Ты кто?

Авторизация в веб-приложениях

Почти во всех приложениях нужна авторизация, чтобы определить: кто шлет запросы и что ему можно. Существует три понятия:

1. **Идентификация** — проверка, что пользователь как-то представился и существует
2. **Аутентификация** — проверка верных данных пользователя для входа: логин, пароль
3. **Авторизация** — проверка, что пользователь имеет право использовать данный функционал

Классический сценарий: пользователь заходит на сайт и отправляет свои почту и пароль на сервер. Сервер:

1. Находит в базе данных аккаунт клиента по почте *(Идентификация)*
2. Сверяет пароль, отправленный клиентом, и тот, что в базе *(Аутентификация)*
3. Проверяет: забанен пользователь или нет *(Авторизация)*

Обычно весь этот процесс называют просто **авторизацией**.

Cookies

Самым часто используемым способом авторизации являются Cookie (куки), так как их поддержка встроена во все браузеры. Способ работает следующим образом:

1. Клиент отправляет запрос с данными, необходимыми для аутентификации, например, логин/пароль
2. Сервер проверяет данные и, если они верны, добавляет к ответу заголовок **Set-Cookie** с каким-то значением
3. Браузер «видит» этот заголовок и сохраняет его значение к себе в локальное хранилище
4. С каждым последующим запросом браузер отправляет заголовок Cookie с сохраненным ранее значением
5. Сервер видит этот заголовок и авторизует пользователя в сервисе — при условии, что значение заголовка Cookie соответствует тому, что сервер записал в свою базу при авторизации

Cookie являются сетевым стандартом и обладают жестко заданным набором дополнительных полей, которые передаются вместе со значением Cookie в заголовке. Браузер сам умеет доставать эти данные из строки.

Пример. Рассмотрим такую cookie:

Cookie: sessionid=ktsOneLove; Domain=.kts.studio; Expires=Thu, 30 Jun 2022 15:57:14 GMT; Secure; HttpOnly; Max-Age=31536000; Path=/; SameSite=Lax;

- **Cookie** — ключ заголовка
- **sessionid** — название cookie, а ktsOneLove — значение
- **Expires** — дата «протухания»
- **HttpOnly** — флаг, который означает, что браузер не должен давать доступ к этой cookie из JavaScript кода. Такой атрибут обычно ставят на cookie, которые содержат важную информацию (например, сессию). Если страница содержит xss-уязвимость, то злоумышленник не сможет получить данные этой cookie.
- **Max-Age** — время жизни
- **Domain** — на каких доменах можно использовать эту cookie. Если перед доменом стоит точка, то cookie действует на всех поддоменах этого сайта. Например .kts.studio будет действовать и на metaclass.kts.studio.
- **Path** — на каких относительных url-адресах сайта действует эта cookie. Если значение равно «**/**», то cookie отправляется на любые запросы к этому сайту. Например, если поставить path=/user, то при запросе kts.studio/user/list заголовок Cookie будет отправлен, а на kts.studio/product/list — не будет.
- **SameSite** — говорит браузеру, можно ли использовать эту cookie на других сайтах. У SameSite есть три режима:

1. *None* — cookie можно использовать на любом сайте
2. *Lax* — cookie можно передавать только для высокоуровневой навигации. Например, со ссылки на сайте, на котором эта cookie была установлена, выполнен переход на другой сайт
3. *Strict* — cookie можно использовать только на сайте, который ее установил

Одной из главной особенностей cookie является ее время жизни. Это время устанавливается в заголовке Set-Cookie, а браузер следит за временем и удаляет cookie из своего хранилища, если время жизни истекло. Установить это время можно с помощью атрибута Expires и Max-Age. Expires задает дату, а Max-Age задает количество секунд, то есть это одинаковая информация, просто представленная в разных форматах. Единственная разница в том, что Expires поддерживается в Internet Explorer, а Max-Age — нет.

Заголовок Authorization

Иногда использовать cookie неудобно и не нужно.

Например, мы хотим запустить небольшое приложение и не хотим подключать базу данных, чтобы хранить информацию о сессиях и пользователях, следить за временем жизни cookie и создавать метод авторизации для получением cookie клиентом.

В таких случаях можно просто договориться о каком-то секретном ключе с пользователями сервиса, который они будут слать с каждым запросом, в заголовке. Или другой пример: мы хотим посылать какой-то токен, в котором содержится много полезной информации (например, JWT) и не хотим, чтобы он протухал. Для таких случаев и был разработан заголовок Authorization.

Самым простым способом для такой авторизации является Basic Auth. Сначала клиент получает логин и пароль, например *admin* / *qwerty123*. Эти данные сохраняются на сервере, в коде программы или в конфигурационных файлах. Клиент объединяет логин и пароль в такую строку: '*admin:qwerty123*', а затем кодирует в base64. Получившаяся строка (*4oCYYWRtaW46cXdlcnR5MTIz*) отправляется вместе с заголовком Authorization и префиксом Basic:

Authorization: Basic *4oCYYWRtaW46cXdlcnR5MTIz*

Сервер выполняет обратные операции и сравнивает логин и пароль с теми, что имеются у него.

У этого способа авторизации есть большой минус: если кто-то украдет логин и пароль, он сможет нелегально пользоваться сервером до тех пор, пока эти данные не будут изменены. Поэтому данный способ стоит использовать только в комбинации с TLS-шифрованием или в закрытых сетях.

Существуют и другие способы авторизации в веб-приложениях, но в основном все они используют передачу каких-то данных через заголовки.