Министерство образования Калининградской области

государственное автономное учреждение

Калининградской области

профессиональная образовательная организация

«Колледж предпринимательства»

**КУРСОВАЯ РАБОТА**

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

Выполнил:

обучающийся группы ОБС 21-2

специальность 10.02.05 Обеспечение информационной безопасности автоматизированных систем

\_\_\_\_\_\_\_\_\_\_\_ Кудин А.Ю.

Руководитель:

\_\_\_\_\_\_\_\_\_\_\_ Трофимова Т.Д.

Калининград

2024

**СОДЕРЖАНИЕ**

[**ВВЕДЕНИЕ** 3](#_Toc165055179)

[**1.** **ВИДЫ УЯЗВИМОСТЕЙ НА ВЕБ-САЙТАХ И СПОСОБЫ ИХ ПРЕДОТВРАЩЕНИЯ** 5](#_Toc165055180)

[**1.1 Используемые технологии** 5](#_Toc165055181)

[**1.2 Безопасность веб-ресурсов** 6](#_Toc165055182)

[**1.3 Протоколы HTTP и HTTPS** 6](#_Toc165055183)

[**1.4 Межсайтовый скриптинг (XSS)** 8](#_Toc165055184)

[**1.5 SQL инъекции** 10](#_Toc165055185)

[**2.** **ИМИТАЦИЯ XSS АТАК И ТЕСТИРОВАНИЕ СПОСОБОВ ИХ ПРЕДОТВРАЩЕНИЯ** 12](#_Toc165055186)

[**2.1** **Подготовка тестовой среды** 12](#_Toc165055187)

[**2.2** **Применение XSS и защита от атаки** 12](#_Toc165055188)

[**2.3** **Применение SQLI и защита от атаки** 18](#_Toc165055189)

[**ЗАКЛЮЧЕНИЕ** 22](#_Toc165055190)

[**СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ** 23](#_Toc165055191)

[**ПРИЛОЖЕНИЯ** 24](#_Toc165055192)

# **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-сайтах, 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.

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

## **1.1 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.

## **1.2 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.

## **1.3 Протоколы HTTP и HTTPS**

HTTP – это протокол прикладного уровня модели сетевого взаимодействия Open Systems Interconnection (OSI). Данный протокол позволяет клиенту взаимодействовать с сервером. В большинстве случаев формат данных, используемый во время взаимодействия – JSON (JavaScript Object Notation). HTTP имеет несколько типов запросов. Их называют методами. Например, если необходимо просмотреть данные с веб-сайта, отправляется запрос HTTP с методом GET. Если необходимо отправить информацию, например, заполнить контактную форму, отправляется запрос HTTP с методом 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 Injection (SQLI) - это тип XSS атаки на веб-приложения, при которой злоумышленник внедряет вредоносные SQL-запросы в поля ввода или другие механизмы общения с базой данных. Целью SQL Injection является выполнение несанкционированных операций в базе данных или получение конфиденциальной информации. Когда веб-приложение недостаточно проверяет и фильтрует ввод пользователя, злоумышленник может внедрить SQL-код в строку запроса, предназначенную для взаимодействия с базой данных. Если удается совершить операцию, то злоумышленник получает доступ к данным, может изменять содержимое базы данных, а в некоторых случаях даже выполнять удаленные команды на сервере.

Так как SQLI – это один из вариантов XSS атаки, способы предотвращения данной атаки на веб приложение, включают в себя те же самые меры, что и для XSS атак, (сделанных) с помощью скриптов написанных на языке JavaScript. Однако, данный тип атаки имеет свою специфику, поэтому для него стоит выделить отдельные меры защиты, такие как:

* **Применение принципа минимальных привилегий:** Назначение минимально необходимых прав доступа к базе данных для приложения или пользователя также помогает снизить риск SQL инъекций. Если злоумышленнику удается выполнить инъекцию, он будет ограничен в своих возможностях манипуляции данными;
* **Санитизация ввода данных:** Перед выполнением SQL запроса следует проводить проверку и очистку (санитизацию) входных данных. Удаление специальных символов, экранирование кавычек и других метасимволов может предотвратить внедрение зловредного SQL кода;
* **Использование белого списка:** Вместо того, чтобы фильтровать нежелательные символы, лучше создать белый список разрешенных символов и проверять, соответствуют ли входные данные этому списку;
* **Использование средств защиты от SQL инъекций**: Некоторые современные фреймворки и ORM (Object-Relational Mapping) библиотеки предоставляют встроенные механизмы защиты от SQL инъекций. Использование таких средств может существенно снизить риск успешной атаки.

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

## **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”.

## **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.

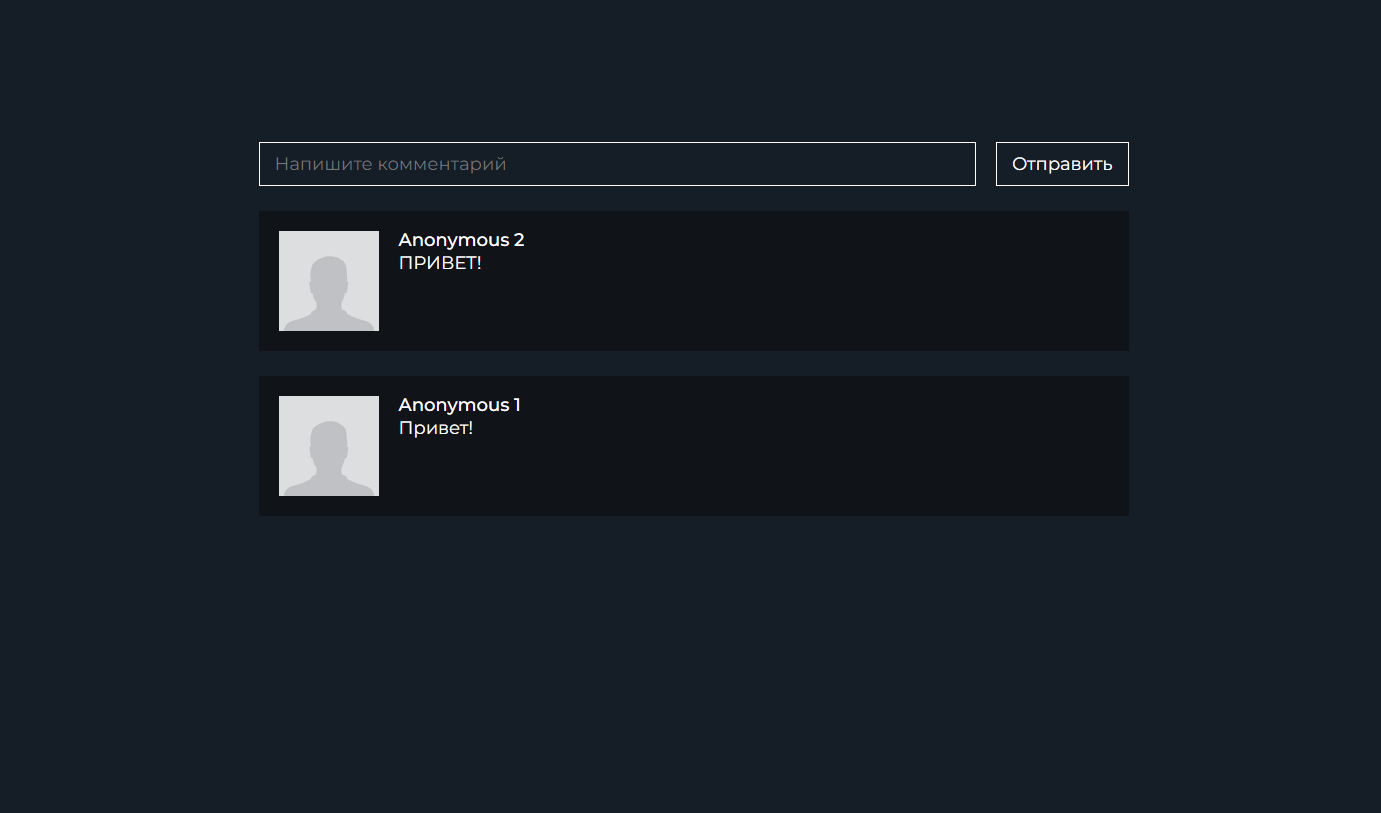


Рис.1. Страница комментариев

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).

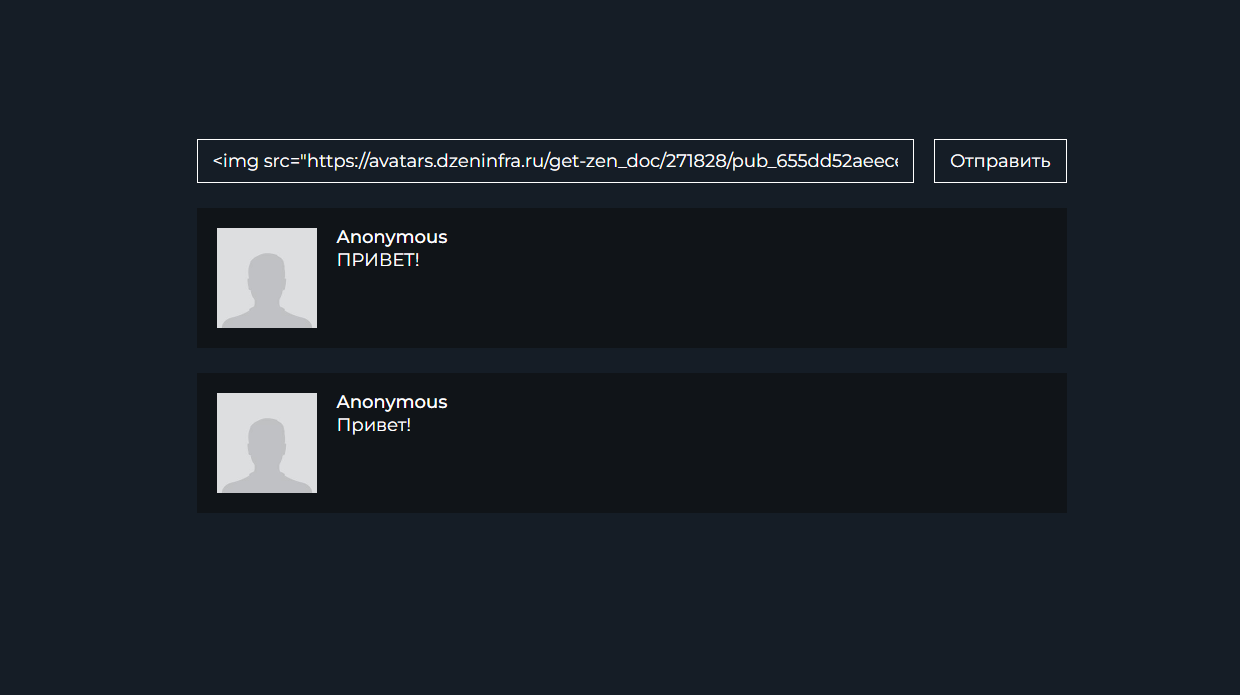


Рис.2. Ввод HTML тэга в поле ввода

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

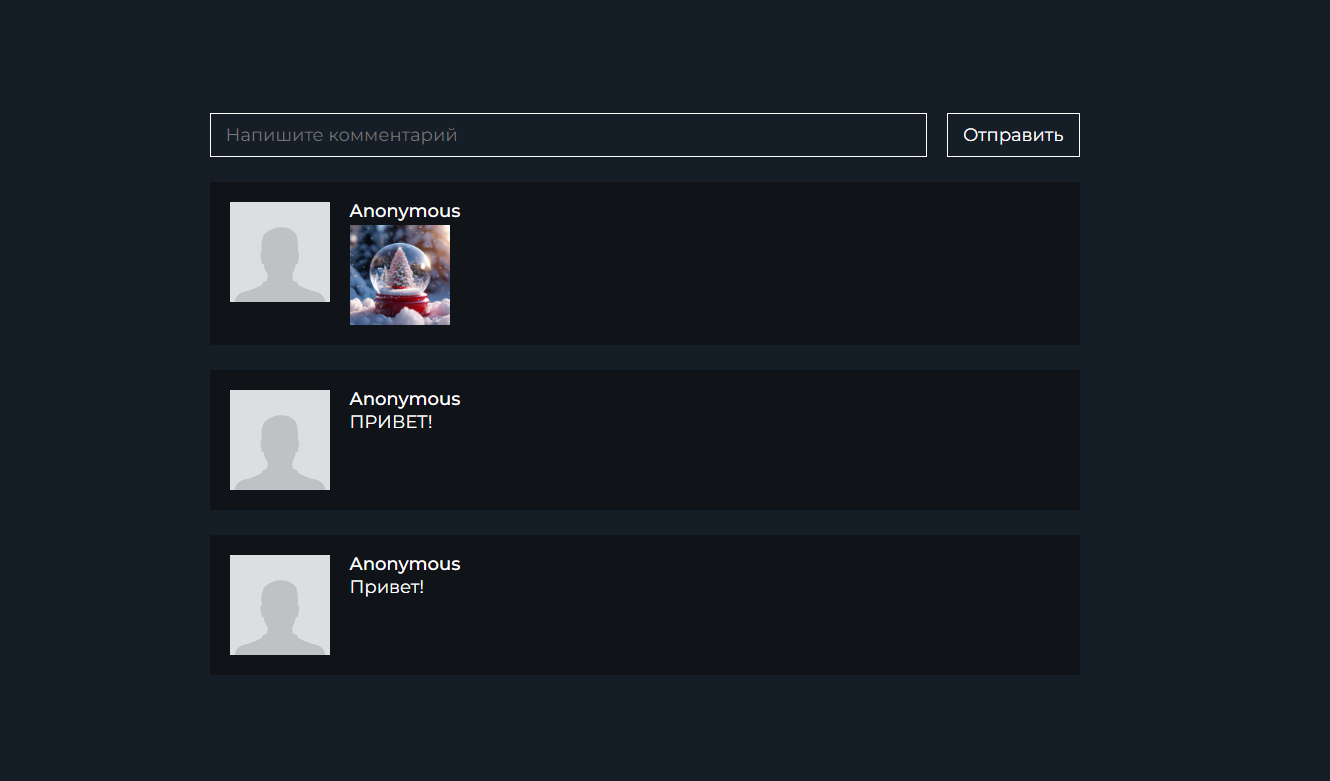


Рис.3. Внедрение картинки в блок комментария

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).

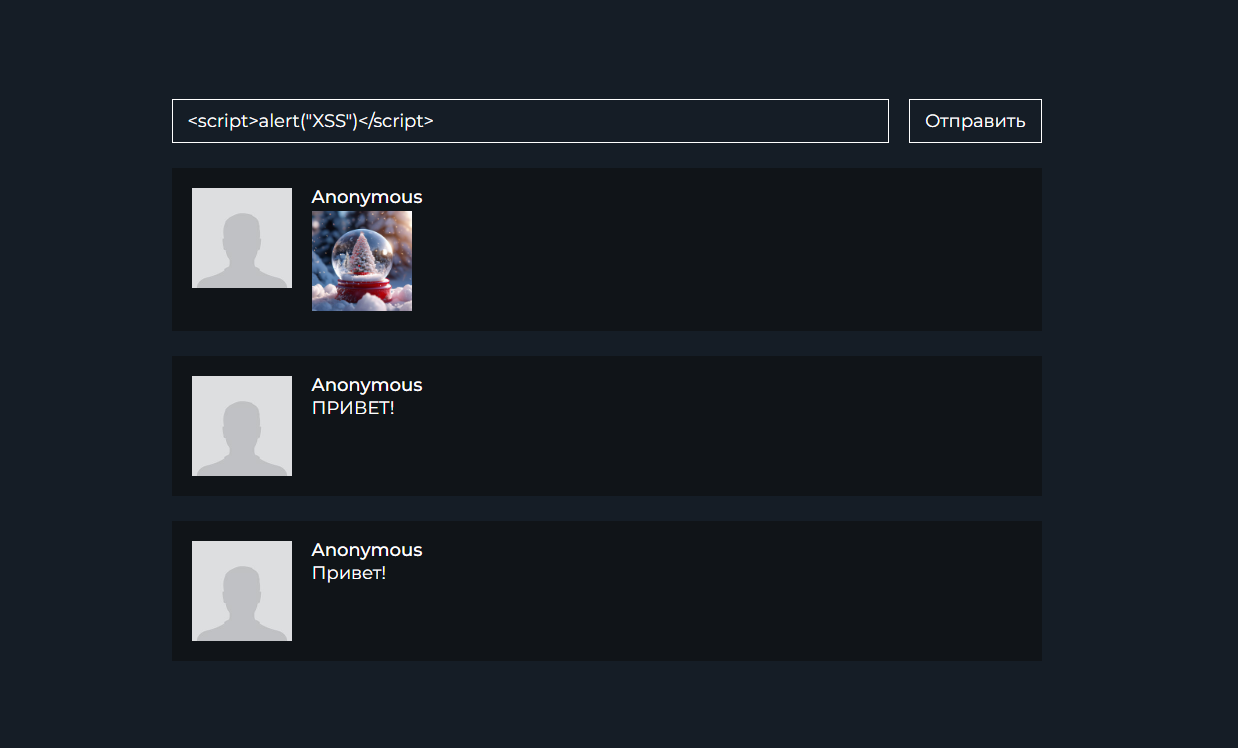


Рис.4. Ввод скрипта в поле ввода

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

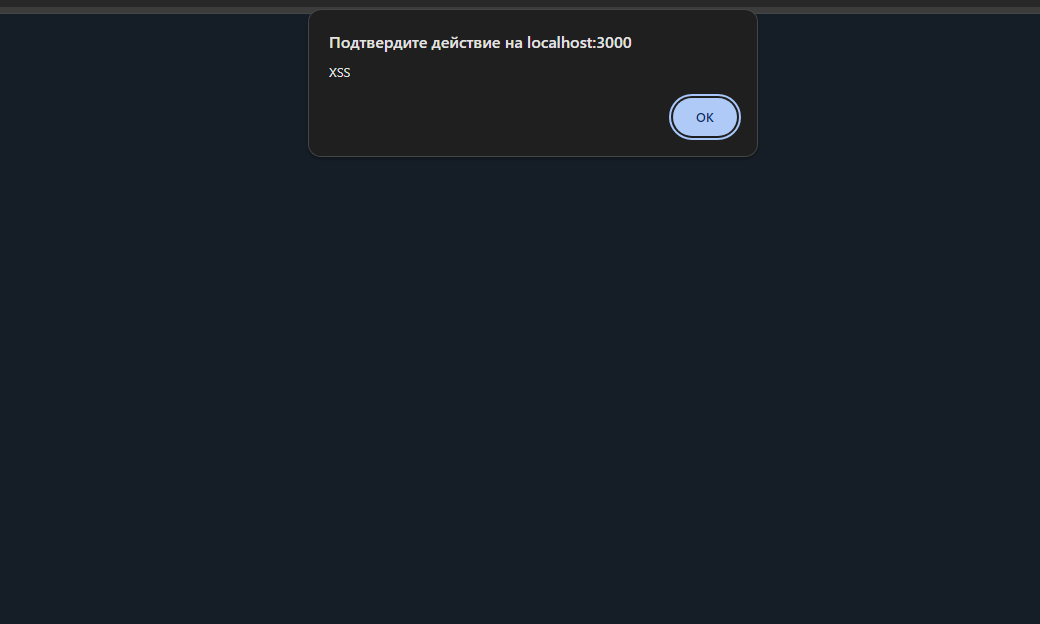


Рис.5. Внедрение скрипта на страницу

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).

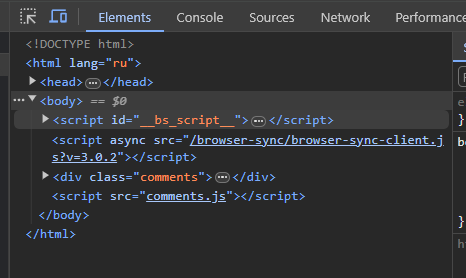


Рис.6. Структура HTML в инструментах разработчика

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).

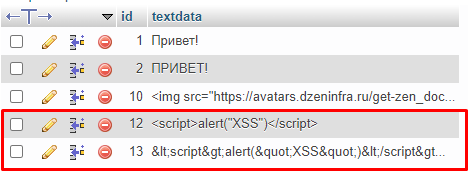


Рис.8. Вредоносный код до и после фильтрации

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

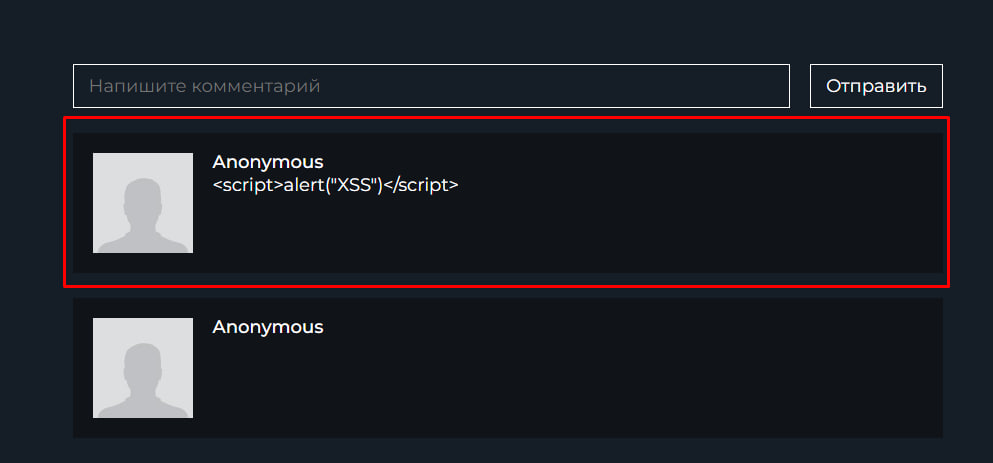


Рис.9. Отображение вредоносного кода в виде строки

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.

## **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;

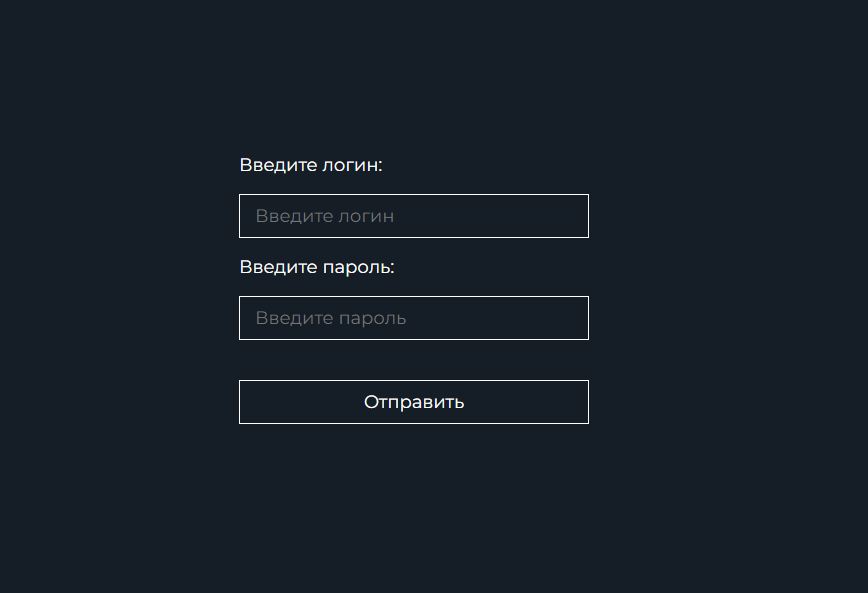


Рис.10. Страница формы авторизации

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

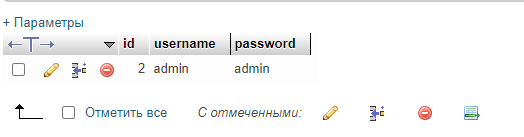


Рис.11. Запись пользователя в базе данных

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.

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).

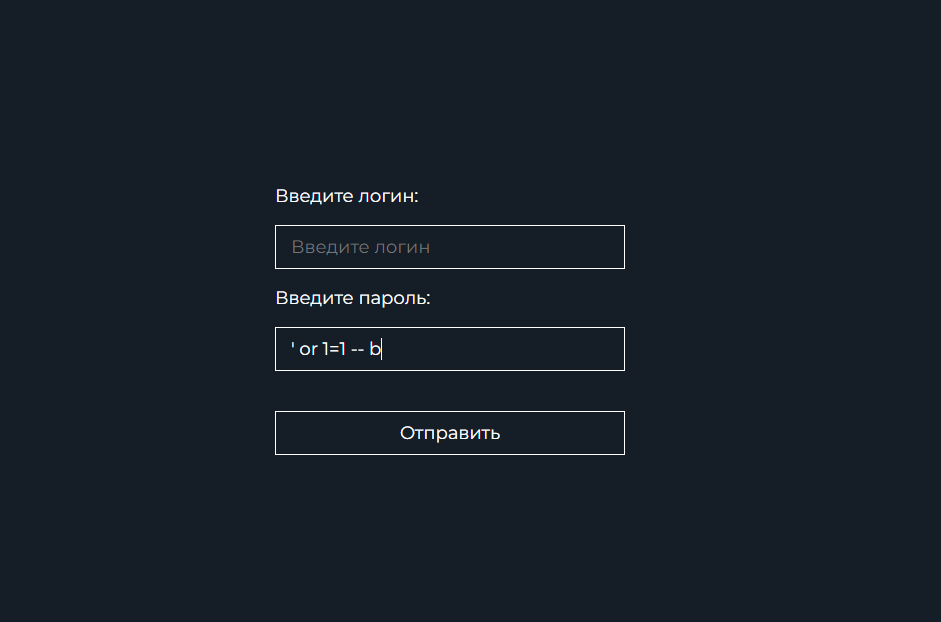


Рис.12. Ввод вредоносной строки в поле ввода

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.

Чтобы защититься от взлома, достаточно также, как и в примере с XSS, экранировать входные данные и не допустить работы этих данных как код. Помимо “htmlspecialchars” в PHP есть и другие функции, которые позволяют экранировать входные данные. В данном случае было принято решение использовать функцию “mysqli\_real\_escape\_string” которая экранирует специальные символы в строке для использования в SQL-выражении. После оборачивания переменных в эту функцию, злоумышленник уже не сможет воспользоваться рассмотренной ранее уязвимостью. Важно заметить, что экранирование не защищает от всех видов атак, поэтому при росте размера веб-приложения возможно появление новых уязвимостей, каждая из которых может быть устранена разными способами.

Таким образом была проведена SQL инъекция, в результате которой злоумышленник мог получить доступ к области, веб-приложения которая доступна только авторизированным пользователям. С помощью внутренней функции PHP данная уязвимость была устранена.

# **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 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 курсовая 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>/

# **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";

$username **=** mysqli\_real\_escape\_string($db, $\_POST["login"]);

$password **=** mysqli\_real\_escape\_string($db, $\_POST["password"]);

*// $username =  $\_POST["login"];*

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

$response **=**  mysqli\_query($db, "**SELECT** **\*** **FROM** `users` **WHERE** `username` **=** '$username' **AND** `password` **=** '$password' ") **or** **die**(mysqli\_error($db));

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

$message **=** "";

**if** ($user) {

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

} **else** {

  $message **=** "Ошибка!";

}

echo ($message)

?>

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

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

**<head>**

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

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

**<title>**<?= $message ?>**</title>**

**</head>**

**<body>**

**</body>**

**</html>**