Беляева К.В. Безопасность веб-разработки: HTTPS, CORS, XSS, CSRF, CSP // Международный журнал информационных технологий и энергоэффективности. – 2023. – Т. 8 № 5(31) ч.2. с. 83–85

Международный журнал информационных технологий и энергоэффективности

Сайт журнала:
http://www.openaccessscience.ru/index.php/ijcse/

УДК 004.056

БЕЗОПАСНОСТЬ ВЕБ-РАЗРАБОТКИ: HTTPS, CORS, XSS, CSRF, CSP

Беляева К.В.
ФГБУО ВО «МИРЭА - Российский технологический университет», Москва, Россия (119454, г. Москва, пр. Вернадского, 78), e-mail: kaleriaa@bk.ru

В данной статье рассматриваются основные понятия и термины в области безопасности веб-разработки, которые необходимо знать для создания безопасных веб-приложений. Описываются такие уязвимости, как XSS, CSRF, а также методы защиты данных, такие как HTTPS. Знание этих основных понятий и терминов позволит разработчикам создавать более безопасные веб-приложения и минимизировать риски возникновения угроз в безопасности.

Ключевые слова: Веб-разработка, безопасность, HTTPS, CORS, XSS, CSRF, CSP.

WEB DEVELOPMENT SECURITY: HTTP, CARS, XSS, CSRF, CSP

Belyaeva K.V.
MIREA - Russian Technological University, Moscow, Russia (119454, Moscow, Vernadskogo Ave., 78), e-mail: kaleriaa@bk.ru

This article covers the basic concepts and terms in the field of web development security that you need to know to create secure web applications. Vulnerabilities such as XSS, CSRF are described, as well as data protection methods such as HTTPS. Knowing these basic concepts and terms will allow developers to create more secure web applications and minimize the risk of security threats.

Keywords: Web development, security, HTTPS, CORS, XSS, CSRF, CSP.

Много лет назад веб-страницы были довольно простыми и имели мало функциональности. Язык JavaScript использовался главным образом для создания эффектов и анимации на страницах, а также для проверки правильности заполнения форм. Однако со временем развитие технологий привело к тому, что JavaScript стал намного мощнее, добавились методы для сетевых запросов и многое другое, что позволяет создавать сложные приложения, работающие как на клиентской, так и на серверной сторонах (Node.js). Так, с увеличением функциональности JavaScript существует риск возникновения уязвимостей в безопасности, поэтому важно правильно использовать этот инструмент и следовать современным методам безопасности при разработке веб-приложений. Следование правилу, что скрипт с одной страницы не мог получить доступ к содержимому другой страницы, было основой веб-безопасности многие годы. Хотя даже это правило разработчики научились обходить, используя разные способы, например, отправка форм на другой сервер, использование в атрибуте src тега script любой домен.

В итоге, запросы на другой источник можно делать с некоторым ограничением – необходимо согласие сервера. CORS (Cross-Origin Resource Sharing) – механизм, который ограничивает доступ к контенту, расположенному на других доменах. Без использования CORS браузеры будут блокировать запросы, которые пытаются получить доступ к ресурсам, находящимся на других доменах. При отправке запроса на другой ресурс, браузер автоматически добавляет в запрос заголовок Origin, который содержит информацию об источнике, откуда был отправлен запрос. Сервер проверяет источник и если доступ разрешен, то добавляется особый заголовок к ответу, содержащий источник либо *. CORS позволяет контролировать доступ к сторонним ресурсам и предотвращает возможность атак межсайтовой подделки запроса (CSRF).

CSRF (Cross-Site Request Forgery) — это атака, при которой злоумышленник может выполнить определенное действие от имени пользователя без его согласия [2]. Это может быть смена пароля в системе, перевод денег и т.д., зависит, конечно же, от системы. Для осуществления вредоносных действий пользователь должен перейти по ссылке или выполнить что-то на подготовленном сайте, которые отправят определенный запрос. Многие сайты используют слабое звено авторизации – хранение сессии пользователей в куках, поэтому классический сценарий с отправкой формы позволит преступникам исполнить свои намерения. То есть, целевой сайт проверяет куки, видит, что посетитель авторизован и обрабатывает форму. Для защиты от CSRF необходимо использовать специальные секретные ключи (создается при авторизации и сохраняется в сессии пользователя) и генерированные на их основе токены, которые добавляются в каждый запрос и проверяются на стороне сервера. Кроме того, можно использовать HTTP заголовки Origin или для отслеживания источника запроса.

XSS (Cross-Site Scripting) – это тип атаки на веб-приложения, когда злоумышленник внедряет вредоносный код на страницу или в формы на сайте. Одним из простых сценариев использования уязвимости может быть: при вводе текста в поле и при последующим его отображением на страницу браузер будет обрабатывать текст между тегами <script> как JS-код, тоже происходит и с другими тегами. Теперь, когда пользователи зайдут на страницу, то вредоносный код загрузится вместе с текстом из поля ввода и/ или при вводе кода в параметре URL, если текст из поля дублируется в запросе. Однако, такое сработает только, если текст из поля не был обработан должны образом, то есть экранированием. Для защиты от XSS можно использовать механизмы фильтрации пользовательского ввода, HTML-экранирование и Content Security Policy (CSP).

CSP (Content Security Policy, политика безопасности контента) — это механизм безопасности веб-приложений, которые позволяют ограничить и контролировать источники загрузки ресурсов на странице. CSP может помочь защитить сайт от целого ряда атак, таких как XSS, clickjacking, подделка запросов и других. Для этого на страницу добавляется HTTP-заголовок Content-Security-Policy и директивы. CSP задает список допустимых источников для загрузки ресурсов, таких как скрипты, стили, изображения, шрифты, видео, звук и другие. Если ресурсы загружаются из недопустимых источников, то браузер блокирует их загрузку и сообщает об ошибке. Данная политика может быть установлена как HTTP-заголовок, тег мета или атрибут HTML-элемента. Обычно ее следует использовать вместе с другими механизмами защиты, такими как HTTPS, CORS и другими.

Таким образом, знание о существующих уязвимостях может помочь их своевременно обнаружить и предотвратить неправомерный действия, а понимание механизмов/ стандартов в области веб-безопасности поможет правильно их применять на практике, сохраняя данные пользователей конфиденциальными.

Список литературы

1. Fetch: запросы на другие сайты // Современный учебник JavaScript URL: https://learn.javascript.ru/fetch-crossorigin (дата обращения: 21.05.2023).
2. Атака CSRF // Современный учебник JavaScript URL: https://learn.javascript.ru/csrf (дата обращения: 21.05.2023).
3. CSP // Skillfactory Media URL: https://blog.skillfactory.ru/glossary/csp/ (дата обращения: 21.05.2023).
4. Что такое XSS-уязвимость и как тестировщику не пропустить ее // Habr URL: https://habr.com/ru/articles/511318/ (дата обращения: 21.05.2023).

References

1. Fetch: requests to other sites // Modern JavaScript tutorial URL: https://learn.javascript.ru/fetch-crossorigin (accessed on: 21.05.2023).
2. CSRF attack // Modern JavaScript textbook URL: https://learn.javascript.ru/csrf (accessed on: 21.05.2023).
3. CSP // Skillfactory Media URL: https://blog.skillfactory.ru/glossary/csp/ (дата обращения: 21.05.2023).
4. What is an XSS vulnerability and how can a tester not miss it) // Habr URL: https://habr.com/ru/articles/511318/ (accessed 21.05.2023).