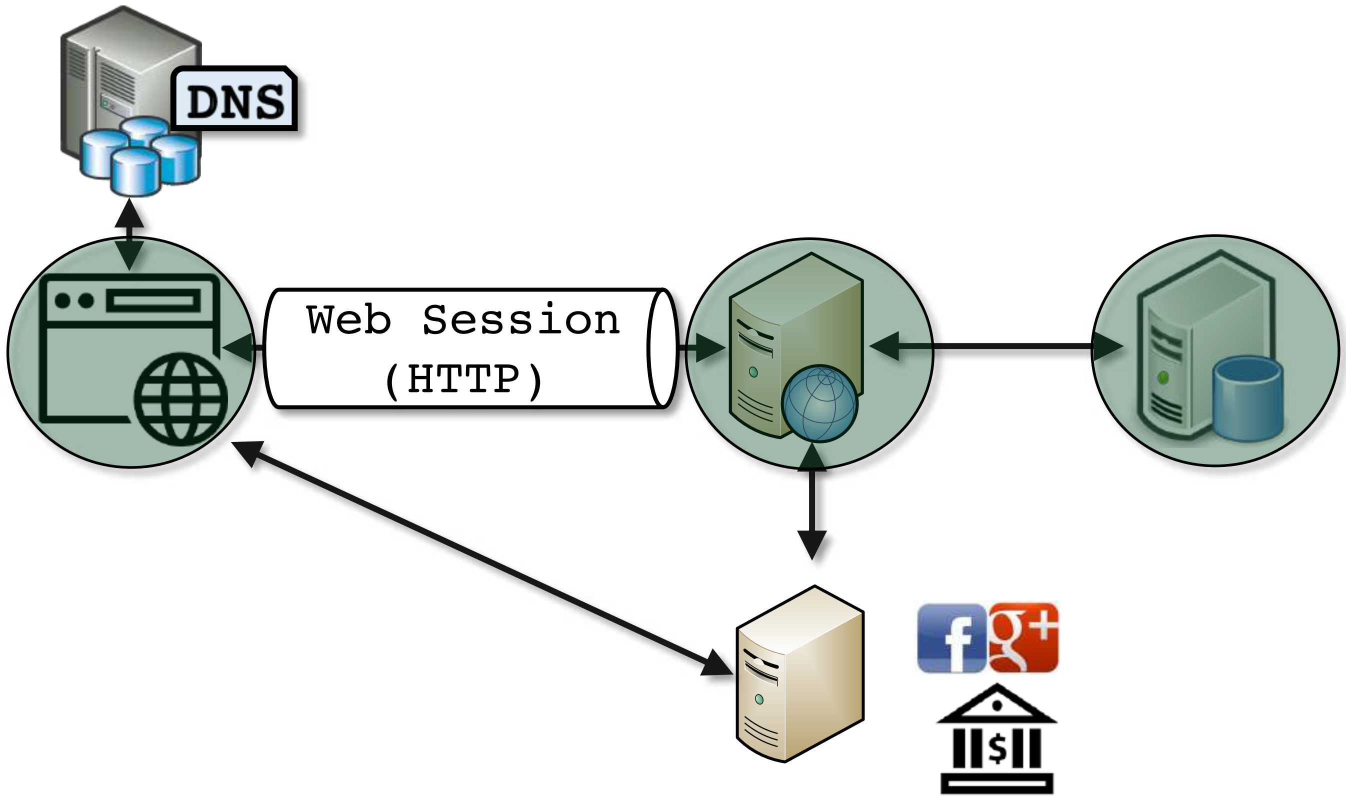
CS5331: Web Security

Lecture 6: Web Authorization/Authorization

A Tale of Two Web Servers



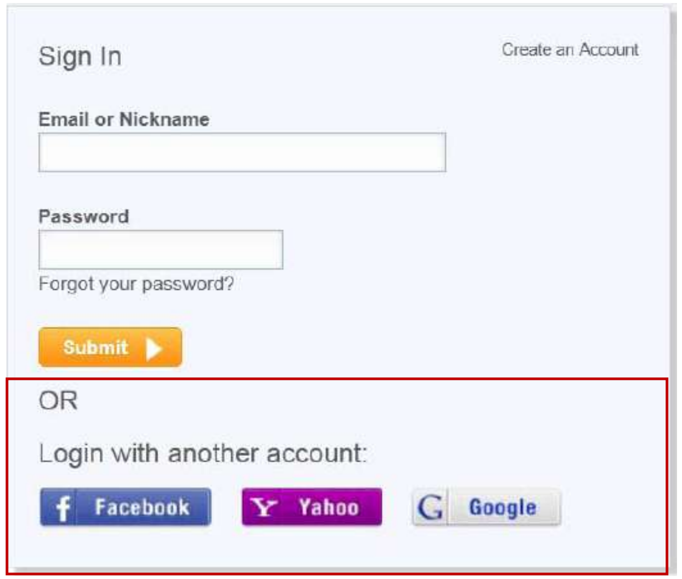
Web Authentication and Authorization

- Authentication
  - Verifying that someone is indeed who they claim to be
- Authorization
  - Deciding which resources a certain user should be able to access, and what they should be allowed to do with those resources.
- Single Sign On
  - Allowing a user to enter one username and password in order to access multiple applications.

Single Sign-On (SSO)

- The idea of Single Sign-On (SSO):
  login once, and authenticate everywhere
- Examples:
  - Kerberos, OpenID & OpenID Connect, OAuth, etc..
  - Usage: Facebook Connect, Google Login, etc…
- Web SSO:
  - Started ~2005, standardized afterwards
  - Offers convenience to the users
  - Widely-used on the Web by billions of users
  - Standardization efforts are still ongoing
  - Its security is still an active research area

- Web Authentication


Sign In

Create an Account

Email or Nickname

Password

Forgot your password?

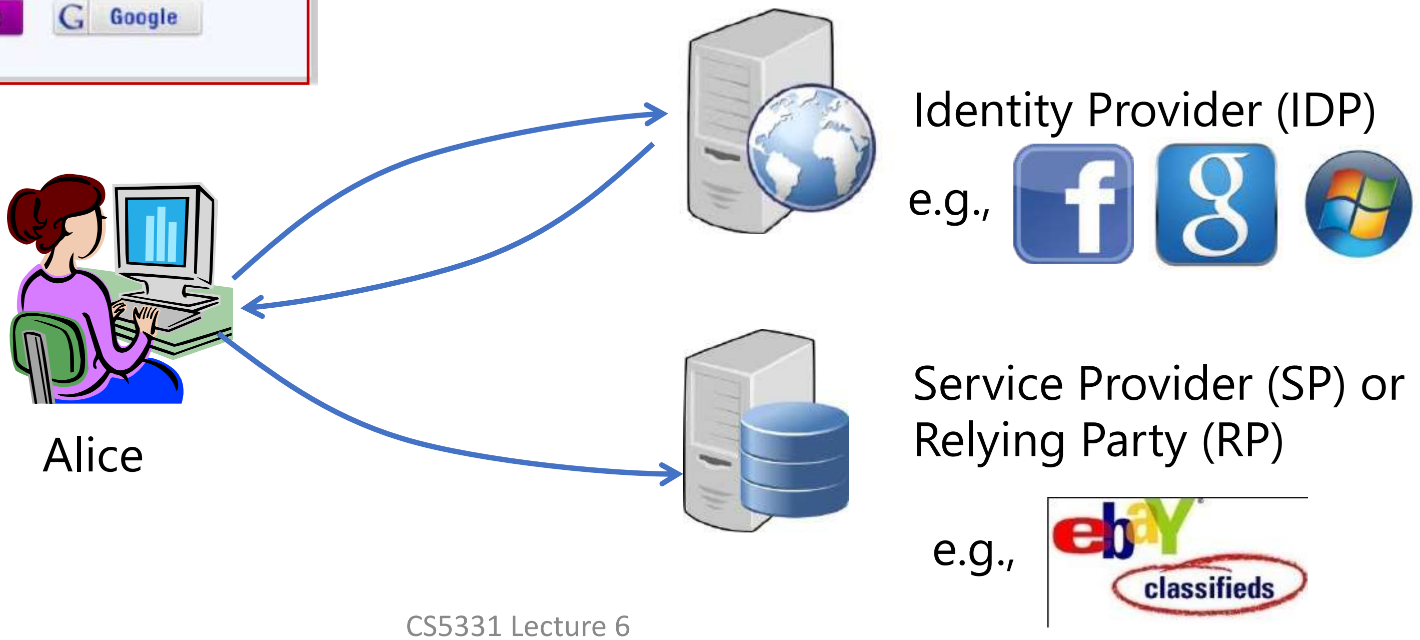
Submit

OR

Login with another account:

Facebook Yahoo Google

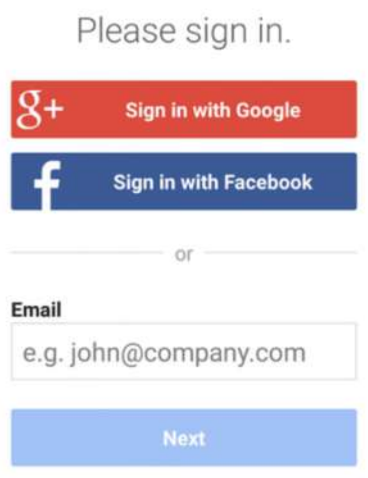
- Web Single Sign-On (SSO):
  - BrowserID (Mozilla)
  - Facebook Connect
    - 250+ Million users, 2,000,000 websites
  - OpenID
    - one billion users, 50,000 websites
  - …



Web Authentication

Web Authentication – OpenID

- OpenID is an open standard for authentication, promoted by the non-profit OpenID Foundation.
  - As of March 2016, there are over a billion OpenID-enabled accounts on the internet, and lots of organizations use OpenID to authenticate users.
    - Google, WordPress, Yahoo, and PayPal
- Entities:
  - User
  - ID Provider (IDP)
  - Web Application (OpenID Consumer)



Security Problems

- Authentication Bugs
  - An attacker could forge an OpenID request that doesn’t ask for the user’s email address, and then insert and unsigned email address into IDPs response.
  - Result: Unauthorized parties to log into victim user’s accounts on the RP.
- Phishing
  - Malicious relaying part may forward the end-user to a bogus identity provider authentication page asking that end-user to input their credentials.
- Privacy and trust issues
  - IDP has the log of OpenID logins. →privacy breach
- Covert Redirect
  - Related to OAuth2.0 and Open ID. 2014
  - An instance of attackers using open redirectors – a well-known threat.

"Signing Me onto Your Accounts through Facebook and Google: a Traffic-Guided Security Study of Commercially Deployed Single-Sign-On Web Services".

Notes

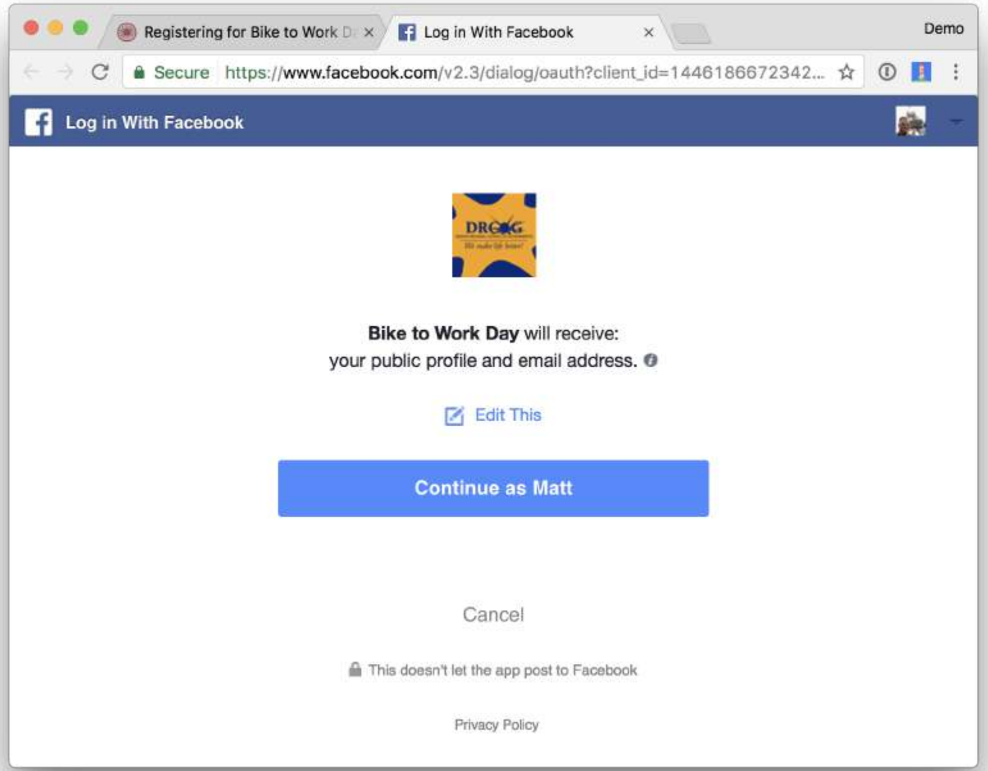
- OpenID is technically a URL that a user owns (e.g. alice2016.openid.com), so some websites offer the option to manually enter an OpenID.
- The latest version of OpenID is OpenID Connect, which combines OpenID authentication and OAuth2 authorization
- *Facebook previously used OpenID but has since moved to Facebook Connect.*

Web Authorization

Web Authorization - OAuth

- **OAuth** is an open standard for access delegation
  - used as a way for Internet users to grant websites or applications access to their information on other websites but without giving them the passwords
  - "secure delegated access" to server resources on behalf of a resource owner.
    - Amazon, Google, Facebook, Microsoft and Twitter
- **OAuth** essentially allows access tokens to be issued to third-party clients by an authorization server, with the approval of the resource owner.
  - The third party then uses the access token to access the protected resources hosted by the resource server.

Typical Scenario


Registering for Bike to Work D

Log in With Facebook

Demo

Secure https://www.facebook.com/v2.3/dialog/oauth?client_id=1446186672342...

Log in With Facebook

Bike to Work Day will receive:
your public profile and email address.

Edit This

Continue as Matt

Cancel

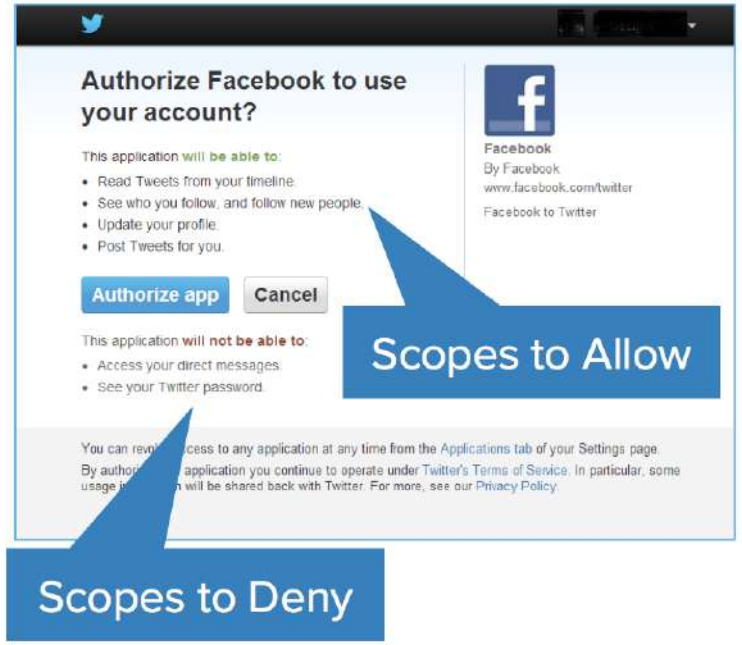
This doesn't let the app post to Facebook

Privacy Policy

OAuth Scopes

- **Scopes** are what you see on the authorization screens when an app requests permissions.
  - They're bundles of permissions asked for by the client when requesting a token.

- **The first key aspect of OAuth.**
  - The permissions are front and center.
  - Not hidden behind the app layer



Authorize Facebook to use your account?

This application will be able to:

- Read Tweets from your timeline.
- See who you follow, and follow new people.
- Update your profile.
- Post Tweets for you.

This application will not be able to:

- Access your direct messages.
- See your Twitter password.

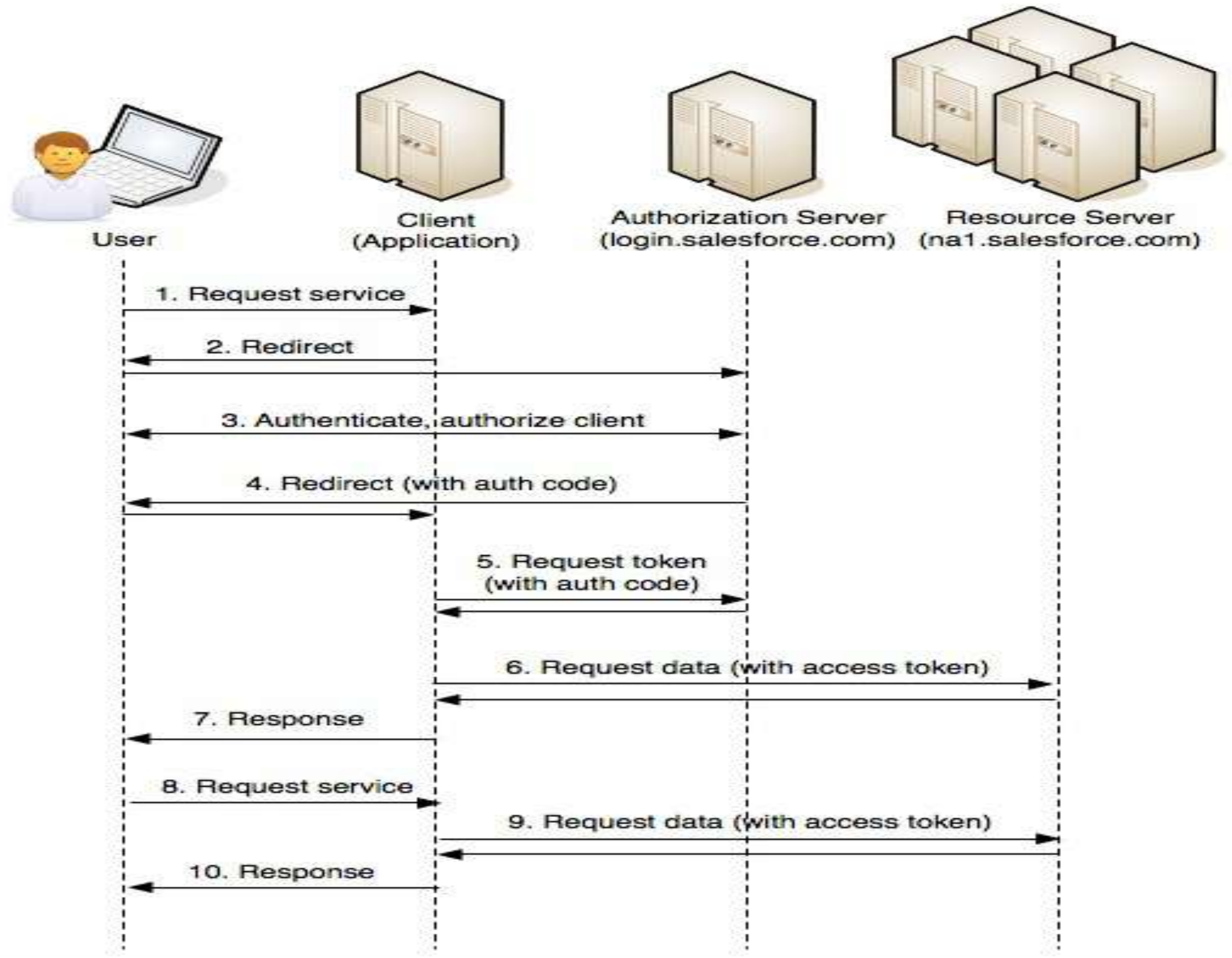
By Facebook
www.facebook.com/twitter
Facebook to Twitter

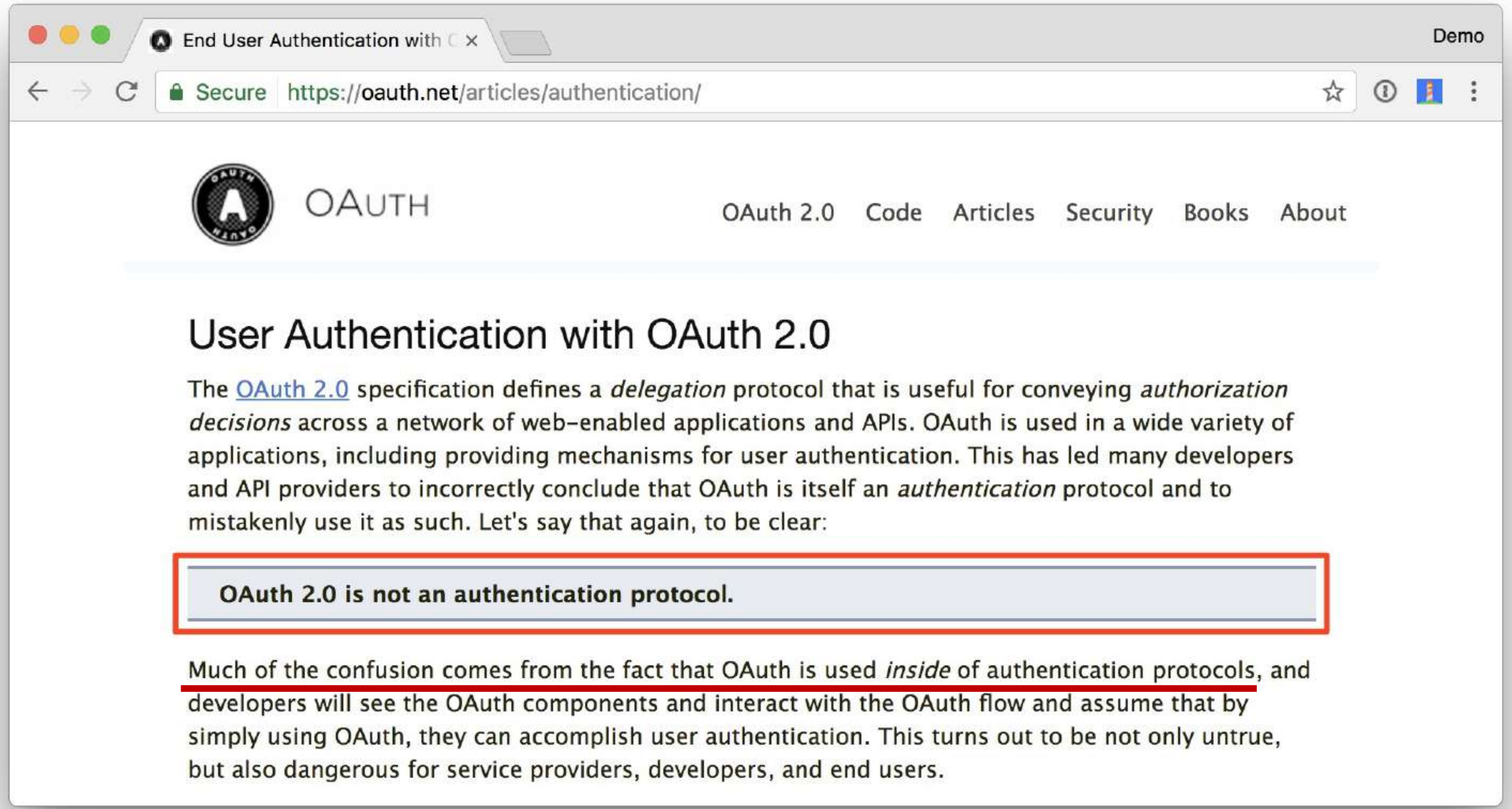
You can revo[illegible]cess to any application at any time from the Applications tab of your Settings page.
By authori[illegible] application you continue to operate under Twitter's Terms of Service. In particular, some usage i[illegible]n will be shared back with Twitter. For more, see our Privacy Policy.

Key Concepts

- **Server or the Resource Provider** controls all OAuth restrictions and is a website or web services API where User keeps her protected data
- **Client or Consumer Application** is typically a web-based or mobile application that wants to access User's Protected Resources
- **User or the Resource Owner** is a member of the Resource Provider, wanting to share certain resources with a third-party web site
- **Authorization Server** (***Identity Provider***) is the server that owns the user identities and credentials. It's who the user actually authenticates and authorizes with
- **Client Credentials** are the consumer key and consumer secret used to authenticate the Client
- **Token Credentials** are the access token and token secret used in place of User's username and password

OAuth – How it works?




End User Authentication with C ×
Demo
Secure https://oauth.net/articles/authentication/
OAUTH
OAuth 2.0 Code Articles Security Books About

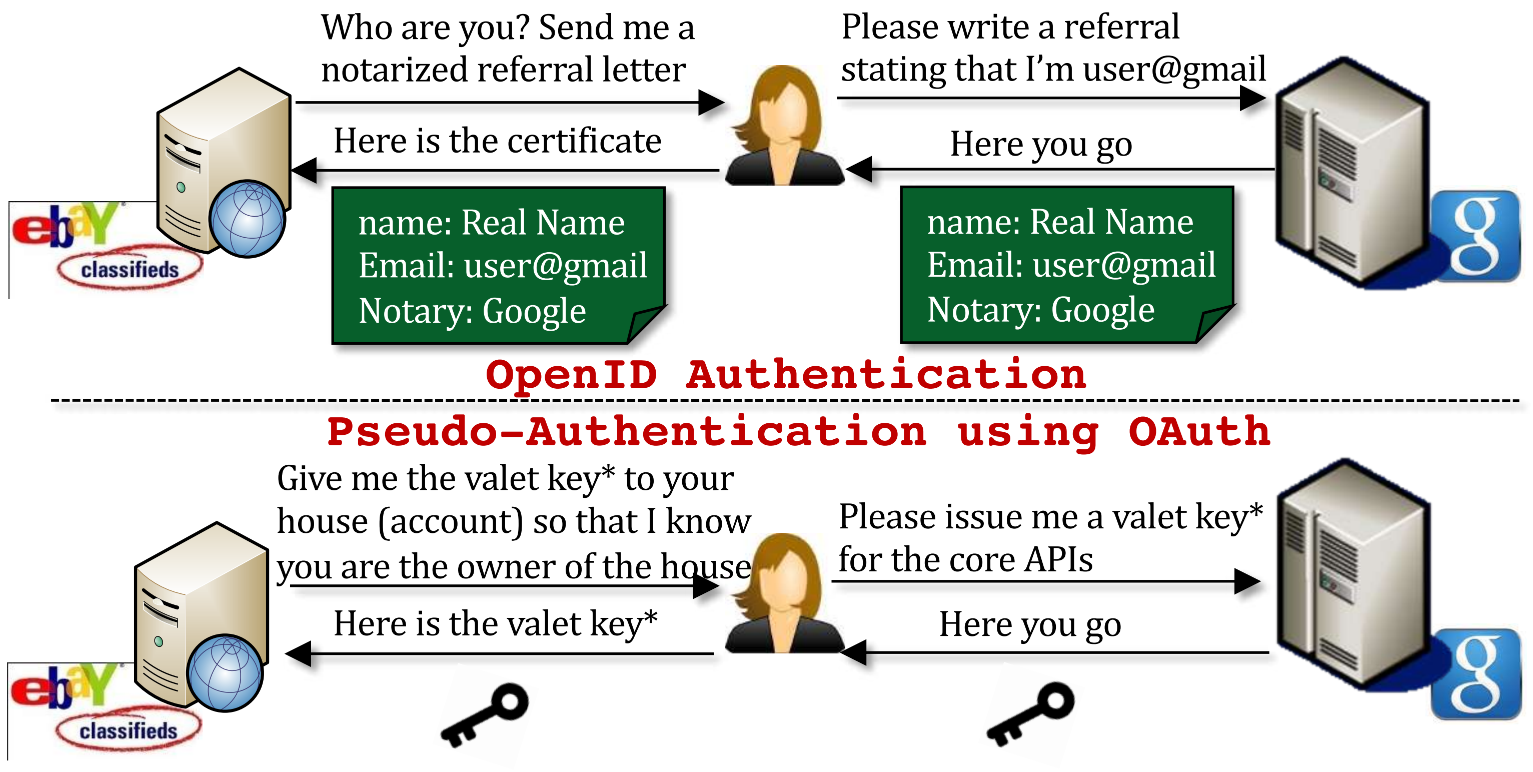
User Authentication with OAuth 2.0

The OAuth 2.0 specification defines a *delegation* protocol that is useful for conveying *authorization decisions* across a network of web-enabled applications and APIs. OAuth is used in a wide variety of applications, including providing mechanisms for user authentication. This has led many developers and API providers to incorrectly conclude that OAuth is itself an *authentication* protocol and to mistakenly use it as such. Let's say that again, to be clear:

OAuth 2.0 is not an authentication protocol.

Much of the confusion comes from the fact that OAuth is used *inside* of authentication protocols, and developers will see the OAuth components and interact with the OAuth flow and assume that by simply using OAuth, they can accomplish user authentication. This turns out to be not only untrue, but also dangerous for service providers, developers, and end users.

Pseudo-Authentication with OAuth 2.0


Who are you? Send me a notarized referral letter
Please write a referral stating that I'm user@gmail
Here is the certificate
Here you go
name: Real Name
Email: user@gmail
Notary: Google
name: Real Name
Email: user@gmail
Notary: Google
OpenID Authentication
Pseudo-Authentication using OAuth
Give me the valet key* to your house (account) so that I know you are the owner of the house
Please issue me a valet key* for the core APIs
Here is the valet key*
Here you go
ebY classifieds

https://en.Wikipedia.org/wiki/OAuth

OpenID vs. OAuth

- OpenID
  - The crucial difference is that in the OpenID *authentication* use case
    - the response from the identity provider is an assertion of identity;
    - In the OAuth *authorization* use case, the identity provider is also an API provider, and the response from the identity provider is an access token that may grant the application ongoing access to some of the identity provider's APIs, on the user's behalf.
- OAuth
  - OAuth is an *authorization* protocol, rather than an *authentication* protocol.
  - Using OAuth on its own as an authentication method may be referred to as pseudo-authentication

Weakness

- Lack Of Data Confidentiality and Server Trust
- Insecure Storage of Secrets
- OAuth Implementation with Flawed Session Management
- Session Fixation Attack with OAuth

Lack Of Data Confidentiality and Server Trust

- Brute Force Attacks Against the Server
  - eavesdrop on the traffic
  - gain access to request parameters and attributes, e.g., oauth_signature, consumer_key, oauth_token, signature_method (HMAC-SHA1), timestamp, etc.
- Lack of Server Trust
  - One way authentication
  - Phishing or other exploits
- Solutions
  - TLS/SSL during the data exchange
  - Reduce the chance of replay attack -- >Nonce

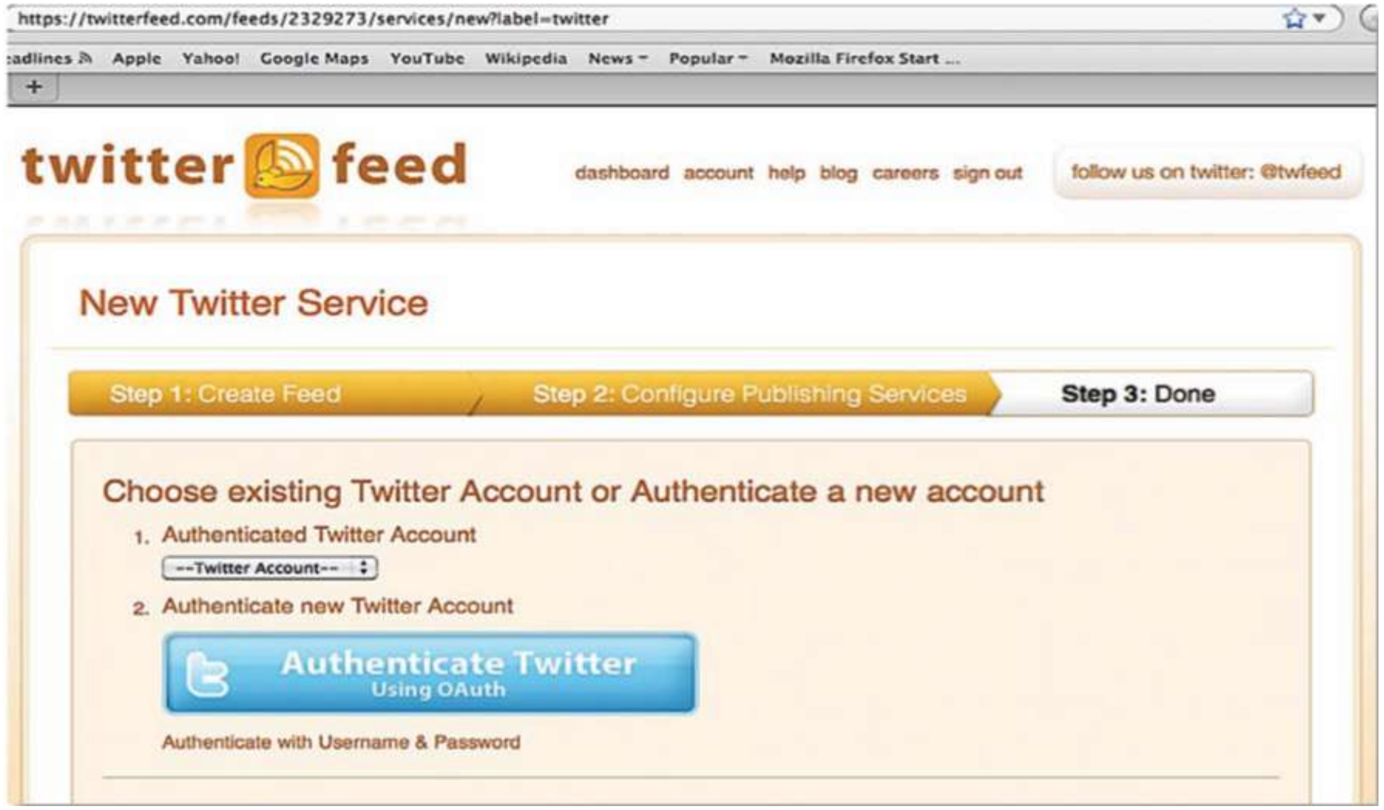
Insecure Storage of Secrets

- The two areas of concern are to protect
  - Shared secrets on the Server
  - Consumer secrets on cross-platform clients
- The threats
  - a compromised consumer secret does NOT directly grant access to User’s protected data.
  - However, compromised consumer credentials could lead to security threats.
  - E.g., The attacker can use the compromised Client Credentials to imitate a valid Client and launch a phishing attack, where he can submit a request to the Server on behalf of the victim and gain access to sensitive data
- Solutions
  - Obfuscate the consumer secret
  - Store the client credentials on the clients backend server

OAuth Implementation with Flawed Session Management

- During the authorization step, the User is prompted by the Server to enter her login credentials to grant permission.
- The user then is redirected back to the Client application to complete the flow.
- The main issue is with a specific OAuth Server implementation, such as Twitter’s, where the user remains logged in on the Server even after leaving the Client application.

- Avon Barksdale registers and signs into his twitterfeed client.
- He then selects a publishing service such as Twitter to post his blog


https://twitterfeed.com/feeds/2329273/services/new?label=twitter

New Twitter Service

Step 1: Create Feed | Step 2: Configure Publishing Services | Step 3: Done

Choose existing Twitter Account or Authenticate a new account

1. Authenticated Twitter Account

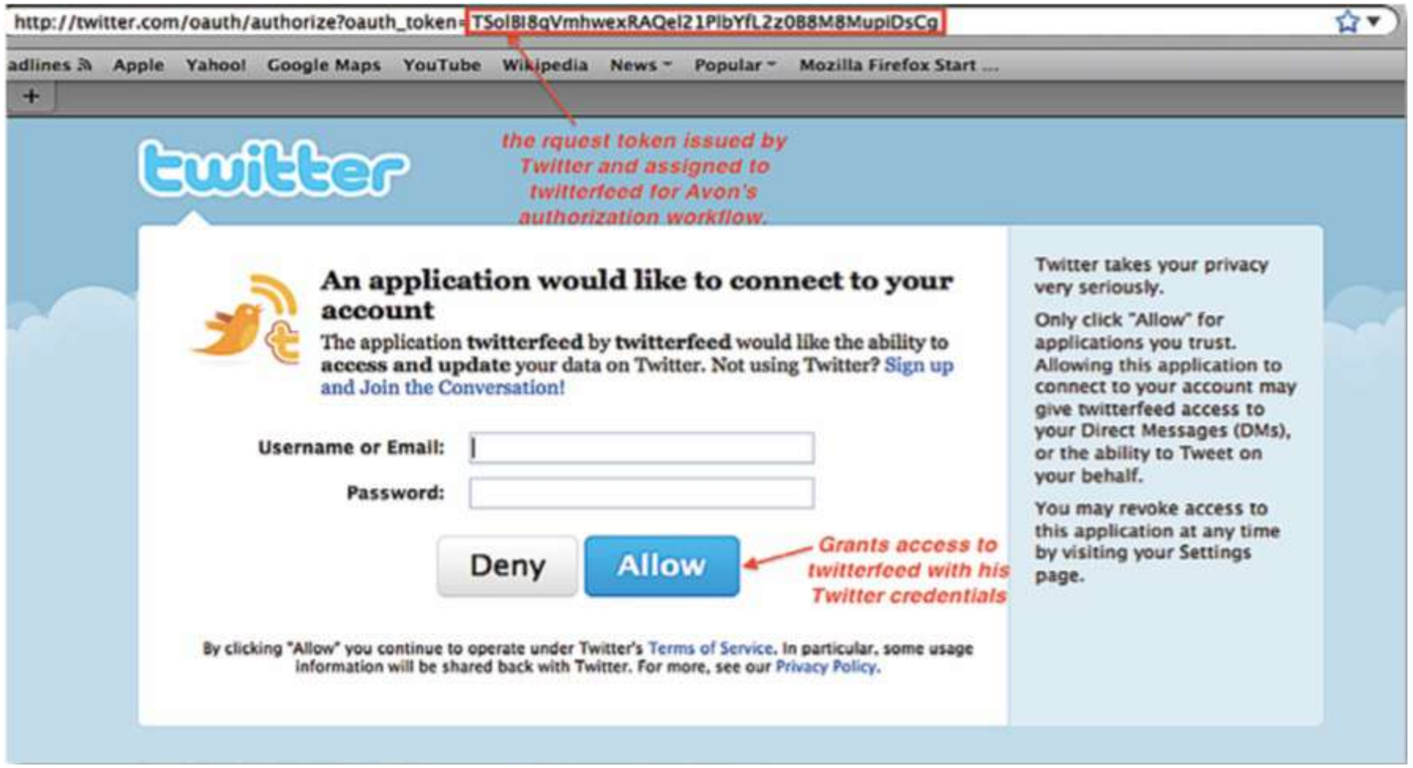
--Twitter Account--

2. Authenticate new Twitter Account

Authenticate Twitter Using OAuth

Authenticate with Username & Password

- Avon is directed to Twitter's authorization endpoint where he signs into Twitter with his username and password and grants access to twitterfeed.


http://twitter.com/oauth/authorize?oauth_token=TSolBI8qVmhwexRAQel21PlbYfL2z0B8M8MupiDsCg

the rquest token issued by Twitter and assigned to twitterfeed for Avon's authorization workflow.

An application would like to connect to your account

The application **twitterfeed** by **twitterfeed** would like the ability to **access and update** your data on Twitter. Not using Twitter? Sign up and Join the Conversation!

Username or Email:

Password:

Grants access to twitterfeed with his Twitter credentials

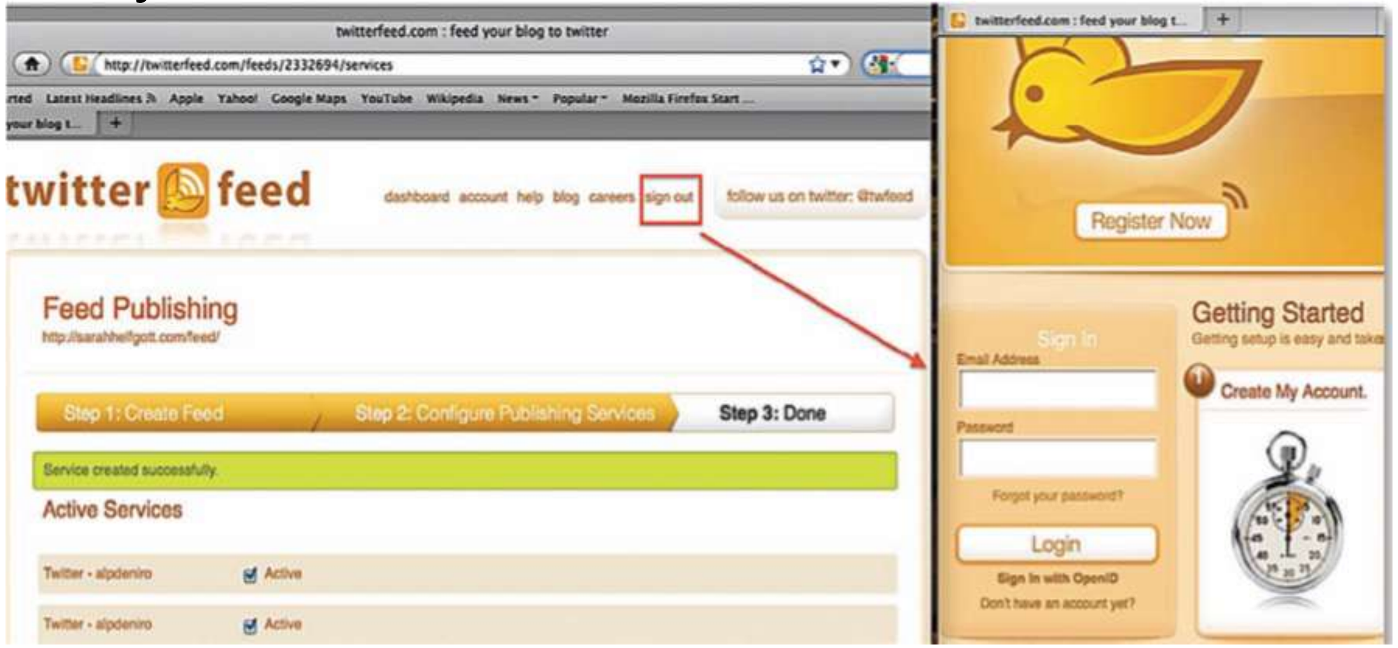
By clicking "Allow" you continue to operate under Twitter's Terms of Service. In particular, some usage information will be shared back with Twitter. For more, see our Privacy Policy.

Twitter takes your privacy very seriously.

Only click "Allow" for applications you trust. Allowing this application to connect to your account may give twitterfeed access to your Direct Messages (DMs), or the ability to Tweet on your behalf.

You may revoke access to this application at any time by visiting your Settings page.

- Now, Avon is redirected back to twitterfeed where he completes the feed; and then signs out of twitterfeed and walks away.



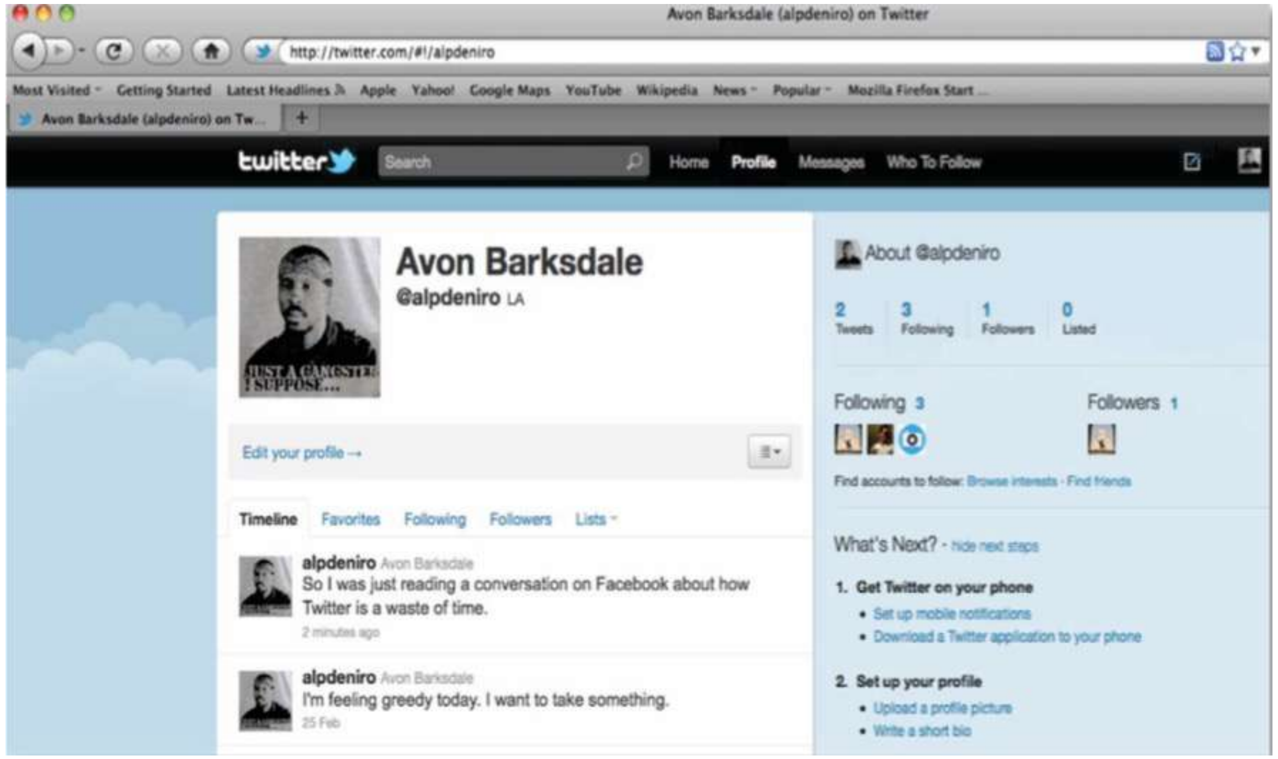
Feed Publishing
http://sarahhelfgott.com/feed/

Service created successfully.

Active Services

Twitter - alpdeniro	Active
Twitter - alpdeniro	Active

- A malicious user with access to the unattended browser can now fully compromise Avon's Twitter account; and deal with the consequences of his action!



Avon Barksdale (alpdeniro) on Twitter
http://twitter.com/#!/alpdeniro
Most Visited Getting Started Latest Headlines Apple Yahoo! Google Maps YouTube Wikipedia News Popular Mozilla Firefox Start
twitter Search Home Profile Messages Who To Follow
Edit your profile →
Timeline Favorites Following Followers Lists
About @alpdeniro
2 Tweets 3 Following 1 Followers 0 Listed
Following 3
Followers 1
Find accounts to follow: Browse interests · Find friends
What's Next? - hide next steps
1. Get Twitter on your phone
• Set up mobile notifications
• Download a Twitter application to your phone
2. Set up your profile
• Upload a profile picture
• Write a short bio

Avon Barksdale
@alpdeniro LA

alpdeniro Avon Barksdale
So I was just reading a conversation on Facebook about how Twitter is a waste of time.
2 minutes ago

alpdeniro Avon Barksdale
I'm feeling greedy today. I want to take something.
25 Feb

OAuth Implementation with Flawed Session Management

- Solutions
  - Resource Providers such as Twitter should always log the User out after handling the third-party OAuth authorization flow.
  - Auto Processing should be turned off. That is, servers should not automatically process requests from clients that have been previously authorized by the resource owner.

A Simple Session Fixation Attack Scenario

- 1. Mallory has determined that http://unsafe.example.com/ accepts any session identifier, accepts session identifiers from query strings and has no security validation.
  - http://unsafe.example.com/ is thus not secure.
- 2. Mallory sends Alice an e-mail:
  - "Hey, check this out, there is a cool new account summary feature on our bank, http://unsafe.example.com/?SID=I_WILL_KNOW_THE_SID".
  - Mallory is trying to fixate the SID to I_WILL_KNOW_THE_SID.
- 3. Alice is interested and visits
  - http://unsafe.example.com/?SID=I_WILL_KNOW_THE_SID.
  - The usual log-on screen pops up

Session Fixation Attack with OAuth

- Session Fixation flaw makes it possible for an attacker to use social-engineering tactics to lure users into exposing their data
- 1) The attacker uses a valid Client app to initiate a request to the Resource Provider to obtain a temporary Request Token. He then receives a redirect URI with this token:
  - http://<resource_provider.com>/oauth/authorize?oauth_token=XyZ
- 2) The attacker uses social-engineering and phishing tactics to lure a victim to follow the redirect link with the server-provided Request Token

Session Fixation Attack with OAuth

- 3) The victim follows the link, and grants client access to protected resources.
  - This process authorizes the Request Token and associates it with the Resource Owner

Resource Provider has no way of knowing whose Request Token is being authorized, and cannot distinguish between the two users.

Session Fixation Attack with OAuth

- 4) The Attacker constructs the callback URI with the “authorized” Access Token and returns to the Client:
  - http://<client_app.com>/feeds/.../oauth_token= XyZ&...
  - If constructed properly, the Attacker’s client account is now associated with victim’s authorized Access Token

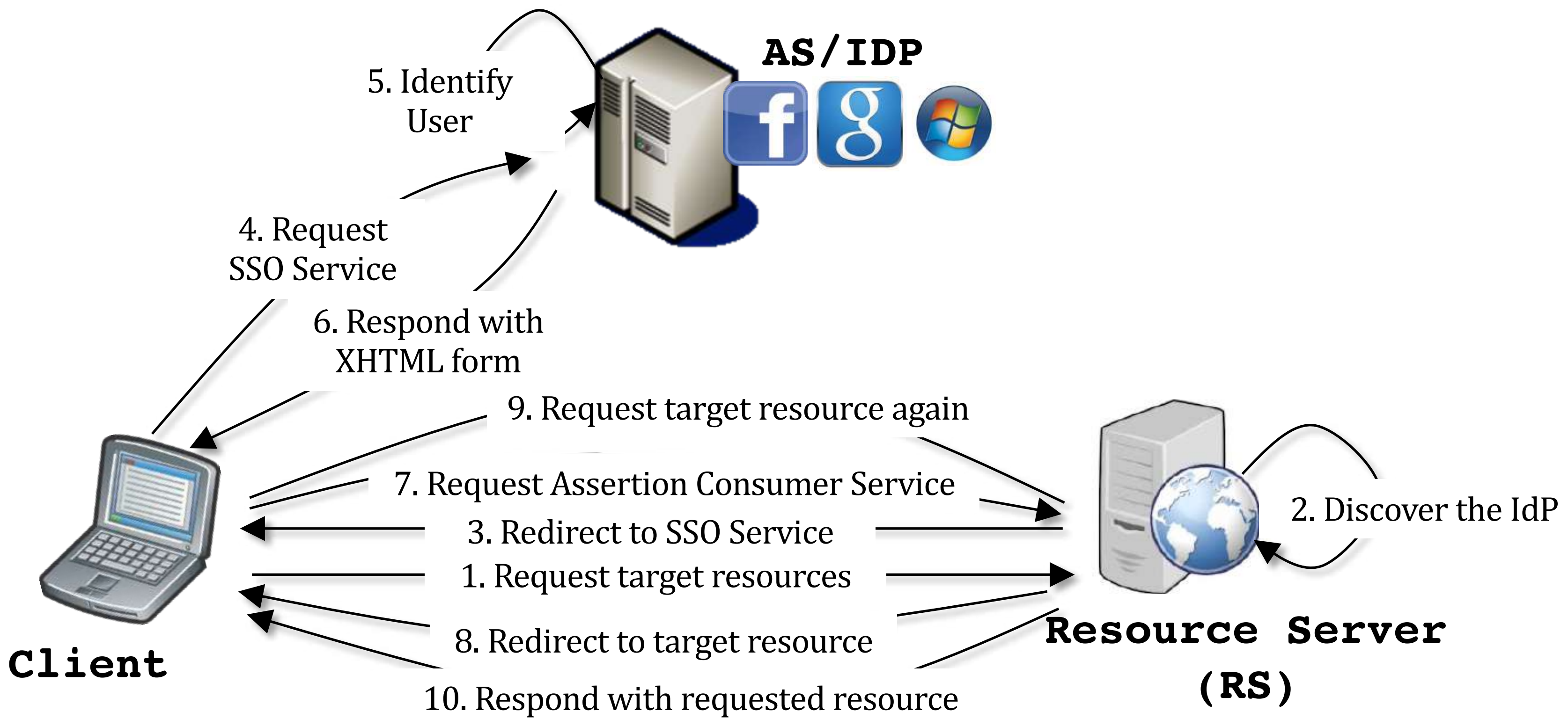
The vulnerability is that there is no way for the Server to know whose key it is authorizing during the handshake.

Web Authentication – SAML

- Security Assertion Markup Language (SAML) is an open standard for exchanging authentication and authorization data between parties.
  - between an identity provider (Authorization server) and a service provider (resource server).
- SAML is an XML-based markup language for security assertions
  - statements that service providers use to make access-control decisions.

SAML – How it works?

shopping website (client) redirects to the bank payment and after payment, the authorisation token is kept and can be reused to "pay" other transactions with the same bank


AS/IDP
5. Identify User
4. Request SSO Service
6. Respond with XHTML form
9. Request target resource again
7. Request Assertion Consumer Service
3. Redirect to SSO Service
1. Request target resources
8. Redirect to target resource
10. Respond with requested resource
2. Discover the IdP
Client
Resource Server (RS)

OpenID, OAuth and SAML

	OAuth2	OpenID	SAML
Token	JSON or SAML2	JSON	XML
Authorization	Yes	No	Yes
Authentication	Pseudo-Authentication	Yes	Yes
Year created	2005	2006	2001
Transport	HTTP	HTTP GET/POST	HTTP Redirect (GET) binding, SAML SOAP binding, HTTP POST binding, and others
Security Risks	Phishing; Not support signature, encryption, channel binding or client verification. Relies heavily on TLS .	Phishing; IDP has a log of OpenID login-->compromised account a bigger privacy breach	XML Signature Wrapping to impersonate any user
Best suited for	API Authorization	SSO for consumer apps	

Note: not well suited for mobile

single sign on for enterprise

https://spin.atomicobject.com/2016/05/30/openid-oauth-saml/

Some Notes on Web SSO

- Web SSO for authorization (access granting):
  - Example: OAuth
  - However, a successful authorization *does not* necessarily mean a validly-performed authentication:
    - Alice authorizes Carol to access Bob's resources: Alice ≠ Carol, Carol ≠ Bob, Alice ≠ Bob
- Web SSO for authentication:
  - Example: OpenID, OpenID Connect (the latest)
  - The latest OpenID Connect runs on top of OAuth (see http://openid.net/connect/faq/)

Web SSO: Security Challenges

- Many vulnerabilities have been found on various Web SSO systems
- New *security challenges* of a Web SSO?
  - Web user, IDP, and SP do not belong to the same organization (administrative domain)
  - Web SSO operates on an open public network
  - Web SSO runs on top of Web and browser:
    - Various web vulnerabilities apply
    - Possible insecure browser-based communications, and insecure browser-relayed message (BRM) handlings

with more internal components being exposed via API, allow for more things such as web to local attacks

Some Notes on Web SSO

- Additional references on Web SSO security:
  - Sun & Beznosov, “The Devil is in the (Implementation) Details: An Empirical Analysis of OAuth SSO Systems”, CCS 12
  - Mainka, et al., “SoK: Single Sign-On Security – An Evaluation of OpenID Connect”, IEEE European Symposium on Security and Privacy, 2017
  - Fett, et al., “The Web SSO Standard OpenID Connect: In-Depth Formal Security Analysis and Security Guidelines”, Computer Security Foundations Symposium, 2017

Web SSO: Security Challenges

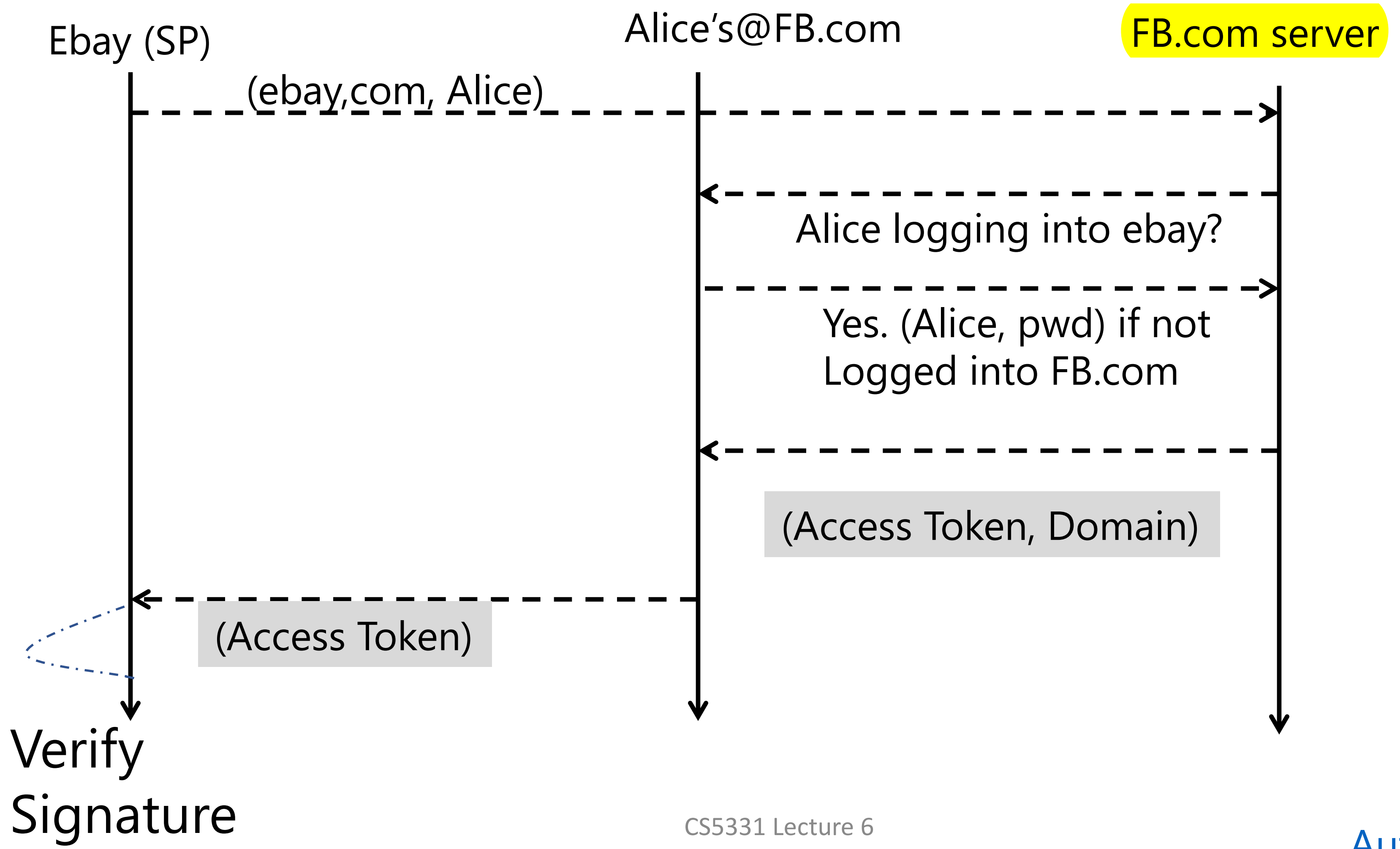
- Many vulnerabilities have been found on various Web SSO systems
- New *security challenges* of a Web SSO?
  - Web user, IDP, and SP do <u>not</u> belong to the same organization (administrative domain)
  - Web SSO operates on an <u>open public</u> network
  - Web SSO runs <u>on top of Web and browser</u>:
    - Various web vulnerabilities apply
    - Possible insecure browser-based communications, and insecure browser-relayed message (BRM) handlings
- Recognized security threats:
  - E.g.: "*OAuth 2.0 Threat Model and Security Considerations*", IETF RFC 6819, Jan 2013

Web SSO: Security Challenges

- Sample security threats (from RFC 6819):
  - Authorization "code“:
    - Eavesdropping or leaking Authorization "codes"
    - Online guessing of Authorization "codes"
  - Implicit grant:
    - Access token leak in browser history
    - CSRF attack against redirect-uri
  - Accessing protected resources:
    - Replay of authorized resource server requests
    - Leak of confidential data in HTTP Proxies
    - Token leakage via log files and HTTP Referrers
  - …

How Does It Work?

- E.g. High-level overview of FB Connect

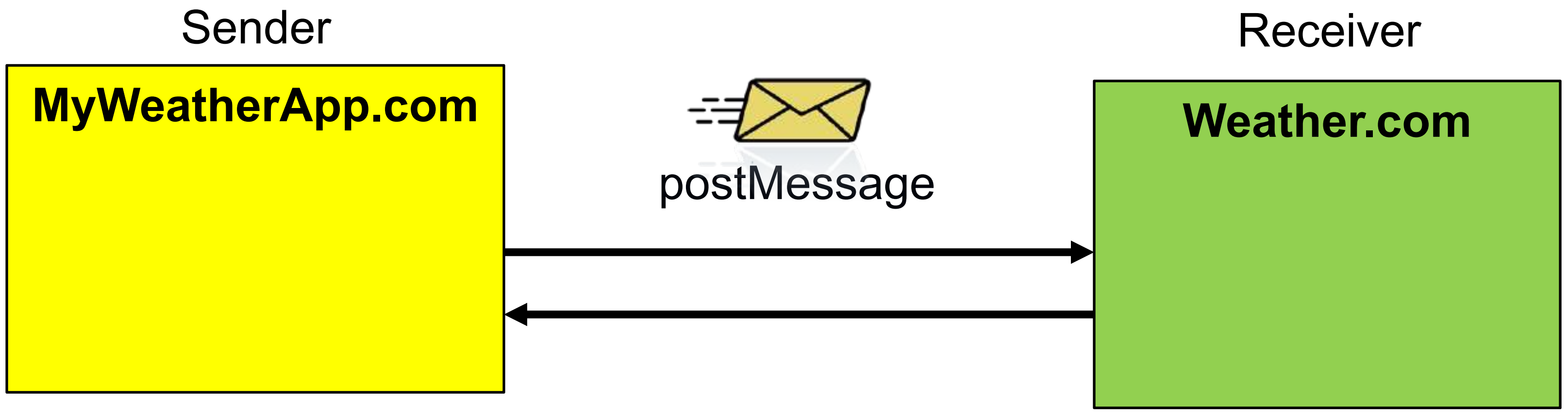


AuthScan

How Does It Work?

- Use of postMessage(), which can be insecure:

```
targetWindow.postMessage(message, targetOrigin,
                    [transfer]);
```



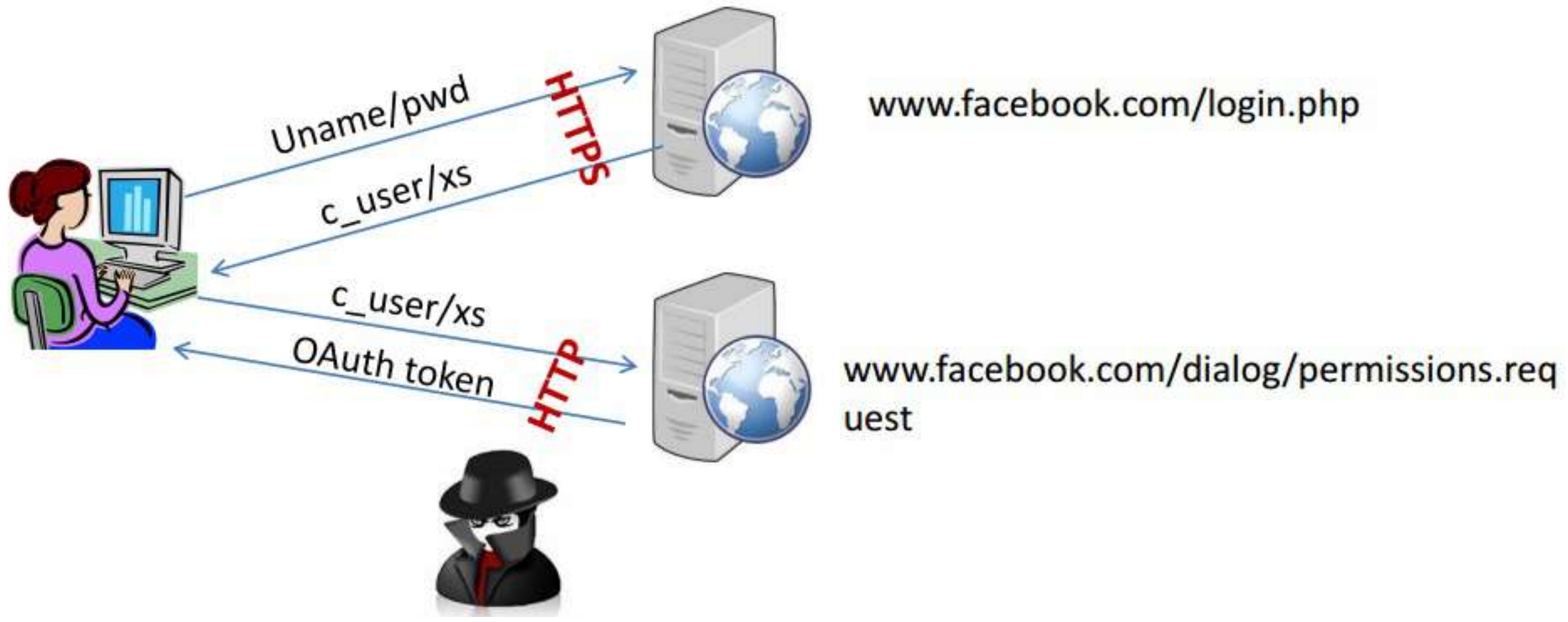
To: Weather.com
Origin: www.myweatherapp.com
Data: “get_weather(94710)”

To: MyWeatherApp.com
Origin:www.weather.com
Data: “Sunny,75”

AuthScan

What can Go Wrong?

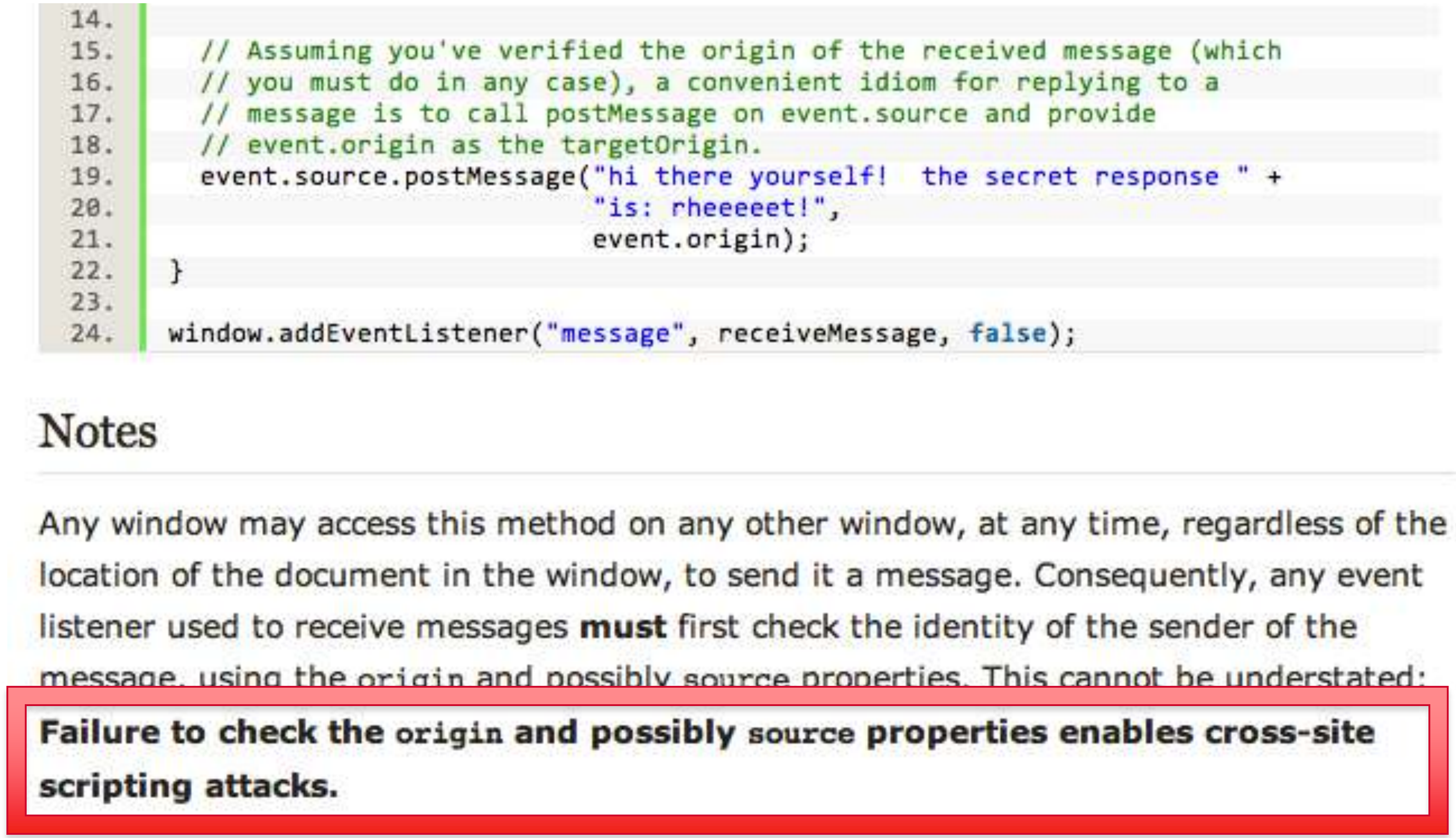
- Example #1
  - Secret Token Leakage
    - Secret tokens are transmitted through **unencrypted** channels
  - Flaw found in secret cookie in Facebook Connect


Uname/pwd
c_user/xs
HTTPS
www.facebook.com/login.php
c_user/xs
OAuth token
HTTP
www.facebook.com/dialog/permissions.req
uest

AuthScan

What can Go Wrong?

- Example #2: Use of insecure postMessage()



```

  // Assuming you've verified the origin of the received message (which
  // you must do in any case), a convenient idiom for replying to a
  // message is to call postMessage on event.source and provide
  // event.origin as the targetOrigin.
  event.source.postMessage("hi there yourself!  the secret response " +
                           "is: rheeeeet!",
                           event.origin);
}

window.addEventListener("message", receiveMessage, false);
```

Notes

Any window may access this method on any other window, at any time, regardless of the location of the document in the window, to send it a message. Consequently, any event listener used to receive messages **must** first check the identity of the sender of the message, using the `origin` and possibly `source` properties. This cannot be understated:

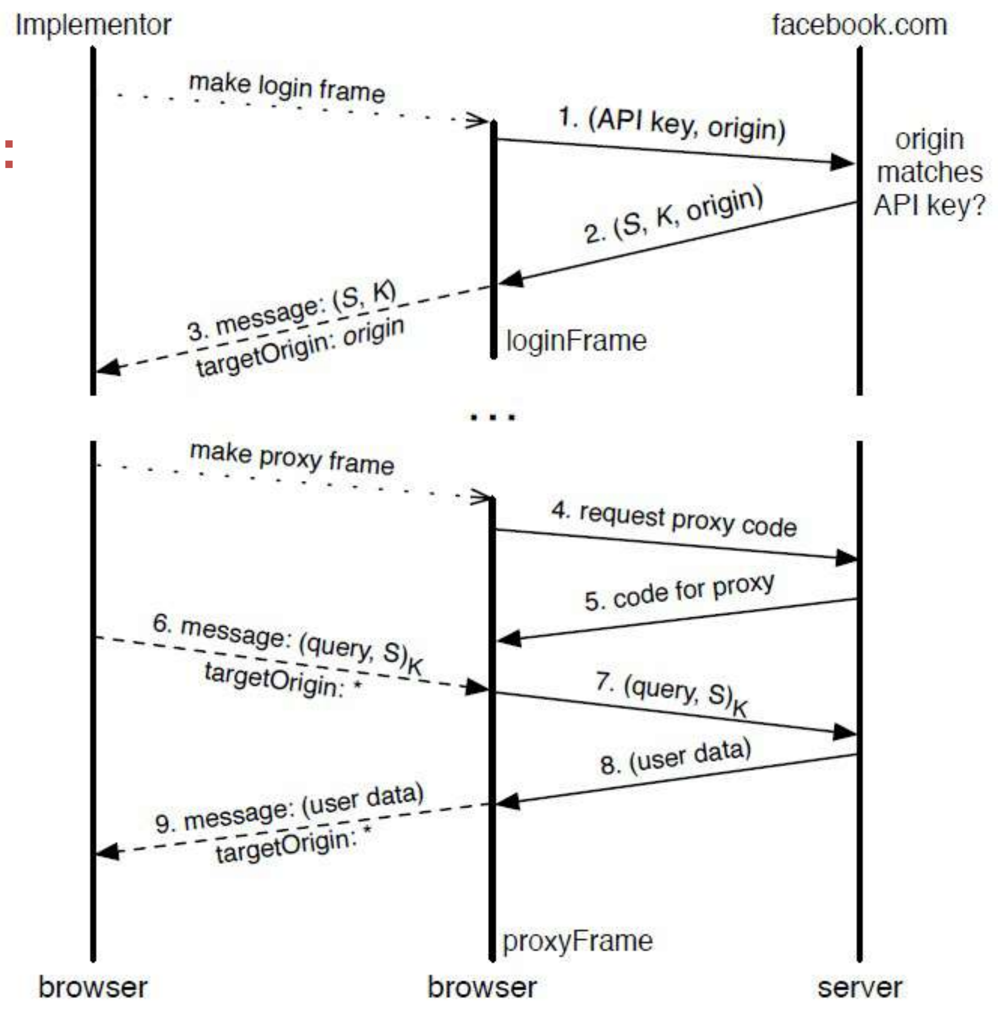
Failure to check the `origin` and possibly `source` properties enables cross-site scripting attacks.

The Emperor's New APIs: On the (In)Secure Usage of New Client-side Primitives

What can Go Wrong?

- Example #2: Use of insecure postMessage()

Original Protocol:



The Emperor's New APIs: On the (In)Secure Usage of New Client-side Primitives

What can Go Wrong?

- Example #2: Use of insecure postMessage()

Original Protocol & Its Frame Hierarchy:

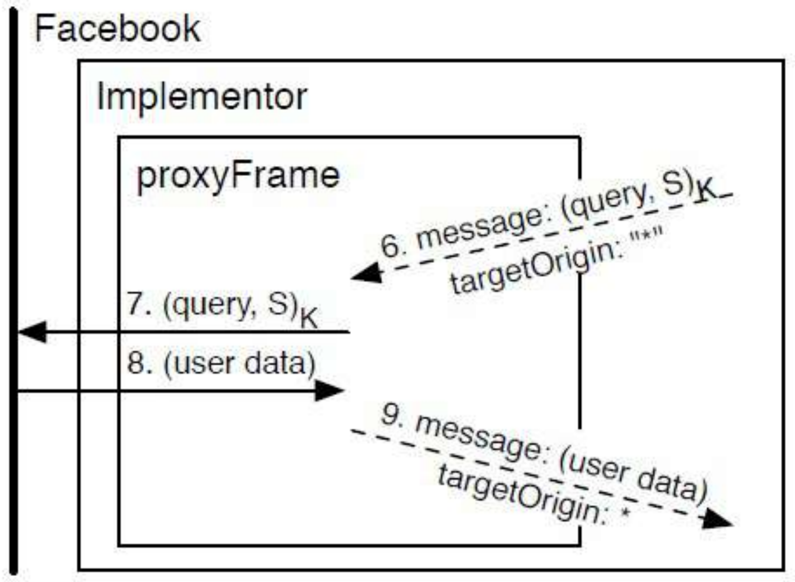


Figure 1: The Facebook Connect protocol. (top) Messages exchanged in the protocol. The dashed arrows represent client-side communication via `postMessage` and the solid arrows represent communication over HTTP. $(\texttt{query,S})_K$ represents a HMAC using the secret K. (bottom) Frame hierarchy for the Facebook Connect protocol. In this example, the `proxyFrame` is inside the main `implementor` window.

[The Emperor's New APIs: On the (In)Secure Usage of New Client-side Primitives]

What can Go Wrong?

- Example #2: Use of insecure postMessage()

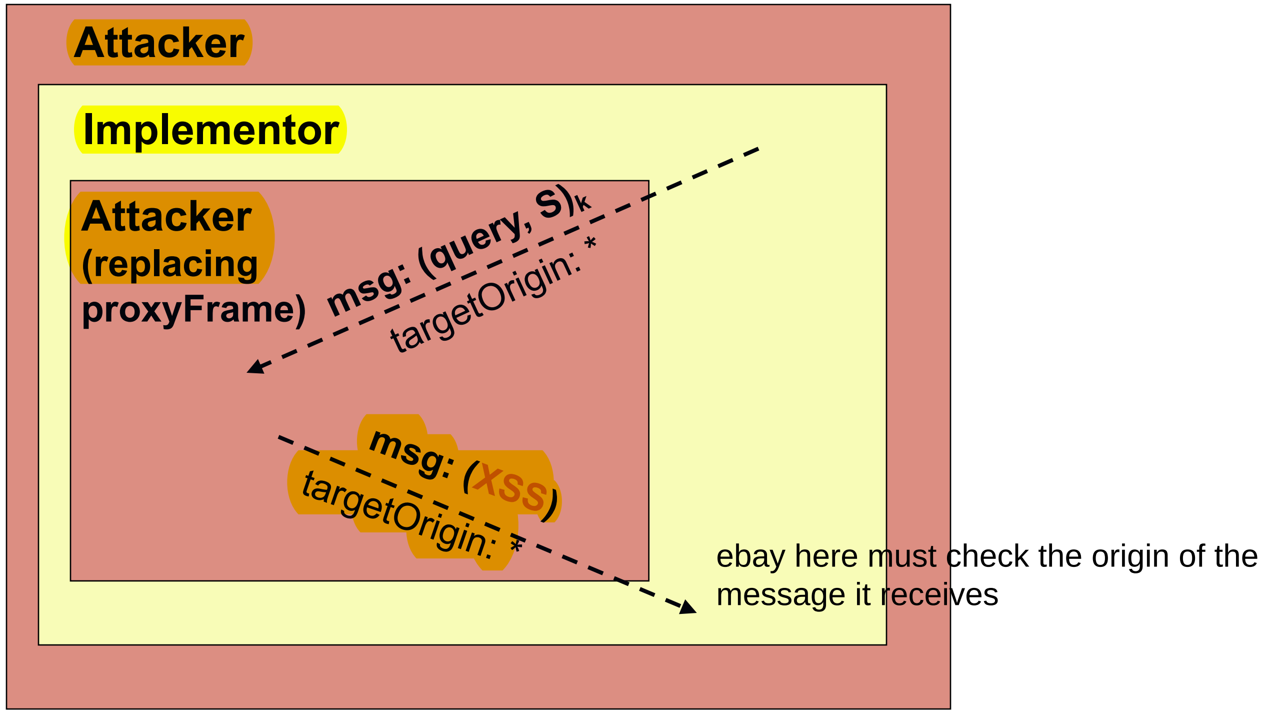
Attack on Integrity:

The origin of half of the messages were verified

Lack of origin checks allow attacker to inject arbitrary data in the communication between the ***implementor*** and ***proxyFrame***.

Attacker can **replace** the **proxyFrame** with own frame. This allows the attacker to fully **XSS** the **implementor.**

Facebook Connect Frame Hierarchy:
(proxyFrame replaced with attacker-controlled proxyFrame)

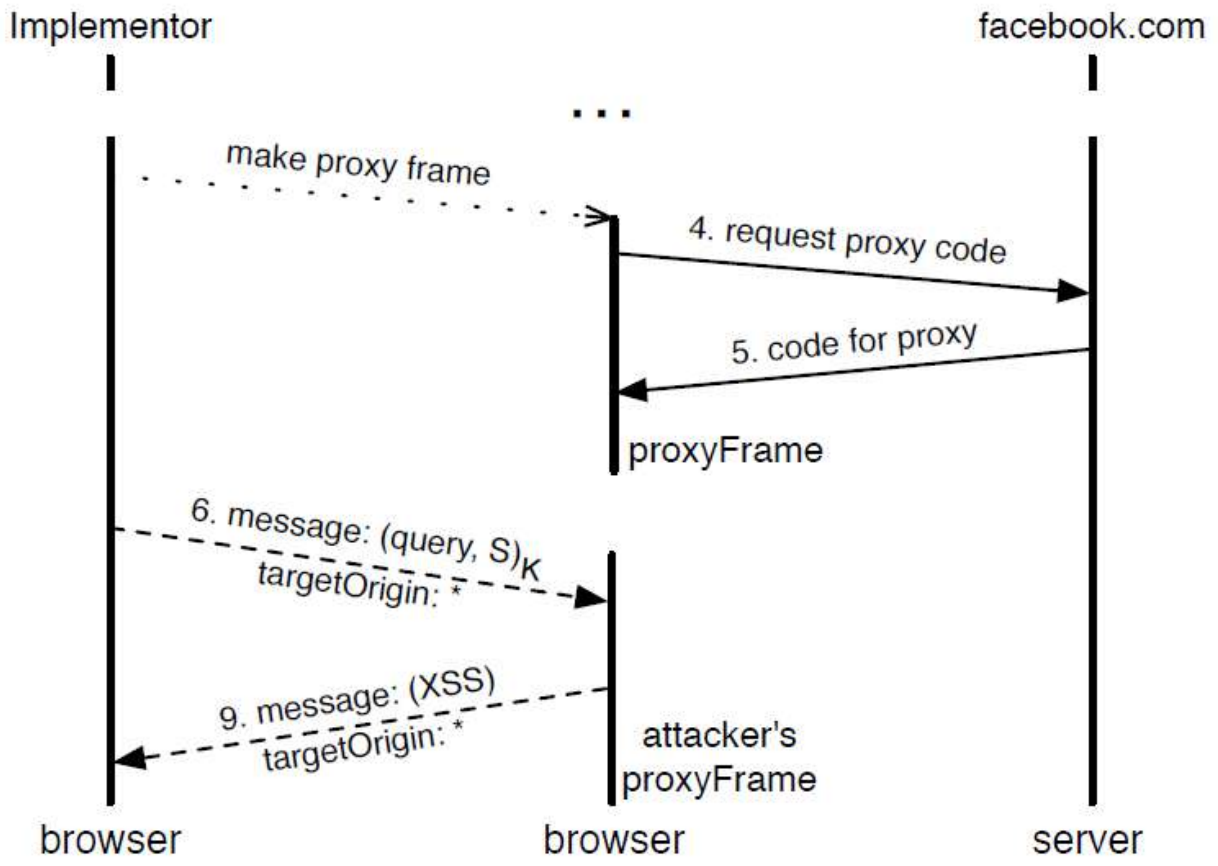

Attacker
Implementor
Attacker (replacing proxyFrame)
msg: (query, S)k
targetOrigin: *
msg: (XSS)
targetOrigin: *
ebay here must check the origin of the message it receives

The Emperor's New APIs: On the (In)Secure Usage of New Client-side Primitives

What can Go Wrong?

- Example #2: Use of insecure postMessage()

Attack on Integrity (Message Communication)



The Emperor's New APIs: On the (In)Secure Usage of New Client-side Primitives

What can Go Wrong?

- Example #2: Use of insecure postMessage()

MITM Attack: Attack on both Confidentiality & Integrity

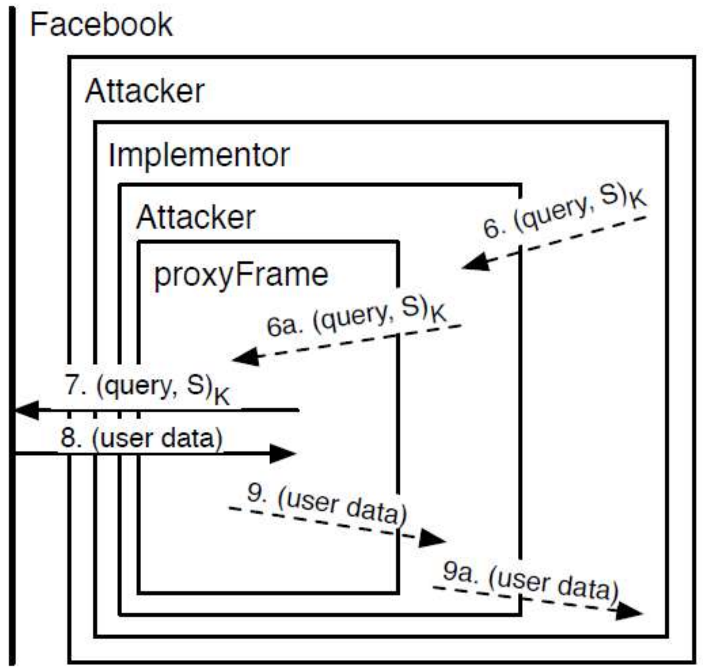


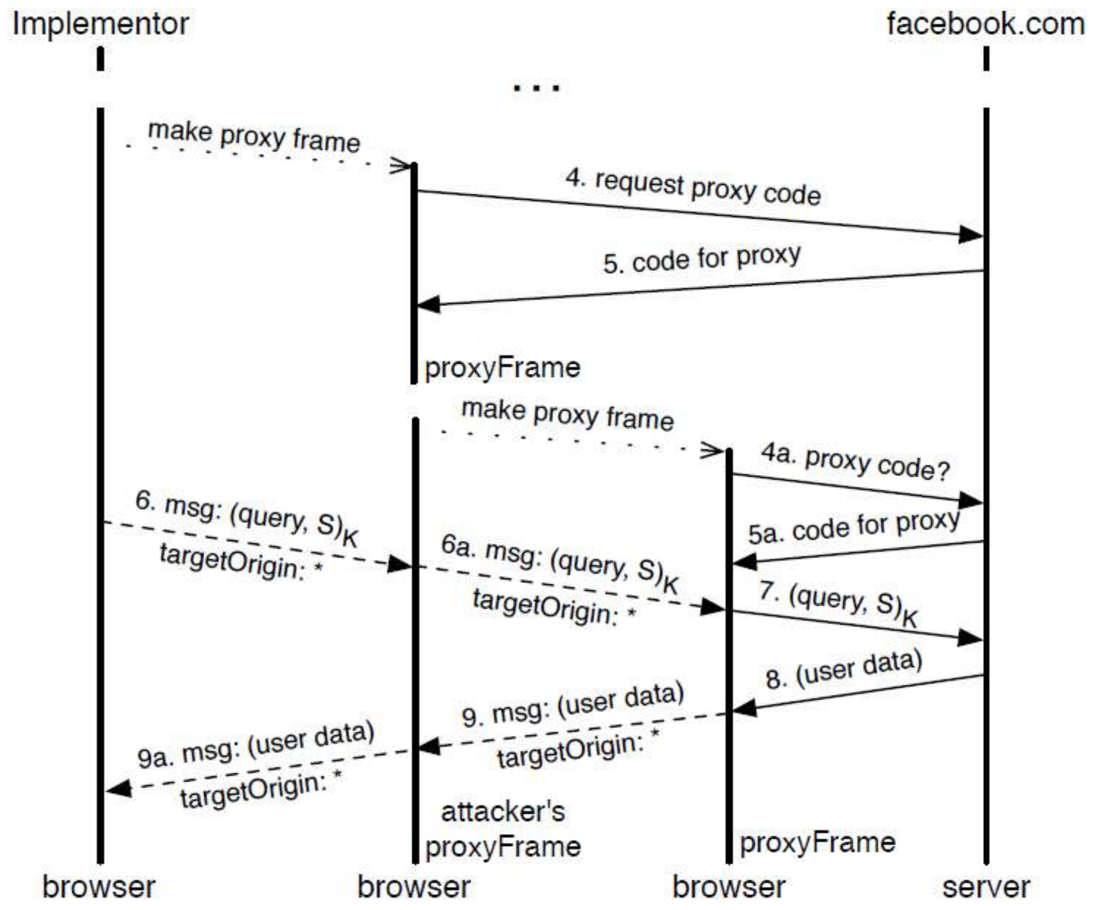
Figure 3: Confidentiality attack on Facebook Connect. (top) Message Exchange—note the replayed messages 6a and 9a. (bottom) Frame hierarchy for the confidentiality attack. Note the presence of two attacker frames—the main window frame and the man-in-the-middle frame.

The Emperor's New APIs: On the (In)Secure Usage of New Client-side Primitives

What can Go Wrong?

- Example #2: Use of insecure postMessage()

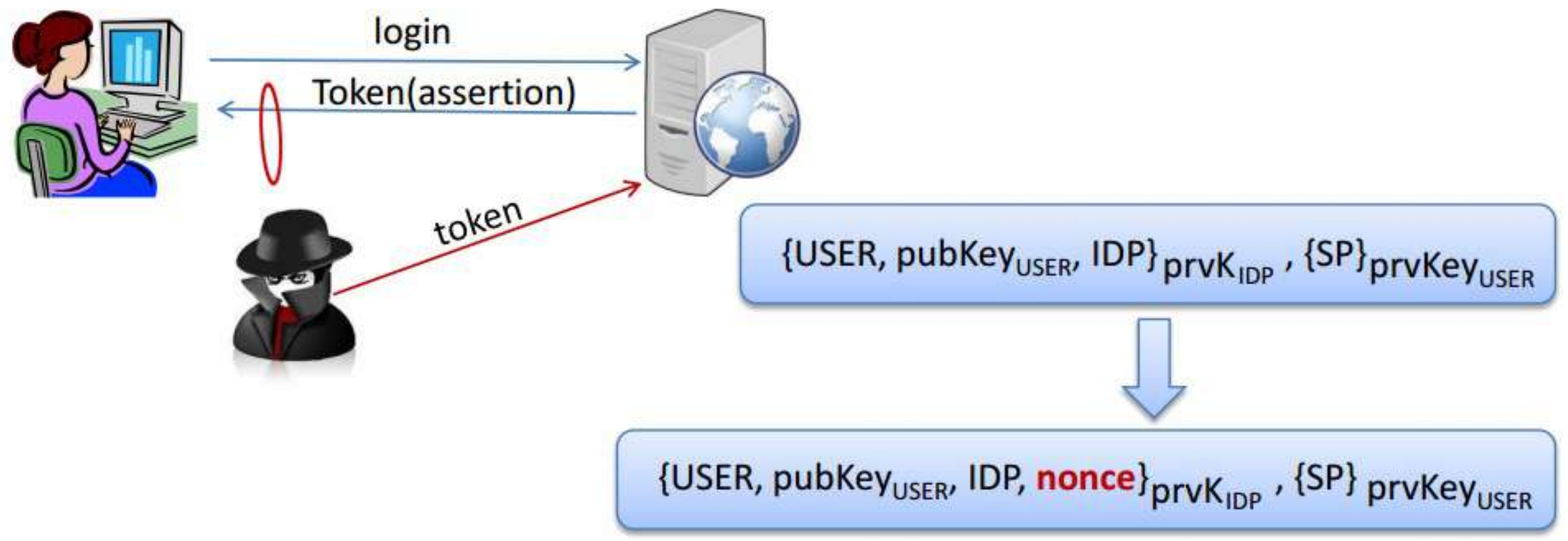
MITM Attack: Attack on both Confidentiality & Integrity



The Emperor's New APIs: On the (In)Secure Usage of New Client-side Primitives

What can Go Wrong?

- Example #3: Replay attacks (BrowserID)
    - Missing Nonce
        - May lead to replay attacks



$$\{USER, pubKey_{USER}, IDP\}_{prvK_{IDP}}, \{SP\}_{prvKey_{USER}}$$

$$\Downarrow$$

$$\{USER, pubKey_{USER}, IDP, \textbf{nonce}\}_{prvK_{IDP}}, \{SP\}_{prvKey_{USER}}$$

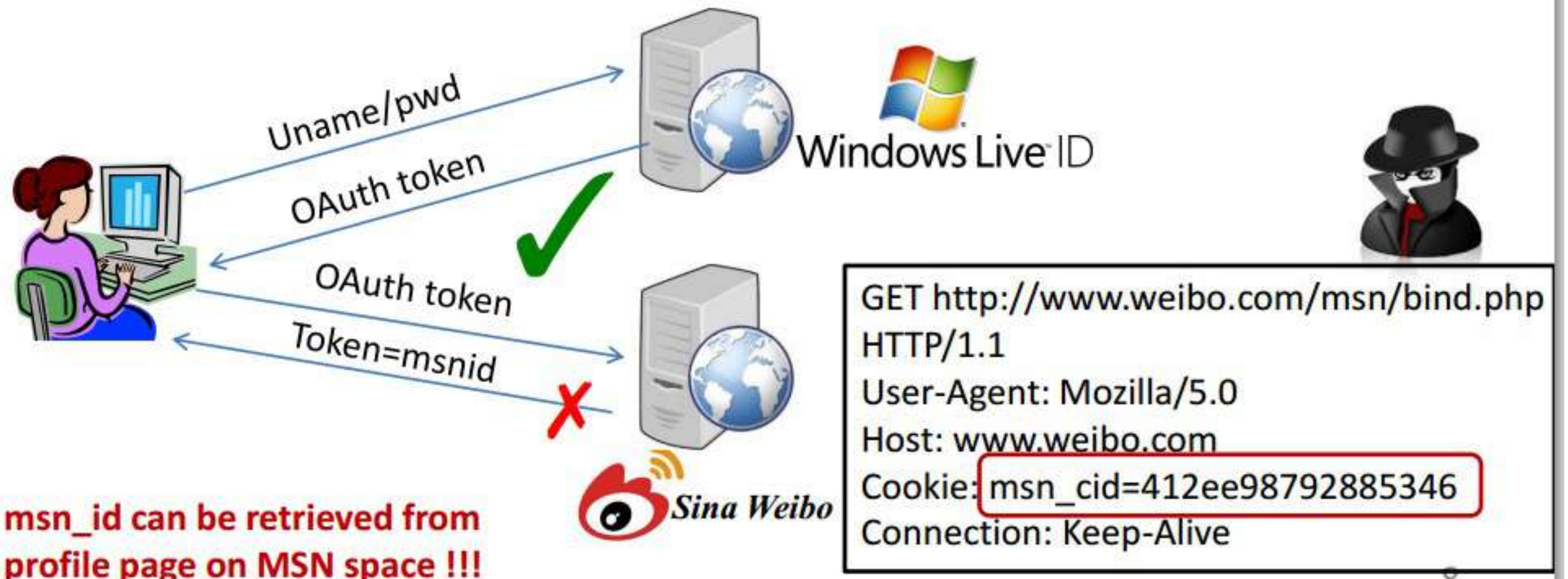
AuthScan

What can Go Wrong?

- Example #4: In Microsoft Live ID

- Using Publicly-Known Values as Tokens
  - Keep **constant** across multiple login sessions and the values are **publicly-known**
  - e.g., email, publicly-known id, hash(email), etc.

- Flaw found in credential cookies in Sina Weibo



```
GET http://www.weibo.com/msn/bind.php
HTTP/1.1
User-Agent: Mozilla/5.0
Host: www.weibo.com
Cookie: msn_cid=412ee98792885346
Connection: Keep-Alive
```

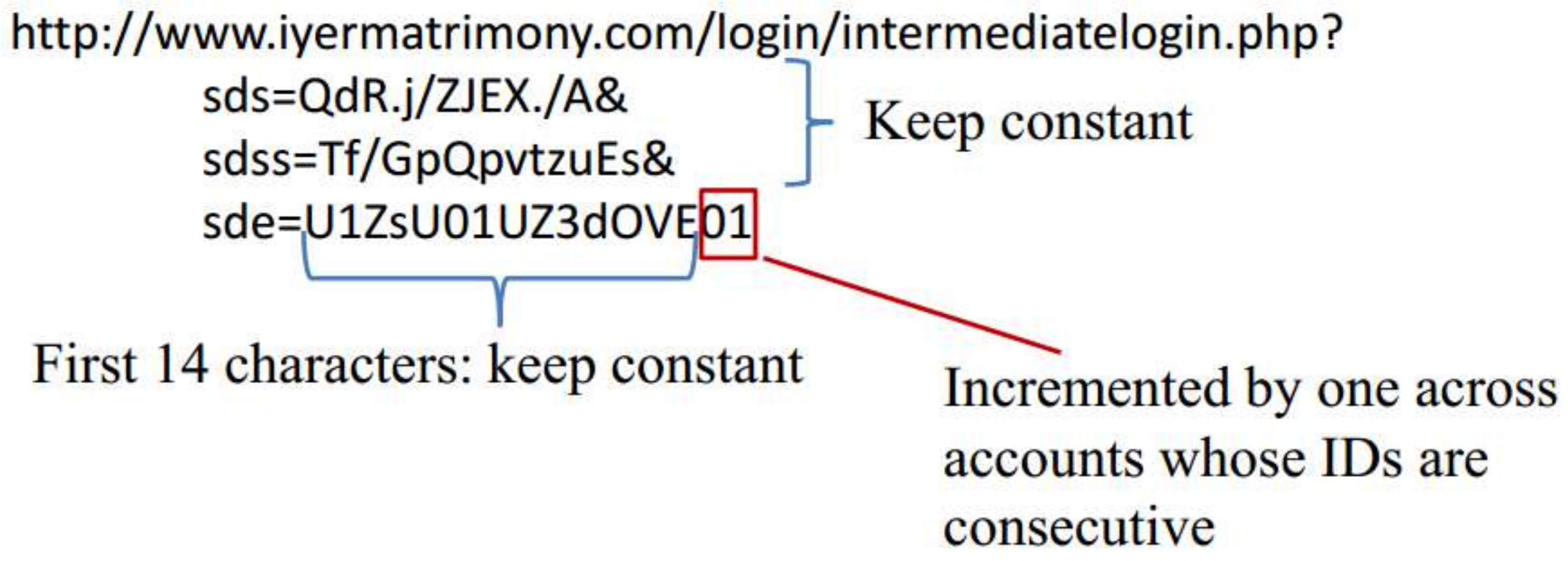
msn_id can be retrieved from profile page on MSN space !!!

8

AuthScan

What can Go Wrong?

- Example #5: Predictable Tokens
    - Guessable Token



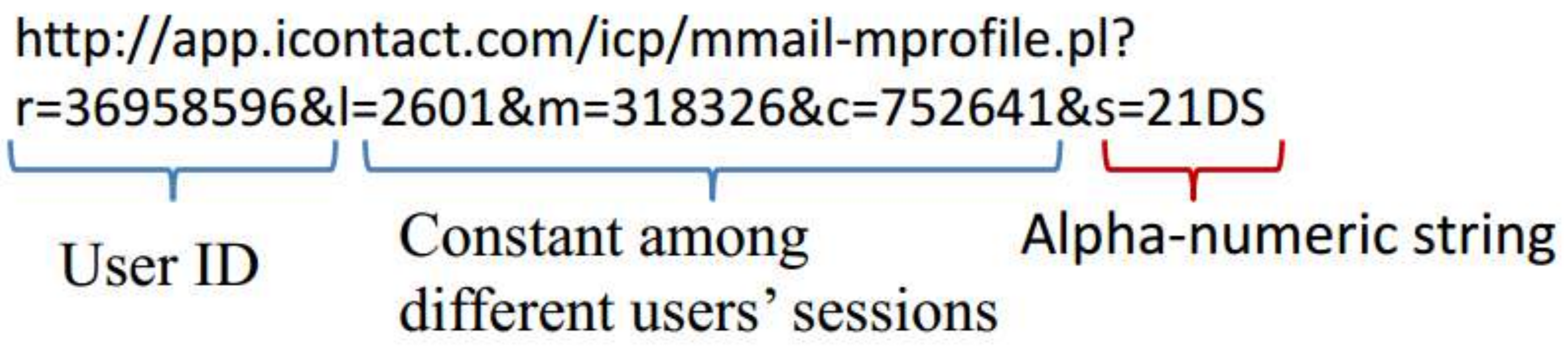
```
http://www.iyermatrimony.com/login/intermediatelogin.php?
        sds=QdR.j/ZJEX./A&
        sdss=Tf/GpQpvtzuEs&
        sde=U1ZsU01UZ3dOVE01
```

Keep constant
First 14 characters: keep constant
Incremented by one across accounts whose IDs are consecutive

AuthScan

What can Go Wrong?

- Example #6: Predictable Tokens
  - Short-Length Token



http://app.icontact.com/icp/mmail-mprofile.pl?
r=36958596&l=2601&m=318326&c=752641&s=21DS

$(10 + 26)^4$ Possible Values
Attacker: 500 "probes"/ min

AuthScan

Some Notes on Web SSO

- Web SSO for authorization (access granting):
  - Example: OAuth
  - However, a successful authorization *does not* necessarily mean a validly-performed authentication:
    - Alice authorizes Carol to access Bob's resources:
      Alice ≠ Carol, Carol ≠ Bob, Alice ≠ Bob
- Web SSO for authentication:
  - Example: OpenID, OpenID Connect (the latest)
  - The latest OpenID Connect runs on top of OAuth
    (see http://openid.net/connect/faq/)

Some Notes on Web SSO

- Additional references on Web SSO security:
  - Sun & Beznosov, “The Devil is in the (Implementation) Details: An Empirical Analysis of OAuth SSO Systems”, CCS 12
  - Mainka, et al., “SoK: Single Sign-On Security – An Evaluation of OpenID Connect”, IEEE European Symposium on Security and Privacy, 2017
  - Fett, et al., “The Web SSO Standard OpenID Connect: In-Depth Formal Security Analysis and Security Guidelines”, Computer Security Foundations Symposium, 2017

Last Recap: Web Vulnerabilities

- We have seen:
  1. XSS (Reflected, Persistent, DOM-bases, MXSS)
  2. SQLI
  3. Directory Traversal
  4. OS Command Injection
  5. MIME Content Sniffing
  6. CSRF
  7. HTTP Parameter Pollution
  8. HTTP Parameter Tampering
  9. HTTP Header Injection
  10. Cookie Flaws
  11. Weak Session IDs
  12. PostMessage Errors
  13. SSO Flaws
  14. File Upload & RFI/LFI
  15. Missing Access Ctrl.
  16. Open Redirects