Using OpenID/OAuth to access Federated Data Services

M. Benno Blumenthal
IRI of Columbia University

GO-ESSP 2011

10 May 2011
CMIP3

Pydap server: http://esgcet.llnl.gov/dap/ipcc4/?thredds

THREDDS catalog
OpenDAP service points
Basic Authentication

Cloud-accessible with basic authentication pass-through
i.e. llnl controlled user/password access to analysis done in the cloud with CMIP3 data
Sample CMIP3 access

WCRP CMIP3 ipcc4/2xco2 ipcc4/2xco2/gfdl_cm2_0
pcmdi.ipcc4.gfdl_cm2_0.2xco2.run1.atm.mo.xml pr options

WCRP CMIP3 ipcc4/2xco2 ipcc4/2xco2/gfdl_cm2_0
pcmdi.ipcc4.gfdl_cm2_0.2xco2.run1.atm.mo.xml pr:
Precipitation data

Independent Variables (Grids)

lat
grid: /lat (degree_north) ordered (89S) to (89N) by 2. N= 90 pts ;grid

lon
grid: /lon (degree_east) periodic (1.25E) to (1.25W) by 2.5 N= 144 pts ;grid

time
grid: /time (days since 0001-01-01 00:00:00) (365) ordered [ (Jan 0001) (Feb 0001) (Mar 0001) ... (Dec 0100)] N= 1200 pts ;grid

Other Info
CMIP3 is cloud accessible

Example

Canadian PhD student used this IRI data library interface to do EOF, correlation and other analysis on CMIP3 model runs as well as other climate data

Now leading Climate and Agriculture group in Pakistan

Accessiblity makes a difference!
Mashup Authentication

A simple data mashup: difference between two variables from two different datasets

If both datasets are access-restricted under different security realms (different userid/password), then the difference cannot be authenticated (Basic/Digest Authentication only has one set of authentication info)
OpenID

OpenID separates user identification from resource access authorization, so a federation of servers can have users with the same id, yet decide separately who gets access to their resources.

It also means resource providers can get out of the user authentication business.

This is half of the solution to the mashup problem.
Man-in-the-Middle

Modern authentication schemes (i.e. other than Basic) defend against man-in-the-middle attacks, i.e. defend against a third-party sitting in the middle of the browser and the authenticating server conversation and relaying requests while copying for nefarious reasons.

This also eliminates third-parties for good reasons. So we need to separate the good third parties from the bad ones. More than likely, this means one must explicitly authenticate third-parties (cloud applications) as well as users.
OAuth

Mechanism to authenticate third-parties
Used by third-party apps to access Google and Facebook data, for example
Not a perfect match to our problem: built for large data provider, tiny app limit, i.e. most apps build for one data source.
Currently missing the “refusal” part of the standard, but that could be part of Oauth 2.0, at which point it becomes “Token” Authorization (refusal does not distinguish between 2nd and 3rd parties, only the process for getting a token differs).
OAuth is token-based

Bearer Token – possession of token is sufficient for access

MAC Token – has associated secret, i.e. token can be used over an unsecure channel
OAuth 1.0 initial access
OAuth 1.0 subsequent access
OAuth App must

- OAuth Protocol
- Identify user
- Remember authorized tokens by (user, data server)
- Probably will have pre-registered with data source authentication service
OAuth 2.0

Authorization protocol in parallel with Basic and Digest Authentication, i.e. the Unauthorized response gives the information needed to get authorization.

Will probably be called “Token” or “Bearer” or “MAC” (i.e. kinds of tokens) in WWW-Authenticate which will provide two endpoints for authorization.

Authorization-uri: endpoint for user to identify with

token-uri: endpoint for app to identify with
OAuth 2.0
Web App Flow(*)

Browser

Authorization Service

Data Service

App
Federated Security for Federated Data

Authentication Service split between user identifier and data protector

Data Apps need to identify users and associated valid tokens

Data Providers need to validate both users and data apps

Need to avoid breaking anonymous/cached access

Could create a bit of a mess – when are the results of a data analysis finally released from the use-constraints of the data that went into it?
As CMIP5 comes on line, it will be protected by a combination of OpenID and MyProxy certificate based security services. OpenID is a great step for data federation because it separates securing of the data from identifying users, and can be used to secure a direct web client-server interaction, but it does not address the issue of allowing flow of restricted data between servers in producing results for web clients. Here we demo using OAuth to address this issues.

OAuth is a commercially used, evolving protocol that addresses the problem of delivering restricted data to web applications, i.e. single-step third-party analysis of data on the behalf of a web client. Combined with OpenID, it allows lightweight mashup authentication, i.e. generating a result based on data from multiple security domains, using techniques compatible with a wider internet community. As OAuth 2.0 addresses some of the current problems with OAuth, it may provide a useful data federation standard.
CMIP3

Pydap server: http://esgce.llnl.gov/dap/ipcc4/?thredds
THREDDS catalog
OpenDAP service points
Basic Authentication

Cloud-accessible with basic authentication pass-through
i.e. Llnl controlled user/password access to analysis done in the cloud with CMIP3 data
Sample CMIP3 access

WCRP CMIP3 ipcc4/2xco2 ipcc4/2xco2/gfdl_cm2_0 pcmdi.ipcc4.gfdl_cm2_0.2xco2.run1.atm.mo.xml pr: Precipitation data

Independent Variables (Grids)

lat
    grid: lat (degree_north) ordered (89.5) to (89N) by 2, N = 90 pts grid
lon
    grid: lon (degree_east) periodic (1.25E) to (1.25W) by 2.5 N = 144 pts grid
time
    grid: time (days since 0001-01-01 00:00:00) (365) ordered [Jan 0001] (Feb 0001) (Mar 0001) ... (Dec 0001)] N = 1200 pts grid
CMIP3 is cloud accessible

Example

Canadian PhD student used this IRI data library interface to do EOF, correlation and other analysis on CMIP3 model runs as well as other climate data

Now leading Climate and Agriculture group in Pakistan

Accessiblity makes a difference!
Mashup Authentication

A simple data mashup: difference between two variables from two different datasets

If both datasets are access-restricted under different security realms (different userid/password), then the difference cannot be authenticated (Basic/Digest Authentication only has one set of authentication info)
OpenID

OpenID separates user identification from resource access authorization, so a federation of servers can have users with the same id, yet decide separately who gets access to their resources.

It also means resource providers can get out of the user authentication business.

This is half of the solution to the mashup problem.
Man-in-the-Middle

Modern authentication schemes (i.e. other than Basic) defend against man-in-the-middle attacks, i.e. defend against a third-party sitting in the middle of the browser and the authenticating server conversation and relaying requests while copying for nefarious reasons.

This also eliminates third-parties for good reasons. So we need to separate the good third parties from the bad ones.

More than likely, this means one must explicitly authenticate third-parties (cloud applications) as well as users.
OAuth

Mechanism to authenticate third-parties
Used by third-party apps to access Google and Facebook data, for example
Not a perfect match to our problem: built for large data provider, tiny app limit, i.e. most apps build for one data source.
Currently missing the "refusal" part of the standard, but that could be part of OAuth 2.0, at which point it becomes "Token" Authorization (refusal does not distinguish between 2nd and 3rd parties, only the process for getting a token differs).
OAuth is token-based

Bearer Token – possession of token is sufficient for access

MAC Token – has associated secret, i.e. token can be used over an unsecure channel
OAuth 1.0 initial access

Browser to App: request
App to Auth: app credentials and callback
Auth to App: oauth_ttoken
App to Browser: redirect to Auth with oauth_token
Auth to Browser: redirect (with callback) to App with oauth_verifier
App to Auth: credentials, oauth_token, Oauth_verifier
Auth to App: validated oauth_token
App to Data: data request with validated oauth_token
Data to App: data response
App to Browser: response created from Data
Subsequent data accesses reuse the validated token, requiring the app to

1) identify the user, and
2) remember the associated (user,server) token
OAuth App must

- OAuth Protocol
- Identify user
- Remember authorized tokens by (user, data server)
- Probably will have pre-registered with data source authentication service
OAuth 2.0

Authorization protocol in parallel with Basic and Digest Authentication, i.e. the Unauthorized response gives the information needed to get authorization

Will probably be called “Token” or “Bearer” or “MAC” (i.e. kinds of tokens) in WWW-Authenticate which will provide two endpoints for authorization

Authorization-uri: endpoint for user to identify with
token-uri: endpoint for app to identify with
Browser to App: request  
App to Auth: app credentials and callback  
Auth to App: oauth_ttoken  
App to Browser: redirect to Auth with oauth_tolen  
Auth to Browser: redirect (with callback) to App with oauth_verifier  
App to Auth: credentials, oauth_token, Oauth_verifier  
Auth to App: validated oauth_token  
App to Data: data request with validated oauth_token  
Data to App: data response  
App to Browser: response created from Data
Federated Security for Federated Data

Authentication Service split between user identifier and data protector

Data Apps need to identify users and associated valid tokens

Data Providers need to validate both users and data apps

Need to avoid breaking anonymous/cached access

Could create a bit of a mess – when are the results of a data analysis finally released from the use-constraints of the data that went into it?