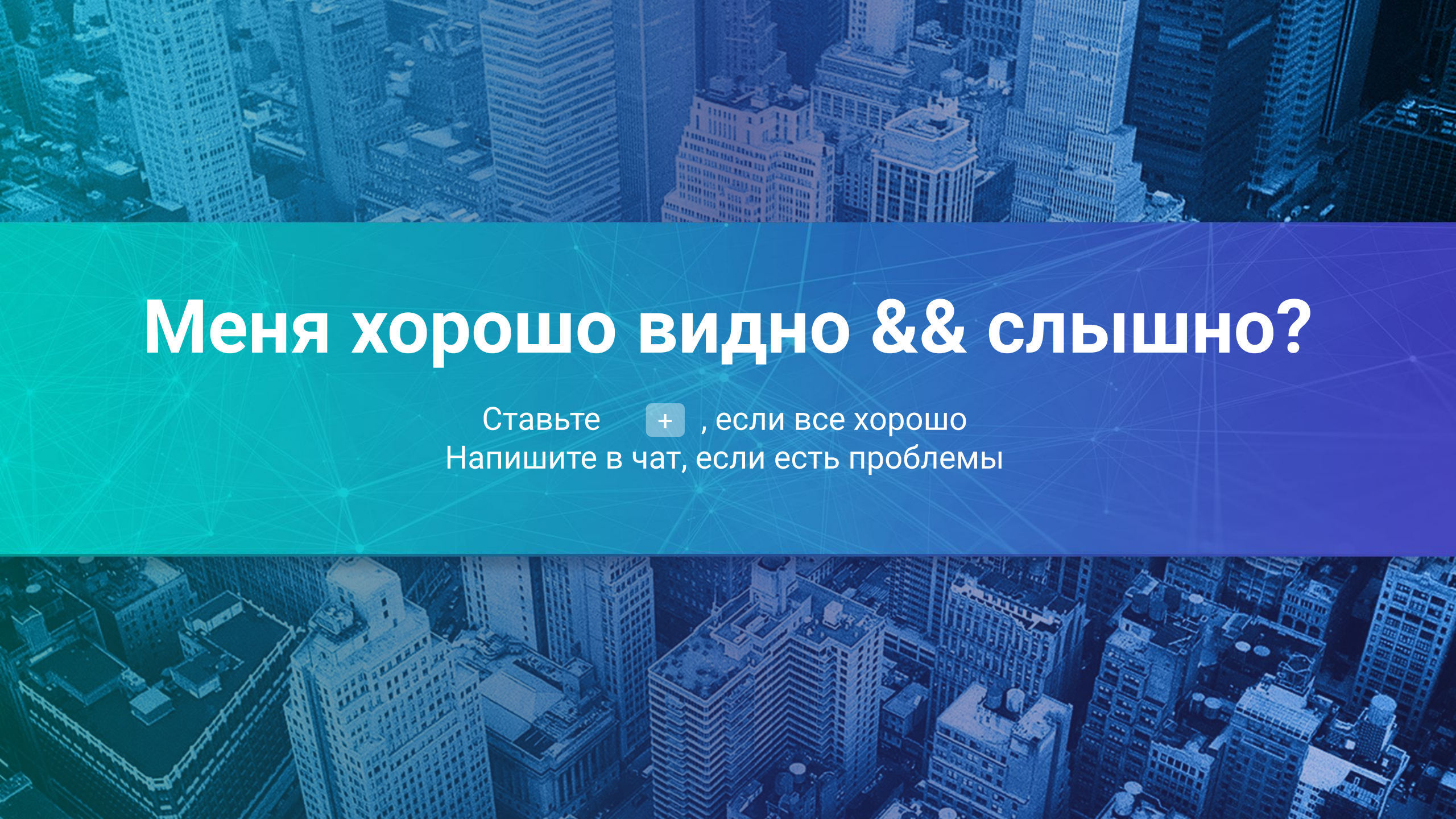

OTUS
ОНЛАЙН-ОБРАЗОВАНИЕ
Онлайн-образование



Меня хорошо видно && слышно?

Ставьте + , если все хорошо
Напишите в чат, если есть проблемы

Проверить, идет ли запись!

Уязвимости OAuth2 и HTTP Response Splitting

Сергей Гоппиков

Head of Backend

Alyce Inc

grey2k@gmail.com telegram: @grey_2k

Преподаватель

Сергей Гоппиков

- Программирую с 14 лет.
- 8 лет опыта на Python/PHP/JavaScript/Kotlin и немного Go.
- 5 лет занимаюсь архитектурой, внедрением практик DevOps и оптимизацией
- Руководил отделом разработки хостинга и регистратора доменных имен Beget.com обеспечивал безопасность парка 500+ серверов, регулярно работали по программам BugBounty
- Руковожу backend командной и разрабатываю архитектуру для гео-распределенного Американского стартапа Alyce.com.

Ежегодно готовимся и проходим аудит безопасности soc2 , вендор - Veracode.com

Правила вебинара

Активно участвуем и делимся опытом

Задаем вопрос в чат или голосом

Off-topic обсуждаем в Slack #web-security-2020-07

Вопросы вижу в чате, могу ответить не сразу

Цели вебинара | После занятия вы сможете

1. Безопасно внедрять и эксплуатировать OAuth2

2. Защищать web-приложения от атак HTTP Response Splitting

Смысл | Зачем вам это уметь

1. Позволит вам проектировать безопасные приложения

2. Видеть потенциальные уязвимости там где кажется их нет

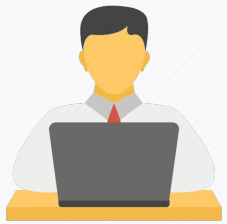
План лекции

- **OAuth2**
  - Введение в OAuth2
  - Authorization Code Flow
  - Implicit Flow
  - Password Credentials Flow
  - Client Credentials Flow
  - Аудит безопасности

- **HTTP Response Splitting**
  - Методология возникновения
  - Виды эксплуатации
  - Инструменты

Уязвимости OAuth2

OAuth2 - что и зачем



User

website.com
Client

user’s request

authorisation url

consent screen

user’s authorisation request

redirect + authorisation code

client’s authorization request

access_token

client’s resources request

response to user

Resource & Authorisation Server
social-network.com

OAuth2 - стандарт и предназначение

Фреймворк авторизации OAuth 2.0 (RFC 6749)

позволяет стороннему приложению получать ограниченный доступ к службе HTTP либо от имени владельца ресурса, организуя взаимодействие по утверждению между владельцем ресурса и службой HTTP, либо позволяя стороннему приложению получать доступ от своего собственного имени. Эта спецификация заменяет и признает устаревшим протокол OAuth 1.0, описанный в RFC 5849.

RFC
https://tools.ietf.org/html/rfc6749

RFC - Proof Key for Code Exchange by OAuth Public Clients
https://tools.ietf.org/html/rfc7636

OAuth2 - совместимость

OAuth 2.0 предоставляет богатую структуру авторизации с четко определенными свойствами безопасности. Однако, будучи богатой и очень расширяемой платформой со многими дополнительными компонентами, эта спецификация сама по себе, вероятно, приведет к широкому спектру несовместимых реализаций.

Кроме того, эта спецификация оставляет некоторые необходимые компоненты частично или полностью неопределенными (например, регистрация клиента, возможности сервера авторизации, обнаружение эндпоинта). Без этих компонентов клиенты должны будут вручную и специально настроены для взаимодействия с определенным сервером авторизации и сервером ресурсов.

Эта структура была разработана с четким расчетом на то, что будущая работа позволит усовершенствовать предписывающие профили и расширения, необходимые для достижения полной совместимости в веб-масштабе.

https://tools.ietf.org/html/rfc6749#section-1.8

OAuth2 - Терминология и действующие лица

Client (OAuth Client)
Клиентское приложение (например сайт), имеющее **client_id** и **client_secret**

Authorization Server (OAuth Server)
Сервер реализующий авторизацию Client'ов, выдает **access_token**, **refresh_token** для запрашиваемых ресурсов (**scopes**)

Resource Server
Сервер предоставляющий ресурсы по запросу с **access_token**

Consent Screen
Страница авторизации и(или) подтверждения доступа к ресурсам

User (пользователь)
Пользователь, предоставляющий доступ к ресурсам от своего имени, может быть получен при авторизации в виде ресурса (**id_token**)

OAuth2 - flows (grants)

Authorization Code Flow (Authorization Code Grant)
Классический и самый безопасный процесс с участием пользователя, основан на процессе авторизации по специальному коду

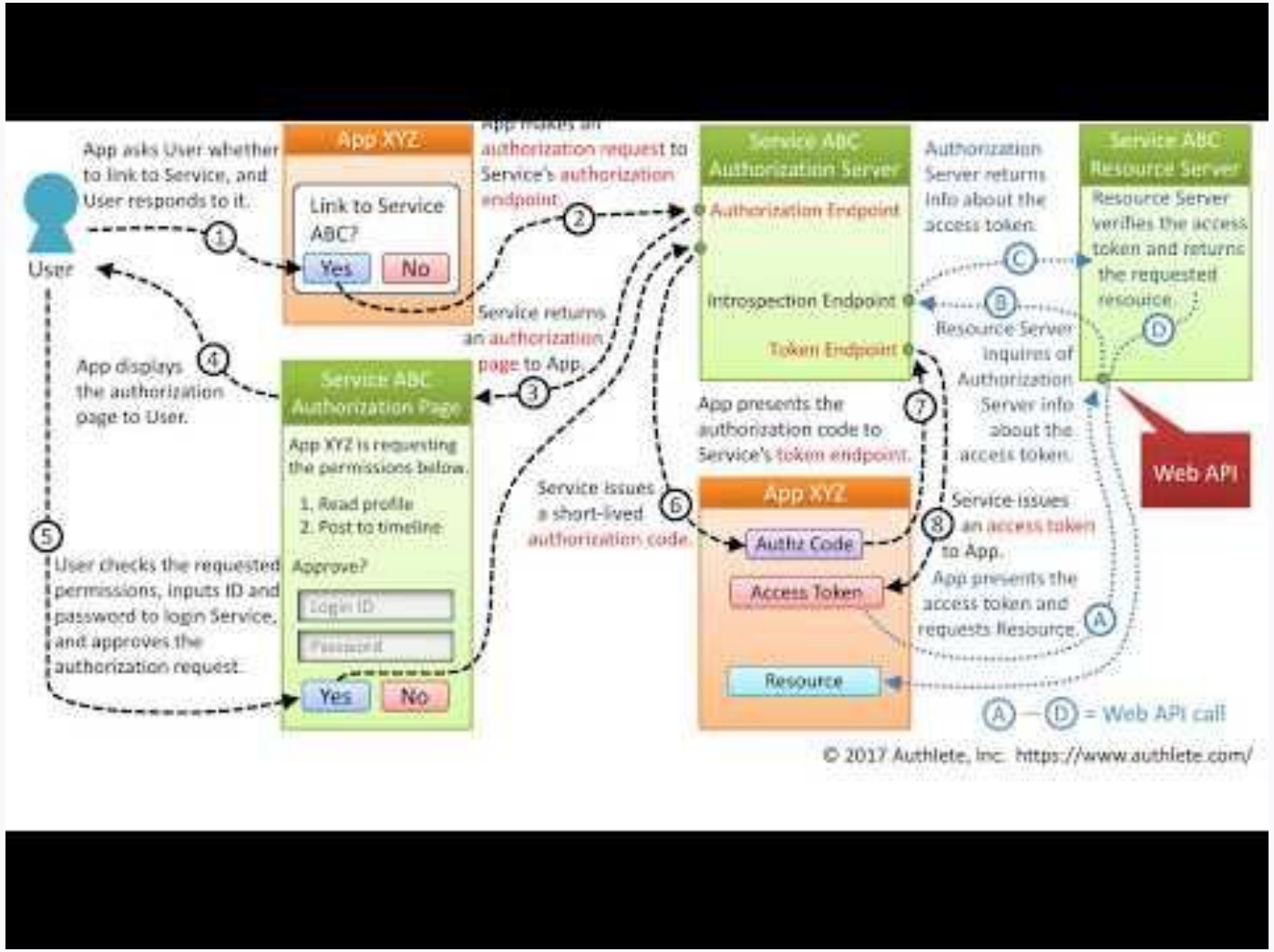
Implicit Flow (Implicit Grant)
Упрощенный способ авторизации, изначально спроектирован для использования приложениями без бэкенда (SPA, Native Apps)

Resource Owner Password Credentials Flow
Аналогично Implicit Flow, но авторизация производится с передачей пароля клиентскому приложению

Client Credentials Flow
Для авторизации без участия пользователя. (Machine-To-Machine)

Refresh Token Flow
Специальный процесс для продления **access_token** клиента с помощью **refresh_token**

OAuth2 - Authorization Code Flow



OAuth2 - Authorization Code Flow

Запрос пользователя для авторизации на OAuth Server

```
GET {Authorization Endpoint}
 ?response type=code                 // - Required,  определяет тип flow
 &client id={Client ID}              // - Required,  идентифицирует приложение (client) на севере
 &redirect uri={Redirect URI}        // - Conditionally required,  uri для возврата в приложение
 &scope={Scopes}                     // - Optional,  ресурсы, доступ к которым запрашивается
 &state={Arbitrary String}           // - Recommended,  параметр для исключения CSRF атак
 &code challenge={Challenge}         // - Optional,  верификация ответа от OAuth сервера
 &code challenge_method={Method}     // - Optional,  верификация ответа от OAuth сервера
HTTP/1.1
HOST: {Authorization Server}
```

Ответ сервера с кодом авторизации

```
HTTP/1.1 302 Found
Location: {Redirect URI}
 ?code={Authorization Code}   // - Always included
 &state={Arbitrary String}    // - Included if the authorization request included 'state'.
```

OAuth2 - Authorization Code Flow

Запрос клиента для получения токена (access_token)

```
POST {Token Endpoint} HTTP/1.1
Host: {Authorization Server}
Content-Type: application/x-www-form-urlencoded
grant type=authorization code  // - Required, определяет тип запроса токена
&code={Authorization Code}     // - Required, код авторизации, полученный на прошлом шаге
&redirect uri={Redirect URI}   // - Required, если в запросе был 'redirect uri'
&code_verifier={Verifier}      // - Required, if the authorization request included 'code_challenge'
```

Ответ сервера с ключом доступа к ресурсу (access_token)

```
HTTP/1.1 200 OK
Content-Type: application/json;charset=UTF-8
Cache-Control: no-store
Pragma: no-cache
{
 "access token": "{Access Token}",     // - Always included
 "token type": "{Token Type}",         // - Always included
 "expires in": {Lifetime In Seconds}, // - Optional
 "refresh token": "{Refresh Token}",   // - Optional
 "scope": "{Scopes}"                   // -  Required если запрошенные scopes отличаются от выданных
}
```

OAuth2 - Authorization Code Flow

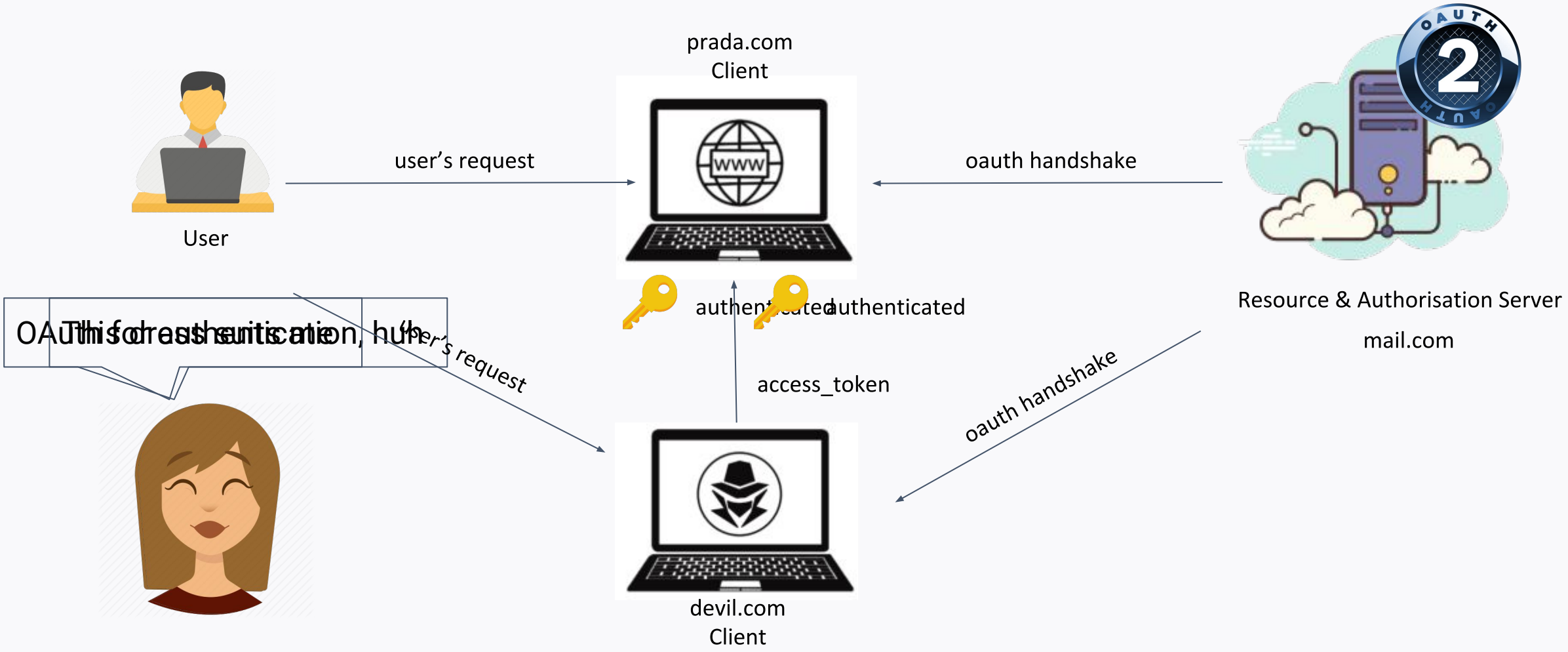
Особенности

- Авторизация доступа к ресурсу происходит **без передачи логина/пароля от сервиса**
- Получение **access_token** происходит через специальный **authorization_code**
- Authorization code должен быть коротким (**не более 10 мин**)
- Код **не должен использоваться повторно**, должен привязан и **валидироваться по client_id**
- **redirect_url должен** допускать только строго определенные (разрешенные) значения
- Специальный **ключ state рекомендуется использовать** для дополнительной защиты от CSRF атак
- **access_token**, **refresh_token**, **client_id**, **client_secret** - должны всегда передаваться по защищенным каналам вне пользовательского контекста и всегда валидироваться
- при компрометации любого ключа или сессии пользователя - **должны отзываться все его токены и сессии**
- при компрометации клиентских ключей - **ключи должны быть перевыпущены**

https://tools.ietf.org/html/rfc6749#section-4.1

OAuth2 - Authorization Code Flow - Эксплуатация

Пример атаки “confused deputy” aka “The Devil Wears Prada”



OAuth2 - Authorization Code Flow - Эксплуатация

Пример атаки “Exploit the redirect URI” aka “Lassie Come Home”

```
https://mail.com/api/auth
    ?response type=code
    &state=123123
    &client id=site1234
    &redirect uri=https%3A%2F%2Fmail.com%2Fapi%2Fauth%0D%0A%3Fresponse_type%3Dcode
%26client_id%3Danna123%26 redirect_url%3Dhttp%3A%2F%2Fanna.com
```

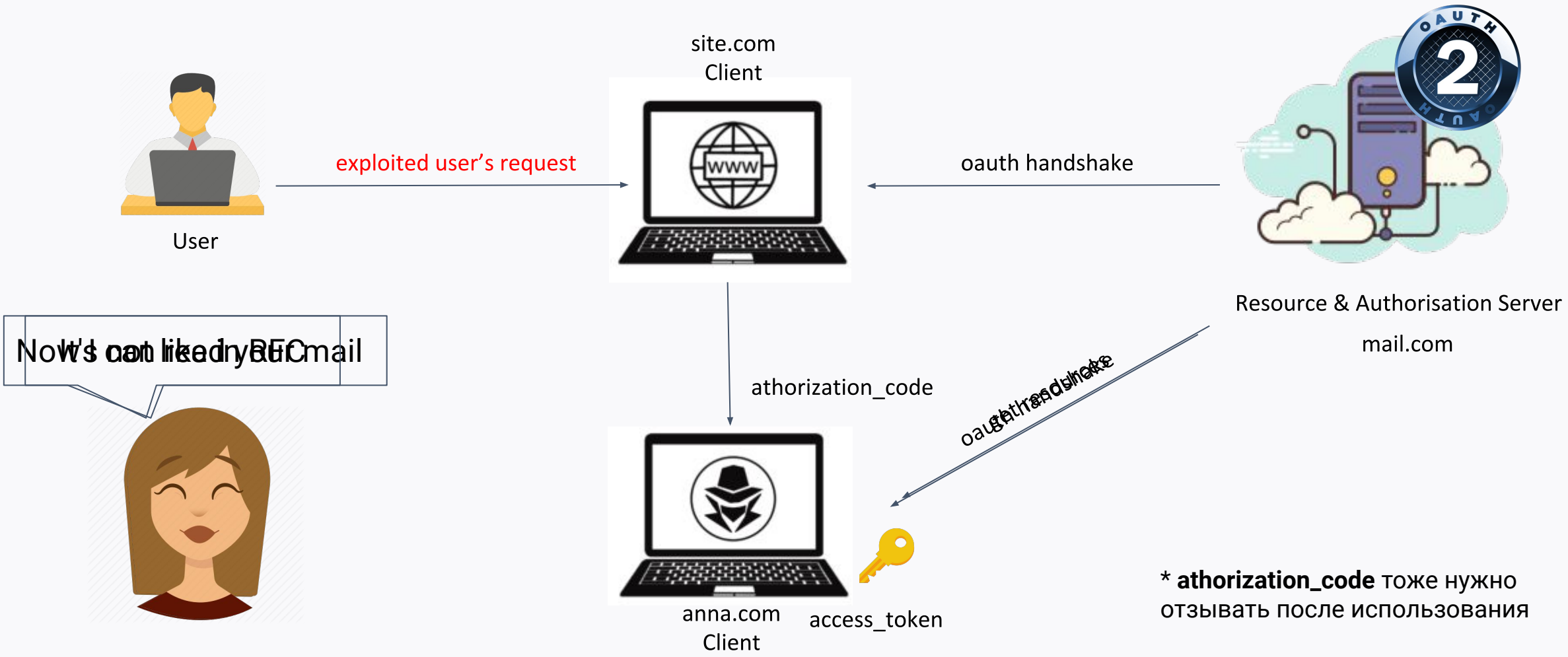
Вместо redirect_uri подставим еще раз URL для аутентификации

```
https://mail.com/api/auth
?response_type=code&client_id=anna123&redirect_url=http://anna.com
```

- Можно вызвать повторный **consent screen**
- Можно попробовать украсть **authorization_code**, который может быть подойдет

OAuth2 - Authorization Code Flow - Эксплуатация

Пример атаки “Exploit the redirect URI” aka “Lassie Come Home”



OAuth2 - Authorization Code Flow - Эксплуатация

Отсутствие валидации access_token (на примере JWT)

header

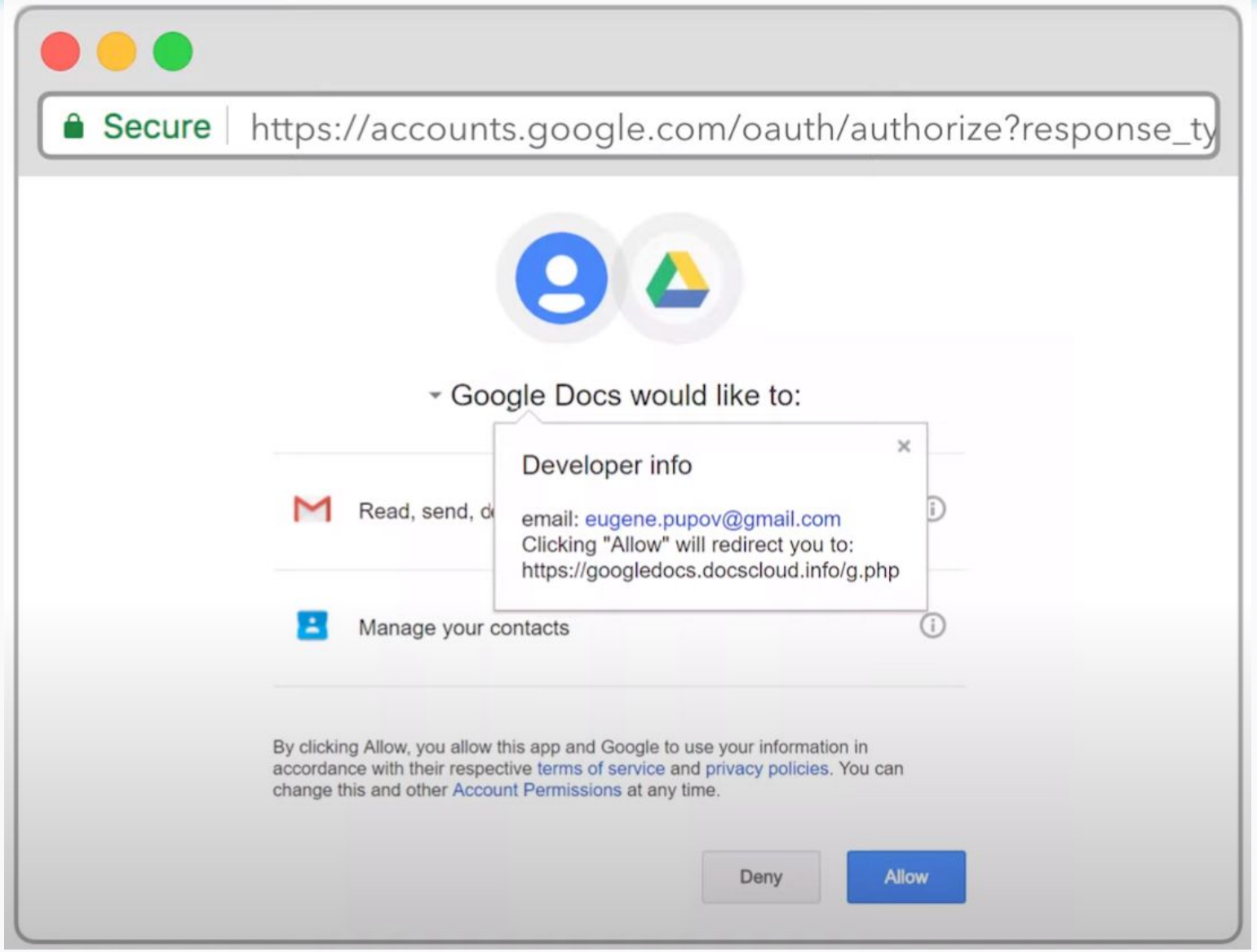
```
{
 "typ": "JWT",
 "alg": "none"
}
```

claims

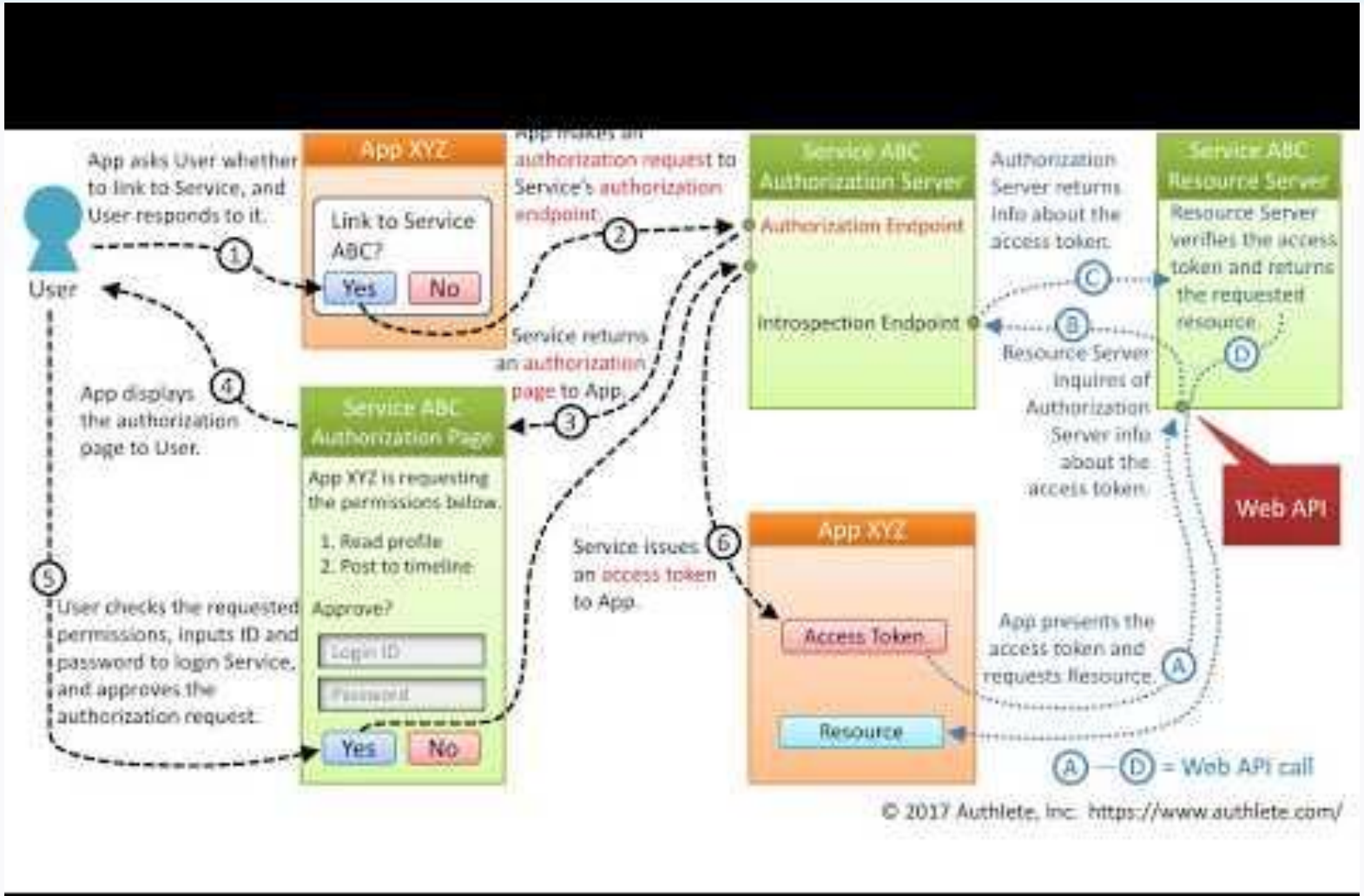
```
{
 "ver": 1,
 "jti": "AT.JwyTSq9j454l3S6dS3USWXLUZpzGJucRwVDlVB5cHsw.US5WSFaQbBYT0/F3kc0o/+VTcuYg7pVvjevSOxdPxB0=",
 "iss": "https://dev-396343.oktapreview.com/oauth2/default",
 "aud": "api://default",
 "iat": 1543803025,
 "exp": 1543806625,
 "cid": "0oahzpp3tcpFrfcWI0h7",
 "uid": "00ui0fjkieyL46ma00h7",
 "scp": [
  "offline_access",
  "photo"
 ],
 "sub": "inquisitive-albatross@example.com"
}
```

OAuth2 - Authorization Code Flow - Эксплуатация

Фишинг consent screen



OAuth2 - Implicit Flow



OAuth2 - Implicit Flow

Запрос пользователя для авторизации на OAuth Server

```
GET {Authorization Endpoint}
 ?response type=token              // - Required,  определяет тип flow
 &client id={Client ID}            // - Required,  идентифицирует приложение (client) на севере
 &redirect uri={Redirect URI}      // - Conditionally required,  uri для возврата в приложение
 &scope={Scopes}                   // - Optional,  ресурсы, доступ к которым запрашивается
 &state={Arbitrary String}         // - Recommended,  параметр для исключения CSRF атак
HTTP/1.1
HOST: {Authorization Server}
```

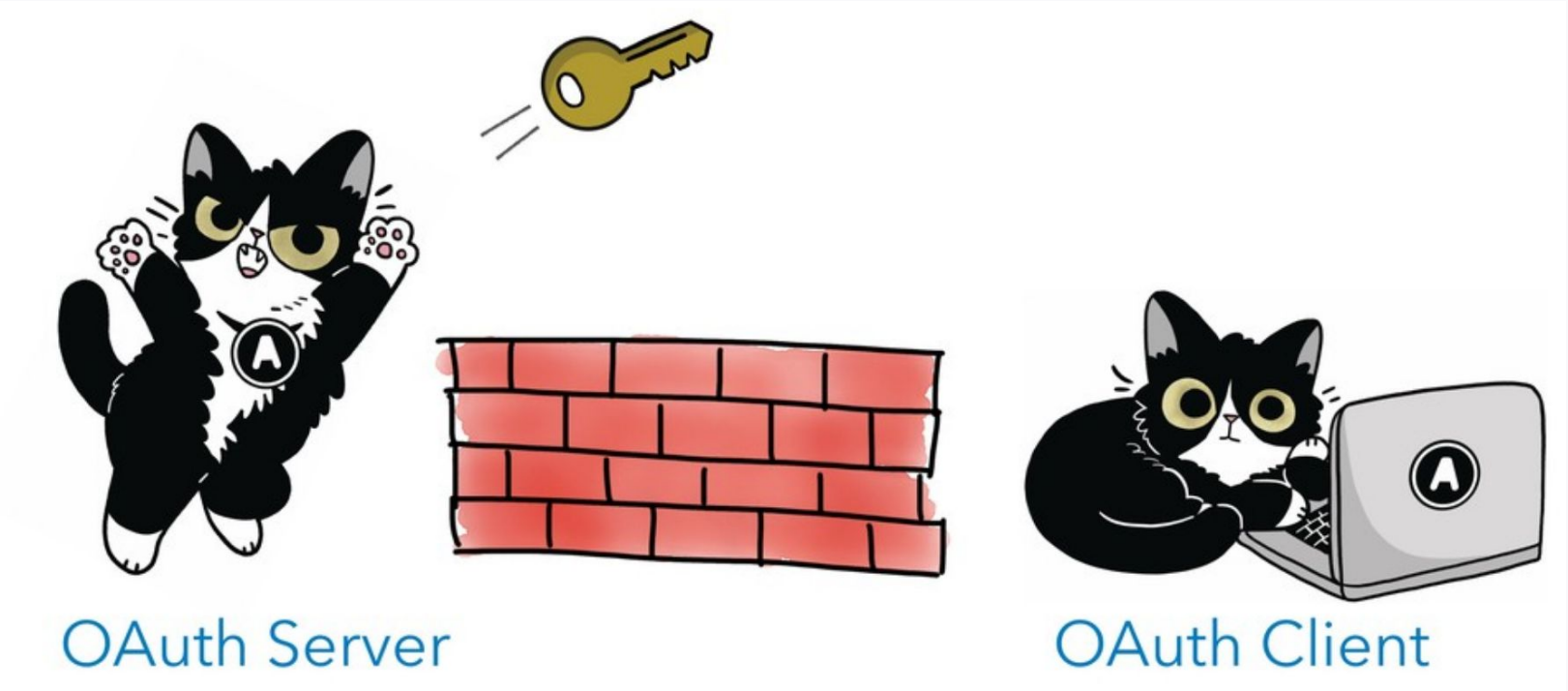
Ответ сервера с ключом доступа к ресурсу (access_token)

```
HTTP/1.1 302 Found
Location: {Redirect URI}
 ?access token={Access Token}        // - Always included
 &token type={Token Type}            // - Always included
 &expires in={Lifetime In Seconds}   // - Optional
 &state={Arbitrary String}           // - Included if the request included 'state'.
 &scope={Scopes}                     // - Required если запрошенные scopes отличаются от выданных
```

OAuth2 - Implicit Flow

Особенности

- Передача ключа через пользовательский контекст (фронтенд)
- Отсутствует защита, токен передается прямо в приложение (браузер) через URL
- Отсутствует refresh_token



https://tools.ietf.org/html/rfc6749#section-4.2

OAuth2 - Implicit Flow - Эксплуатация

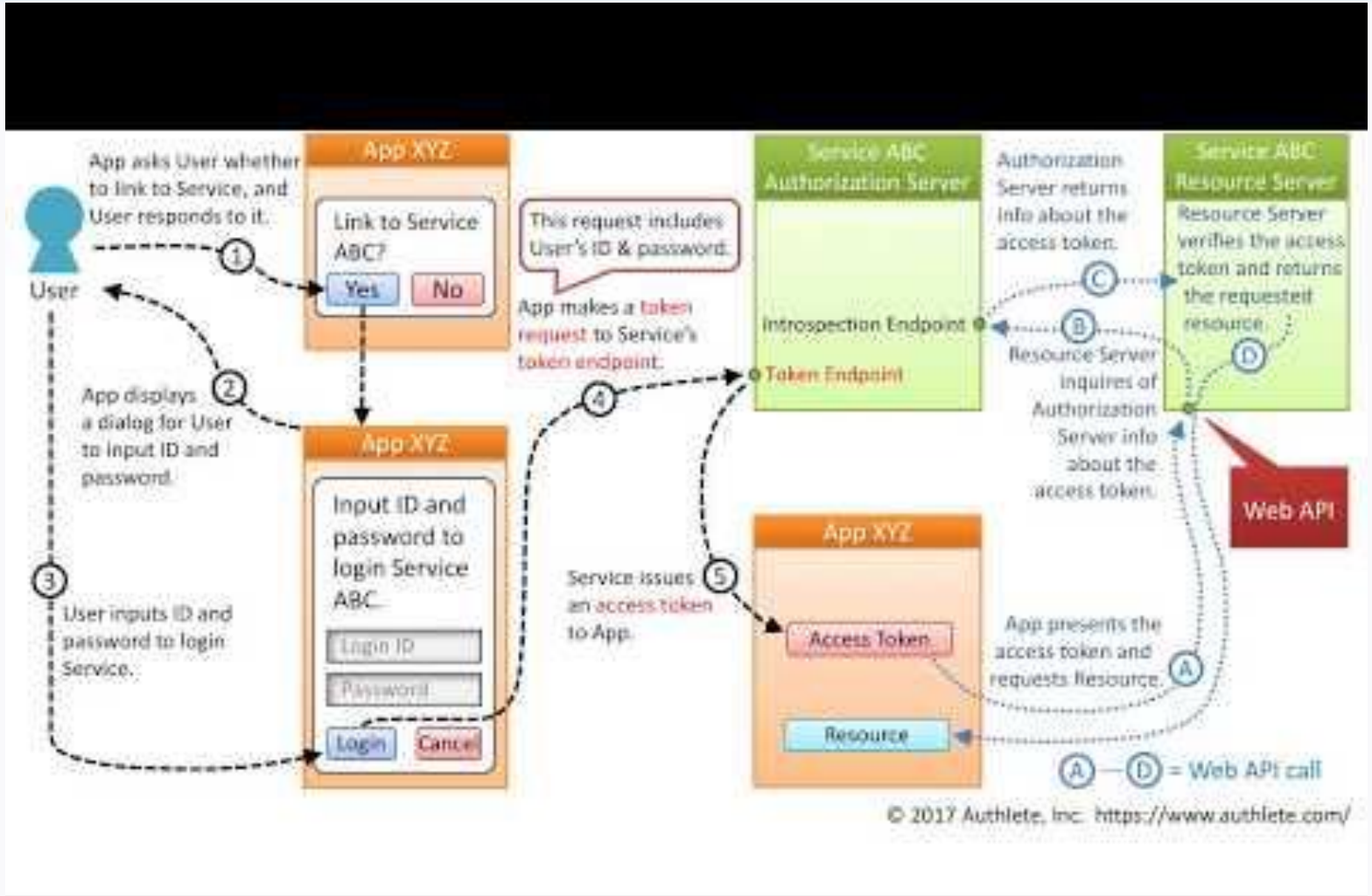
Доступны все методы Authorization Code Flow

+

Весь набор клиентских уязвимостей

- XSS
- Cookie stealing
- Передача токен через Referer заголовок
- Получение истории браузера
- и т.д

OAuth2 - Resource Owner Password Credentials Flow



OAuth2 - Resource Owner Password Credentials Flow

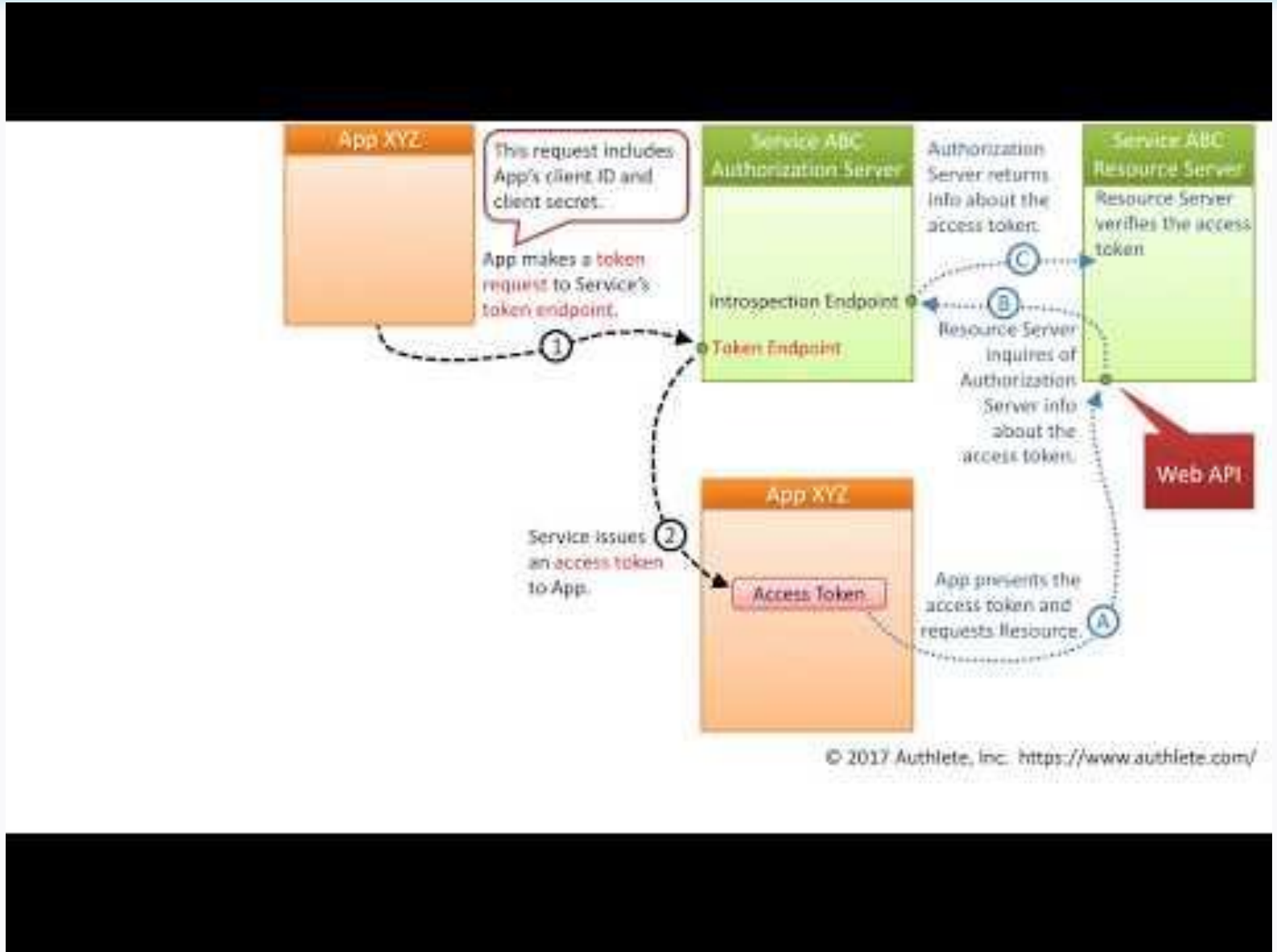
Особенности

- Получение access_token после ввода логина/пароля
- Противоречит основным принципам OAuth2 (делегирование права доступа)
- Используется во встроенных системах
  - холодильники,
  - кофеварки,
  - закрытые системы без возможности перенаправить ответ и открыть consent screen

- **Никогда не используйте для web и современных приложений**
- Запретите его на сервере, если точно не уверены в том что вам это нужно
- Если используете - сделайте защиту от перебора паролей

https://tools.ietf.org/html/rfc6749#section-4.3

OAuth2 - Client Credentials Flow



OAuth2 - Client Credentials Flow

Особенности

- Получение **access_token** с использованием **client_id** и **client_secret**
- Доступ выдается клиенту, а не пользователю
- Используется для взаимодействия сервисов, серверов, приложений между собой
- Только для **confidential** клиентов (https://tools.ietf.org/html/rfc6749#section-2.1), client_id & client_secret должны быть защищены.
- В случае их компрометации - нужно отозвать токены и перевыпустить пару

- **С осторожностью используйте этот механизм для IoT устройств**
- Для IoT посмотрите в сторону OAuth-IoT (еще нет RFC) и других протоколов

https://www.researchgate.net/publication/317936413_OAuth-IoT_An_access_control_framework_for_the_Internet_of_Things_based_on_open_standards

https://tools.ietf.org/html/rfc6749#section-4.4

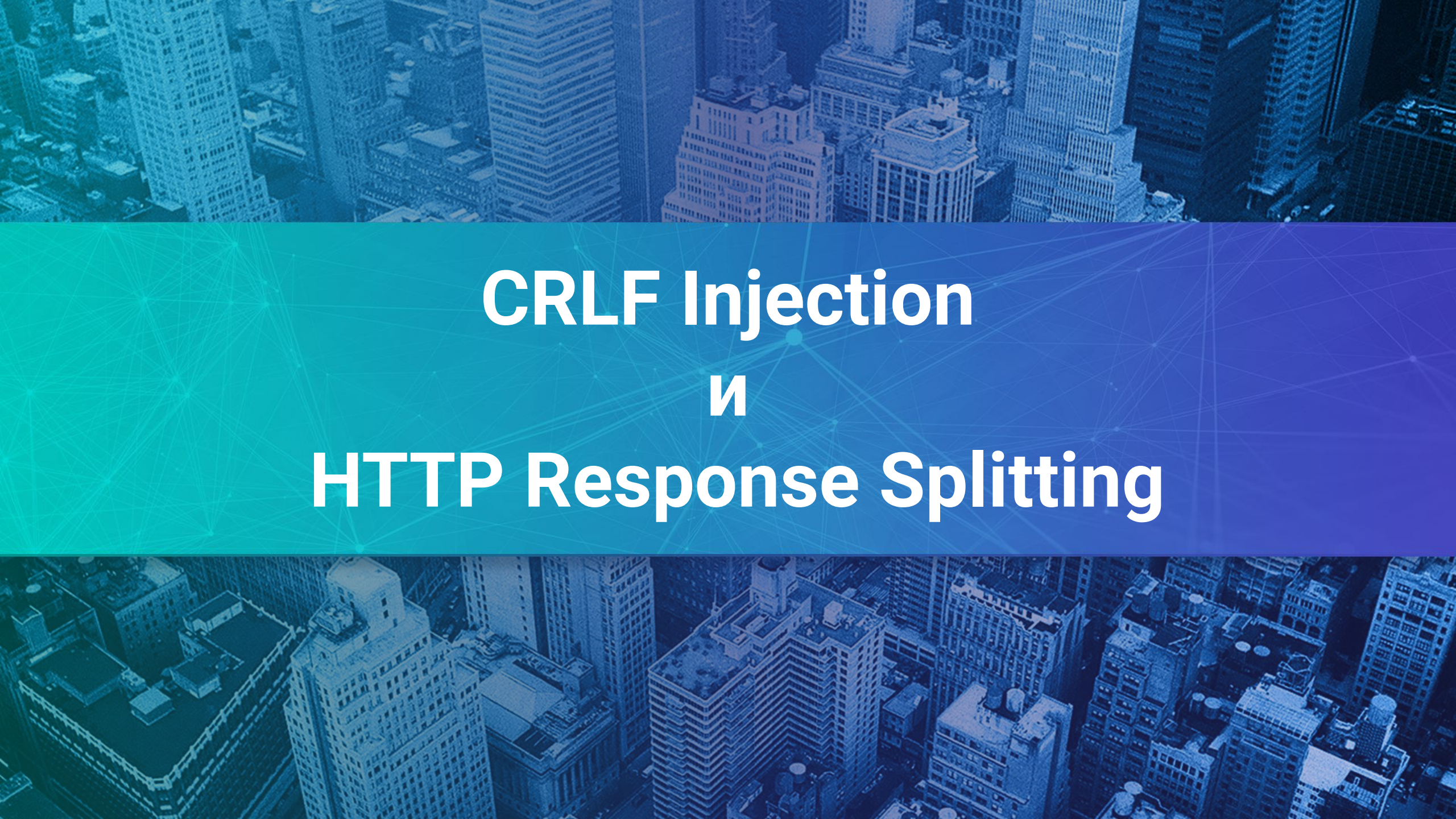
OAuth2 - Аудит безопасности

Если вы провайдер OAuth2 или используете один из провайдеров

- проверьте какие flow используются в вашей системе
- по каждому flow - проверьте соответствие RFC
- соберите документацию, найдите разработчика
- проверьте время жизни токенов и ключей, политики по их отзыву
- проверьте приложения, политику проверки и модерации, правила redirect_uri и т.д

Если вы OAuth Client

- Проверьте flow которое у вас используется, как оно внедрено
- Проверьте решение на соответствие RFC
- Соответствует ли flow - наиболее безопасному
- Используется ли OAuth для аутентификации, какие дополнительные механизмы защиты применены



CRLF Injection
и
HTTP Response Splitting

CRLF Injection - методология возникновения

Недостаточная валидация входных данных запроса

- параметры запроса могут содержать непечатные символы

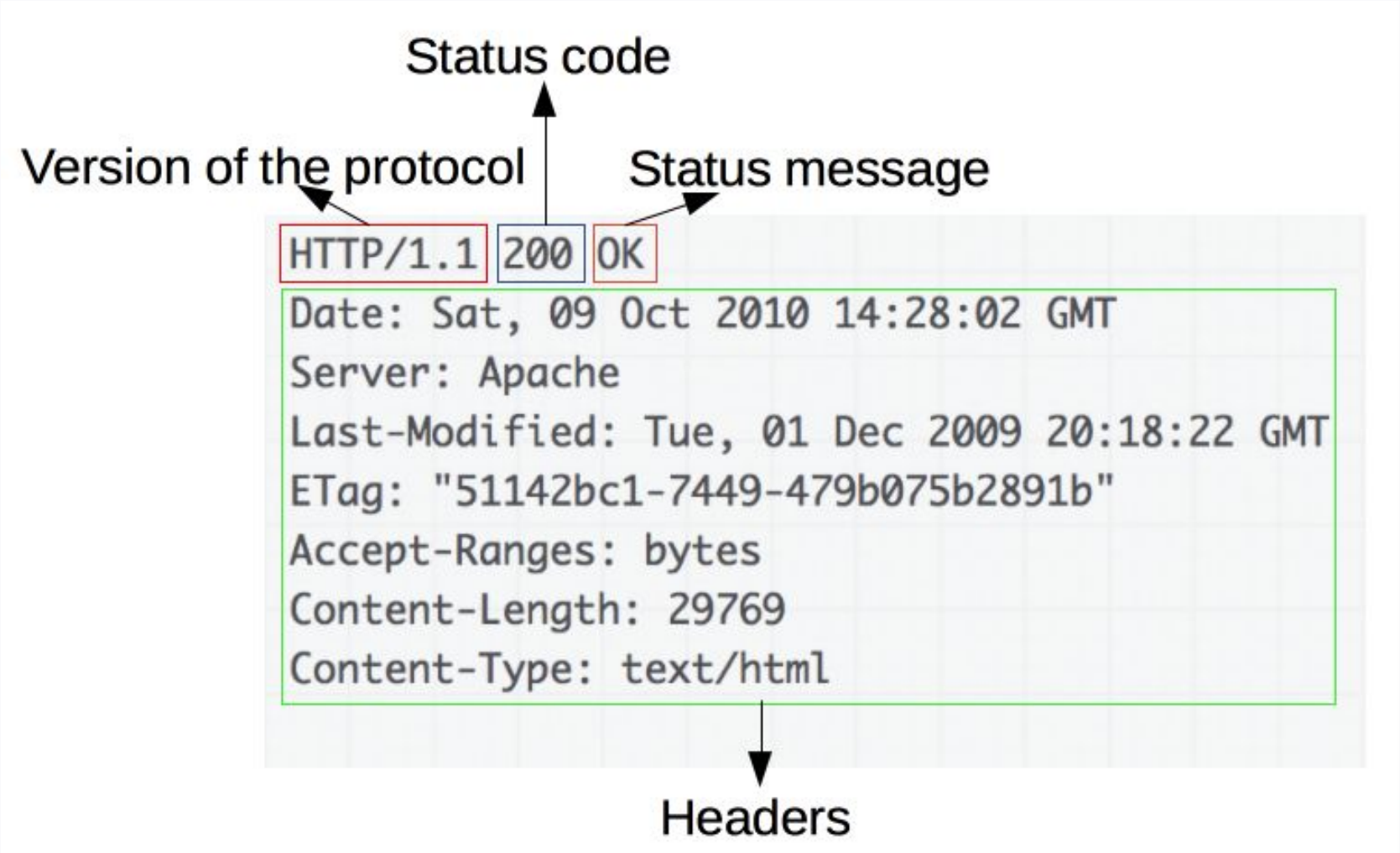
CR (Carriage Return) “\r” %0D или %0d

LF (Line Feed) “\n” %0A или %0a

Рассмотрим CRLF Injection на примере

http://challenge01.root-me.org/web-serveur/ch14/

HTTP Response Splitting - HTTP ответ



HTTP Response Splitting - Сообщение HTTP

Первая строка запроса (ответа)

Заголовки запроса (ответа)

HeaderName: HeaderValue
Header-Name: Header Value

Пустая строка (CRLF)

Данные (HTML , JSON и т.д)

Ссылка на MDN

https://developer.mozilla.org/en-US/docs/Web/HTTP/Messages

Методология возникновения уязвимостей

Недостаточная валидация входных данных HTTP

- сообщение HTTP имеет текстовый формат,
- блоки разделены символами возврата каретки (CR) и переноса строки (LF)
- если не фильтровать эти символы, можно провести инъекцию прямо в HTTP сообщение

CR (Carriage Return) “\r” %0D или %0d

LF (Line Feed) “\n” %0A или %0a

Что будет, если использовать их в заголовках?

HTTP Response Splitting - Эксплуатация

Первая строка ответа

Заголовки ответа

HeaderName: HeaderValue
Header-Name: exploitedValue

CRLF Injection

Инъекция в ответ сервера

Данные (HTML , JSON и т.д)

```
Content-Length: 0

HTTP/1.1 200 OK
Content-Type: text/html
Content-Length: 50

<script>document.write("<h1>Hacked</h1>")</script>
```

https://www.urlencoder.org/

```
%0D%0AContent-Length%3A%200
%0D%0A
%0D%0AHTTP%2F1.1%20200%20OK
%0D%0AContent-Type%3A%20text%2Fhtml
%0D%0AContent-Length%3A%2050
%0D%0A
%0D%0A%3Cscript%3Edocument.write%28%22%3Ch1%3EHacked%3C
%2Fh1%3E%22%29%3C%2Fscript%3E
```

HTTP Response Splitting - Защита

- Не допускайте CR и LF символы в HTTP запросах
- Проверяйте логи на сервере на наличие **%0D** и **%0A** в URL и заголовках

Рефлексия

С какими основными мыслями и инсайтами уходите с вебинара?

Достигли ли вы цели вебинара?

Цели вебинара | Проверка достижения целей

1. Безопасно внедрять и эксплуатировать OAuth2

2. Защищать web-приложения от атак HTTP Response Splitting

Список материалов для изучения

- https://tools.ietf.org/html/rfc6749
- https://medium.com/@darutk/diagrams-and-movies-of-all-the-oauth-2-0-flows-194f3c3ade85
- https://www.jug.ch/events/slides/150311_jug_antoniosanso.pdf
- http://blog.intothesymmetry.com/2013/05/oauth-2-attacks-introducing-devil-wears.html
- https://www.youtube.com/watch?v=aU9RsE4fcRM
- https://stackoverflow.com/questions/41972081/what-are-the-security-risks-of-implicit-flow
- https://stackoverflow.com/questions/17241771/how-and-why-is-google-oauth-token-validation-performed/17439317
- https://nordicapis.com/why-oauth-2-0-is-vital-to-iot-security/
- https://www.netsparker.com/blog/web-security/crlf-http-header/
- https://www.root-me.org/en/Challenges/Web-Client/HTTP-Response-Splitting

Следующий вебинар

Тема: Методологии безопасной разработки (SSDL): обзор сравнение, практическое применения

01.09

Ссылка на вебинар будет в ЛК за 15 минут

Материалы к занятию в ЛК — можно изучать

Обязательный материал обозначен красной лентой


Заполните, пожалуйста,
опрос о занятии по ссылке в чате

Спасибо за внимание!
Приходите на следующие лекции

Сергей Гоппиков

Head of Backend
Alyce Inc
grey2k@gmail.com telegram: @grey_2k