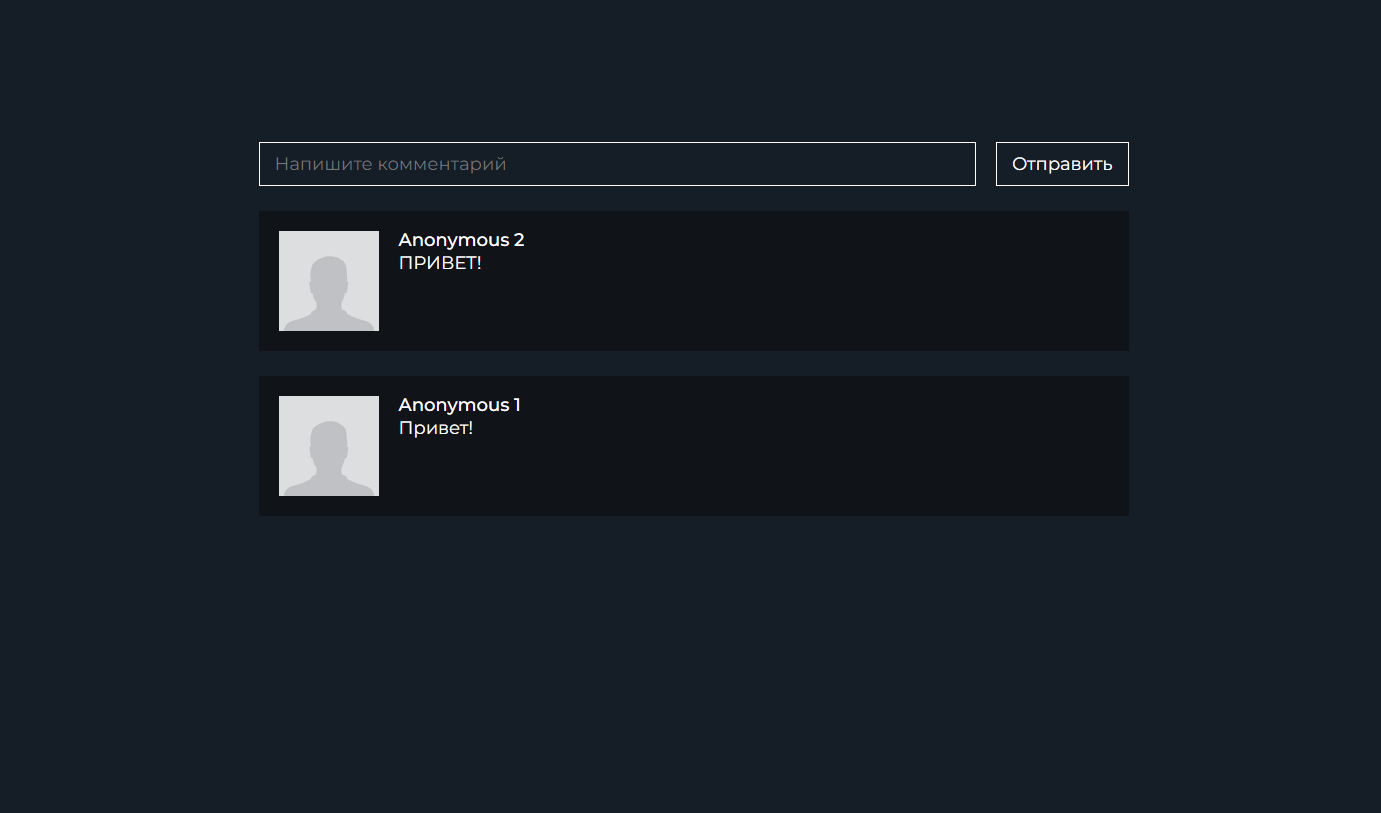


Рис.1. Страница комментариев

На странице присутствует поле ввода, через которое пользователь может ввести текст и сделать запись в базе, а также список комментариев каждый из которых представляет одну запись в базе данных. Но ничто не мешает злоумышленнику вписать в поле ввода скрипт или HTML тэг, который будет подгружаться с сервера прямиком в структуру HTML документа, что позволит злоумышленнику манипулировать элементами на странице. Таким образом, злоумышленник может внедрить свой вредоносный код, который будет выполняться каждый раз при загрузке страницы у всех пользователей, которые зашли на сайт.

Рассмотрим примеры внедрения вредоносного кода в базу данных. Пользователь вводит в поле HTML тэг <img/>, а в атрибут “src”, в котором должна находиться ссылка на картинку, вписывает URL по которому расположена картинка в интернете и отправляет эти данные на сервер (Рисунок 2).

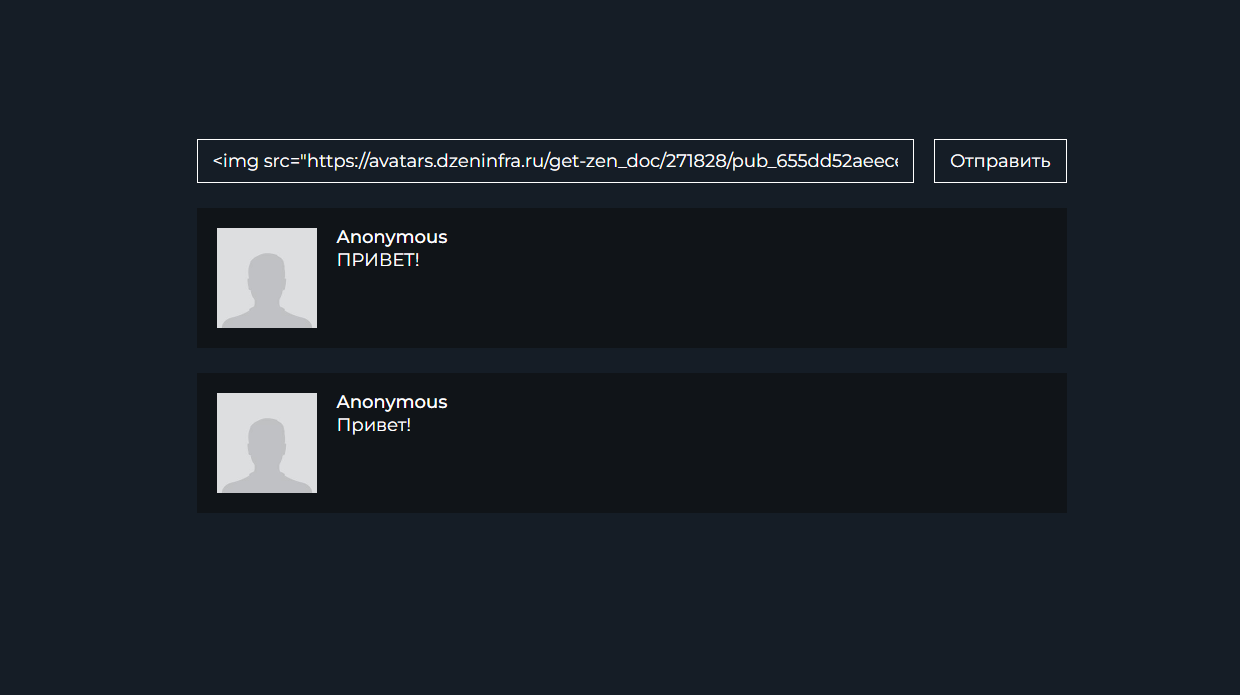


Рис.2. Ввод HTML тэга в поле ввода

После отправки запроса на сервер, и обновления страницы, можно увидеть картинку, которую внедрил пользователь, с помощью уязвимости на сайте (Рисунок 3).

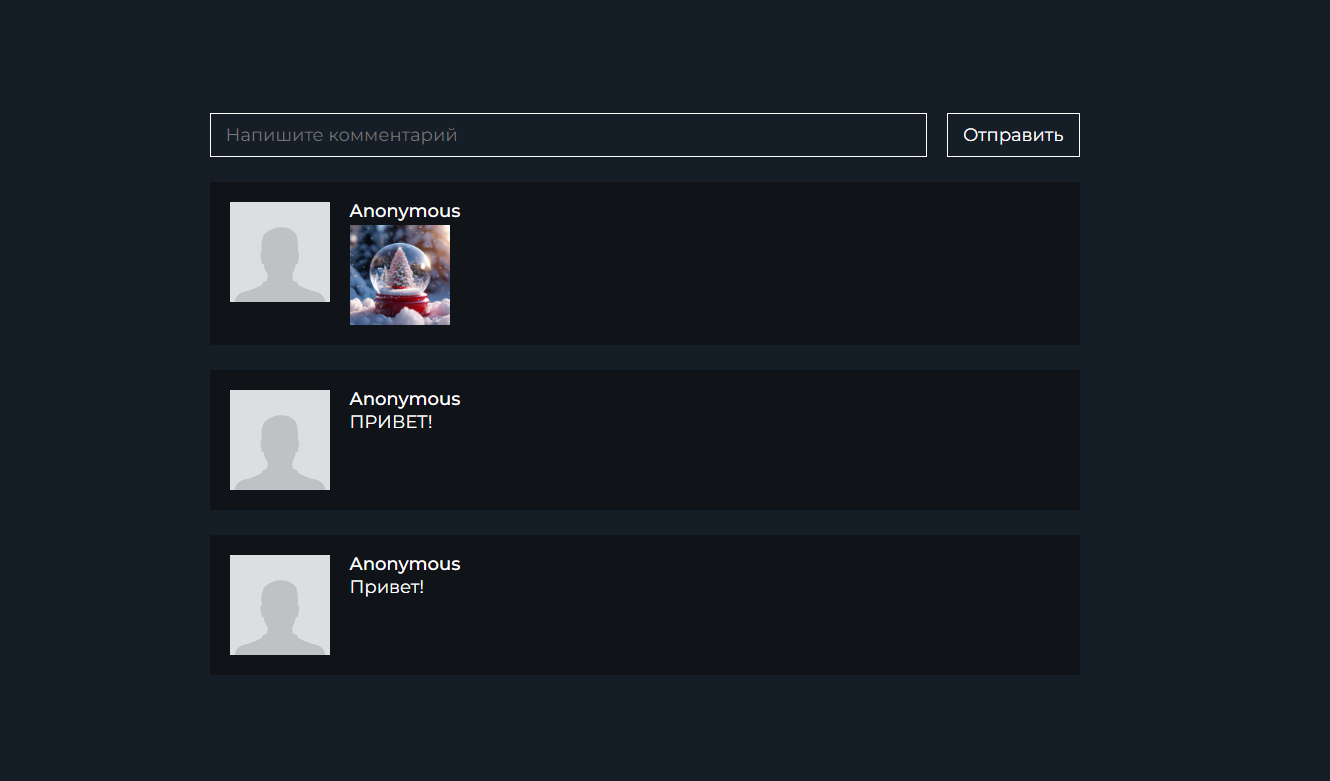


Рис.3. Внедрение картинки в блок комментария

Теперь рассмотрим ситуацию, когда пользователь захотел отправить на сервер не просто тэг с картинкой, а целый скрипт, который способен манипулировать страницей в полной мере. Для этого пользователь должен ввести в строку ввода следующую конструкцию: <script></script>, где между тэгов “script” будет написан JavaScript код. Для начала воспользуемся функцией “alert”, которая открывает диалоговое окно в браузере, и передадим в нее строку “XSS” (Рисунок 4).

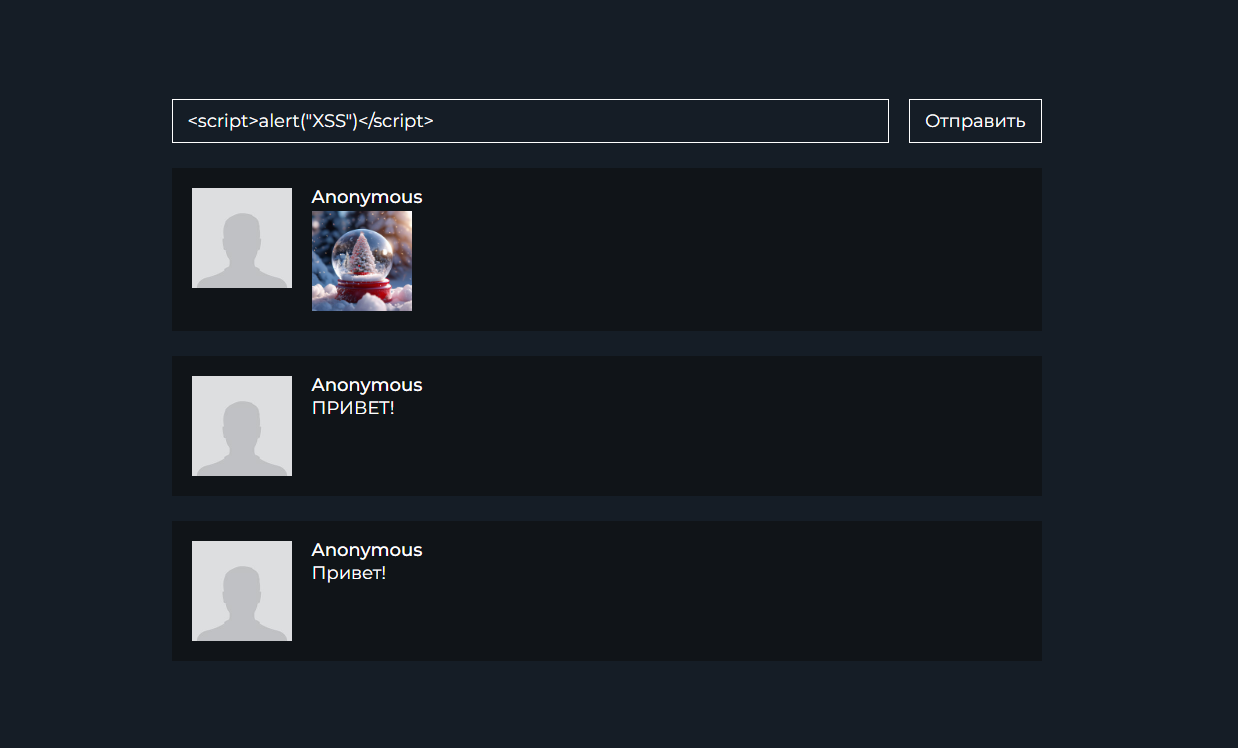


Рис.4. Ввод скрипта в поле ввода

После отправки содержимого на сервер, при переходе на страницу можно увидеть выплывающие диалоговое окно, с переданной ранее строкой “XSS” (Рисунок 5).

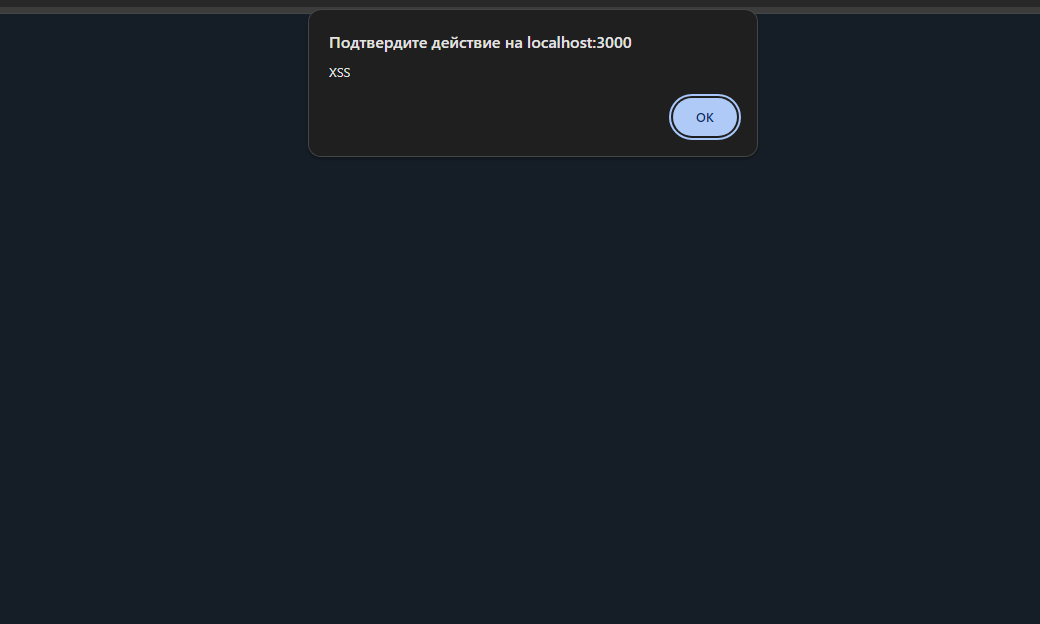


Рис.5. Внедрение скрипта на страницу

Данное диалоговое окно блокирует интерфейс пользователя пока он не скроет его, поэтому появление такого окна при каждой загрузке сайта может вызывать дискомфорт для пользователя. Рассмотренные примеры внедрения вредоносного кода являются безобидными для пользователя, но с помощью данной уязвимости злоумышленник может украсть личные данные или сделать страницу полностью нерабочей.

Теперь рассмотрим случай, когда злоумышленник вовсе может удалить содержимое страницы. Если зайти на страницу и нажать клавишу F12 на клавиатуре, откроются инструменты разработчика, через которые можно посмотреть структуру HTML документа, то есть DOM (Document Object Model) (Рисунок 6).

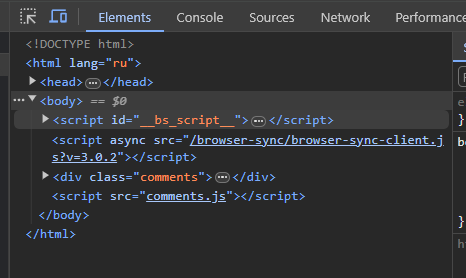


Рис.6. Структура HTML в инструментах разработчика

Для того чтобы удалить содержимое блока комментариев, достаточно знать название класса, присвоенного этому блоку, в данном случае класс называется “comments” Отправляем на сервер следующий скрипт (Рисунок 7):



Рис.7. Скрипт, который удаляет все содержимое блока комментариев

После проделанных действий, при переходе на сайт пользователи вместо привычного блока с комментариями увидят пустую страницу. Таким образом злоумышленник может удалить содержимое конкретного блока или всей страницы.

Чтобы избавиться от данной уязвимости, было принято решение, использовать существующую в языке PHP функцию “htmlspecialchars”, которая экранирует данные, полученные из поля ввода, превращая специальные символы в HTML сущности (Рисунок 8).

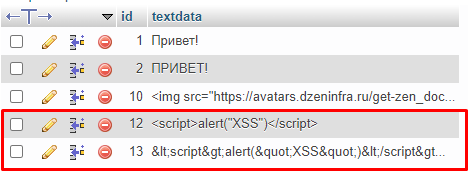


Рис.8. Вредоносный код до и после фильтрации

Чтобы убедиться в том, что экранирование работает, вернемся на страницу и посмотрим, как выглядит последний комментарий (Рисунок 9).

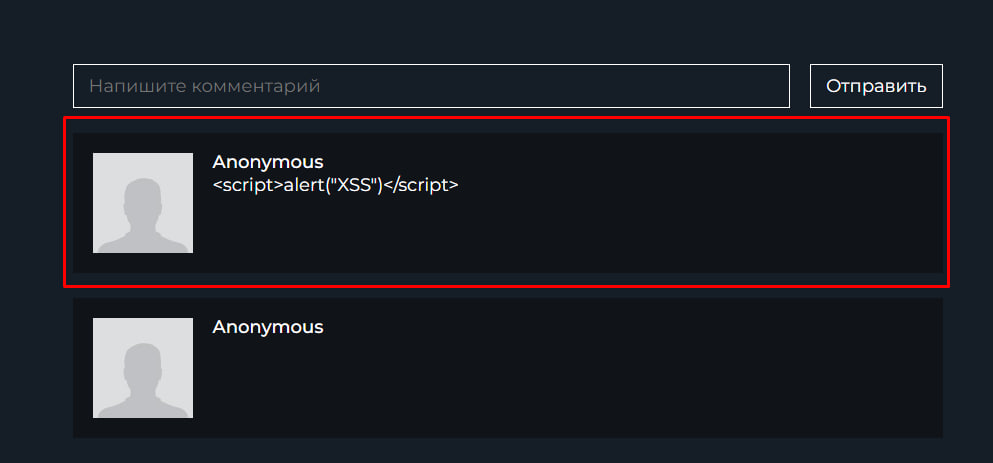


Рис.9. Отображение вредоносного кода в виде строки

Таким образом, была проведена XSS атака на страницу, внедряемый код которой, хранился в базе данных, и влиял на поведение страницы при загрузке. После того, как данные, отправляемые на сервер, были экранированы, уязвимость была устранена. Данные, в свою очередь, не потеряли свой изначальный внешний вид.

## 2.4 Применение SQLI и защита от атаки

Для работы с SQL Инъекциями была разработана веб-страница, которая имитирует форму авторизации в интернете (Рисунок 10). В приложении Д описан код страницы. Также была создана таблица, под названием “users”. Структура данной таблицы представляет собой:

* **id:** Уникальный идентификатор записи в таблице;
* **username:** Имя пользователя;
* **password:** Пароль пользователя;

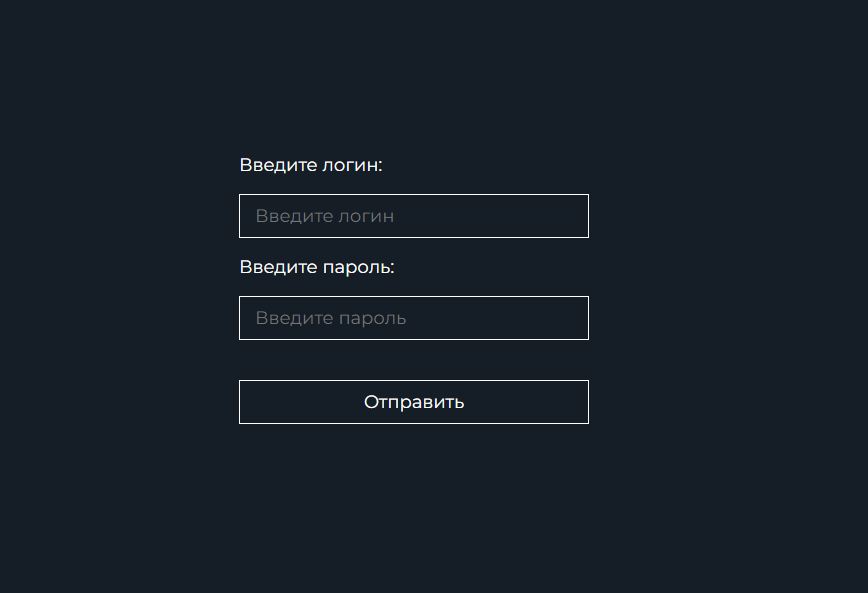


Рис.10. Страница формы авторизации

Таблица users имеет одну запись (Рисунок 11). Эта запись представляет пользователя под именем “admin” с паролем “admin”.

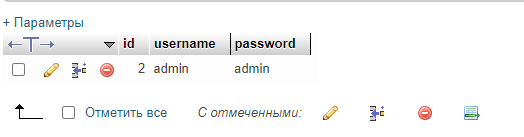


Рис.11. Запись пользователя в базе данных

После того как пользователь вводит данные в форму, его перенаправляет на страницу с сообщением о статусе авторизации: “Ошибка!” либо “Вход успешно выполнен”. Этого функционала уже вполне достаточно, чтобы обзавестись уязвимостью.

Рассмотрим ситуацию, когда злоумышленник с помощью SQL инъекции может успешно авторизоваться. Для начала рассмотрим, как происходит авторизация. Часть кода, которая отвечает за авторизацию находится в приложении Е. После отправки данных на сервер, с помощью SQL запроса делается выборка из базы, которая возвращает запись с теми данными, что совпадают с введенными пользователем данными. Также выводится сообщение о статусе входа.

После отправки данных на сервер, с помощью SQL запроса делается выборка из базы, которая возвращает запись с теми данными, что совпадают с введенными пользователем данными. Также выводится сообщение о статусе входа.

Так как, данные из формы представляют собой строки, злоумышленник может ввести в поля ключевые слова SQL запроса, которые могут полностью изменить выполнение этого SQL запроса, и даже авторизовать его без каких-либо личных данных. Чтобы злоумышленнику добиться работы его введённого вредоносного кода, ему достаточно поставить перед кодом строчную кавычку, которая закроет строку и позволит коду дальше выполниться. Рассмотрим пример: злоумышленник заходит на страницу и вводит в одно из полей ввода следующую строчку: “ ‘ or 1=1 -- b “ (Рисунок 12).

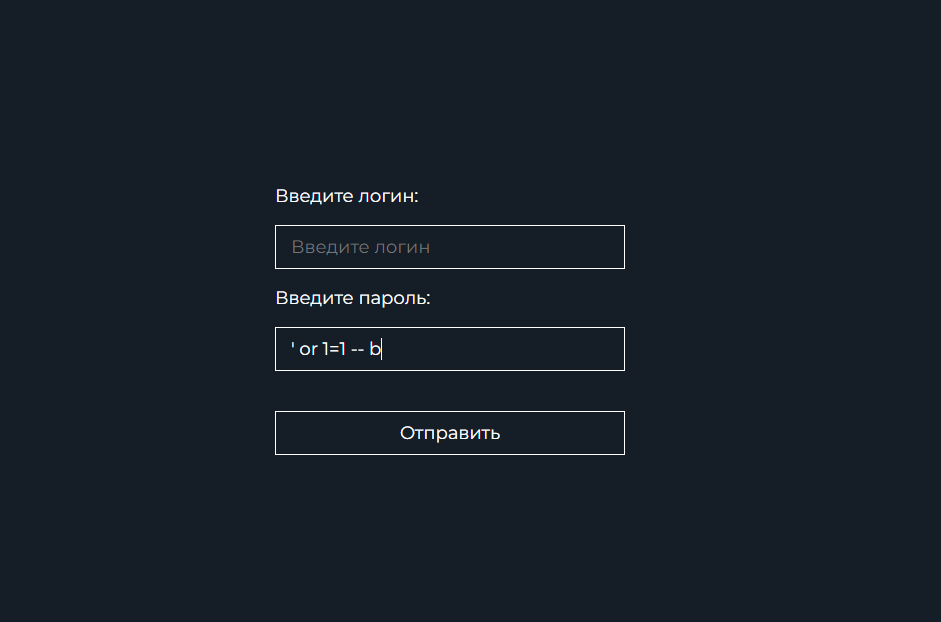


Рис.12. Ввод вредоносной строки в поле ввода

Данное выражение означает, что после закрытой строки, благодаря кавычке, выполняется оператор OR, который в случае, если SQL запрос ранее вернул “false”, вернет свой результат. В представленном случае это 1=1 что равняется истине, и соответственно возвращает “true”. После данного выражения остается запись “-- b”, которая позволяет закомментировать оставшуюся не закрытую кавычку. Чтобы посмотреть наглядно взглянем как это выглядит в коде (Рисунок 13).



Рис.13. Кодовое представление запроса после SQL инъекции

После того, как злоумышленник отправит данные, его перенаправит на страницу, где он увидит сообщение о том, что вход выполнен успешно.

Для защиты от взлома, как и в случае с XSS, важно не допустить выполнения входных данных как кода. Вместо простой экранизации с помощью “htmlspecialchars” или аналогичных функций, в данном случае было принято более надежное решение — использовать параметризованные запросы. Такой подход предотвращает выполнение вредоносных SQL-инструкций, подставляемых через пользовательский ввод. Параметризованные запросы обеспечивают корректную обработку данных и автоматически исключают возможность SQL-инъекций. Однако важно понимать, что ни один метод не дает полной гарантии безопасности. По мере роста и усложнения веб-приложения могут появляться новые уязвимости, требующие различных подходов к защите.

Таким образом была проведена SQL инъекция, в результате которой злоумышленник мог получить доступ к области, веб-приложения которая доступна только авторизированным пользователям. Уязвимость была устранена с помощью параметризованных запросов в PHP, которые обеспечивают безопасную передачу данных в SQL-выражения без риска инъекций.

## 2.5 Применение CSRF и защита от атаки

Для демонстрации CSRF-атаки и способов защиты от неё необходимо реализовать полноценную авторизацию в приложении, так как атака основывается на использовании данных жертвы для выполнения поддельных запросов от её имени. Для реализации авторизации была создана таблица “tokens”, которая соотносит пользователя и его токен авторизации. Для создания токена используется библиотека firebase/php-jwt.  
**JWT (JSON Web Token)** – это современная технология хранения и передачи информации о пользователе, используемая для авторизации при обращении клиента к защищённым ресурсам. JWT востребована из-за своей компактности и способности обеспечивать авторизацию без необходимости сохранять сессии на сервере, что облегчает масштабирование приложений. Токен включает зашифрованные данные и цифровую подпись, позволяя серверу оперативно проверять его подлинность без обращения к базе данных. Структура таблицы “tokens” представляет собой:

* **id:** Уникальный идентификатор записи в таблице;
* **user\_id:** Уникальный идентификатор пользователя из таблицы пользователей “users”
* **token:**  Токен авторизации пользователя (JWT);

Далее был модифицирован файл login.php. Теперь, во время авторизации, сервер устанавливает “accessToken” cookie пользователю в браузер, что позволяет отслеживать его состояние авторизации при каждом запросе, так как браузер отправляет cookie вместе с каждым запросом на сервер.

Для тестирования атаки была создана страница профиля пользователя. При входе на страницу пользователь видит свой логин, и может ввести логин или новый пароль (Рисунок 14).

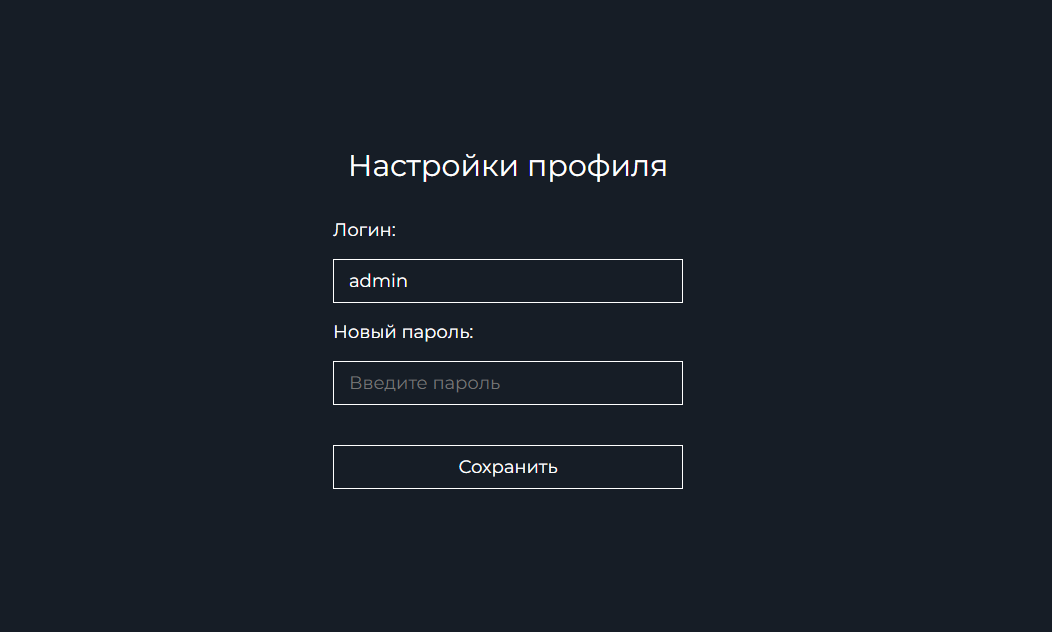


Рис.14. Страница профиля пользователя

Данная страница делает запрос на сервер по нажатию на кнопку сохранить. К запросу прикрепляются cookie, в которых лежит токен пользователя, с помощью которого аутентифицируется пользователь. Если токен валидный, и сервер знает какому пользователю этот токен принадлежит, данные обновляются.

Для демонстрации CSRF, была создана страница, на которой расположена кнопка, по нажатию на которую происходит отправка формы на адрес сервера (Рисунок 15). В этой форме есть уже заполненные поля “login” и “password” (Рисунок 16).

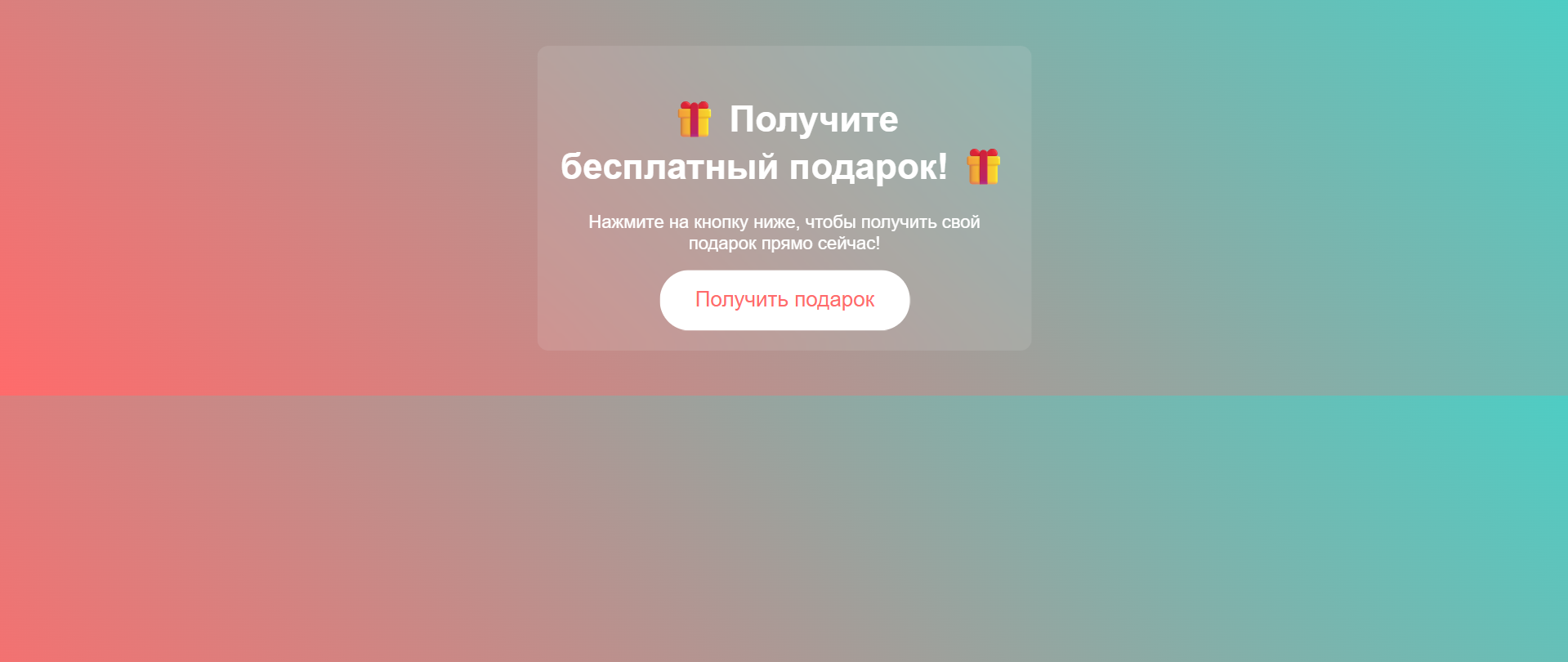


Рис.15. Страница злоумышленника

 <form *id*="form" *action*="http://artem39rma.temp.swtest.ru/update.php" *method*="GET" *style*="display: none">

      <input *type*="hidden" *name*="login" *value*="hacker" />

      <input *type*="hidden" *name*="password" *value*="hackedPassword" />

    </form>

Рис.16. Форма на странице злоумышленника

В момент отправки формы происходит запрос на сервер от лица жертвы, благодаря чему к запросу прикрепляются ее cookie (Рисунок 17).

Сервер проверяет принадлежность токена пользователю и обновляет его данные на те, что указал злоумышленник в своей форме (Рисунок 18).

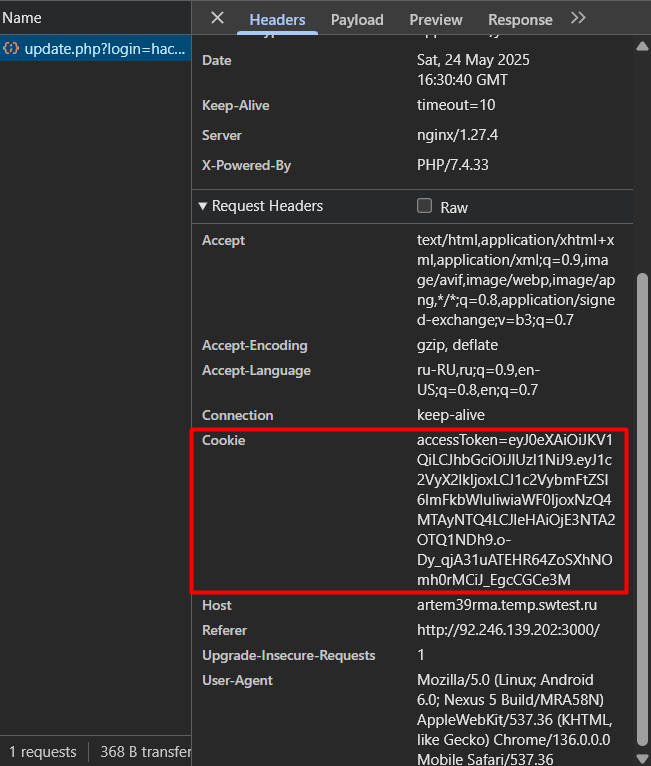


Рис.17. Cookie в запросе, отправленном с другого сайта



Рис.18. Обновленный логин и пароль пользователя

Для выбора способа защиты от CSRF атак, стоить учитывать многие факторы построения веб приложения. Если клиент и сервер находятся на одном домене, использование атрибута “SameSite” со значением “Strict” часто бывает достаточно для предотвращения CSRF атаки, но в таком случае токен авторизации не сможет быть прикреплен к запросам на другой домен в случае, если cookie имеет флаг httponly. В некоторых случаях, токен отсылают на сервер с помощью заголовка Authorization, но это повышает риск XSS атак, так как использование этого заголовка подразумевает ручное прикрепление токена через JavaScript, что делает токен доступным для злоумышленника в случае наличия уязвимости на сайте.

В данном случае все запросы проходят в рамках одного домена, поэтому добавление SameSite – Strict будет достаточно для защиты от уязвимости.

В ходе проделанной работы была успешно продемонстрирована CSRF-атака, основанная на использовании авторизационного токена, хранящегося в cookie. Атака позволила злоумышленнику изменить данные учетной записи пользователя без его ведома, воспользовавшись тем, что браузер автоматически прикрепляет cookie с токеном ко всем запросам на сервер. Это подтвердило уязвимость приложения при отсутствии должной защиты от CSRF.

Для устранения данной уязвимости был реализован один из эффективных способов защиты — установка атрибута SameSite=Strict для cookie, в которой хранится JWT. Этот атрибут предотвращает автоматическую отправку cookie при кросс-доменных запросах, тем самым исключая возможность выполнения CSRF-атаки с другого сайта. Данный способ является подходящим решением в контексте текущего приложения, где клиент и сервер взаимодействуют в пределах одного домена.

Таким образом, реализация механизма авторизации на базе JWT, демонстрация уязвимости и успешное применение защитной меры позволили не только подтвердить опасность CSRF-атак, но и продемонстрировать практический способ их предотвращения в реальных условиях.

# **ЗАКЛЮЧЕНИЕ**

Подводя итоги, были рассмотрены актуальные уязвимости веб-безопасности, и методы их предотвращения. Также, была проведена практическая работа с уязвимостями внедрения вредоносного кода на веб-страницы. Была разработана тестовая среда, на которой были протестированы XSS, CSRF и SQLI атаки и способы их предотвращения.

Так как, уязвимостей в сфере веб-безопасности большое количество, и они появляются каждый день, предотвратить все уязвимости невозможно. Но стоит не забывать тестировать свои страницы на уязвимости и устранять их по возможности. Данная дипломная работа показывает, насколько важно следить за безопасностью своих веб-приложений, поскольку никто не застрахован от внезапного взлома.

По итогам работы, все следующие задачи были выполнены:

* Изучить теоретическую составляющую современных угроз;
* Проанализировать методы реализации угроз на веб-сайты, а также их предотвращение;
* Проанализировать какие методы атак и защиты используются злоумышленниками;
* Создать среду для тестирования методов защиты от атак;
* Провести атаки на тестовую среду;
* Обеспечить защиту тестовой среды от атак.

# **СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ**

1. <https://www.php.net/manual/ru/>
2. <https://ospanel.io/docs/>
3. <https://browsersync.io/>
4. <https://habr.com/ru/articles/511318/>
5. <https://habr.com/ru/companies/alfa/articles/717896/>
6. <http://artem39rma.temp.swtest.ru/>
7. <http://92.246.139.202:3000/>

# **ПРИЛОЖЕНИЯ**

**ПРИЛОЖЕНИЕ А**

Содержимое файла connect.php

<?php

$db **=**  mysqli\_connect("localhost", "root", "root", "main");

?>

**ПРИЛОЖЕНИЕ Б**

Содержимое файла comments.php

**<!DOCTYPE** **html>**

**<html** **lang**="ru"**>**

**<head>**

**<meta** **charset**="UTF-8"**>**

**<meta** **name**="viewport" **content**="width=device-width, initial-scale=1.0"**>**

**<link** **rel**="stylesheet" **href**="styles.css" **/>**

**<title>**Комментарии**</title>**

**</head>**

**<body>**

**<div** **class**="comments"**>**

**<div** **class**="comments\_\_container"**>**

**<form** **novalidate** **method**="post" **enctype**="multipart/form-data" **class**="form comments-form"**>**

**<input** **autocomplete**="off" **name**="textData" **placeholder**="Напишите комментарий" **class**="comments-form\_\_input form\_\_input" **type**="text"**>**

**<button** **name**="submitter" **type**="submit" **class**="form\_\_submit comments-form\_\_submit"**>**Отправить**</button>**

**</form>**

      <?php

**include** "connect.php";

      $data **=** mysqli\_query($db, "**SELECT** **\*** **FROM** `comments`");

      $data **=** mysqli\_fetch\_all($data);

      $data **=** array\_reverse($data);

**foreach** ($data **as** $data) {

      ?>

**<div** **class**="comment"**>**

**<div** **class**="comment\_\_block-img"**>**

**<img** **class**="comment\_\_img" **src**="images/avatar.jpg" **alt**=""**>**

**</div>**

**<div** **class**="comment\_\_block-text"**>**

**<div** **class**="comment\_\_title"**>**Anonymous**</div>**

**<div** **class**="comment\_\_text"**>**<?= $data[1] ?>**</div>**

**</div>**

**</div>**

      <?php

      }

      ?>

**</div>**

**</div>**

**</div>**

**<script** **src**="comments.js"**></script>**

**</body>**

**</html>**

**ПРИЛОЖЕНИЕ В**

Содержимое файла comments.js

**const** form **=** document.querySelector(`.form`);

form.addEventListener("submit", **async** (*e*) **=>** {

  e.preventDefault();

**const** input **=** form.querySelector(`.form\_\_input`);

**const** formData **=** **new** FormData(form);

**if** (input.value) {

**const** response **=** **await** fetch("sendData.php", {

      method: "POST",

      body: formData,

    });

**if** (response.ok) {

      location.reload();

    }

  } **else** {

    alert("Заполните поле!");

  }

});

**ПРИЛОЖЕНИЕ Г**

Содержимое файла sendData.php

<?php

**include** "connect.php";

$textData **=** $\_POST["textData"];

$textData **=** htmlspecialchars($textData);

mysqli\_query($db, "**INSERT INTO** `comments` (`textdata`) **VALUES** ('$textData')");

?>

**ПРИЛОЖЕНИЕ Д**

Содержимое файла form.php

**<!DOCTYPE** **html>**

**<html** **lang**="ru"**>**

**<head>**

**<meta** **charset**="UTF-8" **/>**

**<meta** **name**="viewport" **content**="width=device-width, initial-scale=1.0" **/>**

**<link** **rel**="stylesheet" **href**="styles.css" **/>**

**<title>**Практика**</title>**

**</head>**

**<body>**

**<div** **class**="container"**>**

**<form** **method**="post" **action**="login.php" **novalidate** **autocomplete**="off" **class**="form"**>**

**<label** **class**="form\_\_label" **for**="login"**>**Введите логин:**</label>**

**<input** **placeholder**="Введите логин" **class**="form\_\_input" **name**="login" **type**="text"**>**

**<label** **class**="form\_\_label" **for**="password"**>**Введите пароль:**</label>**

**<input** **placeholder**="Введите пароль" **class**="form\_\_input" **name**="password" **type**="text"**>**

**<button** **type**="submit" **class**="form\_\_submit"**>**Отправить**</button>**

**</form>**

**</div>**

**</body>**

**</html>**

**ПРИЛОЖЕНИЕ Е**

Содержимое файла login.php

<?php

*include* "connect.php";

*require\_once* 'vendor/autoload.php';

*use* \Firebase\JWT\JWT;

$username *=* $\_POST["login"];

$password *=* $\_POST["password"];

$stmt *=* mysqli\_prepare($db, "*SELECT* *\** *FROM* `users` *WHERE* `username` *=* ? *AND* `password` *=* ?");

mysqli\_stmt\_bind\_param($stmt, "ss", $username, $password);

mysqli\_stmt\_execute($stmt);

$result *=* mysqli\_stmt\_get\_result($stmt);

$user *=* mysqli\_fetch\_assoc($result);

$message *=* "";

*if* ($user) {

  $key *=* "your-secret-key";

  $payload *=* array(

    "user\_id" *=>* $user['id'],

    "username" *=>* $user['username'],

    "iat" *=>* time(),

    "exp" *=>* time() *+* (86400 *\** 30)

  );

  $token *=* JWT*::*encode($payload, $key, 'HS256');

  $stmt *=* mysqli\_prepare($db, "*SELECT* id *FROM* tokens *WHERE* user\_id *=* ?");

  mysqli\_stmt\_bind\_param($stmt, "i", $user['id']);

  mysqli\_stmt\_execute($stmt);

  $existingToken *=* mysqli\_fetch\_assoc(mysqli\_stmt\_get\_result($stmt));

*if* ($existingToken) {

    $stmt *=* mysqli\_prepare($db, "*UPDATE* tokens *SET* token *=* ? *WHERE* user\_id *=* ?");

    mysqli\_stmt\_bind\_param($stmt, "si", $token, $user['id']);

  } *else* {

    $stmt *=* mysqli\_prepare($db, "*INSERT INTO* tokens (user\_id, token) *VALUES* (?, ?)");

    mysqli\_stmt\_bind\_param($stmt, "is", $user['id'], $token);

  }

  mysqli\_stmt\_execute($stmt);

  $cookieSet *=* setcookie(

    'accessToken',

**ПРОДОЛЖЕНИЕ ПРИЛОЖЕНИЯ Е**

    $token,

    [

      'expires' *=>* time() *+* (86400 *\** 30),

      'path' *=>* '/',

      'samesite' *=>* 'Strict',

    ]

  );

*if* (*!*$cookieSet) {

    error\_log("Не удалось установить куки");

  }

  $message *=* "Вход успешно выполнен!";

} *else* {

  $message *=* "Ошибка!";

}

?>

<!DOCTYPE *html*>

<html *lang*="ru">

<head>

  <meta *charset*="UTF-8">

  <meta *name*="viewport" *content*="width=device-width, initial-scale=1.0">

  <title><?= htmlspecialchars($message) ?></title>

</head>

<body *style*="

    display: flex;

    align-items: center;

    justify-content: center;

    background: black;

    color: white;

    font-size: 50px;

    margin: 0;

    min-height: 100vh;

">

  <div><?= htmlspecialchars($message) ?></div>

</body>

</html>

**ПРИЛОЖЕНИЕ ?**

Содержимое файла profile.php

<?php

*include* "connect.php";

*require\_once* 'vendor/autoload.php';

*use* \Firebase\JWT\JWT;

$userLogin *=* "";

$error *=* "";

*if* (isset($\_COOKIE['accessToken'])) {

*try* {

    $key *=* "your-secret-key";

    $token *=* $\_COOKIE['accessToken'];

    $decoded *=* JWT*::*decode($token, $key, array('HS256'));

    $stmt *=* mysqli\_prepare($db, "*SELECT* username *FROM* users *WHERE* id *=* ?");

    mysqli\_stmt\_bind\_param($stmt, "i", $decoded*->*user\_id);

    mysqli\_stmt\_execute($stmt);

    $result *=* mysqli\_stmt\_get\_result($stmt);

    $user *=* mysqli\_fetch\_assoc($result);

*if* ($user) {

      $userLogin *=* $user['username'];

    } *else* {

      header("Location: form\_protected.php");

*exit*();

    }

  } *catch* (Exception $e) {

    header("Location: form\_protected.php");

*exit*();

  }

} *else* {

  header("Location: form\_protected.php");

*exit*();

}

?>

<!DOCTYPE *html*>

<html *lang*="ru">

<head>

  <meta *charset*="UTF-8">

  <meta *name*="viewport" *content*="width=device-width, initial-scale=1.0">

  <link *rel*="stylesheet" *href*="styles.css" />

  <title>Настройки профиля</title>

</head>

**ПРОДОЛЖЕНИЕ ПРИЛОЖЕНИЯ ?**

<body>

  <div *class*="container">

    <h1 *style*="margin-bottom: 40px;">Настройки профиля</h1>

    <div *id*="message" *style*="display: none; margin-bottom: 20px; padding: 10px; border-radius: 5px;"></div>

    <form *id*="profileForm" *method*="post" *action*="update.php" *novalidate* *autocomplete*="off" *class*="form">

      <label *class*="form\_\_label" *for*="login">Логин:</label>

      <input *placeholder*="Введите логин" *class*="form\_\_input" *name*="login" *type*="text" *value*="<?= htmlspecialchars($userLogin) ?>">

      <label *class*="form\_\_label" *for*="password">Новый пароль:</label>

      <input *placeholder*="Введите пароль" *class*="form\_\_input" *name*="password" *type*="text">

      <button *type*="submit" *class*="form\_\_submit">Сохранить</button>

    </form>

  </div>

  <script>

    document.getElementById('profileForm').addEventListener('submit', *function*(*e*) {

      e.preventDefault();

*const* formData *=* *new* FormData(*this*);

*const* messageDiv *=* document.getElementById('message');

      fetch('update.php', {

          method: 'POST',

          body: formData

        })

        .then(*response* *=>* response.json())

        .then(*data* *=>* {

          messageDiv.style.display *=* 'block';

*if* (data.success) {

            messageDiv.style.backgroundColor *=* '#d4edda';

            messageDiv.style.color *=* '#155724';

          } *else* {

            messageDiv.style.backgroundColor *=* '#f8d7da';

            messageDiv.style.color *=* '#721c24';

          }

          messageDiv.textContent *=* data.message *||* data.error;

        })

        .catch(*error* *=>* {

          messageDiv.style.display *=* 'block';

          messageDiv.style.backgroundColor *=* '#f8d7da';

          messageDiv.style.color *=* '#721c24';

          messageDiv.textContent *=* 'Произошла ошибка при обновлении данных';

        });

    });

  </script>

</body>

</html>

**ПРИЛОЖЕНИЕ ?**

Содержимое файла update.php

<?php

*include* "connect.php";

*require\_once* 'vendor/autoload.php';

*use* \Firebase\JWT\JWT;

header('Content-Type: application/json');

*if* ($\_SERVER['REQUEST\_METHOD'] *===* 'POST') {

  $requestData *=* $\_POST;

} *else* {

  $requestData *=* $\_GET;

}

*if* (*!*isset($\_COOKIE['accessToken'])) {

  http\_response\_code(401);

  echo json\_encode(['error' *=>* 'Требуется авторизация']);

*exit*();

}

*try* {

  $key *=* "your-secret-key";

  $token *=* $\_COOKIE['accessToken'];

  $decoded *=* JWT*::*decode($token, $key, array('HS256'));

  $userId *=* $decoded*->*user\_id;

  $stmt *=* mysqli\_prepare($db, "*SELECT* username, *password* *FROM* users *WHERE* id *=* ?");

  mysqli\_stmt\_bind\_param($stmt, "i", $userId);

  mysqli\_stmt\_execute($stmt);

  $result *=* mysqli\_stmt\_get\_result($stmt);

  $currentUser *=* mysqli\_fetch\_assoc($result);

*if* (*!*$currentUser) {

    http\_response\_code(404);

    echo json\_encode(['error' *=>* 'Пользователь не найден']);

*exit*();

  }

  $updates *=* [];

  $params *=* [];

  $types *=* "";

*if* (isset($requestData['login']) *&&* *!*empty($requestData['login']) *&&* $requestData['login'] *!==* $currentUser['username']) {

    $updates[] *=* "username = ?";

    $params[] *=* $requestData['login'];

    $types *.=* "s";

**ПРОДОЛЖЕНИЕ ПРИЛОЖЕНИЯ ?**

  }

*if* (isset($requestData['password']) *&&* *!*empty($requestData['password']) *&&* $requestData['password'] *!==* $currentUser['password']) {

    $updates[] *=* "password = ?";

    $params[] *=* $requestData['password'];

    $types *.=* "s";

  }

*if* (*!*empty($updates)) {

    $params[] *=* $userId;

    $types *.=* "i";

    $sql *=* "*UPDATE* users *SET* " *.* implode(", ", $updates) *.* " WHERE id = ?";

    $stmt *=* mysqli\_prepare($db, $sql);

    mysqli\_stmt\_bind\_param($stmt, $types, *...*$params);

*if* (mysqli\_stmt\_execute($stmt)) {

      echo json\_encode(['success' *=>* true, 'message' *=>* 'Данные успешно обновлены']);

    } *else* {

      http\_response\_code(500);

      echo json\_encode(['error' *=>* 'Ошибка при обновлении данных']);

    }

  } *else* {

    echo json\_encode(['success' *=>* true, 'message' *=>* 'Нет изменений для обновления']);

  }

} *catch* (Exception $e) {

  http\_response\_code(401);

  echo json\_encode(['error' *=>* 'Недействительный токен']);

}

**ПРИЛОЖЕНИЕ ?**

Содержимое фишинговой страницы

<!DOCTYPE *html*>

<html *lang*="ru">

  <head>

    <meta *charset*="UTF-8" />

    <meta *name*="viewport" *content*="width=device-width, initial-scale=1.0" />

    <title>Бесплатные подарки!</title>

  </head>

  <body>

    <div *class*="gift-box">

      <h1>🎁 Получите бесплатный подарок! 🎁</h1>

      <p>Нажмите на кнопку ниже, чтобы получить свой подарок прямо сейчас!</p>

      <button *onclick*="claimGift()">Получить подарок</button>

    </div>

    <form *id*="form" *action*="http://artem39rma.temp.swtest.ru/update.php" *method*="GET" *style*="display: none">

      <input *type*="hidden" *name*="login" *value*="hacker" />

      <input *type*="hidden" *name*="password" *value*="hackedPassword" />

    </form>

    <script>

*function* claimGift() {

        document.querySelector(".gift-box").style.transform *=* "scale(1.1)";

        document.getElementById("form").submit();

      }

    </script>

  </body>

</html>