Лабораторная работа – уязвимости XSS

В данной лабораторной работе вам предлагается попробовать свои силы в решении задач, схожих с реальными случаями проведения тестирования на проникновение.

Ваша задача – обнаружение и эксплуатация уязвимостей типа межсайтового скриптинга (Cross-Site Scripting, XSS).

Вам необходимо сделать **любое количество** из предложенных заданий для получения балла.

ВНИМАНИЕ: максимальный балл за эту работу — 15 (если вы выполните заданий больше, чем требуется, максимальная оценка все равно будет 15). Вы можете выбрать любые из предложенных заданий для получения максимальной оценки. Максимальный балл за каждое задание приведен рядом с заданием.

Часть 1 – Необходимые инструменты

1.1. Обязательные требования:

- Для выполнения **всех** заданий вам потребуется инструмент Burp Suite Community Edition — это интегрированная платформа для тестирования безопасности веб-приложений. Скачать Burp Suite Community Edition можно по ссылке:
  https://portswigger.net/burp/communitydownload
  Для загрузки потребуется ввести вашу почту.
- Для выполнения Заданий вам потребуется зарегистрироваться на сайте PortSwigger.
- Для анализа отражения данных, структуры DOM и выполнения JavaScript в ходе лабораторной работы вам необходимо активно использовать инструменты разработчика (DevTools) вашего браузера, в частности панели Elements (Инспектор), Console (Консоль) и Debugger (Отладчик)

1.2. Важные замечания и рекомендации:

- При выполнении заданий на DOM-based XSS помните, что уязвимость эксплуатируется в браузере жертвы. Для проверки работоспособности payload вы должны открывать модифицированные URL в браузере (или во встроенном браузере Burp Suite), а не только анализировать raw-ответ сервера
- Перед выполнением заданий, особенно Задания 9, рекомендуется ознакомиться с актуальным "XSS Cheat Sheet" от PortSwigger: https://portswigger.net/web-security/cross-site-scripting/cheat-sheet
- Для выполнения заданий, связанных с подбором тегов и атрибутов, рекомендуется использовать встроенный в Burp Suite инструмент "XSS Cheat Sheet" (можно установить соответствующее расширение из bApp Store)

Часть 2 - Требования к сдаче заданий и к оформлению отчета

Обязательные требования:

- В решении требуется описать способ атаки или обхода защитных механизмов и приложить доказательства выполнения (скриншоты/логи).
- Все шаги выполнения атаки должны быть подтверждены соответствующими скриншотами и текстовыми комментариями к ним.
- На **каждом** скриншоте должна быть информация о выполнившем работу студенте в формате **Фамилия_Имя_Группа** (Пример: Иванов_Иван_БИБ221).
- На каждом скриншоте, где это применимо, должны быть видны контекстные метаданные, подтверждающие подлинность. **Предпочтительный и рекомендуемый способ:** предоставлять скриншоты интерфейса Burp Suite (например, вкладки Proxy>HTTP history, Repeater, Intruder), на которых четко видна временная метка (Time), автоматически проставляемая Burp для каждого запроса. В качестве альтернативы допускается отображение панели задач операционной системы с датой и временем.
- Из отчета должна быть понятна последовательность ваших рассуждений при проведении атак.
- Используйте шаблон отчета, который прикреплен к заданию.

Требования к разнообразию атак для получения максимального балла

Для того чтобы продемонстрировать глубокое понимание различных векторов XSS, **для получения максимального балла (15) необходимо выполнить атаки, охватывающие разные аспекты уязвимости.** Атаки считаются различными и засчитываются отдельно, если они отличаются по **одному или нескольким** из следующих критериев:

1. **Тип атаки:** Reflected XSS, Stored XSS, DOM-based XSS.
2. **Контекст встраивания/место попадания данных (sink):**
    - Обычный HTML (между тегами)
    - Внутри значения атрибута HTML-тега (например, href, src, onclick)
    - Внутри строки JavaScript (<script>let a = 'ВАШИ_ДАННЫЕ';</script>)
    - Внутрь опасной функции/свойства DOM (document.write, innerHTML, $())
3. **Механизм/источник данных (source):** Параметр URL (location.search, location.hash), данные формы (тело POST-запроса), заголовки HTTP.

Пример: Выполнение двух лабораторных работ на Reflected XSS, но в одной данные попадают в атрибут, а в другой — внутрь тега <script>, считается выполнением **двух разных атак**.

Ваше итоговое решение (набор выбранных заданий) должно демонстрировать понимание этого разнообразия.

Часть 3 – Ссылки на задания

Внимание. Все упражнения в лабораторной работе должны выполняться только на специально предоставленных лабораториях (PortSwigger Labs) или других выделенных тестовых средах. Любые попытки применять описанные техники против сторонних ресурсов, не принадлежащих вам или вашей организации, являются незаконными и неприемлемыми. Выполнение заданий вне изолированной лаборатории недопустимо.

- **Задание 1 –PortSwigger — DOM XSS в функции document.write с использованием источника location.search (уровень – Apprentice) (2,5 балла)**

Лабораторная работа демонстрирует классическую DOM-базированную уязвимость межсайтового скриптинга (DOM XSS). Уязвимость возникает, когда клиентский скрипт небезопасно использует управляемые пользователем данные (в данном случае из параметра URL) и передает их в опасную функцию document.write. Это позволяет злоумышленнику внедрить и выполнить произвольный JavaScript-код в контексте страницы жертвы.

Постановка задачи: Найдите уязвимую функцию поиска на сайте. Внедрите и выполните вредоносный скрипт, который вызовет функцию alert(). Для этого сконструируйте специальный URL, который, будучи открыт, приведет к исполнению кода alert() и решит лабораторную работу.

https://portswigger.net/web-security/cross-site-scripting/dom-based/lab-document-write-sink

- **Задание 2 – PortSwigger — DOM XSS в свойстве innerHTML с использованием источника location.search (уровень – Apprentice) (2,5 балла)**

Лабораторная работа демонстрирует DOM-базированную уязвимость межсайтового скриптинга (DOM XSS), возникающую при небезопасной динамической модификации DOM-дерева. Уязвимость заключается в присваивании управляемых пользователем данных (из параметра URL location.search) напрямую свойству элемента innerHTML. Это свойство интерпретирует строку как HTML-код, что позволяет злоумышленнику внедрить и исполнить произвольный JavaScript.

Постановка задачи: Найдите на сайте функцию поиска в блоге, которая динамически обновляет содержимое страницы. Внедрите вредоносный HTML/JS-код, который при отображении вызовет функцию alert(). Для этого сконструируйте поисковый запрос, приводящий к XSS и решению лабораторной работы.

https://portswigger.net/web-security/cross-site-scripting/dom-based/lab-innerhtml-sink

- **Задание 3 – PortSwigger — DOM XSS в jQuery-селекторе $() с использованием события hashchange (уровень – Apprentice) (3 балла)**

  Лабораторная работа демонстрирует сложный сценарий DOM-базированного межсайтового скриптинга (DOM XSS), использующий особенности клиентских библиотек и событий браузера. Уязвимость возникает из-за небезопасной обработки фрагмента URL (хеша, location.hash) jQuery-функцией $(), которая используется в обработчике события hashchange. Это позволяет злоумышленнику контролировать строку, интерпретируемую jQuery как селектор, и, при определенных условиях, выполнить произвольный JavaScript-код.

  Постановка задачи: Проанализируйте код главной страницы сайта. Создайте и доставьте жертве (виртуальному пользователю) вредоносную веб-страницу (через предоставленный Exploit Server), которая при ее открытии вызовет функцию print() в браузере жертвы. Эксплуатация должна использовать событие изменения фрагмента URL (hashchange).

  Это задание вводит:
  1. Концепцию **Exploit Server** (сервер злоумышленника для доставки эксплойта).
  2. Непрямую эксплуатацию через событие hashchange и iframe.

  https://portswigger.net/web-security/cross-site-scripting/dom-based/lab-jquery-selector-hash-change-event

- **Задание 4 – PortSwigger — Reflected XSS с инъекцией в атрибут при экранировании угловых скобок (уровень – Apprentice) (2,5 балла)**

  Лабораторная работа демонстрирует reflected-уязвимость межсайтового скриптинга (Reflected XSS) в условиях базовой фильтрации входных данных. Сервер корректно экранирует угловые скобки (<, >), преобразуя их в HTML-сущности, что предотвращает создание новых тегов. Однако отражение управляемых пользователем данных происходит внутри значения атрибута существующего HTML-тега, что открывает путь к атаке через инъекцию нового атрибута (например, обработчика событий).

  Постановка задачи: Найдите уязвимую функцию поиска в блоге. Внедрите вредоносный payload, который, будучи отраженным на странице, добавит новый

атрибут (например, обработчик события) в существующий HTML-тег и вызовет функцию alert().

Важное примечание: Факт того, что payload вызывает alert() в вашем браузере, не гарантирует успешной атаки против жертвы. Вам может потребоваться перебрать несколько различных атрибутов (например, onmouseover, onfocus, onload и т.д.) в поиске того, который корректно сработает в контексте браузера жертвы.

https://portswigger.net/web-security/cross-site-scripting/contexts/lab-attribute-angle-brackets-html-encoded

- **Задание 5 – PortSwigger — Stored XSS через инъекцию в атрибут href тега <a> при экранировании двойных кавычек (уровень – Apprentice) (3 балла)**

Лабораторная работа демонстрирует stored-уязвимость межсайтового скриптинга (Stored XSS) с неочевидным вектором атаки. Уязвимость возникает в функционале комментариев, где сервер корректно экранирует двойные кавычки, но не фильтрует и не валидирует содержимое атрибута href тега <a>. Это позволяет злоумышленнику сохранить на странице вредоносную ссылку с псевдопротоколом javascript:, которая выполнится в браузере любого пользователя, кликнувшего на имя автора комментария..

Постановка задачи: Воспользуйтесь функцией добавления комментария. Внедрите и сохраните на сайте вредоносный payload, который вызовет функцию alert() при клике на имя пользователя (ссылку) в вашем комментарии.

https://portswigger.net/web-security/cross-site-scripting/contexts/lab-href-attribute-double-quotes-html-encoded

- **Задание 6 – PortSwigger — Reflected XSS в JavaScript-строку при экранировании угловых скобок (уровень – Apprentice) (2,5 балла)**

Лабораторная работа демонстрирует reflected-уязвимость межсайтового скриптинга (Reflected XSS) в специфическом и опасном контексте. Сервер корректно экранирует угловые скобки, предотвращая создание новых HTML-тегов. Однако управляемые пользователем данные отражаются внутри строкового литерала JavaScript-кода на странице. Чтобы эксплуатировать эту уязвимость, необходимо разорвать строку и внедрить исполняемый JS-код.

Постановка задачи: Найдите уязвимую функцию отслеживания поисковых запросов. Внедрите вредоносный payload, который, будучи отраженным внутри JavaScript-строки, осуществит:

1. **Выход из строкового литерала** (разрыв строки).
2. **Вызов функции** alert().
3. **Корректное завершение окружающего JS-кода** (во избежание синтаксических ошибок).

.

https://portswigger.net/web-security/cross-site-scripting/contexts/lab-javascript-string-angle-brackets-html-encoded

- **Задание 7 – PortSwigger — DOM XSS в функции document.write с использованием источника location.search внутри элемента select (уровень – Practitioner) (3,5 балла)**

Лабораторная работа демонстрирует усложненный сценарий DOM-базированного XSS. Уязвимость по-прежнему заключается в передаче управляемых пользователем данных (из параметра URL location.search) в опасную функцию document.write. Однако критически важным становится **контекст, в который эти данные записываются** — внутри существующего HTML-элемента <select>. Для успешной эксплуатации необходимо не только внедрить JavaScript, но и корректно "разорвать" окружающую разметку, чтобы браузер интерпретировал payload как валидный HTML-код, а не как текстовое содержимое тега.

Постановка задачи: Найдите на странице товара функцию проверки наличия на складе. Проанализируйте, как параметр URL передается в скрипт. Сконструируйте специальный URL с вредоносным значением параметра, которое:

1. **Корректно закроет текущий HTML-тег** (в данном случае </select>).
2. **Внедрит новый HTML-элемент с обработчиком события** (например, <img src=1 onerror=alert(1)>).
3. **При загрузке страницы вызовет функцию** alert()**.**

Это задание учит, что **опасный sink (document.write) — необходимое, но не достаточное условие**. Нужно всегда смотреть, *куда именно* происходит вывод, и экранировать/закрывать окружающие теги.

https://portswigger.net/web-security/cross-site-scripting/dom-based/lab-document-write-sink-inside-select-element

- **Задание 8 – PortSwigger — Stored XSS в событие onclick при многоуровневом экранировании символов (уровень – Practitioner) (4 балла)**

Лабораторная работа демонстрирует сложную stored XSS-уязвимость, защищенную **многоуровневой фильтрацией**. Сервер применяет к пользовательским данным последовательное преобразование:

1. Кодирование угловых скобок (<>) и двойных кавычек (") в HTML-сущности.
2. Экранирование одинарных кавычек (') и обратных слешей (\) в контексте JavaScript.

Однако данные попадают **внутри обработчика события** onclick, который является атрибутом HTML, но его значение выполняется как JavaScript. Это создает возможность для атаки с использованием **HTML-сущностей**, которые декодируются браузером *до* выполнения JavaScript, позволяя обойти фильтрацию.

Постановка задачи: Воспользуйтесь функцией добавления комментария. Проанализируйте, как введенные данные (поле "Website") обрабатываются и встраиваются в страницу. Создайте и сохраните payload, который, будучи обработанным фильтрами, после декодирования браузером вызовет функцию alert() при клике на имя автора комментария.

Это задание учит мыслить не только о том, *что* блокируется, но и о **том, в каком порядке и в каком контексте** данные будут интерпретированы конечной системой (браузером).

https://portswigger.net/web-security/cross-site-scripting/contexts/lab-onclick-event-angle-brackets-double-quotes-html-encoded-single-quotes-backslash-escaped

- **Задание 9 – PortSwigger — Reflected XSS в HTML-контекст при блокировке большинства тегов и атрибутов WAF (уровень – Practitioner) (4,5 балла)**

Лабораторная работа моделирует реалистичный сценарий, когда базовая reflected XSS-уязвимость защищена межсетевым экраном веб-приложений (WAF). WAF блокирует запросы, содержащие большинство известных XSS-векторов (популярные теги и атрибуты событий). Для решения задачи требуется методология обходного тестирования (WAF bypass): необходимо эмпирическим путем определить, какие именно HTML-теги и атрибуты событий не блокируются WAF, и скомбинировать их для создания рабочего эксплойта.

Постановка задачи:

1. **Обнаружение:** Используя технику фаззинга (например, с помощью Burp Intruder и списков тегов/атрибутов), определите:
    - Один HTML-тег, который не блокируется WAF.
    - Один атрибут события (например, onresize, onload), который не блокируется WAF.

2. **Эксплуатация:** Создайте и доставьте жертве (через Exploit Server) вредоносную страницу, которая вызовет функцию print() в браузере жертвы **без какого-либо взаимодействия с пользователем** (атака типа "drive-by").

Важное условие: Эксплойт должен быть полностью автоматическим. Вызов print() вручную в собственном браузере не засчитывается.

https://portswigger.net/web-security/cross-site-scripting/contexts/lab-html-context-with-most-tags-and-attributes-blocked

Часть 4 – Чеклист для самопроверки содержания отчета:

4.1. Обязательные элементы отчета

Отчет должен содержать следующие разделы для каждого выполненного задания:

1. Цель задания: Кратко, что требовалось сделать (например, "Вызвать alert() через отраженный в поиске XSS").
2. Исходные данные: URL лаборатории, учетные данные (если есть).
3. Пошаговое описание эксплуатации: Последовательность ваших действий и рассуждений. Описание должно быть достаточно детальным для воспроизведения. Обязательно укажите:
    - Источник данных (Source): Где вы нашли управляемые данные (например, параметр search в URL).
    - Контекст встраивания (Sink): Куда эти данные попадают и как обрабатываются (например, внутри атрибута href, в функцию document.write).
    - Вредоносный payload: Итоговый эксплуатационный код, который привел к успеху.
4. Доказательства выполнения: Скриншоты, подтверждающие ключевые этапы (см. требования 4.2).
5. Вывод по заданию: Краткий итог (например, "Уязвимость найдена в параметре X, попадающем в sink Y, что позволяет выполнить Z").

4.2. Технические требования к доказательствам (скриншотам)

- На каждом скриншоте должна быть текстовая подпись с вашими данными: Фамилия_Имя_Группа.
- Скриншоты должны наглядно подтверждать описанный шаг. Предпочтительные варианты:

- Burp Suite: Скриншот вкладки Repeater, Proxy > HTTP history или Intruder, где виден полный запрос/ответ и временная метка (Time).
- Браузер: Скриншот страницы с выполненным JavaScript (всплывающее alert()-окно, открытый print()-диалог) и открытой консолью разработчика (DevTools), показывающей соответствующую вкладку (Elements, Console, Debugger).

- Для заданий с доставкой эксплойта (Exploit Server) обязателен скриншот вкладки "Deliver exploit to victim" с отметкой об успехе ("Lab solved" или "Congratulations").

4.3. Аналитическая сводная таблица

В конце отчета необходимо предоставить сводную таблицу по всем выполненным атакам. Это позволит обобщить полученный опыт.

Попробуйте определить источник данных, контекст встраивания и прием, который был использован при атаке (и обходе защиты, если применимо).

- **Контекст встраивания (Sink):** Опасная функция/место, куда попадают данные.
- **Источник данных (Source):** Откуда берутся управляемые данные.
- **Ключевой обходной прием:** Метод, использованный для эксплуатации или обхода фильтров (например, инъекция атрибута, разрыв JS-строки, фаззинг WAF).
- **Рекомендация по защите:** Конкретная, техническая мера, которая предотвратила бы данную конкретную уязвимость.

Так же подумайте и попробуйте привести методы защиты, которые могли бы предостратить данную атаку. Пример заполнения таблиц приведен ниже

№	**Тип XSS (Reflected/Stored/DOM)**	**Атакуемая страница/функционал**	**Контекст встраивания (Sink)**	**Источник данных (Source)**	**Ключевой обходной прием (если был)**	**Рекомендация по защите (1-2 пункта)**
1	DOM-based	Страница поиска	`document.write()`	`location.search` (параметр URL)	Прямая передача данных в опасную функцию	- Замена document.write на безопасные методы (textContent, innerText). - Валидация или кодирование

						данных на стороне клиента перед выводом.
3	DOM-based	Главная страница	jQuery селектор $() в обработчике onhashchange	`location.hash (фрагмент URL)`	Использование события `hashchange` и `iframe` для автоматической доставки эксплойта жертве.	-Валидация значения location.hash перед использованием в селекторе. - Отказ от прямой подстановки пользовательских данных в селектор jQuery; использование методов поиска по ID с префиксом.
9	Reflected	Поиск по блогу	Тег <body> с обработчиком события onresize (инъекция в HTML-контекст	`location.search (параметр URL)`	Фаззинг (Burp Intruder) для обнаружения неблокируемых WAF комбинаций тега (<body>) и атрибута события (onresize).	Настройка WAF на блокировку более широкого спектра событий; валидация на стороне сервера.

4.4. Чек-лист для самопроверки перед сдачей

№	Элемент отчёта	Обязателен?	Примечание
1	Для каждого задания есть разделы: Цель, Исходные данные, Пошаговое описание, Доказательства, Вывод.	Да	
2	Все скриншоты содержат подпись **Фамилия_Имя_Группа**.	Да	
3	Скриншоты являются доказательными (виден запрос/ответ, факт выполнения JS, временные метки Burp).	Да	

4	В конце отчета есть **Аналитическая сводная таблица**, заполненная для всех заданий.	Да	
5	Отчет оформлен в соответствии с предоставленным шаблоном.	Да	