OAuth2 WG
User Authentication for Clients

Phil Hunt

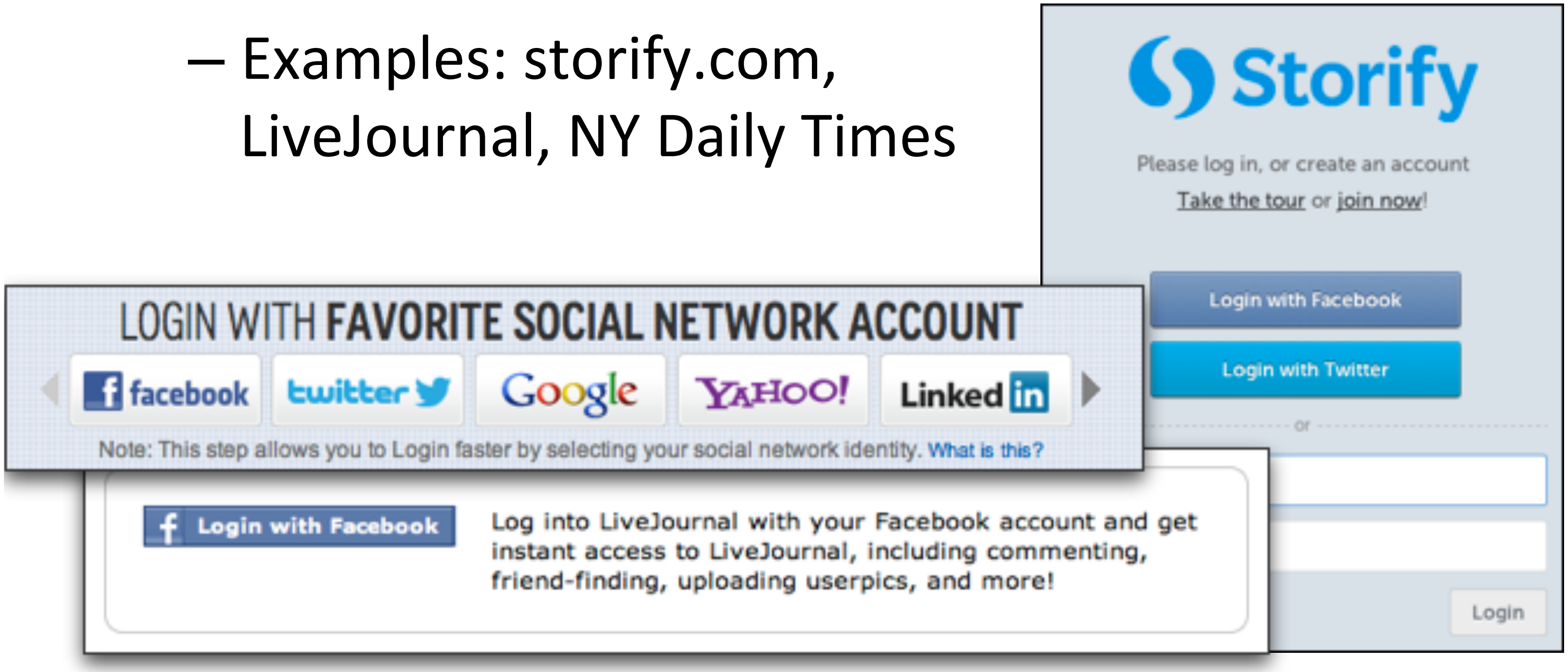
Oracle Corp

July 31, 2013

THE OAUTH USER AUTHENTICATION PROBLEM

What's The Need?

- There is a strong desire by many developers to use social networks and others to facilitate easy user registration and user authentication.
  - Examples: storify.com, LiveJournal, NY Daily Times



Note: This step allows you to Login faster by selecting your social network identity. What is this?

Log into LiveJournal with your Facebook account and get instant access to LiveJournal, including commenting, friend-finding, uploading userpics, and more!

Don't Facebook, Twitter, & Amazon Do Auth?

- Yes
  - Some social networks invite developers to use their OAuth2 service to return information about currently authenticated user
- But is return of profile data proof of authentication?
  - There remain cross-site scripting attacks since client can't detect injected grants
  - No guarantee returned resource profile == authenticated user
- It could be a valid approach with further specification

Is This Correct Usage of OAuth?

- Not at present
  - OAuth2 does not directly perform user authentication itself
  - Authenticating users to clients is "backwards" to the intended flow of providing clients tokens to authenticate with resources.
    - Intended audience is the resource service
- Missing Features Needed
  - Specifications for passing clients AuthN information
  - Processing rules for clients to validate AuthN information from an OAuth AS
  - Potential NIST SP-800-63 compliance

More Background

- OAuth2 is not actually an authentication system, it issues access authorizations
  - Issues tokens to clients that enable access to protected resources at a service provider
  - Client wields tokens, not intended to use them
  - Client unable to validate tokens/assertions
    - Can only use non-standard resource API
  - Client cannot bind original request to issued token
    - Possibility of cross site request forgery attack
- End-user authentication is 'implied' or 'indirect'
  - Client has no way to determine what happened
  - Client cannot force authentication
  - Client is not the intended audience – just the wielder

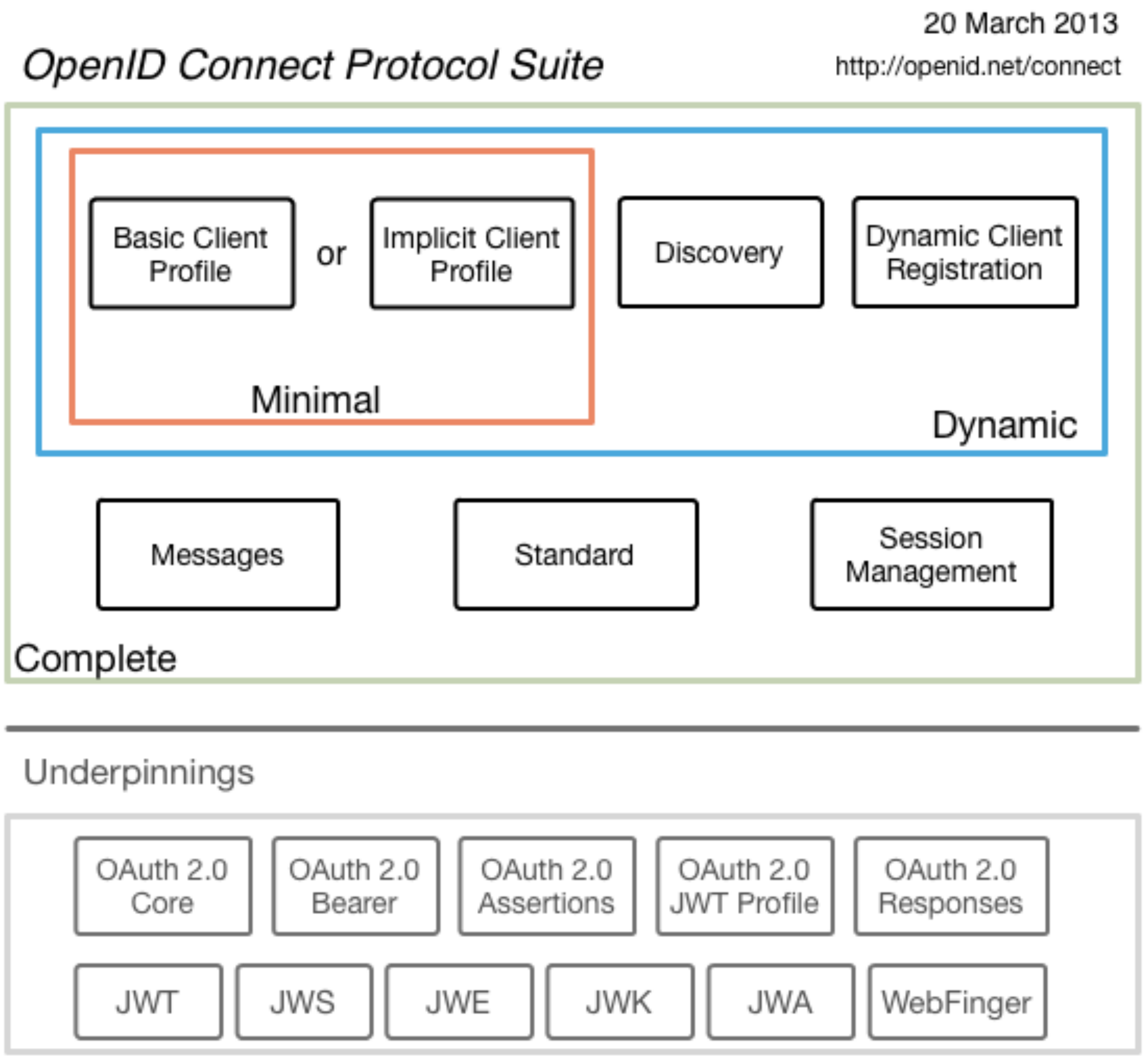
IS OPENID CONNECT A SOLUTION?

OpenID Connect

- A set of specifications from OIDF addressing the authentication issue among others
- Not related to OpenID 2.0, but does take some requirements forward
- Provides a session token (called ID Token)
- Addresses CSRF issue for clients
- and…

Additional Features

- Identity Profile API
- Discovery
- Dynamic Registration
- Implicit and Basic Client Flows
- Session Management API


20 March 2013
OpenID Connect Protocol Suite
http://openid.net/connect
Basic Client Profile
or
Implicit Client Profile
Minimal
Discovery
Dynamic Client Registration
Dynamic
Messages
Standard
Session Management
Complete
Underpinnings
OAuth 2.0 Core
OAuth 2.0 Bearer
OAuth 2.0 Assertions
OAuth 2.0 JWT Profile
OAuth 2.0 Responses
JWT
JWS
JWE
JWK
JWA
WebFinger

Identity Profile Seems Like A Good Thing, but…

- Not all service providers want to be profile providers
  - Their primary business is not profile services
  - The desire to reduce uid/passwords leads many to still want to use any site that can authenticate
- SCIM is an alternate RESTful Identity service that may emerge as a standard profile service
- Many sites still want to manage their own user profile data, just want some other service to perform user authentication.

USER AUTHN PROPOSAL

Proposed features

- New "authenticate" endpoint
  - Parallels 4.1 of RFC6749 but returns additional session information
  - Return of access token is optional
- Gives client apps ability to request re-authentication, re-authorization, and to test login state.
  - similar to OpenID Connect
- Normal access token return supplemented with session state information
- Refresh provides ongoing session information (e.g. logout detection)
- Could be structured as first step towards OpenID Connect support
- Could be added direct to normal authorize flow
  - Specifies additional session info, no normative changes

Flow

```
GET /authenticate?
response_type=code
&client_id=s6BhdRkqt3
&redirect_uri=https%3A%2F%2Fclient.example.com%2Fcb
&state=af0ifjsldkj
&prompt=login
```

Standard OAuth CSRF protection mechanism

Requests that AS force re-authentication of user

Request Parameters

- New "/authenticate" endpoint
- Standard OAuth2
  - response_type – value MUST be "code"
  - client_id – The client identifier (per 2.2)
  - redirect_uri – Required – MUST be pre-registered
  - state – Opaque client generated value to maintain state between request and callback (XSRF protection)
- and…

Request Params Cont'd

- prompt– Space delimited set of authn/authz functions
  - none – Authorization server confirms authentication status
  - login – Authorization server prompts use for re-authentication (even if already authenticated)
  - consent – Re-obtain consent for associated resource service (if any)
  - select_account – prompt user to select account
- display
  - page, popup, touch, wap
- hint – A code or text that may be used by the AS to display text to the user during authentication or authorization operations
  - This could be done by registration information of client

New AS Server Processing

- Normal OAuth2 authorization processing occurs
- AS may not need to ask for consent if only authn information is returned
- AS MUST re-authenticate user if func includes "authn".
- AS MUST return an error if func is "none" and an existing user sign-on session does not exist

Authentication Response

- Upon authentication and consent, as per normal OAuth flow, redirect is passed to client.
- If "state" present in authentication request, the exact value received must be returned
- Normal OAuth2 errors

```
HTTP/1.1 302 Found
Location: https://client.example.com/cb?
code=SplxlOBeZQQYbYS6WxSbIA&state=af0ifjsldkj
```

Token Request

- Client is authenticated
- Code is exchange for authn status + optional access token

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

grant_type=authorization_code&code=SplxlOBeZQQYbYS6WxSbIA
&redirect_uri=https%3A%2F%2Fclient%2Eexample%2Ecom%2Fcb
```

Access Token Response

- Standard response + session information

```
HTTP/1.1 200 OK
Content-Type: application/json;charset=UTF-8
Cache-Control: no-store
Pragma: no-cache
{
   "access_token":"2YotnFZFEjr1zCsicMWpAA",
   "token_type":"example",
   "expires_in":3600,
   "refresh_token":"tGzv3JOkF0XG5Qx2TlKWIA",
   "session":{
     "sub":"5dedcc8b-735c-405f-bd79-e029f9a76822",
     "sub_url":"https://example.com/me",
     "aud":"{client_id}",
     "iss":"{issuer_id}",
     "loginat":"…",
     "alv":"2"
   },
   "example_parameter":"example_value"
}
```

Token can be used to look up profile info at sub_url

sub_url MUST point to profile of authenticated user

Is issuer/audience needed since atomic request/response? (no token)

NIST Assurance Level

New Session Information

- Contains information about user authn state
- Params
  - sub – REQD - The identifier of the subject authenticated.
  - sub_url – The location of the authentication subject profile that may be retrieved using the returned access token
  - iss – REQD - The identifier of the Authentication server (usually authorization server)
  - aud – REQD - The identifier of the client or URI (per SAML definition?)
  - at, exp – SHOULD - Authentication time and authentication expiry time in Internet Date Time Format per RFC3339
  - alv – OPTIONAL - Assurance level per NIST SP 800-63
- Can be passes as JSON struct or JWT (for SP-800-63 compliance)

Access Token Params

- The access token is not necessarily bound to an Identity Profile service (e.g. Twitter API)
- Could be used with any REST API
- Could also be used with SCIM or OpenID Connect
- MAY be used to access a location (defined by sub_url) to retrieve subject profile information

Processing Rules

- The aud parameter MUST be EITHER the client_id or a previously negotiated URI that applies to the client.
- The issuer MUST be associated with the AS
- at MUST NOT be future dated
- exp MUST NOT be in the past
- Level 1 and 2, can use connection-oriented security
  - Session token not required
  - Session token validation not required
  - aud and iss not required
- Level 3, 4 requires validated session token

DISCUSSION

Questions

- Should the WG address the authentication issue?
- Should we include in the next WG charter?
- Is alignment of UA4C with OpenID Connect appropriate?
  - Or is it less confusing to address in very different way?