**Реализация SASL аутентификации.**

<http://post.hppi.troitsk.ru/~mike/sasl_krb/sasl_krb.html>

SASL (простой уровень аутентификации и безопасности) — это фреймворк (каркас) для предоставления аутентификации и защиты данных в протоколах на основе соединений. Он разделяет механизмы аутентификации от прикладных протоколов, в теории позволяя любому механизму аутентификации, поддерживающему SASL, быть использованным в любых прикладных протоколах, которые используют SASL. Фреймворк также предоставляет слой защиты данных. Для использования SASL протокол включает команду для идентификации и аутентификации пользователя на сервере и для опциональной защиты переговоров последующей интерактивности протокола. Если это используется в переговорах, то слой безопасности вставляется между протоколом и соединением.

Механизмы SASL реализуют серию запросов и ответов. Определенные SASL механизмы включают:

* «EXTERNAL», используется, когда аутентификация отделена от передачи данных (например, когда протоколы уже используют IPsec или TLS);
* «ANONYMOUS», для аутентификации гостевого доступа;
* «PLAIN», простой механизм передачи паролей открытым текстом. PLAIN является заменой устаревшему LOGIN ;
* «OTP», механизм одноразовых паролей. OTP заменяет устаревший механизм SKEY;
* «CRAM-MD5», в криптографии является механизмом аутентификации вида запрос-ответ;
* «DIGEST-MD5», алгоритм хеширования, предназначен для создания «отпечатков» или дайджестов (хеш-сумма) сообщения произвольной длины и последующей проверки их подлинности;
* «NTLM», использует три сообщения для аутентификации клиента в среде, ориентированной на соединение и четв`ртое дополнительное сообщение, если требуется проверка целостности;
* «GSSAPI», API для доступа к сервисам безопасности.

SASL был задуман в качестве своего рода диспетчера, позволяющего сетевым клиентам и серверам, договорится о механизме аутентификации, который клиент должен использовать, чтобы подключиться к серверу. Более того, библиотека, входящая в состав SASL, позволяет унифицированным образом провести эту аутентификацию независимо от выбранного механизма. Прозрачность процедуры аутентификации через SASL позволяет как клиенту, так и серверу одновременно поддерживать несколько различных типов аутентификации без необходимости программно реализовывать каждый из них - разработчик может положиться на библиотеку SASL, что та выберет наиболее подходящий из набора, предлагаемого клиентом или сервером.

Клиент и сервер могут находится в разных сетях, которые построены и сконфигурированы разными администраторами, каждый из которых имеет свой взгляд на политику сетевой безопасности. Предположим, что один из них сконфигурировал сервис таким образом, что тот позволяет аутентифицировать пользователя через GSSAPI, с помощью пароля, закодированного по n алгоритму CRAM-MD5 и простого текстового пароля PLAIN. Если в клиенте настроена поддержка какого-нибудь из этих механизмов, то SASL позволяет провести аутентификацию посредством именно этого механизма, причем в случае множественного пересечения будет выбран наиболее сильный. Программная реализация выглядит одинаково - разработчик должен инициализировать библиотеку и затем войти в цикл аутентификации. На каждом шаге цикла программа должна прочитать блок данных, полученный по сети (реплика клиента/сервера) и передать его функции. На выходе функции контролируется возвращаемое значение и один из ссылочных параметров. Ссылочный параметр передается по сети без дополнительной обработки, а возвращаемое значение анализируется и в зависимости от его значение принимается решение о продолжении или остановки цикла аутентификации. Цикл прекращается, когда возвращаемое значение станет равным SASL\_OK, что означает успешную аутентификацию, или функция вернет сообщение об ошибке, что означает отказ в аутентификации. Хотя число шагов, которое требуется для подтверждения идентичности пользователя может быть различно для разных механизмов аутентификации, да и сам способ ввода пароля (или его отсутствие) в клиентском приложении может быть различным, но сам алгоритм аутентификации при этом остается тем же самым.

**Использование JWT токенов.**

JSON Web Token (JWT) — это JSON объект. Он считается одним из безопасных способов передачи информации между двумя участниками. Для его создания необходимо определить заголовок (header) с общей информацией по токену, полезные данные (payload), такие как id пользователя, его роль и т.д. и подписи (signature).

Простыми словами, JWT — это лишь строка в следующем формате: header.payload.signature.

Предположим, что мы хотим зарегистрироваться на сайте. В нашем случае есть три участника — пользователь user, сервер приложения application server и сервер аутентификации authentication server. Сервер аутентификации будет обеспечивать пользователя токеном, с помощью которого он позднее сможет взаимодействовать с приложением.

Приложение использует JWT для проверки аутентификации пользователя следующим образом:

1. Сперва пользователь заходит на сервер аутентификации с помощью аутентификационного ключа (это может быть пара логин/пароль, либо Facebook ключ, либо Google ключ, либо ключ от другой учётки);
2. Затем сервер аутентификации создает JWT и отправляет его пользователю;
3. Когда пользователь делает запрос к API приложения, он добавляет к нему полученный ранее JWT;
4. Когда пользователь делает API запрос, приложение может проверить по переданному с запросом JWT является ли пользователь тем, за кого себя выдает. В этой схеме сервер приложения сконфигурирован так, что сможет проверить, является ли входящий JWT именно тем, что был создан сервером аутентификации.

Header JWT содержит информацию о том, как должна вычисляться JWT подпись. Хедер — это тоже JSON объект.

Payload — это полезные данные, которые хранятся внутри JWT. Эти данные также называют JWT-claims (заявки). Эти поля могут быть полезными при создании JWT, но они не являются обязательными. Но стоит помнить, что чем больше передается информации, тем больший получится в итоге сам JWT. Обычно с этим не бывает проблем, но все-таки это может негативно сказаться на производительности и вызвать задержки во взаимодействии с сервером.

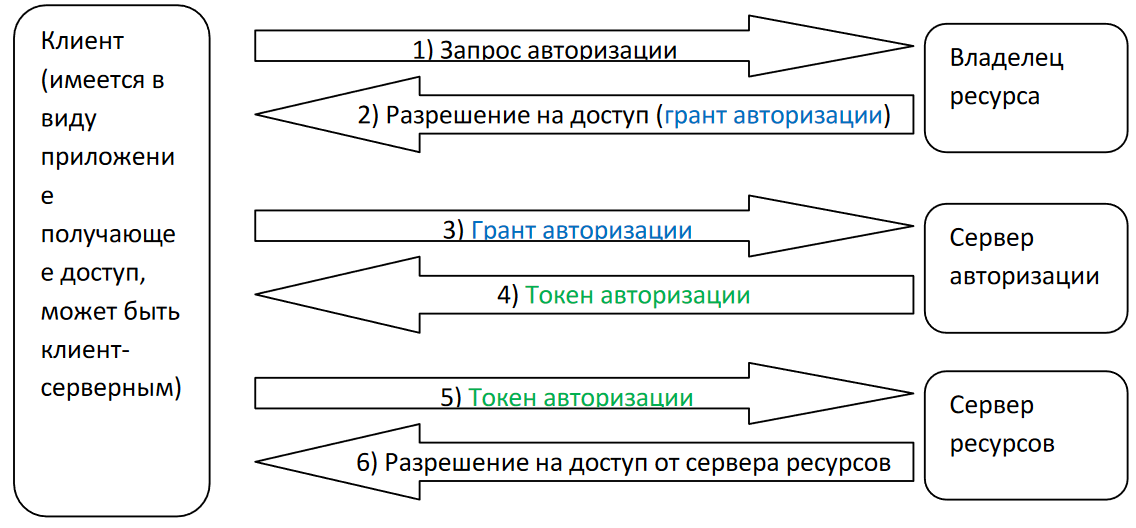
Signature. Алгоритм (возможно base64url) кодирует хедер и payload, созданные ранее. Алгоритм соединяет закодированные строки через точку. Затем полученная строка хешируется алгоритмом, заданным в хедере на основе нашего секретного ключа.

Очень важно понимать, что использование JWT не скрывает и не маскирует данные автоматически. Причина, почему JWT используются — это проверка, что отправленные данные были действительно отправлены авторизованным источником. Как было продемонстрировано выше, данные внутри JWT закодированы и подписаны, обратите внимание, это не одно и тоже, что зашифрованы. Цель кодирования данных — преобразование структуры. Подписанные данные позволяют получателю данных проверить аутентификацию источника данных.

**Реализация различных способов доступа Auth 2.0.**

Является открытым фреймворком (протоколом) авторизации, позволяющим получить сторонним приложениям ограниченный доступ к ресурсам HTTP-сервиса, позволяет предоставить права на использование некоторого ресурса (например, API какого-либо сервиса). При этом в общем случае нельзя определить, кто в настоящий момент пользуется правами. Схематично сценарий протокола OAuth 2.0 представлен на рисунке. Он описывает взаимодействие между упомянутыми выше четырьмя сторонами и включает следующие шаги:

1. Клиент запрашивает авторизацию у владельца ресурса. Запрос авторизации может быть направлен владельцу ресурса напрямую, как показано на рисунке, или косвенно через сервер авторизации. Предпочтительным является второй вариант.
2. Клиент получает разрешение на доступ (grant), структуру данных, представляющую авторизацию владельца ресурса, выраженную с использованием одного из четырех типов разрешений: код авторизации (authorization code), неявное разрешение (implicit), пароль владельца ресурса (resource owner password credentials) и учетные данные клиента (client credentials), так же есть еще Device authorization для доступа устройств. Тип разрешения на доступ зависит от метода, используемого клиентом для запроса авторизации, и типов разрешений, поддерживаемых сервером авторизации. Типы разрешений, поддерживаемые сервером авторизации, определяются при его разработке, исходя из его прикладных целей и задач. Настоящий стандарт регламентирует использование в качестве типа разрешения код авторизации.
3. Клиент запрашивает токен доступа посредством аутентификации на сервере авторизации и предоставления разрешения на доступ.
4. Сервер авторизации аутентифицирует клиента, проверяет разрешение на доступ и, если оно действительно, выдает токен доступа.
5. Клиент запрашивает защищенный ресурс на сервере ресурсов и аутентифицируется, представляя токен доступа.
6. Сервер ресурсов проверяет токен доступа и, если он действителен, обслуживает запрос.



Стандарт OAuth 2.0 определяет следующие четыре роли:

* владелец ресурса — сущность, обладающая правом на выдачу доступа к защищенным ресурсам. В случае, если владелец является человеком, его называют конечным пользователем;
* сервер ресурсов — сервер, содержащий защищаемые ресурсы и обладающий возможностью получения и формирования ответа на запросы к защищаемым ресурсам посредством использования маркера доступа;
* клиент — приложение, осуществляющее доступ к защищенным ресурсам от имени Владельца. Термин "клиент" явно не определяет какое-либо конкретное исполнение (будь то сервер, персональный компьютер или мобильное приложение); Приложением может являться браузер, толстый клиент или мобильное приложение. Клиент может быть public и confedential. Public не может безопасно хранить свои учётные данные. Этот клиент работает на устройстве владельца ресурса, например, это браузерные или мобильные приложения. Confedential может безопасно хранить свои учетные данные, например бэкенд приложения.
* сервер авторизации — сервер, осуществляющий выпуск маркеров доступа для клиентских приложений после успешной аутентификации и авторизации Владельца ресурсов.

Протокол OAuth обладает возможностью аутентификации не только Пользователя, но и клиентского приложения, осуществляющего доступ к ресурсам.

Рассмотрим пять способов получения доступа (grant) в auth 2.0, начиная с самого простого.

1. Client credentials grant flow

Предполагает самый простой способ получения прав доступа к ресурсам. Клиент отправляет на сервер авторизации client id и client secret, на что возвращается access token с которым клиент может обратиться к серверу ресурсов. Данный способ используется для доступа к собственным ресурсам или предоставление доступа к серверу ресурсов, согласованному с сервером авторизации, когда клиент совпадает с владельцем ресурсов. Допускается только для защищенных клиентов (confedential, клиент, который может безопасно хранить свои учётные данные, к такому типу клиентов относят web-приложения, имеющие backend.).

1. Authorization code flow

Самая распространенная схема и одна из защищенных. На первом шаге клиент перенаправляет resource owner с помощью user-agent на страницу аутентификации Authorization server. В URI он указывает client ID и redirection URI. Redirection URI используется для понимания, куда вернуть resource owner после того, как авторизация пройдёт успешно. Взаимодействуя с сервером авторизации через user-agent, resource owner проходит аутентификацию на сервере авторизации. Resource owner проверяет права, которые запрашивает клиент на consent screen и разрешает их выдачу. Resource owner возвращается клиенту с помощью user-agent обратно на URI, который был указан как redirection URI. В качестве query-параметра будет добавлен authorization code — строка, подтверждающая то, что resource owner выдал необходимые права сервису. С этим authorization code клиент отправляется на сервер авторизации, чтобы получить в ответ access token. Сервер авторизации валидирует authorization code, убеждаясь, что токен корректный и выдаёт клиенту access token. С его помощью клиент сможет получить доступ к заветному ресурсу.

1. Implicit grant flow

Неявный доступ, но он не рекомендуется. В данном случае вместо кода сразу возвращается access token, это небезопасно. В другом варианте пароль и логин передается через клиента на сервер авторизации, что, совсем небезопасно потому, что пароль и логин пользователя становятся известны клиентскому приложению.

1. Authorization Code Flow with Proof Key for Code Exchange (PKCE)

Это усовершенствованный вариант Authorization code flow. В нем происходит генерация еще двух случайных кодов: Code Verifier и Code Challenge. Авторизация реализуется с посылкой Code Challenge, после введения пользователем логина и пароля авторизационным сервером отдается Authorization Code, далее идет обращение за access token с только что выданным Authorization Code и с Code Verifier. В результате Identity-провайдер (Авторизационный сервер) валидирует на основании трех кодов. Если злоумышленник получит один код из системы, он ничего не сможет сделать с ним — нужны все три кода.

**Authorization code. Implicit flow. Client credentials. Device authorization.**

Существует 4 способа получения grant — grant type:

1. Authorization code — используется для confedencial клиентов — web-сервисов.
2. Client credentials — используется для confedential клиентов, которые запрашивают доступ к своим ресурсам или ресурсам, заранее согласованным с сервером авторизации.
3. Implicit — использовался public-клиентами, которые умеют работать с redirection URI (например, для браузерных и мобильных приложений), но был вытеснен authorization code grant с PKCE (Proof Key for Code Exchange — дополнительная проверка, позволяющая убедиться, что token получит тот же сервис, что его и запрашивал. Прочитать подробнее — RFC 7636).

Resource owner password credentials. В RFC 6819, посвящённому безопасности в OAuth 2.0, данный тип grant считается ненадёжным. Если раньше его разрешалось использовать только для миграции сервисов на OAuth 2.0, то в данный момент его не разрешено использовать совсем.

Device authorization (добавлен в RFC 8628) – используется для авторизации устройств, которые могут не иметь веб-браузеров, но могут работать через интернет. Например, это консольные приложения, умные устройства или Smart TV.

Digest аутентификация.

CRAM-md5.

ОAuth.

Протокол SSL.

Протокол TLS.