РАЗРАБОТКА ВЕБ-САЙТОВ*, ИНФОРМАЦИОННАЯ БЕЗОПАСНОСТЬ*, БЛОГ КОМПАНИИ MAIL.RU GROUP

Как ты реализуешь аутентификацию, приятель?

ПЕРЕВОД

AloneCoder 28 ноября 2017 в 12:32 49,3k

Оригинал: Ahmed Shamim Hasan S…

Все знают о стандартной аутентификации пользователя в приложении. Это олдскульная процедура регистрации — пользователь вводит адрес почты, пароль и т. д., — а затем при входе мы сравниваем почту и/или пароль с сохранёнными данными. Если совпадает, даём доступ. Но времена изменились, и сегодня появилось много других методов аутентификации. Если хотите оставаться востребованным программистом/разработчиком в этом меняющемся, словно калейдоскоп, мире разработки ПО, то вы должны знать обо всех этих новых методах.

Нельзя отрицать, что в любых приложениях и ОС «аутентификация» — крайне важный элемент обеспечения сохранности пользовательских данных и регулирования доступа к информации. Чтобы понять, какой метод аутентификации для вас лучше, нужно разбираться в достоинствах и недостатках всех методов, а также неплохо представлять, как же они работают.

Здесь я постараюсь рассказать о большинстве распространённых сегодня методов аутентификации. Это не подробное техническое руководство, а лишь способ познакомить вас с ними. Хотя методы описаны с учётом применения в вебе, эти идеи можно реализовать и в других условиях.

Будем считать, вы уже знаете о том, что большая часть веба/интернета построена на протоколе HTTP. Также вам нужно знать, как работают веб-приложения, что означает аутентификация пользователя в приложении и что такое клиент-серверная архитектура.

Готовы? Поехали.

Аутентификация на основе сессий

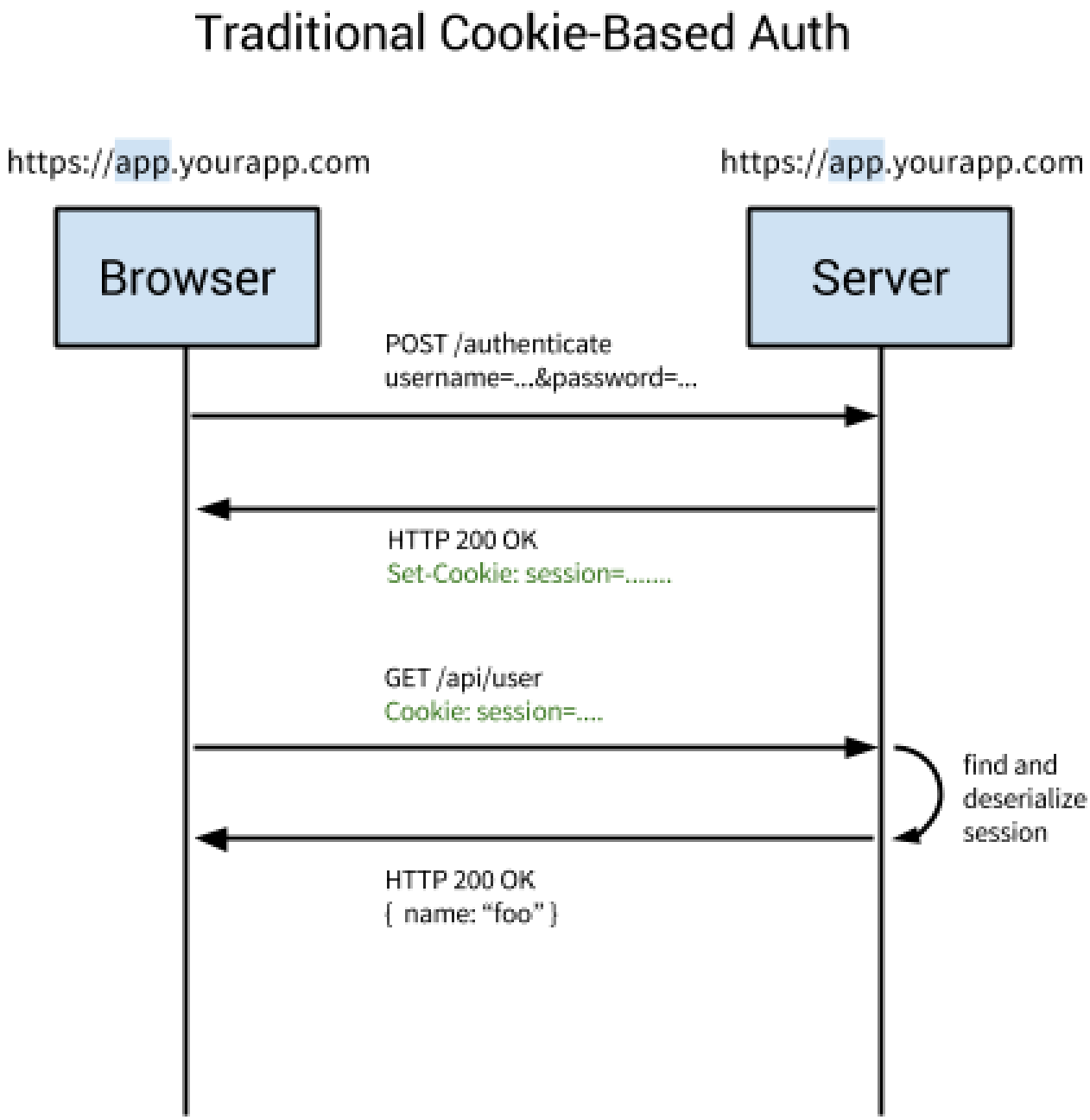
Протокол HTTP не отслеживает *состояния*, и, если мы аутентифицируем пользователя с помощью имени и пароля, наше приложение не будет знать, тот ли это человек, что и в предыдущем запросе. Нам придётся аутентифицировать снова. При каждом запросе HTTP не знает ничего о том, что происходило

до этого, он лишь передаёт запрос. Так что, если вам нужны личные данные, придётся снова логиниться, чтобы приложение знало, что это точно вы. Может сильно раздражать.

Чтобы избавиться от этого неудобства, придумали аутентификацию на основе сессий/кук, с помощью которых реализовали **отслеживание состояний** (stateful). Это означает, что аутентификационная запись или сессия должны храниться и на сервере, и на клиенте. Сервер должен отслеживать активные сессии в базе данных или памяти, а на фронтенде создаётся кука, в которой хранится идентификатор сессии. Это аутентификация на основе куки, самая распространённый и широко известный метод, используемый уже давно.

Процедура аутентификации на основе сессий:

1. Пользователь вводит в браузере своё имя и пароль, после чего клиентское приложение отправляет на сервер запрос.
2. Сервер проверяет пользователя, аутентифицирует его, шлёт приложению уникальный пользовательский токен (сохранив его в памяти или базе данных).
3. Клиентское приложение сохраняет токены в куках и отправляет их при каждом последующем запросе.
4. Сервер получает каждый запрос, требующий аутентификации, с помощью токена аутентифицирует пользователя и возвращает запрошенные данные клиентскому приложению.
5. Когда пользователь выходит, клиентское приложение удаляет его токен, поэтому все последующие запросы от этого клиента становятся неаутентифицированными.



У этого метода несколько недостатков.

- При каждой аутентификации пользователя сервер должен создавать у себя запись. Обычно она хранится в памяти, и при

большом количестве пользователей есть вероятность слишком высокой нагрузки на сервер.
- Поскольку сессии хранятся в памяти, масштабировать не так просто. Если вы многократно реплицируете сервер, то на все новые серверы придётся реплицировать и все пользовательские сессии. Это усложняет масштабирование. (Я считал, этого можно избежать, если иметь выделенный сервер для управления сессиями, но это сложно реализовать, да и не всегда возможно.)

Аутентификация на основе токенов

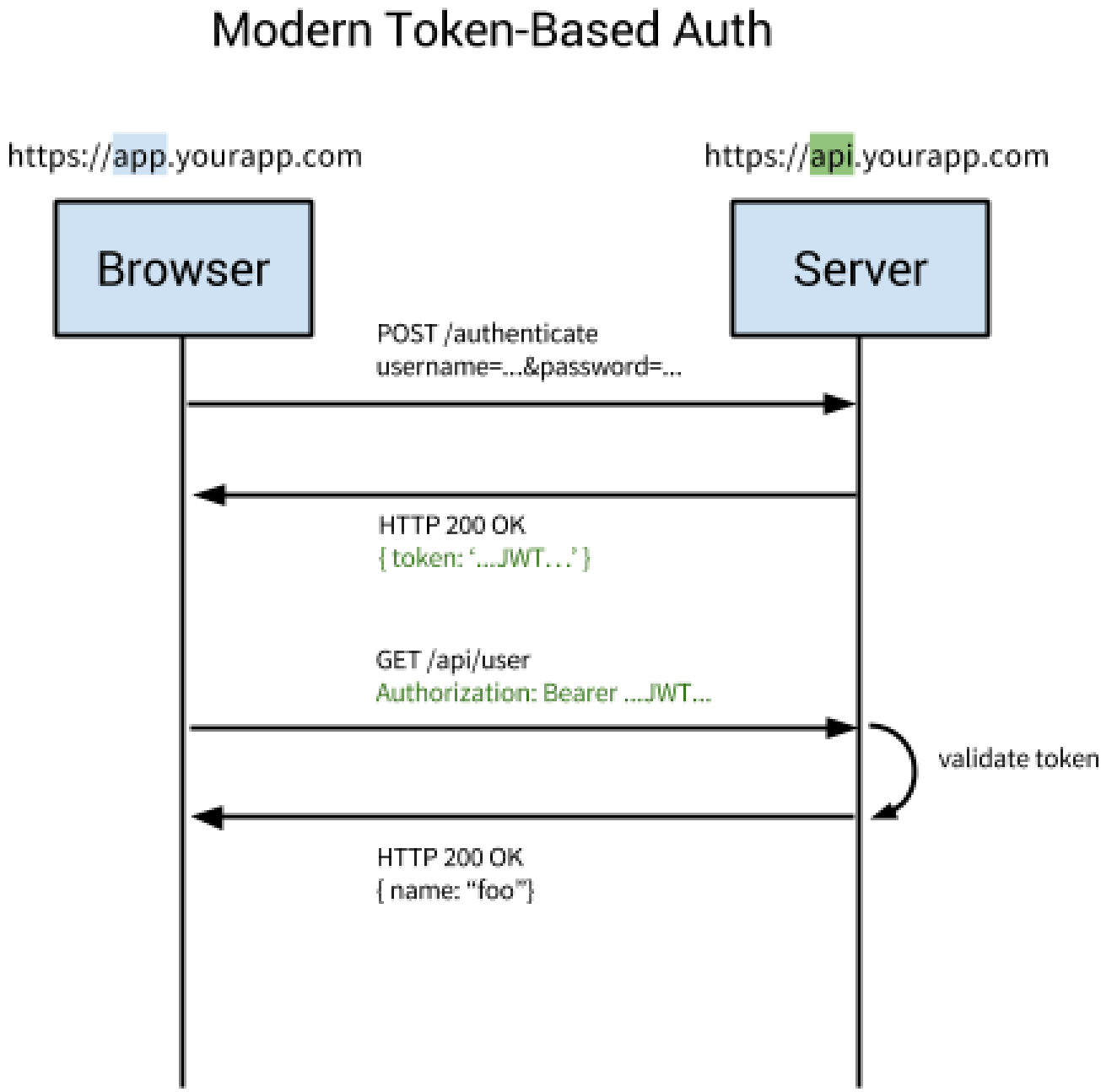
Аутентификация на основе токенов в последние годы стала очень популярна из-за распространения одностраничных приложений, веб-API и интернета вещей. Чаще всего в качестве токенов используются Json Web Tokens (JWT). Хотя реализации бывают разные, но токены JWT превратились в стандарт де-факто.

При аутентификации на основе токенов **состояния не отслеживаются**. Мы не будем хранить информацию о пользователе на сервере или в сессии и даже не будем хранить JWT, использованные для клиентов.

Процедура аутентификации на основе токенов:

1. Пользователь вводит имя и пароль.
2. Сервер проверяет их и возвращает токен (JWT), который может содержать метаданные вроде user_id, разрешений и т. д.

3. Токен хранится на клиентской стороне, чаще всего в локальном хранилище, но может лежать и в хранилище сессий или кук.
4. Последующие запросы к серверу обычно содержат этот токен в качестве дополнительного заголовка авторизации в виде *Bearer {JWT}*. Ещё токен может пересылаться в теле POST-запроса и даже как параметр запроса.
5. Сервер расшифровывает JWT, если токен верный, сервер обрабатывает запрос.
6. Когда пользователь выходит из системы, токен на клиентской стороне уничтожается, с сервером взаимодействовать не нужно.



Более подробное описание.

У метода есть ряд преимуществ:

- Главное преимущество: поскольку метод никак не оперирует состояниями, серверу не нужно хранить записи с пользовательскими токенами или сессиями. Каждый токен самодостаточен, содержит все необходимые для проверки данные, а также передаёт затребованную пользовательскую информацию. Поэтому токены не усложняют масштабирование.
- В куках вы просто храните ID пользовательских сессий, а JWT позволяет хранить метаданные любого типа, если это корректный JSON.
- При использовании кук бэкенд должен выполнять поиск по традиционной SQL-базе или NoSQL-альтернативе, и обмен данными наверняка длится дольше, чем расшифровка токена. Кроме того, раз вы можете хранить внутри JWT дополнительные данные вроде пользовательских разрешений, то можете сэкономить и дополнительные обращения поисковые запросы на получение и обработку данных.
  Допустим, у вас есть API-ресурс /api/orders, который возвращает последние созданные приложением заказы, но просматривать их могут только пользователи категории админов. Если вы используете куки, то, сделав запрос, вы генерируете одно обращение к базе данных для проверки сессии, ещё одно обращение — для получения пользовательских данных и проверки, относится ли пользователь к админам, и третье обращение — для получения данных.
  А если вы применяете JWT, то можете хранить пользовательскую категорию уже в токене. Когда сервер запросит его и расшифрует, вы можете сделать одно обращение к базе данных, чтобы получить нужные заказы.

- У использования кук на мобильных платформах есть много ограничений и особенностей. А токены сильно проще реализовать на iOS и Android. К тому же токены проще реализовать для приложений и сервисов интернета вещей, в которых не предусмотрено хранение кук.

Благодаря всему этому аутентификация на основе токенов сегодня набирает популярность.

Беспарольная аутентификация

Первой реакцией на термин «беспарольная аутентификация» может быть «Как аутентифицировать кого-то без пароля? Разве такое возможно?»

В наши головы внедрено убеждение, что пароли — абсолютный источник защиты наших аккаунтов. Но если изучить вопрос глубже, то выяснится, что беспарольная аутентификация может быть не просто безопасной, но и безопаснее традиционного входа по имени и паролю. Возможно, вы даже слышали мнение, что пароли устарели.

Беспарольная аутентификация — это способ конфигурирования процедуры входа и аутентификации пользователей без ввода паролей. Идея такая:

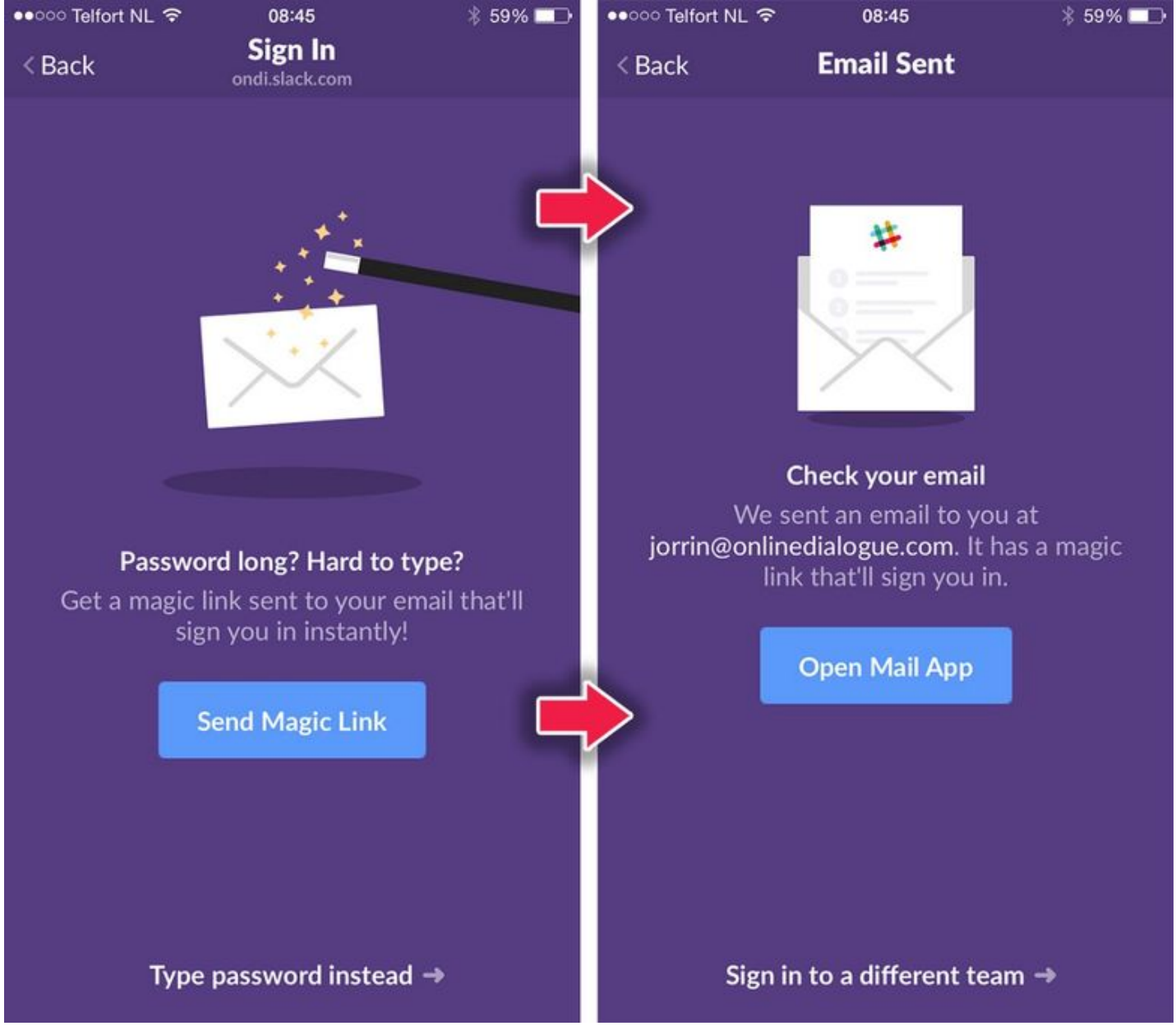
Вместо ввода почты/имени и пароля пользователи вводят только свою почту. Ваше приложение отправляет на этот адрес одноразовую ссылку, пользователь по ней кликает и

автоматически входит на ваш сайт / в приложение. При беспарольной аутентификации приложение считает, что в ваш ящик пришло письмо со ссылкой, если вы написали свой, а не чужой адрес.

Есть похожий метод, при котором вместо одноразовой ссылки по SMS отправляется код или одноразовый пароль. Но тогда придётся объединить ваше приложение с SMS-сервисом вроде twilio (и сервис не бесплатен). Код или одноразовый пароль тоже можно отправлять по почте.

И ещё один, менее (пока) популярный (и доступный только на устройствах Apple) метод беспарольной аутентификации: использовать Touch ID для аутентификации по отпечаткам пальцев. Подробнее о технологии.

Если вы пользуетесь Slack, то уже могли столкнуться с беспарольной аутентификацией.



Medium предоставляет доступ к своему сайту только по почте. Я недавно обнаружил, что Auth0, или Facebook AccountKit, — это отличный вариант для реализации беспарольной системы для вашего приложения.

Что может пойти не так?

Если кто-то получит доступ к пользовательским почтам, он получит и доступ к приложениям и сайтам. Но это не ваша головная боль — беспокоиться о безопасности почтовых аккаунтов

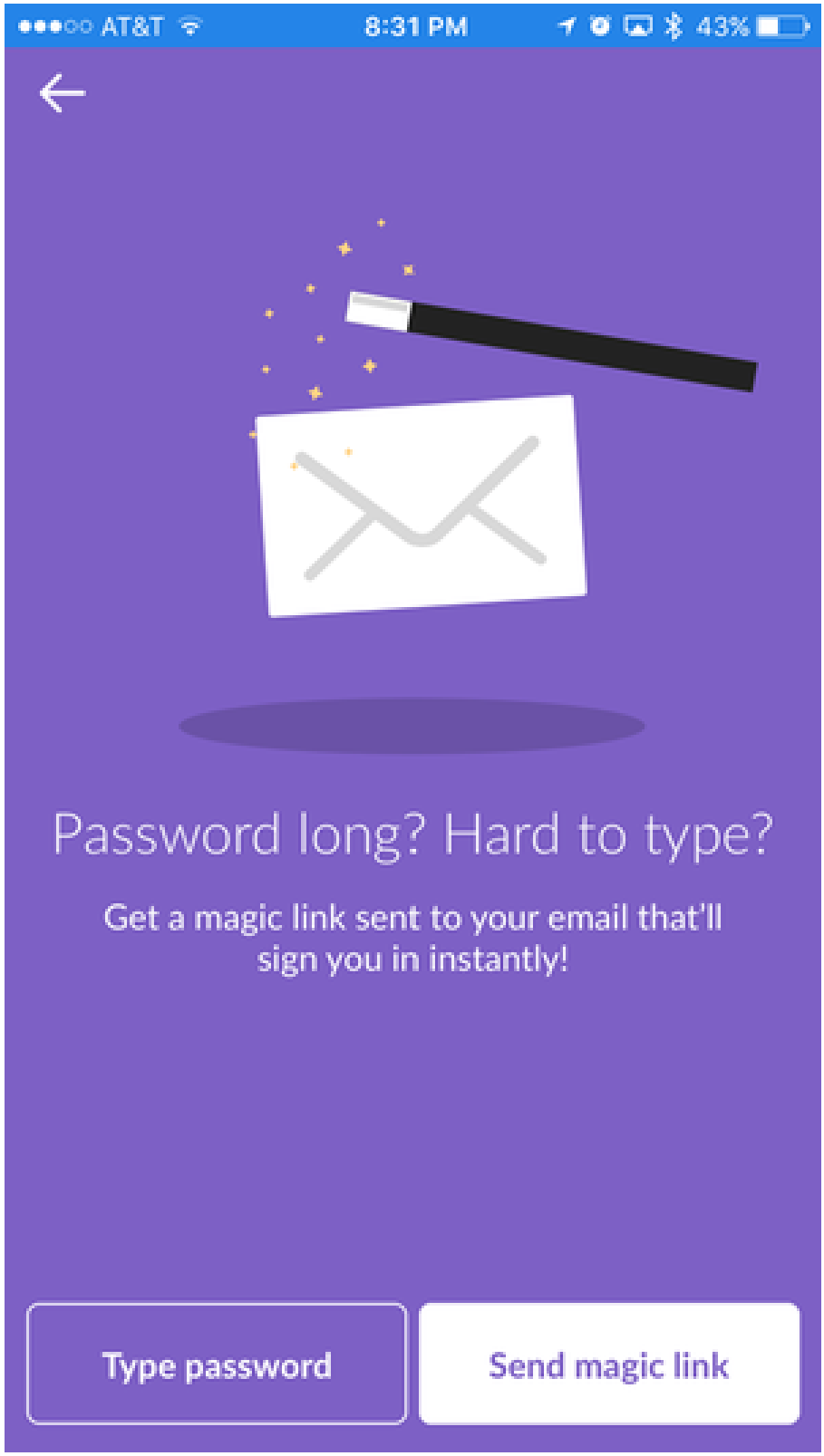
пользователей. Кроме того, если кто-то получит доступ к чужой почте, то сможет перехватить аккаунты в приложениях с беспарольной аутентификацией, воспользовавшись функцией восстановления пароля. Но мы ничего не можем поделать с почтой наших пользователей. Пойдём дальше.

В чём преимущества?

Как часто вы пользуетесь ссылкой «забыли пароль» для сброса чёртового пароля, который так и не смогли вспомнить после нескольких неудачных попыток входа на сайт / в приложение? Все мы бываем в такой ситуации. Все пароли не упомнишь, особенно если вы заботитесь о безопасности и для каждого сайта делаете отдельный пароль (соблюдая все эти «должен состоять не менее чем из восьми символов, содержать хотя бы одну цифру, строчную букву и специальный символ»). От всего этого вас избавит беспарольная аутентификация. Знаю, вы думаете сейчас: «Я использую менеджер паролей, идиот». Уважаю. Но не забывайте, что подавляющее большинство пользователей не такие техногики, как вы. Это нужно учитывать.

Беспарольная аутентификация хороша не только для пользователей, но и для вас как разработчика. Вам не нужно реализовывать механизм восстановления паролей. Все в выигрыше.

Если вы думаете, что какие-то пользователи предпочтут старомодные логин/пароль, то предоставьте им оба варианта, чтобы они могли выбирать.



Сегодня беспарольная аутентификация быстро набирает популярность.

Единая точка входа (Single Sign On, SSO)

Обращали внимание, что, когда логинишься в браузере в каком-нибудь Google-сервисе, например Gmail, а потом идёшь на Youtube или иной Google-сервис, там не приходится логиниться? Ты автомагически получаешь доступ ко всем сервисам компании. Впечатляет, верно? Ведь хотя Gmail и Youtube — это сервисы Google, но всё же раздельные продукты. Как они аутентифицируют пользователя во всех продуктах после единственного входа?

Этот метод называется единой точкой входа (Single Sign On, SSO).

Реализовать его можно по-разному. Например, использовать центральный сервис для оркестрации единого входа между несколькими клиентами. В случае с Google этот сервис называется Google Accounts. Когда пользователь логинится, Google Accounts создаёт куку, которая сохраняется за пользователем, когда тот ходит по принадлежащим компании сервисам. Как это работает:

1. Пользователь входит в один из сервисов Google.
2. Пользователь получает сгенерированную в Google Accounts куку.
3. Пользователь идёт в другой продукт Google.
4. Пользователь снова перенаправляется в Google Accounts.
5. Google Accounts видит, что пользователю уже присвоена кука, и перенаправляет пользователя в запрошенный продукт.

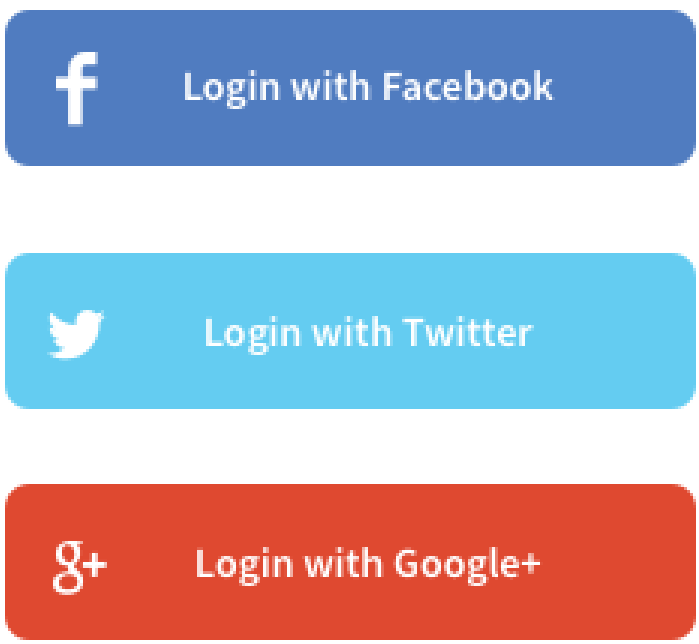
Очень простое описание единой точки входа: пользователь входит один раз и получает доступ ко всем системам без необходимости входить в каждую из них. В этой процедуре используется три

сущности, доверяющие другу прямо и косвенно. **Пользователь** вводит пароль (или аутентифицируется иначе) у **поставщика идентификационной информации** (identity provider, IDP), чтобы получить доступ к **поставщику услуги** (service provider (SP). Пользователь доверяет IDP, и SP доверяет IDP, так что SP может доверять пользователю.

Выглядит очень просто, но конкретные реализации бывают очень сложными. Подробнее об этом методе аутентификации.

Аутентификация в соцсетях

Уверен, эта картинка знакома всем:



Это часто называют **аутентификацией в соцсетях** (Social sign-in) или **социальным логином** (Social Login). Вы можете аутентифицировать пользователей по их аккаунтам в соцсетях. Тогда пользователям не придётся регистрироваться отдельно в вашем приложении.

Формально социальный логин — это не отдельный метод аутентификации. Это разновидность единой точки входа с упрощением процесса регистрации/входа пользователя в ваше приложение.

Лучшее из двух миров

Пользователи могут войти в ваше приложение одним кликом, если у них есть аккаунт в одной из соцсетей. Им не нужно помнить логины и пароли. Это сильно улучшает опыт использования вашего приложения. Вам как разработчику не нужно волноваться о безопасности пользовательских данных и думать о проверке адресов почты — они уже проверены соцсетями. Кроме того, в соцсетях уже есть механизмы восстановления пароля.

Как использовать

Как разработчик вы должны разбираться в работе этого метода аутентификации. Большинство соцсетей в качестве механизма аутентификации используют авторизацию через OAuth2 (некоторые используют OAuth1, например Twitter). Разберёмся, что такое OAuth. Соцсеть — это **сервер ресурсов**, ваше приложение — **клиент**, а пытающийся войти в ваше приложение пользователь

— **владелец ресурса**. Ресурсом называется пользовательский профиль / информация для аутентификации. Когда пользователь хочет войти в ваше приложение, оно перенаправляет пользователя в соцсеть для аутентификации (обычно это всплывающее окно с URL'ом соцсети). После успешной аутентификации пользователь должен дать вашему приложению разрешение на доступ к своему профилю из соцсети. Затем соцсеть возвращает пользователя обратно в ваше приложение, но уже с токеном доступа. В следующий раз приложение возьмёт этот токен и запросит у соцсети информацию из пользовательского профиля. Так работает OAuth (ради простоты я опустил технические подробности).

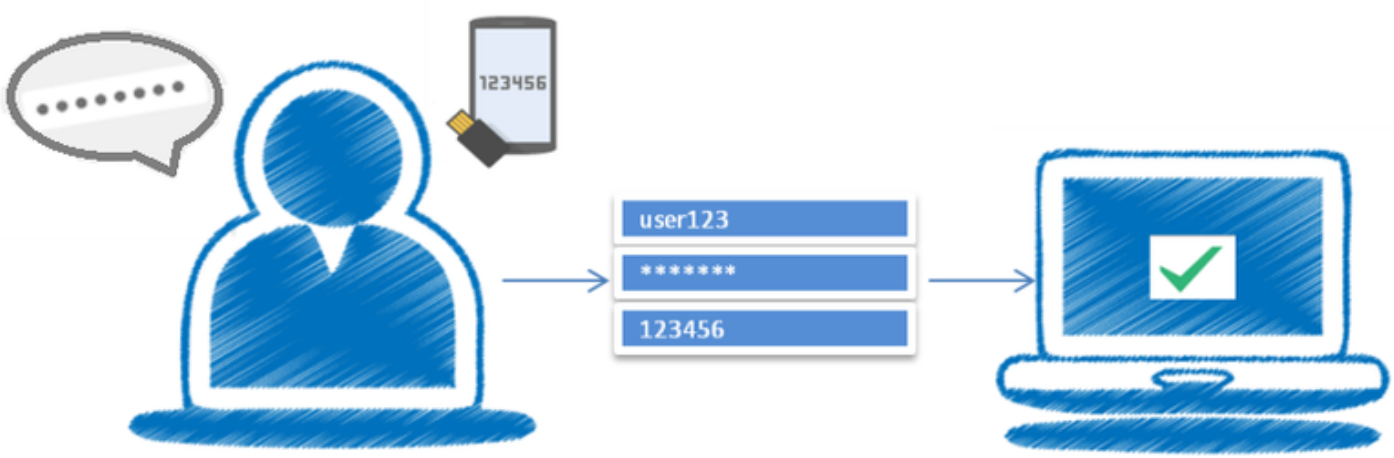
Для реализации такого механизма вам может понадобиться зарегистрировать своё приложение в разных соцсетях. Вам дадут app_id и другие ключи для конфигурирования подключения к соцсетям. Также есть несколько популярных библиотек/пакетов (вроде Passport, Laravel Socialite и т. д.), которые помогут упростить процедуру и избавят от излишней возни.

Двухфакторная аутентификация (2FA)

Двухфакторная аутентификация (2FA) улучшает безопасность доступа за счёт использования двух методов (также называемых факторами) проверки личности пользователя. Это разновидность многофакторной аутентификации. Наверное, вам не приходило в голову, но в банкоматах вы проходите двухфакторную аутентификацию: на вашей банковской карте должна быть записана правильная информация, и в дополнение к этому вы

вводите PIN. Если кто-то украдёт вашу карту, то без кода он не сможет ею воспользоваться. *(Не факт! — Примеч. пер.)* То есть в системе двухфакторной аутентификации пользователь получает доступ только после того, как предоставит несколько отдельных частей информации.

Другой знакомый пример — двухфакторная аутентификация Mail.Ru, Google, Facebook и т. д. Если включён этот метод входа, то сначала вам нужно ввести логин и пароль, а затем одноразовый пароль (код проверки), отправляемый по SMS. Если ваш обычный пароль был скомпрометирован, аккаунт останется защищённым, потому что на втором шаге входа злоумышленник не сможет ввести нужный код проверки.



Вместо одноразового пароля в качестве второго фактора могут использоваться отпечатки пальцев или снимок сетчатки.

При двухфакторной аутентификации пользователь должен предоставить два из трёх:

- **То, что вы знаете:** пароль или PIN.
- **То, что у вас есть:** физическое устройство (смартфон) или приложение, генерирующее одноразовые пароли.
- **Часть вас:** биологически уникальное свойство вроде ваших отпечатков пальцев, голоса или снимка сетчатки.

Большинство хакеров охотятся за паролями и PIN-кодами. Гораздо труднее получить доступ к генератору токенов или биологическим свойствам, поэтому сегодня двухфакторка обеспечивает высокую безопасность аккаунтов.

То есть это универсальное решение? Возможно, нет.

И всё же двухфакторка поможет усилить безопасность аутентификации в вашем приложении. Как реализовать? Возможно, стоит не велосипедить, а воспользоваться существующими решениями вроде Auth0 или Duo.

Аутентификация или авторизация?

Некоторые путают термины «аутентификация» и «авторизация». Это разные вещи.

- **Аутентификация** — это проверка вашей личности. Когда вы входите в приложение с именем и паролем, вы аутентифицируетесь.
- **Авторизация** — это проверка наличия у вас доступа к чему-либо. Это может быть набор разрешений на какие-то действия. Например, если вы создали в приложении ресурс, то вы можете

быть единственным, кому разрешено удалять этот ресурс (потому что вы владелец), а другие пользователи для того не «авторизованы».

Ещё тут?

Поздравляю, вы успешно дочитали длинную, нудную и скучную статью.

Комментарии (84)

Похожие публикации

21 совет по эффективному использованию Composer

7

ПЕРЕВОД

AloneCoder • 12 января в 15:49

Линейная регрессия с помощью Go

22

ПЕРЕВОД

AloneCoder • 25 декабря 2017 в 19:09

REST — это новый SOAP

154

ПЕРЕВОД

AloneCoder • 21 декабря 2017 в 15:07

Популярное за сутки

Яндекс открывает Алису для всех разработчиков. Платформа Яндекс.Диалоги (бета)

69

BarakAdama • вчера в 10:52

Почему следует игнорировать истории основателей успешных стартапов 20

ПЕРЕВОД

m1rko • вчера в 10:44

Как получить телефон (почти) любой красотки в Москве, или интересная особенность MT_FREE 24

ИЗ ПЕСОЧНИЦЫ

cab404 • вчера в 20:27

Java и Project Reactor 10

zealot_and_frenzy • вчера в 10:56

Пользовательские агрегатные и оконные функции в PostgreSQL и Oracle 6

erogov • вчера в 12:46

Лучшее на Geektimes

Как фермеры Дикого Запада организовали телефонную сеть на колючей проволоке 31

NAGru • вчера в 10:10

Энтузиаст сделал новую материнскую плату для ThinkPad X200s 49

alizar • вчера в 15:32

Кто-то посылает секс-игрушки с Amazon незнакомцам. Amazon не знает, как их остановить

85

Pochtoycom • вчера в 13:06

Илон Маск продолжает убеждать в необходимости создания колонии людей на Марсе

140

marks • вчера в 14:19

Дела шпионские (часть 1)

16

TashaFridrih • вчера в 13:16