УДК 004

Проведение простой XSS-атаки

Языков Кирилл Александрович
Приамурский государственный университет имени Шолом-Алейхема
Студент

Аннотация
В статье на учебном примере рассматривается эксплуатация уязвимости межсайтового скриптинга (XSS). В локальной среде OpenServer Panel (OSPanel) развёрнуто простое веб-приложение на PHP с базой данных MySQL. Показано, как вывод пользовательских данных в HTML без обработки позволяет выполнить произвольный JavaScript в браузере посетителя – на примере отражённой и хранимой XSS, вплоть до доступа к cookie сессии. Приведены способы защиты: экранирование вывода, политика безопасности контента и флаг HttpOnly.
Ключевые слова: XSS, межсайтовый скриптинг, информационная безопасность, веб-уязвимости, PHP, экранирование вывода, Content Security Policy, HttpOnly, OSPanel.

Performing a simple XSS attack

Yazykov Kirill Alexandrovich
Sholom-Aleichem Priamursky State University
Student

Abstract
The paper uses an educational example to examine the exploitation of a cross-site scripting (XSS) vulnerability. A simple PHP web application with a MySQL database is deployed in a local OpenServer Panel (OSPanel) environment. It is shown how outputting user-supplied data into HTML without processing makes it possible to execute arbitrary JavaScript in a visitor’s browser – using reflected and stored XSS as examples, up to gaining access to the session cookie. Methods of protection are described: output escaping, a Content Security Policy, and the HttpOnly flag.
Keywords: XSS, cross-site scripting, information security, web vulnerabilities, PHP, output escaping, Content Security Policy, HttpOnly, OSPanel.

1 Введение

1.1 Актуальность

Большинство современных веб-приложений принимает данные от пользователя и затем отображает их на страницах: поисковые запросы, комментарии, имена, сообщения в чатах. Если такие данные выводятся в HTML без обработки, браузер не может отличить их от разметки и

исполняемого кода самого сайта и выполняет всё, что в них содержится. На этом и основана атака типа «межсайтовый скриптинг» (Cross-Site Scripting, XSS), при которой в страницу внедряется и выполняется чужой JavaScript. Межсайтовый скриптинг на протяжении многих лет входит в перечень OWASP Top 10 наиболее распространённых рисков безопасности веб-приложений. Понимание механизма такой атаки на простом примере необходимо как разработчикам для построения защиты, так и специалистам по информационной безопасности. Локальный запуск учебного стенда через OSPanel позволяет изучить уязвимость безопасно, не затрагивая сторонние ресурсы.

1.2 Обзор исследований

Вопросам разработки веб-приложений и их безопасности посвящён ряд работ. А.А. Сенина рассмотрела основные современные технологии разработки веб-приложений, их преимущества и недостатки [1]. В.Г. Ротарь, М. Ткачев и А.Е. Трофимова сравнили актуальные JavaScript-фреймворки для создания клиентской части веб-приложений [2]; именно исполнение JavaScript в браузере лежит в основе XSS-атак. С.А. Лесько описал модели и сценарии реализации угроз для интернет-ресурсов, включая SQL-инъекции и межсайтовый скриптинг [3]. Г.В. Беликов, И.Д. Крылов и В.А. Селищев рассмотрели защиту веб-приложений от внедрения вредоносного кода с применением политики безопасности контента [4]. Наиболее полно классификация атак межсайтового скриптинга и методов защиты от них представлена в зарубежной работе Ш. Гупты и Б.Б. Гупты [5].

1.3 Цель исследования

Целью работы является демонстрация на локальном учебном стенде механизма простой XSS-атаки – отражённой и хранимой – с выполнением произвольного JavaScript в браузере посетителя, а также описание способов устранения уязвимости.

2 Материалы и методы

Для выполнения работы использовались Windows 10, локальная серверная среда OpenServer Panel (OSPanel), включающая веб-сервер Apache, интерпретатор PHP и СУБД MySQL/MariaDB, а также браузер для просмотра страниц. Проект размещён в каталоге домена localhost, поэтому страницы стенда открываются по адресу http://localhost/xss_demo/.

Учебный стенд состоит из нескольких файлов. Файл db.php содержит параметры подключения к базе данных; search_vuln.php и search_safe.php – уязвимый и защищённый варианты страницы поиска (отражённая XSS); guestbook_vuln.php и guestbook_safe.php – уязвимый и защищённый варианты гостевой книги с хранением сообщений в MySQL (хранимая XSS); cookie_probe.js – небольшой скрипт для проверки доступности cookie из JavaScript. В уязвимых версиях пользовательские данные выводятся в HTML

напрямую, без экранирования, – именно это создаёт уязвимость. Проверка проводилась локально в браузере.

3 Результаты

Подключение к базе данных вынесено в отдельный файл db.php. Параметры соответствуют настройкам OSPanel по умолчанию: хост localhost, пользователь root, пустой пароль; имя учебной базы – xss_demo. Для корректного отображения кириллицы задана кодировка utf8mb4 (рис. 1).

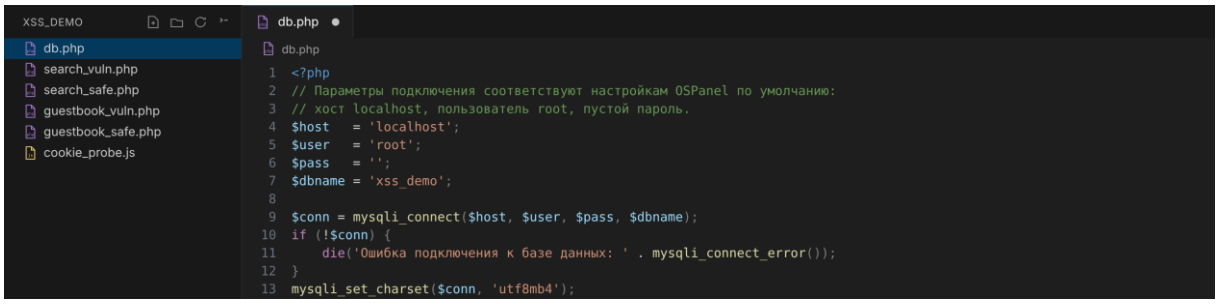


Рисунок 1 – Код файла подключения к базе данных db.php

Начнём с отражённой (reflected) XSS – самого простого вида этой атаки. В файле search_vuln.php значение поля поиска берётся из параметра запроса q и сразу выводится на страницу функцией echo без какой-либо обработки – и в результат поиска, и обратно в поле ввода. Именно отсутствие экранирования делает возможной инъекцию (рис. 2).

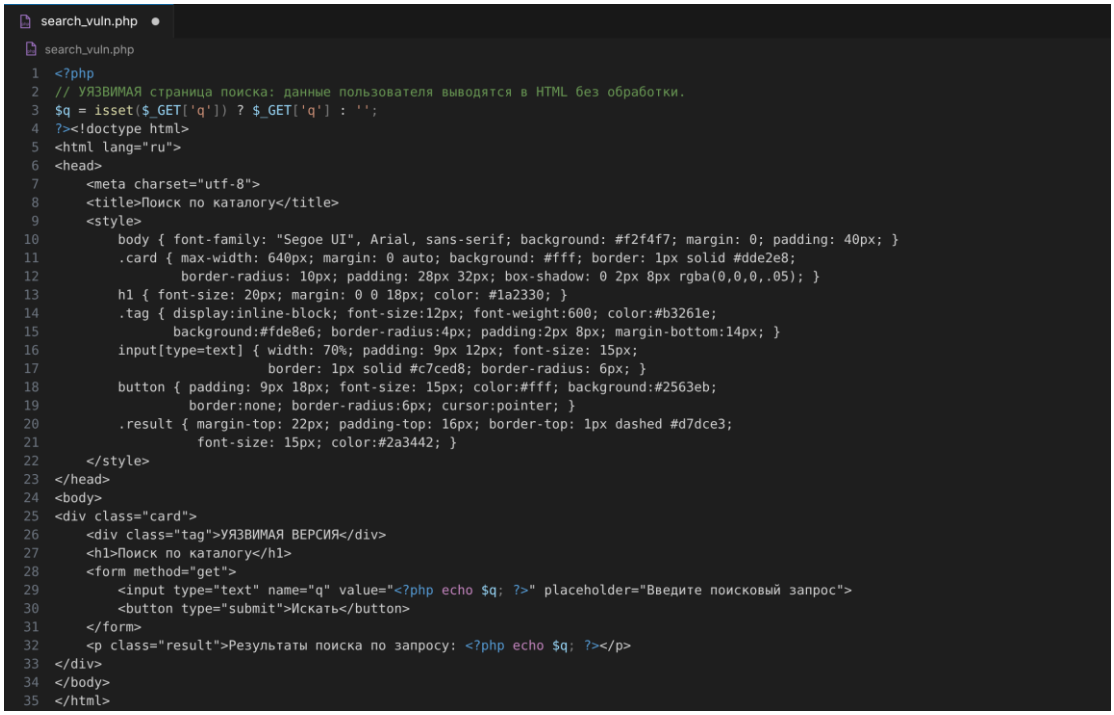


Рисунок 2 – Уязвимый вывод данных пользователя (search_vuln.php)

При обычном использовании страница работает ожидаемо: пользователь вводит запрос, например «смартфон», и видит строку «Результаты поиска по запросу: смартфон». Текст запроса просто подставляется в страницу (рис. 3).

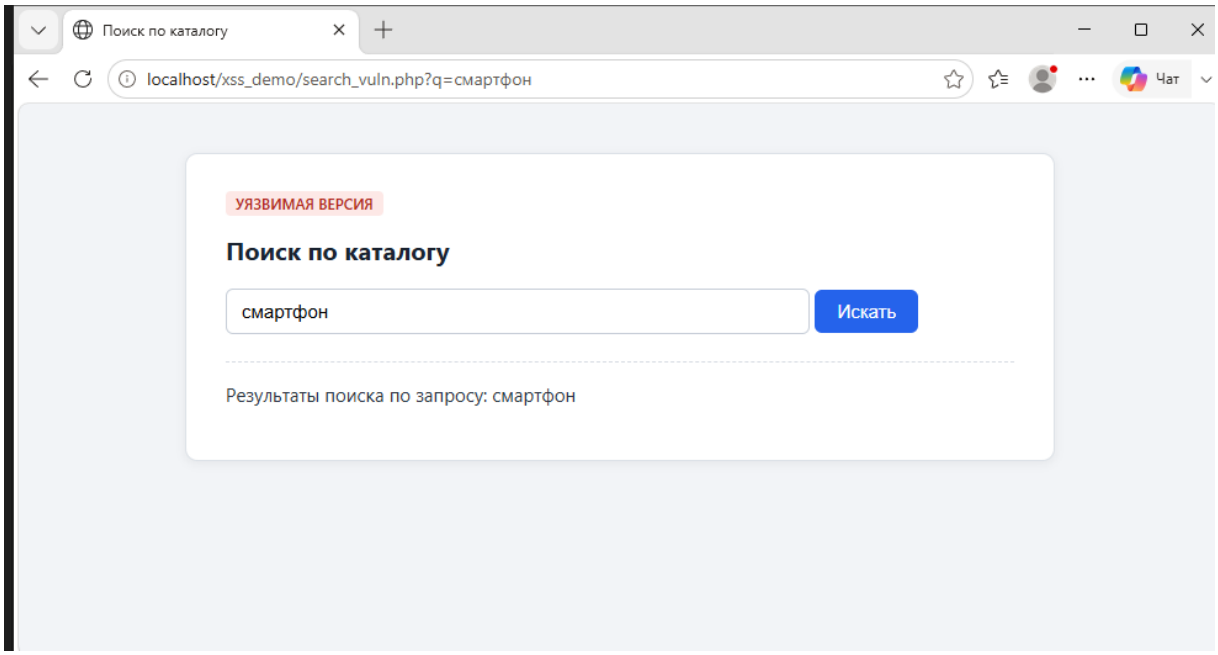


Рисунок 3 – Обычная работа страницы поиска

Теперь вместо обычного слова введём в поле поиска строку с тегом script: <script>alert('XSS')</script>

Браузер получает эту строку уже как часть HTML-кода страницы и воспринимает её не как текст, а как настоящий тег script, поэтому выполняет его содержимое. На экране появляется всплывающее окно с надписью «XSS» – доказательство того, что чужой JavaScript был выполнен на странице сайта (рис. 4). Этот же запрос можно заранее «зашить» в ссылку и отправить жертве: перейдя по ней, она выполнит код на сайте от своего имени. Вместо безобидного alert злоумышленник может выполнить любой сценарий – например, похитить данные пользователя или подменить содержимое страницы.

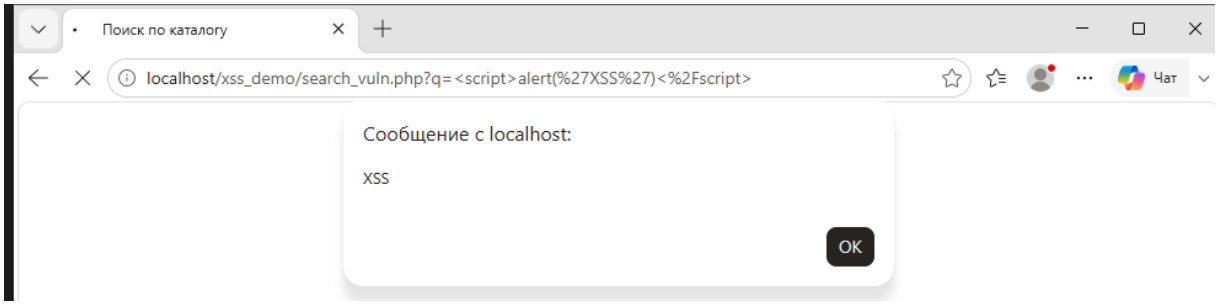


Рисунок 4 – Срабатывание отражённой XSS: выполнение внедрённого скрипта

Более опасной является хранимая (stored) XSS. В отличие от отражённой, вредоносный код не передаётся в ссылке, а сохраняется на

сервере – например, в базе данных – и выполняется у каждого, кто откроет страницу. Для демонстрации используется гостевая книга guestbook_vuln.php: посетители оставляют сообщения, которые сохраняются в таблице MySQL и выводятся всем. Важно, что запись в базу выполнена через подготовленное выражение (защита от SQL-инъекции), и тем не менее страница остаётся уязвимой – потому что при выводе сообщения снова подставляются в HTML без экранирования (рис. 5). Это показывает, что XSS – проблема именно вывода данных, а не их хранения.

guestbook_vuln.php

```php
<?php
// УЯЗВИМАЯ гостевая книга: демонстрация хранимой (stored) XSS.
// Cookie сессии создаётся с настройками по умолчанию (без HttpOnly),
// поэтому её значение доступно из JavaScript.
session_start();
require 'db.php';

// Добавление нового сообщения. Для записи в БД используется подготовленное
// выражение (это защита от SQL-инъекции, но НЕ от XSS).
if ($_SERVER['REQUEST_METHOD'] === 'POST') {
    $author  = isset($_POST['author'])  ? $_POST['author']  : 'Аноним';
    $message = isset($_POST['message']) ? $_POST['message'] : '';
    $stmt = mysqli_prepare($conn, "INSERT INTO entries (author, message) VALUES (?, ?)");
    mysqli_stmt_bind_param($stmt, 'ss', $author, $message);
    mysqli_stmt_execute($stmt);
    header('Location: guestbook_vuln.php');
    exit;
}

$res = mysqli_query($conn, "SELECT author, message, created_at FROM entries ORDER BY id DESC");
?><!doctype html>
<html lang="ru">
<head>
    <meta charset="utf-8">
    <title>Гостевая книга</title>
    <style>
        body { font-family: "Segoe UI", Arial, sans-serif; background: #f2f4f7; margin: 0; padding: 40px; }
        .card { max-width: 680px; margin: 0 auto; background: #fff; border: 1px solid #dde2e8;
                border-radius: 10px; padding: 28px 32px; box-shadow: 0 2px 8px rgba(0,0,0,.05); }
        h1 { font-size: 20px; margin: 0 0 6px; color: #1a2330; }
        .tag { display:inline-block; font-size:12px; font-weight:600; color:#b3261e;
               background:#fde8e6; border-radius:4px; padding:2px 8px; margin-bottom:14px; }
        input, textarea { width: 100%; box-sizing: border-box; padding: 9px 12px; font-size: 14px;
                          border: 1px solid #c7ced8; border-radius: 6px; margin-bottom: 10px; font-family: inherit; }
        button { padding: 9px 18px; font-size: 15px; color:#fff; background:#2563eb;
                 border:none; border-radius:6px; cursor:pointer; }
        .msg { border-top: 1px solid #eceff3; padding: 12px 0; font-size: 15px; color:#2a3442; }
        .msg .who { font-weight: 600; color:#1a2330; }
        .msg .when { color:#8a95a3; font-size: 12px; margin-left: 6px; }
        .list { margin-top: 20px; }
    </style>
</head>
<body>
<div class="card">
    <div class="tag">УЯЗВИМАЯ ВЕРСИЯ</div>
    <h1>Гостевая книга</h1>
    <form method="post">
        <input type="text" name="author" placeholder="Ваше имя">
        <textarea name="message" rows="3" placeholder="Ваше сообщение"></textarea>
        <button type="submit">Отправить</button>
    </form>
    <div class="list">
        <?php while ($row = mysqli_fetch_assoc($res)): ?>
            <div class="msg">
                <span class="who"><?php echo $row['author']; ?></span>
                <span class="when"><?php echo $row['created_at']; ?></span>
                <div><?php echo $row['message']; ?></div>
            </div>
        <?php endwhile; ?>
    </div>
</div>
</body>
</html>
```

Рисунок 5 – Уязвимый вывод сообщений в гостевой книге (guestbook_vuln.php)

Пусть злоумышленник оставит в гостевой книге сообщение, которое вместо текста содержит скрипт, читающий cookie сайта: <script>alert(document.cookie)</script>. Это сообщение сохраняется в базе и теперь будет отдаваться каждому посетителю. Как только обычный пользователь открывает гостевую книгу, скрипт срабатывает в его браузере и показывает содержимое cookie, в том числе идентификатор сессии PHPSESSID (рис. 6). Получив это значение, злоумышленник смог бы войти на сайт под чужой учётной записью без пароля. Так одна вредоносная запись угрожает всем посетителям страницы.

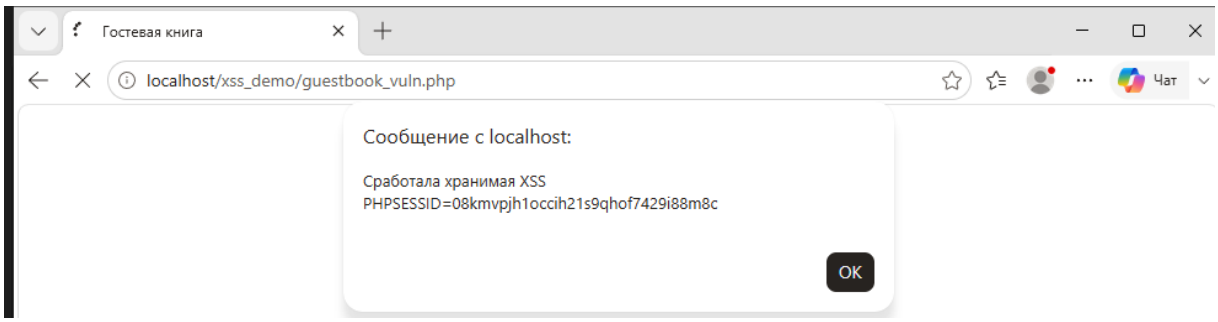


Рисунок 6 – Срабатывание хранимой XSS и доступ к cookie сессии

Основная причина уязвимости в обоих случаях одна – вывод пользовательских данных в HTML без обработки. Главный способ защиты – экранирование вывода. В PHP для этого применяется функция htmlspecialchars, которая заменяет спецсимволы HTML на безопасные сущности: символ < превращается в <, > – в >, кавычки – в " и '. После такой обработки строка <script> перестаёт быть тегом и отображается как обычный текст. В файле search_safe.php тот же запрос перед выводом пропускается через htmlspecialchars, а дополнительно устанавливается заголовок Content-Security-Policy (CSP), запрещающий выполнение посторонних скриптов (рис. 7).

search_safe.php

```php
<?php
// ЗАЩИЩЁННАЯ страница поиска.
// 1. Политика безопасности контента запрещает выполнение сторонних скриптов.
header("Content-Security-Policy: default-src 'self'; script-src 'self'; style-src 'self' 'unsafe-inline'");

$q = isset($_GET['q']) ? $_GET['q'] : '';
// 2. Экранирование вывода: спецсимволы HTML превращаются в безопасные сущности.
$safe = htmlspecialchars($q, ENT_QUOTES, 'UTF-8');
?><!doctype html>
<html lang="ru">
<head>
    <meta charset="utf-8">
    <title>Поиск по каталогу</title>
    <style>
        body { font-family: "Segoe UI", Arial, sans-serif; background: #f2f4f7; margin: 0; padding: 40px; }
        .card { max-width: 640px; margin: 0 auto; background: #fff; border: 1px solid #dde2e8;
                border-radius: 10px; padding: 28px 32px; box-shadow: 0 2px 8px rgba(0,0,0,.05); }
        h1 { font-size: 20px; margin: 0 0 18px; color: #1a2330; }
        .tag { display:inline-block; font-size:12px; font-weight:600; color:#136f3b;
               background:#e4f6ea; border-radius:4px; padding:2px 8px; margin-bottom:14px; }
        input[type=text] { width: 70%; padding: 9px 12px; font-size: 15px;
                           border: 1px solid #c7ced8; border-radius: 6px; }
        button { padding: 9px 18px; font-size: 15px; color:#fff; background:#2563eb;
                 border:none; border-radius:6px; cursor:pointer; }
        .result { margin-top: 22px; padding-top: 16px; border-top: 1px dashed #d7dce3;
                  font-size: 15px; color:#2a3442; }
    </style>
</head>
<body>
<div class="card">
    <div class="tag">ЗАЩИЩЁННАЯ ВЕРСИЯ</div>
    <h1>Поиск по каталогу</h1>
    <form method="get">
        <input type="text" name="q" value="<?php echo $safe; ?>" placeholder="Введите поисковый запрос">
        <button type="submit">Искать</button>
    </form>
    <p class="result">Результаты поиска по запросу: <?php echo $safe; ?></p>
</div>
</body>
</html>
```

Рисунок 7 – Защищённый вывод: экранирование и политика безопасности контента (search_safe.php)

Если теперь передать защищённой странице ту же полезную нагрузку <script>alert('XSS')</script>, всплывающее окно не появляется: скрипт не выполняется, а выводится как обычный текст прямо в строке результата поиска (рис. 8). Атака нейтрализована.

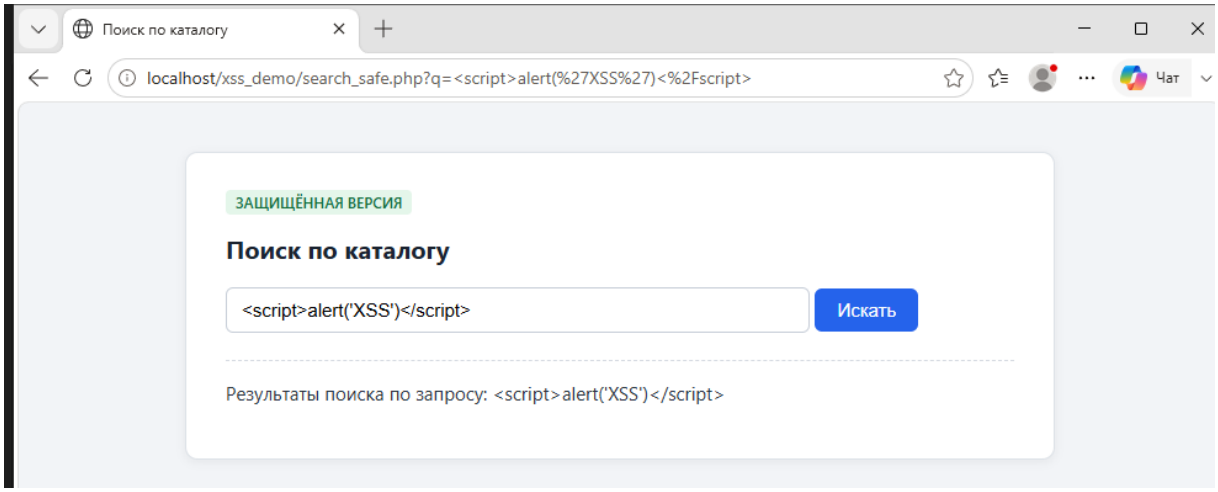


Рисунок 8 – Защищённая страница: полезная нагрузка отображается как текст

Аналогично защищается и гостевая книга. В файле guestbook_safe.php каждое сообщение выводится через htmlspecialchars, поэтому ранее

сохранённая вредоносная запись теперь отображается как безобидный текст и не выполняется. Кроме того, cookie сессии создаётся с флагом HttpOnly, из-за чего она становится недоступной для JavaScript. Встроенная на страницу проверка document.cookie подтверждает это: даже если бы скрипт выполнился, прочитать идентификатор сессии он бы уже не смог (рис. 9).

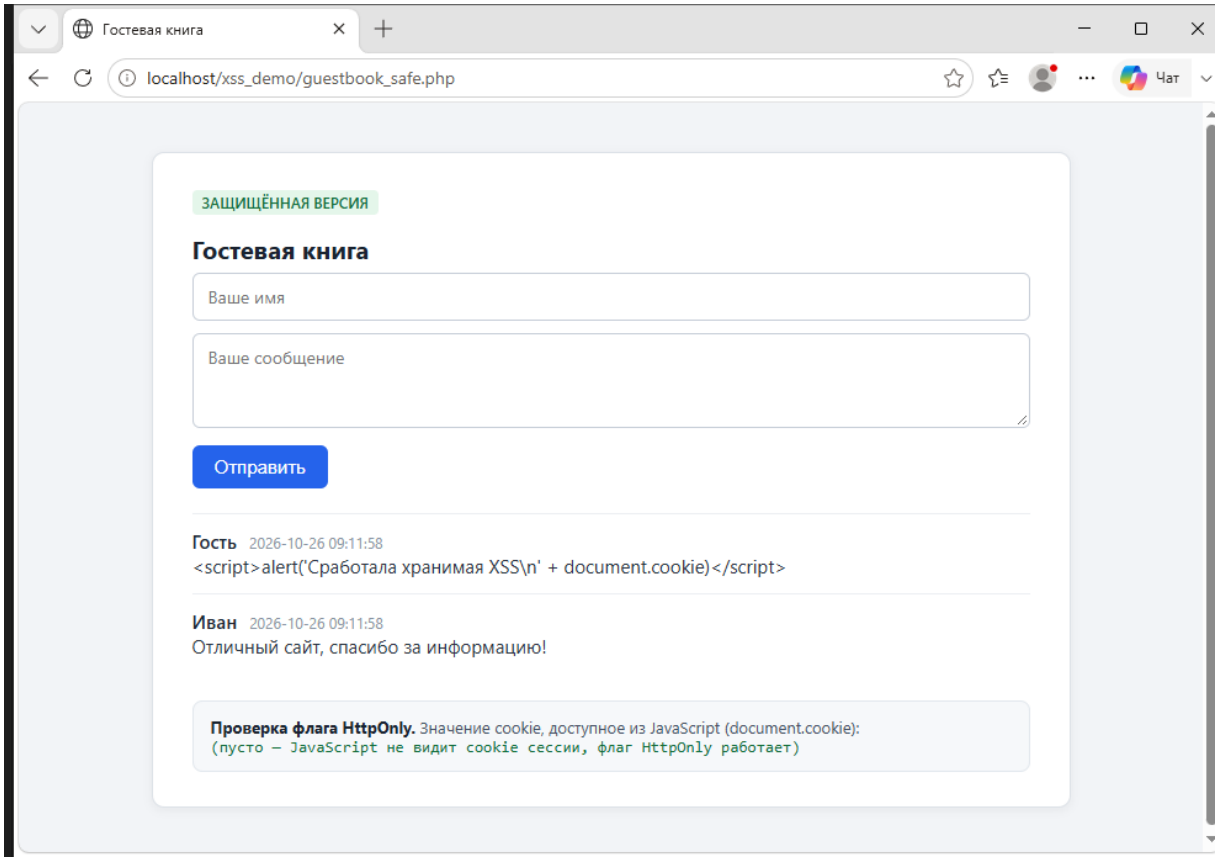


Рисунок 9 – Защищённая гостевая книга: нагрузка обезврежена, cookie скрыта флагом HttpOnly

Перечисленные меры дополняют друг друга и работают на разных уровнях. Их назначение и ограничения сведены в таблице 1.

Таблица 1 – Методы защиты от XSS и их ограничения

Метод защиты	**От чего защищает**	**Ограничения**
Экранирование вывода (htmlspecialchars)	Отражённая и хранимая XSS	Применяется к каждому выводу; выбор способа зависит от контекста (HTML, атрибут, URL, JavaScript)
Content Security Policy (CSP)	Выполнение встроенных и сторонних скриптов	Требует аккуратной настройки; при ошибках может блокировать легитимные скрипты и стили

Флаг HttpOnly для cookie	Кражу cookie сессии через JavaScript	Не предотвращает саму XSS, а лишь ограничивает возможный ущерб
Валидация и фильтрация ввода	Часть некорректных данных на входе	Не заменяет экранирование вывода; при неполных правилах легко обходится

4 Выводы

В ходе работы на локальном учебном стенде, развёрнутом средствами OSPanel, продемонстрирован механизм простой XSS-атаки. Показано, что вывод пользовательских данных в HTML без обработки позволяет выполнить произвольный JavaScript в браузере посетителя: в случае отражённой XSS – по специально подготовленной ссылке, в случае хранимой – у каждого, кто открывает страницу, вплоть до доступа к cookie сессии. Рассмотрены способы устранения уязвимости – экранирование вывода функцией htmlspecialchars, политика безопасности контента (CSP) и флаг HttpOnly для cookie. Материал может использоваться как учебный пример при изучении безопасности веб-приложений и демонстрирует ключевое правило: любые данные от пользователя перед выводом на страницу необходимо экранировать.

Важно: все описанные действия выполнялись исключительно в образовательных целях на собственном локальном сайте, принадлежащем автору. Проведение подобных действий в отношении информационных систем и сайтов без согласия их владельца является противоправным. В соответствии со статьёй 272 Уголовного кодекса Российской Федерации «Неправомерный доступ к компьютерной информации» неправомерный доступ к охраняемой законом компьютерной информации, повлёкший её копирование, предусматривает уголовную ответственность. Автор не несёт ответственности за использование материалов статьи в противоправных целях.

Библиографический список

1. Сенина А. А. Обзор основных современных технологий разработки Web-приложений //Технологии Microsoft в теории и практике программирования. 2016. С. 233-235. URL: https://vufind.lib.tpu.ru/vufind/Record/620958
2. Ротарь В. Г., Ткачев М., Трофимова А. Е. Обзор актуальных фреймворков для разработки клиентской части современных веб-приложений //Технологии Microsoft в теории и практике программирования. 2021. С. 170-171. URL: https://earchive.tpu.ru/bitstream/11683/68039/1/conference_tpu-2021-C04_p170-171.pdf
3. Лесько С. А. Модели и сценарии реализации угроз для интернет-ресурсов //Российский технологический журнал. 2020. Т. 8. №. 6. С. 9-33. URL: https://doi.org/10.32362/2500-316X-2020-8-6-9-33

4. Беликов Г. В., Крылов И. Д., Селищев В. А. Защита веб-приложений от уязвимости PHP Code Injection //Известия Тульского государственного университета. Технические науки. 2022. №. 5. С. 122-128. URL: https://cyberleninka.ru/article/n/zaschita-veb-prilozheniy-ot-uyazvimosti-php-code-injection
5. Gupta S., Gupta B. B. Cross-Site Scripting (XSS) attacks and defense mechanisms: classification and state-of-the-art //International Journal of System Assurance Engineering and Management. 2017. Vol. 8. P. 512-530. URL: https://vufind.lboro.ac.uk/PrimoRecord/cdi_proquest_journals_1880783584
6. Уголовный кодекс Российской Федерации от 13.06.1996 № 63-ФЗ. Статья 272. Неправомерный доступ к компьютерной информации. URL: https://www.consultant.ru/document/cons_doc_LAW_10699/ (дата обращения: 24.10.2026).
7. Материалы разработки веб-приложения для проведения XSS-атаки: https://drive.google.com/drive/folders/1q5cBpsnsiTZa1yCtSmtNViFuf9yaVA6b