В этой главе будут рассмотрены наиболее популярные способы аутентификации.   
Для ясности уточним понятия идентификации, аутентификации и авторизации и разницу между ними.

Идентификация (Identification) — процедура распознавания пользователя по его идентификатору (имени). Эта функция выполняется, когда пользователь делает попытку войти в сеть. Пользователь сообщает системе по ее запросу свой идентификатор, и система проверяет в своей базе данных его наличие.

Аутентификация (Authentication) — процедура проверки подлинности заявленного пользователя, процесса или устройства. Эта проверка позволяет достоверно убедиться, что пользователь (процесс или устройство) является именно тем, кем себя объявляет. При проведении аутентификации проверяющая сторона убеждается в подлинности проверяемой стороны, при этом проверяемая сторона тоже активно участвует в процессе обмена информацией. Обычно пользователь подтверждает свою идентификацию, вводя в систему уникальную, не известную другим пользователям информацию о себе (например, пароль или сертификат).

Идентификация и аутентификация являются взаимосвязанными процессами распознавания и проверки подлинности субъектов (пользователей). Именно от них зависит последующее решение системы: можно ли разрешить доступ к ресурсам системы конкретному пользователю или процессу. После идентификации и аутентификации субъекта выполняется его авторизация.

Авторизация (Authorization) — процедура предоставления субъекту определенных полномочий и ресурсов в данной системе. Иными словами, авторизация устанавливает сферу его действия и доступные ему ресурсы. Если система не может надежно отличить авторизованное лицо от неавторизованного, то конфиденциальность и целостность информации в этой системе могут быть нарушены. Организации необходимо четко определить свои требования к безопасности, чтобы принимать решения о соответствующих границах авторизации.

* **Идентификация**— это процедура распознавания субъекта по его идентификатору (проще говоря, это определение имени, логина или номера).
* **Аутентификация** – это процедура проверки подлинности (пользователя проверяют с помощью пароля, письмо проверяют по электронной подписи и т.д.)

Чтобы определить чью-то подлинность, можно воспользоваться тремя факторами:

1. **Пароль** – то, что мы знаем (слово, PIN-код, код для замка, графический ключ)
2. **Устройство** – то, что мы имеем (пластиковая карта, ключ от замка, USB-ключ)
3. **Биометрика** – то, что является частью нас (отпечаток пальца, портрет, сетчатка глаза)

**Авторизация** – это предоставление доступа к какому-либо ресурсу (например, к электронной почте).  
  
Существует несколько различных схем аутентификации и авторизации. Рассмотрим более подробно некоторые из них.

**Аутентификация по многоразовому паролю**

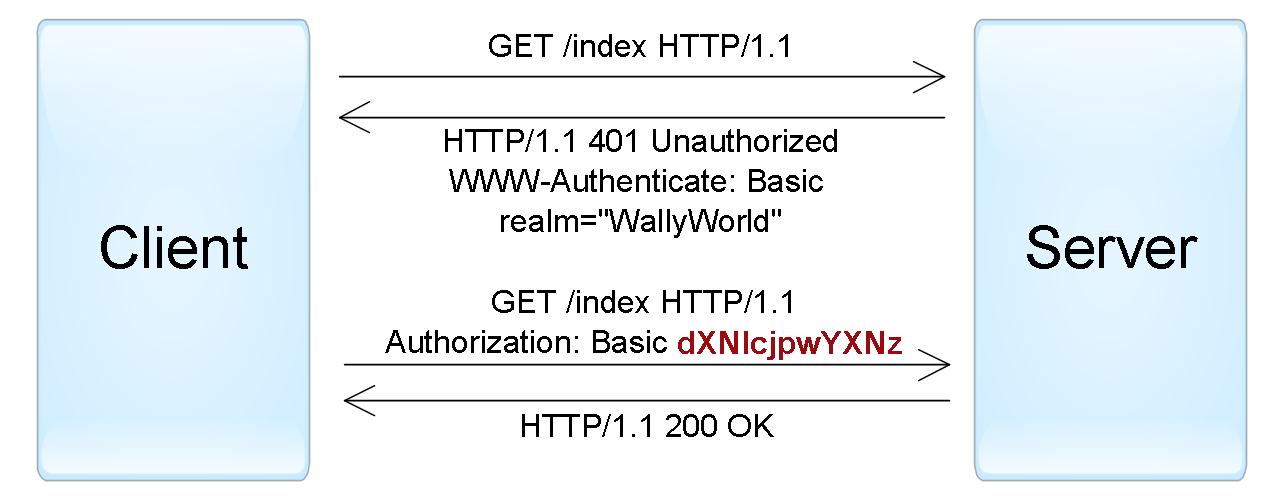
Аутентификация по многоразовому паролю считается самым популярным видом аутентификации. Хотя современные технологии внедряют все более сложные и надежные способы аутентификации, этот способ имеет место быть в несложных приложениях.

Этот способ основывается на том, что пользователь должен предоставить логин и пароль для успешной идентификации и аутентификации в системе. Пара логин/пароль задается пользователем при его регистрации в системе, при этом в качестве логина, как правило, выступает адрес электронной почты или номер телефона пользователя.

Применительно к веб-приложениям, существует несколько стандартных протоколов для аутентификации по паролю, которые мы рассмотрим ниже.

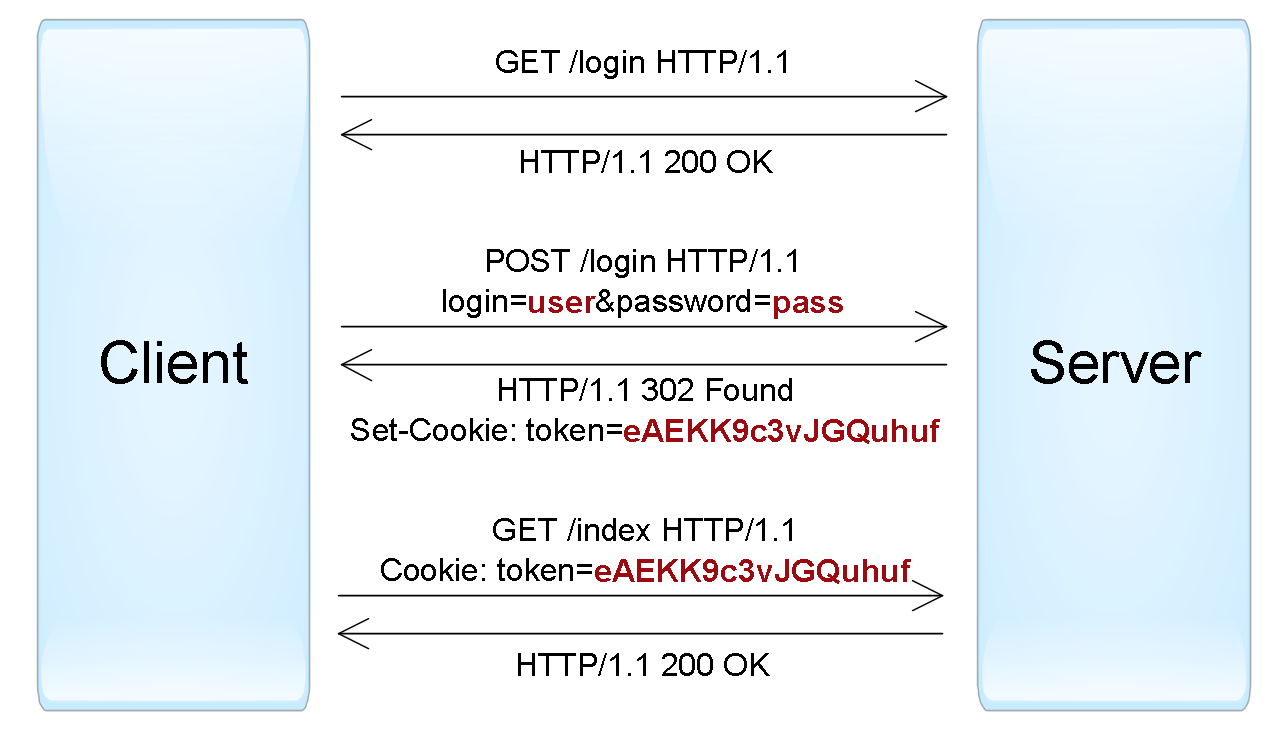
**HTTP-аутентификация**

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 логина и пароля. 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:
5. **Basic** — наиболее простая схема, при которой логин и пароль пользователя передаются в заголовке Authorization в незашифрованном виде (закодированном base64). Однако при использовании HTTPS (HTTP over SSL) протокола, является относительно безопасной.  
     
   *Пример HTTP аутентификации с использованием Basic схемы.*
6. **Digest** — схема аутентификации «вызов-ответ», при которой сервер посылает уникальную произвольную информацию (например, текущее время), а браузер конкатенирует эту информацию с паролем и хеширует полученную строку, затем передает этот хеш серверу, который сравнивает это значение с вычисленным у него на основе тех же данных. Такая схема считается более безопасной альтернативой схемы Basic при незащищенных соединениях, но она подвержена атаке «человек посередине» (при которой происходит замена схемы на basic). Кроме того, использование этой схемы не позволяет применить современные хеш-функции для хранения паролей пользователей на сервере.
7. **NTLM** (известная как Windows authentication) — также основана на подходе «вызов-ответ», 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 атаке повторного воспроизведения (pass-the-hash-attack).
8. **Negotiate** — еще одна схема из семейства Windows authentication, которая позволяет клиенту выбрать между NTLM и Kerberos аутентификацией. Kerberos — более безопасный протокол, основанный на технологии единого входа (Single Sign-On), при использовании которой пользователь переходит из одного раздела портала в другой без повторной аутентификации.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.

**Аутентификация с помощью форм (Forms authentication)**

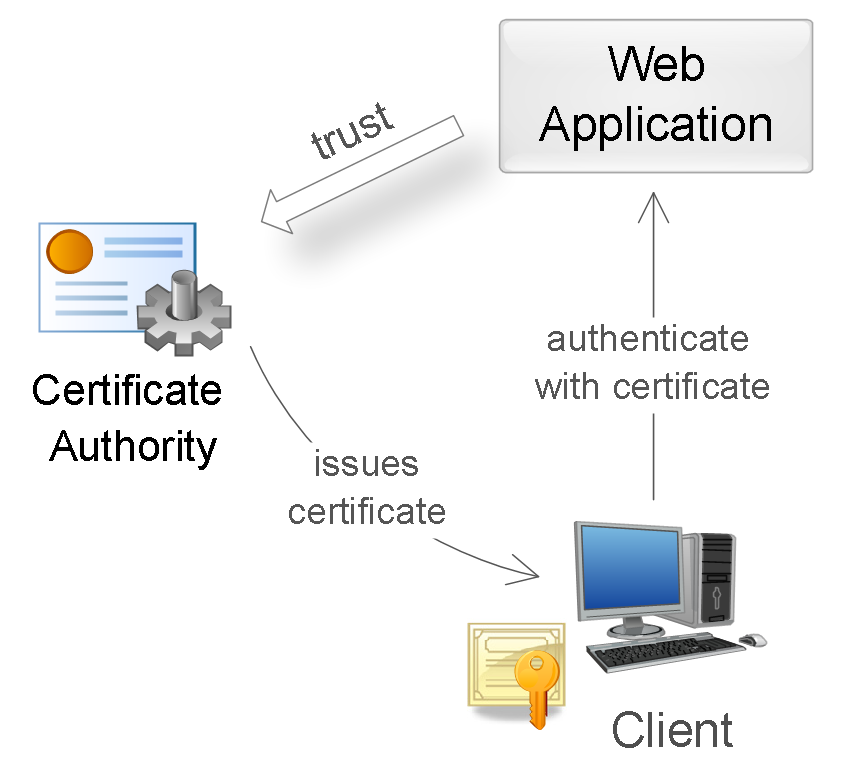
Для этого протокола нет определенного стандарта, поэтому все его реализации специфичны для конкретных систем, а точнее, для модулей аутентификации фреймворков разработки.  
  
Работает это по следующему принципу: в веб-приложение включается HTML-форма, в которую пользователь должен ввести свои логин/пароль и отправить их на сервер запросом HTTP POST для аутентификации. В случае успеха веб-приложение создает сессию, которая, как правило, сохраняется в cookies браузера. При последующих веб-запросах данные о текущей сессии автоматически передаются на сервер, что позволяет приложению получить информацию о текущем пользователе для авторизации запроса.  
  
  
  
Необходимо понимать, что перехват данных о текущей сессии зачастую дает аналогичный уровень доступа, что и знание пары логин/пароль. Поэтому все коммуникации между клиентом и сервером в случае аутентификации с помощью форм должны производиться только по защищенному соединению HTTPS.

**Другие протоколы аутентификации по многоразовому паролю**

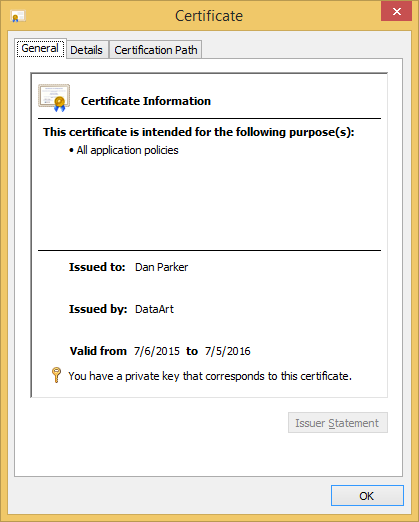
Протоколы, описанные выше, успешно используются для аутентификации пользователей на веб-сайтах. Но при разработке клиент-серверных приложений с использованием веб-сервисов (например, iOS или Android), наряду с HTTP аутентификацией, часто применяются нестандартные протоколы, в которых данные для аутентификации передаются в других частях запроса.  
  
Существует всего несколько мест, где можно передать пару логин/пароль в HTTP запросах:

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

##### Аутентификация по сертификатам

Сертификат представляет собой набор атрибутов, идентифицирующих владельца, подписанный центром сертификации (ЦС). ЦС выступает в роли посредника, который гарантирует подлинность сертификатов. Также сертификат криптографически связан с закрытым ключом, которых 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 (смарт-карта, USB-токен). 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 центром сертификации (проверка цепочки сертификатов).
2. Сертификат должен быть действительным на текущую дату (проверка срока действия).
3. Сертификат не должен быть отозван соответствующим ЦС (проверка списков исключения).

  
*Пример X.509 сертификата.*  
  
После успешной аутентификации веб-приложение может выполнить авторизацию запроса на основании таких данных сертификата, как имя владельца, эмитент, серийный номер сертификата или 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. Однако трудности с распространением и поддержкой сертификатов делает такой способ аутентификации малодоступным в широких кругах.

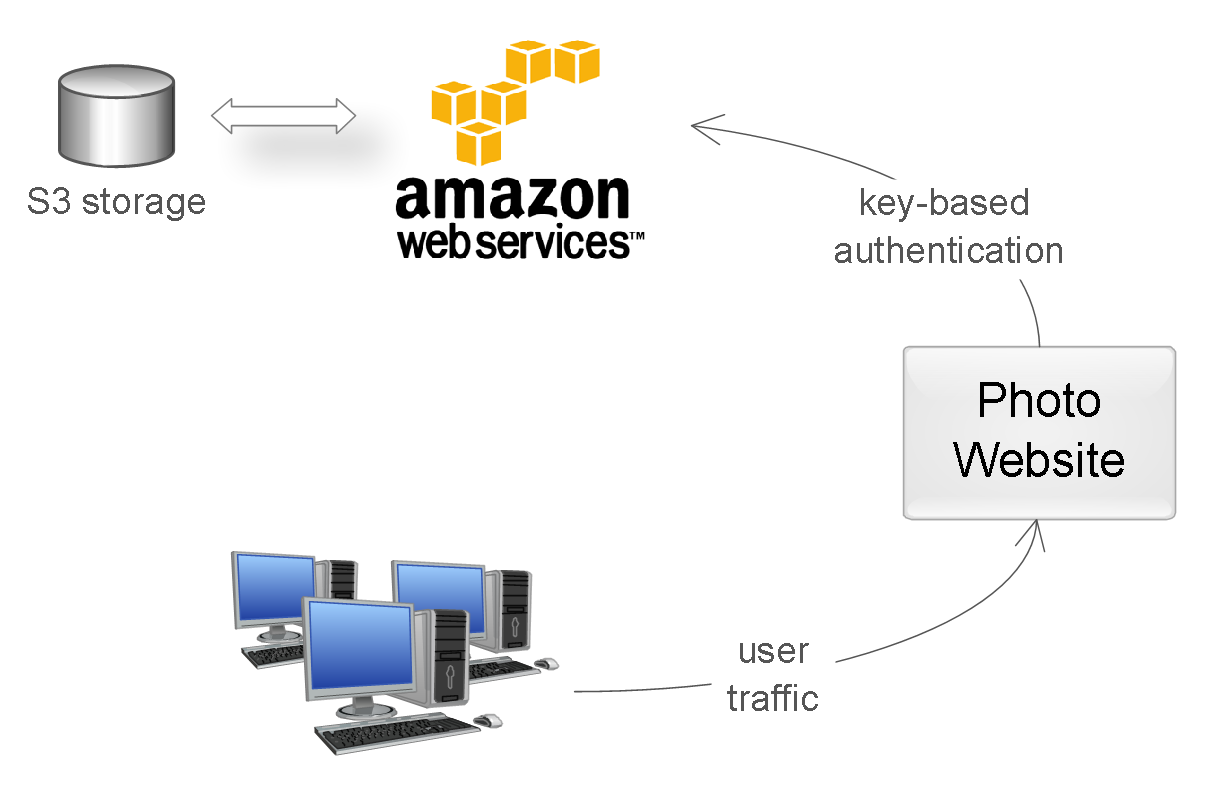
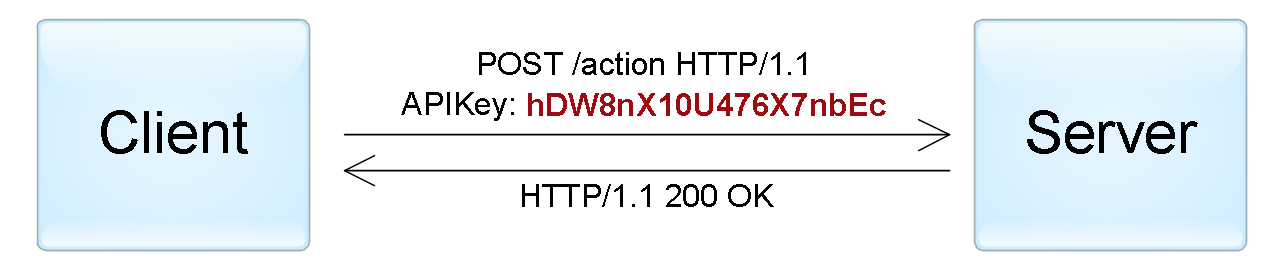
##### Аутентификация по одноразовым паролям

Аутентификация по одноразовым паролям обычно применяется дополнительно к аутентификации по паролям для реализации так называемой двухфакторной аутентификации (*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 может быть 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](http://www.emc.com/security/rsa-securid/index.htm); программной — приложение [Google Authenticator](https://play.google.com/store/apps/details?id=com.google.android.apps.authenticator2).
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 скретч-карта (карта из картона или пластика с нанесённой на ней (под защитным непрозрачным и стирающимся слоем) некой секретной информацией) со списком заранее сформированных одноразовых паролей. Для каждого нового входа в систему требуется ввести новый одноразовый пароль с указанным номером.

В веб-приложениях такой механизм аутентификации часто реализуется посредством расширения аутентификации с помощью форм: 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.

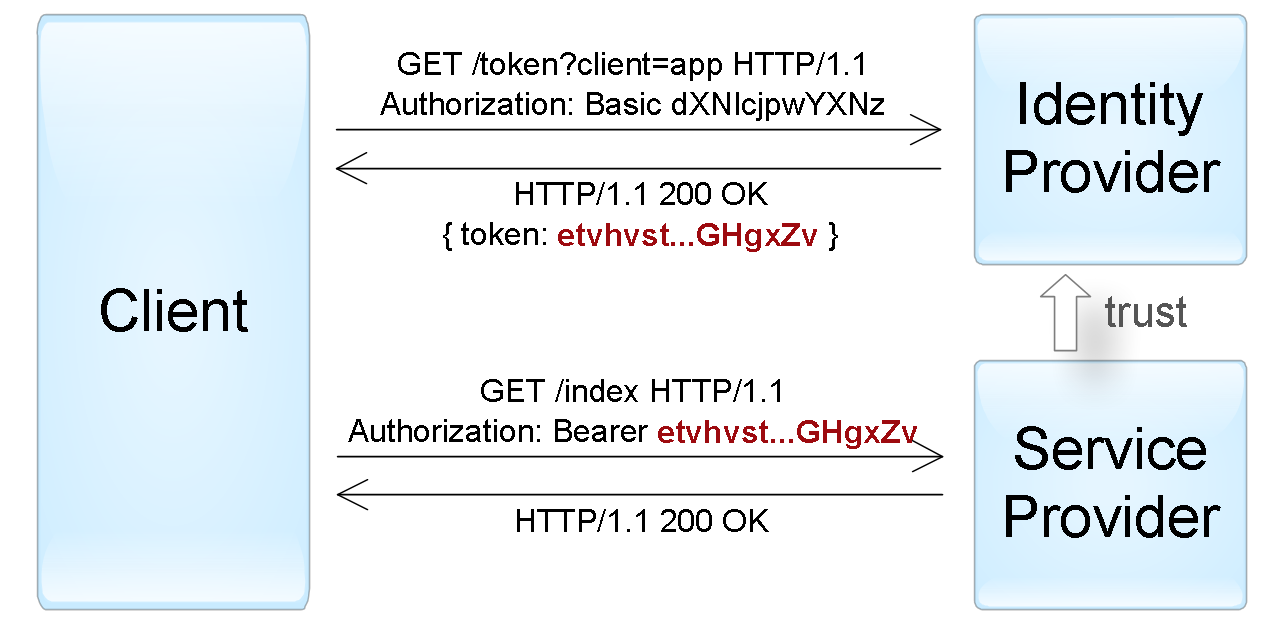
##### Аутентификация по ключам доступа

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 логин/пароль.  
  
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. 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-аутентификацией, сервер может послать клиенту некоторое уникальное значение или временную отметку, а клиент — возвратить хеш или HMAC этого значения, вычисленный с использованием секретной части ключа. Это позволяет избежать передачи всего ключа в оригинальном виде и защищает от атак повторного воспроизведения.

##### Аутентификация по токенам

Аутентификация по токенам чаще всего применяется при построении распределенных систем единого входа (SSO), где одно приложение делегирует функцию аутентификации пользователей другому приложению. 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 сервер аутентификации (identity provider - IP) предоставляет достоверные сведения о пользователе в виде токена, а приложение использует этот токен для идентификации, аутентификации и авторизации пользователя.  
На общем уровне, весь процесс выглядит следующим образом:

1. Клиент аутентифицируется на сервере аутентификации одним из способов, специфичным для него (пароль, ключ доступа, сертификат, Kerberos, итд.).
2. Клиент просит сервер аутентификации предоставить ему токен для конкретного риложения. Сервер аутентификации генерирует токен и отправляет его клиенту.
3. Клиент аутентифицируется в приложении при помощи этого токена.

  
  
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 приложение должно выполнить следующие проверки:

1. Токен был выдан доверенным сервером (проверка поля *issuer*).
2. Токен предназначается текущему 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 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

1. **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](http://auth.myservice.com/)», «aud»: «[myservice.com](http://myservice.com/)», «exp»: «1435937883», «userName»: «John Smith», «userRole»: «Admin» }.  
S9Zs/8/uEGGTVVtLggFTizCsMtwOJnRhjaQ2BMUQhcY

1. **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.

#### 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 |

**Глава 2**

В предыдущей главе были рассмотрены наиболее известные и применимые способы аутентификации. Каждый из них имеет свою специфику применения.

Одной из задач данной курсовой работы была реализация авторизации пользователей в небольшом онлайн-магазине. Для этих целей лучше всего подходит аутентификация по многоразовым или одноразовым паролям. Это привычные для пользователей способы аутентификации, которые, к тому же, при правильной реализации обеспечивают достаточную безопасность для стандартного функционала интернет-магазина.

Рассмотрим возможные уязвимости выбранных способов аутентификации, возможные ошибки разработчика и как их предотвратить.

**Распространенные п**риемы обхода парольной защиты и методы противодействия им.

*1. Полный перебор (метод грубой силы, bruteforce).*

Самая простая (с технической точки зрения) атака на пароль – перебор всех комбинаций допустимых символов (начиная от односимвольных паролей). Современные вычислительные мощности позволяют перебрать все пароли длиной до пяти-шести символов за несколько секунд.

Некоторые системы не позволяют реализовать атаки, основанные на переборе, поскольку реагируют на несколько попыток неправильно набранного пароля подряд.

Однако существует множество систем, позволяющих бесконечный перебор. Например, к защищенному паролем файлу (архив rar или zip, документ Microsoft Office и т.д.) можно пробовать разные пароли бесконечно. Существует множество программ, которые позволяют автоматизировать эту процедуру: Advanced RAR Password Recovery, Advanced PDF Password Recovery, Advanced Office XP Password Recovery.

И если получается похитить файл, содержащий хеши паролей доступа к операционной системе, то можно заниматься подбором паролей уже в обход системы, с помощью специальных программ.

Важной характеристикой пароля, затрудняющей полный перебор, является его длина. *Современный пароль должен иметь длину не менее 12 символов.*

Два лишних символа в пароле увеличивают время перебора в 40000 раз, а четыре символа — уже в 1.600.000.000 раз. Однако вычислительные мощности компьютеров постоянно растут (еще несколько лет назад безопасным считался пароль длиной 8 символов).

*2. Перебор в ограниченном диапазоне.*

Известно, что многие пользователи, составляя пароль, используют символы, находящиеся в определенном диапазоне. Например, пароль, состоящий только из русских букв или только из латинских букв или только из цифр. Такой пароль значительно легче запомнить, однако задача противника, осуществляющего перебор, неимоверно упрощается.

Пусть n = 70 — количество символов, из которых можно составить пароль, причем 10 из них — цифры, 30 — буквы одного языка и 30 — буквы другого языка. Пусть мы составляем пароль длиной m = 4 символа.

Если пароль составляется абсолютно случайно, то количество возможных комбинаций (которые необходимо перебрать) составляет 704 = 24010000. Однако противник может сделать предположение, что пароль состоит из символов одного диапазона (пусть даже, неизвестно, какого). Всего таких паролей 104 + 304 + 304 = 10000 + 810000 + 810000 = 163000. Если он оказался прав, то количество комбинаций (а следовательно, время, которое необходимо затратить на перебор) уменьшилось в 147 раз. Это число резко возрастает, когда увеличивается длина пароля и число диапазонов символов, из которых он может быть составлен.

Как следствие, *надежный пароль должен содержать в себе символы из различных диапазонов*. Рекомендуется использовать русские и английские, прописные и строчные буквы, цифры, а также прочие символы (знаки препинания, подчеркивание и т.д.).

*3. Атака по словарю*

В качестве пароля очень часто выбирается какое-то слово или распространенная фраза. Программа автоматического перебора паролей проверяет слова, содержащиеся в заданном файле со словарем (существует огромное количество доступных словарей такого рода для разных языков). Словарь из двухсот тысяч слов проверяется такой программой за несколько секунд.

Многие пользователи считают, что если применить к задуманному слову некоторое простое преобразование, например, написать его задом наперед или русскими буквами в английской раскладке или намеренно сделать ошибку, то это обеспечит безопасность. На самом деле, по сравнению с подбором случайного пароля подбор пароля по словарю с применением различных преобразований (сделать первую букву заглавной, сделать все буквы заглавными, объединить два слова и т.д.) делает невыполнимую задачу вполне возможной.

Надежный пароль не должен строиться на основе слов естественного языка.

*4. Атака по персональному словарю*

Если атака по словарю и перебор паролей небольшой длины либо составленных из символов одной группы не помогает, злоумышленник может воспользоваться тем фактом, что для облегчения запоминания, многие пользователи выбирают в качестве пароля личные данные (номер сотового телефона, дату рождения, записанную наоборот, кличку собаки и т.д.).

В том случае, если цель злоумышленника — обойти парольную защиту именно этого пользователя, он может составить для него персональный словарь личных данных, после чего использовать программу автоматического перебора паролей, которая будет генерировать пароли на основе этого словаря.

Надежный пароль должен быть полностью бессмысленным.

Существуют и другие атаки на парольную аутентификацию, такие как социальный инжиниринг или фишинг, но они напрямую связаны с грамотностью пользователя. Поскольку от технической реализации приложения в случае подобных атак ничего не зависит, то рассматривать их здесь не будем.

Резюмируем рекомендации разработчикам по предотвращению атак на парольную аутентификацию:

* ***Программное наложение технических ограничений*** (пароль должен быть не слишком коротким, он должен содержать буквы, цифры, знаки пунктуации и т.п.);
* ***управление сроком действия паролей***, их периодическая смена;
* ограничение доступа к *файлу паролей*;
* ограничение числа неудачных попыток входа в систему;
* обучение пользователей (хотя бы всплывающие подсказки с предупреждениями и рекомендациями);
* использование программных ***генераторов паролей*** (такая программа позволяет придумать «бессмысленный» пароль).

Также не стоит забывать о технических уязвимостях приложения, связанных с ошибками разработчиков. Рассмотрим рекомендации, о которых не стоит забывать при разработке приложения.

* Если веб-приложение само генерирует и распространяет пароли пользователям, то обязательно требуется смена пароля после первого входа (т.к. текущий пароль где-то записан).
* Не допускать передачу паролей по незащищенному HTTP-соединению либо в строке URL.
* Использовать только безопасные хеш-функции для хранения паролей пользователей.
* Предоставлять пользователям возможность изменения пароля и уведомлять их о произошедшей смене их пароля.
* Веб-приложение использует уязвимую функцию восстановления пароля, которую можно использовать для получения несанкционированного доступа к другим учетным записям.
* Требовать 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.
* Не допускать передачу session tokens по незащищенному HTTP-соединению, либо в строке URL.
* Уничтожать сессии пользователя после короткого периода неактивности и обязательно предоставлять функцию выхода из аутентифицированной сессии.
* Обязательно проверять все введенные пользователем данные на предмет соответствия этих данных ожиданию сервера во избежание SQL-инъекций.
* Защищать страницы от прямого доступа к ним через обращение по адресу в адресной строке.

### Уязвимости технологий OTP

Технология одноразовых паролей считается достаточно надежной. Однако объективности ради отметим, что и у нее есть недостатки, которым подвержены все системы, реализующие принцип OTP в чистом виде. Подобные уязвимости можно разделить на две группы. К первой относятся потенциально опасные "дыры", присущие всем методам реализации. Наиболее серьезная из них - возможность подмены сервера аутентификации. При этом пользователь будет отправлять свои данные прямо злоумышленнику, который может тут же использовать их для доступа к настоящему серверу. В случае метода "запрос-ответ" алгоритм атаки немного усложняется (компьютер хакера должен сыграть роль "посредника", пропуская через себя процесс обмена информацией между сервером и клиентом). Впрочем, стоит отметить, что на практике осуществить такую атаку совсем не просто.

Другая уязвимость присуща только синхронным методам и связана с тем, что существует риск рассинхронизации информации на сервере и в программном или аппаратном обеспечении пользователя. Допустим, в какой-то системе начальными данными служат показания внутренних таймеров, и по каким-то причинам они перестают совпадать друг с другом. В этом случае все попытки пользователей пройти аутентификацию будут неудачными (ошибка первого рода). К счастью, в подобных случаях ошибка второго рода (допуск "чужого") возникнуть не может. Впрочем, вероятность возникновения описанной ситуации также крайне мала.

Некоторые атаки применимы только к отдельным способам реализации технологии одноразовых паролей. Для примера опять возьмем метод синхронизации по таймеру. Как мы уже говорили, время в нем учитывается не с точностью до секунды, а в пределах какого-то установленного заранее интервала. Это делается с учетом возможности рассинхронизации таймеров, а также появления задержек в передаче данных. И именно этим моментом теоретически может воспользоваться злоумышленник для получения несанкционированного доступа к удаленной системе. Для начала хакер "прослушивает" сетевой трафик от пользователя к серверу аутентификации и перехватывает отправленные "жертвой" логин и одноразовый пароль. Затем он тут же блокирует его компьютер (перегружает его, обрывает связь и т. п.) и отправляет авторизационные данные уже от себя. И если злоумышленник успеет сделать это так быстро, чтобы интервал аутентификации не успел смениться, то сервер признает его как зарегистрированного пользователя.

Понятно, что для такой атаки злоумышленник должен иметь возможность прослушивания трафика, а также быстрого блокирования компьютера клиента, а это задача не из легких. Проще всего соблюсти эти условия тогда, когда атака задумывается заранее, причем для подключения к удаленной системе "жертва" будет использовать компьютер из чужой локальной сети. В этом случае хакер может заранее "поработать" над одним из ПК, получив возможность управлять им с другой машины. Защититься от такой атаки можно только путем использования "доверенных" рабочих машин (например, собственный ноутбук или КПК) и "независимых" защищенных (например, с помощью SSL) каналов выхода в Интернет.