Безопасность приложений

Конфиденциальность, целостность и доступность данных пользователей

Безопасность приложений

Безопасность приложений — критически важный аспект разработки, особенно в эпоху облачных сервисов и мобильных приложений. Она обеспечивает конфиденциальность, целостность и доступность данных пользователей. Ключевыми компонентами безопасности являются управление доступом, которое включает аутентификацию и авторизацию, и безопасная передача данных.

Как сделать приложение безопасным?
В первую очередь необходимо определить какие процессы за это отвечают

Аутентификация

Аутентификация — это процесс проверки подлинности пользователя или системы. Другими словами, это ответ на вопрос: «Кто вы?». Цель — убедиться, что пользователь действительно является тем, за кого себя выдает.

Примеры: Вход по логину и паролю, отпечатку пальца, Face ID, SMS-коду.

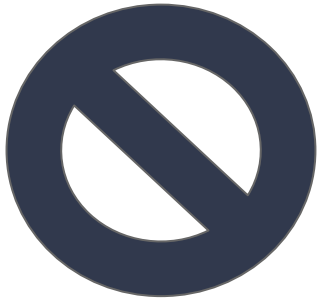
Суть: Система сравнивает предоставленные учетные данные (например, пароль) с хранящимися в базе данных. При успехе создается сессия или токен, подтверждающий личность пользователя.

Авторизация

Авторизация — это процесс определения, какие действия, ресурсы или данные разрешены уже аутентифицированному пользователю. Это ответ на вопрос: «Что вам разрешено делать?».

Примеры: Администратор может удалять пользователей, а обычный пользователь — только просматривать свой профиль. Доступ к определенным папкам или API-эндпоинтам.

Суть: После того как система узнала кто пользователь (аутентификация), она проверяет его права (роли, разрешения) для доступа к запрошенному ресурсу.



Важно: Аутентификация всегда предшествует авторизации. Нельзя проверить права, не зная, кто их запрашивает.

Basic Authentication

Базовая аутентификация - устаревший простой метод, пригодный только с HTTPS в простых сценариях.

Как это работает: Клиент отправляет запрос с заголовком Authorization, содержащим слово Basic и строку **login:password**, закодированную в **Base64**.

Пароль хранится в захешированном виде с использованием криптографических хеш-функций. Сервер НЕ расшифровывает и не декодирует хеш из БД. Вместо этого он хеширует введенный пароль и сравнивает результат с хранимым хешом:

Пример: Authorization: Basic dXNlcjpwYXNzd29yZA==

Преимущества: Просто реализовать и использовать.
Недостатки:

- Логин и пароль отправляются с каждым запросом.
- Пароль (в форме Base64) легко перехватить и так же легко декодировать. Безопасно только при использовании поверх HTTPS (SSL/TLS).
- Нет встроенного механизма выхода (сессии).

Применение: В основном для простых сценариев, скриптов или внутренних (private) API.

Альтернатива

Digest Access Authentication
Улучшенная версия Basic Auth, разработанная для устранения его главного недостатка — передачи пароля в открытом виде (даже в Base64). Пароль никогда не передается по сети.

API Keys (Ключи API)
Уникальный длинный токен (строка), который клиент передает для идентификации и аутентификации, обычно в заголовке или как параметр запроса. API Key идентифицирует *приложение* или *проект*, а Basic Auth обычно идентифицирует *пользователя*.

Пример: X-API-Key: abc123def456

OpenID Connect (OIDC)
Надстройка над OAuth 2.0. Если OAuth 2.0 решает задачу "доступа к ресурсам", то OIDC решает задачу "аутентификации пользователя" (верификации личности)

Добавляет к ответу OAuth 2.0 стандартный токен **id_token** в формате JWT.
Современный стандарт для аутентификации.

Современный подход

JWT-токен для аутентификации +
OAuth2.0 для делегирования доступа

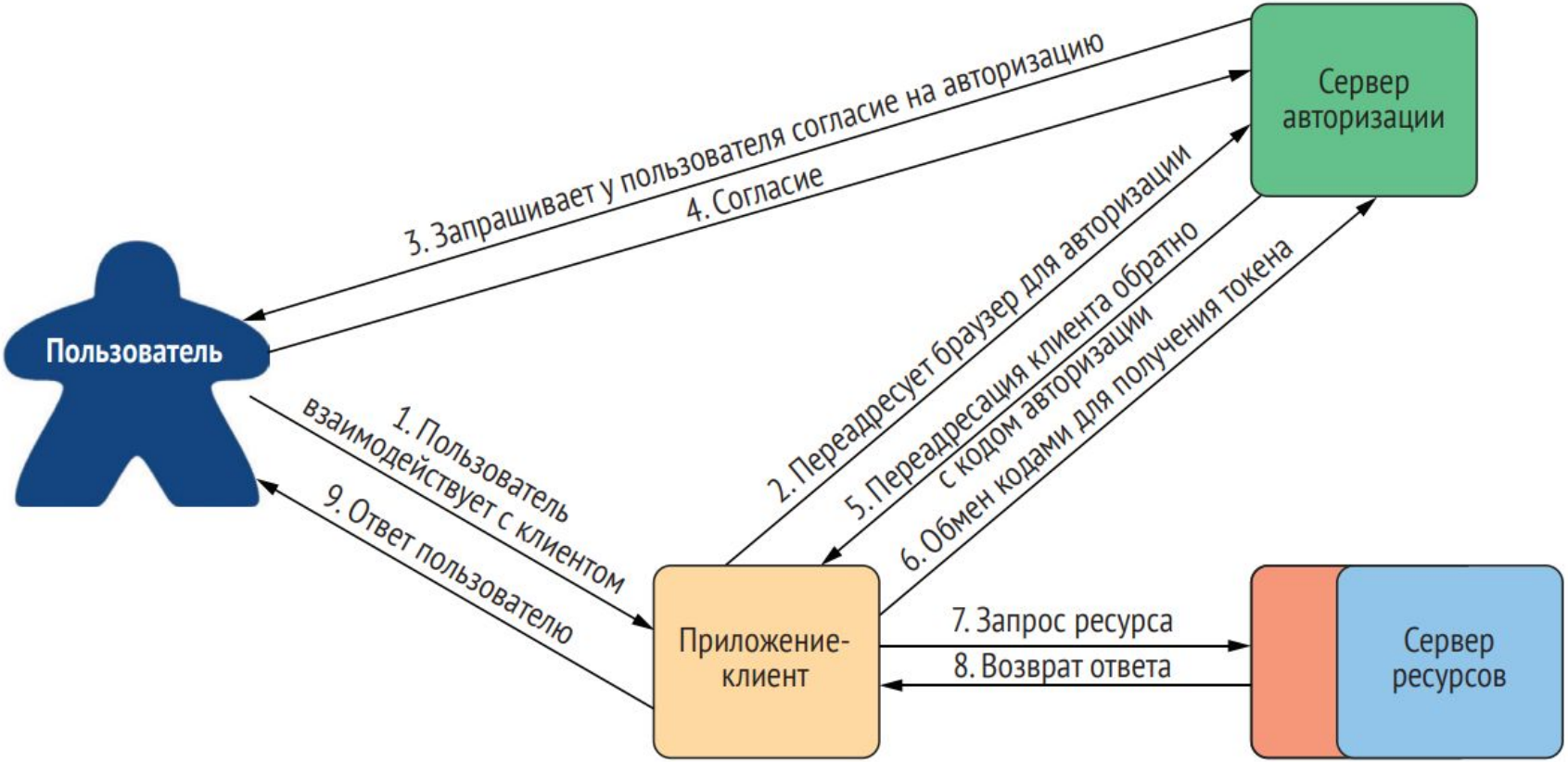
OAuth 2.0

OAuth 2.0 — это современный авторизационный фреймворк (протокол делегирования доступа). Он позволяет приложению (клиенту) получить ограниченный доступ к защищенным ресурсам пользователя на другом сервисе (ресурсном сервере), не раскрывая пароль пользователя.

Принцип работы:

1. Пользователь в вашем приложении нажимает «Войти через Google/Facebook».
2. Приложение перенаправляет его на сервис провайдера (Google).
3. Пользователь вводит свои учетные данные напрямую в Google (пароль не видно вашему приложению).
4. Google после успешной аутентификации выдает вашему приложению специальный Access Token (токен доступа).
5. Ваше приложение использует этот токен для запроса данных пользователя (например, email) у Google API.

Основная цель: Делегирование доступа. Широко используется для социального входа (Social Login)



Изображение из книги Крейга Уоллса “Spring в действии” (рис. 8.1)

Что из себя представляет JWT (JSON Web Token)?

JWT — это современный, компактный и автономный стандарт для безопасной передачи информации между сторонами в виде JSON-объекта. Это формат токена, который часто используется для аутентификации и авторизации.

Структура токена (3 части, разделенные точками):

- Header: Алгоритм шифрования и тип токена.
- Payload (Полезная нагрузка): Данные (утверждения — «claims»). Например, ID пользователя, его роли, срок действия токена (exp).
- Signature (Подпись): Создается на основе Header, Payload и секретного ключа. Гарантирует, что токен не был изменен.

Клиент хранит JWT (обычно в localStorage или куках) и отправляет его в заголовке Authorization: Bearer <токен> при каждом запросе к защищенным API. Сервер ресурсов проверяет подпись JWT и данные в Payload (например, роль и срок действия). Если все верно — предоставляет доступ.

Роли каждого токена

ДА, их несколько…

Authorization Code: Одноразовый, для безопасного обмена

Access Token (JWT): Короткоживущий, для доступа к API

Refresh Token: Долгоживущий, для получения новых Access Token

ID Token (JWT): Только для аутентификации (OIDC)

Почему это актуальный стек?

Authorization code - защищает от перехвата

JWT подписан — нельзя подделать

Короткое время жизни Access Token (минуты/часы)

Refresh token хранится в защищенном хранилище на backend, используется для получения новых Access Token без повторного входа пользователя и может быть отозван в любой момент

Единая точка аутентификации (AS), но распределенная авторизация (любой сервис может проверить JWT)

Микросервисная архитектура: Один Access Token для всех сервисов