Сириус
IT-Колледж

Авторизация пользователя

Методы аутентификации и авторизации. OAuth2.0. JWT

Сириус
IT-Колледж

Основные понятия

Идентификация используется для определения, существует ли конкретный пользователь в системе. Проводится, например, по номеру телефона или логину.

Аутентификация — это процесс подтверждения права на доступ с помощью ввода пароля, пин-кода, использования биометрических данных и других способов.

Авторизация определяет набор привилегий и прав, доступных конкретному пользователю. Например, открывает доступ к просмотру и отправке электронных писем.

Сириус
IT-Колледж

Аутентификация по паролю

Этот метод основывается на том, что пользователь должен предоставить username и password для успешной идентификации и аутентификации в системе. Пара username/password задается пользователем при его регистрации в системе, при этом в качестве username может выступать адрес электронной почты пользователя.

Применительно к веб-приложениям, существует несколько стандартных протоколов для аутентификации по паролю, которые мы рассмотрим ниже.

Сириус
IT-Колледж

HTTP authentication

Этот протокол, описанный в стандартах HTTP 1.0/1.1, существует очень давно и до сих пор активно применяется в корпоративной среде. Применительно к веб-сайтам работает следующим образом:

Сервер, при обращении неавторизованного клиента к защищенному ресурсу, отсылает HTTP статус “401 Unauthorized” и добавляет заголовок “WWW-Authenticate” с указанием схемы и параметров аутентификации.
Браузер, при получении такого ответа, автоматически показывает диалог ввода username и password.
Пользователь вводит детали своей учетной записи.
Во всех последующих запросах к этому веб-сайту браузер автоматически добавляет HTTP заголовок “Authorization”, в котором передаются данные пользователя для аутентификации сервером.
Сервер аутентифицирует пользователя по данным из этого заголовка. Решение о предоставлении доступа (авторизация) производится отдельно на основании роли пользователя, ACL или других данных учетной записи.

Сириус IT-Колледж

HTTP authentication

Весь процесс стандартизирован и хорошо поддерживается всеми браузерами и веб-серверами. Существует несколько схем аутентификации, отличающихся по уровню безопасности:

1. **Basic** — наиболее простая схема, при которой username и password пользователя передаются в заголовке Authorization в незашифрованном виде (base64-encoded). Однако при использовании HTTPS (HTTP over SSL) протокола, является относительно безопасной.

Сириус IT-Колледж

HTTP authentication

2. **Digest** — challenge-response-схема, при которой сервер посылает уникальное значение nonce, а браузер передает MD5 хэш пароля пользователя, вычисленный с использованием указанного nonce. Более безопасная альтернативв Basic схемы при незащищенных соединениях, но подвержена man-in-the-middle attacks (с заменой схемы на basic). Кроме того, использование этой схемы не позволяет применить современные хэш-функции для хранения паролей пользователей на сервере.
3. **NTLM** (известная как Windows authentication) — также основана на challenge-response подходе, при котором пароль не передается в чистом виде. Эта схема не является стандартом HTTP, но поддерживается большинством браузеров и веб-серверов. Преимущественно используется для аутентификации пользователей Windows Active Directory в веб-приложениях. Уязвима к pass-the-hash-атакам.

Сириус
IT-Колледж

HTTP authentication

4. **Negotiate** — еще одна схема из семейства Windows authentication, которая позволяет клиенту выбрать между NTLM и Kerberos аутентификацией. Kerberos — более безопасный протокол, основанный на принципе Single Sign-On. Однако он может функционировать, только если и клиент, и сервер находятся в зоне intranet и являются частью домена Windows.

Стоит отметить, что при использовании HTTP-аутентификации у пользователя нет стандартной возможности выйти из веб-приложения, кроме как закрыть все окна браузера.

Сириус
IT-Колледж

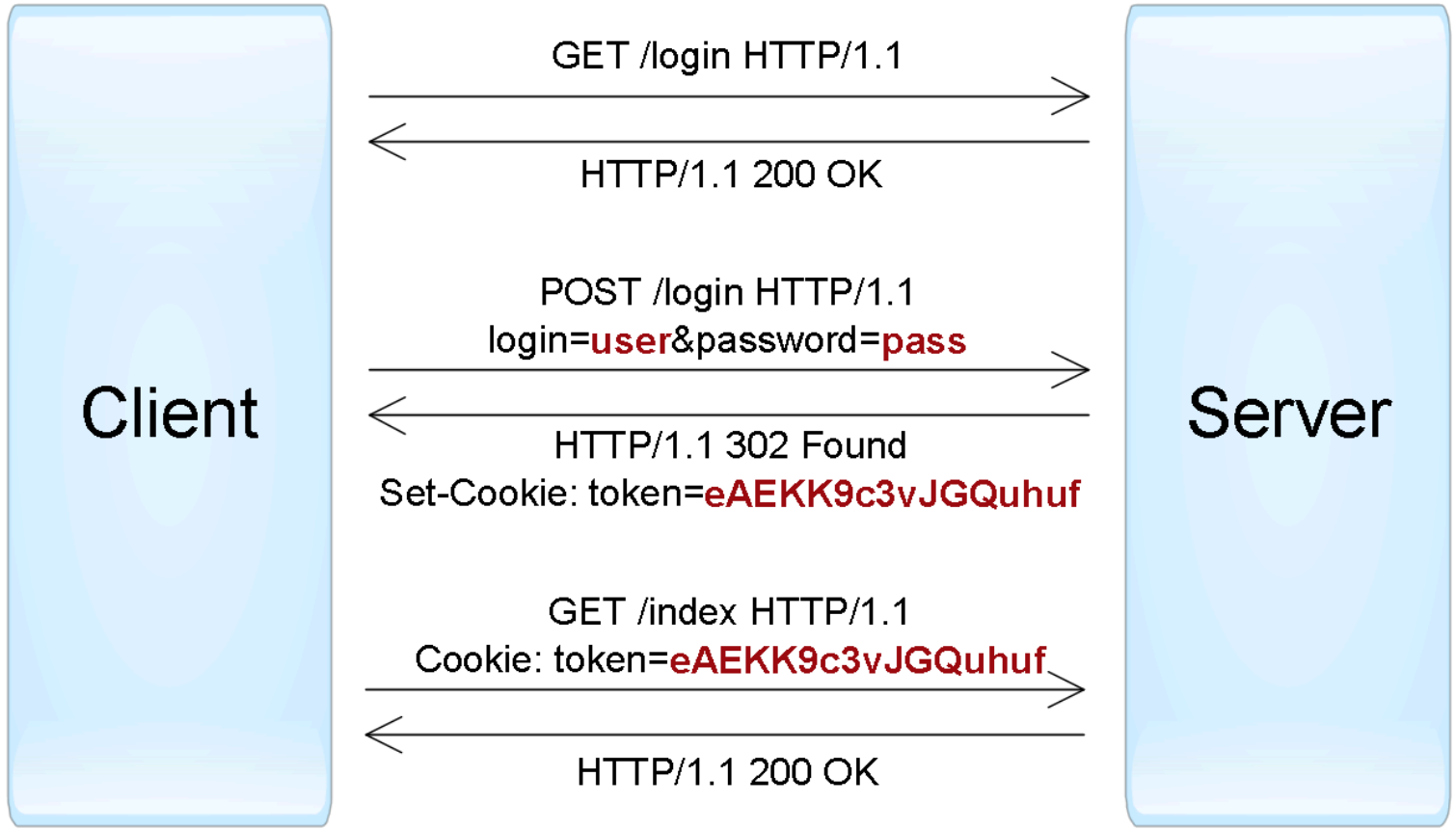
Forms authentication

Для этого протокола нет определенного стандарта, поэтому все его реализации специфичны для конкретных систем, а точнее, для модулей аутентификации фреймворков разработки.

Работает это по следующему принципу: в веб-приложение включается HTML-форма, в которую пользователь должен ввести свои username/password и отправить их на сервер через `HTTP POST` для аутентификации. В случае успеха веб-приложение создает `session token`, который обычно помещается в `browser cookies`. При последующих веб-запросах `session token` автоматически передается на сервер и позволяет приложению получить информацию о текущем пользователе для авторизации запроса.

Сириус
IT-Колледж

Forms authentication



Сириус
IT-Колледж

Forms authentication

Приложение может создать `session token` двумя способами:

1. Как идентификатор аутентифицированной сессии пользователя, которая хранится в памяти сервера или в базе данных. Сессия должна содержать всю необходимую информацию о пользователе для возможности авторизации его запросов.
2. Как зашифрованный и/или подписанный объект, содержащий данные о пользователе, а также период действия. Этот подход позволяет реализовать stateless-архитектуру сервера, однако требует механизма обновления сессионного токена по истечении срока действия.

Сириус IT-Колледж

Другие протоколы аутентификации по паролю

Два протокола, описанных выше, успешно используются для аутентификации пользователей на веб-сайтах. Но при разработке клиент-серверных приложений с использованием веб-сервисов (например, iOS или Android), наряду с HTTP аутентификацией, часто применяются нестандартные протоколы, в которых данные для аутентификации передаются в других частях запроса.

Существует всего несколько мест, где можно передать username и password в HTTP запросах:

1. **URL query** — считается небезопасным вариантом, т. к. строки URL могут запоминаться браузерами, прокси и веб-серверами.
2. **Request body** — безопасный вариант, но он применим только для запросов, содержащих тело сообщения (такие как `POST`, `PUT`, `PATCH`).
3. **HTTP header** — оптимальный вариант, при этом могут использоваться и стандартный заголовок `Authorization` (например, с Basic-схемой), и другие произвольные заголовки.

Сириус
IT-Колледж

Аутентификация по сертификатам

Сертификат представляет собой набор атрибутов, идентифицирующих владельца, подписанный certificate authority (CA). CA выступает в роли посредника, который гарантирует подлинность сертификатов. Также сертификат криптографически связан с закрытым ключом, который хранится у владельца сертификата и позволяет однозначно подтвердить факт владения сертификатом.

На стороне клиента сертификат вместе с закрытым ключом могут храниться в операционной системе, в браузере, в файле, на отдельном физическом устройстве (smart card, USB token). Обычно закрытый ключ дополнительно защищен паролем или PIN-кодом.

В веб-приложениях традиционно используют сертификаты стандарта X.509. Аутентификация с помощью X.509-сертификата происходит в момент соединения с сервером и является частью протокола SSL/TLS. Этот механизм также хорошо поддерживается браузерами, которые позволяют пользователю выбрать и применить сертификат, если веб-сайт допускает такой способ аутентификации.

Сириус
IT-Колледж

Аутентификация по сертификатам

Во время аутентификации сервер выполняет проверку сертификата на основании следующих правил:

1. Сертификат должен быть подписан доверенным certification authority (проверка цепочки сертификатов).
2. Сертификат должен быть действительным на текущую дату (проверка срока действия).
3. Сертификат не должен быть отозван соответствующим CA (проверка списков исключения).

После успешной аутентификации веб-приложение может выполнить авторизацию запроса на основании таких данных сертификата, как `subject` (имя владельца), `issuer` (эмитент), `serial number` (серийный номер сертификата) или `thumbprint` (отпечаток открытого ключа сертификата).

Сириус
IT-Колледж

Аутентификация по одноразовым паролям

Аутентификация по одноразовым паролям обычно применяется дополнительно к аутентификации по паролям для реализации **two-factor authentication (2FA)**. В этой концепции пользователю необходимо предоставить данные двух типов для входа в систему: что-то, что он знает (например, пароль), и что-то, чем он владеет (например, устройство для генерации одноразовых паролей). Наличие двух факторов позволяет в значительной степени увеличить уровень безопасности, что м. б. востребовано для определенных видов веб-приложений.

Другой популярный сценарий использования одноразовых паролей — дополнительная аутентификация пользователя во время выполнения важных действий: перевод денег, изменение настроек и т. п.

Сириус
IT-Колледж

Аутентификация по ключам доступа

Этот способ чаще всего используется для аутентификации устройств, сервисов или других приложений при обращении к веб-сервисам. Здесь в качестве секрета применяются ключи доступа (**access key, API key**) — длинные уникальные строки, содержащие произвольный набор символов, по сути заменяющие собой комбинацию username/password.

В большинстве случаев, сервер генерирует ключи доступа по запросу пользователей, которые далее сохраняют эти ключи в клиентских приложениях. При создании ключа также возможно ограничить срок действия и уровень доступа, который получит клиентское приложение при аутентификации с помощью этого ключа.

Сириус
IT-Колледж

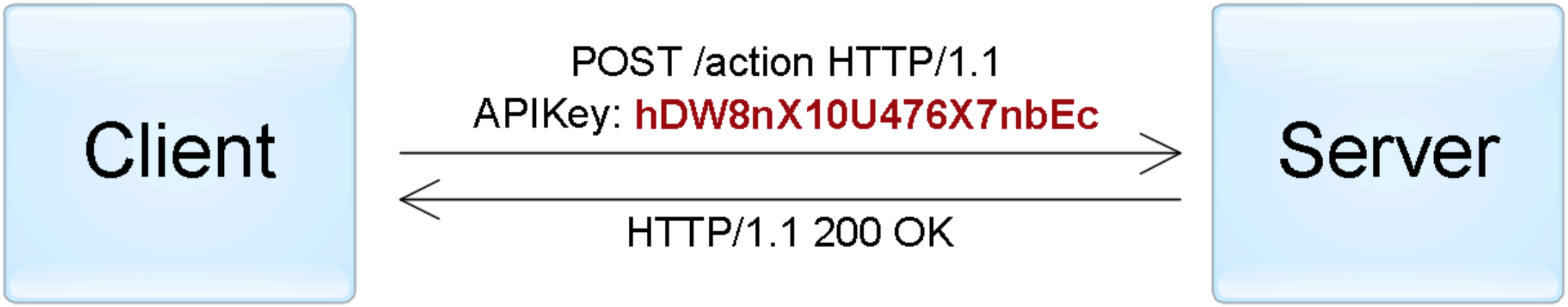
Аутентификация по ключам доступа

Использование ключей позволяет избежать передачи пароля пользователя сторонним приложениям. Ключи обладают значительно большей энтропией по сравнению с паролями, поэтому их практически невозможно подобрать. Кроме того, если ключ был раскрыт, это не приводит к компрометации основной учетной записи пользователя — достаточно лишь аннулировать этот ключ и создать новый.

С технической точки зрения, здесь не существует единого протокола: ключи могут передаваться в разных частях HTTP-запроса: `URL query`, `request body` или `HTTP header`. Как и в случае аутентификации по паролю, наиболее оптимальный вариант — использование `HTTP header`. В некоторых случаях используют HTTP-схему **Bearer** для передачи токена в заголовке (`Authorization: Bearer [token]`). Чтобы избежать перехвата ключей, соединение с сервером должно быть обязательно защищено протоколом **SSL/TLS**.

Сириус
IT-Колледж

Аутентификация по ключам доступа



Кроме того, существуют более сложные схемы аутентификации по ключам для незащищенных соединений. В этом случае, ключ обычно состоит их двух частей: публичной и секретной. Публичная часть используется для идентификации клиента, а секретная часть позволяет сгенерировать подпись. Например, по аналогии с `digest authentication` схемой, сервер может послать клиенту уникальное значение `nonce` или `timestamp`, а клиент — возвратить хэш или HMAC этого значения, вычисленный с использованием секретной части ключа. Это позволяет избежать передачи всего ключа в оригинальном виде и защищает от replay attacks.

Сириус
IT-Колледж

Аутентификация по токенам

Такой способ аутентификации чаще всего применяется при построении распределенных систем **Single Sign-On (SSO)**, где одно приложение (**service provider** или **relying party**) делегирует функцию аутентификации пользователей другому приложению (**identity provider** или **authentication service**). Типичный пример этого способа — вход в приложение через учетную запись в социальных сетях. Здесь социальные сети являются сервисами аутентификации, а приложение доверяет функцию аутентификации пользователей социальным сетям.

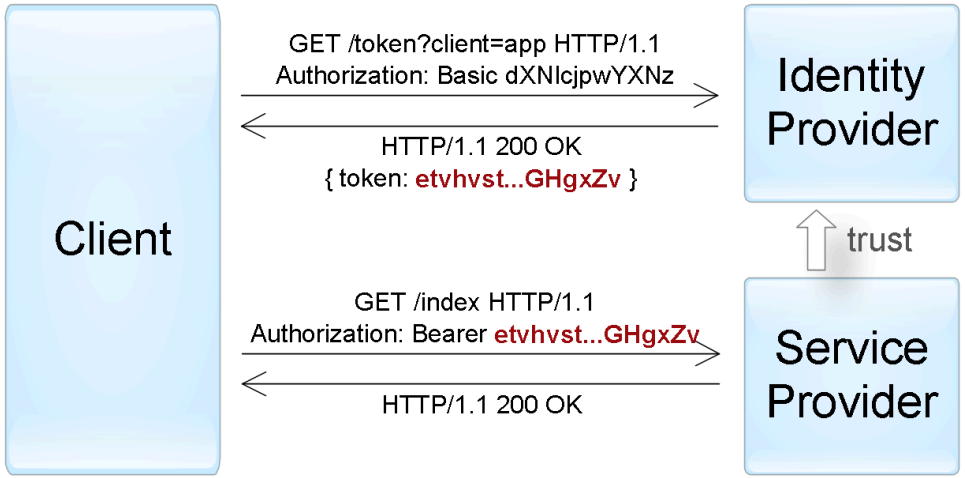
Реализация этого способа заключается в том, что **identity provider (IP)** предоставляет достоверные сведения о пользователе в виде токена, а **service provider (SP)** приложение использует этот токен для идентификации, аутентификации и авторизации пользователя.

Сириус IT-Колледж

Аутентификация по токенам

На общем уровне, весь процесс выглядит следующим образом:

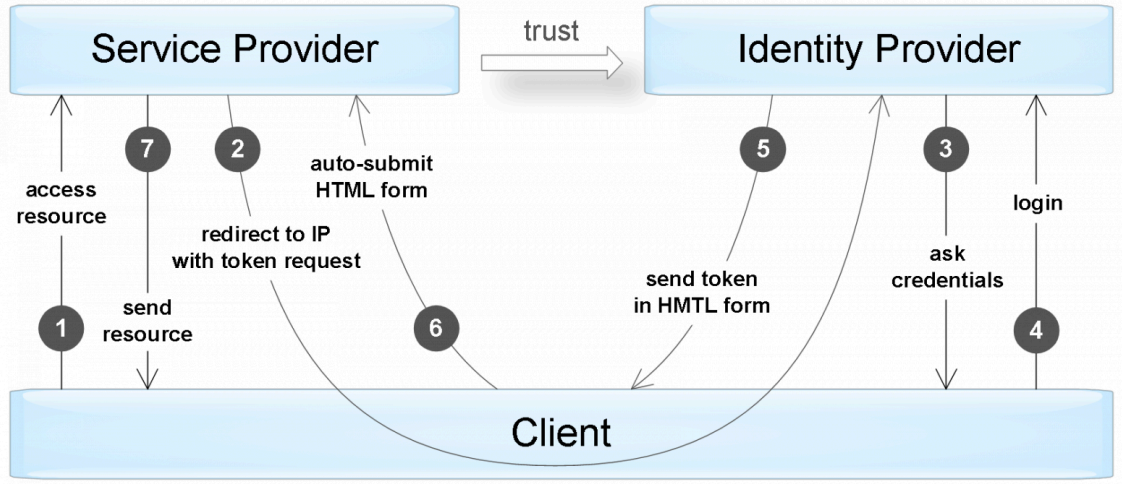
1. Клиент аутентифицируется в **identity provider** одним из способов, специфичным для него (пароль, ключ доступа, сертификат, Kerberos, итд.).
2. Клиент просит **identity provider** предоставить ему токен для конкретного **SP-приложения**. **Identity provider** генерирует токен и отправляет его клиенту.
3. Клиент аутентифицируется в **SP-приложении** при помощи этого токена.



Сириус
IT-Колледж

Аутентификация по токенам

Процесс, описанный выше, отражает механизм аутентификации активного клиента, т. е. такого, который может выполнять запрограммированную последовательность действий (например, iOS/Android приложения). Браузер же — пассивный клиент в том смысле, что он только может отображать страницы, запрошенные пользователем. В этом случае аутентификация достигается посредством автоматического перенаправления браузера между веб-приложениями **identity provider** и **service provider**.



Сириус
IT-Колледж

Аутентификация по токенам

Существует несколько стандартов, в точности определяющих протокол взаимодействия между клиентами (активными и пассивными) и IP/SP-приложениями и формат поддерживаемых токенов. Среди наиболее популярных стандартов — **OAuth, OpenID Connect, SAML, и WS-Federation**.

Сам токен обычно представляет собой структуру данных, которая содержит информацию, кто сгенерировал токен, кто может быть получателем токена, срок действия, набор сведений о самом пользователе (`claims`). Кроме того, токен дополнительно подписывается для предотвращения несанкционированных изменений и гарантий подлинности.

Сириус
IT-Колледж

Аутентификация по токенам

При аутентификации с помощью токена SP-приложение должно выполнить следующие проверки:

1. Токен был выдан доверенным **identity provider** приложением (проверка поля `issuer`).
2. Токен предназначается текущему SP-приложению (проверка поля `audience`).
3. Срок действия токена еще не истек (проверка поля `expiration date`).
4. Токен подлинный и не был изменен (проверка подписи).

В случае успешной проверки SP-приложение выполняет авторизацию запроса на основании данных о пользователе, содержащихся в токене.

Сириус IT-Колледж

Аутентификация по токенам. Форматы токенов

Существует несколько распространенных форматов токенов для веб-приложений:

1. **Simple Web Token (SWT)** — наиболее простой формат, представляющий собой набор произвольных пар имя/значение в формате кодирования `HTML form`. Стандарт определяет несколько зарезервированных имен: `Issuer`, `Audience`, `ExpiresOn` и `HMACSHA256`. Токен подписывается с помощью симметричного ключа, таким образом оба IP- и SP-приложения должны иметь этот ключ для возможности создания/проверки токена.

   Пример SWT токена (после декодирования).

```
Issuer=http://auth.myservice.com&
Audience=http://myservice.com&
ExpiresOn=1435937883&
UserName=John Smith&
UserRole=Admin&
HMACSHA256=KOUQRPSpy64rvT2KnYyQKtFFXUIggnesSpE7ADA4o9w
```

Сириус IT-Колледж

Аутентификация по токенам. Форматы токенов

2. **JSON Web Token (JWT)** — содержит три блока, разделенных точками: заголовок, набор полей (`claims`) и подпись. Первые два блока представлены в JSON-формате и дополнительно закодированы в формат `base64`. Набор полей содержит произвольные пары имя/значения, притом стандарт JWT определяет несколько зарезервированных имен (`iss`, `aud`, `exp` и другие). Подпись может генерироваться при помощи и симметричных алгоритмов шифрования, и асимметричных. Кроме того, существует отдельный стандарт, отписывающий формат зашифрованного JWT-токена.

   Пример подписанного JWT токена (после декодирования 1 и 2 блоков).

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

Сириус IT-Колледж

Аутентификация по токенам. Форматы токенов

3. **Security Assertion Markup Language (SAML)** — определяет токены (`SAML assertions`) в XML-формате, включающем информацию об эмитенте, о субъекте, необходимые условия для проверки токена, набор дополнительных утверждений (`statements`) о пользователе. Подпись SAML-токенов осуществляется при помощи ассиметричной криптографии. Кроме того, в отличие от предыдущих форматов, SAML-токены содержат механизм для подтверждения владения токеном, что позволяет предотвратить перехват токенов через man-in-the-middle-атаки при использовании незащищенных соединений.

Сириус
IT-Колледж

Авторизация в FastAPI

OpenAPI поддерживает множество стандартных "схем" безопасности.

Используя их, вы можете получить преимущества стандартизированных инструментов, включая интерктивную документацию.

Сириус IT-Колледж

Авторизация в FastAPI

OpenAPI определяет следующие схемы безопасности:

- `apiKey`
- `http`: `bearer` (заголовок `Authorization` со значением `Bearer` плюс токен, наследуется из OAuth2), HTTP Basic аутентификация, HTTP Digest, и тд.
- `oauth2`: все стандартные методы OAuth2 (называемые "потоками" ("flows")).
  - Некоторые с использованием доверенных **authentication provider** (Google, Facebook, Twitter, GitHub, и тп): `implicit`, `clientCredentials`, `authorizationCode`
  - Один особый поток, подходящий для аутентификации непосредственно в самом приложении: `password`
- `openIdConnect`

Сириус IT-Колледж

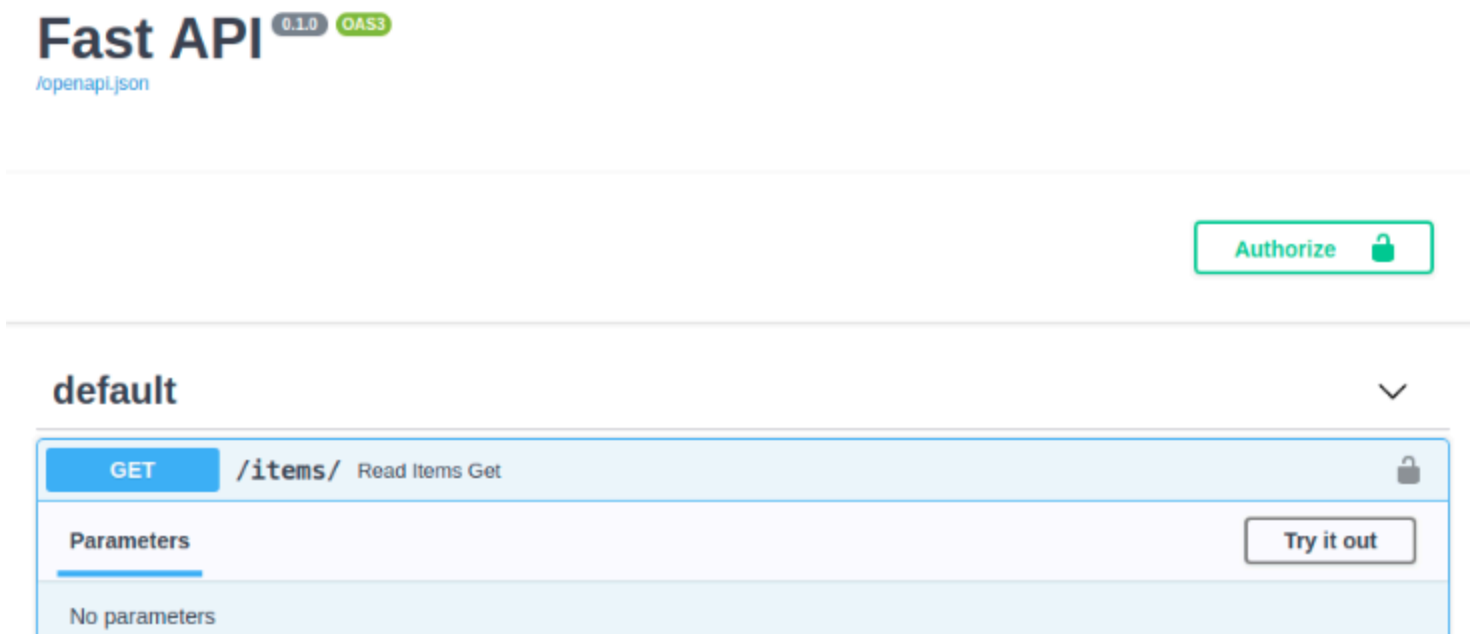
Реализация в FastAPI

Файл `main.py`

```python
from typing import Annotated

from fastapi import Depends, FastAPI
from fastapi.security import OAuth2PasswordBearer

app = FastAPI()

oauth2_scheme = OAuth2PasswordBearer(tokenUrl="token")


@app.get("/items/")
async def read_items(token: Annotated[str, Depends(oauth2_scheme)]):
    return {"token": token}
```

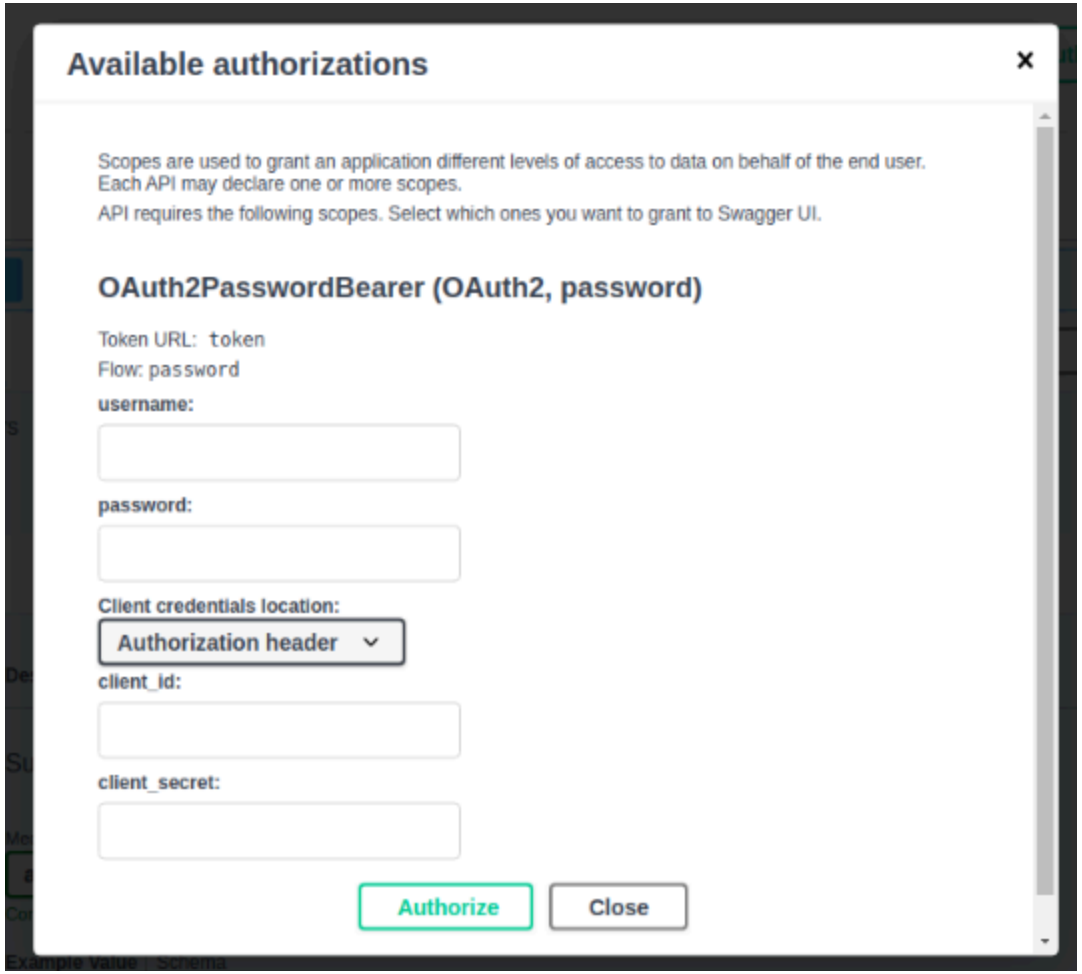
Сириус
IT-Колледж

Реализация в FastAPI



Сириус
IT-Колледж

Реализация в FastAPI



Сириус
IT-Колледж

Реализация в FastAPI

Аутентификация по паролю является одним из способов, определенных в OAuth2, для обеспечения безопасности и аутентификации.

OAuth2 был разработан для того, чтобы бэкэнд или API были независимы от сервера, который аутентифицирует пользователя.

Но в нашем случае одно и то же приложение FastAPI будет работать с API и аутентификацией.

Сириус IT-Колледж

Реализация в FastAPI

- Пользователь вводит на фронтенде `имя пользователя` и `пароль` и нажимает `Enter`.
- Фронтенд (работающий в браузере пользователя) отправляет эти `имя пользователя` и `пароль` на определенный URL в нашем API (объявленный с помощью параметра `tokenUrl="token"`).
- API проверяет эти `имя пользователя` и `пароль` и выдает в ответ "токен".
  - "Токен" - это просто строка с некоторым содержимым, которое мы можем использовать позже для верификации пользователя.
  - Обычно срок действия токена истекает через некоторое время.
- Фронтенд временно хранит этот токен в каком-то месте.
- Пользователь щелкает мышью на фронтенде, чтобы перейти в другой раздел на фронтенде.
- Фронтенду необходимо получить дополнительные данные из API. Но для этого необходима аутентификация для конкретной конечной точки. Поэтому для аутентификации в нашем API он посылает заголовок `Authorization` со значением `Bearer` плюс сам токен. Если токен содержит `foobar`, то содержание заголовка `Authorization` будет таким: `Bearer foobar`.

Сириус IT-Колледж

Класс `OAuth2PasswordBearer` в FastAPI

FastAPI предоставляет несколько средств на разных уровнях абстракции для реализации этих функций безопасности.

В данном примере мы будем использовать OAuth2, с аутентификацией по паролю, используя токен Bearer. Для этого мы используем класс `OAuth2PasswordBearer`.

При создании экземпляра класса `OAuth2PasswordBearer` мы передаем в него параметр `tokenUrl`. Этот параметр содержит URL, который клиент (фронтенд, работающий в браузере пользователя) будет использовать для отправки имени пользователя и пароля с целью получения токена.

Сириус IT-Колледж

Класс `OAuth2PasswordBearer` в FastAPI

```python
from typing import Annotated

from fastapi import Depends, FastAPI
from fastapi.security import OAuth2PasswordBearer

app = FastAPI()

oauth2_scheme = OAuth2PasswordBearer(tokenUrl="token")


@app.get("/items/")
async def read_items(token: Annotated[str, Depends(oauth2_scheme)]):
    return {"token": token}
```

Этот параметр не создает конечную точку / операцию пути, а объявляет, что URL `/token` будет таким, который клиент должен использовать для получения токена. Эта информация используется в OpenAPI, а затем в интерактивных системах документации API.

Сириус IT-Колледж

Класс `OAuth2PasswordBearer` в FastAPI

Переменная `oauth2_scheme` является экземпляром `OAuth2PasswordBearer`, но она также является "вызываемой".

Ее можно вызвать следующим образом:

```
oauth2_scheme(some, parameters)
```

Поэтому ее можно использовать вместе с `Depends`.

Сириус IT-Колледж

Создание зависимости `get_current_user`

`get_current_user` будет так же зависеть от `oauth2_scheme` созданной ранее.

Наша новая зависимость `get_current_user` будет получать `token` как `str` из подзависимости `oauth2_scheme`:

```
async def get_current_user(token: Annotated[str, Depends(oauth2_scheme)]):
    user = fake_decode_token(token)
    return user
```

Сириус IT-Колледж

Получаем пользователя

`get_current_user` будет использовать (пока фальшивую) служебную функцию, которая получает токен в виде `str` модель Pydantic `User`:

```
    def fake_decode_token(token):
        return User(
            username=token + "fakedecoded", email="john@example.com", full_name="John Doe"
        )

    async def get_current_user(token: Annotated[str, Depends(oauth2_scheme)]):
        user = fake_decode_token(token)
        return user
```

Сириус
IT-Колледж

Инъекция текущего пользователя

Теперь мы можем использовать ту же зависимость `get_current_user` в операциях пути:

```
    @app.get("/users/me")
    async def read_users_me(current_user: Annotated[User, Depends(get_current_user)]):
        return current_user
```

> Обратите внимание, что мы определяем тип для `current_user` как модель Pydantic `User`.

OAuth2, JWT

```python
from datetime import datetime, timedelta, timezone
from typing import Annotated

from fastapi import Depends, FastAPI, HTTPException, status
from fastapi.security import OAuth2PasswordBearer, OAuth2PasswordRequestForm
from jose import JWTError, jwt
from passlib.context import CryptContext
from pydantic import BaseModel

SECRET_KEY = "09d25e094faa6ca2556c818166b7a9563b93f7099f6f0f4caa6cf63b88e8d3e7"
ALGORITHM = "HS256"
ACCESS_TOKEN_EXPIRE_MINUTES = 30

fake_users_db = {
    "johndoe": {
        "username": "johndoe",
        "full_name": "John Doe",
        "email": "johndoe@example.com",
        "hashed_password": "$2b$12$EixZaYVK1fsbw1ZfbX3OXePaWxn96p36WQoeG6Lruj3vjPGga31lW",
        "disabled": False,
    }
}
```

Сириус IT-Колледж

OAuth2, JWT

```python
class Token(BaseModel):
    access_token: str
    token_type: str


class TokenData(BaseModel):
    username: str | None = None


class User(BaseModel):
    username: str
    email: str | None = None
    full_name: str | None = None
    disabled: bool | None = None


class UserInDB(User):
    hashed_password: str
```

Сириус IT-Колледж

OAuth2, JWT

```
pwd_context = CryptContext(schemes=["bcrypt"], deprecated="auto")

oauth2_scheme = OAuth2PasswordBearer(tokenUrl="token")

app = FastAPI()


def verify_password(plain_password, hashed_password):
    return pwd_context.verify(plain_password, hashed_password)


def get_password_hash(password):
    return pwd_context.hash(password)


def get_user(db, username: str):
    if username in db:
        user_dict = db[username]
        return UserInDB(**user_dict)
```

Сириус IT-Колледж

OAuth2, JWT

```python
def authenticate_user(fake_db, username: str, password: str):
    user = get_user(fake_db, username)
    if not user:
        return False
    if not verify_password(password, user.hashed_password):
        return False
    return user


def create_access_token(data: dict, expires_delta: timedelta | None = None):
    to_encode = data.copy()
    if expires_delta:
        expire = datetime.now(timezone.utc) + expires_delta
    else:
        expire = datetime.now(timezone.utc) + timedelta(minutes=15)
    to_encode.update({"exp": expire})
    encoded_jwt = jwt.encode(to_encode, SECRET_KEY, algorithm=ALGORITHM)
    return encoded_jwt
```

Сириус IT-Колледж

OAuth2, JWT

```
async def get_current_user(token: Annotated[str, Depends(oauth2_scheme)]):
    credentials_exception = HTTPException(
        status_code=status.HTTP_401_UNAUTHORIZED,
        detail="Could not validate credentials",
        headers={"WWW-Authenticate": "Bearer"},
    )
    try:
        payload = jwt.decode(token, SECRET_KEY, algorithms=[ALGORITHM])
        username: str = payload.get("sub")
        if username is None:
            raise credentials_exception
        token_data = TokenData(username=username)
    except JWTError:
        raise credentials_exception
    user = get_user(fake_users_db, username=token_data.username)
    if user is None:
        raise credentials_exception
    return user
```

Сириус IT-Колледж

OAuth2, JWT

```
async def get_current_active_user(
    current_user: Annotated[User, Depends(get_current_user)]
):
    if current_user.disabled:
        raise HTTPException(status_code=400, detail="Inactive user")
    return current_user
```

Сириус IT-Колледж

OAuth2, JWT

```python
@app.post("/token")
async def login_for_access_token(
    form_data: Annotated[OAuth2PasswordRequestForm, Depends()]
) -> Token:
    user = authenticate_user(fake_users_db, form_data.username, form_data.password)
    if not user:
        raise HTTPException(
            status_code=status.HTTP_401_UNAUTHORIZED,
            detail="Incorrect username or password",
            headers={"WWW-Authenticate": "Bearer"},
        )
    access_token_expires = timedelta(minutes=ACCESS_TOKEN_EXPIRE_MINUTES)
    access_token = create_access_token(
        data={"sub": user.username}, expires_delta=access_token_expires
    )
    return Token(access_token=access_token, token_type="bearer")
```

Сириус IT-Колледж

OAuth2, JWT

```python
@app.get("/users/me/", response_model=User)
async def read_users_me(
    current_user: Annotated[User, Depends(get_current_active_user)]
):
    return current_user


@app.get("/users/me/items/")
async def read_own_items(
    current_user: Annotated[User, Depends(get_current_active_user)]
):
    return [{"item_id": "Foo", "owner": current_user.username}]
```