# Требования к отчёту

1) Титульный лист:
    a. Название университета:
        i. Должно быть действующее, проверяется тут: https://itmo.ru/sveden/common/
        ii. Или просто взять такое: Федеральное государственное автономное образовательное учреждение высшего образования «Национальный исследовательский университет ИТМО»
    b. Название дисциплины «Информационная безопасность»
    c. Работа X: название работы
    d. ФИО полностью
    e. Номер группы
    f. Год
2) Содержание отчёта берёте из раздела работы «Что сдают студенты»
3) Формат: PDF
4) Отчёт предоставляется в электронном виде:
    a. адрес markina_t@itmo.ru
    b. тема: Фамилия И.О., номер группы, номер работы

# Работа 1: Разработка защищенного REST API с интеграцией в CI/CD

**Назначение:** получить практический опыт разработки безопасного backend-приложения с автоматизированной проверкой кода на уязвимости. Освоить принципы защиты от OWASP Top 10 и интеграцию инструментов безопасности в процесс разработки.

## Задание

Выполните следующие шаги для создания простого, но защищенного веб-API:

1. **Выбор стека и инициализация проекта:**
    - Выберите язык программирования (Python/Flask, JavaScript/Node.js + Express, Java/Spring Boot).
    - Создайте новый проект с помощью менеджера пакетов (npm, pip, maven).
    - Инициализируйте git-репозиторий и свяжите его с удаленным репозиторием на GitHub или GitLab.
2. **Разработка функционального API:**
    - Реализуйте 3 метода:
        1) POST /auth/login — метод для аутентификации пользователя (принимает логин и пароль).
        2) GET /api/data — метод для получения каких-либо данных (например, список пользователей или постов). Доступ должен быть только у аутентифицированных пользователей.
        3) Придумайте сами
    - Для хранения данных используйте любую удобную базу данных (SQLite для простоты или PostgreSQL) или даже простой файл/массив в памяти, если это учебный пример.
3. **Внедрение базовых мер защиты:**
    - **Защита от SQLi (SQL-инъекций):** Используйте параметризованные запросы (Prepared Statements) или ORM (например, Sequelize, SQLAlchemy, Hibernate). Не используйте конкатенацию строк для формирования SQL-запросов.
    - **Защита от XSS:** Санитизируйте (экранируйте) все пользовательские данные, которые возвращаются в ответах API. Используйте встроенные функции фреймворка (например, escape() в Express).
    - **Защита от Broken Authentication:**
        - Реализуйте выдачу JWT-токена upon successful login.

        - Напишите middleware, которое будет проверять JWT-токен на всех защищенных эндпоинтах.
        - Пароли ни в коем случае не должны храниться в открытом виде. Обязательно хэшируйте их с помощью алгоритмов типа bcrypt, scrypt или Argon2.
4. **Настройка CI/CD pipeline с security-сканерами:**
    - В корне проекта создайте конфигурационный файл для GitHub Actions (.github/workflows/ci.yml) или GitLab CI (.gitlab-ci.yml).
    - **SAST (Static Application Security Testing):** Настройте в pipeline запуск статического анализатора кода.
        - *Для Python:* bandit или safety
        - *Для JavaScript:* npm audit или snyk test
        - *Для Java:* spotbugs или использование плагина OWASP Dependency-Check
    - **SCA (Software Composition Analysis):** Настройте проверку зависимостей на известные уязвимости.
        - Используйте OWASP Dependency-Check или Snyk.
        - Настройте step в pipeline, который будет запускать сканер и выдавать отчет.
    - При каждом push в репозиторий или создании pull request должны автоматически запускаться проверки.
5. **Тестирование и документирование:**
    - Протестируйте работу API с помощью Postman или curl. Убедитесь, что аутентификация работает, а доступ без токена — запрещен.
    - Сделайте скриншоты отчетов от SAST/SCA-инструментов, которые показывают, что проверки прошли успешно (не найдено критических уязвимостей) или что найденные уязвимости были вами исправлены.

## Что сдают студенты

1. Ссылка на **публичный git-репозиторий** (GitHub/GitLab) с кодом проекта.
2. Файл README.md в репозитории, содержащий:
    - Описание проекта и API (какие эндпоинты есть, как их вызывать).
    - **Подробное описание реализованных мер защиты** (как именно защищены от SQLi, XSS, как реализована аутентификация).
    - Скриншоты отчетов SAST/SCA из раздела "Actions" / "CI/CD" вашего репозитория.
3. Ссылка на последний успешный запуск pipeline в вашем репозитории.

## Критерии оценки (макс. 10 баллов)

- **3 балла:** Работоспособность API. Эндпоинты аутентификации и получения данных возвращают корректные ответы.
- **4 балла:** Реализация мер защиты:
    - 1 балл за защиту от SQLi.
    - 1 балл за защиту от XSS.
    - 2 балла за корректную реализацию аутентификации (JWT + хэширование паролей).
- **2 балла:** Настройка CI/CD pipeline с запуском как минимум одного SAST/SCA инструмента.
- **1 балл:** Качество оформления отчета (четкость, структурированность, наличие скриншотов), качество документации и оформления кода.

## Источники для теории

1. **OWASP Top 10:** https://owasp.org/www-project-top-ten/ (Обратите внимание на A01:2021-Broken Access Control, A03:2021-Injection, A07:2021-Identification and Authentication Failures).
2. **OWASP Cheat Sheet Series:** https://cheatsheetseries.owasp.org/ (В частности, cheat sheets по Authentication, SQL Injection Prevention, XSS Prevention).
3. **Документация по JWT (JSON Web Tokens):** https://jwt.io/introduction
4. **Документация по GitHub Actions:** https://docs.github.com/en/actions или **GitLab CI:** https://docs.gitlab.com/ee/ci/

## Контрольные вопросы

1. Почему хэширование пароля с помощью алгоритмов вроде bcrypt предпочтительнее использования SHA-256?
2. В чем заключается основная разница между SAST и DAST-подходом к тестированию безопасности?
3. Опишите механизм работы JWT-токена. Какая информация в нем содержится и как сервер проверяет его подлинность?
4. Какие риски возникают, если не проводить аудит сторонних библиотек (зависимостей) в проекте?

# Работа 2. Анализ и устранение уязвимости на примере реального CVE с использованием Vulhub

**Назначение:** приобрести практический опыт работы с уязвимым программным обеспечением в контролируемой среде. Научиться воспроизводить эксплуатацию известной уязвимости (CVE), анализировать ее причины и реализовывать меры по ее устранению.

## Задание

Выполните следующие шаги для анализа и устранения конкретной уязвимости из коллекции Vulhub:

1. **Выбор и подготовка лабораторного окружения:**
    - Убедитесь, что на вашем компьютере установлены Docker и Docker Compose.
    - Клонируйте репозиторий Vulhub: git clone https://github.com/vulhub/vulhub.git
    - Перейдите в каталог с интересующей вас уязвимостью (например, cd vulhub/nginx/CVE-2021-23017). Выбор уязвимости: рекомендуется начать с чего-то не слишком сложного, например, уязвимость в компоненте web-приложения (например, vulhub/flask/CVE-2018-1000656) или в популярном сервисе.
    - Внимательно изучите файл README.md в выбранном каталоге. В нем содержится описание уязвимости, версия уязвимого ПО, инструкции по запуску и часто — пример эксплуатации.
2. **Запуск уязвимого окружения и воспроизведение атаки:**
    - Запустите уязвимый сервис командой docker-compose up -d.
    - Дождитесь полного запуска контейнеров. Проверьте, что сервис доступен (обычно по http://localhost:8080 или другому порту, указанному в инструкции).
    - Внимательно следуя инструкциям в README.md, воспроизведите шаги по эксплуатации уязвимости. Ваша цель — добиться ожидаемого результата (например, получения несанкционированного доступа, чтения чужых файлов, выполнения кода).
    - **Важно:** Фиксируйте все свои действия (команды, HTTP-запросы через curl или Burp Suite) для включения в отчет.
3. **Анализ root cause уязвимости:**
    - Изучите описание CVE на сайте https://cve.mitre.org/ или NVD.
    - Проанализируйте, в чем заключается ошибка, приведшая к уязвимости. Это ошибка логики? Неправильная обработка ввода? Проблема в конфигурации?
    - Изучите файлы в каталоге vulhub, чтобы понять, как сконфигурировано уязвимое окружение.
    - Если возможно, просмотрите исходный код уязвимого компонента (часто он уже находится в каталоге в виде src/ или указана ссылка на коммит с фиксом).

4. **Разработка и применение мер защиты:**
    - На основе анализа предложите способ устранения уязвимости. Это может быть:
        - **Изменение конфигурации** (если уязвимость вызвана небезопасными настройками по умолчанию).
        - **Обновление версии ПО** в файле docker-compose.yml на ту, где уязвимость исправлена.
        - **Внесение правок в код** (если это учебное приложение и уязвимость в его коде). Например, добавление валидации пользовательского ввода, экранирование данных.
    - Остановите текущие контейнеры (docker-compose down).
    - Примените ваше исправление: измените Dockerfile, docker-compose.yml или исходный код приложения.
    - Пересоберите и запустите исправленное окружение: docker-compose up --build -d.
5. **Верификация исправления:**
    - Повторите те же шаги по эксплуатации уязвимости, которые вы выполняли на шаге 2.
    - Убедитесь, что атака теперь **не проходит**. Ваше исправленное приложение должно отклонять malicious-запросы, возвращать ошибки или вести себя ожидаемым безопасным образом.
    - Протестируйте, что основная функциональность приложения после ваших правок не сломалась.

## Что сдают студенты

1. **Подробный отчет**, содержащий:
    - **Название выбранной уязвимости (CVE ID)** и краткое ее описание.
    - **Последовательность действий** по воспроизведению уязвимости (с скриншотами и командами).
    - **Анализ root cause:** Объяснение причины уязвимости своими словами.
    - **Описание примененного исправления:** что именно было изменено и почему этот метод эффективен.
    - **Доказательство устранения уязвимости:** Скриншоты, демонстрирующие, что атака на исправленную версию больше не работает.
2. **Папка с вашими исправлениями:** если вносились изменения в конфигурационные файлы (docker-compose.yml, Dockerfile) или исходный код, необходимо приложить эти файлы.

## Критерии оценки (макс. 10 баллов)

- **3 балла:** Успешный запуск окружения и корректное воспроизведение уязвимости.
- **3 балла:** Глубина анализа root cause уязвимости (понимание, где и почему произошла ошибка).
- **3 балла:** Корректность и эффективность примененного исправления, верификация его работы.
- **1 балл:** Качество оформления отчета (четкость, структурированность, наличие скриншотов).

## Источники для теории

1. **Официальный репозиторий Vulhub:** https://github.com/vulhub/vulhub — основная база знаний для этой работы.
2. **National Vulnerability Database (NVD):** https://nvd.nist.gov/ — для поиска подробных описаний CVE.
3. **OWASP Top 10:** https://owasp.org/www-project-top-ten/ — чтобы классифицировать найденную уязвимость по категориям (Инъекции, Небезопасные десериализации и т.д.).
4. **Docker Documentation:** https://docs.docker.com/ — для работы с контейнерами.

## Контрольные вопросы

1. Какие преимущества дает использование Docker и Vulhub для изучения кибербезопасности по сравнению с установкой уязвимого ПО прямо на свою машину?
2. Опишите разницу между уязвимостью (vulnerability) и эксплойтом (exploit). Что вы воспроизводили в этой работе?
3. Почему после применения исправления (патча) критически важно проверить, что основная функциональность приложения не пострадала (регрессионное тестирование)?
4. Как знание и умение воспроизводить известные уязвимости (CVE) помогает в работе специалиста по безопасности (например, пентестера или SOC-аналитика)?

# Работа 3: Аудит безопасности веб-приложения

**Назначение:** освоить методику комплексного анализа защищенности веб-приложения, сочетая автоматизированное сканирование (DAST) и проактивное моделирование угроз (Threat Modeling). Получить навыки документирования результатов аудита в виде профессионального отчета.

## Задание

Выполните полный цикл аудита безопасности для тестового приложения OWASP Juice Shop.

1. **Подготовка тестового стенда:**
    - Установите и запустите OWASP Juice Shop. Самый простой способ — через Docker: docker run --rm -p 3000:3000 juicyshop/juice-shop.
    - Убедитесь, что приложение доступно по адресу http://localhost:3000.
    - Установите и настройте OWASP ZAP (Zed Attack Proxy). Рекомендуется использовать автономную версию (Standalone).
2. **Автоматизированное сканирование (DAST):**
    - **Настройка ZAP:**
        - Запустите ZAP.
        - В поле "URL to attack" укажите адрес Juice Shop (http://localhost:3000).
        - Нажмите "Attack". ZAP начнет автоматическое сканирование (Ajax Spider и Active Scan).
    - **Анализ результатов:**
        - После завершения сканирования перейдите на вкладку "Alerts". Отсортируйте уязвимости по степени риска (High, Medium, Low).
        - Проведите верификацию найденных уязвимостей. Для этого найдите соответствующую уязвимость в Juice Shop и убедитесь, что ее можно эксплуатировать (например, реально украсть cookie через XSS или получить несанкционированный доступ к API).
        - Сфокусируйтесь на нахождении и подтверждении как минимум 5 уязвимостей, среди которых должны быть **SQL Injection** (SQLi) и **Cross-Site Scripting** (XSS).
    - **Дополнительное исследование:** Используйте встроенный в Juice Shop "Score Board" для поиска дополнительных уязвимостей, которые ZAP мог пропустить.
3. **Моделирование угроз (Threat Modeling) с помощью STRIDE:**
    - **Построение диаграммы потока данных (Data Flow Diagram — DFD):**
        - Упрощенно визуализируйте ключевые компоненты Juice Shop: браузер пользователя, веб-сервер, базу данных.

        - Отобразите на диаграмме основные потоки данных: аутентификация пользователя, поиск товаров, отправка отзывов, оформление заказов.
    - **Анализ угроз по методике STRIDE:**
        - **Spoofing (Маскировка):** можно ли impersonate другого пользователя? (например, подменить сессию или JWT-токен).
        - **Tampering (Изменение данных):** можно ли изменить цену товара, отзывы или данные профиля?
        - **Repudiation (Отказ от операций):** можно ли отрицать совершение действия (например, покупки)? Есть ли логи?
        - **Information Disclosure (Раскрытие информации):** можно ли получить доступ к данным других пользователей, API-ключам, исходному коду?
        - **Denial of Service (Отказ в обслуживании):** можно ли "положить" приложение одной HTTP-посылкой?
        - **Elevation of Privilege (Повышение привилегий):** можно ли из роли обычного пользователя получить права администратора?
    - **Пример:** на потоке данных "Аутентификация" угроза *Spoofing* может быть реализована через кражу сессионного токена (найденную вами уязвимость XSS).
4. **Подготовка финального отчета:**
    - Создайте структурированный документ.
    - **Таблица уязвимостей:** для каждой найденной уязвимости (как через ZAP, так и через Threat Modeling) заполните таблицу со столбцами:
        - **Название** (например, "Reflected XSS в поисковом запросе")
        - **Описание** (краткое описание и шаги воспроизведения)
        - **Уровень риска (CVSS)** (воспользуйтесь онлайн-калькулятором CVSS для оценки, например, от First.org)
        - **Категория OWASP Top 10** (например, A03:2021-Injection)
        - **Предложение по исправлению** (конкретная рекомендация: "Валидировать ввод на стороне сервера", "Экранировать вывод" и т.д.)
    - **Рекомендации по устранению рисков:** на основе анализа угроз дайте 3-5 общих рекомендаций по усилению безопасности приложения (например, "Внедрить строгую политику CSP", "Регулярно обновлять зависимости").

## Что сдают студенты

1. **Отчет об аудите безопасности** в формате PDF, содержащий:
    - Краткое резюме.
    - Диаграмму потока данных (DFD) с разметкой угроз по STRIDE.

    - Таблицу с найденными уязвимостями.
    - Раздел с рекомендациями по исправлению.
2. **Скриншоты**, подтверждающие найденные уязвимости (например, всплывающее окно при XSS или результат успешной SQL-инъекции).
3. **Экспорт отчета из OWASP ZAP** в формате HTML или PDF.

## Критерии оценки (макс. 10 баллов)

- **3 балла:** Качество DAST-сканирования (найдено и верифицировано не менее 5 уязвимостей, включая SQLi/XSS).
- **3 балла:** Глубина и качество моделирования угроз (логичная DFD, корректное применение STRIDE к элементам диаграммы).
- **3 балла:** Качество предложенных исправлений (релевантность, конкретность, соответствие лучшим практикам).
- **1 балл:** Качество оформления отчета (четкость, структурированность, наличие скриншотов).

## Источники для теории

1. **Официальный сайт OWASP Juice Shop:** https://owasp.org/www-project-juice-shop/ — учебный полигон.
2. **OWASP ZAP Getting Started:** https://www.zaproxy.org/getting-started/ — руководство по основным функциям сканера.
3. **Методика STRIDE:** https://learn.microsoft.com/ru-ru/azure/security/develop/threat-modeling-tool-threats — описание категорий угроз.
4. **Руководство по расчету CVSS:** https://www.first.org/cvss/v4.0/user-guide — понимание количественной оценки серьезности уязвимостей.

## Контрольные вопросы

1. Какие типы уязвимостей чаще всего обнаруживаются автоматическими сканерами (DAST), а какие обычно остаются незамеченными и требуют ручного тестирования?
2. Объясните на примере Juice Shop, как обнаруженная вами уязвимость (например, XSS) может быть использована для реализации угрозы из модели STRIDE (например, Spoofing)?
3. Почему при оценке уязвимости по CVSS важно учитывать не только базовые (Base), но и временные (Temporal) и environmental (Environmental) метрики?
4. В чем заключается основная ценность моделирования угроз (Threat Modeling) по сравнению с проведением только автоматизированного сканирования?

# Работа 4 Анализ уязвимостей веб-приложения с помощью OWASP ZAP

**Назначение:** освоить базовые навыки динамического тестирования безопасности (DAST) на примере тестового приложения.

## Задание

1. Установите OWASP ZAP (бесплатный инструмент).
2. Запустите встроенный браузер ZAP и перейдите на тестовый сайт (например, http://testphp.vulnweb.com/).
3. Проведите "Быстрое сканирование" (Quick Scan) сайта.
4. Проанализируйте результаты сканирования: найдите 3-5 различных типов уязвимостей (например, XSS, SQLi).
5. Сделайте скриншоты найденных уязвимостей и кратко опишите суть каждой.

## Что сдают студенты

Краткий отчет в (2-3 стр.) со скриншотами и описанием уязвимостей.

## Критерии оценки (макс. 10 баллов)

- **3 балла:** Корректное выполнение сканирования.
- **5 баллов:** Найдено и корректно классифицировано не менее 3 уязвимостей.
- **2 балла:** Качество оформления отчета (четкость, структурированность, наличие скриншотов).

## Источники для теории

- Официальная документация OWASP ZAP: https://www.zaproxy.org/docs/
- OWASP Top 10: Краткое описание уязвимостей: https://owasp.org/www-project-top-ten/
- https://owasp.org/www-chapter-dorset/assets/presentations/2020-01/20200120-OWASPDorset-ZAP-DanielW.pdf

## Контрольные вопросы

1. Чем динамическое тестирование (DAST) отличается от статического (SAST)?
2. Какие последствия может иметь уязвимость Cross-Site Scripting (XSS) для пользователя сайта?
3. Почему для тестирования нельзя использовать реальные коммерческие сайты?

# Работа 5: Аудит паролей с помощью менеджера паролей

**Назначение:** осознать важность использования надежных уникальных паролей и освоить инструмент для их хранения.

## Задание

1. Установите бесплатный менеджер паролей (например, Bitwarden).
2. Создайте новую учетную запись и мастер-пароль высокой надежности.
3. Проанализируйте пароли, которые вы используете в своих аккаунтах (например, в соцсетях, почте). Оцените их слабость (проверьте, не были ли они скомпрометированы, через сервис https://haveibeenpwned.com/ — используйте только email, **не пароль!**).
4. Для 5 своих аккаунтов создайте с помощью менеджера новые надежные пароли (не менее 12 символов, буквы разного регистра, цифры, спец. символы) и сохраните их в менеджере.
5. Настройте генерацию одноразовых кодов (TOTP) для 2FA для одного из аккаунтов (если он поддерживается).

## Что сдают студенты

Скриншоты, снятые при выполнении задания, а также скриншот главного интерфейса менеджера паролей с созданной базой (с замазанными конфиденциальными данными) и краткие ответы на вопросы: какие пароли были слабыми, что такое 2FA.

## Критерии оценки (макс. 5 баллов)

- **1 балл:** Создание и настройка менеджера паролей.
- **2 балла:** Аудит и замена паролей на надежные для 5 аккаунтов.
- **1 балл:** Настройка 2FA.
- **1 балл:** Качество оформления отчета (четкость, структурированность, наличие скриншотов).

## Источники для теории

- Статья "Как придумать надежный пароль": https://www.kaspersky.ru/blog/perfect-password/
- Объяснение двухфакторной аутентификации (2FA): https://blog.cloudflare.com/ru-ru/two-factor-authentication-2fa-ru-ru/

## Контрольные вопросы

1. Почему использование одного пароля на нескольких сайтах опасно?
2. Как двухфакторная аутентификация увеличивает безопасность аккаунта даже при утечке пароля?
3. Почему мастер-пароль от менеджера должен быть особенно надежным?

# Работа 6: Криптография на практике: шифрование файлов и сообщений

**Назначение:** получить практический опыт использования симметричного и асимметричного шифрования.

## Задание

1. **Симметричное шифрование:** установите программу VeraCrypt. Создайте небольшой зашифрованный контейнер (файл), защищенный паролем. Смонтируйте его как виртуальный диск и скопируйте в него несколько файлов.
2. **Асимметричное шифрование:** установите почтовый клиент Thunderbird и дополнение "Enigmail" или используйте онлайн-генератор PGP. Сгенерируйте свою пару ключей (открытый и закрытый). Экспортируйте свой открытый ключ в файл.
3. (Опционально) Обменяйтесь открытыми ключами с одногруппником и отправьте друг другу зашифрованное текстовое сообщение.

## Что сдают студенты

- Скриншот интерфейса VeraCrypt с смонтированным томом.
- Файл с экспортированным открытым PGP-ключом.
- Краткое описание разницы между симметричным и асимметричным шифрованием.

## Критерии оценки (макс. 5 баллов)

- **3 балла:** Создание и использование зашифрованного контейнера в VeraCrypt.
- **2 балла:** Генерация пары PGP-ключей.

## Источники для теории

- Официальное руководство для начинающих по VeraCrypt: https://www.veracrypt.fr/en/Beginner%27s%20Tutorial.html
- Простое объяснение PGP: https://proton.me/blog/what-is-pgp

## Контрольные вопросы

1. В чем основное различие между симметричным и асимметричным шифрованием?
2. Как решается проблема безопасной передачи ключа в асимметричной криптографии?
3. Для каких реальных задач применяется шифрование (приведите 3 примера)?

# Работа 7: Безопасность браузера и анализ сетевого трафика

**Назначение:** изучить настройки безопасности браузера и понять, какие данные передаются между браузером и сайтом.

## Задание

1. **Настройка браузера:** тщательно изучите настройки безопасности и конфиденциальности вашего браузера (например, Chrome или Firefox). Включите расширенную защиту от отслеживания, отключите сохранение паролей в браузере, почистите куки и кэш.
2. **Анализ трафика:** Откройте в браузере "Инструменты разработчика" (F12), перейдите на вкладку "Сеть" (Network). Откройте любой сайт. Проанализируйте список HTTP-запросов: найдите запросы типа GET/POST, посмотрите, какие данные передаются в заголовках (headers).
3. Найдите на каком-нибудь сайте форму входа и попробуйте ввести тестовые данные. Посмотрите в инструментах разработчика, как передается логин и пароль (в открытом виде или зашифровано?).

## Что сдают студенты

Краткий отчет с скриншотами:

- Скриншот настроек безопасности браузера.
- Скриншот вкладки "Сеть" с выделенными запросами.
- Ответ на вопрос: "Как чаще всего передаются данные аутентификации (логин/пароль) в современных веб-приложениях?".

## Критерии оценки (макс. 5 баллов)

- **2 балла:** Анализ и настройка параметров безопасности браузера.
- **2 балла:** Проведение анализа сетевого трафика, формулирование вывода о передаче данных.
- **1 балл:** Качество оформления отчета (четкость, структурированность, наличие скриншотов).

## Источники для теории

- Статья "Как настроить браузер для максимальной безопасности": https://www.cnews.ru/articles/2023-05-15_kak_nastrat_brauzer_dlya
- Объяснение HTTP-методов GET/POST: https://developer.mozilla.org/ru/docs/Web/HTTP/Methods

## Контрольные вопросы

1. Что такое cookies и какую угрозу конфиденциальности они могут нести?

2. Почему важно, чтобы форма входа на сайт передавала данные по защищенному HTTPS-соединению?
3. Чем опасна функция "Сохранить пароль" в браузере?

# Работа 8: Мини-исследование: Утечка данных и цифровая гигиена

**Назначение:** научиться проверять наличие своих данных в утечках и формировать правила безопасного поведения в сети.

## Задание

1. Проверьте свой основной email-адрес на сайте https://haveibeenpwned.com/ (сервис проверяет, не фигурирует ли ваша почта в известных утечках данных). **Внимание: Используйте только email! Не вводите здесь свои пароли!**
2. Если ваши данные были найдены, проанализируйте, в каких утечках они фигурируют и какие данные были скомпрометированы (пароли, имена, номера телефонов и т.д.).
3. Проведите аудит своих публичных данных в социальных сетях (ВК, Telegram и др.). Какая информация о вас доступна незнакомым людям?
4. На основе проведенного анализа составьте 5-7 личных правил цифровой гигиены (например, "не использовать один пароль на разных сайтов", "проверять настройки конфиденциальности в соцсетях раз в полгода").

## Что сдают студенты

Краткий отчет (1-2 страницы) с результатами проверки, анализом своих цифровых следов и списком личных правил цифровой гигиены.

## Критерии оценки (макс. 5 баллов)

- **1 балл:** Проверка данных на наличие в утечках.
- **1 балл:** Аудит публичности в социальных сетях.
- **2 балла:** Качество составленных правил (осмысленность, практическая применимость).
- **1 балл:** Качество оформления отчета (четкость, структурированность, наличие скриншотов).

## Источники для теории

- Статья "Что делать, если ваши данные оказались в утечке": https://www.kaspersky.ru/blog/leak-check/
- Гайд "Цифровая гигиена: как защитить свои данные в интернете": https://rb.ru/young/digital-hygiene/

## Контрольные вопросы

1. Почему даже старые утечки данных могут быть опасны сегодня?
2. Как информация из вашего профиля в соцсети может быть использована для фишинга или взлома ваших аккаунтов?
3. Какие самые распространенные ошибки цифровой гигиены совершают пользователи?