COMPUTER SCIENCE

STATELESS АВТОРИЗАЦИЯ С ИСПОЛЬЗОВАНИЕМ JWT

[1]***Бакунова Оксана Михайловна,***
[2]***Корзун Алексей Андреевич,***
[2]***Колосенко Николай Сергеевич,***
[3]***Малиновская Татьяна Ивановна***

Республика Беларусь, БГУИР,
[1]*старший преподаватель, исследователь технических наук, магистр технических наук;*
[2]*студент;*
[3]*магистр технических наук*

DOI: https://doi.org/10.31435/rsglobal_wos/12062018/5733

ARTICLE INFO

Received: 13 April 2018
Accepted: 21 May 2018
Published: 12 June 2018

KEYWORDS

stateless authorization,
web,
API,
JSON,
performance

ABSTRACT

A lot of web-developers have or will encounter the problem of authorization in their projects. Classic authorization scheme involves generating a random string (named token), then storing it with a set of attached privileges. But this approach requires a centralized storage for tokens, which all of project's computing systems will be accessing. In the process of project's growth, sooner or later, this centralized storage will no longer be able of dealing with the given amount of tokens and will require a serious upgrade. However, there are authorization solutions that don't require storing a generated token out there. We are going to take a look at the one that is open standard and is represented as a simple JSON object. An important aspect of this solution is that it also doesn't require a very powerful machine to process even hundreds of thousands requests per second.

Citation: Бакунова О. М., Корзун А. А., Колосенко Н. С., Малиновская Т. И. (2018) Stateless авторизация с использованием JWT. *Web of Scholar. 6(24),* Vol.1. doi: 10.31435/rsglobal_wos/12062018/5733

Введение. При разработке проектов, рассчитанных на работу под высокой нагрузкой, разработчики неминуемо сталкиваются с необходимостью обеспечения горизонтального масштабирования вычислительных систем. Одной из критических частей подобных проектов является проверка прав доступа клиента к ресурсу (авторизация). Классическая система авторизации предполагает генерацию случайной строки, именуемой токеном, к которой привязывается список выданных прав, и её хранение в централизованном хранилище, к которому обращаются все вычислительные подсистемы. По мере увеличения нагрузки на вычислительную систему и её масштабирования такой подход приводит к необходимости масштабирования и самого хранилища токенов. Однако существуют решения, позволяющие миновать необходимость хранения выданных токенов. Одним из таких решений является стандарт JWT.

Результаты и обсуждение. JSON Web Token (JWT) — открытый стандарт, который определяет компактный и самодостаточный способ передачи информации между запросами в

виде JSON объекта. Вся необходимая информация хранится в самом токене и не может быть изменена третьей стороной [1].

Структура JWT состоит из 3-х частей: header, payload и signature. Header и payload представляют собой JSON объекты, закодированные алгоритмом base64. Signature же является результатом хеш-функции, применённой к header и payload. Все 3 части объединяются друг с другом посредством точек в фиксированной последовательности: header, payload и signature. Информация, сохранённая в токене, может быть прочитана из него самого при помощи обратного преобразования из base64 и чтения JSON документа.

Header обычно содержит два значения: тип токена и алгоритм подписи. Соответствующие поля называются typ и alg. Header служит для определения характеристик payload и signature. Он не ограничивается только этими значениями и может содержать любую дополнительную информацию, в зависимости от специфики вашего приложения. Например, можно включить информацию о версии самого токена, для того, чтобы знать, из каких полей необходимо читать значения. Пример содержимого header представлен на рисунке 1.

```
{
  "alg": "HS256",
  "typ": "JWT"
}
```

Рис. 1. Пример содержимого JWT header

Payload содержит перечень claims о сущности самого токена, а также любую дополнительную метаинформацию. Выделяют 3 типа значений: зарегистрированные, публичные и приватные. Зарегистрированными называют определённый в спецификации перечень claims. Всего существует 7 подобных полей:

- iss — определяет выпускателя токена;
- sub — позволяет определить сущности, к которым предоставляется доступ. Например, идентификатор пользователя;
- aud — определяет получателя токена;
- exp — задаёт время истечения токена;
- nbf — задаёт время, начиная с которого токен вступает в свои полномочия;
- iat — указывает время, в которое токен был выписан;
- jti — уникальный идентификатор токена.

Публичными называют поля, перечисленные в регистре IANA [2]. Это стандартизированные имена для наиболее часто используемых понятий. Например, таких как псевдоним пользователя, ссылка на профиль, предпочтительный язык, различные дополнительные хеш-значения и т.д.

Приватными же называют все остальные значения, не определённые в стандарте. Здесь можно расположить список выданных прав, дополнительную информацию о пользователе или приложении, которому выдан токен, а также любые другие поля, соответствующие специфике конкретного приложения. Пример содержимого payload представлен на рисунке 2.

```
{
  "sub": "1234567890",
  "name": "John Doe",
  "iat": 1516239022
}
```

Рис. 2. Пример содержимого JWT payload

Signature формируется путём объединения сформированных header и payload и дальнейшим вычислением хеша этой строки при помощи выбранного алгоритма и секретного ключа. В качестве подписи предлагается использовать один из следующих алгоритмов: HMAC, RSA или ECDSA. Алгоритм HMAC является симметричным, т.е. ключ представляет из себя обычную строку. При использовании этого алгоритма проверить правильность подписи может только тот, кто её создал. RSA и ECDSA являются асимметричными алгоритмами и разделяются на публичный и приватный ключ. Подпись формируется приватным ключом, а публичный ключ может быть использован для проверки корректности содержимого любым желающим. Это может быть особенно полезно, когда поставщик токенов не централизован и каждый из участников информационной системы может создавать токены от своего имени. В таком случае каждый поставщик токенов имеет свой секретный ключ и каждый прочий участник может с лёгкостью убедиться в том, что издатель и заявитель токена не подделаны. Стоит понимать, что использование алгоритмов RSA и ECDSA для простых систем с одним источником токенов, который их же и проверяет, излишне. В этом случае использование алгоритма HMAC будет более целесообразным.

Все перечисленные выше алгоритмы формируют signature таким образом, что изменение даже одного символа в payload приведёт к полному изменению итоговой signature, что исключает возможность подделки значений токена третьей стороной. Таким образом, signature является гарантом того, что все значения токена не были изменены и что токен создан тем, кто заявлен подписчиком в payload.

Пример полностью сформированного JWT представлен на рисунке 3.

Рис. 3. Пример полностью сформированного JWT

На сегодняшний день JWT широко применяется в качестве токена доступа в протоколе OAuth2 или просто при построении web-приложений по принципу Single Page Application. Кроме того, существует надстройка над стандартом OAuth2 — OpenID Connect. Этот протокол был представлен ещё в 2014 году и использовал на тот момент ещё черновую версию спецификации JWT. Тем не менее, JWT используется для обмена ключами и непосредственного формирования токена доступа [3]. Такие компании, как Google, Microsoft и Steam используют этот протокол для авторизации своих пользователей для внешних сайтов (знакомая вам кнопка «Войти через Google»). Пример содержимого access token Google представлен на рисунке 4.

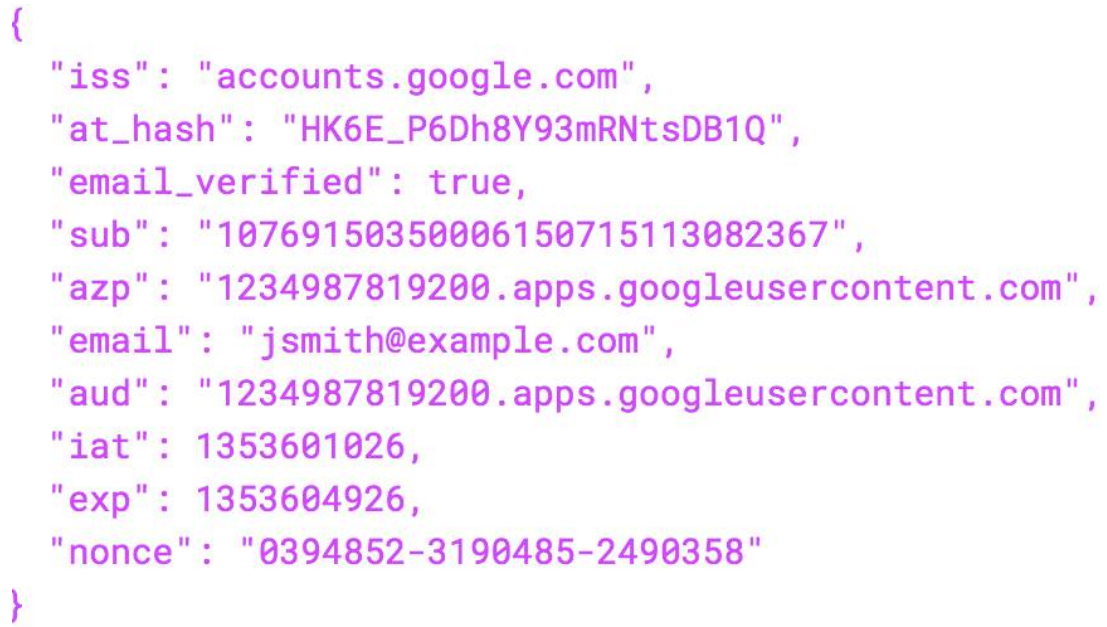

Рис. 4. Пример JWT payload Google OpenID Connect

Google использует только зарегистрированные и публичные поля, определённые в стандарте OpenID Connect. В поле aud хранится информация о приложении, которому был выдан доступ, поле sub указывает на уникальный идентификатор пользователя, а в поле exp задано время истечения токена. По истечению этого времени токен станет невалиден и его будет необходимо обновить. Впоследствии, передавая данный токен в каждом запросе, вы сможете выполнять операции от имени пользователя, указанного в sub. При обработке запроса Google будет достаточно проверить подпись и метку истечения, после чего можно всецело довериться полученным данным. Нет нужды выполнять дополнительных операций ввода-вывода ко внешним сервисам, вроде базы данных, для получения информации о содержимом токена.

Введение поля, отвечающего за время жизни токена, обусловлено необходимостью решения задачи контроля за выданными токенами и прекращения доступа определённого пользователя или приложения. Время жизни обычно не превышает 1 часа. Зачастую при этом выдаётся специальный refresh token, который используется для запроса нового access token, что позволяет обращаться к базе данных лишь раз в час вместо обращения на каждый запрос. Однако при таком подходе невозможно выполнить отзыв токенов моментально, т.к. они будут ещё иметь какой-то период жизни.

Для реализации возможности моментального отзыва рассмотрим следующую схему. Предположим, что мы выступаем в качестве OAuth2 провайдера, т.е. издаём токены для сторонних приложений. Основными сущностями являются пользователи и приложения. Приложения, после процесса авторизации, получают возможность выполнять действия от имени пользователя. Система должна позволять моментально инвалидировать все токены в системе, а также отдельно для каждого приложения, пользователя или их связки. Для реализации поставленной задачи нам всё же придётся прибегнуть к какому-то хранилищу, но вместо хранения всех выданных токенов мы пойдём от обратного и будем записывать только критерии отзыва. В качестве хранилища возьмём любую быструю key-value базу данных. Пусть это будет Redis, который способен выполнять сотни тысяч операций чтения в секунду даже на довольно слабом оборудовании. При необходимости инвалидировать токены по какому-то критерию, мы формируем запись в хеш-таблицу Redis. В качестве ключа выступит вычислимое выражение, определяющее группу, на которую должен быть применён инвалидатор. Например, для инвалидации всех токенов пользователя с идентификатором 1 по отношению к приложению test-app мы используем ключ *user:1:app:test-app*. Значением указываем текущее время. При обработке входящего запроса, после проверки токена по базовым критериям, выполняем запрос в Redis с помощью функции HMGET по следующим ключам: *global*, *user:{sub}*, *app:{aud}* и *user:{sub}:app:{aud}*, где имена в фигурных скобках соответствуют таковым в payload токена. Затем разбираем ненулевые результаты. Если хотя бы у одного из полученных результатов значение времени окажется больше, чем заявлено в iat токена, то он был инвалидирован, а значит должен быть отклонён.

Данный компромисс обеспечивает константное время O(4) для операции доступа к базе данных, а хранение только метки времени позволит хранить миллионы подобных записей с минимальными затратами оперативной памяти. Кроме того, такой подход может позволить и вовсе отказаться от ограничения на время жизни токена и дополнительных механизмов обновления access token, т.к. теперь мы можем отозвать любой токен в любой момент.

Выводы. Таким образом получается компактный, простой и полностью самодостаточный токен, который можно использовать как для процессов аутентификации пользователей, так и для авторизации приложений или собственных сервисов. Все данные хранятся в самом токене и не могут быть изменены. Соответственно, для восстановления информации о пользователе и списке авторизованных прав нет нужды заводить отдельное хранилище. Данный стандарт нашёл широкое применение в различных областях и получил поддержку таких крупных компаний, как Google, Twilio и Auth0. Он показывает себя одинаково хорошо, как в небольших проектах, так и в огромных распределённых системах, где участниками могут выступать как обычные пользователи, так и сами сервисы, выполняющие запросы друг к другу в рамках микросервисной архитектуры. Существует огромное количество готовых программных реализаций для создания и проверки токенов под все популярные языки программирования. JWT может применяться не только для авторизации к web API, но и для разового обмена информацией или даже для взаимодействия систем, работающих вне протокола HTTP.

ЛИТЕРАТУРА

1. M. Jones, J. Bradley, N. Sakimura. RFC 7519 – JSON Web Token (JWT) – IETF, 2015. – [Электронный ресурс]. URL: https://tools.ietf.org/html/rfc7519
2. John Bradley, Brian Campbell, Michael B. Jones, Chuck Mortimore. JSON Web Token Claims – IANA, 2015. – [Электронный ресурс]. URL: https://www.iana.org/assignments/jwt/jwt.xhtml
3. Nat Sakimura, John Bradley, Michael B. Jones, Breno de Medeiros, Chuck Mortimore. OpenID Connect Core 1.0 incorporating errata set 1 – 2014. – [Электронный ресурс]. URL: http://openid.net/specs/openid-connect-core-1_0.html