Министерство образования и науки Российской Федерации

Федеральное государственное автономное образовательное учреждение

высшего образования «Уральский федеральный

университет имени первого президента России Б. Н. Ельцина»

Институт фундаментального образования

Кафедра интеллектуальных информационных технологий

**JWT - авторизация через токены**

Пояснительная записка

|  |  |
| --- | --- |
| Руководитель: | Свинцов Д.М. |
| Студент гр. ФО-350005: | Акифьев В.Л. |

Екатеринбург 2017

Оглавление

[ВВЕДЕНИЕ 3](#_Toc501129680)

[Структура JWT 4](#_Toc501129681)

[Заголовок (header) 4](#_Toc501129682)

[Набор полей (нагрузка) 4](#_Toc501129683)

[Токены 6](#_Toc501129684)

[Схема создания/использования токенов: 7](#_Toc501129685)

[Схема рефреша токенов 8](#_Toc501129686)

[В случае кражи(обоих такенов) 8](#_Toc501129687)

[ЗАКЛЮЧЕНИЕ 10](#_Toc501129688)

# ВВЕДЕНИЕ

Аутентификация - это процесс проверки учётных данных пользователя (логин/пароль). Проверка подлинности пользователя путём сравнения введённого им пароля с паролем, сохранённым в базе данных пользователей;

Авторизация - это проверка прав пользователя на доступ к определенным ресурсам.

JSON Web Token (далее JWT) - открытый стандарт RFC 7519, который определяет компактный и информационно полный способ передавать информацию защищенно в виде JSON объекта. JWT может быть подписан как секретной строкой, используя алгоритм HMAC, так и парой ключей (публичный/приватный) используя алгоритм RSA.

# Структура JWT

Структура JWT:

1. Заголовок(header)
2. Набор полей(payload)
3. Сигнатура

## Заголовок (header)

Заголовок состоит из 2х частей: вид токена (JWT), и алгоритм, который используется в подписи. Для примера:

{

"alg": "RS256",

"typ": "JWT"

}

Затем, этот JSON подвергается Base64UrlEncode для формирования первой составляющей JWT.

## Набор полей (нагрузка)

Нагрузка, вторая часть токена, содержит данные. Данные эти являются по сути различными фактами/полями о какой-либо сущности (например, о пользователе) и могут в себя включать дополнительные мета-данные. Существует 3 вида данных в нагрузке: зарезервированные, публичные и частные.

Зарезервированные данные не обязательны для заполнения, однако предоставляют целый набор полезной информации общего назначения. Например, iss (кто выдал токен), exp (токен годен до).

Публичные данные могут быть определены по желанию тех, кто использует JWT. Но, чтобы избежать столкновений они должны быть определены в IANA JSON реестре веб-токенов или быть определены как URI, который содержит устойчивое к коллизиям пространство имен.

Частные данные предоставляются для обмена информацией между собеседниками, которые имплементируют чтение/запись этой информации в необходимом формате.

Пример частных данных в нагрузке:

{

"id": "1234567890",

"name": "John Doe",

"admin": true

}

Затем, этот JSON подвергается Base64UrlEncode для формирования второй составляющей JWT.

###### Подпись

Чтобы создать часть JWT с подписью, необходимо взять Base64UrlEncode заголовок, Base64UrlEncode нагрузку, ключи или пароль и подписать это все с помощью алгоритма указанного в заголовке.

* { «alg»: «HS256», «typ»: «JWT» }.{ «iss»: «auth.myservice.com», «aud»: «myservice.com», «exp»: «1435937883», «userName»: «John Smith», «userRole»: «Admin» }.S9Zs/8/uEGGTVVtLggFTizCsMtwOJnRhjaQ2BMUQhcY

Затем, эта подпись подвергается Base64UrlEncode для формирования третьей составляющей JWT.

Подпись используется для проверки того, что отправитель JWT, тот, кто говорит, что это он, и для того, чтобы можно было гарантировать, что сообщение не было изменено по пути следования между собеседниками.

# Токены

Токены предоставляют собой средство авторизации для каждого запроса от клиента к серверу. Токены(и соответственно сигнатура токена) генерируются на сервере основываясь на секретном ключе(который хранится на сервере) и payload'e. Токен в итоге хранится на клиенте и используется при необходимости авторизации какого либо запроса. Такое решение отлично подходит при разаработке SPA.

При попытке хакером подменить данные в header'ре или payload'е, токен cтанет не валидным, поскольку сигнатура не будет соответствовать изначальным значениям. А возможность сгенерировать новую сигнатуру у хакера отсутствует, поскольку секретный ключ для зашифровки лежит на сервере.

access token - используется для авторизации запросов и хранения дополнительной информации о пользователе (аля user\_id, user\_role или еще что либо, эту информацию также называет payload).

refresh token - выдается сервером по результам успешной аутентификации и используется для получения нового access token'a и обновления refresh token'a.

Каждый токен имеет свой срок жизни, например access: 30мин, refresh: 60 дней.

Поскольку токены это не зашифрованная информация крайне не рекомендуется хранить в них такую информацию как пароли.

Схема создания/использования токенов:

1. Пользователь логинится в приложении, передавая логин/пароль на сервер
2. Сервер проверят подлинность логина/пароля, в случае удачи генерирует и отправляет клиенту два токена(access, refresh) и время смерти access token'а (expires\_in поле, в unix timestamp). Также в payload refresh token'a добавляется user\_id

"accessToken": "...",

"refreshToken": "...",

"expires\_in": 1502305985425

1. Клиент сохраняет токены и время смерти access token'а, используя access token для последующей авторизации запросов
2. Перед каждым запросом клиент предварительно проверяет время жизни access token'а (из expires\_in)и если оно истекло использует refresh token чтобы обновить ОБА токена и продолжает использовать новый access token

Ключевой момент что в момент рефреша то есть обновления access token'a обновляются ОБА токена. Но как же refresh token может сам себя обновить, он ведь создается только после успешной аунтефикации ? refresh token в момент рефреша сравнивает себя с тем refresh token'ом который лежит в БД и вслучае успеха, а также если у него не истек срок, система рефрешит токены. Внимание при обновлении refresh token продливается также и его срок жизни.

Возникает вопрос зачем refresh token'y срок жизни, если он обновляется каждый раз при обновлении access token'a ? Это сделано на случай если юзер будет в офлайне более 60 дней, тогда придется заново вбить логин/пароль.

С такой схемой юзер сможет быть залогинен только на одном устройстве. Тоесть в любом случае при смене устройства ему придется логинется заново.

# Схема рефреша токенов

1. Клиент проверяет перед запросом не истекло ли время жизни access token'на
2. И если истекло клиент отправляет на auth/refresh-token URL refresh token
3. Сервер берет user\_id из payload'a refresh token'a по нему ищет в БД запись данного юзера и достает из него refresh token
4. Сравнивает refresh token клиента с refresh token'ом найденным в БД
5. Проверяет валидность и срок действия refresh token'а
6. В случае успеха сервер:
   1. Пересоздает и записывает refresh token в БД
   2. Создает новый access token
   3. Отправляет оба токена и новый expires\_in access token'а клиенту
7. Клиент повторяет запрос к API c новым access token'ом

# В случае кражи(обоих такенов)

1. Хакер воспользовался **access token'ом**
2. Закончилось время жизни **access token'на**
3. **Клиент хакера** отправляет **refresh token**
4. Хакер получает новую пару токенов
5. На сервере создается новая пара токенов(**"от хакера"**)
6. Юзер пробует зайти на сервер >> обнаруживается что токены невалидны
7. Сервер перенаправляет юзера на форму аутентификации
8. Юзер вводит логин/пароль
9. Создается новая пара токенов >> пара токенов **"от хакера"** становится не валидна

**Проблема:** Поскольку **refresh token** продлевает срок своей жизни каждый раз при рефреше токенов >> хакер пользуется токенами до тех пор пока юзер не залогинится.

# ЗАКЛЮЧЕНИЕ

Мы прошлись по тому, что такое JWT, как они создаются и как валидируются, каким образом они могут быть использованы для установления доверительных отношений между пользователем и приложением. Но это лишь кусочек пазла большой темы авторизации и обеспечения защиты приложения. **СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

1. https://habrahabr.ru/post/340146/https://docs.pylonsproject.org/projects/webob/en/stable/
2. https://gist.github.com/zmts/802dc9c3510d79fd40f9dc38a12bccfchttps://docs.pylonsproject.org/projects/webob/en/stable/wiki-example.html
3. https://blog.indev-group.eu/json-web-token-api-authorization/
4. https://auth0.com/blog/cookies-vs-tokens-definitive-guide/