

Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Кузбасский государственный технический университет
имени Т. Ф. Горбачева»

Кафедра информатики и информационных систем

Составитель В. В. Романова

БЕЗОПАСНОСТЬ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

Методические материалы

Рекомендовано цикловой методической комиссией
Информационных систем и программирования
в качестве электронного издания
для использования в образовательном процессе

Кемерово 2026

Рецензент:

С. А. Асанов, старший преподаватель кафедры информационных и автоматизированных производственных систем

Романова Вера Васильевна

Безопасность программного обеспечения : методические материалы для обучающихся специальности СПО 09.02.11 Разработка и управление программным обеспечением / Кузбасский государственный технический университет имени Т. Ф. Горбачева, кафедра информатики и информационных систем; сост. В. В. Романова. – Кемерово, 2026. – 1 файл (839 КБ). – Текст : электронный.

Методические материалы по дисциплине «Безопасность программного обеспечения» для обучающихся специальности СПО 09.02.11 Разработка и управление программным обеспечением содержат указания для проведения лабораторных работ и самостоятельных работ.

© Кузбасский государственный технический университет имени Т. Ф. Горбачева, 2026

© Романова В. В., составление, 2026

Лабораторная работа №1

Анализ кода на наличие уязвимостей

Цель работы: Получение навыков статического анализа программного кода для выявления уязвимостей безопасности информационной системы.

Теоретические положения

Статический анализ – это процесс автоматизированной проверки исходного кода программы без его фактического выполнения. Цель анализа – выявление потенциальных уязвимостей, ошибок программирования и небезопасных конструкций.

Методы анализа кода на уязвимости.

1. Статический анализ (SAST).

SAST-анализ кода без его запуска (через ручной ревью и/или инструменты):

- поиск опасных функций (eval, exec, system, os.exec, Runtime.exec, плохие SQL-конкатенации);
- проверка на использование слабых хеш-функций (MD5, SHA-1 без salt и т. п.);
- проверка валидации входных данных и путей файлов.

2. Ручной code-review по методологии STRIDE.

При просмотре кода можно использовать модель STRIDE как «чек-лист»:

- проверить, где возможны Spoofing (слабая аутентификация);
- Tampering (не валидированный ввод, SQL-инъекции);
- Information disclosure (вывод внутренних ошибок, трассировка стека, дампы переменных);
- DoS (бесконечные циклы, тяжелые операции без лимитов);
- Elevation of privilege (слабые проверки ролей, прямая смена role=admin в JSON или URL).

3. Использование OWASP Top 10 и CWE.

При анализе удобно ориентироваться на:

- OWASP Top 10: уязвимые компоненты, инъекции, аутентификация/сессии, XSS, неправильная конфигурация.
- CWE-списки (например, CWE-79, CWE-89, CWE-918) для более точной классификации найденных фрагментов кода.

Обзор средств статического анализа.

Таблица 1.1 – Иностранные средства

Название	Языки	Особенности
Its4	C/C++	Выявляет опасные функции (strcpy/memcpy), даёт советы по улучшению
RATS	C/C++, PHP, Perl, Python	Комбинирует проверки безопасности с семантическим анализом
Flawfinder	C/C++	Присваивает коэффициенты риска, сортирует по степени опасности
Flexelint (PC-lint)	C/C++	Анализ потоков данных, выявление нулевых указателей, переполнений
Coverity	C/C++, Java	Обработка миллионов строк кода, 100% покрытие трассы

Таблица 1.2 – Российские средства

Название	Языки	Особенности
АК-ВС	C/C++, Java, Pascal, C#, PHP, Assembler	Статический и динамический анализ, базы сигнатур
Аист-С	C/C++	Статический анализ исходных текстов
Viva64	C/C++	Выявление проблем при переходе с 32-bit на 64-bit системы

Примеры уязвимостей в коде (с фрагментами).

1. SQL-инъекция.

js

// Node.js + raw SQL (уязвимо)

```
const query = `SELECT * FROM users WHERE login =
'${req.body.login}`;
```

```
db.query(query, (err, result) => { /* ... */ });
```

Злоумышленник может передать login = ' OR '1'='1' -- и получить всех пользователей.

Рекомендуемый паттерн: использовать параметризованные запросы или ORM.

js

```
const query = "SELECT * FROM users WHERE login = ?";
```

```
db.query(query, [login], (err, result) => { /* ... */ });
```

2. XSS (ввод пользовательских данных).

```
js
```

```
// Уязвимо: вставка в шаблон без эскейпинга.
```

```
res.send(`<div>${userComment}</div>`);
```

Если userComment = "<script>alert('xss')</script>", то в браузере выполнится JS.

Решение:

- всегда экранировать HTML-вывод;
- использовать шаблонизаторы с автоматическим эскейпингом (Pug, Handlebars, JSX и т.п.).

3. Небезопасная обработка исключений.

```
js
```

```
try {
```

```
    const user = await db.getUser(req.params.id);
```

```
    res.json(user);
```

```
} catch (err) {
```

```
    res.status(500).send(err.stack); // Опасно – вывод трассировки
```

В стеке могут быть пути к файлам, конфиги, части кода.

Детальное описание критических уязвимостей.

SQL-инъекция в login.php.

Код:

```
php
```

```
$sql = "SELECT * FROM users WHERE username = '$username'
```

```
AND password = '$password'";
```

Атака:

```
sql
```

```
username: admin' OR '1'='1' --
```

```
password: anything
```

Последствия: Получение доступа без пароля

Исправление:

```
php
```

```
$stmt = $pdo->prepare("SELECT * FROM users WHERE  
username = ? AND password = ?");
```

```
$stmt->execute([$username, $hashed_password]);
```

Хранение паролей в открытом виде

Код:

```
php
```

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

Проблема: Пароль передаётся и хранится в открытом виде

Исправление:

```
php
$hashed_password = password_hash($password, PASSWORD_BCRYPT);
```

// При проверке:

```
if (password_verify($password, $hash_from_db)) { ... }
```

SQL-инъекция в dashboard.php

Код:

```
php
$sql = "SELECT * FROM users WHERE id = $user_id";
```

Исправление:

```
php
$stmt = $pdo->prepare("SELECT * FROM users WHERE id = ?");
$stmt->execute([$user_id]);
```

Задания к лабораторной работе №1

1. Изучить средства статического анализа кода (RATS, Flawfinder, Coverity).
2. Разработать (взять небольшой фрагмент веб-приложения (например, Express-сервер с регистрацией и авторизацией, или пример из учебных материалов) с намеренными уязвимостями.
3. Выявить 10-15 уязвимостей различных типов.
4. Классифицировать уязвимости по уровням риска.
5. Предложить исправления для каждой уязвимости.

Лабораторная работа №2

SQL-инъекции – эксплуатация и защита уязвимого приложения. XSS атаки – создание и предотвращение межсайтового скриптинга

Цель работы: Изучить принципы эксплуатации SQL-инъекций и XSS-атак на веб-приложениях, освоить методы защиты от данных уязвимостей и получить практические навыки их выявления и устранения.

Теоретические положения

SQL-инъекция – это атака, при которой злоумышленник внедряет вредоносный SQL-код во входные данные и изменяет логику запроса к базе данных. Такая уязвимость возникает, когда запрос формируется через конкатенацию строк вместо параметризованных запросов. SQL-инъекции возникают, когда приложение использует **непроверенные данные от пользователя** для формирования SQL-запроса. Вместо того чтобы обрабатывать ввод как простые данные, злоумышленник может заставить интерпретатор SQL выполнить вредоносный код.

Внедрение SQL-кода (англ. SQL injection) – один из распространённых способов взлома сайтов и программ, работающих с базами данных, основанный на внедрении в запрос произвольного SQL-кода. Внедрение SQL, в зависимости от типа используемой СУБД и условий внедрения, может дать возможность атакующему выполнить произвольный запрос к базе данных (например, прочитать содержимое любых таблиц, удалить, изменить или добавить данные), получить возможность чтения и/или записи локальных файлов и выполнения произвольных команд на атакуемом сервере.

Атака типа внедрения SQL может быть возможна из-за некорректной обработки входных данных, используемых в SQL-запросах.

Разработчик прикладных программ, работающих с базами данных, должен знать о таких уязвимостях и принимать меры противодействия внедрению SQL.

XSS (Cross-Site Scripting) – это внедрение вредоносного скрипта на страницу, который выполняется в браузере пользователя. XSS опасна тем, что позволяет красть cookies, подменять контент страницы и выполнять действия от имени пользователя. Является раз-

новидностью атаки «внедрение кода».

Специфика подобных атак заключается в том, что вредоносный код может использовать авторизацию пользователя в веб-системе для получения к ней расширенного доступа или для получения авторизационных данных пользователя. Вредоносный код может быть вставлен в страницу как через уязвимость в веб-сервере, так и через уязвимость на компьютере пользователя.

Для термина используют сокращение «XSS», чтобы не было путаницы с каскадными таблицами стилей, использующими сокращение «CSS».

XSS находится на третьем месте в рейтинге ключевых рисков Web-приложений согласно OWASP. Долгое время программисты не уделяли им должного внимания, считая их неопасными. Однако это мнение ошибочно: на странице или в HTTP-Cookie могут быть весьма уязвимые данные (например, идентификатор сессии администратора или номера платёжных документов), а там, где нет защиты от CSRF, атакующий может выполнить любые действия, доступные пользователю. Межсайтовый скриптинг может быть использован для проведения DoS-атаки. Основные меры защиты: параметризованные запросы для SQL, экранирование вывода, CSP-заголовки, фильтрация ввода и использование безопасных функций фреймворка.

Задания к лабораторной работе №2

Задание 1 – SQL Injection.

Платформа: portswigger.net/web-security/sql-injection. PortSwigger Web Academy содержит интерактивные лаборатории по SQL Injection разного уровня сложности – от базового обхода авторизации до UNION-атак.

Регистрация (если ещё не зарегистрированы):

Откройте <https://portswigger.net/web-security> и создайте бесплатный аккаунт.

Перейдите: Academy и ниже SQL injection.

Лаборатория А – Обход авторизации (Login bypass):

Откройте лабораторию: «SQL injection vulnerability in WHERE clause allowing retrieval of hidden data»

Изучите фильтр категорий товаров. Модифицируйте URL-параметр category, добавив payload: ' OR 1=1--

Убедитесь, что отображаются скрытые товары. Сделайте скриншот.

Лаборатория В – UNION Attack (извлечение данных):

Откройте лабораторию: «SQL injection UNION attack, retrieving data from other tables»

Определите количество столбцов запроса через ORDER BY:

' ORDER BY 1--' ORDER BY 2--

Увеличивайте до получения ошибки. Запишите максимальное рабочее значение.

Извлеките данные пользователей из таблицы users через UNION SELECT:

' UNION SELECT username, password FROM users--

Войдите как пользователь administrator с найденным паролем для завершения работы.

Задание 2 – Cross-Site Scripting/XSS

Платформа: xss-game.appspot.com. XSS Game от Google – шесть уровней нарастающей сложности, на каждом нужно выполнить произвольный JavaScript в контексте страницы.

Шаги выполнения:

Перейдите на <https://xss-game.appspot.com> и начните с уровня 1.

Введите в поле поиска базовый XSS-payload:
<script>alert('XSS')</script>

Если уровень блокирует тег <script>, попробуйте обходные варианты:

<svg onload=alert(1)>

Пройдите уровни 1, 2 и 3 (обязательно). Уровни 4-6 по желанию.

Задание 3 – Аутентификация и IDOR

Платформа: portswigger.net/web-security.

Лаборатория А – Insecure Direct Object Reference (IDOR):

Откройте: «Insecure direct object references»

Войдите под учётными данными: wiener/peter

Найдите функцию чата и скачайте лог переписки. Изучите URL при скачивании файла.

Измените номер файла в URL, чтобы получить чужой файл (transcript номер 1).

Найдите пароль пользователя carlos в файле и войдите от его имени.

Лаборатория В – Username Enumeration:

Откройте: «Username enumeration via different responses»

Перейдите на страницу авторизации. Отправьте несколько запросов с разными логинами

Лабораторная работа №3

Составление модели угроз для типового веб-приложения

Цель работы: Разработать модель угроз для типового веб-приложения, выявить основные векторы атак и предложить базовые меры защиты.

Теоретические положения

Модель угроз – это систематизированный перечень потенциальных угроз информационной безопасности, ранжированных по степени актуальности для конкретной информационной системы с учётом возможных нарушителей, их мотивов, квалификации и имеющихся уязвимостей.

1. Краткое описание типового веб-приложения.

Условно рассмотрим приложение с архитектурой:

- клиент (браузер, мобильный клиент);
- веб-сервер (Nginx/Apache, Node.js, Python, Java и т. п.);
- бэкенд-приложение (логика бизнес-процессов);
- база данных (PostgreSQL/MySQL/MongoDB);
- внешние API.

Таблица 3.1 – Примеры угроз для типового веб-приложения

Компонент / канал	Категория STRIDE	Угроза (сценарий)	Краткое описание последствий
Форма входа (логин/пароль)	Spoofing, Elevation of privilege	Подбор логина/пароля (брут-форс, утечка паролей)	Доступ злоумышленника к аккаунту пользователя.
Форма входа (логин/пароль)	Information disclosure	Хранение паролей в открытом виде или слабая хеш-функция	Массовая утечка учетных данных при взломе базы данных.
Форма входа (логин/пароль)	Elevation of privilege	Уязвимость, позволяющая менять роль пользователя (например, role=admin)	Обычный пользователь становится администратором.
Браузер (клиент)	Information disclosure	XSS-атака: выполнение вредоносного JS-кода на странице	Похищение сессионных кук, данных пользователя, кражей токенов.
Браузер (клиент)	Tampering	Подмена данных в форме перед отправ-	Оформление заказа с неверной суммой.

Компонент / канал	Категория STRIDE	Угроза (сценарий)	Краткое описание последствий
		кой (например, изменение цены заказа)	
HTTP-трафик	Information disclosure	Передача данных без HTTPS, перехват данных на промежуточном узле	Перехват логинов, паролей, токенов.
База данных	Tampering, Information disclosure	SQL-инъекция, прямой доступ к БД	Чтение/изменение/удаление данных, утечка всей базы.
Сервер приложения	Denial of service	Перегрузка сервера большими запросами (DoS/DDoS)	Недоступность сервиса для легитимных пользователей.
API-интерфейсы	Tampering, Elevation of privilege	Подмена JWT-токена, отсутствие валидации прав доступа	Доступ к данным/функциям без соответствующих прав.
Админка приложения	Spoofing, Elevation of privilege	Утечка логина/пароля администратора, слабая аутентификация	Полный контроль над системой, удаление данных.

Источники угроз (модель нарушителя).

ФСТЭК рекомендует выделять типы нарушителей и оценивать их возможности (базовый/средний/высокий уровень).

Таблица 3.2 – Пример нарушителей для веб-приложения

Тип нарушителя	Возможности (примерно по ФСТЭК)	Возможные действия
Внешний хакер (сети общего пользования)	Интернет-доступ, общедоступные утилиты, сценарии атак из открытых источников.	DoS, XSS, SQL-инъекции, брутфорс логинов, эксплуатация известных уязвимостей.
Инсайдер (сотрудник)	Часть доступа к внутренним сетям, знание внутренней архитектуры, учетные данные.	Копирование данных, саботаж, доступ к логам/базам, использование служебных аккаунтов.
Внешний партнер (API-клиент)	Доступ к API-интерфейсам по договору, ключи/токены.	Злоупотребление правами, перебор сессий, использование уязвимостей API-безопасности.

Отбор угроз из Банка данных угроз ФСТЭК (БДУ).

Для типового веб-приложения из БДУ ФСТЭК обычно выбирают угрозы, относящиеся к:

- несанкционированный доступ к информации через веб-приложения (например, подбор учетных данных, утечки сессий);
- несанкционированное изменение данных (SQL-инъекции, подмена входных параметров);
- несанкционированное воздействие на доступность (DDoS, перегрузка сервера);
- автоматическое распространение вредоносного кода (через веб-страницы, уязвимые библиотеки);
- несанкционированный доступ к конфигурационным данным и ключам (через утечки, уязвимости CI/CD, неправильные права доступа).

В учебной работе можно перечислить несколько конкретных категорий угроз (без привязки к точным кодам из БДУ) – это уже считается адекватной «моделью по ФСТЭК».

Пример таблицы угроз по ФСТЭК-подходу

Ниже приведен пример структуры таблицы, которая подходит под требование ФСТЭК-модели (объект воздействия, тип нарушителя, сценарий, последствия, возможные меры защиты).

Таблица 3.3 – Таблица угроз по ФСТЭК

№	Угроза (сокращённое описание)	Объект воздействия	Тип нарушителя	Сценарий реализации	Кратко о последствиях	Возможные меры защиты
1	Подбор/утечка учетных данных пользователей	Персональные данные, логины/пароли	Внешний хакер	Брут-форс логинов, использование утечек из других сервисов, перехват кук	Массовый доступ к аккаунтам, кража данных, использование для фишинга	Сложные пароли, MFA, капча/Limit – по запросам, хеширование с salt, хранение паролей только в хеше
2	SQL-инъекция в веб-приложении	База данных, логика приложения	Внешний хакер	Ввод в форму или URL вредоносного SQL-кода	Чтение / изменение / удаление данных, кража ПДн, получение прав адми-	Подготовленные выражения (Prepared Statements / ORM), валидация и эскейпинг

№	Угроза (сокращённое описание)	Объект воздействия	Тип нарушителя	Сценарий реализации	Кратко о последствиях	Возможные меры защиты
					нистратора	входных данных, WAF
3	XSS-атака через веб-интерфейс	Клиент (браузер пользователя), сессионные куки	Внешний хакер	Подгрузка вредоносного JS-скрипта на страницу	Похищение сессии, кража токенов, совершение действий от имени пользователя	Эскейпинг выводимых данных, CSP-заголовки, проверка загружаемого контента, использование httpsOnly – кук
4	Неавторизованный доступ к API	REST-API, JWT-токены, ключи	Внешний хакер, инсайдер	Подбор/перехват токена, использование утечек ключей	Чтение/изменение данных, выполнение действий с правами другого пользователя	Валидация JWT-подписи, короткий срок жизни токенов, лимиты по запросам, IAM-политики
5	DoS/DDoS-атака на веб-сервер	Веб-сервер, бэкенд, база данных	Внешний хакер	Массовый запрос к ресурсу, перегрузка канала	Недоступность сервиса для легитимных пользователей	Rate-limiting, CDN/WAF, защита на уровне сетевого периметра, автоскейлинг
6	Доступ инсайдера к данным и конфигурациям	БД, конфиги, ключи, логи	Инсайдер	Использование служебных учетных данных, копирование данных	Утечка ПДн, саботаж, вывод данных во внешние системы	Разделение прав, минимальные привилегии, аудит действий, шифрование ключей, логирование и мониторинг

Модель угроз по ФСТЭК – это структурированный перечень угроз, объектов, нарушителей и сценариев, который выстраивается на основе Банка данных угроз (БДУ) и специфики информационной

системы. Для типового веб-приложения критичны угрозы, связанные с несанкционированным доступом к данным (утечки, SQL-инъекции, XSS), а также с недоступностью сервиса (DoS/DDoS) и злоупотреблениями со стороны инсайдеров.

Задания к лабораторной работе №3

1. Задание 1. Отбор угроз из БДУ ФСТЭК.

Ознакомьтесь с Банком данных угроз (БДУ) ФСТЭК и типичными угрозами для веб-приложений.

Выберите не менее 5–7 угроз, которые актуальны для вашего веб-приложения, например:

- несанкционированный доступ к учетным данным;
- SQL-инъекция и другие уязвимости веб-приложения;
- XSS-атаки;
- DoS/DDoS;
- утечка конфигураций и ключей;
- несанкционированный доступ инсайдера.

2. Задание 2. Формирование структуры модели угроз.

Для каждой выбранной угрозы заполните таблицу, в которой должны быть столбцы:

- код/наименование угрозы (или условное описание);
- объект воздействия;
- тип нарушителя;
- сценарий реализации;
- возможные последствия;
- предлагаемые меры защиты (кратко).

Убедитесь, что структура вашей модели соответствует рекомендованной структуре модели угроз по методике ФСТЭК от 05.02.2021.

Лабораторная работа №4

Хэширование паролей с salt и adaptive functions

Цель работы: Изучить принципы безопасного хранения паролей и на практике освоить хэширование паролей с использованием salt и adaptive functions.

Теоретические положения

Хеширование – одностороннее преобразование пароля в фиксированную строку (хеш), по которой нельзя восстановить исходный пароль. Для хранения используют именно хеш, а не открытый текст.

Назначение «соли» (salt)

Соль – случайная последовательность, добавляемая к паролю перед хешированием. Для каждого пользователя должна быть уникальная соль, чтобы одинаковые пароли давали разные хеши и чтобы атаки по таблицам (rainbow-таблицам) становились неэффективными.

Adaptive functions (bcrypt, Argon2, PBKDF2).

Adaptive функции специально заточены для хеширования паролей:

- медленные по замыслу, чтобы затруднить брут-форс;
- поддерживают параметры (итерации/rounds, память, параллельность), которые можно повышать со временем.

Примеры:

- bcrypt – де-факто стандарт для хеширования паролей;
- Argon2 – современный алгоритм, рекомендованный для защищённых приложений;
- PBKDF2 – широко поддерживается в стандартах (включая Java и .NET).

Задания к лабораторной работе №4

1. Изучить основы хэширования и отличия хэш-функций от шифрования.
2. Реализовать на Python генерацию хэшей паролей с использованием соли и адаптивной функции (bcrypt).
3. Разработать простой брутфорс-скрипт для подбора пароля по его хэшу и оценить влияние соли и фактора работы на скорость подбора.
4. Сравнить стойкость хэшей, полученных разными способами (MD5, MD5+salt, bcrypt).

Лабораторная работа №5

Анализ сетевого трафика с помощью Wireshark.

Сканирование уязвимостей OWASP ZAP и Burp Suite

Цель работы: Получить практические навыки анализа сетевого трафика с использованием Wireshark, освоить методы сканирования веб-уязвимостей с помощью инструментов OWASP ZAP и Burp Suite, научиться выявлять и классифицировать основные типы атак на веб-приложения.

Теоретические положения

Wireshark – бесплатный сетевой анализатор пакетов (sniffer), который позволяет:

- перехватывать и записывать трафик в реальном времени;
- просматривать и декодировать большой набор сетевых протоколов (Ethernet, IP, TCP, UDP, HTTP, DNS, SSL/TLS).

Чаще всего используется для:

- диагностики сетевых проблем;
- анализа производительности;
- выявления подозрительной активности (вредоносный трафик, несанкционированные соединения).

Основные шаги работы с Wireshark.

В интерфейсе Wireshark выбирается сетевой интерфейс (Wi-Fi, Ethernet и т. п.), с которого будет собираться трафик.

Захват трафика.

Нажать Start (аналог tcpdump);

Можно сразу задать фильтр захвата (например, host 192.168.1.100 или tcp port 80).

Фильтрация и анализ.

Использовать дисплей-фильтры:

- http – показать только HTTP-трафик;
- ip.addr == 192.168.1.100 – трафик с конкретного IP;
- tcp.port == 443 – HTTPS-трафик.

Анализировать:

- триплеты «источник/порт → назначение/порт»;
- размер пакетов, дельты по времени, ошибки;
- содержимое полезной нагрузки (если не зашифровано или есть ключ).

2. Основные применения в безопасности.

С помощью Wireshark можно:

- обнаружить нестандартные соединения (например, внешние хосты, куда не должен идти трафик);
- видеть открытый текст логин/паролей, токенов, если используется HTTP без TLS;
- анализировать поведение мальвары, DDoS-трафик или сканирования портов;
- отслеживать утечки данных (большие POST-запросы с чувствительной информацией).

3. Сканирование уязвимостей с помощью OWASP ZAP.

OWASP (Open Web Application Security Project) – это международная организация, которая занимается вопросами безопасности приложений. Они создают стандарты и рекомендации, помогая разработчикам и тестировщикам защитить свои приложения от распространенных уязвимостей.

OWASP Top 10 – это список наиболее критичных уязвимостей веб-приложений, который обновляется раз в несколько лет. Вот краткий обзор текущих (2021) категорий:

Broken Access Control – Нарушение контроля доступа.

Пример: Пользователь может получить доступ к ресурсам, к которым он не должен иметь доступ.

Cryptographic Failures – Ошибки шифрования.

Пример: Недостаточное шифрование данных, например, незащищенные пароли или номера кредитных карт.

Injection – Уязвимости, связанные с инъекциями (SQL, NoSQL, OS Command Injection).

Пример: SQL-инъекция, когда злоумышленник может выполнять произвольные команды в базе данных.

Insecure Design – Небезопасное проектирование.

Пример: Невключение мер безопасности на стадии проектирования приложения.

Security Misconfiguration – Ошибки конфигурации безопасности.

Пример: Использование стандартных настроек безопасности, слабые пароли администратора.

Vulnerable and Outdated Components – Уязвимые и устаревшие компоненты.

Пример: Использование библиотек с известными уязвимостями.

Identification and Authentication Failures – Ошибки идентификации и аутентификации.

Пример: Несовершенная система проверки паролей, позволяющая злоумышленникам получить доступ.

Software and Data Integrity Failures – Нарушение целостности программного обеспечения и данных.

Пример: Внедрение сторонних компонентов без проверки их безопасности.

Security Logging and Monitoring Failures – Ошибки журналирования и мониторинга безопасности.

Пример: Неспособность адекватно отслеживать подозрительную активность в системе.

Server-Side Request Forgery (SSRF) – Подделка запросов на стороне сервера.

Пример: Хакер заставляет сервер отправить запрос на внутренние ресурсы, к которым сервер имеет доступ.

OWASP ZAP (Zed Attack Proxy) – открытый инструмент для автоматического и полуавтоматического тестирования веб-приложений на уязвимости. Он работает как HTTP(S)-прокси и «прослушивает» весь трафик между браузером и веб-сервером.

3.2. Основные режимы работы.

Пассивное сканирование (Passive scan)

ZAP просто анализирует трафик, проходящий через прокси, без активных атак.

Выявляет:

отсутствие защитных заголовков (например, X-Content-Type-Options, X-Frame-Options, Content-Security-Policy);

раскрытие технической информации (версии, stack-трейсы);

подозрительные типы контента.

Активное сканирование (Active scan)

ZAP отправляет специально сформированные запросы, пытаясь вызвать проявления уязвимостей:

SQL-инъекции;

XSS (ввод `<script>` и подобных строк в параметры);

LFI/RFI, уязвимости файловых загрузок и т. п.

Результаты группируются по типу и уровню критичности

(High/Medium/Low/Info).

3.3. Пример сценария в работе.

В настройках браузера выставить прокси ZAP (localhost:8080).

Открыть тестируемое веб-приложение в браузере.

В ZAP запустить пассивное + активное сканирование.

Изучить вкладку Alerts и экспортировать отчёт по уязвимостям для анализа.

4. Сканирование уязвимостей с помощью Burp Suite.

Burp Suite – это инструмент для тестирования безопасности веб-приложений. Программа позволяет видеть, как данные передаются между браузером и сервером, и искать ошибки в этом процессе.

Примеры уязвимостей, которые помогает находить Burp Suite:

XSS (межсайтовый скриптинг) – уязвимость, используя которую злоумышленник может внедрить на веб-страницу вредоносный код. Такой код будет выполнен в браузере пользователей, что может привести к кражам данных. Например, код на JavaScript, который показывает пользователю фальшивое окно авторизации для кражи паролей.

SQL-инъекции – пользовательский ввод, который может изменить запрос к базе данных и открыть злоумышленникам доступ к информации в ней. Например, поле на сайте, куда пользователь вводит своё имя, позволяет злоумышленнику вставить команды для доступа к базе с именами, телефонами и адресами других пользователей.

Уязвимости в настройках сервера. Неправильные настройки веб-сервера или ПО могут открывать доступ к важной информации или снижать уровень защиты данных. Например, сервер может быть настроен так, что отображает список всех файлов в каталоге, а злоумышленник сможет зайти и посмотреть их содержимое.

Ошибки в управлении сессиями. Сессии пользователя часто управляются с помощью файлов cookie, которые содержат информацию об активности пользователя на сайте: какие он вводил данные, какие страницы просматривал. Если управление сессиями настроено неправильно, злоумышленники могут похитить эти данные и выдать себя за пользователя, получая доступ к его аккаунту.

Разберём на примере, как Burp Suite находит такую уязвимость, как межсайтовый скриптинг (Cross-Site Scripting, XSS). Сна-

чала настраивают браузер так, чтобы весь его трафик проходил через Burp Suite. Затем открывают нужный сайт и вводят в поиске или в другой форме, в которую пользователи вводят данные, какое-нибудь слово, например «тест». Burp Suite «ловит» этот запрос и показывает его на экране.

Теперь вместо слова «тест» подставляют специальный код, который имитирует вредоносный скрипт, и отправляют запрос на сайт. Если сайт отображает этот код, значит, злоумышленник мог бы вставить туда настоящий вирус.

Таблица 5 – Основные компоненты Burp Suite

Компонент	Функция
Repeater	Основной компонент, с которым придётся работать большую часть времени. Позволяет вручную отправлять изменённые запросы на сайт. Это полезно, когда нужно тестировать одну и ту же часть сайта с разными настройками (например, на предмет того, можно ли вставить вредоносный код)
Intruder	Инструмент для автоматического тестирования. Он отправляет много запросов с разными значениями, чтобы найти слабые места в поле ввода или настройках сайта, – например, может симитировать подбор пароля. В бесплатной версии работает медленнее, чем в Pro, поэтому не подойдёт для тестирования большого приложения вроде банковского сервиса
Proxy	«Ловит» запросы, которые браузер отправил на сайт. Например, если отправить логин и пароль, Proxy позволяет увидеть, как они передаются, и изменить их, чтобы проверить, правильно ли сайт их обрабатывает
Target	Создаёт карту сайта. Она показывает все страницы, которые посетил пользователь, и то, как они связаны друг с другом
Scanner (доступен в Burp Suite Pro)	Автоматически ищет уязвимости – например, защищён ли сайт от XSS и SQL-инъекций
Decoder	Используется для расшифровки или кодирования данных. Полезен, если нужно понять, что значит зашифрованный текст, или преобразовать его (например, выяснить, что содержат файлы cookie)
Collaborator (доступен в платной версии)	Помогает проверять уязвимости, которые могут требовать взаимодействия с внешними сервисами, например для захвата данных или проверки доступности сервера с других устройств. Collaborator создаёт специальные ссылки, чтобы видеть, как сайт реагирует на такие запросы
Comparer	Сравнивает два ответа от сервера, чтобы найти различия и понять, что изменилось после определённых действий на сайте – например, до и после входа в систему
Sequencer	Анализирует надёжность различных токенов и идентификаторов сессий на сайте. Например, можно загрузить в него список токенов, и он покажет, насколько случайно они генерируются, насколько одни символы используются чаще, чем другие. Если токен сессии легко угадать, злоумышленник сможет захватить сессию пользователя

Компонент	Функция
Logger	Записывает и хранит все запросы и ответы, которые проходят через Burp Suite. Полезно для поиска ошибок или повторного анализа
Extender	Позволяет добавлять в Burp Suite новые возможности с помощью плагинов – например, для поиска менее очевидных уязвимостей, которые не «ловятся» с помощью стандартных инструментов

У Burp Suite активное сообщество: пользователи создали базу VApp Store с расширениями, которые повышают эффективность инструмента. Например, есть уязвимости, связанные с «отравлением» веб-кеша. В этом случае пользователю показывают вредоносный контент – от поддельных страниц входа до файлов, заражённых вредоносным ПО. Пользователи Burp Suite разработали и выложили в открытый доступ расширение, которое помогает найти такую уязвимость.

Как установить.

1. Зайти на официальный сайт Burp Suite и скачать одну из версий. Burp Suite Community Edition – бесплатная версия с базовым функционалом. Professional Edition – платная версия с расширенными возможностями, например с автоматическим сканированием уязвимостей.

2. Запустить скачанный установочный файл и следовать инструкциям на экране.



Рисунок 5.1 – Установка Burp Suite

3. Открыть Burp Suite. При первом запуске программа отправит системное сообщение: создать новый проект или открыть суще-

ствующий. Нужно выбрать «Temporary Project» и нажать «Start Burp».

Burp Suite установлен и готов к использованию. Теперь можно перейти к настройке прокси, чтобы начать перехватывать трафик

Настройка прокси.

Burp Suite перехватывает трафик между браузером и сервером с помощью встроенного прокси-сервера. Настроим его, используя браузер Firefox:

1. Переходим на вкладку Proxy → Intercept. Важно убедиться, что кнопка Intercept активна (включена). Это позволяет перехватывать запросы браузера.



Рисунок 5.2 – Активная кнопка Intercept

2.

3. Меняем настройки браузера, чтобы трафик проходил через Burp Suite. Например, в Firefox в меню настроек нужно найти «Настройки сети», указать адрес 127.0.0.1 и порт 8080. Это стандартные настройки Burp Suite.

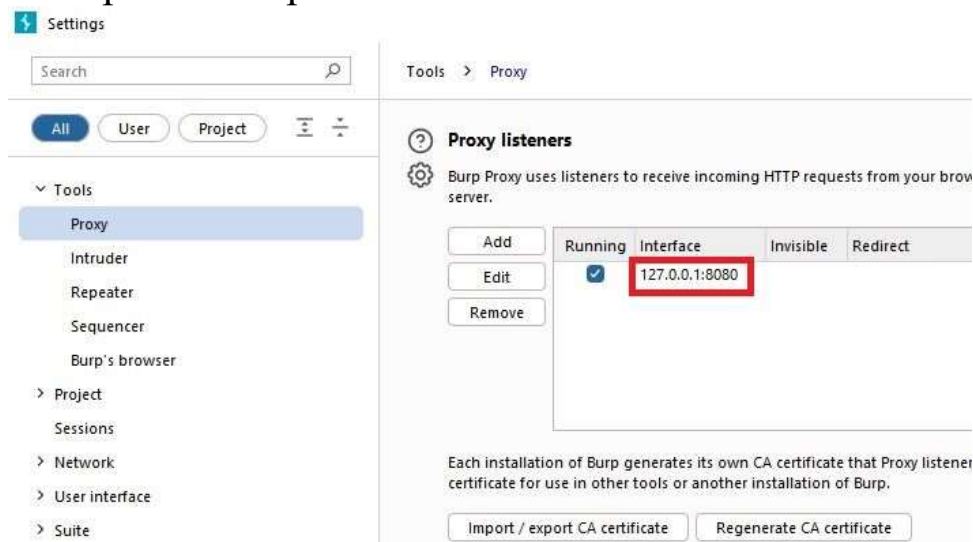


Рисунок 5.3 – Настройки сети

4. В разделе Proxy Settings есть все нужные настройки для браузера. Настройки прокси в браузере можно найти через поиск.

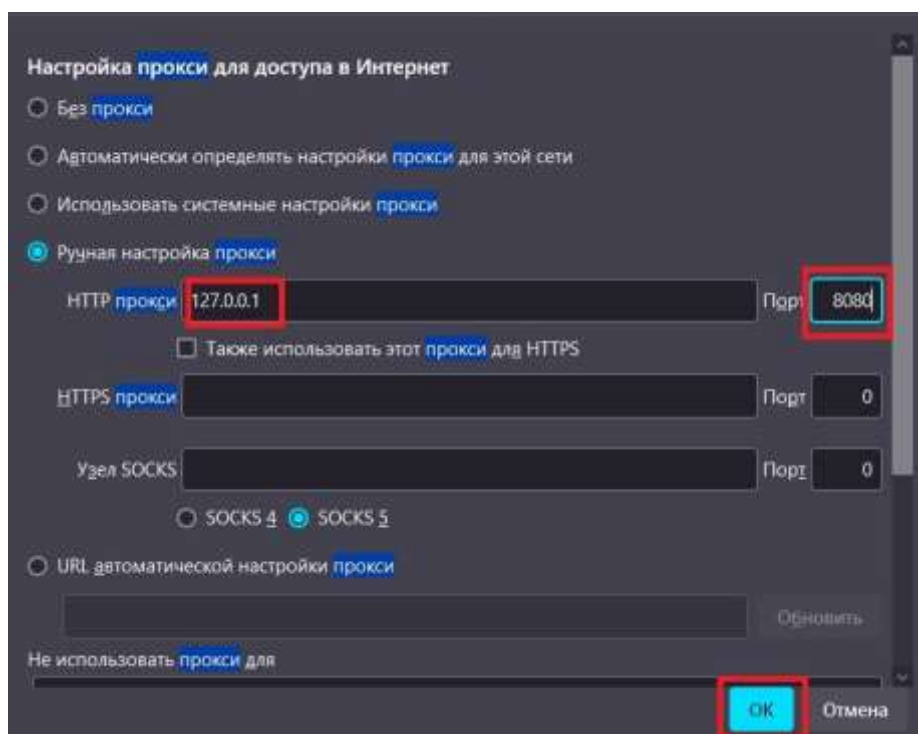


Рисунок 5.4 – Настройки прокси

5. Вводим адрес и порт, которые есть в настройках Burp Suite. Иногда программа может не работать, потому что адрес или порт заняты другим приложением. Тогда в Burp Suite можно изменить данные: например, вместо 8080 указать 8081. То же самое сделать в браузере.

6. Устанавливаем сертификат безопасности (для HTTPS) Burp Suite, чтобы просматривать зашифрованный трафик. Программа генерирует свой сертификат и сама его подписывает. Если не добавить его в браузер как доверенный, сайты, которые используют SSL-сертификаты, могут не открыться. Чтобы добавить сертификат в браузер, переходим на сайт <http://burpsuite>, скачиваем сертификат, а затем импортируем его.



Рисунок 5.5 – Сертификат CA Certificate

Чтобы скачать сертификат, нажимаем CA Certificate.

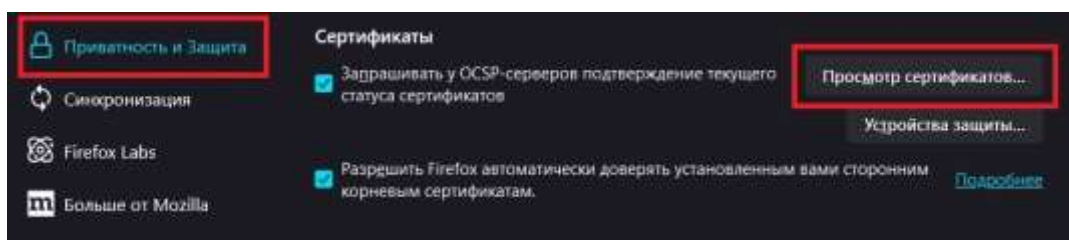


Рисунок 5.6 – Сертификаты

7. Заходим в раздел «Приватность и защита» → «Просмотр сертификатов».

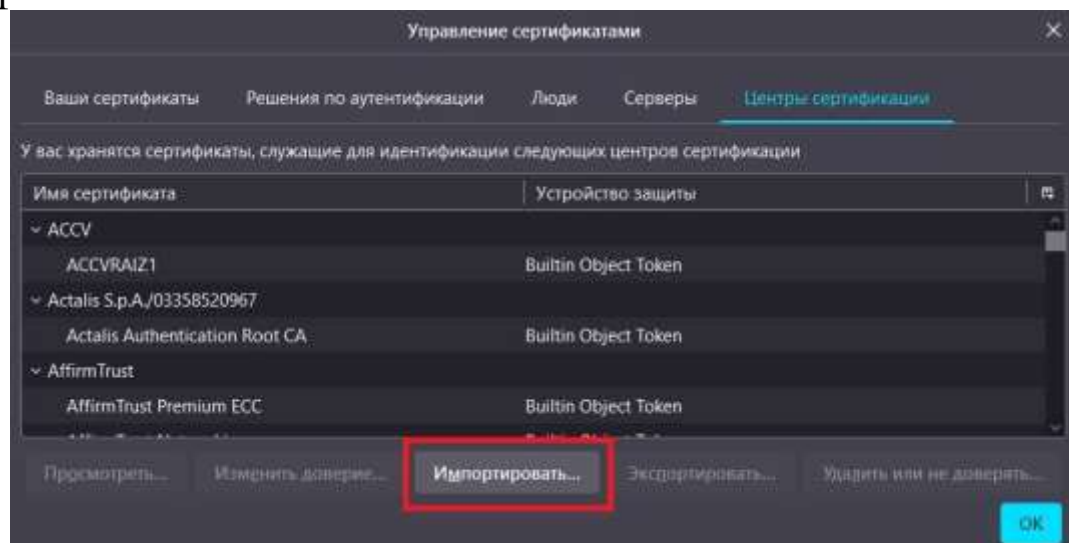


Рисунок 5.7 – Импорт Сертификатов

Нажимаем «Импортировать», чтобы загрузить сертификат.

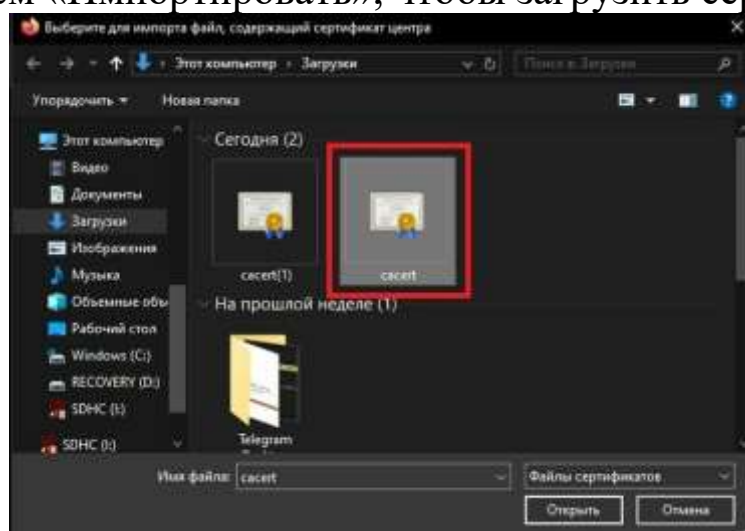


Рисунок 5.8 – Загрузка Сертификатов

Выбираем файл и нажимаем «Ок».

4. Попробуем зайти на любой сайт. Если Burp Suite настроен

правильно, запросы будут перехватываться – информация появится во вкладке Intercept.

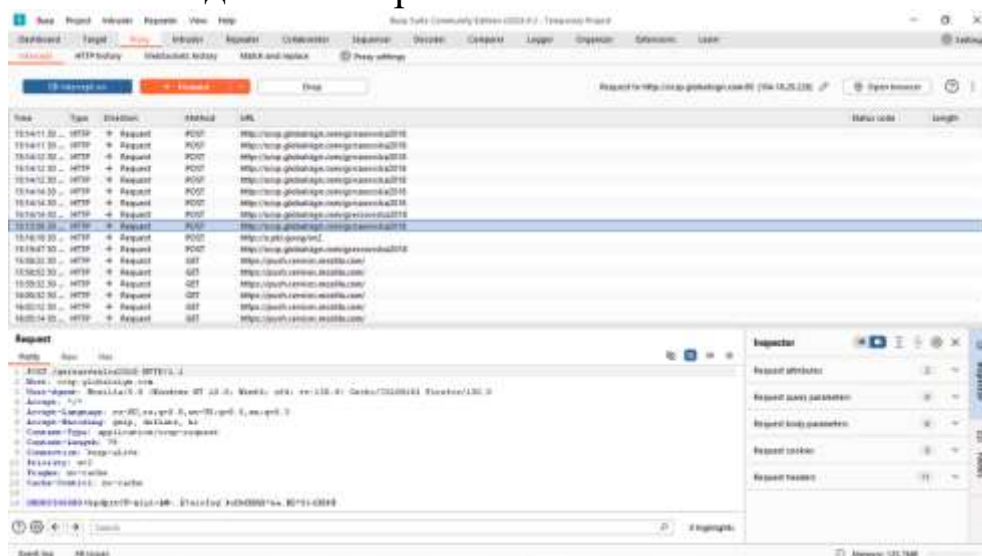


Рисунок 5.9 – Перехват запросов

Информация об обмене данными отображается в приложении. Теперь Burp Suite готов перехватывать и анализировать HTTP/HTTPS-трафик, что позволяет исследовать, как веб-приложения обрабатывают данные.

Задание к лабораторной работе №5

1. Задание:

1. Загрузить и запустить уязвимое веб-приложение OWASP Juice Shop: <https://owasp-juice.shop>.
2. Используя одну из XSS-уязвимостей: получить сессию авторизации пользователя и зайти от его имени на сайт; перенаправить пользователя на другой сайт.

2. Задание Wireshark -Анализ HTTP-авторизации:

1. Откройте любое веб-приложение с авторизацией
2. Введите логин и пароль
3. В Wireshark найдите POST-запрос к /login
4. В разделе **Follow TCP Stream** просмотрите содержимое запроса:

http

POST /api/login HTTP/1.1

Host: test.com

Content-Type: application/x-www-form-urlencoded

username=admin&password=123456

2.1 Анализ HTTPS-трафика.

HTTPS-трафик зашифрован. Чтобы его проанализировать:

1. Установите переменную окружения SSLKEYLOGFILE
2. Настройте браузер для экспорта ключей сессии.
3. Укажите путь к файлу в Wireshark: Edit → Preferences →

Protocols → TLS → (Pre)-Master-Secret log filename.

3. Задание

Рассмотрим кейс с уязвимым сайтом (используем стенд, например, We;come DVWA – Damn Vulnerable Web Application или OWASP WebGoat | OWASP Foundation).

1. Установка DVWA

Скачайте DVWA с We;come DVWA – Damn Vulnerable Web Application.

Установите XAMPP или LAMP для локального сервера.

Разместите файлы DVWA в папке htdocs (или аналогичной).

Настройте базу данных в config.inc.php.

Запустите сервер и откройте DVWA в браузере.

Обнаружение уязвимости.

Проверим форму входа на SQL-инъекцию:
Вводим – если появляется ошибка SQL, уязвимость есть.

Пробуем OR 1=1 -- – если авторизация проходит, инъекция подтверждена.

3. Извлечение данных с помощью UNION-атаки.

Если приложение выводит результаты запроса, можно использовать UNION для получения данных из других таблиц.

Пример запроса:

SQL:

```
' UNION SELECT username, password FROM users --
```

В ответе могут оказаться хэши паролей, которые затем можно взломать через John the Ripper password cracker или hashcat – advanced password recovery.

Лабораторная работа №6

Настройка HTTPS и создание самоподписанных сертификатов

Цель работы: Получить практические навыки настройки HTTPS-соединения для веб-сервера, освоить процесс генерации самоподписанных SSL/TLS-сертификатов с использованием инструментария OpenSSL, а также научиться настраивать веб-серверы (Apache/Nginx) для работы с этими сертификатами.

Теоретические положения

HTTPS (HyperText Transfer Protocol Secure) – это расширение протокола HTTP, обеспечивающее шифрование передаваемых данных между клиентом и сервером с использованием криптографических протоколов SSL/TLS. SSL-сертификат – это цифровая подпись вашего сайта, подтверждающая его подлинность. Использование сертификата позволяет защитить как владельца сайта, так и его клиентов.

SSL-сертификат даёт возможность владельцу применить к своему сайту технологию SSL-шифрования. Таким образом, назначение SSL-сертификата – обеспечить безопасное соединение между сервером и браузером пользователя, надёжно защитить данные от перехвата и подмены.

SSL (Secure Sockets Layer) и его преемник **TLS (Transport Layer Security)** – это криптографические протоколы, обеспечивающие защищенное соединение в сети Интернет.

После применения SSL невозможно подделать пакеты с данными через взлом Wi-Fi сети, провайдера, вмешиваясь в обмен информацией со стороны сервера или иными способами. Цифровая подпись сайта, которая нужна для работы протокола защищенной передачи данных в сети, обеспечивает шифрованное соединение между пользователем и сайтом. То есть информация защищается от посторонних – провайдера, иных подключенных к единой сети компьютеров, администраторов, подключенных к Wi-Fi устройств.

SSL сертификаты нужны банкам, интернет-магазинам или для установки на корпоративных сайтах. Они также необходимы для социальных сетей, форумов, информационных и иных ресурсов.

Виды сертификатов и их стоимость.

Подписанный самостоятельно – по сути, любой может дать се-

бе сертификат и его подписать. Этот вариант не подтверждает фактические сведения о компании (название, расположение, адрес, страна, контакты), а лишь используется для создания зашифрованного подключения.

Подписанный не доверенным центром сертификации – данные, предоставленные владельцем, подтверждались не официальными сертифицированными организациями, а какими-то «проверяющими» центрами, без проверки фактических сведений.

Подписанный центром сертификации (ЦС) – наиболее достоверный, при выдаче были проверены сведения о компании, вплоть до места ее физического присутствия.

От доверенных компаний – подтверждается через партнеров, что сайт принадлежит компании, сведения о владельце сохраняются и передаются в ЦС, проверяется существование компании.

Сегодня браузеры различают предоставляемые сайтами сертификаты, на подписанные ЦС или сертификаты от доверенных компаний ошибки и предупреждения не выдаются. В то же время, самоподписанные и изготовленные не доверенными компаниями выделяются как «НЕБЕЗОПАСНЫЙ РЕСУРС».

По умолчанию браузеры не доверяют самоподписанным сертификатам, рекомендуется использовать его только для внутренних целей или в целях тестирования.

OpenSSL – это криптографическая библиотека с открытым исходным кодом, предоставляющая инструменты для работы с SSL/TLS, генерации ключей и сертификатов.

Создание самоподписанный SSL сертификат и установить его на Windows. Устанавливаем OpenSSL.

Эта утилита позволит нам быстро создать самоподписанный ssl сертификат для всех наших дальнейших работ.

Вам нужно сделать несколько простых шагов:

1. Скачать OpenSSL по ссылке (<https://slproweb.com/products/Win32OpenSSL.html>), в соответствии с разрядностью вашей системы (у большинство 64 бита, но бывают и исключения);

2. Выполнить установку программы;

3. Далее вам необходимо придумать и создать папку куда будет создан самоподписанный SSL сертификат утилитой OpenSSL, например, это будет не системный диск C, а вспомогательный –

D:\My Certs;

4. Запустить от имени администратора командную строку (CMD) и выполнить команду `cd /d D:\My Certs` – она отправит нас в созданную ранее нами папку для выпуска самоподписанного SSL сертификата;

5. Осталось выполнить еще одну команду:

`openssl req -x509 -sha256 -nodes -days 365 -newkey rsa:2048 -keyout privateKey.key -out certificate.crt`

COPY

6. После отправки команды на генерацию самоподписанного SSL сертификата система предложит вам заполнить несколько полей, сделать это можно в произвольном формате. Далее нам остается перейти в созданную папку, это D:\My Certs и проверить их наличие. В папке должен корректно создаться 2 файла:

certificate.crt

privateKeykey

Вы увидите информацию о сертификате: версию, серийный номер, срок действия, владельца и публичный ключ. После этих действий ваш сайт будет доступен по протоколу HTTPS.

Установка Веб-сервера Apache для 1С Windows 10 (для примера).

Для настройки веб-сервера 1С нужно установить соответствующий компонент в установщике 1С.

Подготовка:

Убедитесь, что платформа 1С:Предприятие установлена с компонентой «**Модули расширения веб-сервера**».

Скачайте Apache 2.4 для Windows (например, с arachelounge.com). **Важно:** разрядность Apache (32 или 64 бит) должна совпадать с разрядностью установленной платформы 1С.

После установить веб-сервер Apache, разархивировав его в каталоге C:\Apache24. После чего в командной строке его нужно установить и запустить.

```

Администратор: Командная строка - httpd.exe -k start
Microsoft Windows [Version 10.0.19042.631]
(c) Корпорация Майкрософт (Microsoft Corporation), 2020. Все права защищены.

C:\Windows\system32>cd C:\Apache24\bin

C:\Apache24\bin>httpd.exe -k install
Installing the 'Apache2.4' service
The 'Apache2.4' service is successfully installed.
Testing httpd.conf....
Errors reported here must be corrected before the service can be started.

C:\Apache24\bin>httpd.exe -k start

```

Рисунок 6.1 – Установка веб-сервера Apache

После чего по заходу на localhost будет открываться стартовая страница Apache.

Публикация базы 1С:

Запустите 1С:Предприятие от имени Администратора.

Откройте нужную базу в режиме Конфигуратор.

Выберите меню Администрирование -> Публикация на веб-сервере....

После чего нужно опубликовать базу данных 1С. Задаём имя и путь каталога к серверу.

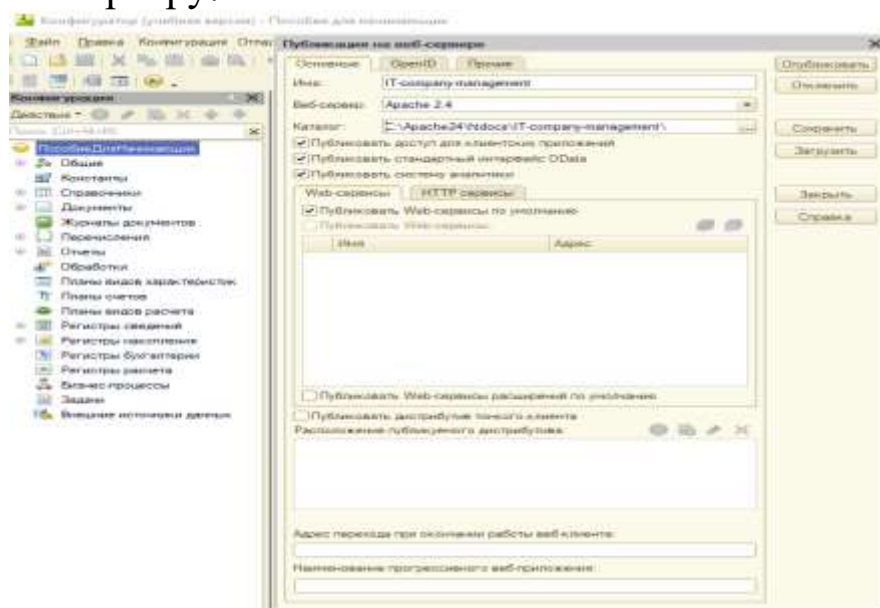


Рисунок 6.2 – Публикация 1С на веб-сервере

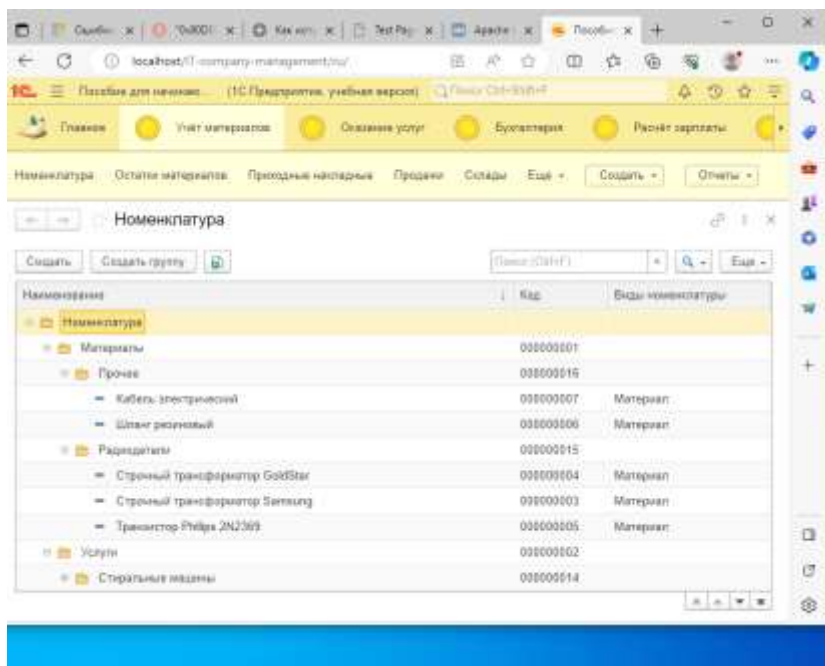


Рисунок 6.3 – Публикация 1С на веб-сервере

Теперь база данных открывается в браузере. База будет доступна по адресу: http://localhost/имя_базы.

Задание к лабораторной работе №6

1. Задание №1. Установка и настройка веб-сервера Apache.
2. Задание №2. Установка OpenSSL.
3. Задание №3. Создание самоподписанного SSL-сертификата.
4. Задание №4. Настройка HTTPS в Apache.
5. Задание №5. Подключение модуля 1С к Apache.
6. Задание №6. Создание каталога публикации и настройка виртуального хоста.
7. Задание №7. Тестирование доступа к 1С через веб-клиент.

Лабораторная работа №7

Защита от brute-force атак с ограничением попыток входа

Цель работы: Изучить защиту веб-приложений от атак подбора паролей (brute-force) путём ограничения количества попыток входа, временной блокировки аккаунтов и IP-адресов.

Теоретические положения

Brute-force (атака полным перебором) — это метод несанкционированного доступа к системе, при котором злоумышленник пытается подобрать пароль путем систематического перебора всех возможных комбинаций символов или использования словаря распространенных паролей.

Брутфорс не опирается на методы социальной инженерии (в некоторых случаях только частично), базируясь на мощности аппаратной части и эффективности программного обеспечения. Сегодня специалисты выделяют несколько способов выполнения взлома:

- Простая брутфорс-атака. Систематический подбор логинов и паролей с их отправкой для авторизации с помощью специального приложения. Самый простой способ, не требующий активации дополнительных модулей, но и наименее эффективный. Актуален для сети мощных компьютеров, каждый из которых может генерировать за короткое время множество комбинаций паролей и логинов.

- Атака по словарю. Способ учитывает особенности человеческой психики. В частности то, что многим пользователям попросту лень придумывать сложные пароли и хранить их отдельно, поэтому они используют простейшие варианты типа `qwerty1234` или `password`. Благодаря многочисленным сливам данных составлена обширная база с наиболее часто используемыми паролями. Именно ими оперирует приложение для брутфорса.

- Подстановка учетных данных. Хакерами используются похищенные базы с логинами и паролями от различных сервисов. Метод эффективен, так как многие пользователи в разных сервисах используют одни и те же логины и пароли. В этом случае задача злоумышленников — методом подбора определить, где еще зарегистрированы пользователи, данные о логинах и паролях которых удалось заполучить. Подбираются не логины и пароли, а сервера для авторизации.

- **Обратная брутфорс-атака.** Способ, при котором используется небольшая база часто используемых простых паролей, а подбираются логины, под которыми могут быть зарегистрированы пользователи. Метод эффективен для взлома ресурсов с большим количеством активных пользователей – например, учетных записей в социальных сетях.

- **Гибридная атака.** Использование базового словаря с популярными паролями дополняется алгоритмами подбора. Эффективен для быстрого взлома небольшого количества учетных записей – например, форумов с невысокой или средней посещаемостью.

- **Таргетированная атака.** Целью становится конкретный человек. На него собирается максимально подробное досье, содержащее как можно больше информации, в том числе той, которая может показаться второстепенной – например, важные даты, девичья фамилия бабушки или кличка домашнего питомца. Все эти данные вносятся в приложение для взлома, которое комбинирует их различными способами.

Способы защиты от брутфорс-атак.

Брутфорс – эффективный, но не стопроцентный метод взлома. Ряд приемов поможет усложнить доступ злоумышленникам к персональным данным, а в ряде случаев сделать его невозможным. Наиболее эффективные из них:

- **Двухфакторная авторизация.** Для доступа к ресурсу нужно ввести не только логин и пароль, но и код, который отправляется в СМС на мобильный телефон пользователя. Хакеры, даже точно зная логин и пароль, не смогут авторизоваться, не имея доступа к привязанному номеру телефона.

- **Создание сложных паролей.** Чем длиннее пароль, тем его сложнее подобрать. На взлом пароля длиной 16 символов, состоящего из чередующихся строчных и заглавных букв, цифр и специальных символов, потребуется в миллионы раз больше запросов, чем на комбинацию из четырех цифр, особенно если это «1111» или «1234».

- **Ограничение попыток входа.** Временная блокировка аккаунта после нескольких неудачных попыток авторизации, с последующей заменой пароля на более надежный. Как вариант – постепенно увеличивающийся тайм-аут на авторизацию после каждой неудачной попытки.

- Использование капчи. Для авторизации пользователю нужно не только ввести логин и пароль, но и подтвердить, что он человек (достаточно просто установить галочку в соответствующем поле, чего приложения-брутфорсеры делать не умеют).

- Блокировка по IP. Ограничение попыток авторизации с одного IP-адреса. Белые списки адресов, с которых разрешена авторизация.

- Push-уведомления. Пользователь получает сообщения на смартфон при каждой авторизации на ресурсе. Если вход выполнен не им, он может быстро принять меры – например, разлогиниться на всех устройствах и поменять пароль.

- Хеширование и шифрование паролей. Даже если хакеры получают базу данных с данными авторизации пользователей, расшифровать такие пароли будет намного сложнее.

- Web Application Firewall. Защита веб-приложений от программных атак, в том числе брутфорса. Можно воспользоваться готовыми решениями: AWS WAF, ModSecurity, Cloudflare и их аналогами.

На практике защита реализуется на уровне веб-сервера, приложения или сети.

Уровень веб-сервера (Nginx/Apache):

Пример для Nginx (limit_req): позволяет ограничить количество запросов в секунду (RPS) для конкретного URL, например, страницы авторизации.

```
limit_req_zone $binary_remote_addr zone=login:10m rate=1r/s;
location /login {
    limit_req zone=login burst=5;
}
```

Уровень приложения (код):

Хранение счетчика неудачных попыток (в БД или Redis) с привязкой к IP или логину.

После NN неудачных попыток – блокировка доступа на TT минут.

Интеграция CAPTCHA после 2-3 неудачных попыток.

Уровень ОС/Сети:

Использование Fail2Ban: мониторинг логов приложения. Если IP совершает серию неудачных попыток входа, Fail2Ban автоматически добавляет его в правило iptables/nftables для блокировки.

Ключевые функции защиты:

php

// Подсчёт неудачных попыток

```
function countFailedAttempts($pdo, $username, $ip) {
    // Возвращает количество ошибок за последние 15 минут
}
```

// Блокировка аккаунта

```
function lockAccount($pdo, $username) {
    // Блокирует на 15 минут
}
```

// Блокировка IP

```
function blockIP($pdo, $ip) {
    // Блокирует на 60 минут
}
```

Основные параметры защиты:

php

// Рекомендуемые значения параметров

```
define('MAX_LOGIN_ATTEMPTS', 5); // Максимум попы-
ток входа
```

```
define('ACCOUNT_LOCK_TIME', 15); // Блокировка акка-
унта на 15 минут
```

```
define('IP_BLOCK_TIME', 60); // Блокировка IP на 60
минут
```

```
define('ATTEMPTS_WINDOW', 15); // Временное окно
для подсчета (минут)
```

```
define('DELAY_BASE_SECONDS', 2); // Базовая задержка
при ошибке (секунды)
```

Структура базы данных для хранения попыток:

sql

-- Таблица пользователей

```
CREATE TABLE users (
    id INT PRIMARY KEY AUTO_INCREMENT,
    username VARCHAR(50) UNIQUE NOT NULL,
    password_hash VARCHAR(255) NOT NULL,
    is_active BOOLEAN DEFAULT TRUE
);
```

```
-- Таблица попыток входа
CREATE TABLE login_attempts (
  id INT PRIMARY KEY AUTO_INCREMENT,
  username VARCHAR(50) NOT NULL,
  ip_address VARCHAR(45) NOT NULL,
  attempt_time      TIMESTAMP      DEFAULT      CUR-
RENT_TIMESTAMP,
  was_successful BOOLEAN DEFAULT FALSE
);
```

```
-- Таблица заблокированных аккаунтов
CREATE TABLE locked_accounts (
  id INT PRIMARY KEY AUTO_INCREMENT,
  username VARCHAR(50) NOT NULL,
  unlock_time TIMESTAMP NOT NULL
);
```

```
-- Таблица заблокированных IP
CREATE TABLE blocked_ips (
  id INT PRIMARY KEY AUTO_INCREMENT,
  ip_address VARCHAR(45) UNIQUE NOT NULL,
  unlock_time TIMESTAMP NOT NULL
);
```

Механизм задержки (Delay):

Искусственная задержка между попытками входа — важный механизм защиты, который:

Делает перебор неэффективным — снижает количество попыток в секунду.

Увеличивается с каждой ошибкой — прогрессивная задержка.

Не влияет на легитимных пользователей — заметна только при ошибках.

php

// Пример прогрессивной задержки

\$delay = \$base_delay * (1 + floor(\$failed_attempts / 3));

sleep(\$delay); // Пауза перед ответом сервера

Рекомендации по безопасности для разработчиков

При разработке веб-приложения:

Использовать **хэширование паролей** (password_hash() в PHP, bcrypt).

Не сообщать пользователю, что именно неверно: логин или пароль.

Реализовать **двухфакторную аутентификацию** для критических операций.

Ограничить количество запросов с одного IP (rate limiting).

Использовать **CAPTCHA** после нескольких неудачных попыток.

Для администраторов:

Настроить мониторинг подозрительной активности.

Использовать **fail2ban** для автоматической блокировки IP.

Регулярно обновлять CMS и плагины.

Использовать **Web Application Firewall (WAF)**.

Включить логирование всех попыток входа.

Задание к лабораторной работе №7

Выполнить задания представленные в таблице

№	Задание	Результат
1	Создать БД с таблицами users, login_attempts, locked_accounts	SQL-схема
2	Реализовать форму входа с подсчётом попыток	PHP + HTML
3	Добавить блокировку аккаунта после 5 ошибок на 15 минут	Функция lockAccount()
4	Добавить блокировку IP после 10 ошибок на 60 минут	Функция blockIP()
5	Создать защищённую страницу и выход	dashboard.php, logout.php
6	Протестировать защиту (10 неудачных попыток)	Отчёт о тестировании

Лабораторная работа №8

Настройка TLS

Цель работы: Изучить принципы работы протокола TLS, освоить практические навыки настройки защищённых каналов связи с использованием OpenSSL, а также научиться настраивать веб-серверы для поддержки TLS и создавать самоподписанные сертификаты.

Теоретические положения

TLS (Transport Layer Security) – это криптографический протокол, обеспечивающий защищённую передачу данных между клиентом и сервером по сети. Он является преемником протокола SSL (Secure Sockets Layer) и в настоящее время является общепринятым стандартом для защиты сетевых соединений.

Установка OpenSSL.

1.1. Для Linux (Ubuntu/Debian):

```
bash
sudo apt update
sudo apt install openssl
openssl version
```

Ожидаемый результат:

text

OpenSSL 3.0.2 15 Mar 2022 (Library: OpenSSL 3.0.2)

1.2. Для Windows:

Скачайте OpenSSL с официального сайта

Установите в C:\OpenSSL-Win64

Добавьте путь в переменную PATH

Вставить скриншот результата команды openssl version.

Асимметричное шифрование (RSA).

3.1. Генерация ключевой пары RSA.

```
bash
```

```
# Генерация закрытого ключа (2048 бит)
```

```
openssl genpkey -algorithm RSA -pkeyopt rsa_keygen_bits:2048 -
out server.key
```

```
# Извлечение открытого ключа
```

```
openssl pkey -in server.key -out server.pub -pubout
```

3.2. Просмотр информации о ключах

```
bash
```

```
# Информация о закрытом ключе
```

```
openssl pkey -in server.key -text -noout
```

```
# Информация об открытом ключе
```

```
openssl pkey -in server.pub -pubin -text -noout
```

3.3. Асимметричное шифрование сообщения

```
bash
```

```
# Шифрование открытым ключом
```

```
openssl rsautl -encrypt -pubin -inkey server.pub -in message.txt -  
out message.rsa.enc
```

```
# Расшифровка закрытым ключом
```

```
openssl rsautl -decrypt -inkey server.key -in message.rsa.enc -out  
message.rsa.dec
```

Вставить скриншоты выполнения команд.

Часть 4. Создание самоподписанного сертификата.

4.1. Генерация сертификата (одной командой):

```
bash
```

```
openssl req -x509 -newkey rsa:2048 -nodes -keyout server.key -out  
server.crt -days 365 -subj  
"/C=RU/ST=Moscow/L=Moscow/O=KuzSTU/OU=IT/CN=localhost"
```

Параметры:

-x509 – создание самоподписанного сертификата.

-newkey rsa:2048 – генерация новой RSA-пары (2048 бит).

-nodes – не шифровать закрытый ключ.

-days 365 – срок действия 1 год.

-subj – информация о владельце.

4.2. Пошаговая генерация (альтернативный метод):

```
bash
```

```
# Шаг 1: Генерация запроса на сертификат (CSR)
```

```
openssl req -new -key server.key -out server.csr
```

```
# Шаг 2: Создание самоподписанного сертификата
```

```
openssl x509 -req -days 365 -in server.csr -signkey server.key -out  
server.crt
```

4.3. Просмотр информации о сертификате:

```
bash
```

```
openssl x509 -in server.crt -text -noout
```

Задание к лабораторной работе №8

1. Генерация ключей и сертификатов:

Создайте приватный RSA-ключ.

Сформируйте запрос на подпись сертификата (CSR) и на его основе создайте самоподписанный сертификат с помощью OpenSSL.

2. Настройка веб-сервера:

Установите или настройте веб-сервер (Nginx или Apache) для работы по HTTPS. Подключите созданные сертификаты в конфигурационный файл сервера.

3. Проверка и отладка:

Проверьте успешность настройки с помощью команды `openssl s_client -connect localhost:443`.

Дополнительно: попробуйте зайти на сайт через браузер и проанализируйте предупреждение о «недоверенном сертификате» в инструментах разработчика. Объясните, почему браузер выдает ошибку "Небезопасное соединение" для самоподписанного сертификата.

Лабораторная работа №9

Создание blockchain smart contract с защитой от reentrancy attacks

Цель работы: Изучить механизм уязвимости reentrancy в Solidity, понять причины её возникновения и научиться применять методы защиты для создания безопасных децентрализованных приложений

Теоретические положения

Блокчейн-приложение – это программное обеспечение, которое использует технологию распределённого реестра (блокчейн) для хранения данных и/или выполнения бизнес-логики. В отличие от традиционных приложений с централизованной базой данных, блокчейн-приложения работают в децентрализованной сети, где данные распределены между множеством узлов.

Типы блокчейн-приложений.

По типу блокчейна:

Первый критерий выбора – кто может участвовать в сети. От этого зависит баланс между децентрализацией и контролем.

Публичные (Public) блокчейны.

Открыты для всех. Любой может присоединиться к сети, читать данные, отправлять транзакции. Это классический блокчейн в понимании большинства – максимально децентрализованный и устойчивый. Примеры: Ethereum, Bitcoin, Solana.

Публичные блокчейны идеально подходят для продуктов, где важна открытость и независимость от конкретной организации: DeFi, NFT, открытые реестры.

Приватные (Private) блокчейны.

Доступ ограничен. Участники определяются администратором сети. Это решение для корпоративных задач, где децентрализация нужна внутри компании или группы доверенных партнёров, но не для широкой публики. Примеры: Hyperledger Fabric, R3 Corda.

Консорциумные (Consortium) блокчейны.

Управляются группой организаций. Частично децентрализованы, но с контролируемым доступом. Это компромисс между публичными и приватными сетями – подходит для отраслевых решений, где несколько компаний хотят совместно управлять общими данными. Примеры: Quorum, Hyperledger Besu.

По функциональности:

Помимо типа сети, блокчейн-приложения различаются по тому, какую задачу они решают. Понимание этих категорий поможет точнее сформулировать требования к вашему проекту.

Децентрализованные приложения (dApps).

Приложения, работающие на блокчейне. Обычно включают смарт-контракты для бизнес-логики и фронтенд для взаимодействия с пользователем – часто это мобильное приложение или веб-интерфейс. Это наиболее распространённый тип блокчейн-приложений.

Примеры: DeFi-платформы, NFT-маркетплейсы, децентрализованные биржи.

Токенизированные системы.

Приложения для создания и управления цифровыми активами (токенами). Токены могут представлять ценные бумаги, права собственности, баллы лояльности – практически любой актив, который можно оцифровать. Примеры: токенизация недвижимости, программы лояльности на блокчейне.

Системы отслеживания (Track & Trace).

Приложения для прозрачного отслеживания товаров и документов в цепочке поставок. Каждый этап фиксируется в блокчейне, создавая неизменяемую историю происхождения. Примеры: отслеживание происхождения продуктов, контроль подлинности лекарств.

Системы цифровой идентификации.

Приложения для управления цифровыми идентичностями без централизованного хранилища. Пользователь контролирует свои данные и решает, кому их предоставить. Примеры: верификация документов, KYC на блокчейне.

Определившись с типом приложения, переходим к выбору конкретной платформы – это решение определит технологический стек и возможности вашего продукта.

Платформы для разработки блокчейн-приложений.

Выбор платформы – одно из ключевых решений в проекте. Оно определяет производительность, стоимость транзакций, язык разработки смарт-контрактов и в целом возможности приложения. Смена платформы после старта разработки – болезненный и дорогой процесс, поэтому важно сделать осознанный выбор на старте.

Публичные платформы.

Если ваше приложение ориентировано на широкую аудиторию и открытость – публичные блокчейны обеспечат готовую инфраструктуру и доступ к существующей экосистеме пользователей.

Ethereum.

Самая популярная платформа для смарт-контрактов с крупнейшей экосистемой. Здесь больше всего разработчиков, инструментов и готовых решений. Если вы создаёте DeFi-продукт или NFT-проект – Ethereum часто становится первым выбором благодаря сетевому эффекту.

Solana.

Высокопроизводительный блокчейн с очень низкими комиссиями. Подходит для приложений, где критична скорость и низкая стоимость транзакций – например, игры или высоконагруженные DEX.

Polygon.

Layer 2 решение для Ethereum. Сочетает совместимость с Ethereum и высокую производительность. Если вам нужна экосистема Ethereum, но с более низкими комиссиями – Polygon может стать оптимальным выбором.

BNB Chain (Binance).

Блокчейн от крупнейшей криптобиржи. Высокая производительность, но более централизованный, чем Ethereum или Solana.

Приватные и консорциумные платформы.

Для корпоративных решений, где важна конфиденциальность и контроль, существуют специализированные платформы. Они требуют больше усилий на развёртывание, но дают полный контроль над сетью.

Hyperledger Fabric.

Модульная платформа от Linux Foundation, разработанная специально для корпоративных блокчейн-решений. Поддерживает приватные каналы и гибкую настройку консенсуса – можно создать сеть точно под ваши требования.

R3 Corda.

Платформа, разработанная специально для финансового сектора с учётом регуляторных требований.

Для быстрого выбора – сводная таблица основных характеристик.

Таблица 9.1 – Сравнительная таблица платформ

Платформа	Тип	TPS	Смарт-контракты	Лучший сценарий
Ethereum	Публичный	15-30	Solidity	DeFi, NFT
Solana	Публичный	65K	Rust	Игры, DEX
Polygon	L2	7K	Solidity	Масштабируемые dApps
Hyperledger	Приватный	3.5K	Go, Java	Корпоративные решения
Corda	Приватный	-	Kotlin	Финансы

4. Смарт-контракты: основа децентрализованных приложений

Если блокчейн – это фундамент децентрализованного приложения, то смарт-контракты – его мозг. Именно в них содержится бизнес-логика: правила взаимодействия, условия транзакций, алгоритмы распределения средств. Понимание того, как работают смарт-контракты, критически важно для принятия правильных архитектурных решений.

Смарт-контракт – это программа, хранящаяся в блокчейне и автоматически исполняющаяся при выполнении определённых условий. В отличие от обычного кода, который можно изменить в любой момент, смарт-контракт после публикации становится неизменяемым – это и преимущество (доверие), и вызов (ошибки нельзя просто исправить).

Языки программирования смарт-контрактов.

Выбор языка определяется платформой. У каждого есть свои особенности, которые влияют на скорость разработки и безопасность.

Solidity (Ethereum, Polygon, BNB Chain).

Самый популярный язык для смарт-контрактов. Синтаксис похож на JavaScript, что упрощает вход для веб-разработчиков. Огромное сообщество и множество готовых библиотек.

Rust (Solana).

Используется для программ на Solana. Более сложный в освоении, но обеспечивает высокую производительность и безопасность памяти. Если ваш проект требует максимальной скорости – Rust оправдывает инвестиции в обучение команды.

Go, Java, JavaScript (Hyperledger Fabric).

Архитектура блокчейн-приложения.

Блокчейн-приложение – это не только смарт-контракт. Это комплексная система, включающая несколько уровней, каждый из которых решает свои задачи. Ошибочно думать, что «всё будет в блокчейне» – такой подход приводит к высоким затратам и низкой производительности. Правильная архитектура балансирует между децентрализацией и эффективностью.

Компоненты архитектуры.

Рассмотрим каждый компонент подробнее, чтобы понять его роль и особенности в контексте блокчейн-разработки.

1. Frontend.

Веб или мобильное приложение для взаимодействия с пользователем. Для блокчейн-приложений есть важные особенности:

- Интеграция с кошельками (MetaMask, WalletConnect).
- Отображение состояния блокчейна.
- Подпись транзакций пользователем.

Пользователь не должен понимать, как работает блокчейн под капотом – интерфейс должен быть таким же удобным, как у обычных приложений.

2. Backend (серверная часть).

Не все операции должны идти в блокчейн – это дорого и медленно. Backend обрабатывает:

- Аутентификацию и авторизацию.
- Кэширование данных блокчейна.
- Сложную бизнес-логику, не требующую децентрализации.
- Интеграции с внешними системами.

3. Блокчейн-нода.

Узел сети, через который приложение взаимодействует с блокчейном. Есть несколько вариантов:

- Собственная нода (максимальный контроль).
- Нода-как-сервис (Infura, Alchemy, QuickNode).
- Легкий клиент (для мобильных приложений).

4. Смарт-контракты.

Бизнес-логика, выполняемая в блокчейне. Должны содержать только то, что действительно требует децентрализации – проверки, переводы, фиксацию состояний.

5. Off-chain хранилище.

Хранение данных, которые не подходят для блокчейна:

- Большие файлы (IPFS, Arweave).

- Часто изменяемые данные.

Персональные данные (GDPR).

Безопасность блокчейн-приложений.

Безопасность в блокчейне критична по простой причине: ошибки приводят к потере средств без возможности отката. Нет «откатить транзакцию», нет «восстановить из бэкапа» — если активы украдены, они украдены навсегда. Это делает безопасность не просто важной, а абсолютно критичной частью проекта.

Распространённые уязвимости смарт-контрактов.

Понимание типичных уязвимостей помогает предотвратить их на этапе разработки.

Reentrancy (повторный вход).

Атакующий вызывает функцию контракта повторно до завершения первого вызова. Это позволяет «обмануть» контракт и вывести средства многократно. Reentrancy-атака (атака повторного входа) — это уязвимость смарт-контрактов, возникающая когда функция делает внешний вызов другого контракта до того, как обновит собственное состояние. Злоумышленник может использовать это для повторного входа в функцию и многократного выполнения действий (например, вывода средств) пока баланс еще не обновлен.

Печально известный пример:* взлом The DAO в 2016 году — похищено \$60 млн. Эта атака привела к хард-форку Ethereum и появлению Ethereum Classic.

Решение: паттерн Checks-Effects-Interactions, модификатор ReentrancyGuard.

Integer Overflow/Underflow

Переполнение числовых типов при арифметических операциях. Например, вычитание из нуля даёт максимальное число.

Решение: использование Solidity 0.8+ (встроенная защита) или библиотеки SafeMath.

Access Control

Отсутствие или неправильная проверка прав доступа к функциям. Критические функции оказываются доступны всем.

Решение: использование модификаторов `onlyOwner`, ролевой модели OpenZeppelin.

Front-running

Атакующий видит транзакцию в mempool и отправляет свою с

большей комиссией, опережая жертву. Это позволяет «перехватывать» выгодные сделки.

Решение: commit-reveal схемы, flashbots, приватные транзакции.

Напишем смарт-контракт, который будет передавать указанное кол-во денег, показывать сумму переданных денег и остаток накопленных денег.

1. Создание файла MyContract.

В открытом RemixIDE перейдём на вкладку Fileexplorers, далее создайте в папке contracts новый файл и назовите его «MyContract» где мы и будем писать умный смарт-контракт.

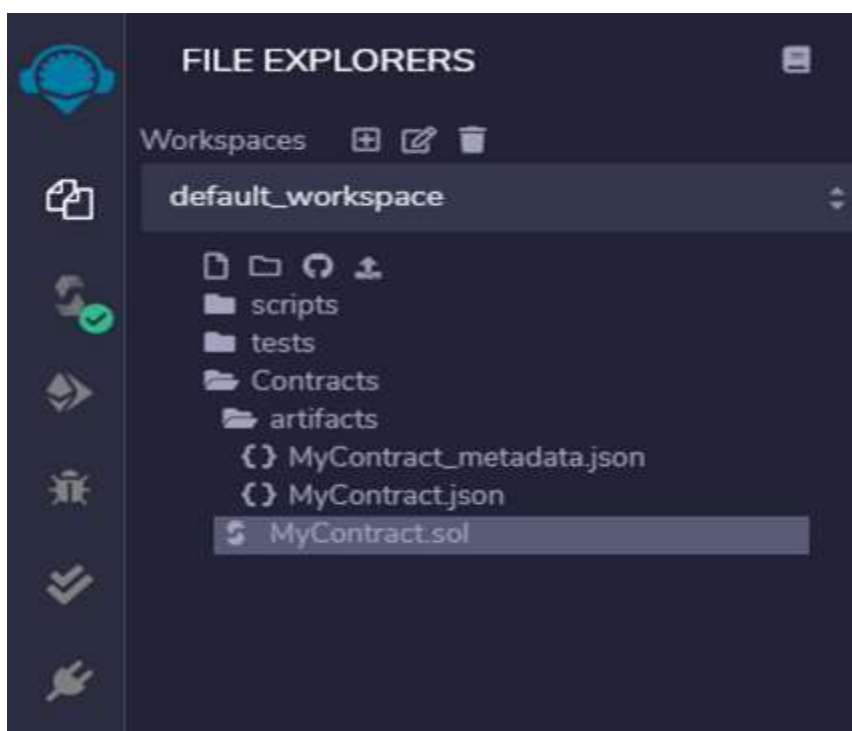


Рисунок 9.1 – Создание файла «MyContract»

Открываем файл и первым делом укажем директиву для компилятора и назовём наш контракт MyContract.

Переменная сумма будет содержать накопленные деньги, отправленные исполнителями договора создателю договора. Поле значения будет содержать стоимость контракта. Когда исполнители исполняют контракт, поле значения будет изменено, чтобы отразить сбалансированную стоимость контракта. В конструкторе контракта мы устанавливаем значения этих двух переменных.

Чтобы проверить собранную сумму в любой момент времени, мы предоставляем метод публичного контракта `getAmount`.

Чтобы получить сбалансированную стоимость контракта в любой момент времени, мы определяем метод `getBalance` следующим образом:

```
function getBalance() public view returns(uint) {
    returnvalue;
}
```

Наконец, мы пишем метод контракта (Отправить). Это позволяет клиентам отправлять деньги создателю контракта:

```
function send(uint newDeposit) public {
    value = value - newDeposit;
    amount = amount + newDeposit;
}
```

В итоге у вас должен получиться код договора, как на Рисунок 9.2.

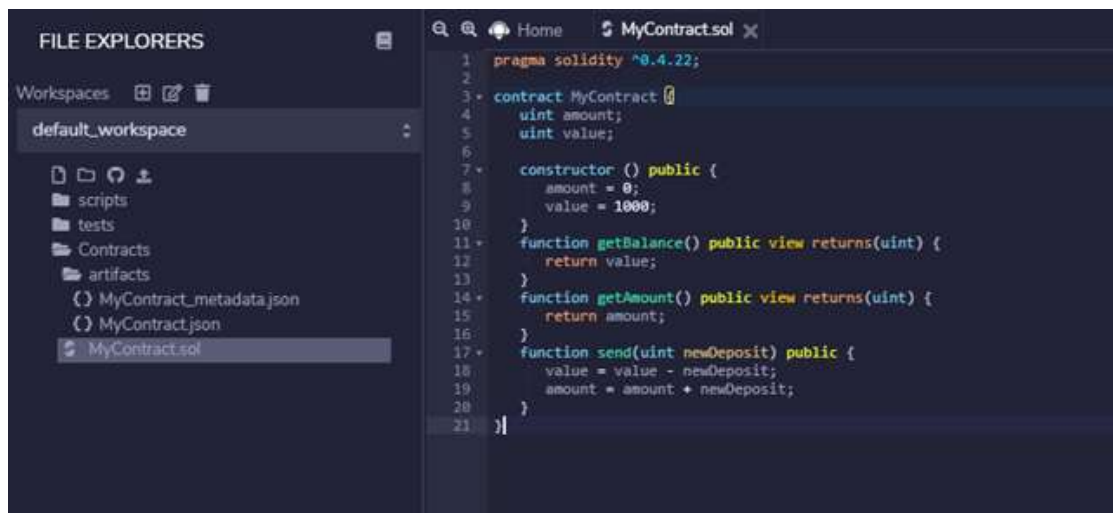


Рисунок 9.2 – Код договора «MyContract»

2. Компиляция и развертывание контракта.

Далее во вкладке `SOLIDITYCOMPILER` мы сможем скомпилировать договор, нажав на кнопку с заголовком «Начать компиляцию» (Рисунок 9.3).

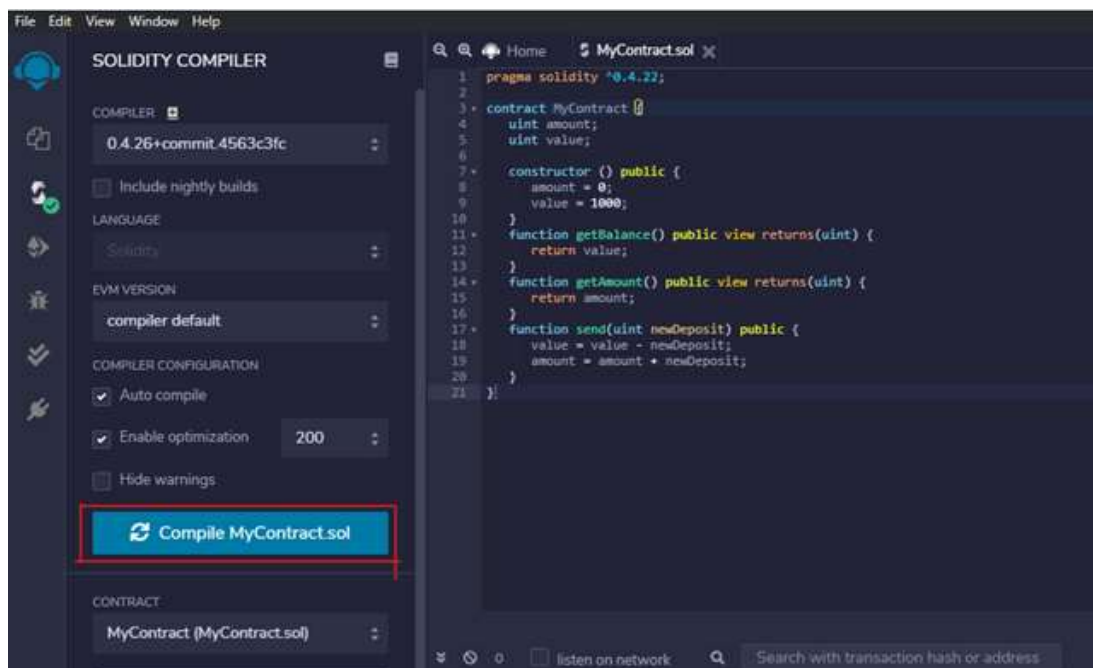


Рисунок 9.3 – Компиляция контракта

Если есть какая-либо опечатка, исправьте это в окне кода. Убедитесь, что код скомпилирован полностью без ошибок.

Нажмите на пункт меню «Выполнить». Ниже вы увидите кнопку Deploy, нажмите на нее, чтобы развернуть контракт. Контракт будет развернут на встроенном блокчейне Remix. Когда вы щелкнете по развернутому контракту, вы увидите различные открытые методы, предоставляемые контрактом (Рисунок 9.4).

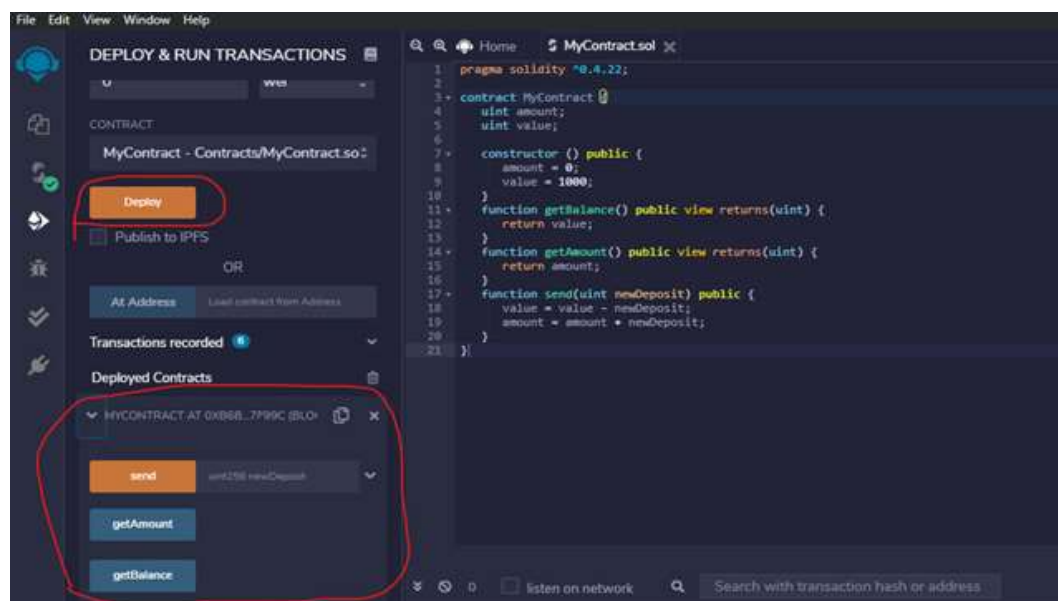


Рисунок 9.4 – Развернутый контракт

Первый метод `send` (Отправка) содержит поле редактирования перед ним. Здесь вы введете параметры, необходимые для метода контракта. Два других метода не принимают никаких параметров.

Теперь введите некоторую сумму, например 100, после функции отправки, отображаемой в окне контракта. Нажмите кнопку отправки. Это выполнит метод отправки контракта, уменьшая значение поля значения контракта и увеличивая значение поля суммы.

Предыдущее действие отправки денег уменьшило стоимость контракта на 100. Теперь вы можете проверить это, вызвав метод `getBalance` контракта. Стоимость контракта теперь снижена до 900. Чтобы рассмотреть сумму денег, собранную на данный момент по этому контракту, нажмите на кнопку `GetAmount`. Значение поля количества изменилось с 0 до 100.

Попробуйте выполнить несколько операций отправки и проверьте значения контракта и поля суммы, чтобы сделать вывод, что развернутый контракт выполняется должным образом

Итог работы: Мы написали и протестировали умный смарт-контракт в `RemixIDE`, который передавал указанное кол-во денег, показывал сумму переданных денег и остаток накопленных денег.

Теперь, когда вы поняли, как работает базовый контракт, рассмотрим, почему он уязвим. Если в методе `send` вместо простого изменения переменных мы бы вызывали внешний адрес (например, перевод средств), злоумышленник мог бы создать контракт, который "зацикливает" вызов этой функции до того, как состояние контракта обновится.

Для защиты мы применим паттерн `Check-Effects-Interactions (CEI)`.

Код защищенного контракта (`MySecureContract`).

Создайте новый файл `MySecureContract.sol` в `Remix IDE`:

`text`

`pragma solidity ^0.8.0;`

`contract MySecureContract {`

`uint256 public amount;`

`uint256 public value;`

`constructor() {`

`amount = 0;`

```

    value = 1000;
}

// Защита с помощью паттерна CEI
function send(uint256 newDeposit) public {
    // 1. Check: Проверка условий
    require(value >= newDeposit, "Insufficient contract value");

    // 2. Effects: Сначала обновляем состояние (безопасная зо-
на)
    value = value - newDeposit;
    amount = amount + newDeposit;

    // 3. Interactions: Только потом внешние взаимодействия
    // (в данном примере логика проста, но если бы был пере-
вод,
    // он стоял бы здесь)
}

function getAmount() public view returns(uint256) {
    return amount;
}

function getBalance() public view returns(uint256) {
    return value;
}
}

```

4. Дополнительное задание: Использование ReentrancyGuard
 Для более сложных контрактов рекомендуется использовать библиотеку OpenZeppelin.

Задание к лабораторной работе №9

1. Задания для выполнения:

Создание уязвимого контракта: напишите простой контракт "Кошелек" с функцией вывода средств, который подвержен атаке reentrancy.

Разработка атакующего контракта: напишите второй контракт, который вызывает функцию вывода из первого и использует fallback-функцию для совершения рекурсивного вызова.

Внедрение защиты: модифицируйте первый контракт, используя паттерн **Check-Effects-Interactions** или библиотеку ReentrancyGuard от OpenZeppelin.

Содержание самостоятельной работы

Цель самостоятельной работы обучающихся – получить новые знания по дисциплине «Безопасность программного обеспечения».

Самостоятельная работа необходима для формирования у обучающихся способности самостоятельно решать задачи профессиональной деятельности, формирования умения и навыков планирования времени, формирования стремления развиваться и совершенствоваться.

Виды самостоятельной работы обучающихся указаны в табл. 3.

Таблица 3 – Виды самостоятельной работы

№ п/п	Вид СРС
1	1. Безопасная работа с файлами 2. Реализация безопасной десериализации данных 3. Аудит логов безопасности и выявление подозрительной активности 4. Настройка CORS политик для веб-приложений 5. Защита от DDOS атак с помощью rate limiting 6. Безопасная работа с памятью в приложениях 7. Создание безопасного API с валидацией всех входных данных
2	1. Реализация end-to-end шифрования для мессенджера на Signal Protocol 2. Создание secure OAuth 2.0 провайдера с PKCE и защитой от атак 3. Имплементация JWE (JSON Web Encryption) для защищённых токенов 4. Разработка безопасного voting system с homomorphic encryption 5. Создание cryptocurrency wallet с ECDSA и hierarchical deterministic keys 6. Реализация secure password manager с client-side encryption 7. Настройка HSM эмулятора для аппаратной защиты ключей 8. Разработка secure file storage с encryption at rest и in transit 9. Имплементация zero-knowledge proof для аутентификации без пароля

Обучающиеся должны изучить интернет-ресурсы и литературу по вопросам, представленным в табл. 3.

Список литературы

Основная литература

1. Федорова, Г. Н. Разработка, внедрение и адаптация программного обеспечения отраслевой направленности : учебное пособие / Г. Н. Федорова. – Москва : КУРС : ИНФРА-М, 2026. – 336 с. – (Среднее профессиональное образование). – ISBN 978-5-906818-41-6. – Текст : электронный. – URL: <https://znanium.ru/catalog/product/2213137>

2. Казарин, О. В. Основы информационной безопасности: надежность и безопасность программного обеспечения : учебник для СПО / О. В. Казарин, И. Б. Шубинский. – 2-е изд. – Москва : Юрайт, 2025. – 352 с. – ISBN 978-5-534-19384-8. – URL: <https://urait.ru/book/osnovy-informacionnoybezopasnosti-nadezhnost-i-bezopasnost-programmnogo-obespecheniya-580668>.

Дополнительная литература

1. Семакин, И. Е. Основы алгоритмизации и программирования : учебник для студентов среднего профессионального образования / И. Е. Семакин, А. П. Шестаков ; И. Е. Семакин, А. П. Шестаков. – 6-е изд., стер. – Москва : Академия, 2024. – 304 с. – (Профессиональное образование). – URL: <https://academia-moscow.ru/reader/?id=768351>

2. Гвоздева, В. А. Информатика, автоматизированные информационные технологии и системы : учебник / В.А. Гвоздева. – Москва : ФОРУМ : ИНФРА-М, 2025. – 542 с. – (Среднее профессиональное образование). – ISBN 978-5-8199-0856-3. – Текст : электронный. – URL: <https://znanium.ru/catalog/product/2169724>

3. Гаврилов, М. В. Информатика и информационные технологии: учебник для СПО / М. В. Гаврилов, В. А. Климов. – 6-е изд., пер. и доп. – Москва : Юрайт, 2025. – 319 с. – ISBN 978-5-534-20333-2. – URL: <https://urait.ru/book/informatika-i-informacionnyetehnologii-560669> (дата обращения: 01.12.2025). – Текст : электронный.

4. Голицына, О. Л. Информационные системы и технологии : учебное пособие / О. Л. Голицына, Н. В. Максимов, И. И. Попов. – Москва : ФОРУМ : ИНФРА-М, 2023. – 400 с. – (Среднее профессиональное образование). – ISBN 978-5-00091-592-9. – Текст : элек-

тронный. – URL: <https://znanium.ru/catalog/product/2013719> (дата обращения: 27.03.2026). – Режим доступа: по подписке.

5. Спирина, М. С. Дискретная математика : сборник задач с алгоритмами решений : учебное пособие для студентов, обучающихся по профессиям и специальностям среднего профессионального образования / М. С. Спирина, П. А. Спирин. – 5-е изд., стер. – Москва : Академия, 2024. – 288 с. – URL: <https://academia-moscow.ru/reader/?id=796464#>

6. Щербак, А. В. Информационная безопасность : учебник для СПО / А. В. Щербак. – 2-е изд., пер. и доп. – Москва : Юрайт, 2025. – 252 с. – ISBN 978-5-534-20154-3. – URL: <https://urait.ru/book/informacionnaya-bezopasnost-567521>

7. Сычев, Ю. Н. Защита информации и информационная безопасность : учебное пособие / Ю. Н. Сычев. – Москва : ИНФРА-М, 2021. – 201 с. – (Среднее профессиональное образование). – ISBN 978-5-16-016583-7. – Текст : электронный. – URL: <https://znanium.ru/catalog/product/1191479>

СОДЕРЖАНИЕ

ЛАБОРАТОРНАЯ РАБОТА №1 АНАЛИЗ КОДА НА НАЛИЧИЕ УЯЗВИМОСТЕЙ	3
ЛАБОРАТОРНАЯ РАБОТА №2 SQL ИНЪЕКЦИИ – ЭКСПЛУАТАЦИЯ И ЗАЩИТА УЯЗВИМОГО ПРИЛОЖЕНИЯ. XSS АТАКИ – СОЗДАНИЕ И ПРЕДОТВРАЩЕНИЕ МЕЖСАЙТОВОГО СКРИПТИНГА	7
ЛАБОРАТОРНАЯ РАБОТА №3 СОСТАВЛЕНИЕ МОДЕЛИ УГРОЗ ДЛЯ ТИПОВОГО ВЕБ-ПРИЛОЖЕНИЯ	11
ЛАБОРАТОРНАЯ РАБОТА №4 ХЭШИРОВАНИЕ ПАРОЛЕЙ С SALT И ADAPTIVE FUNCTIONS	16
ЛАБОРАТОРНАЯ РАБОТА №5 АНАЛИЗ СЕТЕВОГО ТРАФИКА С ПОМОЩЬЮ WIRESHARK. СКАНИРОВАНИЕ УЯЗВИМОСТЕЙ OWASP ZAP И BURP SUITE	18
ЛАБОРАТОРНАЯ РАБОТА №6 НАСТРОЙКА HTTPS И СОЗДАНИЕ САМОПОДПИСАННЫХ СЕРТИФИКАТОВ	29
ЛАБОРАТОРНАЯ РАБОТА №7 ЗАЩИТА ОТ BRUTE-FORCE АТАК С ОГРАНИЧЕНИЕМ ПОПЫТОК ВХОДА	34
ЛАБОРАТОРНАЯ РАБОТА №8 НАСТРОЙКА TLS	40
ЛАБОРАТОРНАЯ РАБОТА №9 СОЗДАНИЕ BLOCKCHAIN SMART CONTRACT С ЗАЩИТОЙ ОТ REENTRANCY ATTACKS	43
СОДЕРЖАНИЕ САМОСТОЯТЕЛЬНОЙ РАБОТЫ	55
СПИСОК ЛИТЕРАТУРЫ	56