# Cross-site scripting (XSS). Контексты встраивания и правильный способ экранирования. Учебный стенд. Задание на лабораторную

## Оглавление



## XSS уязвимость

Хотя сама атака XSS [1] известна давно (впервые попала в top-10 уязвимостей в 2003 году), и методы защиты от нее известны. Если не рассматривать вариант встраивания XSS при передаче по сети (используя MiM атаку), то основной вектор атаки – это поиск уязвимых к XSS страниц на сервере и подбор пользовательского ввода таким образом, чтобы встроить нужный злоумышленнику JS код в страницу, отдаваемую непосредственно сервером.

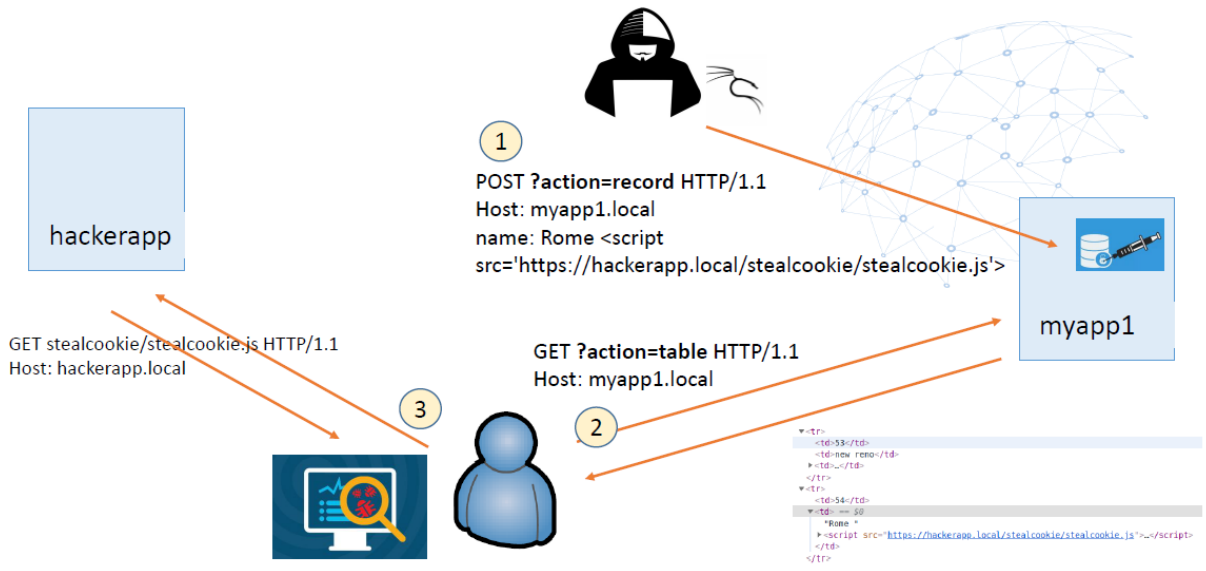


*Рисунок 1. Stored XSS атака*

Различают несколько типов XSS

- Stored XSS – когда злоумышленнику удается сохранить вредоносный код на сервере, и сервер в будущем сам отдает этот код всем пользователям, заходящим на данную страницу (такая схема приведена на рис. 1).

- Reflected XSS – когда не удается сохранить код на сервере, однако сервер возвращает в ответе тот самый текст, которые был ему передан в запросе. Сервер «отражает» в своем ответе данные, переданные в запросе (рис. 2).
- DOM XSS – в данной атаке, вредоносный код не возвращается в ответе с сервера, а формируется на клиенте в результате работы штатного JavaScript и манипуляции над DOM страницы.

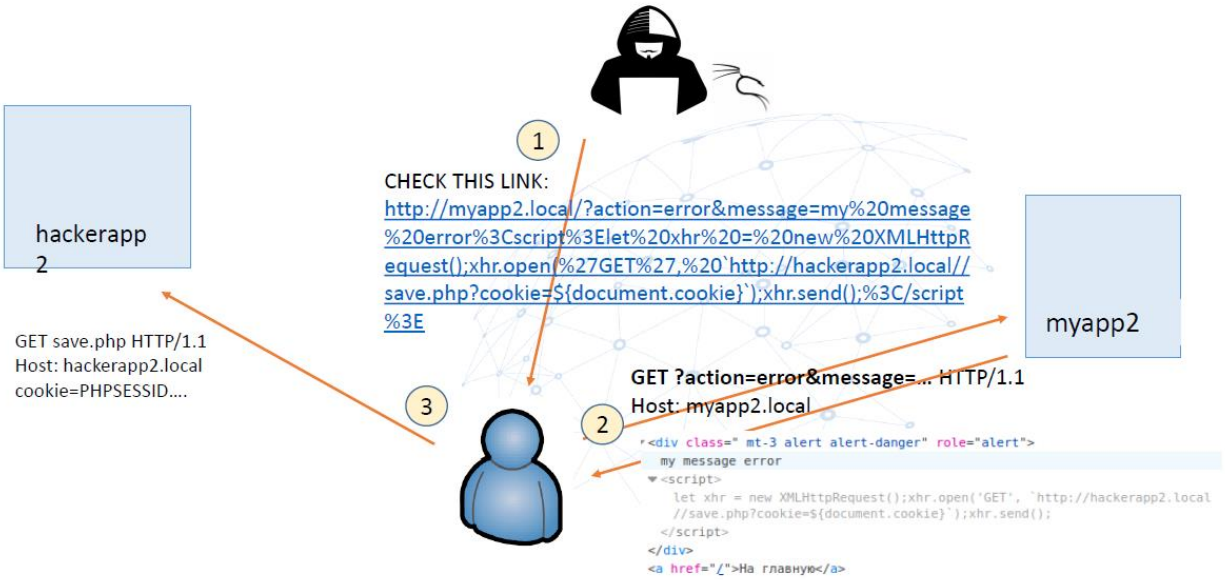


*Рисунок 2 Reflected XSS атака*

Отдельно различают еще скрытые XSS атаки (Blind Stored XSS) – это такой вариант XSS, когда вредоносный код успешно сохраняется на сервере, однако обнаружить это не получается. Это происходит, когда внедренный злоумышленником код не отображается на доступных ему страницах приложения, однако при этом он может быть отображен где-то в закрытой (служебной) части сайта. Типичный пример такой ситуации – внедрение уязвимости в системные журналы событий, журналы ошибок и т.п. Обычно такие журналы недоступны пользователю, однако их будет просматривать администратор. Если удается внедрить XSS на закрытую часть сайта, это потенциально позволяет атаковать пользователей системы с повышенными (администраторов, модераторов). Типичный пример атаки:

- Злоумышленник внедряет вредоносный код в журналы приложения (например, в сообщение об ошибке)
- Администратор через служебный интерфейс просматривает системные журналы и в этот момент его браузер подвергается атаке XSS

Зачастую, вопросу необходимости корректного экранирования данных, сохраняемых в журналы, не придается должного значения.

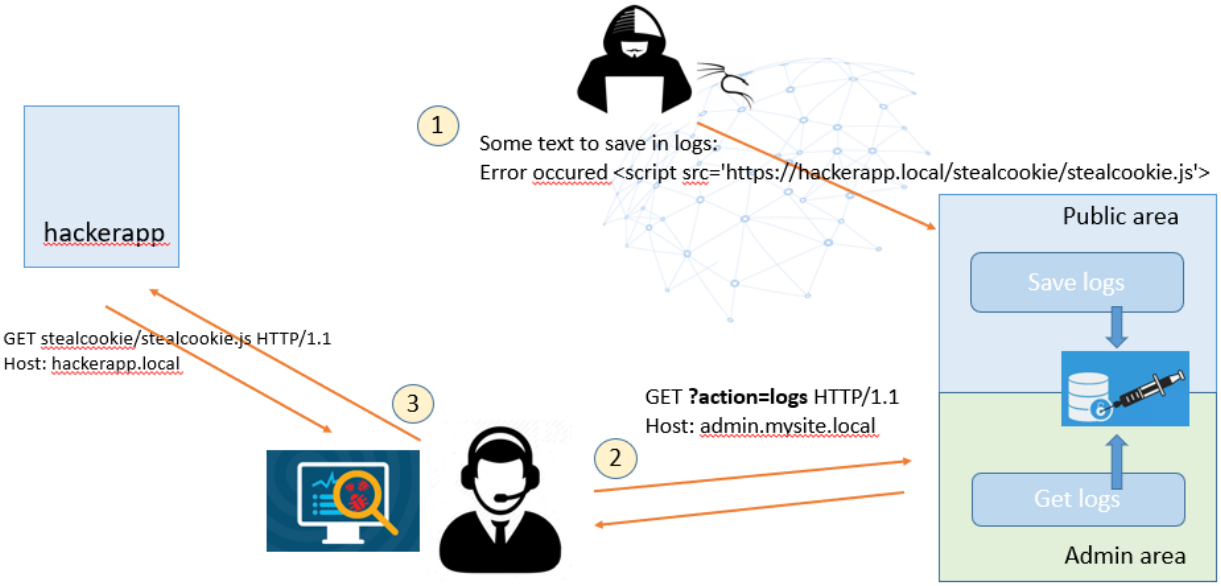


*Рисунок 3 Blind XSS*

## Способ внедрения XSS на страницу

Основной механизм атаки – выполнение вредоносного кода, подготовленного злоумышленником, в браузере пользователя при открытии атакуемого приложения. У злоумышленника есть для этого следующие возможности:

1. Передача кода в данных запроса (query params), в адресной строке
2. Передача кода в данных формы (любые input элементы, включая скрытые)
3. Передача кода в других данных, передаваемых в теле запроса (например, в составе JSON, передаваемого в теле HTTP запроса к серверу)
4. Передача кода в заголовках HTTP запроса. Так как заголовки могут легко модифицироваться на клиенте, они так же активно используются для атаки.

Рассмотрим, какими именно способами может быть добавлен JavaScript в тело страницы. Есть несколько способов, в том числе, не используя тег <script> вообще. Можно перечислить следующие:

- Явное указание тегов <script> и включение кода непосредственно между открывающим и закрывающим тегом
  <script>alert(1)</script>
- Загрузка JavaScript приложения, расположенного в отдельном js файле (в том числе, и с другого домена), с указанием атрибута src тега <script>
  - <script src='http://mysite.com/myapp.js'></script>
- Вызов JavaScript через псевдопротокол javascript [3,4]
  - Это можно делать в значениях атрибутов тегов, которые подразумевают URL, таких как src, href, action
  - <a href="javascript:alert(1);">empty anchor</a>
  - Так же псевдопротокол можно использовать в стилях CSS
    - { background url : javascript:alert (1)"; }
- Вызов JavaScript в обработчиках событий – специальных атрибутах тегов, таких как onerror, onclick, onmouseover и т.п.
  - <img src=x onerror=alert(1)>

Любой из них является штатным и позволяет злоумышленнику запустить на исполнение свой код, если приложение не использует специальные защитные механизмы. Рассмотрим их в селдующем разделе.

## Методы защиты

Основные методы защиты от XSS – **фильтрация** входных данных (пользовательского ввода) и **экранирование** небезопасного ввода при передаче его в браузер. Причем фильтрация рассматривается как необходимый, но недостаточный для защиты механизм, по нескольким причинам:

- Фильтрация по черным спискам неэффективна
- Эффективная фильтрация по белым спискам зачастую невозможна в силу специфики приложения (приходится пропускать «опасные» символы)
- Фильтрацию можно попробовать обойти, используя специальные схемы кодирования

По этой причине эффективным способом защиты признано экранирование данных перед передачей их в браузер. Однако правильное экранирование для защиты от XSS это не вполне тривиальная задача. Это вызвано тем, что браузер по-разному обрабатывает разные части страницы, и схема экранирования, эффективная в одном случае, будет неэффективна в другом. В качестве примера таких разных частей страницы можно перечислить: данные между HTML тегами, данные атрибутов тегов, названия тегов и атрибутов, данные внутри тегов <script>, данные стилей и т.п.

Вне зависимости от того, куда именно встраиваются данные, переданные пользователем, они должны проходить обязательное экранирование. Однако необходимо иметь в виду два важных момента:

1. Некоторые части страницы небезопасны настолько, что невозможно гарантировать защиту от XSS (вне зависимости от экранирования) если пользовательский ввод будет встроен в эти части страницы. Тут наилучшее решение – просто не допускать встраивание в них пользовательского ввода совсем.
2. Для других частей страницы (условно безопасных) допускается отображение данных, введенных пользователем, при их корректном экранировании. Корректность экранирования зависит от контекста внедрения.

Способ правильного экранирования пользовательского ввода при внедрении его на страницу зависит от контекста внедрения, т.е. от того, что именно будет окружать этот пользовательский ввод. Можно выделить следующие контексты внедрения кода:

- Контекст HTML - для простого HTML кода.
- Контекст атрибутов тегов HTML – названия, значения атрибутов тегов.
- Контекст ссылки (URL) – когда браузер обрабатывает что-то как ссылку (значения тегов href и src, action атрибут формы и т.п.)
- JavaScript контекст - в этом контексте, браузер позволяет напрямую выполнять команды (т.е. это контекст выполнения команд). Переключение в него – тег <script>, обработчики событий onerror и т.п., псевдопротокол javascript.
- Контекст стилей – каскадные таблицы стилей. Атрибут style, тег <style>.

Для каждого из указанных контекстов требуется применять свою схему экранирования.

Могут использоваться следующие схемы кодирования при экранировании:

- HTML Entity Encoding

- HTML Encoding для атрибутов
- URL Encoding
- JavaScript Encoding
- CSS Encoding

Для практического ознакомления с разными схемами экранирования и контекстами внедрения разработан учебный стенд для отработки защиты от XSS.

## Учебный стенд для отработки защиты XSS

Для закрепления материала по атаке XSS предлагается учебный стенд, на котором будут отработаны следующие вопросы:

1. Разные контексты встраивания XSS – HTML, URL, Script, CSS
2. Способы защиты страницы от встраивания XSS.

Материал данного стенда основан на рекомендациях OWASP по защите страниц от встраивания XSS [2].

В учебном стенде, для удобства и простоты использования, все атаки выполняются в форме Reflected XSS атак.

Для запуска стенда, скачайте подготовленную виртуальную машину по адресу

https://drive.google.com/file/d/123FUnm-QPc1fe499AQqFyki346O3HwRJ/view?usp=sharing

И запустите в браузере приложение

http://xsssamples.local/xssExamples.php

Следуйте инструкциям приложения, ознакомьтесь с различными видами атак. Пример внешнего вида учебного стенда приведен на рис.


*Рисунок 4 Окно учебного стенда*

## Лабораторная работа по XSS

Для выполнения задания вам потребуется специальное учебное приложение OWASP Mutillidae II.

Оно развернуто на той же виртуальной машине, что и учебный стенд для отработки защиты от XSS. Скачайте подготовленную виртуальную машину по адресу

https://drive.google.com/file/d/123FUnm-QPc1fe499AQqFyki346O3HwRJ/view?usp=sharing

Приложение доступно по ссылке http://nowasp.local/mutillidae/index.php

В данном приложении есть несколько страниц, уязвимых к XSS

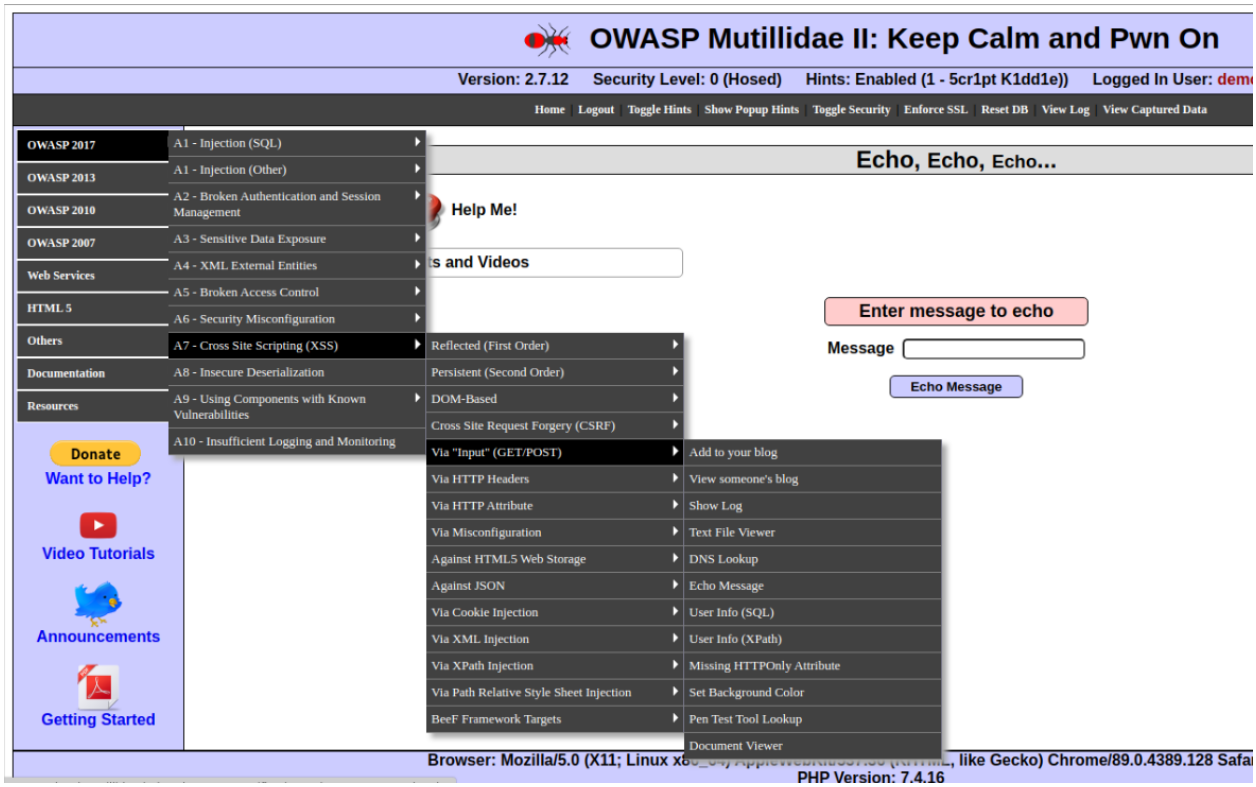


*Рисунок 5. Страницы, уязвимые к XSS в OWASP multillidae*

Необходимо отметить, что в данном приложении есть несколько уровней безопасности – от легкого, до максимального. При этом уровень Hosed выключает использование любых механизмов безопасности, при уровне Secure используются хорошие механизмы защиты.

Просмотреть текущий уровень безопасности можно в верхней части страницы. Для переключения режима безопасности используется ссылка «Toggle Security» так же в верхней части страницы.

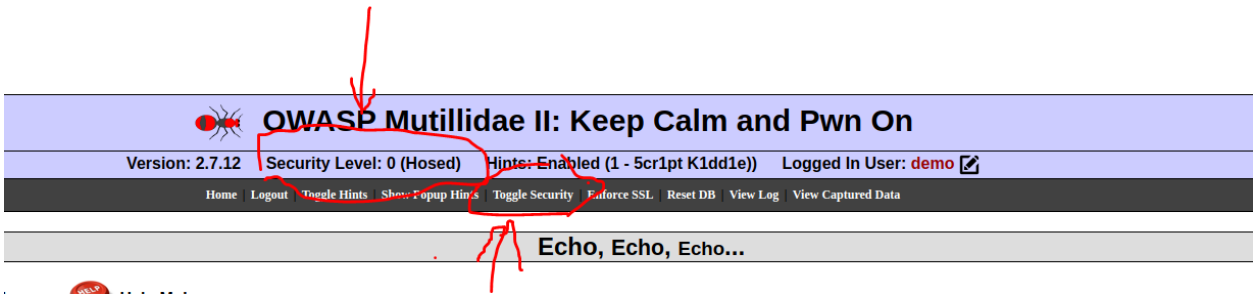


*Рисунок 6. Режим безопасности multillidae*

При выполнении работ вам необходимо провести атаку против незащищенных веб-страниц приложения. Последовательность атаки для каждой страницы следующая:

a. Обнаружить элементы страницы, уязвимые к XSS. Понять контекст встраивания пользовательского ввода на странице (объяснить в отчете).
b. Провести proof атаку, показывающую alert сообщение о взломе. Для этого необходимо сформировать корректный ввод для успешного встраивания XSS в зависимости от контекста (задокументировать в отчете последовательность атаки и пользовательский ввод).
c. Провести XSS атаку, обеспечивающую кражу cookie авторизованного пользователя. Продемонстрировать использование украденного идентификатора сессии пользователя (в другом браузере) для кражи его сессии. Для кражи cookie можно использовать приложение hackerapp.local учебного стенда либо страницу capture data приложения multillidae. Желательно провести скрытую атаку (без перенаправления пользователя).
d. Переключить учебный сайт в усиленный режим безопасности и провести атаку. Объяснить суть работы используемых защитных механизмов приложения.
e. (Дополнительное) Произвести атаку “прикрепления к Beef”. Продемонстрировать доступ к приложению из Beef.

Примечание: страницы capturedata и view captured data доступны в самом приложении multillidae.

1. View captured data http://nowasp.local/mutillidae/index.php?page=captured-data.php
2. Capture data http://nowasp.local/mutillidae/index.php?page=capture-data.php

Отчет об атаке должен содержать описание атакуемой страницы (уязвимые элементы), описание контекста встраивания, подобранный вредоносный ввод, результат атаки.

В данной лабораторной предлагается выбрать страниц для взлома (не больше 5) и на них продемонстрировать проведение атаки. За каждую страницу начисляются отдельные балы.

**Важное примечание:**

Балы не начисляются за проведение однотипных взломов. Т.е. например, проведение одинаковой reflected XSS атаки с встраиванием в простой HTML контекст два раза на разных страницах будет засчитано как одна атака. Для того, чтобы атаки были разные и засчитывались отдельно, необходимо выполнение одного из следующих условий:

1. Разные типы атак (reflected, stored, blind stored, DOM based)
2. Разные контексты встраивания (простой HTML, атрибуты тегов, скрипт, и т.п.)
3. Разные способы передачи вредоносного кога (параметры запроса, тело запроса, заголовки)

За каждую отличающуюся атаку будут засчитаны балы (не больше 5 атак).

## Дополнительная литература

1. Описание XSS (OWASP) https://owasp.org/www-project-top-ten/2017/A7_2017-Cross-Site_Scripting_(XSS)
2. Cross Site Scripting Prevention Cheat Sheet (OWASP) https://cheatsheetseries.owasp.org/cheatsheets/Cross_Site_Scripting_Prevention_Cheat_Sheet.html
3. Javascript URI scheme Draft https://datatracker.ietf.org/doc/html/draft-hoehrmann-javascript-scheme
4. About preudo-protocol javascript: https://www.programmersought.com/article/3480302229/