Протокол OAuth

**Ваня: слайд 2**

Благодаря открытому протоколу OAuth, необязательно заводить для каждой сайта отдельные логин и пароль – можно авторизоваться через Google, Facebook или другой ресурс. Сегодня мы расскажем, как устроен протокол OAuth 2.0.

Социальные сети, потоковая передача контента, воркспейсы – везде мы заходим через учетные записи, которые могут содержать личную информацию. Изолированные приложения становятся взаимосвязанными: Twitter позволяет новостным сайтам твитить напрямую, Discord ищет предполагаемых друзей на Facebook, а Jira создает учетки с помощью профилей Github.

**Ваня: слайд 3**

Раньше для авторизации сайты и приложения использовали простую схему логин/пароль. Это существенно усложняло задачу взаимодействия приложений: чтобы позволить приложению получить доступ к данным другого приложения, требовалось передать учетные данные. Но это порождало множество проблем, связанных с небезопасным хранением учетных данных и получением неограниченного доступа. Для обеспечения делегированного доступа был создан открытый протокол авторизации OAuth.

**Рома: слайд 4**

Сразу отметим, что протокол OAuth 2.0 обратно не совместим с протоколом OAuth 1.0[1]. Вместо того, чтобы дополнить OAuth 1.0, было принято решение разработать другой протокол[10]. Поэтому принцип работы OAuth 1.0 и OAuth 2.0 отличается. OAuth 2.0 это протокол авторизации, основанный на HTTP, что дает возможность применять его практически на любой платформе. Но общие принципы все равно прослеживаются, поэтому дальше речь пойдет именно об второй версии данного протокола.

**Рома: слайд 5**

Сначала хотелось бы привести небольшой пример из жизни. Допустим, у нас есть крутая машина. Мы собираемся и едем в ресторан. Там, соответственно, нас встречает мужик, который паркует машины. И тут возникает проблема. У нас в машине есть некоторые важные вещи, которые мы никак не можем взять с собой. Но производитель машины все предусмотрел и изначально выдал нам два ключа: так называемый Валлет key и Мастер key. Мастер ключ позволяет открывать бордочок, где у нас лежит миллион долларов, а Валлет ключ позволяет открыть двери и управлять двигателем. Так вот, такой Валлет ключ позволяет делегировать часть своих полномочий. То есть, владелец отдает

Валлет ключ парковщику и идет с Мастер ключом ужинать в ресторан. В итоге, предоставляется ограниченный доступ к ресурсам третьей стороне. В этом и суть OAuth 2.0. При этом парковщик не знает, что за человек этот собственник, но должен в итоге вернуть Валлет ключ.

Таким образом OAuth – это открытый протокол, стандарт авторизации, то есть протокол, отвечающий за выдачу прав доступа к ресурсам пользователя. Иногда встречается информация, что он отвечает и за аутентификацию, но это не так. Да, в рамках авторизации произойдет аутентификация, но сам протокол не об этом. Вследствие доработанный протокол и получил название OAuth 2.0.

**Ваня: слайд 6**

А теперь подробнее разберем более конкретный пример, а также роли, которые есть в этом протоколе. Допустим есть обычный человек – владелец ресурсов. Пусть это будут 18+ фотографии. При этом такие фотографии он хранит на гугл диске. Допустим, наш человек нашел некий сайт или приложение, которое обещает удалить повторяющиеся или похожие фотографии по какому-то алгоритму, а может даже отсортировать, если будет выбрана эта опция. Владелец должен предоставить доступ к гугл диску, чтобы приложение не сделало ничего лишнего. Для этого будет использоваться авторизационный сервер гугла.

Итак, определим роли:

1) Владелец ресурсов, человек или сервер, который может предоставить доступ к защищенным ресурсам. Когда человек является владельцем, он называется конечным пользователем.
2) Сервер ресурсов, на котором размещены защищенные ресурсы.
3) Клиент, то есть приложение, которое делает защищенные запросы к серверу ресурсов от имени владельца и с его разрешения.
4) Сервер авторизации. Это гугловский сервер авторизации.

Сервер авторизации должен знать наше приложение, для этого приложение регистрируют на авторизационном сервере и сервер выдает приложению так называемые client_id и client secret. Client_id – идентификатор нашего приложения, его можно показывать другим. Client_secret – секретный ключ, его нельзя показывать. Кстати, сервер авторизации может быть тем же сервером, что и сервер ресурсов или, как в нашем случае, отдельным объектом. Запросы к серверу авторизации включают некий scope – перечень запросов на доступ к ресурсам, которые нужны для работы приложения. В нашем случае, это разрешение для работы с фотографиями на гугл диске. Условно можно выделить четыре типа областей:

1) доступ для чтения;
2) доступ на запись;

3) доступ для чтения и записи;
4) без доступа.

А теперь кое-что интересное.

Разрешения (grants) диктуют клиенту порядок операций по получению access-токена. Этот уникальный порядок еще называется потоком.

Вся коммуникация между владельцем ресурса, клиентом и сервером авторизации происходит через строки запроса с параметрами. В этих параметрах содержится информация, необходимая серверу авторизации для понимания того, какому потоку следовать.

**Рома: слайд 7**

Согласно OAuth 2.0 существует 4 типа предоставления разрешения к ресурсам:

1) Authrization Code Grant
2) Implicit Grant
3) Resource Owner Password Credentials Grant
4) Client Credentials Grant

И вот рассмотрим их поочередно.

Authrization Code Grant: У нас есть 4 роли. В данном типе наши роли клиента и сервера авторизации будут разбиты на части. Итак пусть человек берет в руки телефон и хочет управлять с помощью некого сайта или приложения управлять контентом на гугл диске. Сайт, или клиент, вместе со Scope разрешений и Client_id перебрасывает человека на страницу, которая является интерфейсом, для работы с гугловским сервером авторизации. Помните же эту пресловутую кнопочку авторизации с помощью гугла и что идет после нее? Итак, человек логинится в службе гугла и высвечивает, известное всем, уведомление о том, какие разрешения будут переданы. Что-то типо: Вы предоставляется доступ к…, где дальше будет кнопка согласится или отказаться. И вот разрешения, которые будут там указаны по итогу будут переданы клиенту. Затем приложение, с помощью API получает от авторизационного сервера Authorization Code. После этого, сервер приложения делает еще один запрос на сервер авторизации, но уже посылает только что полученный Authorization Code и Client_secret для получения Access token. Тут сразу стоит сказать, что же это такое. Представьте, что это некая карточка доступа в закрытое ограниченное помещение, причем доступ предоставляется на определенное количество времени. Ну и наконец, сервер клиент шлет токен доступа на сервер ресурсов, для получения данных. Но, опять же, по истечении срока действия Access token, его нужно обновлять. Наверняка вы встречали такое, что приходится несколько раз логиниться через гугл аккаунт в одном и том же приложении. И в итоге сервер ресурсов посылает данные приложению. Теперь ваши 18+ фото находятся у стороннего

приложения. Кстати, уведомление о разрешении доступа должно предоставляться один раз, если Scope разрешений не менялся.

**Ваня: слайд 21**

Теперь рассмотрим Implicit Grant: Вообще, это упрощенная версия Authrization Code Grant. Implicit переводится как неявный. Неявным он называется потому что приложение получает Access token сразу от авторизационного сервера без схемы с Authorization Code. В этом случае исчезает необходимость в клиентском сервере. Потому что токен теперь идет через фронт в сторону клиента. Принцип тут такой же. Владелец ресурсов использует сайт, сайт вместе со Scope разрешений, перекидывает Client_id на авторизационный сервер. Владелец ресурсов логинится в службе гугла с уведомлением, какие разрешения будут переданы приложению. Авторизационный сервер посылает токен доступа. И с этим токеном приложение уже обращается к серверу ресурсов для получения данных. Этот тип был разработан для оптимизации работы клиентов веб-сайтов, реализованных в браузере на языке Java-script. В итоге это улучшает скорость отклика и сокращает количество обратных обращений. Но это так же негативно влияет на безопасность, поэтому рекомендуется использовать первый тип.

**Рома: слайд 25**

Третий тип: Resource Owner Password Credentials Grant. Сравнивая этот тип с предыдущим, можно отметить, что теперь владелец ресурса вводит пароль и логин от гугл диска сразу на сайте(приложении), поэтому приложению уже не нужно логиниться на сервере авторизации. Теперь сайт отправляет на авторизационный сервер сразу Scope, Client_id, логин и пароль от гугл аккаунта человека. Приложение получает от сервера авторизации Access Token и приложение отправляет на сервер этот токен, с последующим получением в ответ данных. В данном типе используются учетные данные владельца ресурса и логин с паролем довереются, то есть отдаются приложению. Соответственно, приложение должно иметь высокую степень доверия по отношению к владельцу ресурса. Этот тип разработан как вариант, если клиент является частью операционный системы устройства, где хранятся ресурсы. И использовать этот тип не лучший вариант в плане безопасности.

**Рома: слайд 31**

Последний вариант: Client Credentials Grant. Работает, когда клиент является владельцем ресурса. Фактически, этот тип используется между серверами, когда интерфейс приложения отсутствует в цепочке системы. В отличие от предыдущего примера, вместо пароля и логина от ресурса сервер(приложение) посылает на авторизационный сервер сразу Scope, Client_id и Client_secret. А в ответ приходит уже Access token. А уже токен посылается на сервер ресурсов с ответом в виде данных. Интересно, что в виду отсутствия интерфейса приложения, Client_secret не будет подсмотрен и украден.