DataArt 16 июля 2015 в 19:35

Обзор способов и протоколов аутентификации в веб-приложениях

Блог компании DataArt, Информационная безопасность, Разработка веб-сайтов, Программирование

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.

Перед тем, как перейти к техническим деталям, давайте немного освежим терминологию.

- **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.

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.

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:

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.

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.

Пример HTTP аутентификации с использованием Basic схемы.

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.

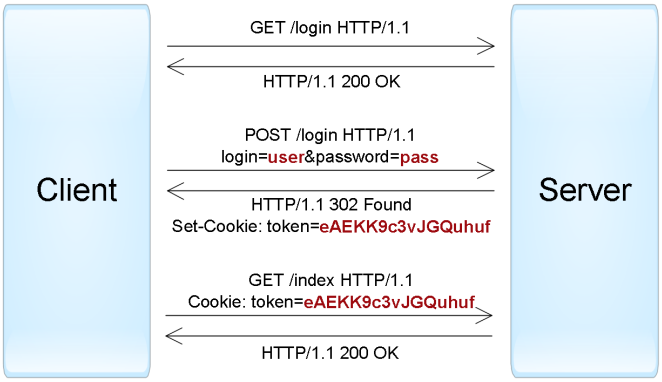
4. **Negotiate** — еще одна схема из семейства Windows authentication, которая позволяет клиенту выбрать между NTLM и Kerberos аутентификацией. Kerberos — более безопасный протокол, основанный на принципе Single Sign-On. Однако он может функционировать, только если и клиент, и сервер находятся в зоне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

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.



Пример 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. Несколько стандартных форматов таких токенов рассматриваются в секции «Аутентификация по токенам».

Необходимо понимать, что перехват session token зачастую дает аналогичный уровень доступа, что и знание username/password. Поэтому все коммуникации между клиентом и сервером в случае forms authentication должны производиться только по защищенному соединению HTTPS.

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:

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.

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.**

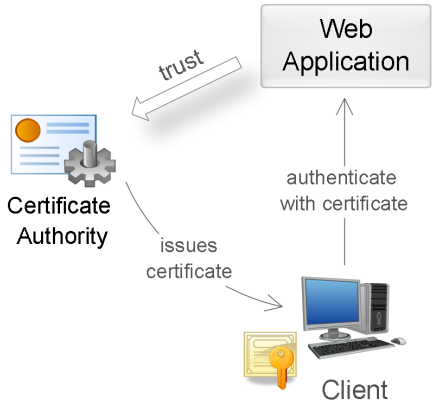
- **Веб-приложение не уничтожает сессии пользователя после короткого периода неактивности либо не предоставляет функцию выхода из аутентифицированной сессии.**

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.

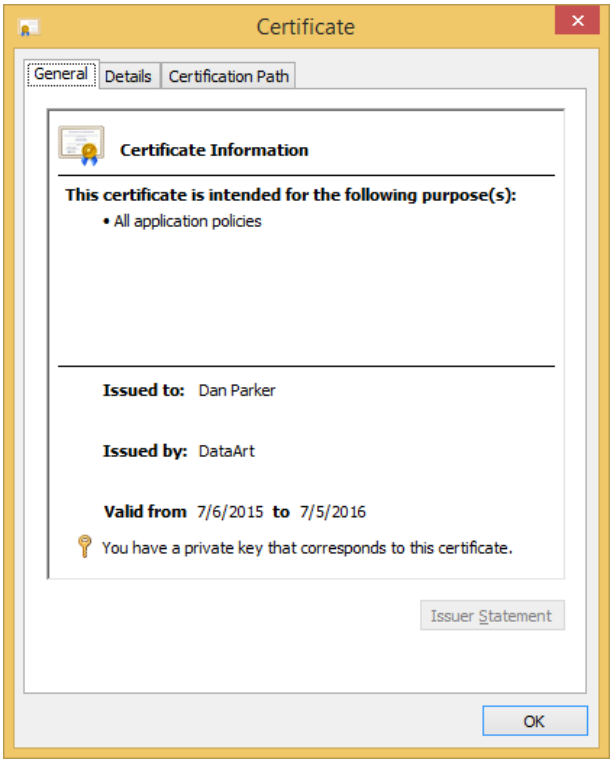
В веб-приложениях традиционно используют сертификаты стандарта X.509. Аутентификация с помощью X.509-сертификата происходит в момент соединения с сервером и является частью протокола SSL/TLS. Этот механизм также хорошо поддерживается браузерами, которые позволяют пользователю выбрать и применить сертификат,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



Использование сертификата для аутентификации.

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).



Пример X.509 сертификата.

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).

Использование сертификатов для аутентификации — куда более надежный способ, чем аутентификация посредством паролей. Это достигается созданием в процессе аутентификации цифровой подписи, наличие которой доказывает факт применения закрытого ключа в конкретной ситуации (non-repudiation). Однако трудности с распространением и поддержкой сертификатов делает такой способ аутентификации малодоступным в широких кругах.

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.

Существуют разные источники для создания одноразовых паролей. Наиболее популярные:

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.

Аппаратный токен RSA SecurID генерирует новый код каждые 30 секунд.

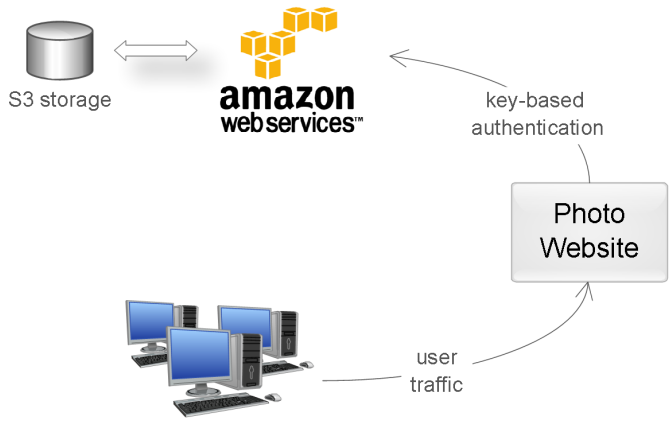
В веб-приложениях такой механизм аутентификации часто реализуется посредством расширения forms authentication: после первичной аутентификации по паролю, создается сессия пользователя, однако в контексте этой сессии пользователь не имеет доступа к приложению до тех пор, пока он не выполнит дополнительную аутентификацию по одноразовому паролю.

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.

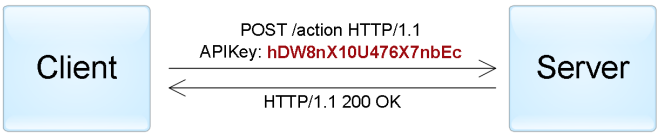
Хороший пример применения аутентификации по ключу — облако Amazon Web Services. Предположим, у пользователя есть веб-приложение, позволяющее загружать и просматривать фотографии, и он хочет использовать сервис Amazon S3 для хранения файлов. В таком случае, пользователь через консоль AWS может создать ключ, имеющий ограниченный доступ к облаку: только чтение/запись его файлов в Amazon S3. Этот ключ в результате можно применить для аутентификации веб-приложения в облаке AWS.



Пример применения аутентификации по ключу.

Использование ключей позволяет избежать передачи пароля пользователя сторонним приложениям (в примере выше пользователь сохранил в веб-приложении не свой пароль, а ключ доступа). 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.



Пример аутентификации по ключу доступа, переданного в HTTP заголовке.

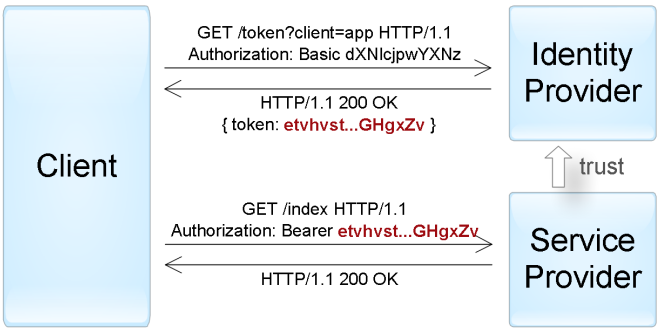
Кроме того, существуют более сложные схемы аутентификации по ключам для незащищенных соединений. В этом случае, ключ обычно состоит их двух частей: публичной и секретной. Публичная часть используется для идентификации клиента, а секретная часть позволяет сгенерировать подпись. Например, по аналогии с 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.

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.

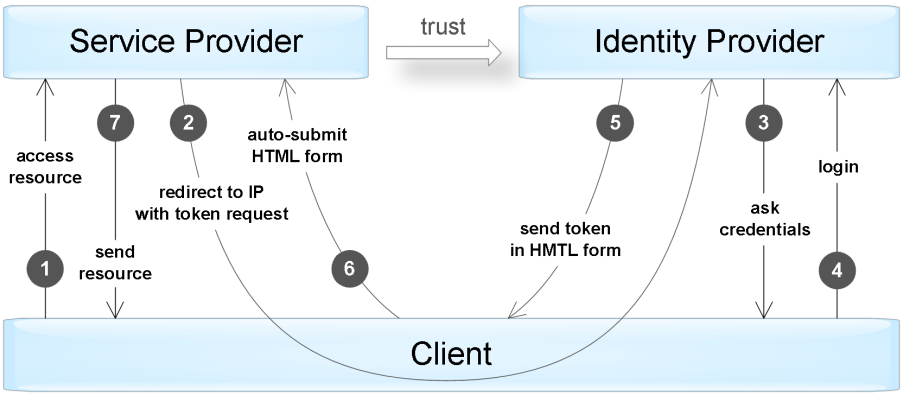
Реализация этого способа заключается в том, что identity provider (IP) предоставляет достоверные сведения о пользователе в виде токена, а service provider (SP) приложение использует этот токен для идентификации, аутентификации и авторизации пользователя.
На общем уровне, весь процесс выглядит следующим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



Пример аутентификации «активного» клиента при помощи токена, переданного посредством Bearer схемы.

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.



Пример аутентификации «пассивного» клиента посредством перенаправления запросов.

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. Некоторая информация об этих протоколах — ниже в статье.

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:

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.

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
   ```

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
   ```

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.

Стандарт SAML

Стандарт Security Assertion Markup Language (SAML) описывает способы взаимодействия и протоколы между identity provider и service provider для обмена данными аутентификации и авторизации посредством токенов. Изначально версии 1.0 и 1.1 были выпущены в 2002 – 2003 гг., в то время как версия 2.0, значительно расширяющая стандарт и обратно несовместимая, опубликована в 2005 г.

Этот основополагающий стандарт — достаточно сложный и поддерживает много различных сценариев интеграции систем. Основные «строительные блоки» стандарта:

1. **Assertions** — собственный формат SAML токенов в XML формате.
2. **Protocols** — набор поддерживаемых сообщений между участниками, среди которых — запрос на создание нового токена, получение существующих токенов, выход из системы (logout), управление идентификаторами пользователей, и другие.
3. **Bindings** — механизмы передачи сообщений через различные транспортные протоколы. Поддерживаются такие способы, как HTTP Redirect, HTTP POST, HTTP Artifact (ссылка на сообщения), SAML SOAP, SAML URI (адрес получения сообщения) и другие.
4. **Profiles** — типичные сценарии использования стандарта, определяющие набор assertions, protocols и bindings необходимых для их реализации, что позволяет достичь лучшей совместимости. Web Browser SSO — один из примеров таких профилей.

Кроме того, стандарт определяет формат обмена метаинформацией между участниками, которая включает список поддерживаемых ролей, протоколов, атрибутов, ключи шифрования и т. п.

Рассмотрим краткий пример использования SAML для сценария Single Sign-On. Пользователь хочет получить доступ на защищенный ресурс сервис-провайдера (шаг № 1 на диаграмме аутентификации пассивных клиентов). Т. к. пользователь не был аутентифицирован, SP отправляет его на сайт identity provider'а для создания токена (шаг № 2). Ниже приведен пример ответа SP, где последний использует SAML HTTP Redirect binding для отправки сообщения с запросом токена:

```
HTTP/1.1 302 Found
Location: https://idp.example.com/SAML/SSO/Browser?SAMLRequest=aQC5kAsTB...3QIX3
f7M&RelayState=kzdeP2qmdr576&SigAlg=http%3A%2F%2Fwww.w3.org%2F200%2F09%2Fxmldsig
%23rsa-sha1&Signature=OPqCEvoc8Vz0WsQdctKbzLBNj74
```

- URI соответствующего сервиса identity provider'а берется из его метаданных.
- SAMLRequest — XML-запрос на создание нового токена (кодировка deflate + base64).
- RelayState — произвольная строка, описывающая состояние SP (будет возвращена в ответе).
- SigAlg — алгоритм подписи сообщения.
- Signature — подпись сообщения (кодировка base64)

В случае такого запроса, identity provider аутентифицирует пользователя (шаги №3-4), после чего генерирует токен. Ниже приведен пример ответа IP с использованием HTTP POST binding (шаг № 5):

```
...
<body onload="document.forms[0].submit()">
<form method="post" action="https://sp.example.com/SAML/SSO/POST">
    <input type="hidden" name="SAMLResponse" value="XHU2KeRbb... 94X/rX4" />
    <input type="hidden" name="RelayState" value="kzdeP2qmdr576" />
    <input type="submit" value="Submit" />
</form>
...
```

- URI соответствующего сервиса service provider'а берется из его метаданных.
- SAMLResponse — XML-ответ, содержащий SAML assertion (кодировка base64).
- RelayState — строка, переданная в параметре RelayState-запроса.

После того как браузер автоматически отправит эту форму на сайт service provider'а (шаг № 6), последний декодирует токен и аутентифицирует пользователя. По результатам успешной авторизации запроса пользователь получает доступ к запрошенному ресурсу (шаг № 7).

Стандарты WS-Trust и WS-Federation

WS-Trust и WS-Federation входят в группу стандартов WS-*, описывающих SOAP/XML-веб сервисы. Эти стандарты разрабатываются группой компаний, куда входят Microsoft, IBM, VeriSign и другие. Наряду с SAML, эти стандарты достаточно сложные, используются преимущественно в корпоративных сценариях.

Стандарт **WS-Trust** описывает интерфейс сервиса авторизации, именуемого Secure Token Service (STS). Этот сервис работает по протоколу SOAP и поддерживает создание, обновление и аннулирование токенов. При этом стандарт допускает использование токенов различного формата, однако на практике в основном используются SAML-токены.

Стандарт **WS-Federation** касается механизмов взаимодействия сервисов между компаниями, в частности, протоколов обмена токенов. При этом WS-Federation расширяет функции и интерфейс сервиса STS, описанного в стандарте WS-Trust. Среди прочего, стандарт WS-Federation определяет:

- Формат и способы обмена метаданными о сервисах.
- Функцию единого выхода из всех систем (single sign-out).
- Сервис атрибутов, предоставляющий дополнительную информацию о пользователе.
- Сервис псевдонимов, позволяющий создавать альтернативные имена пользователей.
- Поддержку пассивных клиентов (браузеров) посредством перенаправления.

Можно сказать, что WS-Federation позволяет решить те же задачи, что и SAML, однако их подходы и реализация в некоторой степени отличаются.

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).

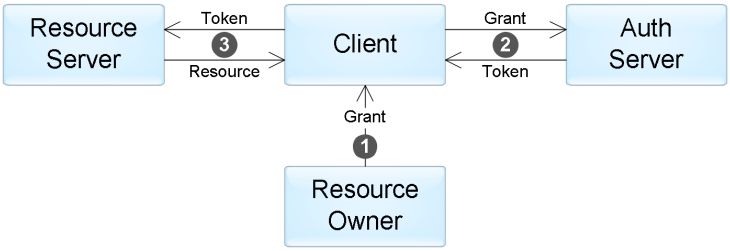
Первая версия стандарта разрабатывалась в 2007 – 2010 гг., а текущая версия 2.0 опубликована в 2012 г. Версия 2.0 значительно расширяет и в то же время упрощает стандарт, но обратно несовместима с версией 1.0. Сейчас OAuth 2.0 очень популярен и используется повсеместно для предоставления делегированного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

1. Пользователь (resource owner) дает разрешение приложению (client) на доступ к определенному ресурсу в виде гранта. Что такое грант, рассмотрим чуть ниже.
2. 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.
3. Приложение использует этот токен для получения требуемых данных от сервера ресурсов (в нашем случае — сервис Gmail).



Взаимодействие компонентов в стандарте OAuth.

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

1. **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.
2. **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.
3. **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).

4. **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.

Стоит заметить, что OpenID Connect, заменивший предыдущие версии стандарта OpenID 1.0 и 2.0, также содержит набор необязательных дополнений для поиска серверов авторизации, динамической регистрации клиентов и управления сессией пользователя.

Заключение

В этой статье мы рассмотрели различные методы аутентификации в веб-приложениях. Ниже — таблица, которая резюмирует описанные способы и протоколы:

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

Надеюсь, что информация оказалась полезна, и вы сможете применить ее при дизайне и разработке новых приложений. До новых встреч!

Автор: Дмитрий Выростков, Solutions Architect в DataArt.

Теги: dataart, безопасность, security

Хабы: Блог компании DataArt, Информационная безопасность, Разработка веб-сайтов, Программирование

+48 1475 362k 19 Поделиться

DataArt
Технологический консалтинг и разработка ПО

153,0 Карма 79,5 Рейтинг

DataArt @DataArt
Пользователь

Facebook Twitter ВКонтакте Instagram Google+

ПОХОЖИЕ ПУБЛИКАЦИИ

25 июля 2019 в 19:53
Музей DataArt. Модемы US Robotics
+21 8,2k 11 48

11 июля 2019 в 19:38
Музей DataArt. Распаковываем и запускаем «Радио-86РК»
+45 17,4k 34 73

20 июня 2019 в 18:46
Музей DataArt. Катушки с ОС 6.1 для ЕС ЭВМ
+26 7,9k 18 76

ВАКАНСИИ КОМПАНИИ DATAART

Salesforce Developer, Financial Software Solutions
DataArt • Воронеж

Больше вакансий компании

Комментарии 19

hudson 16 июля 2015 в 20:10 0

Подскажите пожалуйста, а LDAP/AD аутентификация в данной классификации где находится? Интересует такой вариант SSO, когда веб-приложение аутентифицирует пользователя по его системному логину без ввода какой-либо дополнительной информации в веб приложении.

DataArt 16 июля 2015 в 20:49 0

Аутентификация по паролю -> HTTP authentication -> Kerberos (схема Negotiate).
В этом случае пользователю не придется дополнительно что-либо вводить для аутентификации: браузер автоматически аутентифицируется используя Kerberos тикет, полученный от AD при логине в Windows (по сути это вариант аутентификации по токену). На стороне веб-приложения мы сможем получить информацию о доменном аккаунте аутентифицированного пользователя.

hudson 16 июля 2015 в 20:53 0

Спасибо!

mayorovp 17 июля 2015 в 07:09 +1

LDAP-аутентификация и AD-аутентификация — это совершенно разные вещи. LDAP-аутентификация — это когда веб-приложение принимает от пользователя логин с паролем и проверяйет их по LADP-базе. Этот примитивный способ, в котором SSO и не пахнет, тем не менее, часто выдается за полноценную аутентификацию в AD.

AD-аутентификация — да, она реализуется через схему Negotiate. Но у нее есть недостатки — к примеру, нормально работать с этой схемой может только IE — остальным браузерам понадобится другой способ аутентификации и никакого SSO в итоге не случится.

Поэтому если нужно сделать SSO — надо использовать WS-Federation или OpenID-connect, а уже на стороне Identity Provider можно применить Negotiate как один из вариантов внутренней аутентификации.

mihmig 21 июля 2015 в 11:41 0

ну почему, Гугл Хром нормально работает с NTLM(AD) авторизацией.
Используем в паре проектов — вы не представляете, насколько ниже нагрузка на сопровождающих:
нет ввода пароля — нет его «забывания» и неправильного ввода, блокировок и заявок на разблокировку…
Если пользователь вошёл в домен — значит может и работать в программе.

Один момент только портит всю картину — иногда по непонятным причинам (при заливке и скачиваниии файла) у пользователя IE идёт запрос пароля. Админы домена не могут ничего путного сказать…

mayorovp 21 июля 2015 в 17:10 0

А я вот, приходя на работу, вынужден логиниться сразу аж в трех рабочих сайтах. Сначала пытался изображать из себя крутого безопасника, набирая один и тот же пароль в 4х окнах подряд — но потом устал и запомнил его в браузере.

При этом у тех, кто переносит IE, все работает как задумывалось. И да — админы домена тоже ничего путного сказать не могут.

mihmig 21 июля 2015 в 21:20 0

Дык, у меня Хром-то как раз без сбоев работает, а вот Осёл капризит. Сравним инфраструктуры в ЛС?

mayorovp 21 июля 2015 в 21:35 0

К сожалению, не знаю я нашей инфраструктуры…

Но, в любом случае, выходит что «чистый» Negotiate так и остается мечтой. А значит, при большом числе внутренних ресурсов WS-Federation или OpenID-Connect будут не лишними.

BeLove 17 июля 2015 в 00:53 0

Крутое покрытие темы, спасибо!

Envek 17 июля 2015 в 01:31 0

В прошлом году делал Identity Provider на основе протокола OpenID Connect — очень понравилось. В протоколе есть и аутентификация и авторизация и механизм передачи данных о пользователе от провайдера потребителю. Делалась возможность авторизовывать пользователя на подчинённых сайтах через учётную запись главного сайта, подчинённые сайты получали только те данные, на которые они попросили разрешения (и пользователь одобрил), эти же разрешения позволяли подчинённым сайтам взаимодействовать с API основного сайта от имени пользователя (была возможность засылать уведомления пользователю в личный кабинет). В общем, почти так же круто, как у фейсбука. Самым же весёлым было то, что пользователь на основном сайте мог зарегистрироваться через социальные сети и в результате получалась цепочка из Identity Provider — Service Provider. Использовал гем с внезапным названием openid_connect.

BigD 17 июля 2015 в 08:36 0

Authentication — Authorization — Accounting: вот стандартная триада управления доступами (AAA), про Identification не уверен.

gospodinmir 17 июля 2015 в 10:04 0

Однозначно в «избранное».

Captcha 17 июля 2015 в 11:39 0

Великолепный материал. Хабр — торт!

EvilsInterrupt 17 июля 2015 в 11:41 0

@ DataArt: А что на Ваш взгляд подходит для RESTful приложений?

DataArt 19 июля 2015 в 09:25 0

Зависит от конкретной ситуации, но если нужно аутентифицировать пользователей в RESTful APIs, то логичнее всего будет использовать токены. Как я уже писал, в этом случае клиент (мобильное приложение, браузерное JavaScript приложение или серверное приложение) должно предварительно получить токен от сервера аутентификации. Посмотрите пример архитектуры вот тут: identityserver.github.io/Documentation/docs/overview/bigPicture.html

EvilsInterrupt 19 июля 2015 в 09:51 0

@ DataArt: Я сделал так:
1. Перед каждым действием клиенты запрашивают у моего приложения токены сроком действия на 1 час.
2. В каждом запросе на ту или иную операцию клиенты указывают токены.

Запрос пароля организовал так:
1. В JSON-документе заполняются два атрибута username, password, шлется серверу. Пароли хрянятся в PBKDF2 на сервере, в ответ шлется JWT-токен по алгоритму HS256
2. Действие по пункту выше с использованием HTTPS

dimcha 17 июля 2015 в 14:53 0

Благодарю за материал, очень познавательно, закинул в мемориз ;)

Но у меня есть вопросы:
1. Почему многие сервисы реализуют регистрацию и доступ через OAuth только для нескольких провайдеров (как правило g+ и fb)? Неужели нет стандарта, который упростит добавление подобных возможностей к себе на сайт «одним кликом»?
2. (вытекает из первого) Почему до сих пор не реализована возможность регистрации и доступа к сервисам через кастомных провайдеров OAuth? Например, у меня есть почта, свой блог, свое «облако» (тот-же ownCloud, например) и я не пользуюсь fb и g+. Я ведь мог бы поднять у себя, на своем домене такого провайдера и в SRV записях прописать свой сервер авторизации(CA). А сайт, на котором я регистрируюсь, исходя из домена, который я указал, ломится на мой CA и получает регистрационные данные с него так, как

он это делает в случае с g+? Неужели это никому не нужно?

DataArt 19 июля 2015 в 09:40 0

Спасибо. По #1 — для добавления поддержки нового провайдера требуется зарегистрировать сервис у этого провайдера (получить ключи доступа) и реализовать механизм псевдо-аутентификации (вызовы API этого провайдера). Плюс управление учетными записями пользователей (например, слияние аккаунтов из разных провайдеров). В целом, это несложный процесс, и для многих языков программирования существуют пакеты, которые это упрощают. Но автоматического добавления нового провайдера по одному клику стандарт OAuth не поддерживает.
По #2 — такое предусматривает расширения новейшего стандарта OpenID Connect (http://openid.net/connect/) — посмотрите в секциях Discovery и Dynamic Registrations. Думаю, что в скором времени появятся реализации того, что вы описали, если, конечно, это окажется востребованным.

garbuz 6 октября 2015 в 01:16 0

Мы сейчас используем связку rest + cas + ldap. Cas единожды настраивается на LDAP, потом все клиенты стучаться в кас за аутентификцией или запрашивают тикеты для других приложений, таким образом реализован доступ к рест сервису по тикетам и SSO между приложениями

Paragon открыла свой драйвер NTFS для Linux, предложив включить его в ядро

+62 13,1k 26 15

Разбор: вся изнанка онлайн-школ

Мегапост