УДК 519.683, 519.681

УПРАВЛЕНИЕ ДОСТУПОМ К ЗАЩИЩЕННЫМ РЕСУРСАМ ПРИ ПОМОЩИ JSON WEB TOKENS

Рябов Никита Владимирович[1], Шавловский Марк Вячеславович[2], Подкопаев Валентин Васильевич[3]

[1]Аспирант;
ГБОУ ВО МО «Университет «Дубна»;
Институт системного анализа и управления;
141980, Московская обл., г. Дубна, ул. Университетская, 19;
e-mail: ryabov_nv95@mail.ru.

[2]Студент;
ГБОУ ВО МО «Университет «Дубна»;
Институт системного анализа и управления;
141980, Московская обл., г. Дубна, ул. Университетская, 19;
e-mail: shavl.mark@yandex.ru.

[3]Студент;
ГБОУ ВО МО «Университет «Дубна»;
Институт системного анализа и управления;
141980, Московская обл., г. Дубна, ул. Университетская, 19;
e-mail: max.astin@yandex.ru.

В данной статье рассматривается технология применения json web токенов (JWT) для управления доступом в веб-приложении, структура и процесс создания токенов, а также варианты их хранения на стороне клиента.

Ключевые слова: JWT, токен, *cookies, localStorage*, аутентификация, веб-приложение.

CONTROL ACCESS TO PROTECTED RESOURCES WITH JSON WEB TOKENS

Ryabov Nikita[1], Shavlovsky Mark[2], Podkopaev Valentin[3]

[1]PhD student;
Dubna State University;
Institute of the system analysis and management;
141980, Dubna, Moscow reg., Universitetskaya str., 19;
e-mail: ryabov_nv95@mail.ru.

[2]Student;
Dubna State University;
Institute of the system analysis and management;
141980, Dubna, Moscow reg., Universitetskaya str., 19;
e-mail: shavl.mark@yandex.ru.

[3]Student;
Dubna State University;
Institute of the system analysis and management;
141980, Dubna, Moscow reg., Universitetskaya str., 19;
e-mail: max.astin@yandex.ru.

The article shows the technology of using JSON web tokens (JWT) to control access in a web-application, the structure and process of creating tokens, options for storing tokens on the client side.

Keywords: JWT, token, cookies, localStorage, authentication, web-application.

Введение

Вопросы безопасности стоят очень остро в современном мире. Крупные компании обязаны тратить большие усилия на сохранение приватности собственных данных и персональных данных пользователей. Это не всегда получается и компаниям приходится терпеть репутационные и финансовые убытки.

Проекты ниже уровнем сталкиваются с хакерскими атаками гораздо реже, но это не значит, что о безопасности можно вообще не думать.

Часть, имеющаяся в большинстве веб-приложений – управление доступом пользователей к содержимому сайта. Самый распространенный подход – сессии. На стороне сервера выполняется вся тяжёлая работа. Клиент проводит аутентификацию со своими учётными данными и получает *id* сессии, которое присоединяется к каждому исходящему запросу. В самом *id* сессии нет ничего необычного. Это просто идентификатор, а остальную работу делает сервер. На сервере идентификатор связывается с учетной записью пользователя. У сессий есть свои преимущества, но также и ряд недостатков, одни из которых – плохая масштабируемость и повышенное использование памяти.

Другой подход – аутентификация при помощи токенов, которые содержат полномочия сессии и идентифицируют пользователя, его группу, привилегии. Самый распространенный вариант применения подобной технологии – *JSON Web Token* (*JWT*). Это открытый стандарт для создания токенов доступа, в котором утверждены требования [1]. Самое главное отличие от сессий – в аутентификации при помощи токенов сам токен, или сессия не хранится на сервере. Для уменьшения использования памяти, простоте масштабируемости и гибкости выдается строка со всей необходимой информацией (токен), которая проверяется после каждого запроса, сделанного клиентом на сервер. О применение *JWT* и пойдёт речь в статье.

Структура JWT

Рассмотрим использование *JWT* на примере абстрактного веб-приложения. В таком случае существует три участника: пользователь, сервер аутентификации и сервер самого приложения. Сервер аутентификации будет обеспечивать пользователя токеном, с помощью которого он позднее сможет взаимодействовать с приложением (см. рис. 1).

Проверка аутентификации происходит следующим образом:

1. Пользователь заходит на сервер аутентификации с помощью аутентификационного ключа (пара логин/пароль, *Facebook* ключ, *Google* ключ, ключ от другой учётной записи пользователя).
2. Сервер аутентификации создает *JWT* и отправляет его пользователю.
3. Пользователь, делая запрос к *API* приложения, добавляет к нему полученный ранее *JWT*.

В *JWT* отсутствуют сессии – это означает, что они не хранятся. Это может быть полезно, если приложения должны масштабироваться горизонтально. Если приложение запускает на нескольких серверах, то совместное использование данных сеанса становится не простой задачей.

Сессии через какое-то время должны истекать и удаляться сборщиком мусора. В *JWT* дата истечения срока может быть закодирована вместе с пользовательскими данными. Поэтому слой безопасности, проверяющий подлинность *JWT*, также может проверять время истечения срока действия и просто отказывать в доступе, если токен истёк.

Claims в *JWT* кодируются как объект *JSON*, который используется как *payload* (полезная нагрузка) структуры *JSON Web Signature (JWS)* или как открытый текст структуры *JSON Web Encryption (JWE)* [1].

JWS – даёт только подпись и содержащиеся в ней данные (*claims*, так они названы в номенклатуре JWT) доступные для всех. *JWE* – предлагает шифрование, поэтому только кто-то с ключом может его расшифровать. *JWS* используется в большинство случаев, поэтому в этой статье речь пойдёт больше о нём.

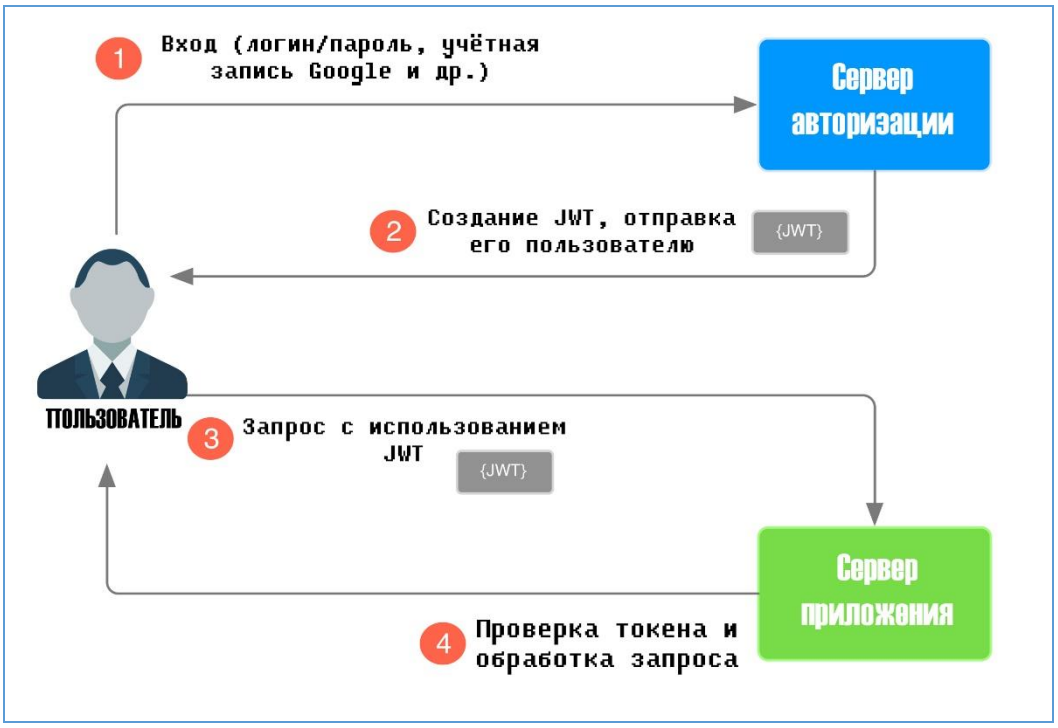

Рис. 1. Использование JWT

JWT состоит из:

header – информация о алгоритме подписи, тип полезной нагрузки, в формате *JSON*;

payload – полезные данные (или *claims*) в *JSON* формате;

signature – подпись [2].

Каждая часть, упомянутая выше кодируется *base64url*, затем они склеиваются вместе с разделителем ".". Формат выглядит так: *header.payload.signature*.

Процесс создания JWT

Заголовок *JWT* содержит информацию о том, как должна вычисляться *JWT* подпись. ***Header*** — это тоже *JSON* объект, который выглядит следующим образом:

header = {

"alg": "HS256",

"typ": "JWT"

}

Поле *alg* определяет алгоритм хэширования. Он будет использоваться при создании подписи. *HS256* – не что иное, как *HMAC-SHA256*. Также может использоваться алгоритм *RS256*. В отличие от предыдущего, он является ассиметричным и создает два ключа: публичный и приватный. С помощью приватного ключа создается подпись, а с помощью публичного только лишь проверяется подлинность подписи, поэтому нам не нужно беспокоиться о его безопасности. Параметр *alg* – обязательный.

Поле *typ* говорит о том, что данный заголовок относится к *JSON Web Token*.

Payload — это полезные данные, которые хранятся внутри *JWT*. Эти данные также называют *JWT-claims*. В нижеприведённом примере сервер аутентификации создает *JWT* с информацией о имени и параметром является ли пользователь админом.

payload = {

"name": "Ivan Ivanov",

"admin": true

}

В *payload* можно разместить что угодно, однако существует список, который успешно распознается большинством реализаций:

- *exp* – временная метка, указывающая, когда токен становится недействительным. В стандарте сказано, что текущая дата и время должна быть до указанного значения, чтобы разрешить обработку токена [2];
- *nbf* (*not before*) – временная метка, указывающая, когда токен становится действительным. Текущая дата и время должны быть больше и равны указанному значению, чтобы разрешить обработку токена [2];
- *iat* (*issued at*) – временная метка, указывающая, когда токен был выпущен [2].

Signature – рассчитывается для заголовка и полезной нагрузки за один раз. Таким образом их подлинность можно проверить за одну итерацию.

Алгоритм вычисления подписи выглядит так:

const SECRET_KEY = 'secret'

const unsignedToken = base64urlEncode(header) + '.' + base64urlEncode(payload)

const signature = HMAC-SHA256 (unsignedToken, SECRET_KEY)

Функция *base64urlEncode* кодирует заголовок и полезную нагрузку, созданные ранее. Алгоритм соединяет закодированные строки через точку. Затем полученная строка хешируется алгоритмом, заданным в заголовке на основе секретного ключа.

Полученный *JWT* токен должен выглядеть примерно так:

eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1c2VySWQiOiJiMDhmODZhZi0zNWRhLTQ4ZjItOGZhYi1jZWYzOTA0NjYwYmQifQ.zzPsBTVQeE9UALVfGZcCfL3I8EVnH2ZnWNVeqXLS1aA [3]

После этого этапа сервер должен отправить пользователю полученный токен.

Использование JWT в приложении и способы хранения

После того, как сервер возвращает токен на клиентской стороне его необходимо сохранить. Существуют разные подходы хранения токенов: *cookies, memory, localStorage*. При каждом запросе на защищенный ресурс добавляется токен. Он отправляется либо как *cookie,* либо как *Authorization* заголовок в *HTTP* запросе.

Cookies

Cookies – фрагмент данных, полученный от веб-приложения, которые хранятся на компьютере пользователя и отправляются с каждым запросом.

Их главный недостаток – они подвержены *CSRF*-атакам – это тип атаки, которая возникает, когда вредоносный веб-сайт, почта, блог, сообщение заставляют браузер пользователя выполнять нежелательно действие на доверенном сайте, где пользователь прошёл аутентификацию [4].

Успешная *CSRF* может привести к переводу средств, изменению пароля или покупке различных предметов под пользователем, на которого направлена атака. Злоумышленники не знают целевого пользователя до тех пор, пока неавторизованная транзакция не будет проведена. В случае, когда целевой пользователь является администратором, то атака может скомпрометировать всё веб-приложение.

Методы социального инжиниринга применяются часто для воспроизведения данной атаки. Целевому пользователю отправляется ссылка или фишинговый сайт, выглядящий идентично сайтам, на похищение данных с которого и направлена атака.

При переходе пользователя на фишинговый сайт от его имени скрытым образом отправляется запрос на ресурс, похищение данных с которого и является конечной задачей. В случае наличия авторизации пользователя на данном сайте у него имеется активная сессия в *cookie*. В таком случае запрос выполняется незаметно для пользователя. На рисунке 2 представлена схема атаки.

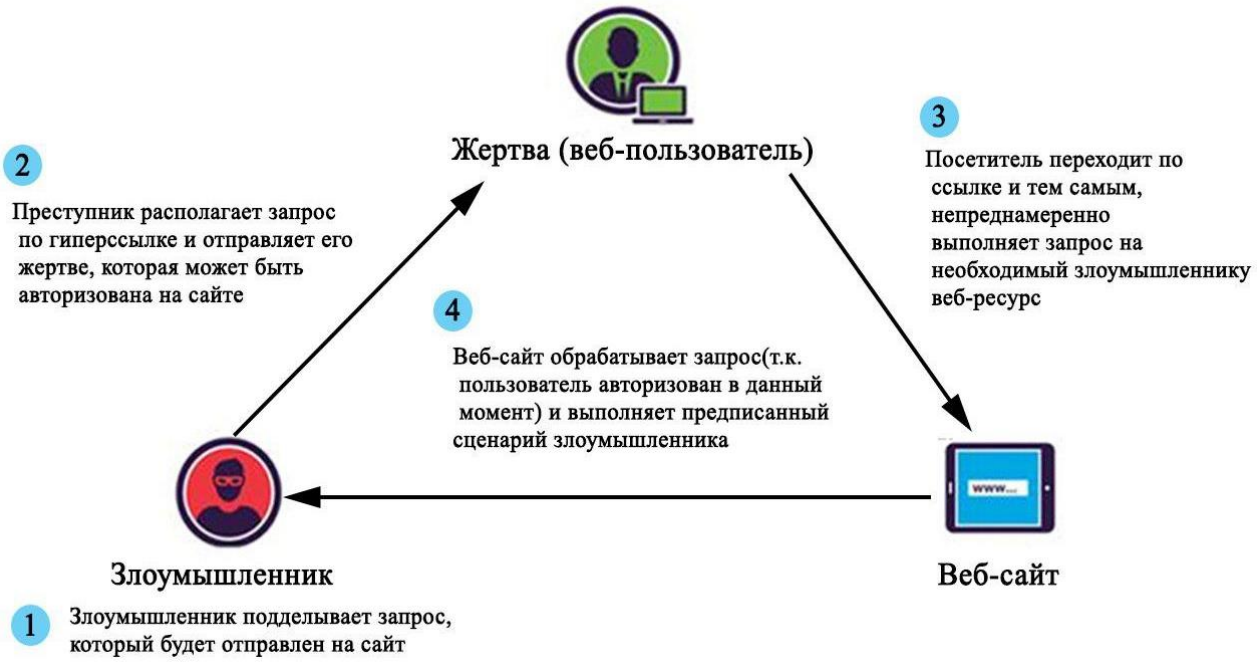

Рис. 2. Схема CSRF-атак

HTML Web Storage

При помощи веб-хранилища веб-приложения могут хранить данные локально в браузере пользователя.

Веб-хранилище можно рассматривать упрощенно как усовершенствование файлов *cookie*, обеспечивая гораздо большую емкость хранилища. Максимальный размер значения, хранимого в *localStorage* – 5120KB. Это дает гораздо больше возможностей для работы, чем обычный *cookie*, максимальный размер которого – 4 КБ.

Веб-хранилище – на весь домен. Все страницы из одного источника могут хранить и иметь доступ к одинаковым данным.

Главный недостаток *localStorage* в том, что веб-хранилище подвержено *XSS* атакам.

Cross-site scripting (*XSS*, иногда используется и сокращение *CSS*) – это тип инъекции, в которой вредоносные скрипты вводятся в другие надежные сайты [5]. Атака *XSS* происходит, когда злоумышленник использует веб-приложение для отправки вредоносного кода, скрипта на стороне браузера, другому пользователю. Провести *XSS*-атаку возможно в веб-приложениях, которые использует входные данные от пользователя в том виде, в каком они приходят, без проверки и кодирования. Браузер конечного пользователя не подозревает, что запускаемому скрипту нельзя доверять и сценарий будет выполнен, т.к. он пришёл из надежного источника. По этой причине вредоносный код может получить доступ к *cookies*, токенам или другой информацию, которая сохраняется браузером и используется на сайте.

XSS-уязвимости можно разделить на два типа: рефлективные и хранимые.

Рефлективным называются такие атаки, где введённый скрипт отражается от веб-сервера в сообщении об ошибке или результатах поиска, которые включают в себя данные, отправленные на сервер в качестве части запроса. Когда пользователь переходит по вредоносной ссылке, внедрённый код перемещается на уязвимый веб-сайт, который отражает атаку обратно в браузер пользователя. Браузер выполняет код, т.к. он пришёл с надёжного источника.

К хранимым атакам относятся те, в которых инъецируемый скрипт постоянно хранится на целевых серверах. В базе данных, форуме сообщений, журнале посетителей, в поле комментариев. Схема такой атаки показана на рисунке ниже. (см. Рис. 3)

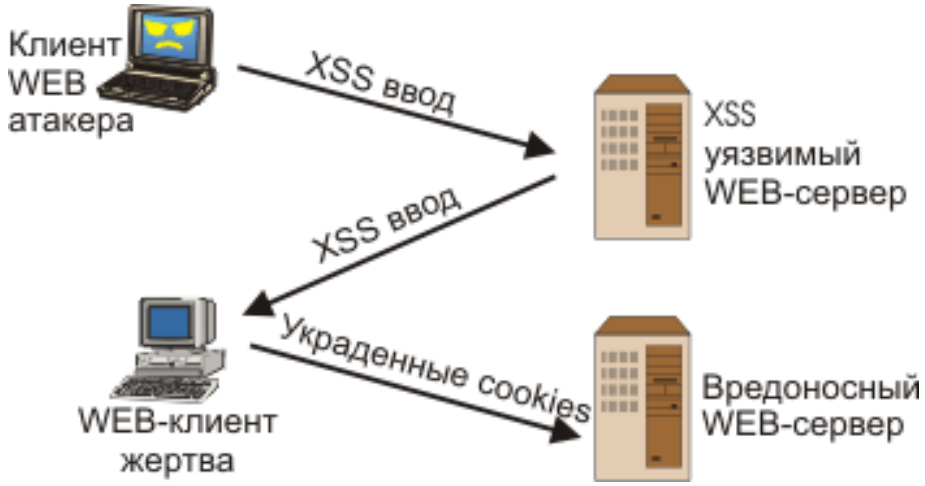

Рис. 3. Схема XSS-атаки

Bearer token

Bearer token – это защищенный токен. Любой пользователь, имеющий ключ, может получить доступ к защищенным ресурсам [6].

При успешной аутентификации на сервере клиент получает ключ, который генерирует сервер. В ключ может добавлена различная полезная нагрузка, т.е. информация о пользователе. Ключ сохраняется в *localStorage* при помощи клиентской части приложения. Каждый раз, когда пользователь выполняет запрос к ресурсу, который защищен, клиентская часть приложения добавляет в заголовок ключ, который получает из *localStorage*.

Authorization: Bearer <token> или *Authorization: Token <token>*.

На серверной части осуществляется поиск ключевого слово *Token* или *Bearer,* если оно находится, то из заголовка извлекается только переданный токен. Он расшифровывается и проверяется его действительность. Если операция выполнена успешно, то пользователь получает доступ к запрашиваемому ресурсу. Операция проверки действительности указанного токена выполняется каждый раз, когда пользователь пытается получить доступ к защищенным ресурсам.

Заключение

В результате работы проведен анализ, рассмотрены преимущества и недостатки технологии предоставления доступа – *JWT*.

При выборе подхода к реализации аутентификации необходимо в первую очередь ориентироваться на задачу, проводить анализ возможных решений, возможных недостатков как в удобстве использования для пользователей, так и в вопросах безопасности.

Также некоторые компании делают в одном проекте несколько вариантов аутентификации. Подобный подход может компенсировать недостатки одной технологии преимуществами другой. Однако для таких задач требуется большее количество времени на анализ, проектирование, тестирование безопасности, разработку, а также хороший уровень понимания работы нескольких подходов. Всё это повышает требуемый уровень квалификации сотрудников.

Список литературы

1. Jones M., Bradley J., Sakimura N. JSON Web Token (JWT). — [Электронный ресурс]. URL: https://tools.ietf.org/html/rfc7519.
2. Jones M., Bradley J., Sakimura N. JSON Web Signature (JWS). — [Электронный ресурс]. URL: https://tools.ietf.org/html/rfc7515.
3. JSON Web Tokens (JWT) in Auth0. — [Электронный ресурс]. URL: https://auth0.com/docs/jwt.
4. Wichers D., Petefish P., Sheridan E., Righetto D. Cross-Site Request Forgery (CSRF) Prevention Cheat Sheet. — [Электронный ресурс]. URL: https://www.owasp.org/index.php/Cross-Site_Request_Forgery_(CSRF)_Prevention_Cheat_Sheet.
5. Cross-site Scripting (XSS). — [Электронный ресурс]. URL: https://www.owasp.org/index.php/Cross-site_Scripting_(XSS).
6. Jones M., Hardt D. The OAuth 2.0 Authorization Framework: Bearer Token Usage. — [Электронный ресурс]. URL: https://tools.ietf.org/html/rfc6750.