SLIDE 1

Обзор способов и протоколов аутентификации в веб-приложениях

Я расскажу о применении различных способов аутентификации для веб-приложений, включая аутентификацию по паролю, по сертификатам, по одноразовым паролям, по ключам доступа и по токенам. Коснусь технологии единого входа (Single Sign-On), рассмотрю различные стандарты и протоколы аутентификации.

SLIDE 2

Идентификация — это заявление о том, кем вы являетесь. В зависимости от ситуации, это может быть имя, адрес электронной почты, номер учетной записи, итд.

Аутентификация — предоставление доказательств, что вы на самом деле есть тот, кем идентифицировались (от слова “authentic” — истинный, подлинный).

Авторизация — проверка, что вам разрешен доступ к запрашиваемому ресурсу.

Например, при попытке попасть в закрытый клуб вас *идентифицируют* (спросят ваше имя и фамилию), *аутентифицируют* (попросят показать паспорт и сверят фотографию) и *авторизуют* (проверят, что фамилия находится в списке гостей), прежде чем пустят внутрь.  
  
Аналогично эти термины применяются в компьютерных системах, где традиционно под *идентификацией* понимают получение вашей учетной записи (identity) по username или email; под *аутентификацией* — проверку, что вы знаете пароль от этой учетной записи, а под *авторизацией* — проверку вашей роли в системе и решение о предоставлении доступа к запрошенной странице или ресурсу.  
  
Однако в современных системах существуют и более сложные схемы аутентификации и авторизации, о которых я расскажу далее. Но начнем с простого и понятного.

SLIDE 3

Виды аутентификации

1. Аутентификация по паролю

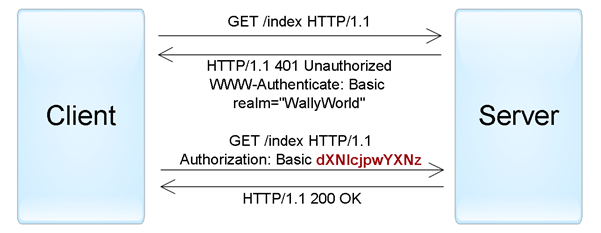
Этот метод основывается на том, что пользователь должен предоставить username и password для успешной идентификации и аутентификации в системе. Пара username/password задается пользователем при его регистрации в системе, при этом в качестве username может выступать адрес электронной почты пользователя.  
  
Применительно к веб-приложениям, существует несколько стандартных протоколов для аутентификации по паролю, которые мы рассмотрим ниже.

1.1 HTTP authentication

1. Сервер, при обращении неавторизованного клиента к защищенному ресурсу, отсылает HTTP статус “401 Unauthorized” и добавляет заголовок “WWW-Authenticate” с указанием схемы и параметров аутентификации.
2. Браузер, при получении такого ответа, автоматически показывает диалог ввода username и password. Пользователь вводит детали своей учетной записи.
3. Во всех последующих запросах к этому веб-сайту браузер автоматически добавляет HTTP заголовок “Authorization”, в котором передаются данные пользователя для аутентификации сервером.
4. 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.

**SLIDE 3**

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

**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.  
  
*Пример HTTP аутентификации с использованием Basic схемы.*

**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.

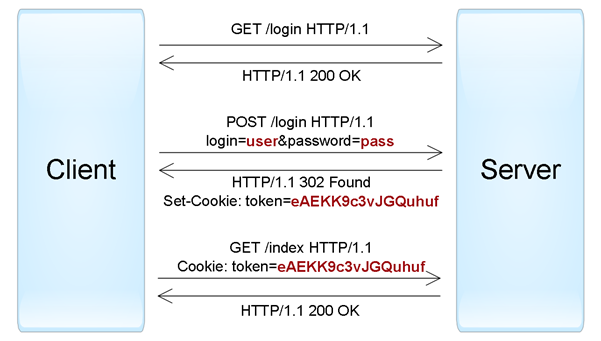
**NTLM** (известная как Windows authentication) — также основана на challenge-response подходе, при котором пароль не передается в чистом виде. Эта схема не является стандартом HTTP, но поддерживается большинством браузеров и веб-серверов. Преимущественно используется для аутентификации пользователей Windows Active Directory в веб-приложениях. Уязвима к pass-the-hash-атакам.

**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.

SLIDE 4

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.

SLIDE 5

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:

**URL query** — считается небезопасным вариантом, т. к. строки URL могут запоминаться браузерами, прокси и веб-серверами.

**Request body** — безопасный вариант, но он применим только для запросов, содержащих тело сообщения (такие как POST, PUT, PATCH).

**HTTP header** —оптимальный вариант, при этом могут использоваться и стандартный заголовок Authorization (например, с Basic-схемой), и другие произвольные заголовки.

**SLIDE 6**

###### Распространенные уязвимости и ошибки реализации

Аутентификации по паролю считается не очень надежным способом, так как пароль часто можно подобрать, а пользователи склонны использовать простые и одинаковые пароли в разных системах, либо записывать их на клочках бумаги. Если злоумышленник смог выяснить пароль, то пользователь зачастую об этом не узнает. Кроме того, разработчики приложений могут допустить ряд концептуальных ошибок, упрощающих взлом учетных записей.  
  
Ниже представлен список наиболее часто встречающихся уязвимостей в случае использования аутентификации по паролю:

Веб-приложение позволяет пользователям создавать простые пароли.

Веб-приложение не защищено от возможности перебора паролей (brute-force attacks).

Веб-приложение само генерирует и распространяет пароли пользователям, однако не требует смены пароля после первого входа (т.е. текущий пароль где-то записан).

Веб-приложение допускает передачу паролей по незащищенному HTTP-соединению, либо в строке URL.

Веб-приложение не использует безопасные хэш-функции для хранения паролей пользователей.

Веб-приложение не предоставляет пользователям возможность изменения пароля либо не нотифицирует пользователей об изменении их паролей.

Веб-приложение использует уязвимую функцию восстановления пароля, которую можно использовать для получения несанкционированного доступа к другим учетным записям.

Веб-приложение не требует повторной аутентификации пользователя для важных действий: смена пароля, изменения адреса доставки товаров и т. п.

Веб-приложение создает session tokens таким образом, что они могут быть подобраны или предсказаны для других пользователей.

Веб-приложение допускает передачу session tokens по незащищенному HTTP-соединению, либо в строке URL.

Веб-приложение уязвимо для session fixation-атак (т. е. не заменяет session token при переходе анонимной сессии пользователя в аутентифицированную).

Веб-приложение не устанавливает флаги HttpOnly и Secure для browser cookies, содержащих session tokens.

Веб-приложение не уничтожает сессии пользователя после короткого периода неактивности либо не предоставляет функцию выхода из аутентифицированной сессии.

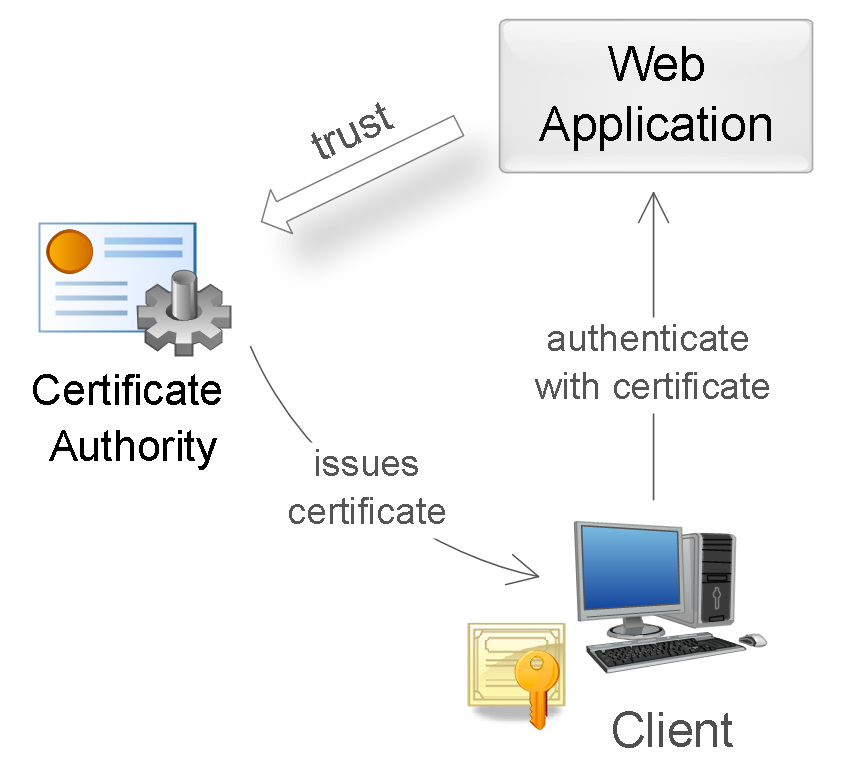
SLIDE 7

Аутентификация по сертификатам

Сертификат представляет собой набор атрибутов, идентифицирующих владельца, подписанный certificate authority (CA). CA выступает в роли посредника, который гарантирует подлинность сертификатов (по аналогии с ФМС, выпускающей паспорта). 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.



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

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

SLIDE 8

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.

1.Аппаратные или программные токены, которые могут генерировать одноразовые пароли на основании секретного ключа, введенного в них, и текущего времени. Секретные ключи пользователей, являющиеся фактором владения, также хранятся на сервере, что позволяет выполнить проверку введенных одноразовых паролей. Пример аппаратной реализаций токенов — RSA SecurID; программной — приложение Google Authenticator.

2.Случайно генерируемые коды, передаваемые пользователю через SMS или другой канал связи. В этой ситуации фактор владения — телефон пользователя (точнее — SIM-карта, привязанная к определенному номеру).

3.Распечатка или scratch card со списком заранее сформированных одноразовых паролей. Для каждого нового входа в систему требуется ввести новый одноразовый пароль с указанным номером.

SLIDE 9

Аутентификация по ключам доступа

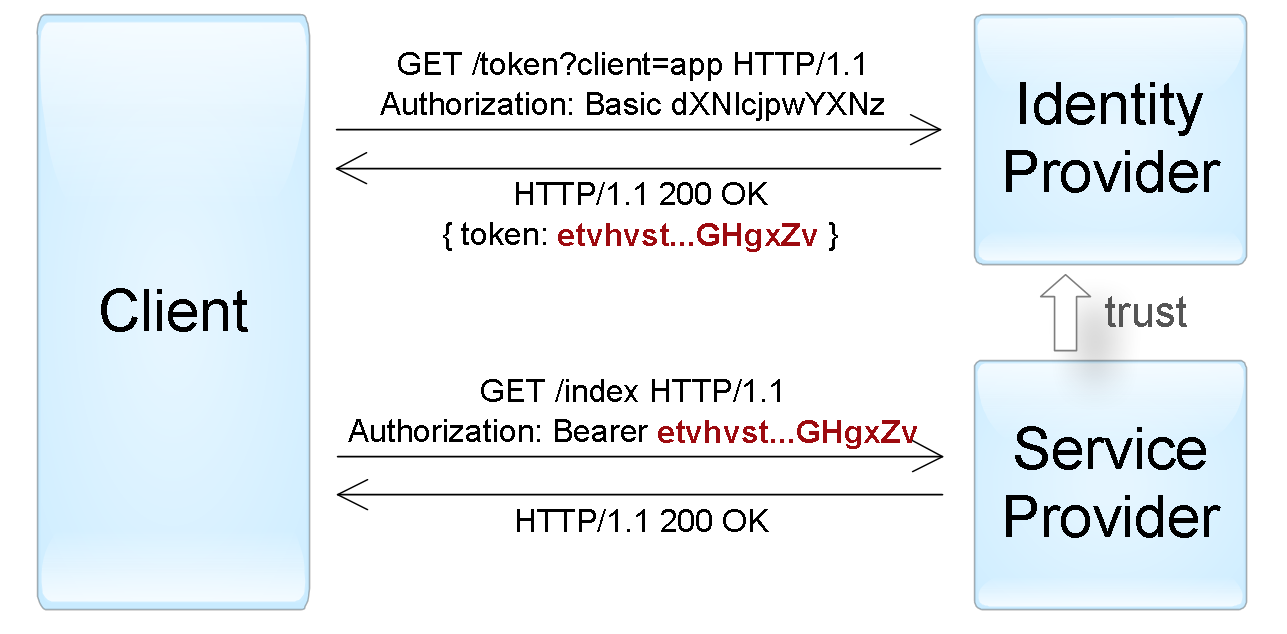
Этот способ чаще всего используется для аутентификации устройств, сервисов или других приложений при обращении к веб-сервисам. Здесь в качестве секрета применяются ключи доступа (access key, API key) — длинные уникальные строки, содержащие произвольный набор символов, по сути заменяющие собой комбинацию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

SLIDE 10

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.

На общем уровне, весь процесс выглядит следующим образом:



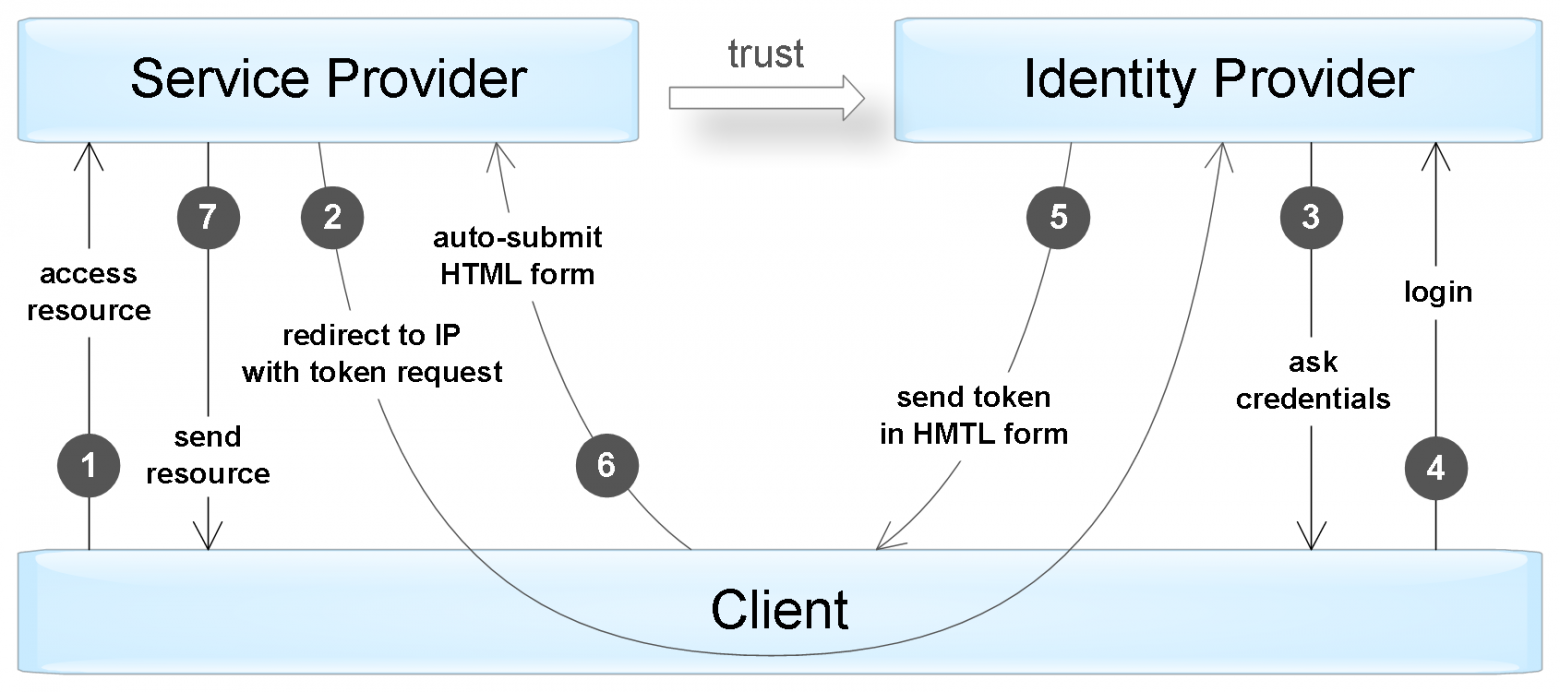
Клиент аутентифицируется в identity provider одним из способов, специфичным для него (пароль, ключ доступа, сертификат, Kerberos, итд.).

Клиент просит identity provider предоставить ему токен для конкретного SP-приложения. Identity provider генерирует токен и отправляет его клиенту.

Клиент аутентифицируется в SP-приложении при помощи этого токена.

SLIDE 11

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.



SLIDE 12

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.

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

Токен был выдан доверенным identity provider приложением (проверка поля issuer).

Токен предназначается текущему SP-приложению (проверка поля audience).

Срок действия токена еще не истек (проверка поля expiration date).

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.

SLIDE 14 Форматы токенов

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.
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.
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.

SLIDE 15

Стандарты OAuth и OpenID Connect

В отличие от SAML и WS-Federation, стандарт OAuth (Open Authorization) не описывает протокол аутентификации пользователя. Вместо этого он определяет механизм получения доступа одного приложения к другому от имени пользователя. Однако существуют схемы, позволяющие осуществить аутентификацию пользователя на базе этого стандарта (об этом — ниже).

Первая версия стандарта разрабатывалась в 2007 – 2010 гг., а текущая версия 2.0 опубликована в 2012 г. Версия 2.0 значительно расширяет и в то же время упрощает стандарт, но обратно несовместима с версией 1.0. Сейчас OAuth 2.0 очень популярен и используется повсеместно для предоставления делегированного доступа и третье-сторонней аутентификации пользователей.

Чтобы лучше понять сам стандарт, рассмотрим пример веб-приложения, которое помогает пользователям планировать путешествия. Как часть функциональности оно умеет анализировать почту пользователей на наличие писем с подтверждениями бронирований и автоматически включать их в планируемый маршрут. Возникает вопрос, как это веб-приложение может безопасно получить доступ к почте пользователей, например, к Gmail?

> Попросить пользователя указать данные своей учетной записи? — плохой вариант.

> Попросить пользователя создать ключ доступа? — возможно, но весьма сложно.

Как раз эту проблему и позволяет решить стандарт OAuth: он описывает, как приложение путешествий (client) может получить доступ к почте пользователя (resource server) с разрешения пользователя (resource owner). В общем виде весь процесс состоит из нескольких шагов:

Пользователь (resource owner) дает разрешение приложению (client) на доступ к определенному ресурсу в виде гранта. Что такое грант, рассмотрим чуть ниже.

Приложение обращается к серверу авторизации и получает токен доступа к ресурсу в обмен на свой грант. В нашем примере сервер авторизации — Google. При вызове приложение дополнительно аутентифицируется при помощи ключа доступа, выданным ему при предварительной регистрации.

Приложение использует этот токен для получения требуемых данных от сервера ресурсов (в нашем случае — сервис Gmail).

SLIDE 16 Взаимодействие компонентов в стандарте OAuth.

SLIDE 17 Виды грантов

Стандарт описывает четыре вида грантов, которые определяют возможные сценарии применения:

Authorization Code — этот грант пользователь может получить от сервера авторизации после успешной аутентификации и подтверждения согласия на предоставление доступа. Такой способ наиболее часто используется в веб-приложениях. Процесс получения гранта очень похож на механизм аутентификации пассивных клиентов в SAML и WS-Federation.

Implicit — применяется, когда у приложения нет возможности безопасно получить токен от сервера авторизации (например, JavaScript-приложение в браузере). В этом случае грант представляет собой токен, полученный от сервера авторизации, а шаг № 2 исключается из сценария выше.

Resource Owner Password Credentials — грант представляет собой пару username/password пользователя. Может применяться, если приложение является «интерфейсом» для сервера ресурсов (например, приложение — мобильный клиент для Gmail).

Client Credentials — в этом случае нет никакого пользователя, а приложение получает доступ к своим ресурсам при помощи своих ключей доступа (исключается шаг № 1).

Стандарт не определяет формат токена, который получает приложение: в сценариях, адресуемых стандартом, приложению нет необходимости анализировать токен, т. к. он лишь используется для получения доступа к ресурсам. Поэтому ни токен, ни грант сами по себе не могут быть использованы для аутентификации пользователя. Однако если приложению необходимо получить достоверную информацию о пользователе, существуют несколько способов это сделать:

1. Зачастую API сервера ресурсов включает операцию, предоставляющую информацию о самом пользователе (например, /me в Facebook API). Приложение может выполнять эту операцию каждый раз после получения токена для идентификации клиента. Такой метод иногда называют *псевдо-аутентификацией*.
2. Использовать стандарт **OpenID Connect**, разработанный как слой учетных данных поверх OAuth (опубликован в 2014 г.). В соответствии с этим стандартом, сервер авторизации предоставляет дополнительный identity token на шаге № 2. Этот токен в формате JWT будет содержать набор определенных полей (claims) с информацией о пользователе.

SLIDE 18 Заключение

|  |  |  |
| --- | --- | --- |
| **Способ** | **Основное применение** | **Протоколы** |
| **По паролю** | Аутентификация пользователей | HTTP, Forms |
| **По сертификатам** | Аутентификация пользователей в безопасных приложениях; аутентификация сервисов | SSL/TLS |
| **По одноразовым паролям** | Дополнительная аутентификация пользователей (для достижения two-factor authentication) | Forms |
| **По ключам доступа** | Аутентификация сервисов и приложений | - |
| **По токенам** | Делегированная аутентификация пользователей; делегированная авторизация приложений | SAML, WS-Federation, OAuth, OpenID Connect |