УДК 004

М. З. МУЛЛАНУРОВ
mullanurov.marat@mail.ru
Науч. руководитель – д-р техн. наук, проф. Г. Р. ВОРОБЬЕВА

Уфимский государственный авиационный технический университет

СРАВНИТЕЛЬНЫЙ АНАЛИЗ МЕТОДОВ АУТЕНТИФИКАЦИИ В ВЕБ-ПРИЛОЖЕНИЯХ

Аннотация. В данной статье представлен сравнительный анализ распространенных методов аутентификации в веб-приложениях: базовой аутентификации, аутентификации с помощью сессий и аутентификации с помощью JWT, рассмотрены их преимущества и недостатки.
***Ключевые слова*:** аутентификация; авторизация; сессии; JWT; веб-технологии, веб-безопасность.

Введение

В современном интернете трудно представить серьезный сайт, который бы не реализовывал механизмов аутентификации и авторизации запросов пользователей, которых за все время существования всемирной паутины было придумано большое множество, со своими преимуществами и недостатками. Поэтому проблема выбора подходящего метода аутентификации под конкретный проект является актуальной.

Существует путаница между понятиями идентификация, аутентификация и авторизация. Этими понятиями обозначают разные процессы, хотя они чаще всего происходят вместе и выполняют одну задачу – определяют, кто может получить доступ к определенным информационным ресурсам:

1. Идентификация – определение того, какой пользователь делает запрос, для этого он вводит, например, свой логин или email;

2. Аутентификация – проверка подлинности пользователя, для этого он вводит, например, свой пароль;

3. Авторизация – предоставление пользователю прав на определенные ресурсы.

Так как эти понятия тесно связаны, в данной статье будет использоваться в основном только слово аутентификация.

Рассмотрим распространенные методы аутентификации:

Базовая аутентификация

При базовой аутентификации пользователь отправляет свой логин и пароль, закодированные с помощью base64, в заголовке HTTP-запроса Authorization, сервер проверяет подлинность пользователя и предоставляет ему права на ресурс [1].

Схема базовой аутентификации представлена на рисунке (рис. 1).

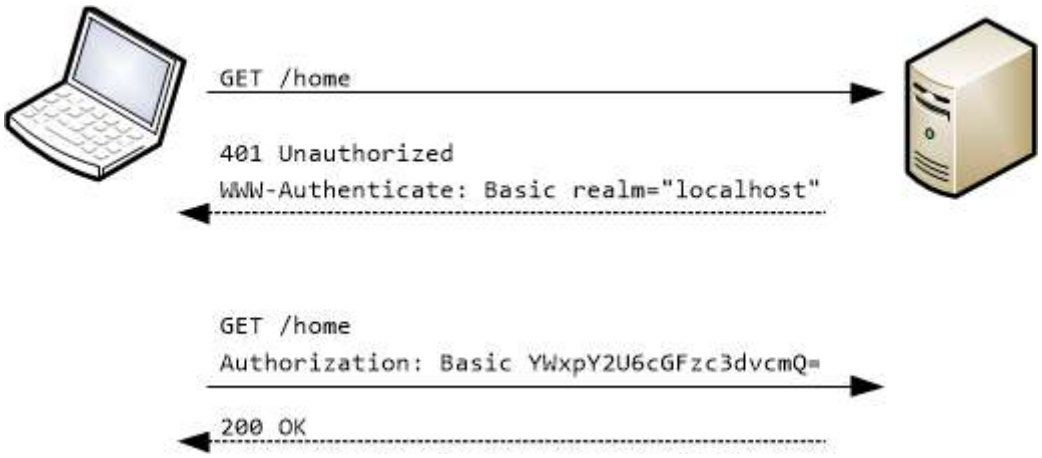

Рис. 1. Схема базовой аутентификации

Эта схема просто реализуется, так как не требует пересылки и получения дополнительной информации, вроде cookie, идентификаторов сессии, токенов и т.д. Также, так как это был один из первых видов аутентификации, его поддержка имеется практически во всех браузерах.

Однако у этой схемы имеется существенный недостаток – любой злоумышленник, если он сможет перехватить HTTP-запрос, сможет узнать учетные данные пользователя и выполнять запросы от его имени, что ограничивает применение данного вида аутентификации.

Данный метод можно применять, например, для прототипов веб-приложений, когда еще не реализован более безопасный способ аутентификации, однако крайне не рекомендуется применять его для серьезных проектов.

Аутентификация с помощью сессий

Для того, чтобы не отправлять при каждом запросе логин и пароль, были

описаны методы, которые отправляют пользователю какие-либо зашифрованные данные, которые пользователь должен будет отправлять на сервер при каждом запросе, а сервер будет проверять каким-либо образом, выдавал ли он эти данные, либо нет. Одним из таких методов являются сессии.

Аутентификация с помощью сессий работает следующим образом [2]:

1. Пользователь вводит логин и пароль и отправляет их на сервер;

2. Сервер проверяет подлинность введенных данных и создает сессию – запись в базе данных, и отправляет клиенту идентификатор сессии;

3. Клиент сохраняет у себя идентификатор сессии, обычно, в cookies;

4. При каждом последующем запросе клиент будет отсылать идентификатор сессии, который будет проверяться на сервере (будет искаться соответствующая запись в базе данных);

5. При разлогинивании, истечении времени жизни сессии или удалении сессии из базы вручную, пользователю придется снова вводить логин и пароль.

Схема базовой аутентификации по сессиям представлена на рисунке (рис. 2) [2].

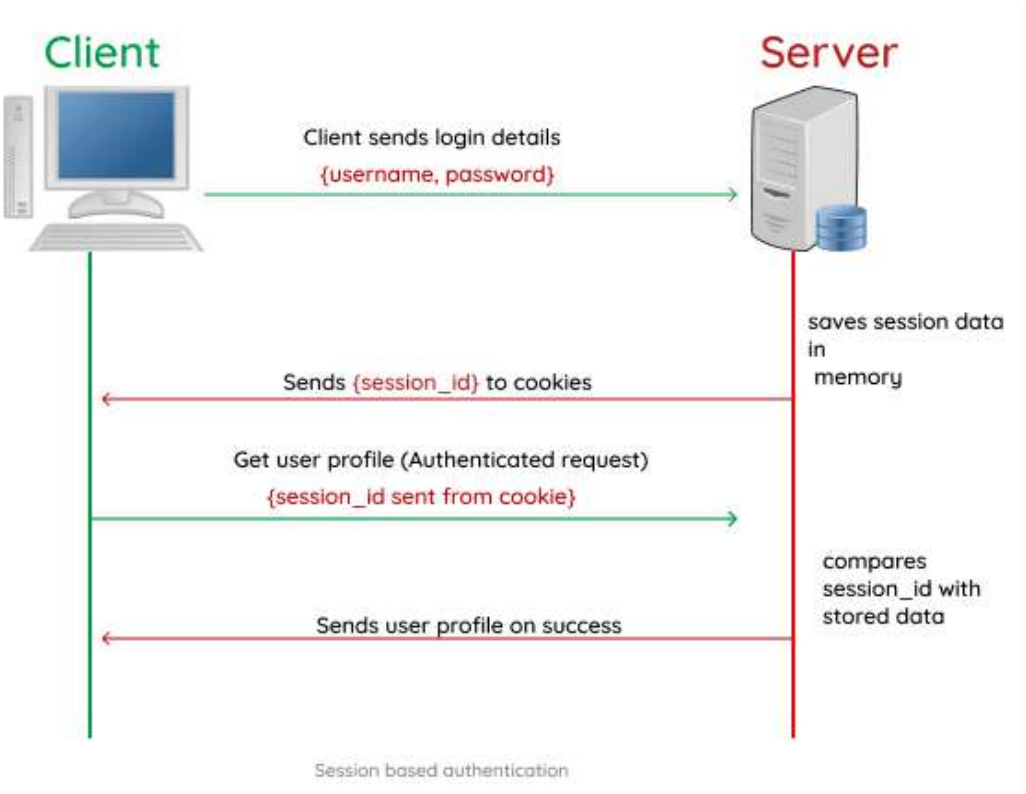

Рис. 2. Аутентификации по сессиям

Этот метод относительно просто реализуется, его поддерживает большинство современных фреймворков, он предоставляет довольно хороший уровень безопасности.

В случае кражи сессии, злоумышленник получает доступ к аккаунту пользователя, однако пользователю достаточно заново ввести логин и пароль, чтобы сессия стерлась из базы данных на сервере, и злоумышленник потеряет доступ.

Недостатком является хранение сессий в базе данных, что уменьшает производительность веб-сервера, из-за того, что при каждом запросе происходит обращение к базе данных, из-за чего она может стать «бутылочным горлышком» всей системы [3]. Кроме того, использование cookies может вести к уязвимостям к таким атакам, как CSRF и XSS.

Этот метод является более безопасным, чем базовая аутентификация и может использоваться в серьезных проектах, однако в очень крупных проектах с огромным количеством пользователей и запросов в секунду, использование базы данных для хранения сессий может стать узким местом системы.

Аутентификация с помощью JWT

Для того, чтобы решить главную проблему сессий – хранение данных для аутентификации на сервере – были придуманы JWT-токены, хранящие информацию о клиенте и подписанные секретным ключом сервера.

JWT (JSON Web Token) – компактный и безопасный способ передачи информации между двумя сторонами, использующий для представления данных формат JSON. Стандарт JWT представлен в RFC 7519 [4].

JWT состоит из трех частей, представляющих собой строки, закодированные в формате base64 и разделенные точкой:

– Заголовок (header) – данные в формате JSON, которые описывают тип токена и алгоритм шифрования;

– Полезная нагрузка (payload) – содержит данные, передающиеся между двумя устройствами. Представляет собой данные в формате JSON, в котором указаны параметры со своими значениями (claims);

– Подпись (signature) – содержит подпись, зашифрованную секретным ключом выдавшего токен.

На рисунке 3 [3] приведен пример JWT.

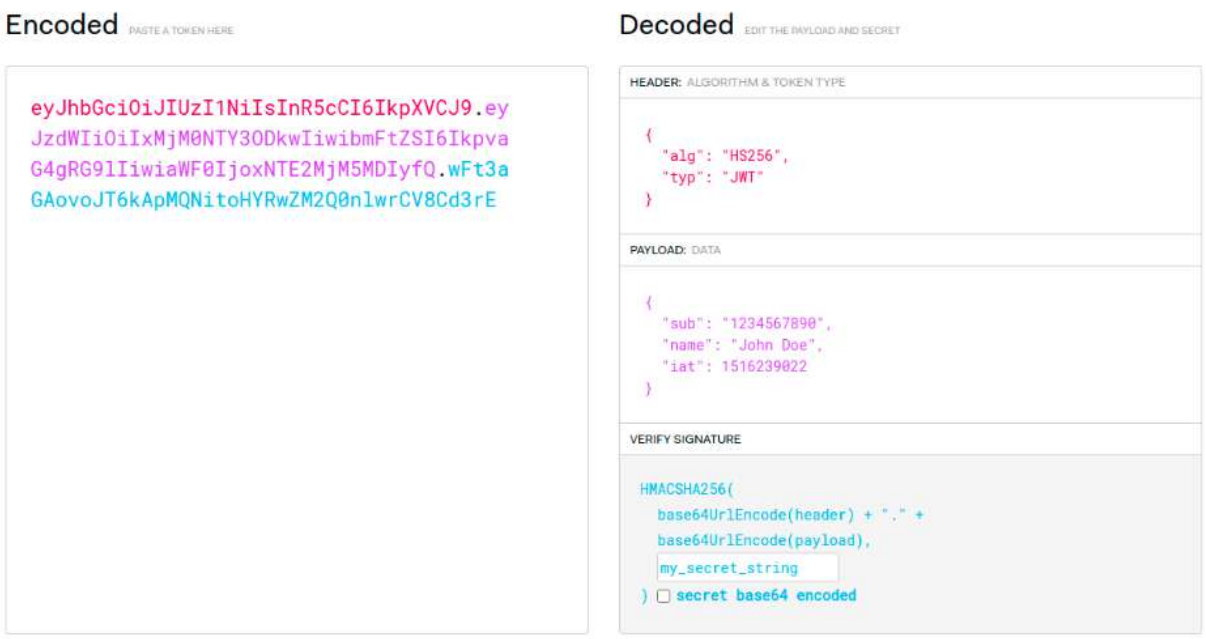

Рис. 3. Пример JWT

В приведенном выше примере в заголовке указан алгоритм шифрования (HS256) и тип токена (JWT).

В полезной нагрузке указаны данные для передачи между устройствами: sub – тема, name – имя и iat – время, когда токен станет невалидным.

Подписью является заголовок и полезная нагрузка, разделенные точкой и закодированные в base64, зашифрованные с использованием алгоритма, указанного в заголовке и секретного ключа выдавшего токен.

Подпись позволяет защитить токен от подделки – если злоумышленник подменит какие-либо данные в заголовке или полезной нагрузке, подпись, вычисленная у получателя токена и подпись, записанная в токене, не совпадут.

Подделать токен злоумышленник сможет, только если узнает секретный ключ выдавшего токен.

Аутентификация с использованием JWT-токенов в веб-приложении выполняется следующим образом:

1. Пользователь отправляет логин и пароль на сервер;

2. Сервер проверяет подлинность пользователя и выдает два токена – access token с коротким временем жизни и refresh token с длинным;

3. Пользователь сохраняет у себя токены и при последующих запросах указывает access token в заголовке Authorization;

4. При истечении срока жизни токена доступа пользователь отправляет на сервер refresh token и получает новую пару токенов.

Таким образом на сервере не требуется хранить никаких токенов – он должен просто выписывать и проверять их, а храниться они будут у клиента. Таким образом уменьшается нагрузка на базу данных. Использование JWT позволяет выделить отдельный сервер для аутентификации, выписывающий токен, а другие серверы будут только проверять его, что упрощает построение архитектуры с микросервисами. Еще одним преимуществом является возможность аутентификации с помощью других программ, кроме веб-браузера, например, мобильных приложений, так как им достаточно отсылать токен вместе с HTTP-запросом в заголовке Authorization.

Однако у данного подхода имеются и недостатки. Основной из них – все еще сохраняется возможность кражи токена, благодаря которому злоумышленник сможет получить доступ к аккаунту пользователя. Наибольшую опасность представляет кража refresh token, так как срок его жизни зачастую исчисляется месяцами, тогда как у access token срок жизни обычно не превышает 10 минут.

Для того, чтобы предотвратить доступ злоумышленника после кражи refresh token, применяется подход, когда на сервере аутентификации действующие refresh token хранятся в базе данных. После выдачи нового, либо когда пользователь выходит из системы, старый стирается из базы данных. Благодаря

этому пользователь может прекратить доступ злоумышленника к аккаунту, заново введя логин и пароль. Такой подход не приводит к замедлению системы из-за частых обращений к базе данных, так как refresh token выдается после истечения срока жизни access token, который составляет обычно несколько минут.

Таким образом, благодаря масштабируемости, аутентификацию с помощью JWT можно применять для проектов любой сложности, однако нужно продумать систему безопасности, чтобы предотвратить кражу токенов или секретного ключа сервера.

СПИСОК ЛИТЕРАТУРЫ

1. HTTP аутентификация - HTTP | MDN [Электронный ресурс]. – Режим доступа: https://developer.mozilla.org/ru/docs/Web/HTTP/Authentication
2. What really is the difference between session and token based authentication - DEV Community [Электронный ресурс]. – Режим доступа: https://dev.to/thecodearcher/what-really-is-the-difference-between-session-and-token-based-authentication-2o39
3. Как устроен токен и что можно решить с JWT-авторизацией [Электронный ресурс]. – Режим доступа: https://highload.today/blogs/jwt-avtorizatsiya/
4. RFC 7519: JSON Web Token (JWT) [Электронный ресурс]. – Режим доступа: https://www.rfc-editor.org/rfc/rfc7519