

Авторизация с помощью OAuth 2.0 на Go



Tproger

всё о программировании



Разбор OAuth 2 в доступной форме с примером реализации на Go,

19K открытий 23K показов



Вячеслав Аврамёнок

младший разработчик компании Selectel

С появлением большого числа веб-сервисов, возникла необходимость в предоставлении ресурсов одного сервиса другому. Самое первое реализованное решение заключалось в предоставлении пользователем логина и пароля сервису А, чтобы тот мог получить данные от сервиса В. Однако такое решение имеет следующий ряд проблем и ограничений:

- Сервису А необходимо хранить логин и пароль пользователя у себя, более того, пароль хранится в незашифрованном виде.
- Нельзя выбрать подмножество ресурсов, а также ограничить время их доступности.
- Пользователь не может выборочно запретить сервису доступ к его ресурсам. При изменении логина или пароля все сервисы потеряют доступ.
- Взлом одного из сервисов приведет к утечке логина и пароля пользователя, и, следовательно, доступу к приватным данным другого

сервиса.

Таким образом, для решения обозначенных выше проблем, была предложена спецификация OAuth, реализовав которую, становится возможным безопасное получение сторонним сервисом ограниченного доступа к ресурсам другого.

Примечание Статья носит сугубо информативный характер, и не преследует цели максимально подробно изложить материал по теме — для этого есть документация. Следует упомянуть что статья появилась на свет по причине изучения данного вопроса самим автором. Если вы не в курсе что такое OAuth, то ниже вашему вниманию представлен материал, который я попытался обобщить и изложить в доступной форме.

Существует две версии спецификации: [OAuth 1.0](#) и [OAuth 2.0](#), которые, к слову, обратно несовместимы. Первая версия была опубликована в 2007 году, и представляет собой документ с рекомендациями по делегированию ресурсов стороннему сервису без раскрытия логина и пароля пользователем, в то время как вторая версия (2012 год) уже является интернет стандартом, учитывает недостатки первой версии, и расширяет возможные сценарии использования (выборочный доступ к ресурсам, 4 вида авторизации, использование нативных приложений, refresh token и т.д.). Когда говорят об OAuth, то, как правило, имеется в виду именно вторая версия, т.к. она чаще всего используется, и далее поговорим о ней подробнее.

Действующие лица

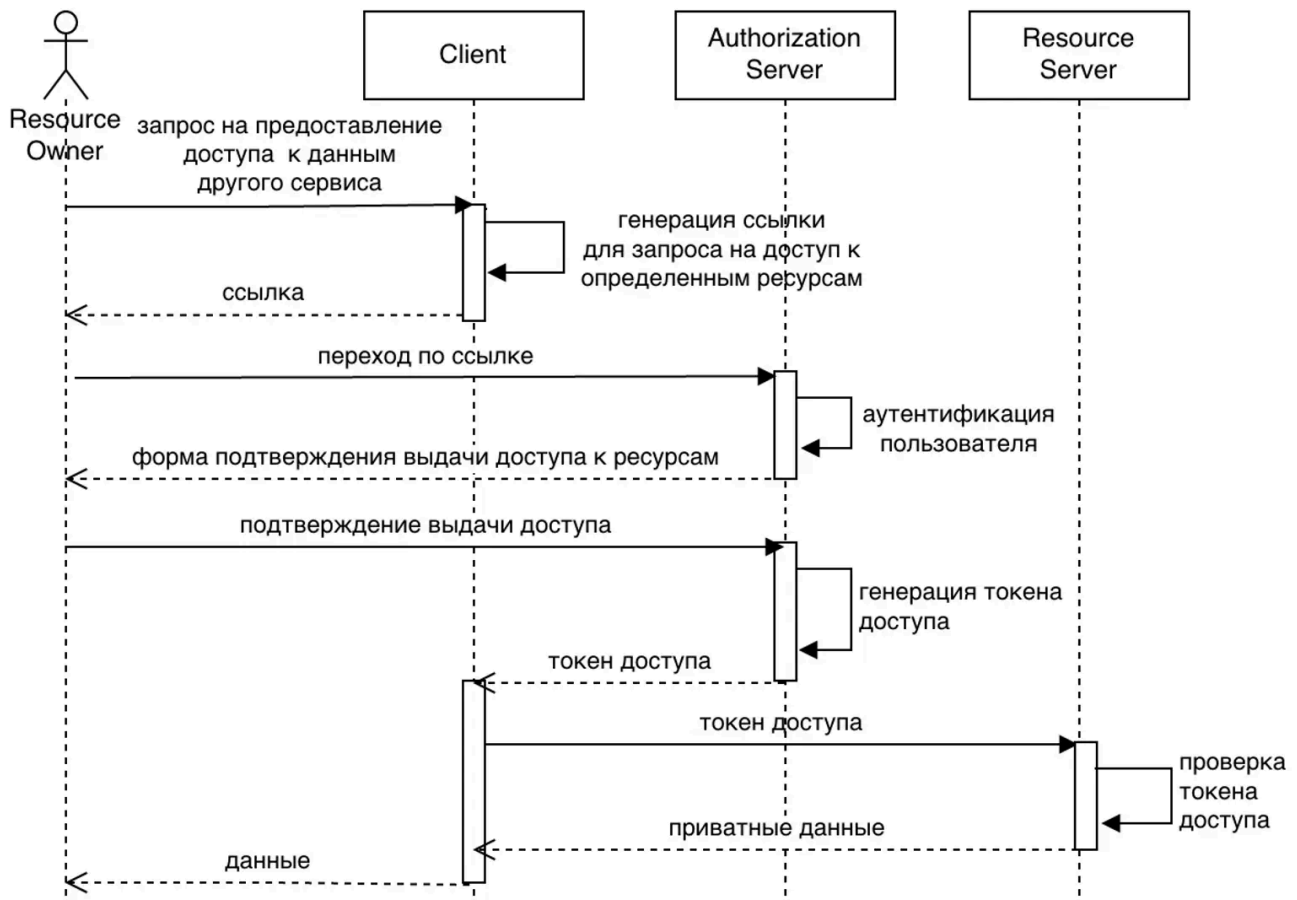
В спецификации OAuth 2.0 выделяют следующие роли:

- Client. Сервис (приложение), желающий получить доступ к пользовательским данным.
- Resource Server. Сервис, владеющий пользовательскими данными.
- Authorization Server. Сервис, предоставляющий пользователю интерфейс выбора дать или нет доступ к ресурсам стороннему сервису.
- Resource Owner. Пользователь, дающий доступ к ресурсам.

Рассмотрим работу OAuth с точки зрения Client как с наиболее частым кейсом использования разработчиками, а подробности реализации

Authorization Server — это уже документация.

Общий флоу работы с OAuth



- Пользователь делает запрос на предоставление Client доступа к приватным данным другого сервиса.
- Client генерирует и отправляет пользователю ссылку, которая перенаправит его на Authorization Server стороннего сервиса.
- Пользователь аутентифицируется на Authorization Server введя логин и пароль, либо ничего не вводя, т.к. в браузере уже будет активная сессия.
- Authorization Server покажет пользователю форму в которой спросит разрешается ли предоставить доступ к перечисленным ресурсам стороннему сервису.
- Если пользователь соглашается, то Authorization Server делает редирект пользователя вместе с токеном авторизации (или кодом авторизации) на эндпоинт Client (см. Регистрация приложения).
- Client читает предоставленный токен доступа (или читает код авторизации и меняет его на токен доступа через специальный эндпоинт на

Authorization Server), затем обращается с этим токеном к Resource Server.

- Resource Server проверяет токен доступа и отправляет приватные данные пользователя.
- Client получает данные, пользователь теперь может с ними взаимодействовать.

Вообще, при получении токена доступа также может приходиться refresh token (т.е. это на усмотрение Authorization Server), задача которого — получение новых токенов доступа, если старые стали невалидными. Стоит отметить, что все сетевые операции должны выполняться с использованием HTTPS.

Как обеспечить безопасность токенов аутентификации

tproger.ru

Регистрация приложения

Перво-наперво необходимо зарегистрировать Client на Authorization Server. Во время регистрации необходимо заполнить такую информацию как имя, веб-сайт, лого приложения и т.д. Также обязательным элементом регистрации является указание ссылки URI куда Authorization Server будет делать редирект пользователя после апрува на выдачу доступа к его ресурсам, расположенным на Resource Server. После регистрации будут выданы Client ID и Client Secret, которые используются в процессе получения авторизации.

Виды авторизации пользователя

Authorization Code

Приложение (Client) делает редирект пользователя на Authorization Server, где пользователь аутентифицируется и ему отображается интерфейс с выбором — дать или нет доступ приложению на перечисленные ресурсы. В случае апрува выдачи доступа, пользователь редиректится обратно в приложение вместе с кодом авторизации по ссылке URI, которая задавалась в процессе регистрации Client на Authorization Server. Далее код авторизации обменивается на токен доступа с помощью которого происходит взаимодействие с ресурсами пользователя. Сразу возникает вопрос: зачем

вообще нужен код авторизации, почему не вернуть токен доступа сразу? Фишка кода авторизации в том, что с его помощью токен доступа получает безопасный Client, без посредника в виде браузера пользователя (который установлен на устройстве с вирусами, к примеру). Является рекомендуемым способом авторизации.

Implicit

Данная авторизация отличается от предыдущей лишь тем, что в ней отсутствует код авторизации и токен доступа возвращается сразу. Необходим такой упрощенный сценарий для случаев когда Client реализован в браузере или мобильном приложении, т.е. на стороне пользователя. Такой подход увеличивает отклик приложения, но является небезопасным по причине упомянутой выше.

Resource owner password credentials

Client запрашивает у пользователя логин и пароль, а затем обменивает их на токен доступа. Такой вариант авторизации должен использоваться при достаточном доверии между пользователем и Client (например, если Client – стандартное приложение ОС – first party app), или если других доступных вариантов авторизации нет. Хотя в этом подходе и раскрываются логин и пароль пользователя, Client использует их единожды при получении токена доступа, таким образом исключается необходимость их хранения.

Client credentials

Используется в случаях когда Client взаимодействует с Resource Server от своего имени, т.е. Client является Resource Owner (например, приложение обновляет статистику о количестве активных пользователей).

Получение авторизации

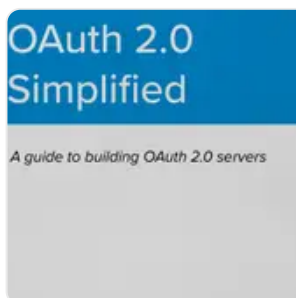
Рассмотрим наиболее часто встречающийся кейс: имеется клиент-браузер и веб-сервер — последний находится на сервере к которому нет публичного доступа. Это значит что веб-сервер может безопасно использовать Client Secret, полученный на этапе регистрации приложения. Теперь по порядку:

Создание ссылки, которая перенаправит пользователя на Authorization Server:

https://authorization-server.com/auth?response_type=code&client_id=CLIENT_ID  `lir`

- `response_type=code` — говорит о том, что сервер ожидает получить код авторизации.
- `client_id` — значение полученное во время регистрации приложения.
- `redirect_uri` — ссылка, на которую, после успешной авторизации, Authorization Server делает редирект пользователя вместе в токеном (или кодом доступа).
- `scope` — один и больше значений, обозначающих к каким ресурсам пользователя необходимо иметь доступ. Необходимые значения берутся из соответствующей документации на Authorization Server.
- `state` — случайная строка, которую Authorization Server должен вернуть обратно на `redirect_uri`, её необходимо будет проверить на равенство. Нужно это для защиты от CSRF (`redirect_uri` ведь не защищен, а это значит что недобросовестное лицо может собрать `uri` со своим токеном или кодом доступа, сделать так, чтобы другой пользователь перешел по этому `uri`, и тогда пользователь будет работать с чужими ресурсами, а здесь уже и приватные данные можно засветить, к примеру).

Если пользователь согласился предоставить доступ к своим ресурсам, то Authorization Server сделает редирект обратно на Client.




OAuth 2.0 Simplified - A guide to building OAuth 2.0 servers

 example-app.com

- `code` — Authorization Server возвращает код авторизации.
- `state` — Authorization Server возвращает то же значение, что было передано в шаге 1.

Необходимо сравнить переменную `state` чтобы быть уверенным в валидности запроса.

Client меняет код авторизации на токен доступа делая POST запрос:

`https://authorization-server.com/token?grant_type=authorization_code&code=AU`  `.0I`

- `grant_type=authorization_code` — вид авторизации.
- `code` — код авторизации, полученный на шаге 2.
- `redirect_uri` — должен быть таким же как в шаге 1.
- `client_id` — значение полученное во время регистрации приложения.
- `client_secret` — значение полученное во время регистрации приложения.

В случае успеха Authorization Server отвечает токеном доступа и временем в течении которого токен валиден, иначе — ошибкой.

Пример авторизации в vk с помощью OAuth, взаимодействие с ресурсами пользователя

Применим полученные знания на практике. Задача: делегировать стороннему сервису доступ к базовой информации пользователя в vk с использованием OAuth.

Решение:

Регистрируем приложение (сторонний сервис) в [разделе для разработчиков](#) Указываем `redirect_uri` в настройках приложения.

Test App

Information

Settings

Stored functions

Statistics

Management

Help

Settings

App ID

7612089

Secure key

Service token

App status

First API request

App installation

Optional

Open API

Enabled

Open API

Website address:

Website focus:

Base domain:

Authorized redirect URI:

[Add another](#)

Save

Далее развернем веб-сервер и имплементируем для него пару эндпоинтов: один для редиректа пользователя на Authorization Server, а другой для обработки зарегистрированного выше `redirect_uri`. Задачу будем решать на Go.

Создадим следующие переменные:

```

tpl := template.Must(template.ParseGlob("templates/*.html"))
clientID := "7607677"
clientSecret := "XVC9zJmiYs6AVM83f3er"
redirectURI := "http://localhost:8080/me"
scope := []string{"account"}
state := "12345"

```



Собираем ссылку для редиректа пользователя на Authorization Server:

```

func index(w http.ResponseWriter, r *http.Request) {
    scopeTemp := strings.Join(scope, "+")

```




```

url := fmt.Sprintf("https://oauth.vk.com/authorize?response_type=code&
    client_id=%s&redirect_uri=%s&scope=%s&state=%s", clientID, redirectURI, sc
    tpl.ExecuteTemplate(w, "index.html", url)
}

```

Принимаем ответ от Authorization Server'a:

```

func me(w http.ResponseWriter, r *http.Request) {
    stateTemp := r.URL.Query().Get("state")
    if stateTemp[len(stateTemp)-1] == '}' { // Забавная особенность,
        // api vk отдает некорректный стейт, его приходится исправлять.
        // Эту багу я наблюдаю уже как неделю. Какой вывод можно сделать?
        // Сторонние приложения, авторизуясь через OAuth vk, не проверяют
        // стейт - а значит подвержены уязвимости CSRF.
        stateTemp = stateTemp[:len(stateTemp)-1]
    }
    if stateTemp == "" {
        respErr(w, fmt.Errorf("state query param is not provided"))
        return
    } else if stateTemp != state {
        respErr(w, fmt.Errorf("state query param do not match original one, got=%s",
        return
    }
    code := r.URL.Query().Get("code")
    if code == "" {
        respErr(w, fmt.Errorf("code query param is not provided"))
        return
    }
    url := fmt.Sprintf("https://oauth.vk.com/access_token?grant_type=authorizator
    redirect_uri=%s&client_id=%s&client_secret=%s", code, redirectURI, clientID, clie
    req, _ := http.NewRequest("POST", url, nil)
    resp, err := http.DefaultClient.Do(req)
    if err != nil {
        respErr(w, err)
        return
    }
    defer resp.Body.Close()
    token := struct {
        AccessToken string `json:"access_token"`
    }{}
    bytes, _ := ioutil.ReadAll(resp.Body)
    json.Unmarshal(bytes, &token)
    url = fmt.Sprintf("https://api.vk.com/method/%s?v=5.124&access_token=%s", "use
    req, err := http.NewRequest("GET", url, nil)
    if err != nil {
        respErr(w, err)
        return
    }
    resp, err := http.DefaultClient.Do(req)
    if err != nil {
        respErr(w, err)
        return
    }
    defer resp.Body.Close()
}

```

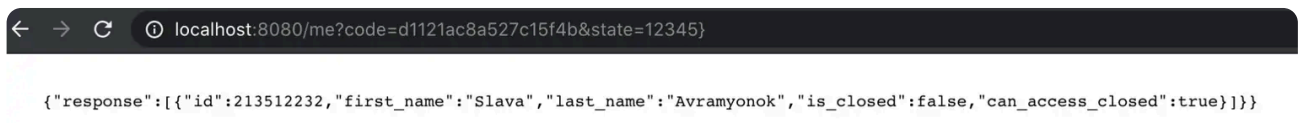
```
bytes, err := ioutil.ReadAll(resp.Body)
if err != nil {
    respErr(w, err)
    return
}
tmpl.ExecuteTemplate(w, "me.html", string(bytes))
}
```

Запускаем веб-сервер:

```
func main() {
    http.HandleFunc("/", index)
    http.HandleFunc("/me", me)
    log.Println("-> Server has started")
    log.Print(http.ListenAndServe(":8080", nil))
    log.Println("-> Server has stopped")
}
```



Обращаемся на localhost:8080/, переходим по сгенерированной ссылке на Authorization Server, получаем следующий результат:



Резюме

В статье были разобраны основы спецификации OAuth, а также решена тестовая задача по делегирования прав доступа пользовательской информации vk стороннему сервису. Для большего понимания OAuth 2.0 рекомендуется почитать соответствующий white paper. [Здесь](#) можно найти код веб-сервиса разработанного выше.

Источники

- [OAuth 2 Simplified](#).
- [The OAuth 2.0 Authorization Framework](#).
- [The OAuth 1.0 Protocol](#).



Следите за новыми постами по любимым темам



API (+)

Golang (+)

19K открытий 23K показов

♡ 2 💬 1



1 комментарий

Сначала интересные ▾

Написать комментарий



Dmitry Kuzmenec

02 июня 2021

api vk отдает некорректный стейт, его приходится исправлять - лишняя фигурная скобка в шаблоне :)



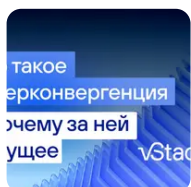
0



0

Ответить

Также рекомендуем



Что такое гиперконвергенция и почему за ней будущее

Гиперконвергенция — современный способ сделать проще, эффективнее и быстрее работу всей IT-инфраструктуры. Рассказываем, что это такое и почему она выгодна бизнесу на примере vStack HCP.



Context Collapse: как микросервисы могут сойти с ума

Даже самая идеальная микросервисная архитектура может упасть. В статье обсудим зарубежный материал, где автор рассказывает о проблеме Context Collapse.



Hexagonal Architecture: Почему старая добрая многослойка больше не работает?

Что такое Hexagonal Architecture. Показываем основные возможности применения гексагональной архитектуры в программировании. Рассматриваем пошаговую инструкцию и основные нюансы



10 библиотек Python, которые меняют карьеру

10 библиотек Python, которые помогут прокачаться в аналитике, ML и разработке. Как они работают и почему меняют карьеру.